Milestone 1 — Relational databases

Deadline: Sunday 3 March 2019, End of day

This milestone includes two questions:
1. Load the data in the provided VM/PostgreSQL **20 pt**
2. Optimize your database, and document the process **80 pt**

The fastest correct database for question 2 will get 0.5 pt as a bonus on the course grade.

Input data

You will use the following data files as input:

Degrees.txt
`DegreeId, Dept, DegreeDescription, TotalECTS`
- DegreeId is a unique positive integer.
- Dept is alphabetic, max 50 characters. It can include spaces, i.e., `[ a-zA-Z]+.`
- DegreeDescription is alphabetic, max 200 characters. It can include spaces, i.e., `[ a-zA-Z]+.`
- TotalECTS is a positive integer, less than or equal to 200.

Students.txt
`StudentId, StudentName, Address, BirthyearStudent, Gender`
- StudentId is a unique positive integer.
- StudentName is alphabetic, max 50 characters. It can include spaces, i.e., `[ a-zA-Z]+.`
- Address is alphabetic, max 200 characters. It can include spaces, i.e., `[ a-zA-Z0-9]+.`
- BirthyearStudent is a positive integer, less than 3000.
- Gender is a character (M or F).

StudentRegistrationsToDegrees.txt
`StudentRegistrationId, StudentId, DegreeId, RegistrationYear`
- StudentRegistrationId is a unique positive integer.
- StudentId is a FK to Students.
- DegreeId is a FK to Degrees.
- RegistrationYear is a positive integer, less than 3000.

Teachers.txt
`TeacherId, TeacherName, Address, BirthyearTeacher, Gender`
- TeacherId is a unique positive integer.
- TeacherName is alphabetic, max 50 characters. It can include spaces, i.e., `[ a-zA-Z]+.`
- Address is alphanumeric, max 200 characters. It can include spaces, i.e., `[ a-zA-Z0-9]+.`
- BirthyearTeacher is a positive integer, less than 3000.
- Gender is a character (M or F).

Courses.txt
`CourseId, CourseName, CourseDescription, DegreeId, ECTS`
- CourseId is a unique positive integer.
- CourseName is alphanumeric, max 50 characters. It can include spaces, i.e., `[ a-zA-Z0-9]+.`
- CourseDescription is alphanumeric, max 200 characters. It can include spaces, i.e., `[ a-zA-Z0-9]+.`
- DegreeId is a FK to Degrees.

- ECTS is an integer between 1 and 7 (inclusive).

CourseOffers.txt
`CourseOfferId, CourseId, Year, Quartile`
- CourseOfferId is a unique positive integer.
- CourseId is a FK to Courses.
- Year is a positive integer, less than 3000.
- Quartile is an integer between 1 and 4 (inclusive).

TeacherAssignmentsToCourses.txt
`CourseOfferId, TeacherId`
- CourseOfferId is a FK to CourseOffers.
- TeacherId is a FK to teachers.

StudentAssistants.txt
`CourseOfferId, StudentRegistrationId`
- CourseOfferId is a FK to CourseOffers.
- StudentRegistrationId is a FK to StudentRegistrationsToDegrees.

CourseRegistrations.txt
`CourseOfferId, StudentRegistrationId, Grade`
- CourseOfferId is a FK to CourseOffers.
- StudentRegistrationId is a FK to StudentRegistrationsToDegrees.
- Grade is an integer between 1 and 10 (inclusive).

You can download the data files from
https://surfdrive.surf.nl/files/index.php/s/p8daOGaJBDLw1Yv/download

Assumptions

Assume the following:
- Passing/failing/dropping out of a course offer
  a) A student passes a course offer if his grade is 5 or higher.
  b) A student fails the course offer if he gets a grade less than 5.
  c) A student can drop out of a course offer, in which case he will not get a grade and he will not fail/pass the course offer.

- GPA of a student is computed as the weighted average of his grades *at all passing course offers*, with each grade weighted by the course's ECTS.

- A student can register more than once for the same course (independent of her grades), but only at different course offers. Whenever a student is registered at two different offers of the same course, her grades are considered as these correspond to two different courses. Similarly, her ECTS are added normally.

- A student completes a degree if she gets a passing grade to a set of course offers whose ECTS sum up (or exceed) the total ECTS required for the degree. To improve their GPA -- or to learn more -- students can continue registering to courses even after they cover the required ECTS for their degrees.

- A student is considered active if she still did not acquire the minimum ECTS in at least one degree that she is enrolled, i.e., she did not yet complete the degree.

- A course can be offered multiple times per year, possibly from different teachers. There can be more than one concurrent offers of a course. There is exactly one responsible teacher for each course offer.

- A student can be enrolled to more than one degree concurrently.

1. Load all data in the provided postgreSQL image (20 pt)

The input files should be read from **folder /mnt/ramdisk/tables/**. You need to connect to an existing database **(named "uni")** for loading the data. Your script will be given 18 minutes to load all data. After this, it will be killed.

Deliverable:
You will submit the url of a **public git repository** that contains the code for creating the database and loading all data. The repository should not contain folders. For an example of the expected structure of the repository, see https://github.com/papapetrou/2id70Test

The git repository should include:
(a) Bash script load.sh, which will be automatically called from our tests to initialise the database creation and loading process.
(b) SQL commands and/or any additional java source code to be invoked from load.sh, for loading all data. If you provide Java source code, you need to make sure that it can be compiled from within load.sh, using only javac and java (version 10). You cannot install additional tools in the VM. Also, you should not use third-party code, programs, or libraries, apart from Postgresql and Postgresql jdbc driver (if needed, include the jdbc driver in the git repository).
(c) Three SQL queries, in a file called queriesAfterLoading.sql. Each query should be in a single line, with no empty lines between them. The purpose of these queries is to help you (and us) verify the correctness of your schema and loading process. If your schema cannot correctly answer these queries, then most likely you should rethink your design or loading process. The queries should provide answers to the following questions:
   I. For Course offer with course offer id equal to 1, return the following attributes in the described order: All attributes for the course offer, the course, the degree which this course belongs to, and the teacher that teaches the course offer. Correct answer will be a single row, with the following schema:
   `[int CourseOfferId, int CourseId, int Year, int Quartile, varchar CourseName, varchar CourseDescription, int DegreeId, int ECTS, varchar Dept, varchar DegreeDescription, int TotalECTS, int TeacherId, varchar TeacherName, varchar Address, int BirthyearTeacher, char Gender]`
   In the provided dataset, part of the correct answer will be as follows (strings replaced with … for brevity):
   `[ 1 | 36686 | 2008 | 4 | … | … | 6747 | 3 | … | … | 174 | 34727 | … | … | 1965 | M]`
   II. For student with student registration id=140, return all attributes describing the course offers where this student was a student assistant, all attributes corresponding to this student, and all attributes corresponding to the degree where this student is registered. Correct answer will be rows adhering to the following schema (potentially multiple rows):
   `[int CourseOfferId, int CourseId, int Year, int Quartile, int StudentId, varchar StudentName, varchar Address, int BirthyearStudent, char Gender, int DegreeId, varchar Dept, varchar DegreeDescription, int TotalECTS]`
   In the provided dataset, the query will return a single row. Part of the correct row will be as

follows (strings replaced with … for brevity):
```
[24182 | 36532 | 2013 | 4 | 272488 | …  | … | 1989 | M | 9363 | … |
… | 153]
```
III. Return the average grade for student registration id=140, accounting for all course offers where this registration id was registered to (even the ones the student did not pass). Correct answer will be one row adhering to the following schema:
```
[float avg]
```
In the provided dataset, the query will return the following row:
```
[5.0909090909090909]
```

The git will be automatically pulled, and the loading will be executed with the following command (notice the timeout):
```
    timeout 18m bash ./load.sh
```
Then the results of your queries contained in queriesAfterLoading.sql will be compared with the ground truth (the correct results).

Hints:

(a) Everything for this question (and the whole exercise) can be answered using only simple bash commands and commands within psql. It is suggested to follow this approach.

(b) When creating tables, you are allowed to use the UNLOGGED keyword. This will drastically increase the loading performance and reduce the disk space required for storing the tables. If you want, read more about it here: https://www.compose.com/articles/faster-performance-with-unlogged-tables-in-postgresql/ and here https://www.postgresql.org/docs/10/sql-createtable.html

(c) Use ANALYZE after loading all data. This can improve your performance. If you want, read more about it here: https://www.postgresql.org/docs/10/sql-analyze.html

(d) Choose the database structure carefully, and do not be afraid to adapt when you start working on Question 2. The database structure affects the performance you can get for the queries of Question 2.

(e) You can assume that the input data is correct. Therefore, you do not need to enforce constraints or foreign keys, if these are not beneficiary for performance.

2. Optimize your database for an expected workload (80 pt)

You can use the following approaches:

a) Different types of indices;
b) Different types of views;
c) Modifications in the database structure (part of Question 1 — in this case modify the files relevant to question 1);
d) Writing the SQL queries in a better way.

The expected workload is described below, in natural language. You need to write the queries in SQL, and then optimise the database for answering these queries. You can assume that the described queries are *representative* of the real query workload, both in type and in frequency.

Conditions:

- The total database (after import) should take around 8 GB. You have an additional 4 GB disk space to use for answering this question (a total of 12 GB) — using more space will fail in the automated test, and the database may not behave as you expect. You have 5 minutes for creating the auxiliary structures to prepare the database. (Hint: Check on command ANALYZE)

- Each submission will be given 15 minutes to complete all queries. Submissions that do not complete all queries within this time will be killed. All points will be lost on the non-completed queries, even if these are correct.
- The SQL queries should be written in q2Queries.sql — one query per line, with no empty lines between them. The file should have a total of 8 lines, even if you do not know all answers. If you want to skip a query, then write "SELECT 0;" at the corresponding line. This will have zero penalty in terms of time. **If you forget a line in this file, then all remaining SQL queries will be mapped incorrectly at the automated tests, and you will not get any points.**
- For each query that completes in time and returns correct results (identical to the ground truth), you will get 5 points. No points will be given for 'partially correct' queries.
- Result correctness will be checked on **our data**, which is of similar size/structure to the provided one. We will use your load.sh to construct and load your database with the new data, and then execute your SQL queries on the constructed database.
- Be careful to return the tuple attributes in the correct order. For example, if the query in natural language asks for the student ids and their names, then your sql query should return tuples with the *first* attribute being student id and the *second* attribute being the name of the student.

Deliverables:
The following files should be included in the git repository of question 1.

1) Code **(60 pt)**
   A. File q2Create.sql: SQL code to create the auxiliary structures (20 pt)
   B. File q2Queries.sql: SQL queries corresponding to the natural language queries given below. Query at line x from file q2Queries.sql should map to the natural language query Q_x given below (40 pt).

2) Documentation **(20 pt)**
   A. A description (max. 1/2 page) on the followed optimization process, in steps (10 pt)
   B. A description (max. 1/2 page) on the chosen optimizations, the space required from each, and performance experiments with/without these optimizations (10 pt).

The documentation will be saved in a file documentation.pdf, and included in the git repository. Be clear and concise.

Example (do not include this in the submitted files):
Q_0: Student administration needs a list of all student names and addresses that are enrolled to a degree of department %1%, which will be given as a parameter.
Solution:
```
SELECT StudentName, Address FROM Students, StudentRegistrationsToDegrees, Degrees
       WHERE Degrees.Dept=%1% AND
             Degrees.DegreeId=StudentRegistrationsToDegrees.DegreeId AND
             Students.StudentId=StudentRegistrationsToDegrees.StudentId;
```
e.g.,
```
SELECT StudentName, Address FROM Students, StudentRegistrationsToDegrees, Degrees
       WHERE Degrees.Dept='standard a' AND
             Degrees.DegreeId=StudentRegistrationsToDegrees.DegreeId AND
             Students.StudentId=StudentRegistrationsToDegrees.StudentId;
```

Notice: (a) the solution should be given in a single line, with no new line characters. We break the lines in the example only for matters of readability. (b) the order of output attributes in the natural language description should be the same with their order in SQL. Also, no additional attributes should be printed. (c) parameters in your SQL should be denoted with %1%, %2%, … These will be identified by the automated testing suite and replaced with real values.

Queries:

Q_1: For each student and each degree that this student is enrolled to, student administration needs to be able to print a report that includes all passed courses taken by the student (i.e., the ones that have a grade at least 5), and their corresponding grades. Write the SQL query that takes as input parameters the student id (denoted as %1%) and the degree id (denoted as %2%), and returns all course names, and the grades that the student achieved, sorted by year, quartile, and course offer id. This query will be executed 100 times sequentially in our testing workload, with different parameter values. The correct answer will adhere to the following schema:
`[varchar CourseName, int grade]`

Q_2: You want to award your excellent students. List the student ids of all students that never failed a course during at least one of their completed degrees and had a GPA higher than a query parameter %1% (between 9 and 10). The list should be sorted by student id and not contain duplicates. The query will be executed 10 times sequentially in our testing workload, with parameter values 9.0, 9.1, 9.2, … 9.9. The correct answer will adhere to the following schema:
`[int studentId]`

Q_3: The ministry of education says that you should promote the participation of female students in your university. You want to prove that this is not an issue at TU Eindhoven. Considering only the active students, compute the percentage of female students for each degree id (see the definition for degree completion at the preceding assumptions). The correct answer will adhere to the following schema: `[int degreeid, float percentage]`. Answers should be ordered on degree id.
NB: A student enrolled to a degree might not have (successfully) taken a course yet.

Q_4: The ministry of education says that you should promote the participation of female students in the degrees offered by department %1%. Compute the percentage of all female students (both active and with completed degrees) in the degrees offered by department %1%. The query will be executed 10 times sequentially in our testing workload, with different parameter values. The correct answer will adhere to the following schema: `[float percentage]`.

Q_5: TUe student administration wants to examine the introduction of a variable passing grade per course. For example, course 2id70 might have a passing grade 5, whereas course 2id50 might have a passing grade 4 out of 10. The idea is that each course offer should have at least 75% of the students passing. Write a query that will enable the student administrators to find out the percentage of students passing all offers of all courses, where the passing grade %1% is given at query time (i.e., grade should be *greater than or equal* to %1% for the student to pass). Students without a grade (e.g., because they dropped the course) should be ignored in this computation. The query will be executed 5 times sequentially in our testing workload, with parameter values 4, 5, 6, 7, 8. Results should be ordered on course offer id. The correct answer will adhere to the following schema: `[int courseid, float percentagePassing]`.

Q_6: The dean argues for an alternative definition of excellency, compared to query 2. The new definition is as follows: excellent students of each course offer should be the ones that got the highest grade (not necessarily a 10), and the excellent students of the quartile are the ones that were excellent in at least %1% courses. Write the query to retrieve the Student Ids of excellent students for Quartile 1 of 2018, and the number of courses they were excellent, assuming that the value of %1% is a query parameter. The query will be repeated 3 times sequentially in our testing workload, with parameter values 1, 2, and 3. The output should adhere to the following schema: `[int studentid, int numberOfCoursesWhereExcellent]`.

Q_7: Student administration hired a consultant to find out which parameters influence the performance of students. The consultant wants to analyze the GPA of all active students, considering the following attributes: degree, year of birth, and gender. Write a query to construct a cube in Posgresql on these attributes in the aforementioned order. The records should be ordered ascending on degree, then year of birth, and then gender (Female precedes Male). The expected output has the following format: `[int degreeid, int birthyear, char gender, float avgGrade]`.

Q_8: In theory, each course offer gets at least one student assistant for every 50 students. Write a query to list the course names, years, and quartiles of all courses that received less student assistants than the required ones. The list should be sorted by CourseOfferId. The expected output has the following format: `[varchar courseName, in year, int quartile]`.

For clarifications and questions about the milestone (or useful pointers), please use canvas discussions so that your fellow students can also see the answers.

Good luck!