LaSalle College
Montréal

Project 3: 20%

Course Identification

Name of program – Code:	
Course title:	**DATABASE II**
Course number:	420-BD2-AS
Group:	07438
Teacher's name:	Jean-François Parent
Duration:	Étendue
Semester:	Fall 2023

Student Identification

Name: ____	Student number: _______________
Date: ____________________	Result: ________________

☐ I declare that this is an original work, and that I credited all content sources of which I am not the author (online and printed, images, graphics, films, etc.), in the required quotation and citation style for this work.

Standard of the Evaluated Competency
Statement of the evaluated competency – Code
Use a database management system – 00Q7
Evaluated elements of the competency 00Q7
1. Create the database. *2. Formulate read requests, insertion requests, modification requests and deletion requests.* *3. Ensure data confidentiality and consistency.* *4. Program automated data processing operations.* *5. Save and restore the database.*

Instructions

- Create a database (schema) according to the requirements
- It is the teacher’s responsibility to identify language errors. If such errors are found, teachers may apply a penalty of up to 20% of the grade (IPEL – Article 5.7).
- Plagiarism, attempts at plagiarism or complicity in plagiarism during a summative evaluation results in a mark of zero (0). In the case of recidivism, in the same course or in another course, the student will be given a grade of '0' for the course in question. (IPEL – Article 5.16).
- Submit dates on Omnivox must be respected.

- Please consult the correction grid at the end of document for more details.

TOTAL: 100 POINTS

Because this work must be done at home, if we think that the answers are not yours, the department reserves the right to complete your evaluation with a virtual meeting to verify that you attained the required competencies.

Project 3

Directives

Note: This is **NOT** a team work

1. For objects names, only use the recommended characters (A-Z (objects names must begin by a letter), 0-9 and the underscore (_).

2. Each field must have the correct data type. The size must be big enough to store all required data, but not too big to use too much space on the disk.

3. Unless specified otherwise, each field must contain data and must not accept NULL values and/or empty strings ('').

4. Specify a default value where you can do it.

5. Specify constraints where you can do it to prevent users to enter unwanted data.

6. Each table must have a primary key which identifies each row with a unique value (GUID).

7. You must create indexes for each field used in a search, for each foreign key, and for each column which must be unique.

8. Each SQL query must use good indentation (for example, no query on a single line).

9. Write comments to describe what the SQL code does.

10. For each table and object, you must write the revision history. It must contains the date, your name, your student number and the modifications made. For tables, enter this information in the comments of the primary key. For the other objects (stored procedures, functions, etc.) you must inscribe these informations in the comments. For example:
Table:

```
2023-10-29          Bill Torvalds (1244556)          Created the customers table.
2023-10-31          Bill Torvalds (1244556)          Added the firstname column.
```

Stored procedure:

```
--Revision history:
--2023-11-01     Bill Torvalds (1244556)          Created the procedure customers_select.
--2023-11-03     Bill Torvalds (1244556)          Fixed the ORDER BY bug.
```

11. All the newly created tables must include the 2 fields for the date and time of creation/modification (like in project 1).

Project description

This project must re-use the database created in previous project. Marks will be awarded only for the new elements. It is thus not mandatory to fix the errors made in previous projects unless these elements are required in project 3 (in other words, the new SQL code must not crash).

Improve the database created in the previous projects by implementing the following requirements:

(9pt)

- In the table that contains the **modifications** made in the reports table, add a field for the report ID, to always know which report was modified.

Field name	**Modification date**	**Old value**	**New value**	**Report ID (the new field)**
RESULT	2023-10-01	57	60	112233...
STUDENT_ID	2023-10-02	ACDEF-01..	987654-BA...	445566...
TEACHER_ID	2023-10-03	112233-44...	FFEEDD-CC...	778899...

Modify the **trigger** for the UPDATE of the reports so that it fills automatically the newly created field. This trigger must also display, on the screen, a sentence telling how many fields were modified during the UPDATE.

(10pt)

- Create a **trigger** for the INSERT of the **reports**. It should generate an error and cancel the INSERT if the exam date is in the future (display a clear error message for the users). The message must contain the date that was not accepted in a format easy to read for the users (at least day/month/year, and the month must be written in text, for example November). Generate a valid error number with the 3 last numbers of your student number. For example if your student number is 19991**122**, your error number must also end with **122**.

(7pt)

- Modify the **procedure** which inserts **reports** so that it returns, with a parameter, how many results (marks) exist for that student in the database. If this total is 20 or more, do not perform the INSERT and return an error message to explain that a student (display at least his/her student number) reached the maximum number of courses in the school. Generate a valid error number with the 3 last numbers of your student number and add 1(+1).
Ex: 19991122 =>12**3**.

(21pt)

- In the courses table, modify the **primary key** so that it will now use an auto-increment number instead of a GUID.

  Create **procedures** to INSERT and DELETE in the **courses** table. The INSERT procedure must return, with a parameter, the ID of the course that was just created/generated.

  For the INSERT, if a course name starts with 'MS' (case-insensitive) replace the beginning of the course name with 'Microsoft' and keep the remaining text (for example 'Ms Word' will be saved as 'Microsoft Word' in the database).

  Create a **single** trigger for INSERT and DELETE in the courses table. The trigger must display that a course (display its name) was **added** or **deleted**. In the sentence also show the current date (ex: year/month/day). Use variables to have only one line containing DBMS_OUTPUT in the whole trigger.

(31pt)

- Add an allergies field to the students table (to enter, for example, peanuts, eggs, penicillin, etc.). Do not allow NULL values. Specify some text by default when a student does not have any allergies.

  It is very important that the students' allergies are detected by the school staff. So modify all procedures and functions (INSERT, UPDATE, SELECT, DELETE) where the firstname and lastname fields are managed and add the new allergies field.

  Modify the procedures for the INSERT and UPDATE of students. If the student has an allergy to peanuts (or Peanuts, PEANUTS, etc.) save the name of the allergy in uppercase so that the school staff will be able to see it clearly.

- Create a sequence which starts with the 3 last numbers of your student number and which increments by 5. For example, if your student number is 19991122, your sequence must start with 122, et then use 127, 132, 137, etc.

  Use this sequence as the default value for the student number field, with the prefix **STUDENT_**. For example if your student number is 19991122, the default value should be STUDENT_122, and then STUDENT_127, STUDENT_132, etc. The same student number can never be used more than once.

(11pt)

- Create a function that receives a salary ($), a hourly amount for the insurances ($) and the number of worked hours, and that returns the amount of the paycheck. This function should substract the two amounts ($), and multiply the result by the number of worked hours. This will give the amount, in dollars ($), of the paycheck. For example, if a teacher earns 20$ per hour and that she/he pays 2$ per hour in insurances, the hourly pay is actually 18$ per hour (20$ – 2$). This amount must then be multiplied by the worked hours to calculate the paycheck. For example, 18$ per hour multiplied by 10.5 hours = 189$.

  Modify the functions which SELECT the timesheets (the functions with a JOIN) and use the function you just created in the SELECT to show the paycheck for all teachers. Give an alias (with a significant name) to this new field.

- To avoid duplicate entries in the timesheets, add a UNIQUE index so that each combination TEACHER_ID + START_DATE is never entered twice.

(10pt)
Create a file named **TEST_YOURSTUDENT*NUMBER*.sql** (use your actual student number in the filename) to test each **procedure** and each **function** added or modified during this project (including the procedures/functions to test the added and modified triggers). You should have a total of 10 objects to test. Call each procedure and each function at least once (display the parameters in output when applicable). This script should not generate any error. The script must activate the display on the screen (SERVEROUTPUT) to facilitate the tests.

(1pt)
When your database is ready, export it into a text file (.sql). The filename must contain at least your lastname and the date when the backup was made.

This exported file only creates the objects and not the database. You must thus add manually the SQL code to create and use a database (schema) which has the same name than your database. Use the password “123”. The script must create all the objects (database/schema, tables, procedures, etc.) so test it properly.

Your script must NOT delete any existing database/schema if it already contains object(s). Your script must be error-free.

Submit the **two files** (DB creation + test script) on Omnivox.