ECE356 F20 – Lab 3 – More SQL, Stored Procedures, Transactions and DB Performance

Due date: November 13th at 11pm Waterloo time

Group work policy: In this lab, students are expected to work in groups no larger than 3. Groups of 1 or 2 are fine.

Weight: This lab is worth 10% of your final course grade (1/5 of the weight in the Assignments/Labs category).

Objective:

- develop a working knowledge of stored procedures **(graded)**
- gain familiarity with the SQL Transaction Control Language **(graded)**
- experience running queries over a large data set (**not graded**, but highly recommended)
- apply basic query optimization principles **(graded)**

Submission: All parts of this lab must be submitted to the Lab 3 dropbox in LEARN by the due date, which is shown below the title of this document. **You must enroll in a Lab 3 group in LEARN before you can submit to the group dropbox.** Your submission will contain multiple files since the lab manual has multiple parts; each part of the manual describes the required submission format. Should you choose to resubmit the deliverables prior to the deadline, please ensure that you **upload all required files each time** as the dropbox is programmed to overwrite the old submission with the new.

Part 1: Stored procedures [30%]

Start with the normalized Employee database shown in Figure 1. You should use provided *createTables.sql* SQL script to create the appropriate tables and load a sample data set.

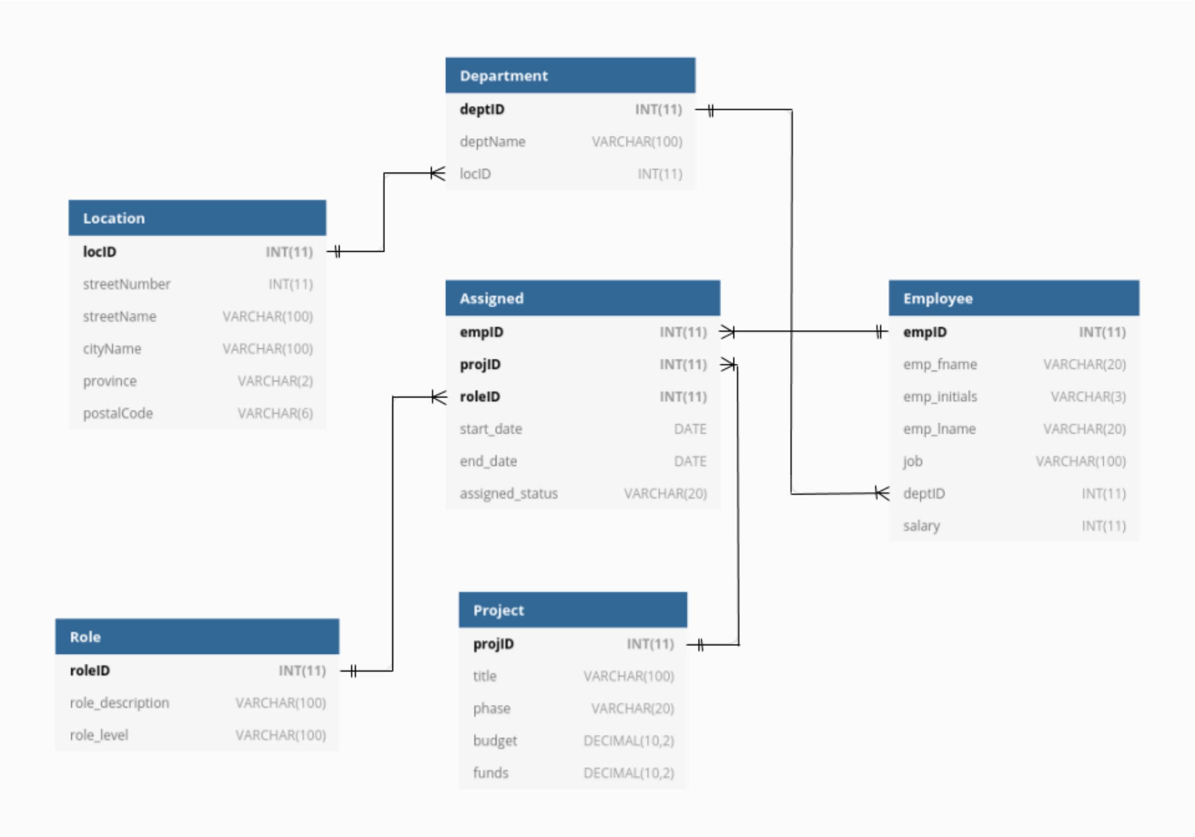

Figure 1: Normalized employee database schema.

Write a stored procedure called **sp_pay_raise**. The procedure should try to raise the salary of a given employee by a given percentage and return an error code associated with this operation. The given percentage must be greater than 0% and less than 10%. If the percentage is not within the valid range, the salary is not increased and the procedure has no effect. Tables 1 and 2 show the input parameters for this procedure, and the error codes that must be returned under different conditions.

Input/Output	**Parameter name**	**Interpretation**	**Data type**
Input	inEmpID	ID of employee to be considered for a raise	int
Input	inPercentageRaise	raise amount as a percentage of salary	Double(4,2)
Output	errorCode	error code	int

Table 1: Parameters

Error Code Value	Condition
−3	Employee does not exist AND the percentage specified is outside the range from 0% to 10% inclusive. Salary is not raised.
−2	Employee does not exist AND the percentage specified is within the range from 0% to 10% inclusive. Salary is not raised.
−1	Employee exists AND the percentage specified is outside the range from 0% to 10% inclusive. Salary is not raised.
0	No error. Inputs are valid and salary is raised.

Table 2: Error code values.

Submission instructions: For this part of the lab, the deliverable is a modified version of the provided *part1.sql* file containing your implementation of the **sp_pay_raise** stored procedure. Please preserve the DROP PROCEDURE IF EXISTS statement at the top of this SQL script.

Part 2: Transaction Management [30%]

Start with the normalized Employee database introduced in Part 1 (see Figure 1). At this point, you may want to reload all your tables using the provided *createTables.sql* SQL script.

Create a transaction within a stored procedure called **sp_pay_raise_kitchener** that will raise the salary of all employees in Kitchener by 3%, provided that no salary of any employee is more than $50,000 after being raised. If any salary after the raise in Kitchener exceeds $50,000, then the pay raise is cancelled for all employees – that is, the entire transaction is rolled back (see COMMIT/ROLLBACK commands in the SQL TCL). If the pay raise is applied and the transaction is committed, you should return 1 in your stored procedure. Otherwise, you should return 0.

Note 1: You must use a transaction to solve the problem, as otherwise your code may not work correctly if the stored procedure is invoked concurrently by two clients, or if a failure occurs during execution.

Note 2: The stored procedure must return a value (0 or 1) as a result set containing one column and one row. The stored procedure must not use any output parameters.

Submission instructions: For this part of the lab, the deliverable is a modified version of the provided *part2.sql* file containing your implementation of the **sp_pay_raise_kitchener** stored procedure. Please preserve the DROP PROCEDURE IF EXISTS statement at the top of this SQL script.

Part 3: Load the Yelp data set [required for Part 4] and run queries [optional]

Important Note: Before using the Yelp data set, you **must** accept the license terms by completing the web form at https://www.yelp.com/dataset/download. You do not have to download the data set directly from Yelp since it is included in the Lab 3 starter code archive, but it is important that you understand and accept the terms of the license.

Yelp is a website that maintains consumer reviews of businesses. Its data set is similar to what you might find in a data warehouse: a small number of tables with a very, very large quantity of data. Yelp regularly provides access to a small subset of the data.

Your task is to write the SELECT statements to answer the questions proposed here. This task is not graded, but it is provided to help you gain experience with loading a large data set into a database, and with executing queries over such a data set.

In order to reduce the data loading time and the running time of the queries for this exercise, we will use a reduced version of the Yelp data set containing only 20% of the original reviews. Please load this database on your own *personal computer* in preparation for Part 4 of this lab, which requires precise measurements of query execution time. The procedure to load the data set is to execute the provided *load_yelp_db_small.sql* SQL script, which loads the data set for you using the LOAD DATA method. The script refers to six CSV files containing the filtered data, which are also provided in the starter code archive.

Follow the steps below to load the reduced Yelp data set:

1. Download the Lab 3 starter code archive from LEARN and save it on the computer from which you access the MySQL database server. This could be your personal computer, or one of the ecelinux hosts.
2. If you are using your own MySQL server, run the mysql client as the MySQL root user, and execute the following command:

   ```
   SET GLOBAL local_infile = 1;
   ```

   This step is necessary to enable local data loading, which could be disabled by default.
3. Start the mysql client as an ordinary MySQL user using an additional command line argument: `--local-infile=1`. For example, on a Linux host you might do so as follows from the bash shell:

   ```
   mysql -u user_$USER --local-infile=1 -p ece356db_$USER
   ```
4. Run the *load_yelp_db_small.sql* script using the `source` command, and wait for the database to be created. Be patient as this step can take a (very) long time.
5. If the SQL script is not able to find the data files then open the script in an editor, find each line containing the keywords "LOAD DATA LOCAL INFILE," and adjust the path to the CSV file. Then repeat step 4.

After the database is created, you can check if the data was properly loaded by executing simple query statements. Examples of such queries can be found commented in the *load_yelp_db_small.sql* script. Search for these queries, copy them, and execute them if you wish. You can also use your own queries if you like.

After loading the Yelp data set successfully, your mission is to write SQL queries for the following questions:

(a) What is the id and review count of the user whose name is 'Shila'?

(b) Which business received the greatest number of reviews? (provide the business id, business name, and review count)

(c) Find the average number of reviews per user across all users in the database.

(d) The average rating written by a user can be determined in two ways:

   1 By directly reading from the Users table "average stars" column

   2 By computing an average of the ratings issued by a user for businesses reviewed

   For how many users do these two quantities differ by more than 0.5?

(e) What fraction of users have written more than 10 reviews?

(f) For each user who wrote more than 10 reviews, what is the average number of characters of these reviews?

Part 4: Database Performance [40%]

The ***explain*** command determines the query execution plan for a query. The query plan will show you where primary keys, foreign keys or additional indexes are used, and allow you to determine where they may be needed (i.e., you will find out where a sequential search is used since there is no index yet).

In this part of the lab, you will improve the performance of two queries on the Yelp database. We recommend that you complete this part of the lab on your personal computer as this will allow you to measure the running time of your query more precisely than on a shared server. In order to make this exercise more manageable, we will use the reduced Yelp data set you loaded earlier in Part 3.

Once you have created the reduced Yelp database, use the explain command to analyze the following queries:

1. Count the reviews written in May 2014. You should use the Review table and name the output column as *count*.
2. A user *(see Review.userid='KGYM_D6JOkjwnzslWO0QHg')* exists in the data set who has written multiple reviews. Write a query that lists the user's name (as name), as well as all the review ids (as review_ids), and business names (as business_names) assigned in each review. You must join the User_data, Review, and Business tables in this query. *Hint: Your output should have a total of 13 rows.*

Your task is to examine the query plan for each of these queries, and add any other indexes that may improve the performance of the query.

For each query:

a) Write the SQL code.

b) Run the explain command to output the "before" query execution plan for your query.

c) Add a single index that can improve the running time of your query. There may be multiple solutions, and you must find the best one (i.e., the one that yields the biggest speedup) to earn full credit.

e) Run the explain command again to output the "after" execution plan, showing that your index is indeed being used.

Submission instructions (three files required):

1. Using the provided template file *part4-1.sql*, add the SQL DDL code to first drop and then create the optimal index for the first query, and then add your SQL DML code for the query.
2. Using the provided template file *part4-2.sql*, add the SQL DDL code to first drop and then create the optimal index for the second query, and then add your SQL DML code for the query.
3. Write a report that presents the "before" and "after" execution plan for each query, quantifies the improvement in performance due to your chosen index, and discusses any alternatives (if applicable) considered. Query running times using your chosen index, as well as using any alternative indexes, should be reported as an average and standard deviation computed over five repetitions. Submit your report as a PDF file named *part4.pdf.*