Ceng352 - Database Management Systems
Project 3 : Database Administration
(Tuning, Triggers, Security)

Spring 2019

1 Project Description

In this project you are asked to perform some basic DBMS administration tasks:

1. *Database tuning:* given a database and a workload, tune the database to make the workload faster.
2. *Making the database active:* add triggers to a database that automatically reacts to actions performed on the database.
3. *Security:* protect the data inside a database using views and access control lists.

You are the database administrator for the METU Student Affairs Information System (METU SAIS). In this project, the METU SAIS database is generated by the Java program *"DatabaseGenerator.java"*, and imported by the script *"import_database.sql"*.

The database has the following schema:

```
Student (student_id, first_name, last_name, age, room, dormitory)
Professor (professor_id, first_name, last_name, title)
Registered (student_id, department)
Faculty (professor_id, department)
Advisor (student_id, professor_id)
Course (course_code, course_name)
Teaching (professor_id, course_code)
Enrolled (student_id, course_code)
```

The `Student`, `Professor` and `Course` tables contain information about the students, the professors and the courses, respectively. The `Registered` table stores which student is registered to which department(s). The `Faculty` table stores which professor is faculty member of which department(s). The `Advisor` table says which student is advised by which professor(s). The `Teaching` table says which professor is teaching which course(s). The `Enrolled` table stores which student is enrolled to which course(s). For each relation, the attributes that form the primary key are underlined. Additionally:

```
Registered.student_id is a foreign key referring to Student.student_id
Faculty.professor_id is a foreign key referring to Professor.professor_id
Advisor.student_id is a foreign key referring to Student.student_id
Advisor.professor_id is a foreign key referring to Professor.professor_id
Teaching.professor_id is a foreign key referring to Professor.professor_id
Teaching.course_code is a foreign key referring to Course.course_code
Enrolled.student_id is a foreign key referring to Student.student_id
Enrolled.course_code is a foreign key referring to Course.course_code
```

2 Task 1 : Database Tuning

The staff at METU Student Affairs access the database extensively by running *"QueryTester.java"*. In *"QueryTester.java"*, you'll see that there are seven kinds of queries that the staff run very often (e.g. finding the number of students registered to a specific department). You are in charge of administrating the database and optimizing it for performance. As you saw, *"QueryTester.java"* takes a while to run: you need to improve that.

You should make the *"QueryTester.java"* run faster as much as you can. You are allowed to use no more than 7-8 indexes (in addition to the default ones created by PostgreSQL). Create a file *"tune.sql"* where you put your indexes. Also, for each index, explain in 1-2 sentences (as a comment in *"tune.sql"*) why you use that particular index. Your file should end with the information of how long does it take for *"QueryTester.java"* to run before and after applying your tuning.

In this assignment, you will only tune the performance by adding indexes. You should create indexes on one or more attributes. You may create several indexes per table, and may decide which one to make a primary index. Be aware that, by default, PostgreSQL builds a primary index on the primary key of each table. You should focus only on optimizing the workload *"QueryTester.java"*. Some other queries may become slower as a result of your optimization, but that's ok as long as test queries run faster. Note that in practice, physical database tuning is more involved and includes table partitioning, materialized views, and other.

To create an index, use the following command:

```
CREATE INDEX index_name ON table_name(column_name1, ...);
```

To make an index the primary index, use the following command:

```
CLUSTER index_name ON table_name;
```

Hint: You may want to check what plan PostgreSQL picks for a given query, and for that you may use EXPLAIN, e.g.

```
EXPLAIN SELECT COUNT(*) FROM Professor WHERE title='Assistant Professor';
```

Testing and Running Your Solution:

For this question, you need to create a file *"tune.sql"* that you will turn-in. After you create the file or update it, do the following to test your solution:

```
createdb project3
psql -f import_database.sql project3
psql -f tune.sql project3
java QueryTester 1
```

If you need to start from scratch (i.e. remove your indexes and get back default indexes), the `dropdb` command may also be helpful. You can use it as:

```
dropdb project3
```

3 Task 2 : Triggers

It is often useful to have the DBMS perform some actions automatically in response to operations on the database. The METU SAIS database already does that to some extent. For example, if you delete a student from the database, the corresponding entry in the `Registered` relation will also be deleted automatically.

Sometimes, we would like the DBMS to perform more complex actions in response to more complex events. For this task, add two triggers to the database that each satisfies one of the followings:

1. A student must be registered to at least one department. A student can be registered to at most two departments. Do not allow number of registered departments for a student to not meet this condition and print a message in such cases of trials.
2. If number of enrolled students for a course gets less than 5, delete corresponding rows from `Course`, `Teaching` and `Enrolled` relations and print a message.

A sample trigger is given in the file *"sample_trigger.sql"*. This sample trigger raises an error message when someone inserts a new student with age older than 50. As a comment inside *"sample_trigger.sql"*, you will find the detailed instructions for loading and running this sample trigger.

Here are the steps you need to take to complete this task:

- Put your trigger definition in a file called *"trigger.sql"* (to start, you can simply copy & paste the contents of *"sample_trigger.sql"*).
- Load your trigger definition with (note that you may need to specify the absolute path): `\i trigger.sql`
- Add your trigger with the CREATE TRIGGER statement as shown in the example and also in the file *"sample_trigger.sql"*.

Testing and Running Your Solution:

Try to perform actions on the database and verify that the triggers fire at the appropriate times and perform the appropriate actions.

4 Task 3 : Security

The METU Student Affairs enforces a strict access control policy to its database. It distinguishes three kinds of users:

- guests
- secretaries
- administrators

The office also classifies the data into two overlapping categories:

- Private Data: data about students (including their registered departments), their professors (including the departments of the professors), and their advisors; data about courses, professors teaching courses, and students enrolled to courses.
- Public Data: names, titles and departments of professors; names of courses.

The access control policy is the following. The guests is the least privileged: secretaries and administrators are allowed to access everything that the guests can access.

P0 Guests may only read the public data.

P1 Secretaries may read all private data; they may change (update, insert, or delete) only the advisor (i.e. assign advisors for students), teaching (i.e. assign professors to courses) and enrollment (i.e. enroll students to courses) data.

P2 Administrators may read, insert, update, and delete all private data.

Your task is to write the SQL access policy statements that enforce this policy. Add all your commands to a file called *"access_control.sql"* and hand-in this file. Please proceed as follows.

4.1 Creating roles and users

As a first step, create one database role for each of the three kinds of users: guests, secretaries and administrators. Call these roles *guest*, *secretary* and *metusaisadmin* respectively (make sure not to use capital letters, they seem to cause problems). To create roles, use the `CREATE ROLE` statement. You should not need to use any of the options available with this statement. Database roles are like groups: you will grant privileges to roles and will associate users with roles.

Now that you have the roles, create one test user for each role (*guest1*, *secretary1*, and *metusaisadmin1*) using the `CREATE USER` statement. Make sure that the user is associated with the appropriate role and the user has a password that he/she can use to login.

Testing and Running Your Solution:

Try to connect to the project3 database as each user. Verify that you can connect but that you cannot view any data nor perform any inserts/updates at this time.

4.2 Creating public views and granting access to these views

Create one or more views on the database. These views should expose only information classified as public data. They should hide everything else. Then, grant access of these views to the appropriate role(s). For this, use the `GRANT STATEMENT`.

Testing and Running Your Solution:

Verify that all your users can see the public data.

4.3 Creating other views and granting other privileges

Create other views and grant other privileges as appropriate to implement the METU Student Affairs access control policies. Note that you do not always have to create views. When appropriate, you can grant access rights directly to existing tables.

Testing and Running Your Solution:

To test your solution, try to run psql as each of the three users and check what operations you are allowed to perform.

5 Starter Code

The starter code is in the package named *"ceng352_project_3.zip"*. There are 5 files in starter code package: *"DatabaseGenerator.java"*, *"QueryTester.java"*, *"dbconn.config"*, *"import_database.sql"*, and *"sample_trigger.sql"*. To run the starter code, make following preparations:

1. In order to connect to a PostgreSQL database from Java, you need a database-specific JDBC driver for PostgreSQL (can be downloaded from `https://jdbc.postgresql.org/download.html`). You should add this driver to your classpath before running the starter code.
2. Start up the PostgreSQL server on your local computer and create a database named `project3`.
3. Edit the username and password for PostgreSQL on *"dbconn.config"*.

Now, you can use following commands on terminal to compile and run the project:

```
javac DatabaseGenerator.java
javac QueryTester.java
java DatabaseGenerator 100
psql -f import_database.sql project3
java QueryTester 1
```

Be aware that:

- The last two lines will take a while to finish.
- You can also run *"import_database.sql"* script from within psql with the following command: `\i import_database.sql`

- You may need to specify the absolute path to the script and also modify *"import_database.sql"* to add absolute pathnames to all filenames.
- The numbers 100 and 1 represent the scale factors; normally you wouldn't change them, but if you want you may increase them to test your database tuning solution more extensively.

6 Submission

You are asked to submit a *".zip"* package with all files from all tasks (*"tune.sql"*, *"trigger.sql"*, and *"access_control.sql"*) in it. You should name your file as *"your_student_id.zip"* using your 7-digits METU student ID. You should submit the *".zip"* package to ODTÜClass before the deadline.