School of Engineering and Computer Science

# SWEN 304 **Database System Engineering**

## Assignment 3

Due: Friday, 22 September at 23:59
Where: Cotton Building, second Floor, SWEN 304 Hand-In Box

The objective of this assignment is to test your understanding of database concepts such as stored procedures, user defined functions, and triggers, and your ability to apply them in PostgreSQL. It is worth 5.0% of your final grade. The assignment is marked out of 100.

**Note**: At the end of this assignment you will find a brief instruction on PostgreSQL.

## Question 1. Triggers [40 marks]

A university is using a relational database to manage its data on students, and their majors. Suppose the following relation schemas are part of the underlying database schema.

- MAJOR (mCode, name)
  with primary key {mCode}
- STUDENT (sId, firstName, lastName, mCode, pointsEarned)
  with primary key {sId} and foreign key STUDENT [mCode] $\subseteq$ MAJOR [mCode]

The following additional requirements are known:

- A major may not be deleted nor changed, if there is a student who has chosen this major.

**Attention:** You are expected to solve this question ***without*** using declarative referential integrity constraints in PostgreSQL!

**Your tasks are as follows:**

**a)** Implement the given referential integrity constraint using triggers and PL/pgSQL trigger procedures.

**Hint:** We recommend that you test your proposed implementation. To start with, create your own database and execute the command

```
\i ~/swen304_a3_q1.sql
```

which will populate your database by table definitions and test data.

**b)** Restore your database to its initial state (as specified by the file `swen304_a3_q1.sql`). Then validate your final solution from part **a)** by executing the command

```
\i ~/swen304_a3_q1_test.txt
```

to perform the final testing.

**Submit the following for the tasks above:**

**1)** The code of your procedures and triggers both in **printed form and electronically**.

**2)** The PostgreSQL output when performing the final testing (as specified by the file `swen304_a3_q1_test.txt`) in **printed form**.

## Question 2. User-Defined Functions [40 marks]

A university is using a relational database to manage its data on students, their majors, courses, and the students' results. Suppose the following relation schemas are part of the underlying database schema.

- MAJOR (mCode, name)
  with primary key {mCode}
- STUDENT (sId, firstName, lastName, mCode, pointsEarned)
  with primary key {sId} and foreign key STUDENT [mCode] $\subseteq$ MAJOR [mCode]
- COURSE (cId, title, points)
  with primary key {cId}
- RESULT (sId, cId, year, grade)
  with primary key {sId + cId + year} and foreign keys
  RESULT [sId] $\subseteq$ STUDENT [sId] and RESULT [cId] $\subseteq$ COURSE [cId]
- GRADUATE (sId, graduationDate)
  with primary key {sId} and foreign key GRADUATE [sId] $\subseteq$ STUDENT [sId]

The following additional requirements are known:

- A student can get more than one grade for the same course, but each in a different year.
- When a student passes a course, she/he will have the points for this course added to her/his earned points. A student may earn the points for the same course only once.
- When a student has earned 360 points or more, she/he is recorded in the GRADUATE table.

Whenever a student passes a course, the database is updated by issuing a sequence of `INSERT` and `UPDATE` commands: first, the student's grade is recorded in the RESULT table, then the points for this course are added to student's earned points in the STUDENT table, and finally the earned points are checked to see whether the student can be recorded in the GRADUATE table.

Unfortunately, many errors may happen when database updates are performed manually. After each examination period, academic administrators spend many hours to fix such errors.

**Your tasks are as follows:**

**a)** List three common errors that may happen when the database update described above is performed manually.

**b)** Implement a PL/pgSQL user-defined function that can perform the required actions automatically whenever a student passes a course,

i.e., your function should insert a new tuple into the RESULT table, update the STUDENT table (if needed), and insert the student into the GRADUATE table (if she/he has earned enough points).

Your user-defined function is expected to have the following interface:

```
coursePass (In_sId int, In_cId char, In_year int, In_grade
char, In_graduationDate date)
```

**Hint:** We recommend that you test your proposed implementation. To start with, create your own database and execute the command

```
\i ~/swen304_a3_q2.sql
```

which will populate your database by table definitions and test data.

**c)** Restore your database to its initial state (as specified by the file `swen304_a3_q2.sql`). Then validate your final solution from part **b)** by executing the command

```
\i ~/swen304_a3_q2_test.txt
```

to perform the final testing.

**Submit the following for the tasks above:**

**1)** Your list of possible errors in **printed form**.

**2)** The code of your user defined function `coursePass()` **both in printed form and electronically**.

**3)** The PostgreSQL output when performing the final testing (as specified by the file `swen304_a3_q2_test.txt`) in **printed form**.

## Question 3. Database Access [20 marks]

Recall the university database from Question 2. In reality there are many different users who need to access the database to record and/or manipulate data.

**Your tasks are as follows:**

**a)** Read the PostgreSQL manual and your textbook to find out:

- What is a database `ROLE`?
- What is `ROLE-BASED ACCESS CONTROL`?
- What kind of a role is `PUBLIC`?
- What is the `GRANT` clause used for?
- What is the `REVOKE` clause used for?

You find a link to the PostgreSQL manual on the SWEN304 web site.
**Attention:** Please ***do not*** print out the entire PostgreSQL manual.

**b)** What is the purpose of executing the following SQL commands before users are permitted to use your database?

```
GRANT CONNECT ON DATABASE <your_database_name> TO PUBLIC;
GRANT SELECT, INSERT, DELETE, UPDATE ON MAJOR, STUDENT,
COURSE, RESULT, GRADUATE TO PUBLIC;
```

**c)** What happens if the RESULT table is missing in the SQL command in part **b)**?

**d)** What happens if the UPDATE operation is missing in the SQL command in part **b)**?

**Submit the following in *print form* for the tasks above:**

**1)** An explanation of the listed concepts.

**2)** An explanation of the purpose of the SQL statements.

**3)** An explanation of all consequences.

**4)** An explanation of all consequences.

## Using PostgreSQL on the Workstations

We have a command line interface to PostgreSQL server, so you need to run it from a Unix prompt in a shell window. To enable the various applications required, first type either

**> need comp302tools**

or

**> need postgresql**

You may wish to add either "`need comp302tools`", or the "`need postgresql`" command to your `.cshrc` file so that it is run automatically. Add this command after the command `need SYSfirst`, which has to be the first `need` command in your `.cshrc` file.

There are several commands you can type at the unix prompt:

**> createdb** 〈database_name〉

Creates an empty database. The database is stored in the same PostgreSQL server used by all the students in the class. Your database may have an arbitrary name, but we recommend to name it either `userid` or `userid_x`, where `userid` is your ECS user name and `x` is a number from 0 to 9. To ensure security, you must issue the following command as soon as you log-in into your database for the first time:

```
REVOKE CONNECT ON DATABASE <database_name> FROM PUBLIC;
```

You only need to do this once (unless you get rid of your database to start again). **Note**, your markers may check whether you have issued this command and if they find you didn't, you may be **penalized**.

> **psql** [ **–d** 〈db name〉 ]

Starts an interactive SQL session with PostgreSQL to create, update, and query tables in the database. The db name is optional (unless you have multiple databases)

**> dropdb** 〈databas_name〉

Gets rid of a database. (In order to start again, you will need to create a database again)

**> pg_dump -i** 〈databas_name〉 > 〈file_name〉

Dumps your database into a file in a form consisting of a set of SQL commands that would reconstruct the database if you loaded that file.

> **psql –d** <database_name> **-f** <file_name>

Copies the file <file_name> into your database <database_name>.

Inside and interactive SQL session, you can type SQL commands. You can type the command on multiple lines (note how the prompt changes on a continuation line). End commands with a ';'

There are also many single line PostgreSQL commands starting with '\' . No ';' is required. The

most useful are

**\?** to list the commands,

**\i** 〈file_name〉
loads the commands from a file (eg, a file of your table definitions or the file of data we provide).

**\dt** to list your tables.

**\d**〈table_name〉 to describe a table.

**\q** to quit the interpreter

**copy** <table_name> **to** <file_name>
Copy your table_name data into the file file_name.

**copy** <table_name> **from** <file_name>
Copy data from the file file_name into your table table_name.

Note also that the PostgreSQL interpreter has some line editing facilities, including up and down arrow to repeat previous commands.
For longer commands, it is safer (and faster) to type your commands in an editor, then paste them into the interpreter!