SWEN 304
Database System Engineering
Assignment 3

David Thomsen, 300052209

Question 1. Triggers

Code:

```
CREATE OR REPLACE FUNCTION insert_update_student()
RETURNS trigger AS $$
DECLARE
  m record;
BEGIN
  SELECT * INTO m FROM major
  WHERE (mcode = NEW.mcode);
  IF m.mcode IS NULL THEN
    RAISE NOTICE 'major code % not found', NEW.mcode;
    RETURN NULL;
  ELSE
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE PLpgSQL;

CREATE TRIGGER insert_update_student
BEFORE INSERT OR UPDATE ON student
FOR EACH ROW EXECUTE PROCEDURE insert_update_student();

CREATE OR REPLACE FUNCTION delete_major()
RETURNS trigger AS $$
DECLARE
  s record;
BEGIN
  SELECT * INTO s FROM student
  WHERE (mcode = OLD.mcode);
  IF s.mcode IS NULL THEN
    RETURN OLD;
  ELSE
    RAISE NOTICE 'student is still enrolled in major %', OLD.mcode;
    RETURN NULL;
```

```
  END IF;
END;
$$ LANGUAGE PLpgSQL;

CREATE TRIGGER delete_major
BEFORE DELETE ON major
FOR EACH ROW EXECUTE PROCEDURE delete_major();

CREATE OR REPLACE FUNCTION update_major()
RETURNS trigger AS $$
DECLARE
  s record;
BEGIN
  SELECT * INTO s FROM student
  WHERE (mcode = OLD.mcode);
  IF s.mcode IS NULL THEN
    RETURN NEW;
  ELSE
    RAISE NOTICE 'student is still enrolled in major %', OLD.mcode;
    RETURN NULL;
  END IF;
END;
$$ LANGUAGE PLpgSQL;

CREATE TRIGGER update_major
BEFORE UPDATE ON major
FOR EACH ROW EXECUTE PROCEDURE update_major();
```

Assumption: While it makes sense to me that an update to the ‘mcode’ of a ‘major’ should cascade to the students, the assignment handout says ‘A major may not be deleted nor changed, if there is a student who has chosen this major’, so I have chosen to restrict this operation.
The ‘update major’ and ‘delete major’ functions are almost identical, but one has to ‘return old’ and the other has to ‘return new’, otherwise they wouldn’t work properly. Possibly there is a way to write ‘if operation is delete then do this, otherwise do that’, but I am unable to find this syntax if there is any.

Test Output:

```
(Q1.Test0) SELECT COUNT (*) FROM STUDENT; -- should give a number
 count
-------
     9
(1 row)

(Q1.Test1) INSERT INTO MAJOR VALUES(SE, Software Engineering); --
should succeed
```

```
INSERT 0 1
(Q1.Test2) INSERT INTO STUDENT VALUES(5001, Lisa, Simpson, SE,
180); -- should succeed
INSERT 0 1
(Q1.Test3) INSERT INTO STUDENT VALUES(5002, Bart, Simpson, Bio,
90); -- should fail
psql:/home/thomsedavi/Ass3/swen304_a3_q1_test:11: NOTICE:  major
code Bio    not found
INSERT 0 0
(Q1.Test4) UPDATE STUDENT SET pointsEarned = pointsEarned + 15
WHERE sId = 5001; -- should succeed
UPDATE 1
(Q1.Test4) SELECT pointsEarned FROM STUDENT WHERE sId = 5001; --
should return 195
 pointsearned
--------------
          195
(1 row)

(Q1.Test5) UPDATE STUDENT SET mCode = Bio WHERE sId = 5001; --
should fail
psql:/home/thomsedavi/Ass3/swen304_a3_q1_test:20: NOTICE:  major
code Bio    not found
UPDATE 0
(Q1.Test6) INSERT INTO MAJOR VALUES(OR, Operations Research); --
should succeed
INSERT 0 1
(Q1.Test6) DELETE FROM MAJOR WHERE mCode = OR; -- should succeed
DELETE 1
(Q1.Test6) SELECT * FROM MAJOR; -- there should be no OR
 mcode  |         name
--------+----------------------
 Stat   | Statistics
 CS     | Computer Science
 Math   | Mathematics
 SE     | Software Engineering
(4 rows)

(Q1.Test7) DELETE FROM MAJOR WHERE mCode = SE; -- should fail
psql:/home/thomsedavi/Ass3/swen304_a3_q1_test:32: NOTICE:  student
is still enrolled in major SE
DELETE 0
(Q1.Test8) INSERT INTO MAJOR VALUES(NE, Networking); -- should
succeed
INSERT 0 1
(Q1.Test8) UPDATE MAJOR SET name = Network Engineering WHERE mCode
= NE; -- should succeed
```

```
UPDATE 1
(Q1.Test8) SELECT name FROM MAJOR WHERE mCode = NE; -- should
return Network Engineering
        name
---------------------
 Network Engineering
(1 row)

(Q1.Test9) UPDATE MAJOR SET mCode = SoftE WHERE mCode = SE;
psql:/home/thomsedavi/Ass3/swen304_a3_q1_test:44: NOTICE:  student
is still enrolled in major SE
UPDATE 0
(Q1.Test9) SELECT mCode FROM STUDENT WHERE sId = 5001;
 mcode
--------
 SE
(1 row)
```

Question 2. User-Defined Functions

Three common errors that may happen when the database update is performed manually:

- A student might be entered into the ‘graduate’ table twice. This would happen if the ‘before’ and ‘after’ number of points earned are not checked. For example, if a student increased their points earned from 365 to 380, the database might only check that their new total is greater than or equal to 360 and insert them into the ‘graduate’ table again, even though they should already be there.
- A student might not have their points increased if they had received a grade for a course previously, but had not previously had a passing grade and now they do. Also, they might have their points total increased twice for the same paper if they had passed the same course previously. In both cases the database is not checking what grades the student had received in previous years.
- The database could refuse to accept a new grade if a student had taken the same course previously. In this case it is not checking whether the years are years of the two grades are different. Also, if it is not checking the years of the grades, then it might enter the same grade twice for the same year.

Code:

```
CREATE OR REPLACE FUNCTION coursePass (In_sId int, In_cId char,
In_year int,
  In_grade char, In_graduationDate date)
RETURNS char AS $$
DECLARE
  s record;
```

```
  c record;
  r record;
BEGIN
  SELECT * INTO s FROM student
  WHERE (sid = In_sId);
  IF NOT FOUND THEN
    RAISE EXCEPTION 'student % not found', In_sId;
  END IF;
  SELECT * INTO c FROM course
  WHERE (cid = In_cId);
  IF NOT FOUND THEN
    RAISE EXCEPTION 'course % not found', In_cId;
  END IF;
  SELECT * INTO r FROM result
  WHERE (sid = In_sId AND cid = In_cId AND year = In_year);
  IF FOUND THEN
    RAISE EXCEPTION 'result %, %, % found', In_sId, In_cId,
In_year;
  END IF;
  SELECT * INTO r FROM result
  WHERE (sid = In_sId AND cid = In_cId AND grade != 'D');
  IF FOUND THEN
    INSERT INTO result VALUES(In_sId, In_cId, In_year, In_grade);
  ELSE
    INSERT INTO result VALUES(In_sId, In_cId, In_year, In_grade);
    IF (In_grade != 'D') THEN
      UPDATE student SET pointsearned = pointsearned + c.points
      WHERE sid = In_sId;
      IF (s.pointsearned < 360 AND s.pointsearned + c.points >=
360) THEN
        INSERT INTO graduate VALUES (In_sId, In_graduationDate);
        RETURN 'INSERT 2';
      END IF;
    END IF;
  END IF;
  RETURN 'INSERT 1';
END;
$$ LANGUAGE PLpgSQL;
```

Test Output:

```
(Q2.Test0) SELECT COUNT (*) FROM RESULT; -- should give a number
 count
-------
    50
(1 row)
```

```
(Q2.Test1) SELECT coursePass(5000, COMP205, 2016, A-, 2016-07-01);
-- wrong student
psql:/home/thomsedavi/Ass3/swen304_a3_q2_test:5: ERROR:  student
5000 not found
(Q2.Test2) SELECT coursePass(5003, SWEN205, 2016, A-, 2016-07-01);
-- wrong course
psql:/home/thomsedavi/Ass3/swen304_a3_q2_test:8: ERROR:  course
SWEN205 not found
(Q2.Test3) SELECT coursePass(5003, COMP202, 2016, B+, 2016-07-01);
-- duplicate
psql:/home/thomsedavi/Ass3/swen304_a3_q2_test:11: ERROR:  result
5003, COMP202, 2016 found
(Q2.Test4) SELECT coursePass(5003, COMP103, 2016, B-, 2016-07-01);
-- now pass grade, was fail
 coursepass
------------
 INSERT 1
(1 row)

(Q2.Test4) SELECT pointsEarned FROM STUDENT WHERE sId = 5003; --
should return 150
 pointsearned
--------------
          150
(1 row)

(Q2.Test5) SELECT coursePass(5003, ENGR101, 2016, A, 2016-07-01);
-- now pass grade, was also pass
 coursepass
------------
 INSERT 1
(1 row)

(Q2.Test5) SELECT pointsEarned FROM STUDENT WHERE sId = 5003; --
should still return 150
 pointsearned
--------------
          150
(1 row)

(Q2.Test6) SELECT coursePass(5003, MATH114, 2016, D, 2016-07-01);
-- now fail grade, was pass
 coursepass
------------
 INSERT 1
(1 row)
```

```
(Q2.Test6) SELECT pointsEarned FROM STUDENT WHERE sId = 5003; --
should still return 150
 pointsearned
--------------
          150
(1 row)

(Q2.Test7) SELECT coursePass(6006, COMP311, 2016, A+, 2016-07-01);
 coursepass
------------
 INSERT 2
(1 row)

(Q2.Test7) SELECT pointsEarned FROM STUDENT WHERE sId = 6006;
 pointsearned
--------------
          375
(1 row)

(Q2.Test8) SELECT * FROM GRADUATE;
 sid  | graduationdate
------+----------------
 7007 | 2015-11-30
 6006 | 2016-07-01
(2 rows)
```

Question 3. Database Access

a)

i) A database ROLE is a user, or group of users, who own database objects and can assign privileges on those objects to other roles. A user must have ‘CREATEROLE’ privilege or be a superuser to create roles. Roles are useful for setting up new users quickly with a suitable set of predefined privileges.

ii) ROLE-BASED ACCESS CONTROL is a mechanism that allows roles to limit how much other roles can access their data. It can limit which tables they have access to, and which operations they are able to perform on these tables, for example they might be able to create new rows, but not update or delete them.

iii) The GRANT clause is used to either allow access to database objects, or grant membership into a role. For example, it can allow a user in one role to have access into another role, which will give them the privileges of that role.

iv) A PUBLIC ROLE is one where the privileges are accessible to all other roles. There are limits, for example a public role does not have the privilege of ‘grant’.

v) The REVOKE clause is used to remove privileges from one or more roles, or from the ‘public’ role.

b) The ‘public’ role will be able to access the database and run the *select*, *insert*, *delete* and *update* operations on the tables *major*, *student*, *course*, *result*, *graduate*.
c) The ‘public’ role will not be able to run the *select*, *insert*, *delete* and *update* operations on the *result* table. Any role that is created subsequently will not automatically be able to run these operations.
d) The ‘public’ role will not be able to run the *update* operation on the *major*, *student*, *course*, *result* and *graduate* tables. Any role that is created subsequently will not automatically be able to run these operations.