TÉCNICO LISBOA

Mestrado em Engenharia Eletrotécnica e de Computadores

Sistemas de Informação e Bases de Dados

Project (Part2) Report

Group 21

Beatriz Marques 80809

José Coelho 81013

Manuel Reis 81074

November 15th, 2017

Expected Results

1. Create database tables

The file `create.sql` (attached to this report) creates the required tables on the database server. It also drops any pre-existing tables with the same names as the ones being created. During the creation of the tables, several decisions were made regarding the type of attributes (columns of each table). An example of a type that could differ is *patient_id* (field in the Patient table), which was declared as an INTEGER, but there are also other valid types, such as a VARCHAR(), since there isn’t any mathematical operation involving these fields.

The code used to create each table is very similar throughout the file. The following code is used to create the Sensor table. All the remaining code can be found in the file attached to the report.

```
CREATE TABLE Sensor (
 snum  INTEGER,
 manuf VARCHAR(255),
 units VARCHAR(255),
 PRIMARY KEY (snum, manuf, units),
 FOREIGN KEY (snum, manuf) REFERENCES Device (serialnum, manufacturer)
);
```

2. Populate tables

The file `populate.sql` which is attached to this report, contains a set of data to populate the database. Several meaningful entries were added to validate the results of the next questions and ensure that they were working.

To guarantee that a table is populated correctly and with no errors, for each entry, its foreign keys must already exist in their respective table. For instance, it is not possible to associate a device to a patient for a certain period of time if that period and patient have not yet been added to the Period and Patient tables.

It's assumed (as a simplification) the integrity of the data inserted, eg in Period, “start” must never exceed “end”.

Below is partially presented the content of the file used. The entire content, which follows the same structure, can be found on the `.sql` file attached.

```
INSERT INTO Patient VALUES (11,'Absalão Gonçalves','2009-6-20','Viseu');
...
INSERT INTO Doctor VALUES (11, 1);
...
INSERT INTO Device VALUES (80558661, 'Johnson & Johnson', 'IYTQUGJCXK');
...
INSERT INTO Sensor VALUES (80558661,'Medtronic',’Blood Pressure(Diastolic)
in mmHg');
...
```

```
INSERT  INTO  Reading  VALUES  (26266100,  'Johnson  &  Johnson',  '2017-08-1
8:14:2', 246.46);
...
INSERT INTO Period VALUES ('2016-11-24 11:22:38', '2016-12-11 06:23:21');
...
INSERT INTO Wears VALUES ('2016-11-24 11:22:38', '2016-12-11 06:23:21', 9,
80558661, 'Medtronic');
...
INSERT INTO Request VALUES (1, 4, 5, '2015-3-22');
...
INSERT  INTO  Study  VALUES  (1,  'HRZFAKYSJA',  '2015-4-3',  4,  'Johnson  &
Johnson', 80558661);
...
INSERT INTO Series VALUES (1, 'Series#1', 'http://series.health.com/1', 1,
'HRZFAKYSJA');
...
INSERT INTO Element VALUES (1, 1);
...
INSERT INTO Region VALUES (1, 1, 0.05, 0.19, 0.65, 0.48);
...
```

3. Query: Highest number of readings

The file `query1.sql` contains a query to retrieve the name(s) of the patient(s) with the highest number of readings of units of “LDL cholesterol in mg/dL” above 200 in the past 90 days. That query reads as follows:

```
SELECT name
FROM Patient
WHERE number IN (
 SELECT patient
 FROM Sensor AS s1, Reading AS r1, Wears AS w1
 WHERE r1.snum = s1.snum
       AND r1.manuf = s1.manuf
       AND r1.snum = w1.snum
       AND r1.manuf = w1.manuf
       AND r1.datetime BETWEEN w1.start AND w1.end
       AND units = 'LDL cholesterol in mg/dL'
       AND DATEDIFF(current_date, r1.datetime) < 90
       AND r1.value > 200
 GROUP BY patient
 HAVING COUNT(patient) >= ALL (
   SELECT COUNT(patient)
   FROM Sensor AS s, Reading AS r, Wears AS w
   WHERE r.snum = s.snum
         AND r.manuf = s.manuf
         AND r.snum = w.snum
         AND r.manuf = w.manuf
         AND r.datetime BETWEEN w.start AND w.end
         AND units = 'LDL cholesterol in mg/dL'
         AND DATEDIFF(current_date, r.datetime) < 90
         AND r.value > 200
   GROUP BY patient));
```

If the database is populated according to the contents of `populate.sql`, the result of the given query is

```
+-------------------+
| name              |
+-------------------+
| Angelini Picanço  |
+-------------------+
```

which is correct (the patient has 4 readings that match the criteria).

Other successful tests were conducted where the populate file was changed to include more than one patient as the query result. This can be achieved by removing/commenting the line 157 on the file `populate.sql` which produces the following result

```
+-------------------+
| name              |
+-------------------+
| Angelini Picanço  |
| Nectarie Sales    |
| Yossef Antas      |
+-------------------+
```

In the query, the period during which a device is assigned to a patient must be taken into consideration. Otherwise, if a device has been associated with more than one patient in the past 90 days, the number of readings will be the same for all those patients and the query might return false results.

4. Query: Patients subject to studies

The file `query2.sql` contains a query to retrieve the name(s) of the patient(s) who have been subject of studies with all devices of manufacturer “Medtronic” in the past calendar year. The requested query reads as follows:

```
SELECT name
FROM Patient AS p
WHERE NOT EXISTS(
   SELECT serialnum
   FROM Device AS d
   WHERE manufacturer = 'Medtronic'
         AND serialnum NOT IN (
     SELECT serial_number
     FROM Study AS s, Request AS r, Patient AS p2
     WHERE s.request_number = r.number
           AND p2.number = r.patient_id
           AND p2.name = p.name
           AND YEAR(s.date) = YEAR(CURRENT_DATE()) - 1));
```

A nested query and double negation were used to obtain the results that fit the criteria. The logic behind the query was based on rewriting the objective of the query. The following shows how that rewriting was made:

“Patients who have been subject of studies with all devices of the manufacturer "Medtronic" in the past calendar year.”

which gets rewritten to…

“Patients for which there is no device of the manufacturer "Medtronic" which was not used to perform a study in the past calendar year.”

which gets rewritten to…

“Who are the patients for which there is no device of the manufacturer "Medtronic" which is not in the set of devices used to do a study in the past calendar year.”

If the database is populated according to the contents of `populate.sql`, the result of the given query is

```
+----------------+
| name           |
+----------------+
| Ramiro Amado   |
| Alesia Pacheco |
+----------------+
```

5. i. Trigger: The doctor who prescribed the exam cannot perform the study

The file `trigger_i.sql` contains a set of triggers to prevent any entry to the Study table that results in a doctor performing a Study on a Request that he himself prescribed.

There are 2 situations that must be considered: when inserting a new row into the table or when updating an already existing row. As such, the trigger is composed by two parts, one regarding the insertion of data and another regarding the update of already existing data.

The trigger calls a procedure that prints an error with a message passed as an argument. The declaration of this procedure is in the file `extra_proc_error.sql` and can be used many times, for different errors with custom error messages.

The triggers are implemented as follows:

```
DELIMITER $$
CREATE TRIGGER doctor_check_create
BEFORE INSERT ON Study
FOR EACH ROW
 BEGIN
   IF (SELECT EXISTS(SELECT *
                     FROM Request, Study
                     WHERE Request.number = Study.request_number
                           AND new.doctor_id = Request.doctor_id
                          AND new.request_number = Study.request_number))
   THEN
     CALL ERROR('Error: The same doctor who prescribed the exam is
performing a study.');
   END IF;
 END$$

DELIMITER ;


DELIMITER $$
CREATE TRIGGER doctor_check_update
BEFORE UPDATE ON Study
FOR EACH ROW
 BEGIN
   IF (SELECT EXISTS(SELECT *
                     FROM Request, Study
                     WHERE Request.number = Study.request_number
                           AND NEW.doctor_id = Request.doctor_id
                           AND NEW.request_number = Study.request_number))
   THEN
     CALL ERROR('Error: The same doctor who prescribed the exam is
performing a study.');
   END IF;
 END$$

DELIMITER ;
```

To test if the triggers are well implemented, the file `test_trigger_i.sql` was created. This file contains several new entries and updates to the table Study, some of which are valid and others aren’t, which means the error message will be displayed.

The output visible on the mysql prompt after running the command:

```
mysql> source test_trigger_i.sql
```

is presented below and matches the expected results.

```
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.01 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
ERROR 1644 (45000) at line 10 in file: 'test_trigger_i.sql': Error: The same
doctor who prescribed the exam is performing a study.
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.01 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
ERROR 1644 (45000) at line 17 in file: 'test_trigger_i.sql': Error: The same
doctor who prescribed the exam is performing a study.
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
ERROR 1644 (45000) at line 26 in file: 'test_trigger_i.sql': Error: The same
doctor who prescribed the exam is performing a study.
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
ERROR 1644 (45000) at line 34 in file: 'test_trigger_i.sql': Error: The same
doctor who prescribed the exam is performing a study.
```

5. ii. Trigger: A Device can't be associated with more than one Patient at the same time

This trigger prevents someone from trying to associate a device to a patient in overlapping periods. As before, there are 2 situations that must be considered: when inserting a new row into the table or when updating an already existing row.

The file `trigger_ii.sql` contains a set of triggers to prevent any entry to the Wears table that results in a device being used in overlapping periods, which is impossible. This trigger also calls the procedure that allows the print of custom error messages, which is present in the file `extra_proc_error.sql`. Both triggers are presented below.

```
DELIMITER $$
CREATE TRIGGER check_overlaps_create
BEFORE INSERT ON Wears
FOR EACH ROW
  BEGIN
    IF (SELECT EXISTS(SELECT *
                      FROM Wears AS w
                      WHERE w.manuf = NEW.manuf
                      AND w.snum = NEW.snum
                      AND NOT (NEW.start < w.start AND NEW.end < w.start
                          OR NEW.start > w.end AND NEW.end > w.end)))
    THEN
      CALL error('Overlapping Periods');
    END IF;

  END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER check_overlaps_update
BEFORE UPDATE ON Wears
FOR EACH ROW
  BEGIN
    IF (SELECT EXISTS(SELECT *
                      FROM Wears AS w
                      WHERE w.snum = NEW.snum
                      AND w.manuf = NEW.manuf
                      AND w.start != OLD.start
                      AND w.end != OLD.end
                      AND NOT (NEW.start < w.start AND NEW.end < w.start
                          OR NEW.start > w.end AND NEW.end > w.end)))
    THEN
      CALL error('Overlapping Periods');
    END IF;

  END$$
DELIMITER ;
```

To test if the triggers are working as expected, the file `test_trigger_ii.sql` was created. This file contains several new entries and updates to the table Wears (table Period was also updated as its values serve as foreign keys to Wears), some of which are valid and others aren't, which means the error message will be displayed.

The output is visible on the mysql prompt after running the command:

```
mysql> source test_trigger_ii.sql
```

Due to the amount of tests needed to completely verify the triggers, the ouput is rather extensive and was therefore omitted, however it matches on all situations to what was predicted.

6. Function: region_overlaps_element()

This function receives as arguments (series_id, index) of an Element A, and the coordinates (x1, y1, x2, y2) of a Region B, and returns true if any region of the element A overlaps with Region B, and false otherwise.
The file `func_region_overlaps.sql` contains the implementation of this function, which is below.
It is assumed that the region coordinates of the same variable can be in any order, i.e. x1 > x2.

```
DELIMITER $$

CREATE FUNCTION region_overlaps_element(series_id INTEGER, elem_index
INTEGER, x1 FLOAT, y1 FLOAT, x2 FLOAT, y2 FLOAT)
  RETURNS BOOLEAN
  BEGIN
    DECLARE result BOOLEAN;
    DECLARE Xmin FLOAT;
    DECLARE Xmax FLOAT;
    DECLARE Ymin FLOAT;
    DECLARE Ymax FLOAT;

    IF x1 > x2
    THEN
      SET Xmin = x2;
      SET Xmax = x1;
    ELSE
      SET Xmin = x1;
      SET Xmax = x2;
    END IF;

    IF y1 > y2
    THEN
      SET Ymin = y2;
      SET Ymax = y1;
    ELSE
      SET Ymin = y1;
      SET Ymax = y2;
    END IF;

    IF EXISTS(SELECT *
              FROM Region AS r
              WHERE series_id = r.series_id
                    AND elem_index = r.elem_index
                    AND NOT ((r.x1 < Xmin AND r.x2 < Xmin)
                             OR (r.x1 > Xmax AND r.x2 > Xmax)
                             OR (r.y1 < Ymin AND r.y2 < Ymin)
                             OR (r.y1 > Ymax AND r.y2 > Ymax)))

    THEN
      SET result = TRUE;
    ELSE
      SET result = FALSE;
    END IF;
    RETURN result;
  END $$

DELIMITER ;
```

To test if the function is working as expected, the file `test_func_region_overlaps.sql` was created. This file contains several new entries to Region and then the function is called multiple times to check if the correct output is given.

Due to the amount of tests needed to completely verify the function, the ouput is rather extensive and was therefore omitted, however it matches on all situations to what was predicted.

Attachments

The following list contains all files that are included and needed to fully setup and test the project. As said before, the procedure *Error* must be defined before the creation of the triggers.

`extra_proc_error.sql` – Stores the error procedure **needed** to the triggers.

`create.sql` - Relates to question 1

`populate.sql` - Relates to question 2

`query1.sql` - Relates to question 3

`query2.sql` - Relates to question 4

`trigger_i.sql, test_trigger_i.sql` – Relates to question 5i

`trigger_ii.sql, test_trigger_ii.sql` – Relates to question 5ii

`func_region_overlaps.sql, test_func_region_overlaps.sql` – Relates to question 6