CS3120 Database Management Systems Laboratory

Assignment-7

Mayank Singla
111901030

Using VSCode to write the queries in a file and then execute that file from the terminal using "`\i file.sql`"

Q1. Create a student table with three columns (roll_no, name, grade). Insert the following three records into the table.

- (1, 'X', 'A')
- (2, 'Y', 'B')
- (3, 'Z', 'C')

Now, create a trigger that will not allow entering any record into the student table with a grade that is not used before in any record in the student table. Use the following records to test the functionality of the trigger.

- (4, 'M', 'B') should be inserted into the table
- (5, 'N', 'D') should give error

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

Run on active connection | Select block

```sql
/
*****************************************************
***************/
/* Q1 */

Run SQL
CREATE TABLE student (
    roll_no SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    grade CHAR(1) NOT NULL
);

Run SQL
INSERT INTO student
VALUES (1, 'X', 'A'),
    (2, 'Y', 'B'),
    (3, 'Z', 'C');
```

```
mayank=# \i solution.sql
CREATE TABLE
INSERT 0 3
mayank=# SELECT * FROM student;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
(3 rows)
```

solution.sql U

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE FUNCTION isValidGrade()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    IF (NEW.grade IN (SELECT grade FROM student)) THEN
        -- If new grade is already present then allowing the insertion
        RETURN NEW;
    ELSE
        -- Returning null won't allow the insertion
        RAISE EXCEPTION 'Unkown Grade Value Provided!!!';
        RETURN NULL;
    END IF;
END;
$$;

CREATE OR REPLACE TRIGGER validate_grade
BEFORE INSERT
ON student
FOR EACH ROW
EXECUTE FUNCTION isValidGrade();
```

```
mayank=# \i solution.sql
CREATE FUNCTION
CREATE TRIGGER
mayank=#
```

```
mayank=# SELECT * FROM student;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
(3 rows)

mayank=# INSERT INTO student VALUES (4, 'M', 'B');
INSERT 0 1
mayank=# SELECT * FROM student;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
       4 | M    | B
(4 rows)

mayank=#
```

```
mayank=# SELECT * FROM student;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
       4 | M    | B
(4 rows)

mayank=# INSERT INTO student VALUES (5, 'N', 'D');
ERROR:  Unkown Grade Value Provided!!!
CONTEXT:  PL/pgSQL function isvalidgrade() line 8 at RAISE
mayank=# SELECT * FROM student;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
       4 | M    | B
(4 rows)

mayank=#
```

Q2. Create a connection table with two columns (name1, name2) that show the relationship between two individuals. Write triggers to maintain the relationship in the table. I.e., if (x,y) is deleted/added from/to the table then (y,x) should also be deleted/added respectively.

To show the functionality of triggers:

- Insert ('A', 'B') and ('C', 'D') into the connection table, and ('B', 'A') and ('D', 'C') should automatically be inserted into the table.
- Delete ('D', 'C') from the connection table, and ('C', 'D') should automatically be deleted from the table.

solution.sql U

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
/*****************************************************************************
**************/
/* Q2 */
▶ Run SQL
CREATE TABLE connection (
    name1 VARCHAR(50) NOT NULL,
    name2 VARCHAR(50) NOT NULL,
    PRIMARY KEY (name1, name2)
);
```

```
mayank=# \i solution.sql
CREATE TABLE
mayank=#
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE FUNCTION insert_connection()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN

    IF (NOT EXISTS (
                SELECT *
                FROM connection
                WHERE name1 = NEW.name2
                    AND name2 = NEW.name1
                )
    ) THEN
        INSERT INTO connection(name1, name2)
        VALUES (NEW.name2, NEW.name1);
    ▶ Run SQL
    END IF;

    ▶ Run SQL
    RETURN NEW;
▶ Run SQL
END;
▶ Run SQL
$$;
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

▶ Run SQL

```sql
CREATE OR REPLACE TRIGGER connection_insert
AFTER INSERT
ON connection
FOR EACH ROW
EXECUTE FUNCTION insert_connection();
```

```
mayank=# \i solution.sql
CREATE FUNCTION
CREATE TRIGGER
mayank=#
```

```
mayank=# SELECT * FROM connection;
 name1 | name2
-------+-------
(0 rows)

mayank=# INSERT INTO connection(name1, name2) VALUES ('A', 'B'), ('C', 'D');
INSERT 0 2
mayank=# SELECT * FROM connection;
 name1 | name2
-------+-------
 A     | B
 C     | D
 B     | A
 D     | C
(4 rows)

mayank=#
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE FUNCTION delete_connection()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    IF (EXISTS (
            SELECT *
            FROM connection
            WHERE name1 = OLD.name2
                AND name2 = OLD.name1
            )
    ) THEN
        DELETE FROM connection
        WHERE name1 = OLD.name2
            AND name2 = OLD.name1;
    ▶ Run SQL
    END IF;

    ▶ Run SQL
    RETURN OLD;
▶ Run SQL
END;
▶ Run SQL
$$;
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE TRIGGER connection_delete
AFTER DELETE
ON connection
FOR EACH ROW
EXECUTE FUNCTION delete_connection();
```

```
mayank=# \i solution.sql
CREATE FUNCTION
CREATE TRIGGER
mayank=#
```

```
mayank=# SELECT * FROM connection;
 name1 | name2
-------+-------
 A     | B
 C     | D
 B     | A
 D     | C
(4 rows)

mayank=# DELETE FROM connection WHERE name1 = 'D' AND name2 = 'C';
DELETE 1
mayank=# SELECT * FROM connection;
 name1 | name2
-------+-------
 A     | B
 B     | A
(2 rows)

mayank=#
```

Q3. Create a new table similar to the student table in question 1 and name it “grades” and use the student_log table from the demo replacing the date column with time). Also, use the student_logs function (change date to time) and log_trigger trigger from the demo. Insert the following records into the table. (1.5 marks)

- (1, ‘X’, ‘A’)
- (2, ‘Y’, ‘B’)
- (3, ‘Z’, ‘C’)

On insertion, the log_trigger should be invoked and log the values into the student_log table.

After inserting the records, create a trigger such that when you update the grade of a student in the grade table, the time in the student_log table should be updated to the current time. To show the functionality of the trigger, update the grade of student ‘Y’ to ‘A’.

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```
/
**********************************************
**************/
/* Q3 */

▶ Run SQL
CREATE TABLE grades (
    roll_no SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    grade CHAR(1) NOT NULL
);

▶ Run SQL
CREATE TABLE student_log (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    time TIME NOT NULL
);
```

```
mayank=# \i solution.sql
CREATE TABLE
CREATE TABLE
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE FUNCTION student_logs()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    INSERT INTO student_log
    VALUES (NEW.roll_no, NEW.name, CURRENT_TIME);
    ▶ Run SQL
    RETURN NEW;
▶ Run SQL
END;
▶ Run SQL
$$;

▶ Run SQL
CREATE OR REPLACE TRIGGER log_trigger
AFTER INSERT
ON grades
FOR EACH ROW
EXECUTE FUNCTION student_logs();

▶ Run SQL
INSERT INTO grades
VALUES (1, 'X', 'A'),
    (2, 'Y', 'B'),
    (3, 'Z', 'C');
```

```
mayank=# \i solution.sql
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 3
mayank=# SELECT * FROM grades;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
(3 rows)

mayank=# SELECT * FROM student_log;
 roll_no | name |      time
---------+------+----------------
       1 | X    | 15:22:45.90895
       2 | Y    | 15:22:45.90895
       3 | Z    | 15:22:45.90895
(3 rows)

mayank=#
```

solution.sql U ×

dbms-lab > labwork > assignment_7 > solution.sql > ...

```sql
CREATE OR REPLACE FUNCTION student_logs_update()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    UPDATE student_log
    SET time = CURRENT_TIME
    WHERE roll_no = NEW.roll_no;

    ▶ Run SQL
    RETURN NEW;
▶ Run SQL
END;
▶ Run SQL
$$;

▶ Run SQL
CREATE OR REPLACE TRIGGER log_trigger_update
AFTER UPDATE
ON grades
FOR EACH ROW
EXECUTE FUNCTION student_logs_update();
```

```
mayank=# \i solution.sql
CREATE FUNCTION
CREATE TRIGGER
mayank=# SELECT * FROM student_log;
 roll_no | name |      time
---------+------+----------------
       1 | X    | 15:22:45.90895
       2 | Y    | 15:22:45.90895
       3 | Z    | 15:22:45.90895
(3 rows)

mayank=# SELECT * FROM grades;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       2 | Y    | B
       3 | Z    | C
(3 rows)

mayank=# UPDATE grades SET grade = 'A' WHERE roll_no = 2;
UPDATE 1
```

```
mayank=# SELECT * FROM grades;
 roll_no | name | grade
---------+------+-------
       1 | X    | A
       3 | Z    | C
       2 | Y    | A
(3 rows)

mayank=# SELECT * FROM student_log;
 roll_no | name |      time
---------+------+-----------------
       1 | X    | 15:22:45.90895
       3 | Z    | 15:22:45.90895
       2 | Y    | 15:31:41.604081
(3 rows)

mayank=#
```