Advanced Management of Data

Exercise 5 Topic 2:

Extensions of SQL

Triggers

- Objective: there are two tables for numbers and when something is inserted into the first table, it is added to the second table as well
- create two new tables for `INTEGER`s named `numbers` and `numbers_log`

```
CREATE TABLE IF NOT EXISTS numbers (number INTEGER NOT NULL);
CREATE TABLE IF NOT EXISTS numbers_log (number INTEGER NOT NULL);
```

- Task: write a new trigger function that is copying the value inserted into the table `numbers` to the table `numbers_log`

Triggers
First trigger function

```
CREATE OR REPLACE FUNCTION numbers_copy() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO numbers_log VALUES (NEW.number);
    RETURN NULL; -- result is ignored since this will be an AFTER trigger
  END;
$$ LANGUAGE plpgsql;
```

- Task: add a trigger to the table `numbers` that fires this trigger function after a new row was added

Triggers
First trigger

```
CREATE TRIGGER numbers_insert AFTER INSERT ON numbers
  FOR EACH ROW
    EXECUTE PROCEDURE numbers_copy();
```

- insert a value into the table `numbers`

```
INSERT INTO numbers VALUES (1);
```

- and check if it is really there by using your GUI or queries like

```
SELECT * FROM numbers;
SELECT * FROM numbers_log;
```

Triggers
Playing around

- insert some more values into the table `numbers`

```
INSERT INTO numbers VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9);
```

- and check again if they are there
- now delete some values from `numbers`

```
DELETE FROM numbers WHERE number < 5;
```

- and see the difference between the two tables
- Task: now create another trigger function and corresponding trigger, that fires before a new row is added to the table `numbers` and prevents the insertion when there are already 50 rows

Triggers
Fire or not

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

- Task: write one more trigger function and trigger, that fires right after a new row is added and inserts another number to `numbers`, that is one bigger than the last one

Triggers
More, more numbers

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

- insert another number to numbers and hope, you did nothing wrong (otherwise you just fired a trigger, that is triggering itself in an endless loop)

Triggers
Run amok?

- your second trigger should prevent your third trigger to insert more than 50 rows
- truncate your table to be able to insert and test more

```
TRUNCATE TABLE numbers;
```

- try something like

```
INSERT INTO numbers VALUES (1), (100);
```

- to see, that the execution of the function for the first value just fires a new trigger and so on, so that the function is never executed for the second value
- you can also see the sequence of insertion in the second table `numbers_log`

Triggers
Manipulation

- until now, we only slightly changed the value to be inserted, by deciding whether we let it in or not, but it is also possible to change it completely
- Task: create another trigger function and corresponding trigger, that fires before a new row is added to the table `numbers` and manipulate the input data, by just doubling its value

Triggers
Manipulation

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

Triggers
Recursion

- unfortunately 2^{50} is out of `INTEGER` range that only goes up to 2^{32} and the DBMS stops the execution with an error
- to test whether your function is working, you can just drop the self firing trigger

```
    DROP TRIGGER IF EXISTS numbers_insert_after ON numbers;
```

- even if you insert a “1”, our doubling function makes this to “2” and than our self firing trigger tries to insert a “3” that is doubled to “6” and our self firing trigger inserts a “7” that is doubled to “14” and so on
- the main problem is, that the increment trigger is firing itself and we run into this recursive trigger
- Task: re-enable the trigger, but rewrite it to avoid recursion

Triggers
No more recursion

```
SAMPLE CODE GOT PROVIDED DURING THE EXERCISE
```

- now, for each number, there is added just one new number

Triggers
More logging

- until now, we have a table with numbers and a table with all numbers, that were inserted to this table, but we don't know, which operations were performed
- create a new table named `numbers_log_query`

```
CREATE TABLE IF NOT EXISTS numbers_log_query (query TEXT NOT NULL);
```

- Task: write a new trigger function and corresponding trigger, that is logging the query, which was performed on the table `numbers`, to the table `numbers_log_query`

Triggers
More logging

```
SAMPLE CODE GOT PROVIDED DURING THE EXERCISE
```

- catch all four different events, that might fire a trigger, after their action was performed, so we ignore erroneous queries, that didn’t change any data

```
SAMPLE CODE GOT PROVIDED DURING THE EXERCISE
```

Triggers
Recursion again

- by playing around with the table `numbers` you can see, that `current_query()` is not updated for the queries fired by your other triggers and therefore you get a copy of the query for each value you insert, as this triggers another `INSERT` in `numbers_more()`
- for example

```
INSERT INTO numbers VALUES (1), (100);
```

- will be logged three times, as it is logged once for the main query and two times more, as there are inserted another two values
- obviously this is another recursive trigger, that we could easily avoid
- Task: rewrite the trigger to avoid recursion

Triggers
No more recursion again

- don’t forget to drop the trigger first, as in contrast to Oracle’s PL/SQL there is no such thing as `CREATE OR REPLACE TRIGGER`

```
DROP TRIGGER IF EXISTS numbers_alter ON numbers;
```

- and then simply add one line like before

```
SAMPLE CODE GOT PROVIDED DURING THE EXERCISE
```