Advanced Management of Data
Exercise 6 Topic 2:
Extensions of SQL

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
Recursion

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

- 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 a **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
```

Triggers
Academic example

- now, we have two tables for logging, what's going on with numbers, but one could easily mess up with them and `UPDATE`, `DELETE` or `TRUNCATE` data from them, so we don't know what happened
- Task: use triggers to prevent this, so that one can only insert something to `numbers_log` and `numbers_log_query`
- disclaimer: of course this won't be an elegant solution but merely a workaround, as one wouldn't grant the database user any privileges, that it shouldn't have and simply revoke those privileges, but consider this just another academic example like the rest of this exercise

Triggers
Exceptional

- row-level trigger returning `NULL` prevent further execution, but triggers on `TRUNCATE` are only allowed as statement-level trigger and those should always return `NULL` and don't prevent the execution
- therefore we have to raise an exception in before to prevent further execution

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

Triggers
Distinct view

- inserting the same values again and again to our `numbers` table to test some triggers might get boring, so let's change our view by creating a new `VIEW` that only shows distinct values and does some sorting

```
CREATE OR REPLACE VIEW numbers_distinct AS SELECT DISTINCT number FROM numbers
ORDER BY number;
```

- unfortunately, this view isn't automatically updatable as it contains a `DISTINCT` clause at the top level, so the system doesn't allow `INSERT`, `UPDATE` and `DELETE` statements on the `VIEW`, as it can't translate them to corresponding statements on the base relation
- therefore we have to solve this
- Task: use triggers to enable the usage of `INSERT`, `UPDATE` and `DELETE` on `numbers_distinct`

Triggers
Working View

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

- by using `TG_OP` to get the type of operation that should be performed, we can do everything with just one trigger function, but it is also possible to use one function for each case
- of course we would need to write more code with different trigger functions and we would have to write even more code, as we would need different triggers and now we can define just one trigger for all cases

SAMPLE CODE GOT PROVIDED DURING THE EXERCISE

- now you can try to change `numbers_distinct` and see how `numbers` is modified