Stored Procedures and Triggers

Content

1. Trigger Function Format
2. Creating a Trigger Function
3. Testing the Trigger Function

Trigger

- A *trigger* defines an operation that is performed when a specific event occurs on a table:
  - inserts a new record / updates an existing record, or deletes a record.
- The function executed as a result of a trigger is called a *trigger function*.

1. Trigger Function Format

- Looks similar to the stored procedure function (same CREATE OR REPLACE FUNCTION command)
- 2 two things:
  - Trigger functions do not use input arguments in the function, but rather are passed arguments from a trigger event
  - Trigger functions have access to special variables from the database engine

CREATE TRIGGER command

```
CREATE TRIGGER name
{ BEFORE | AFTER | INSTEAD OF} {event [OR ... ] } ON table/view
[ FOR [ EACH ] { ROW | STATEMENT }]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function (arguments)
```

WHEN (condition) that determines whether the trigger function will actually be executed

BEFORE, AFTER can be used for tables and views

INSTEAD OF can be only used for views at row-level

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-createtrigger.html

CREATE TRIGGER command - Explain

- Can occur either before or after the *event* occurs (INSERT, UPDATE,DELETE, TRUNCATE on the *table*)
  - multiple events can be specified using OR
  - UPDATE events, it is possible to specify a list of columns using this syntax: UPDATE OF *column_name1* [, *column_name2* ...]
  - INSTEAD OF UPDATE events do not support lists of columns.
- Fire triggers:
  - ROW: for each row that is affected by the event
  - STATEMENT: for each statement that triggers the event, no matter how many rows are returned (even if no rows are returned)

CREATE TRIGGER command - Explain

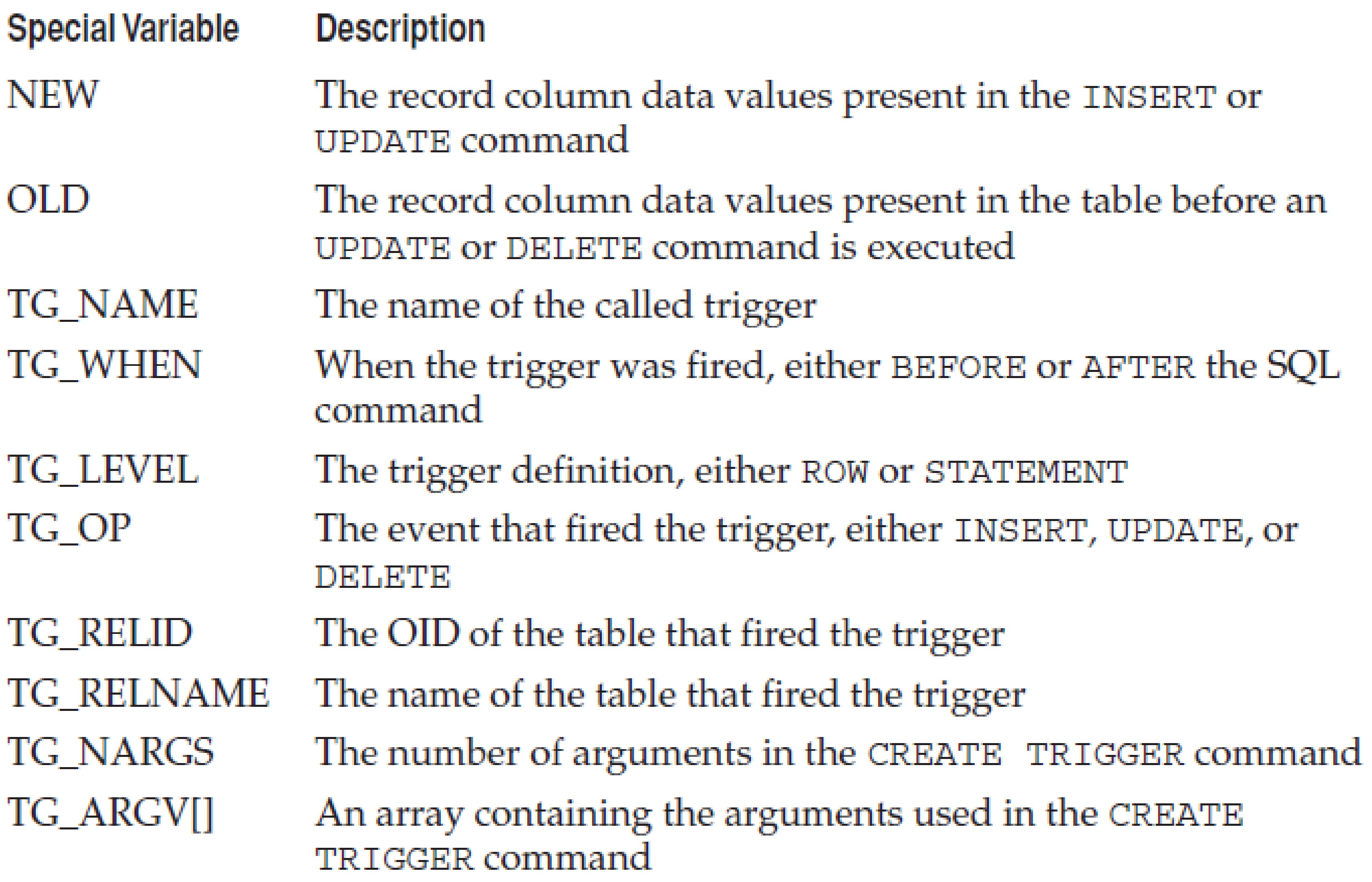
-
  ```
  CREATE TRIGGER check_update
  BEFORE UPDATE ON accounts
  FOR EACH ROW
  WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
  EXECUTE PROCEDURE check_account_update();
  ```

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-createtrigger.html

CREATE TRIGGER command

- *function:* execute when the trigger is fired
  - the arguments in the CREATE TRIGGER command are passed using the TG_ARGV special variable

- When a trigger function is called, the database engine passes a group of special variables to the trigger function →define the environment
  - how the function was called
  - what data is present when the trigger was fired
  - when the trigger was fired
  - …



Special Variable	Description
NEW	The record column data values present in the INSERT or UPDATE command
OLD	The record column data values present in the table before an UPDATE or DELETE command is executed
TG_NAME	The name of the called trigger
TG_WHEN	When the trigger was fired, either BEFORE or AFTER the SQL command
TG_LEVEL	The trigger definition, either ROW or STATEMENT
TG_OP	The event that fired the trigger, either INSERT, UPDATE, or DELETE
TG_RELID	The OID of the table that fired the trigger
TG_RELNAME	The name of the table that fired the trigger
TG_NARGS	The number of arguments in the CREATE TRIGGER command
TG_ARGV[]	An array containing the arguments used in the CREATE TRIGGER command

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/plpgsql-trigger.html

2. Creating a Trigger Function

- Can use the pgAdmin III program to create trigger functions:
  - right-click the Trigger Functions object →select New Trigger Function
  - Set the Language textbox to plpgsql
  - A trigger function updates table records → VOLATILE function
  - Parameter tab, NOT allowed to define arguments.
  - Definition textbox →enter the function code

Example (previous class)

```
-- define a trigger function to update a view (last weeek)
CREATE OR REPLACE FUNCTION delete_class_view_func()
RETURNS trigger AS $$
BEGIN
       -- update monitor id of clazz table to null
       -- if the student played a roll of class monitor
       update clazz set monitor_id = NULL
       WHERE monitor_id IN ( SELECT student_id FROM student
                             WHERE clazz_id = OLD.clazz_id);
       -- delete all enrollment of each student that will be deleted
       delete from enrollment
       where student_id IN (SELECT student_id FROM student
                             WHERE clazz_id = OLD.clazz_id);
       -- delete students in OLD.clazz_id from student table
       delete from student where clazz_id = OLD.clazz_id;
       -- delete clazz
       delete from clazz where clazz_id = OLD.clazz_id;
       RETURN OLD;
END;
$$ LANGUAGE plpgsql VOLATILE;
```

Example (previous class)

```
-- 'INSTEAD OF' trigger
-- DROP TRIGGER delete_class_view ON class_infos;

-- define a INSTEAD OF DELETE trigger
CREATE TRIGGER delete_class_view
INSTEAD OF DELETE ON class_infos
FOR EACH ROW
EXECUTE PROCEDURE delete_class_view_func();
```

Remarks

- RETURN in a trigger function
  - **NULL**
  - One record having the **same structure** as table record on which the trigger is defined
- Trigger « AFTER »:
  - RETURN NULL; -- or RETURN NEW; RETURN OLD;
- Trigger « BEFORE »
  - RETURN NULL; : subsequent triggers are not fired, and the INSERT/UPDATE/DELETE does not occur for this row
  - Trigger before-delete : RETURN OLD;
  - Before update or insert: RETURN NEW;

Example

Given EduBD:

student(**student_id**, first_name, last_name, dob, gender, address, note, *class_id*)

subject(**subject_id**, name, credit, percentage_final_exam)

lecturer(**lecturer_id**, first_name, last_name, dob, gender, address, email)

teaching(***subject_id, lecturer_id***)

grade(**code**, fromScore, toScore)

clazz(**clazz_id**, name, *lecturer_id*, *monitor_id,* number_students)

enrollment(***student_id, subject_id,* semester,** midterm_score, final_score)

Example

When a new student arrives (a new record is inserted into student table), the number of students in her/his class must be automatically updated

```
-- define a trigger
CREATE TRIGGER af_insert
AFTER INSERT ON student
FOR EACH ROW
WHEN (NEW.clazz_id IS NOT NULL)
EXECUTE PROCEDURE tf_af_insert();

-- define a trigger function
CREATE OR REPLACE FUNCTION tf_af_insert() RETURNS TRIGGER AS $$
BEGIN
       update clazz
       set number_students = number_students+1
       where clazz_id = NEW.clazz_id;

       RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Exercise

Given EduBD, write triggers to ensure the following requirement:

- If data on student table is changed, the number of students in clazz table is always correct.

(delete a student, change student class)