CMSC 206: Triggers and stored procedures

SQL

Thomas LAURENT

17/10/2020

University of the Philippines Open University

Table of contents

Introduction

Introduction

Last week we explored the SQL language and saw how to write separate statements and queries to define, manage, populate, and query a database.

This week we will learn about more complex mechanisms that let us build complex logic directly into our database rather than in an external application: triggers and stored procedures.

As always with SQL, the features and syntax will vary greatly based on the DBMS you are using, so the goal here is to make you aware of these tools rather than make you masters of them. If you need them one day you can check up on your DBMS' documentation (and probably on stackoverflow) to see how you can achieve the particular behaviour you want.

Triggers

Definition

Triggers are a mechanism that let us define **when-if-then** rules.

We can tell the DBMS to take an action (e.g. update some data) when something happens (e.g. data is inserted in a table) and a condition is met (e.g. the inserted value is bigger than 10).

Triggers let us embed monitoring logic directly into the database rather than in an external application, making it more efficient and flexible.

I strongly recommend that you check out Jennifer Widom's videos on the subjects in the additional material.

Syntax

Syntax

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER | INSTEAD OF } events ON table_name
 [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_table_name }]
 [ FOR [ EACH ] { ROW | STATEMENT } ]
 [ WHEN ( condition ) ]
EXECUTE action
```

Example

```
CREATE TRIGGER log_update AFTER UPDATE ON accounts FOR EACH
ROW WHEN (OLD.* IS DISTINCT FROM NEW.*) EXECUTE FUNCTION
log_account_update();
```

This will create a trigger that will run the log_account_update() every time an update is performed on the accounts table. The function will be run for each row affected by the update which data is actually changed (see WHEN clause).

Syntax explanation 1/2

CREATE TRIGGER trigger_name : choose the name of the trigger

{ BEFORE | AFTER | INSTEAD OF } : whether the trigger should be run before, after, or instead the action that triggered it

events on table_name: the events (INSERT, UPDATE, DELETE) on the table that will trigger the trigger

Syntax explanation 2/2

[REFERENCING { { OLD | NEW } TABLE [AS] transition_relation_name } [...]]: lets us set an alias for the old or new *transition tables*, tables that contains the rows affected by the event in their state before or after the event.

[FOR [EACH] { ROW | STATEMENT }]: whether the trigger should be run once per triggering statement or once per roe affected in the triggering statement

[WHEN (condition)]: sets a condition for the trigger to be run

EXECUTE action: lets use specify the action run by the trigger. In postgreSQL this will be a trigger function.

Stored procedures and functions

Definition

Up until now we have been interacting with our DBMS though single SQL statements. Although SQL can be very powerful, we might want to embed some more logic in our database, rather than having it all in our client application. This way we can reuse the same logic between projects, or expose some complex feature to the DB users without them having to implement it.

We can do that with stored procedures and function, which let us define complex behaviour by embedding procedural code (with ifs, loops, etc) directly into the database and call this logic in our SQL statement. We mentioned functions last week, talking about the functions offered by the DBMS (e.g. MIN, MAX, ...), but we can define our own functions.

The main difference between a function and a procedure is that a function may return something, while a procedure may not.

(very basic) Syntax

Syntax

```
CREATE [ OR REPLACE ] FUNCTION name ( [ [ argname ] argtype ] )
 [ RETURNS rettype ]
LANGUAGE lang_name
AS 'definition'
```

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE name ( [ [ argname ] argtype ] )
LANGUAGE lang_name
AS 'definition'
```

Examples

PlSQL function example

```
CREATE OR REPLACE FUNCTION increment(i integer)
RETURNS integer
LANGUAGE plpgsql AS $$
BEGIN
RETURN i + 1;
END;
$$;
```

SQL procedure example

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL AS $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;
```

Trigger functions

As we saw in the section on triggers, trigger actions in postgreSQL are represented by trigger functions. To create a trigger function we must declare the return type of the function as trigger.

For row level triggers that should happen on a data change we can then use the NEW variable to refer to the new row (for inserts or updates) and the OLD variable to refer to the old row (for deletes and updates).

Languages

As the examples show, functions and procedures can be written in different languages, fro plan SQL, to C or Java given some DBMS interfacing. Postgresql offers its own procedural language, PL/pgSQL, similar to oracle's PL/SQL.

Considering a developer's proficiency, and things like performance, one might decide to use different languages to implement stored procedures and functions.