Triggers and Stored Procedures

CMSC 206 – Database Management Systems

Revision History
22 Nov 2022: Examples update – Ria Borromeo
10 October 2020: Main content – Thomas Laurent

Outline

1. Introduction
2. Triggers
3. Stored procedures and functions

Introduction

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

In this lecture, 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 with the DBMS that you are using so the goal here is to make you aware of these tools instead of making you masters of them. If you need them one day, you can always check the specific DBMS' documentation (and probably stackoverflow) to see how you can do what you intend to do.

Triggers

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

We can tell the DBMS to do something (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.

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
```

Syntax

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 data which 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

`[ 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.

Syntax Explanation (2/2)

`[ 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.

SQLite Tutorial for Triggers: https://www.sqlitetutorial.net/sqlite-trigger/

Stored Procedures and Functions

Definition

- Up until now we have been interacting with our DBMS through 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 functions, which let us define complex behavior by embedding procedural code (with ifs, loops, etc.) directly into the database and call this logic in our SQL statement. We mentioned in the previous lecture that there are functions offered by the DBMS (MIN, MAX, etc.), but we can define our own functions.
- The main difference between a function and a procedure is that a function may returns 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’
```

Syntax

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 $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;
```

Unfortunately, you cannot create procedures and functions in SQLite.

Languages

- As the examples show, functions and procedures can be written in different languages, from plain SQL to C or Java given some DBMS interfacing. SQLite does not support user-defined procedures and functions but PostgreSQL offers its own procedural language, PL/pgSQL, which is similar to Oracle's PL/SQL.

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