PL/SQL- Trigger

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

NOVEMBER 20, 2023

Contents

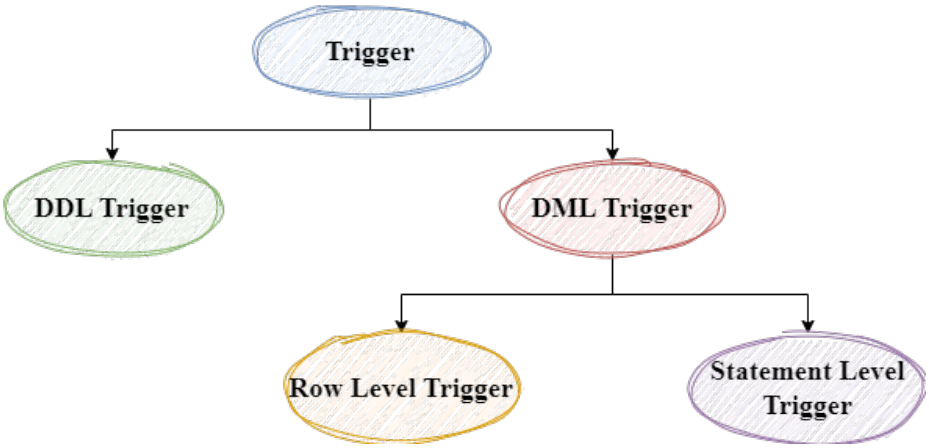
1 Introduction

A trigger is a named block that a system executes automatically when there is any modification (insert, delete, update) to the database. A trigger has two parts:

- Event/condition for which the trigger is to be executed.
- Action that will occur when the trigger executes.

Since trigger is event-oriented, it is not explicitly called like function rather it automatically fires wherever the condition is met.

2 Classification



2.1 DDL Trigger

These triggers fire when you create, change, or remove objects in a database schema. They are useful for controlling or monitoring DDL statements.
Events that work with DDL triggers

- ALTER, CREATE, DROP, GRANT, RENAME, REVOKE.

2.1.1 DDL trigger Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} ddl_event ON {DATABASE | SCHEMA}
[WHEN (logical_expression)]
[DECLARE]
declaration_statements;
BEGIN
execution_statements;
END
```

Few Points:

- **SCHEMA trigger:** The keyword SCHEMA decides its type. A SCHEMA trigger is created on a schema and fires whenever the user who owns it is the current user and initiates the triggering event.
- **DATABASE Triggers:** Use the keyword DATABASE for it. A DATABASE trigger is created on the database and fires whenever any database user initiates the triggering event.

2.1.2 DDL trigger Example:

Suppose we want to log 3 types of DDL (CREATE, ALTER AND DELETE) for a particular user (schema).

```
create or replace trigger ddl_trigger
before create or alter or drop on SCHEMA
begin
dbms_output.put_line('Who did it?'||ora_dict_obj_owner);
dbms_output.put_line('What was the Operation?'||ora_sysevent);
dbms_output.put_line('On what?'||ora_dict_obj_name);
dbms_output.put_line('On type of object it was?'||
   ora_dict_obj_end;
```

2.2 DML Trigger

DML triggers can fire before or after INSERT, UPDATE, and DELETE statements. DML triggers can be statement or row-level activities.

- Statement-level triggers fire and perform a statement or set of statements once no matter how many rows are affected by the DML event.
- Row-level triggers fire and perform a statement or set of statements for each row changed by a DML statement.

2.2.1 DML trigger Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER}
{INSERT | UPDATE | UPDATE OF [column1, column2, ..., column(n)] | DELETE}
ON table_name
[FOR EACH ROW]
[WHEN (logical_expression)]
[DECLARE]
[PRAGMA AUTONOMOUS_TRANSACTION;]
declaration_statements;
BEGIN
execution_statements;
END [trigger_name];
```

Note: The clause [FOR EACH ROW] specifies that it is row-level trigger. For statement-level triggers just omit that part.

2.2.2 DML trigger Example

Suppose the trigger will fire after each insert, update, or delete of the employee table.

```
CREATE OR REPLACE TRIGGER demo_trigger_types1
BEFORE DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
dbms_output.put_line('Row Level Trigger Fires each time.');
END;
```

2.3 OLD and NEW Pseudorecords

When a row-level trigger fires, the PL/SQL runtime system creates and populates the two pseudorecords OLD and NEW. They are called pseudorecords because they have some, but not all, of the properties of records.
For the row that the trigger is processing:

- For an INSERT trigger, OLD contains no values, and NEW contains the new values.
- For an UPDATE trigger, OLD contains the old values, and NEW contains the new values.
- For a DELETE trigger, OLD contains the old values, and NEW contains no values.

2.3.1 OLD and NEW pseudorecords Example

Suppose the trigger will show the salary of employees before and after the update.

```
CREATE OR REPLACE TRIGGER demo_old_new
BEFORE UPDATE ON employees
FOR EACH ROW
WHEN (NEW.EMPLOYEE_ID > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.SALARY - :OLD.SALARY;
dbms_output.put('Old salary:' || :OLD.SALARY);
dbms_output.put('New salary:' || :NEW.SALARY);
dbms_output.put_line('Difference' || sal_diff);
END;
```

2.4 Sequence

Mainly used to handle an auto number field.

```
Syntax:
CREATE SEQUENCE sequence_name
MINVALUE value
MAXVALUE value
START WITH value
INCREMENT BY value
CACHE value;
```

Here, MINVALUE refers to the minimum possible value for the sequence while MAXVALUE refers to the maximum possible value. START WITH indicates from which number the sequence will be started which can be at minimum the MINVALUE. INCREMENT BY says the interval of two consecutive values. Lastly, CACHE prepares the sequence beforehand upto the value just like the normal concept of CACHE.

2.4.1 Example of sequence

```
CREATE SEQUENCE evaluations_sequence
MINVALUE 1
MAXVALUE 999999
```

```
START WITH 1
INCREMENT BY 1
CACHE 20;
```

Now, we can use this sequence inside a trigger. Let's say,

```
CREATE OR REPLACE TRIGGER NEW_EVALUATION_TRIGGER
BEFORE INSERT ON EVALUATIONS
FOR EACH ROW
BEGIN
:NEW.evaluation_id := evaluations_sequence.NEXTVAL
END;
```

Note: Using the keyword NEXTVAL, one can iterate through the sequence number.

2.5 Order of Firing Trigger

Oracle allows more than one trigger to be created for the same timing point, but it has never guaranteed the execution order of those triggers. The Oracle 11g trigger syntax now includes the FOLLOWS clause to guarantee execution order for triggers defined with the same timing point. The following example creates a table with two triggers for the same timing point.

2.5.1 Example firing order

```
CREATE OR REPLACE TRIGGER trigger_2
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
DBMS_OUTPUT.put_line('TRIGGER_2: insertion - Executed');
END;
```

```
CREATE OR REPLACE TRIGGER trigger_1
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
FOLLOWS trigger_2
BEGIN
DBMS_OUTPUT.put_line('TRIGGER_1: update - Executed');
END;/
```

Now the TRIGGER_1 trigger will always follow the TRIGGER_2.