Assignment 4
Class: SE Comp
Year: 2020-21
Performed by: Danyl Fernandes, 72

Q1. Comprehend Triggers with suitable examples. Write the applications of triggers:

Trigger:

- A trigger is a stored procedure in a database which automatically invokes whenever a special event in the database occurs.
- For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- It is a set of actions which get executed automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a particular table.
    - A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
    - A database definition (DDL) statement (CREATE, ALTER, or DROP).
    - A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN)

Purpose of Triggers:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions
- Validate input data

Trigger Classification:

- Classification based on the timing:
    - BEFORE Trigger: This trigger is fired before the occurrences of a specified event.
    - AFTER Trigger: This trigger is fired after the occurrences of a specified event.
- Classification based on the level:
    - STATEMENT level Trigger: This trigger is fired for once for the specified event statement.
    - ROW level Trigger: This trigger is fired for all the records which are affected by the specified event statement (Only for DML).

- Classification based on the Event:
    - DML Trigger: This trigger fires when the DML event such as DELETE, INSERT, or UPDATE is specified.
    - DDL Trigger: This trigger fires when the DDL event such as CREATE, ALTER, is specified.
    - DATABASE Trigger: This trigger fires when the database event such as LOGON, LOGOFF, STARTUP, or SHUTDOWN, is specified.

Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name] ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
     DECLARE
     Declaration-statements
BEGIN
     Executable-statements
EXCEPTION
     Exception-handling-statements
END;
```

Where,

- `CREATE [OR REPLACE] TRIGGER trigger_name` – Creates or replaces an existing trigger with the trigger_name.
- `{BEFORE | AFTER | INSTEAD OF}` – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating triggers on a view.
- `{INSERT [OR] | UPDATE [OR] | DELETE}` – This specifies the DML operation.
- `[OF col_name]` – This specifies the column name that will be updated.
- `[ON table_name]` – This specifies the name of the table associated with the trigger.
- `[REFERENCING OLD AS o NEW AS n]` – This allows you to refer new and old values for various
- DML statements, such as INSERT, UPDATE, and DELETE.
- `[FOR EACH ROW]` – This speci□ies a row-level trigger, i.e., the trigger will be executed for each
- row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- `WHEN (condition)` – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example:

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters:

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

Query:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
     sal_diff number;
BEGIN
     sal_diff := :NEW.salary - :OLD.salary;
     dbms_output.put_line('Old salary: ' || :OLD.salary);
     dbms_output.put_line('New salary: ' || :NEW.salary);
     dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

The following points need to be considered here:

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Applications:

- Automatic generation of derived column values.
- Prevention to invalid transactions.
- Enforcing complex security authorizations.
- Enforcing referential integrity across nodes in a distributed database.
- Enforcing complex business rules.
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates.
- Gather statistics on table access.
- Modify table data when DML statements are issued against views.
- Publish information about database events, user events, and SQL statements to subscribing applications

Q2. Write one example for Before insert, after insert, before delete, after delete, before update, after update trigger:

After Insert:

```
DELIMITER $$
USE `hr`
$$
CREATE
DEFINER=`root`@`127.0.0.1`
TRIGGER `hr`.`emp_details_AINS`
AFTER INSERT ON `hr`.`emp_details`
FOR EACH ROW
BEGIN
     INSERT INTO log_emp_details
     VALUES(NEW.employee_id, NEW.salary, NOW());
END$$
```

Before Insert:

```
USE `hr`;
DELIMITER
$$
CREATE TRIGGER `emp_details_BINS`
BEFORE INSERT
ON emp_details FOR EACH ROW
BEGIN
     SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
          SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
SET NEW.JOB_ID = UPPER(NEW.JOB_ID);END;
$$
```

Before Insert:

```
DELIMITER
$$
USE `test`
$$
CREATE
DEFINER=`root`@`127.0.0.1`
TRIGGER `test`.`student_mast_AUPD`
AFTER UPDATE
ON `test`.`student_mast`FOR EACH ROW
BEGIN
INSERT into stu_log VALUES (user(), CONCAT('Update Student
Record’, OLD.NAME,'Previous Class:',OLD.ST_CLASS,' Present
Class’, NEW.st_class));
END
$$
```

Before Update:

```
DELIMITER
$$
CREATE TRIGGER `student_marks_BUPD`
BEFORE UPDATE
ON student_marks FOR EACH ROW
BEGIN
SET NEW.TOTAL = NEW.SUB1 + NEW.SUB2 + NEW.SUB3 + NEW.SUB4 +
NEW.SUB5;
SET NEW.PER_MARKS = NEW.TOTAL/5;
IF NEW.PER_MARKS >=90 THEN
SET NEW.GRADE = 'EXCELLENT';
ELSEIF NEW.PER_MARKS>=75 AND NEW.PER_MARKS<90 THEN
SET NEW.GRADE = 'VERY GOOD';
ELSEIF NEW.PER_MARKS>=60 AND NEW.PER_MARKS<75 THEN
SET NEW.GRADE = 'GOOD';
ELSEIF NEW.PER_MARKS>=40 AND NEW.PER_MARKS<60 THEN
SET NEW.GRADE = 'AVERAGE';
ELSESET NEW.GRADE = 'NOT PROMOTED';
END IF;
END;
$$
```

After Delete:

```
DELIMITER
$$
CREATE TRIGGER `student_mast_ADEL`
AFTER DELETE ON student_mast FOR EACH ROW
BEGIN
INSERT into stu_log VALUES (user(), CONCAT('Update Student
Record ',
OLD.NAME,' Class:',OLD.ST_CLASS, '-> Deleted on ', NOW()));
END;
$$
```

Before Delete:

```
DELIMITER $$
CREATE TRIGGER before_salaries_delete
BEFORE DELETE
ON salaries FOR EACH ROW
BEGIN
INSERT INTO SalaryArchives(employeeNumber,validFrom,amount)
VALUES(OLD.employeeNumber,OLD.validFrom,OLD.amount);
END$$
```