

Name: Renuka Wankhade

Roll No.:213046

Batch: A2

PRN No.: 22211581

DBMS Lab Assignment - 8

SQL Triggers

A SQL trigger is a set of SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete. A SQL trigger is a special type of [stored procedure](#). It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

```
CREATE [OR REPLACE] TRIGGER trigger_name
  BEFORE | AFTER
  [INSERT, UPDATE, DELETE [COLUMN NAME..]
  ON table_name

  Referencing [ OLD AS OLD | NEW AS NEW ]
  FOR EACH ROW | FOR EACH STATEMENT [ WHEN Condition ]

DECLARE
  [declaration_section
    variable declarations;
    constant declarations;
  ]

BEGIN
  [executable_section
    PL/SQL execute/subprogram body
  ]

EXCEPTION
  [exception_section
    PL/SQL Exception block
  ]

END;
```

CREATE [OR REPLACE] TRIGGER trigger_name : Create a trigger with the given name. If already have overwrite the existing trigger with defined same name.

BEFORE | AFTER : Indicates when the trigger get fire. BEFORE trigger execute before when statement execute before. AFTER trigger execute after the statement execute.

INSERT, UPDATE, DELETE [COLUMN NAME..] : Determines the performing trigger event. You can define more then one triggering event separated by OR keyword.

ON table_name : Define the table name to performing trigger event.

Referencing [OLD AS OLD | NEW AS NEW] : This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.

For each row | for each statement: Trigger must fire when each row gets Affected (ROW Trigger). and fire only once when the entire sql statement is execute (STATEMENT Trigger).

WHEN Condition : Optional. This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers. Trigger fire when specified condition is satisfied.

Assignment 8 Lab Questions

Implementation Questions:

- 1) Implement After update trigger on the Books Table.

Query:

Table for recording the update logs:

- CREATE TABLE Books_Update_Log (
 - Log_ID INT PRIMARY KEY auto_increment,
 - Book_ID INT,
 - Old_Price DECIMAL(10, 2),
 - New_Price DECIMAL(10, 2),
 - Update_Date TIMESTAMP
-);

Trigger Code:

- DELIMITER \$\$
- USE `library` \$\$
- CREATE DEFINER=`root`@`localhost` TRIGGER `Books_After_Update` AFTER UPDATE ON `books` FOR EACH ROW BEGIN
- IF OLD.Price <> NEW.Price THEN
- INSERT INTO Books_Update_Log (Book_ID, Old_Price, New_Price, Update_Date)
- VALUES (NEW.Bid, OLD.Price, NEW.Price, NOW());
- END IF;
- END \$\$
- DELIMITER ;

After Updating:

- update books set price = 2499 where bid = 4;

Output:

	Log_ID	Book_ID	Old_Price	New_Price	Update_Date
▶	1	4	2000.00	2499.00	2024-03-30 11:22:37
✱	NULL	NULL	NULL	NULL	NULL

2) Implement before insert trigger on the Booksby Table.

Query:

Trigger Code:

- DELIMITER \$\$
- USE `library`\$\$
- CREATE DEFINER = CURRENT_USER TRIGGER `library`.`books_BEFORE_INSERT` BEFORE INSERT ON `books` FOR EACH ROW
- BEGIN
- IF NEW.Price IS NULL THEN
- SET NEW.Price = 10.00;
- END IF;
- END\$\$
- DELIMITER ;

Inserted Values Without Adding the price of the book(Price get added by default):

- insert into books(bid, bname, lid, pid) values
- (7, "Computer Networks", 104, 107);

Output:

	Bid	Bname	Price	Lid	Pid
▶	1	Introduction to SQL	1089.00	101	101
	2	Ghost rider One	625.00	102	106
	3	Data Structures and Al...	1100.00	102	103
	4	Artificial Intelligence	2499.00	104	104
	5	Machine Learning Basics	2500.00	105	105
	6	The Complete Reference	2999.00	101	1
	7	Computer Networks	10.00	104	107
	4444	Introduction to Python	1000.00	102	102
*	NULL	NULL	NULL	NULL	NULL

3) Implement After update trigger on the Purchase Table.

Query:

Table for recording the update logs

- CREATE TABLE Purchase_Update_Log (
 - Log_ID INT AUTO_INCREMENT PRIMARY KEY,
 - Purchase_ID INT,
 - Old_Quantity INT,
 - New_Quantity INT,
 - Update_Date TIMESTAMP
-);

Trigger Code:

- DELIMITER \$\$
- USE `library` \$\$
- CREATE DEFINER = CURRENT_USER TRIGGER `library`.`purchase_AFTER_UPDATE` AFTER UPDATE ON `purchase` FOR EACH ROW
- BEGIN
- INSERT INTO Purchase_Update_Log (Purchase_ID, Old_Quantity, New_Quantity, Update_Date)
- VALUES (NEW.prid, OLD.quantity, NEW.quantity, NOW());
- END \$\$
- DELIMITER ;

After Updating;

- update purchase set quantity = 15 where prid = 2;

Output:

	Log_ID	Purchase_ID	Old_Quantity	New_Quantity	Update_Date
▶	1	2	8	15	2024-03-30 11:50:29
*	NULL	NULL	NULL	NULL	NULL

Sample Triggers code:

```
create table customers(  
id number,  
name varchar(30),  
age number,  
addr varchar(50),  
sal number  
);
```

```
insert into customers values(1,'Ramesh', 23, 'Allahbad', 20000);
```

```
insert into customers values(2,'Suresh', 22, 'Kanpur', 22000);
```

```
insert into customers values(3,'Mahesh', 24, 'Gaziabad', 24000);
```

```
insert into customers values(4,'Chandan', 25, 'Noida', 26000);
```

```
insert into customers values(5,'Alex', 21, 'Paris', 28000);
```

```
insert into customers values(6,'Sunita', 20, 'Delhi', 30000);
```

```
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.sal - :OLD.sal;
```

```
dbms_output.put_line('Old salary: ' || :OLD.sal);  
dbms_output.put_line('New salary: ' || :NEW.sal);  
dbms_output.put_line('Salary difference: ' || sal_diff);  
END;  
  
/  
  
DECLARE  
  
    total_rows number(2);  
  
BEGIN  
  
    UPDATE customers  
  
    SET sal = sal + 5000;  
  
    IF sql%notfound THEN  
  
        dbms_output.put_line('no customers updated');  
  
    ELSIF sql%found THEN  
  
        total_rows := sql%rowcount;  
  
        dbms_output.put_line( total_rows || ' customers updated ');  
  
    END IF;  
  
END;  
  
/
```