MIT World Peace University

Database Management System

Assignment 7

Naman Soni Roll No. 10

Contents

1 Aim

Write PL/SQL Triggers for the creation of insert trigger, delete trigger and update trigger on the given problem statements

2 Objective

- TO study and use Triggers using MySQL PL/SQL block

3 Theory

3.1 Triggers In PL/SQL

A trigger in PL/SQL is a database object that is automatically executed in response to certain events, such as an insert, update, or delete operation on a table. A trigger consists of PL/SQL code that is associated with a specific table, and is executed automatically by the Oracle database when the associated event occurs.
Triggers are typically used to implement business rules or other data integrity constraints that cannot be enforced using database constraints or other mechanisms. For example, a trigger can be used to enforce a rule that prevents a customer from placing an order if their account balance is negative.
There are two types of triggers in PL/SQL:

1. **Row-level Triggers:** A row-level trigger is a trigger that is executed once for each row affected by the triggering statement. Row-level triggers can be used to enforce complex data integrity constraints or to audit changes to the data in a table.

2. **Statement-level Triggers:** A statement-level trigger is a trigger that is executed once for each triggering statement, regardless of the number of rows affected by the statement. Statement-level triggers can be used to enforce simple data integrity constraints.

Triggers can be defined to execute either before or after the triggering event. A trigger that executes before the event is called a "before" trigger, while a trigger that executes after the event is called an "after" trigger.

3.2 Advantages of Triggers

Triggers in PL/SQL provide several advantages, including:

1. **Ensuring Data Integrity:** Triggers can be used to enforce complex data integrity constraints that cannot be enforced using database constraints or other mechanisms.

2. **Auditing Changes** riggers can be used to track changes to the data in a table, such as updates, inserts, and deletes. This can be useful for auditing and tracking changes made to sensitive data.

3. **Automating Tasks:** Triggers can be used to automate tasks that would otherwise have to be performed manually, such as generating derived column values.

4. **Enforcing Security:** Triggers can be used to enforce security policies that cannot be enforced using database security mechanisms.

5. **Simplifying Application Logic:** Triggers can be used to simplify application logic by encapsulating complex operations in the database.

3.3 Usages of Triggers

Triggers in PL/SQL can be used for a variety of purposes, including:

1. **Enforcing Data Integrity:** Triggers can be used to enforce complex data integrity constraints that cannot be enforced using database constraints or other mechanisms.

2. **Auditing Changes:** Triggers can be used to track changes to the data in a table, such as updates, inserts, and deletes. This can be useful for auditing and tracking changes made to sensitive data.
3. **Automating Tasks:** Triggers can be used to automate tasks that would otherwise have to be performed manually, such as generating derived column values.
4. **Enforcing Security:** Triggers can be used to enforce security policies that cannot be enforced using database security mechanisms.
5. **Implementing Complex Business Rules:** Triggers can be used to implement complex business rules that cannot be implemented using database constraints or other mechanisms.
6. **Replicating Data:** Triggers can be used to replicate data to other tables or databases.
7. **Trigger-Based Debugging:** Triggers can be used to debug applications by logging information to a table when certain events occur.

3.4 Types of Trigger In PL/SQL

3.4.1 Trigger Level

1. **Table-Level Triggers:** A table-level trigger is a trigger that is fired once for each row affected by the triggering statement. Table-level triggers can be used to enforce complex data integrity constraints or to audit changes to the data in a table.
2. **Schema-Level Triggers:** A schema-level trigger is a trigger that is fired once for each statement, regardless of the number of rows affected by the statement. Schema-level triggers can be used to enforce simple data integrity constraints.

Schema-level triggers are always statement-level triggers, and they are typically used for auditing and enforcing security policies on the schema as a whole. Both table-level and schema-level triggers can be used to enforce data integrity, audit changes to data, and perform maintenance tasks on the database. However, they should be used with care, as poorly-designed triggers can have a negative impact on database performance and scalability.

3.4.2 Trigger Timings

In PL/SQL, triggers can be defined to fire at different timings or events, depending on the needs of the application. The timing of a trigger determines when it is executed in relation to the triggering event. There are two basic types of trigger timings:

1. **Before Triggers:** A before trigger is a trigger that is executed before the triggering event occurs. Before triggers are typically used to enforce data integrity constraints or to perform maintenance tasks on the database.
2. **After Triggers:** An after trigger is a trigger that is executed after the triggering event occurs. After triggers are typically used to audit changes to data or to perform maintenance tasks on the database.

Both BEFORE and AFTER triggers can be defined as row-level triggers or statement-level triggers, as discussed in the previous answer.
In addition to BEFORE and AFTER trigger timings, there is also a special timing called INSTEAD OF, which is used for views. An INSTEAD OF trigger is executed instead of the triggering event, and can be used to implement complex views, such as views that join multiple tables or perform calculations on the data.

3.4.3 Trigger Events

In PL/SQL, triggers can be defined to fire on different events, depending on the needs of the application. The event that triggers a trigger is called the triggering event. There are three basic types of triggering events:

1. **INSERT:** An INSERT trigger is a trigger that is fired when a new row is inserted into a table. INSERT triggers are typically used to enforce data integrity constraints or to perform maintenance tasks on the database.

2. **UPDATE:** An UPDATE trigger is a trigger that is fired when a row is updated in a table. UPDATE triggers are typically used to audit changes to data or to perform maintenance tasks on the database.
3. **DELETE:** A DELETE trigger is a trigger that is fired when a row is deleted from a table. DELETE triggers are typically used to audit changes to data or to perform maintenance tasks on the database.
4. **CREATE:** A CREATE trigger is a trigger that is fired when a new object is created in the database. CREATE triggers are typically used to enforce security policies or to perform maintenance tasks on the database.
5. **ALTER:** An ALTER trigger is a trigger that is fired when an object is altered in the database. ALTER triggers are typically used to enforce security policies or to perform maintenance tasks on the database.
6. **DROP:** A DROP trigger is a trigger that is fired when an object is dropped from the database. DROP triggers are typically used to enforce security policies or to perform maintenance tasks on the database.

3.5 NEW and OLD Clause/Trigger Variables

In PL/SQL, triggers can access the "old" and "new" values of the row being inserted, updated, or deleted. These values are stored in special variables called the "OLD" and "NEW" values.

1. **OLD Values:** The "old" values of a row are the values of the row before it was updated or deleted. These values are stored in the "OLD" values of the row.
2. **NEW Values:** The "new" values of a row are the values of the row after it was updated or inserted. These values are stored in the "NEW" values of the row.

These variables can be used in triggers to perform various actions, such as validating data, enforcing business rules, or logging changes to the table. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.

3.6 Dropping Triggers

In PL/SQL, triggers can be dropped using the DROP TRIGGER statement. This statement removes the trigger from the database and frees up any resources that were being used by the trigger.
Syntax:

```
DROP TRIGGER trigger_name;
```

In this statement, "trigger_name" is the name of the trigger that you want to drop. When you execute this statement, the trigger will be dropped from the schema and all associated metadata will be deleted.
Before dropping a trigger, it is important to ensure that the trigger is no longer needed and that it is safe to remove it from the database. You should also ensure that any dependencies on the trigger have been removed, such as views or stored procedures that reference the trigger.
In addition to dropping individual triggers, you can also drop all triggers for a table or schema using the DROP TRIGGER statement with the "ON" keyword. For example, to drop all triggers for a table named "my_table", you could use the following statement:

```
DROP TRIGGER ON my_table;
```

This statement will drop all triggers that are associated with the "my_table" table, including any triggers that were defined to fire on INSERT, UPDATE, or DELETE events.

4 Platform

- **Operating System:** Garuda Linux
- **IDE or Text Editor:** Visual Studio Code

5 input

Given Database from the Problem Statement for the Assignment for our batch.

6 Queries

```sql
-- Assignment 7 Triggers

-- Consider the following relational schema: BOOK (Isbn, Title,
-- SoldCopies) WRITING (Isbn, Name) AUTHOR (Name, SoldCopies)

-- Define a set of triggers for keeping SoldCopies in AUTHOR updated
-- with respect to: updates on SoldCopies in BOOK insertion of new tuples
-- in the WRITING relation

-- Create Database and Tables

CREATE database if not exists store;
use store;
CREATE TABLE BOOK (
    Isbn VARCHAR(10) PRIMARY KEY,
    Title VARCHAR(100),
    SoldCopies INT
);

CREATE TABLE WRITING (
    Isbn VARCHAR(10),
    Name VARCHAR(50),
    PRIMARY KEY (Isbn, Name),
    FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
);

CREATE TABLE AUTHOR (
    Name VARCHAR(50) PRIMARY KEY,
    SoldCopies INT
);

CREATE TABLE Customer (
    cust_id INT PRIMARY KEY,
    Principal_amount DOUBLE,
    Rate_of_interest DOUBLE,
    Years INT
);


INSERT INTO BOOK (Isbn, Title, SoldCopies)
VALUES ('9783161', 'Crime and Punishment', 500);

INSERT INTO WRITING (Isbn, Name)
VALUES ('9783161', 'Fyodor Dostoevsky');

INSERT INTO AUTHOR (Name, SoldCopies)
VALUES ('Fyodor Dostoevsky', 500);

INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest, Years)
VALUES (1, 1000, 0.05, 5);

-- Create Triggers

DELIMITER //
CREATE TRIGGER update_author_soldcopies
AFTER UPDATE ON BOOK
FOR EACH ROW
BEGIN
    IF NEW.SoldCopies != OLD.SoldCopies THEN
        UPDATE AUTHOR
        SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies) WHERE Name IN (SELECT
    Name FROM WRITING WHERE Isbn = NEW.Isbn);
```

```
    END IF;
END//
DELIMITER ;

delimiter $$
CREATE TRIGGER insert_author_soldcopies
AFTER INSERT ON WRITING
FOR EACH ROW
BEGIN
    UPDATE AUTHOR
    SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn = NEW.Isbn)
    WHERE Name = NEW.Name;
END $$
delimiter ;
```

7 Outputs

```


MariaDB [(none)]>
MariaDB [(none)]> CREATE database if not exists store;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use store;
Database changed
MariaDB [store]> CREATE TABLE BOOK (
    ->     Isbn VARCHAR(10) PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE WRITING (
    ->     Isbn VARCHAR(10),
    ->     Name VARCHAR(50),
    ->     PRIMARY KEY (Isbn, Name),
    ->     FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE AUTHOR (
    ->     Name VARCHAR(50) PRIMARY KEY,
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE Customer (
    ->     cust_id INT PRIMARY KEY,
    ->     Principal_amount DOUBLE,
    ->     Rate_of_interest DOUBLE,
    ->     Years INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]>
MariaDB [store]> INSERT INTO BOOK (Isbn, Title, SoldCopies)
    -> VALUES ('9783161', 'Crime and Punishment', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO WRITING (Isbn, Name)
    -> VALUES ('9783161', 'Fyodor Dostoevsky');
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
```

```
MariaDB [store]> INSERT INTO AUTHOR (Name, SoldCopies)
    -> VALUES ('Fyodor Dostoevsky', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest, Years)
    -> VALUES (1, 1000, 0.05, 5);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> -- Create Triggers
MariaDB [store]>
MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE TRIGGER update_author_soldcopies
    -> AFTER UPDATE ON BOOK
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.SoldCopies != OLD.SoldCopies THEN
    ->         UPDATE AUTHOR
    ->         SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies) WHERE Name IN (
    SELECT Name FROM WRITING WHERE Isbn = NEW.Isbn);
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
MariaDB [store]> delimiter $$
MariaDB [store]> CREATE TRIGGER insert_author_soldcopies
    -> AFTER INSERT ON WRITING
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE AUTHOR
    ->     SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn = NEW.Isbn)
    ->     WHERE Name = NEW.Name;
    -> END $$
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> delimiter ;
MariaDB [store]>
MariaDB [store]> select * from BOOK;
+---------+----------------------+------------+
| Isbn    | Title                | SoldCopies |
+---------+----------------------+------------+
| 9783161 | Crime and Punishment |        500 |
+---------+----------------------+------------+
1 row in set (0.002 sec)

MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
+--------------------+------------+
| Fyodor Dostoevsky |        500 |
+--------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from WRITING;
+---------+--------------------+
| Isbn    | Name               |
+---------+--------------------+
| 9783161 | Fyodor Dostoevsky |
+---------+--------------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from Customer;
+---------+------------------+------------------+-------+
| cust_id | Principal_amount | Rate_of_interest | Years |
+---------+------------------+------------------+-------+
|       1 |             1000 |             0.05 |     5 |
+---------+------------------+------------------+-------+
```

```
1 row in set (0.001 sec)

-- Test Triggers


MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
+--------------------+------------+
| Fyodor Dostoevsky |        500 |
+--------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> UPDATE BOOK SET SoldCopies = 600 WHERE Isbn = '9783161';
Query OK, 1 row affected (0.004 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
+--------------------+------------+
| Fyodor Dostoevsky |        600 |
+--------------------+------------+
1 row in set (0.000 sec)
```

8 Conclusion

Thus, we have learned creating and using triggers in SQL

9 FAQs

9.1 Enlist Advantages of Triggers?

There are several advantages to using triggers in PL/SQL. Some of the main advantages include:

1. **Data Consistency:** Triggers can be used to enforce data consistency in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.

2. **Business logic enforcement:** Triggers can be used to enforce business rules in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.

3. **Audit trail:** Triggers can be used to log changes to the database. For example, a trigger might be defined to log changes to a table in a separate table. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then insert the changes into the audit table.

4. **Simplified application logic:** Triggers can be used to simplify application logic. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.

5. **Performance Optimization:** Triggers can be used to optimize performance in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.

6. **Data Integrity:** Triggers can be used to enforce data integrity in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.

9.2 Enlist Disadvantages of Triggers?

There are several disadvantages to using triggers in PL/SQL. Some of the main disadvantages include:

1. **Performance Overhead:** Triggers can cause performance overhead in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
2. **Complexity:** Triggers can cause complexity in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
3. **Maintenance:** Triggers can cause maintenance issues in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
4. **Security:** Triggers can cause security issues in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
5. **Data Integrity:** Triggers can cause data integrity issues in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
6. **Data Consistency:** Triggers can cause data consistency issues in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.

9.3 What are the Applications of Triggers?

Triggers can be used in a variety of applications. Some of the main applications include:

1. **Data Validation:** Triggers can be used to validate data in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.
2. **Business Logic Enforcement:** Triggers can be used to enforce business rules in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.
3. **Audit Trail:** Triggers can be used to log changes to the database. For example, a trigger might be defined to log changes to a table in a separate table. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then insert the changes into the audit table.
4. **Performance optimization:** Triggers can be used to optimize performance in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.
5. **Data synchronization:** Triggers can be used to synchronize data in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.

6. **Security enforcement:** Triggers can be used to enforce security in the database. For example, a trigger might be defined to prevent updates to a table if the new value of a column does not meet certain criteria. The trigger could check the old and new values of the column using the OLD and NEW variables, and roll back the transaction if the update does not meet the criteria.
7. **Complex calculations:** Triggers can be used to perform complex calculations in the database. For example, a trigger might be defined to automatically update a column in a table when another column is updated. The trigger could use the OLD and NEW variables to determine what changes were made to the table, and then update the column in the table.