Experiment No.8
Implementation of Views and Triggers
Date of Performance:
Date of Submission:

Aim :- Write a SQL query to implement views and triggers

Objective :- To learn about virtual tables in the database and also PLSQL

constructs **Theory:**

SQL Views:

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

DROP VIEW view_name;

Trigger: A trigger is a stored procedure in the 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.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger. 5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Implementation:

Code:
View:-

```
136
137 •  CREATE USER 'ankit'@'localhost';
138
139 •  DROP USER 'ankit'@'localhost';
140
141 •  GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
142
143 •  GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
144
145 •  LOCK TABLES FMS WRITE;
146
147 •  LOCK TABLES SUPPLIERS READ;
148
149 •  UNLOCK TABLES;
150
151 •  CREATE VIEW Farmer_Farm_View AS
152    SELECT FMS.F_ID, FMS.F_NAME, FMS.L_NAME, FMS.AGE, FMS.MOB_NO, FMS.PLACE, FARM.farm_id, FARM.farm_size, FARM.farm_type, FARM.farm_location
153    FROM FMS
154    INNER JOIN FARM ON FMS.F_ID = FARM.farm_id;
155
156 •  SELECT * FROM Farmer_Farm_View;
157
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	farm_id	farm_size	farm_type	farm_location

Code:

Trigger:

```
58
59 •    INSERT INTO FMS VALUES(10, "Ankit", "Bari", 20, 787000098, "Dahanu",9500);
60 •    INSERT INTO FMS VALUES(20, "Yash", "Kerkar", 19, 457495550, "Nallasopara",9000);
61 •    INSERT INTO FMS VALUES(30, "Komal", "Sapatale", 20, 787034546, NULL,5000);
62 •    INSERT INTO FMS VALUES(40, "Mihir", "Dhuri", 18, 787034286, "Kandivali");
63
```

```
158 •    SELECT * FROM Farmer_Farm_View;
159
160 •    CREATE TABLE IF NOT EXISTS FARMER_LOG (
161          LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
162          F_ID INT,
163          ACTION VARCHAR(50),
164          LOG_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
165          FOREIGN KEY (F_ID) REFERENCES FMS(F_ID)
166      );
167
168      DELIMITER //
169
170      DELIMITER //
171
172 •    CREATE TRIGGER fms_insert_trigger AFTER INSERT ON FMS
173      FOR EACH ROW
174      BEGIN
175          INSERT INTO FARMER_LOG (F_ID, ACTION)
176          VALUES (NEW.F_ID, 'Inserted new farmer');
177      END;
178      //
179
180      DELIMITER ;
181
182 •    DROP TRIGGER IF EXISTS fms_insert_trigger;
183
184 •    SELECT * FROM FARMER_LOG;
185
186
```

Output:

Conclusion:

In this database implementation, we have utilized SQL views and triggers to enhance data management and automation.

1. Views: We created a view named "Farmer_Farm_View" to provide a consolidated perspective of farmer details along with their associated farms. This view simplifies querying by presenting relevant information from the FMS and FARM tables in a single virtual table.

2. Triggers: A trigger named "Update_Loan_Amount" was implemented to automatically update the loan amount in the FMS table after each insertion into the SUPPLIERS table. This trigger calculates the total loan amount based on the prices of tools inserted and updates the loan column accordingly, ensuring data accuracy and consistency.

1. Brief about the benefits for using views and triggers.

Ans. Using views and triggers in a database system offers several benefits:

Views:

Simplified Data Access: Views allow users to access data from multiple tables in a simplified and structured manner. They can present complex queries as virtual tables, making it easier for users to retrieve the desired information without needing to understand the underlying database schema.

Enhanced Security: Views can restrict access to sensitive data by exposing only the necessary information to users or applications. They provide a layer of abstraction, allowing administrators to control which columns or rows are accessible to different users or user roles.

Data Consistency: Views can enforce business rules or data integrity constraints by filtering or transforming data before it is presented to users. This ensures that users always view consistent and accurate data, regardless of the underlying changes in the database.

Triggers:

Automated Actions: Triggers can automate repetitive tasks or enforce business logic by automatically executing predefined actions in response to specified database events (e.g., insert, update, delete). This reduces manual intervention and ensures consistent data management practices.

Data Integrity: Triggers can enforce referential integrity, data validation rules, or data consistency checks, preventing invalid or inconsistent data modifications. They can roll back transactions that violate constraints, maintaining the integrity and reliability of the database.

Auditing and Logging: Triggers can capture changes made to the database and log them for auditing purposes. They can record details such as who made the change, when it occurred, and what data was affected, providing a comprehensive audit trail for accountability and compliance purposes.

2. Explain different strategies to update views

Ans. Here are different strategies for updating views:

Simple Views with Single Base Table:

- For views that are not directly updatable due to complex joins, aggregations, or calculations, INSTEAD OF triggers can be used. These triggers intercept INSERT, UPDATE, and DELETE operations on the view and specify custom logic to handle these operations. The trigger logic can translate the requested operation into appropriate modifications on the underlying base tables.

Materialized Views with Periodic Refresh:

- Materialized views store the results of a query physically in the database, allowing for faster access and reducing the need for expensive computations. While materialized views are not typically updated directly, they can be refreshed periodically to synchronize their data with changes in the underlying base tables. Refresh strategies include full refresh (recomputing the entire view), fast refresh (applying incremental changes), and on-demand refresh triggered by specific events.

View Maintenance Scripts:

- For views that are rarely updated or where the update logic is complex, view maintenance scripts can be used. These scripts are manually written to perform the necessary data modifications on the underlying tables based on the requirements of the view. While this approach provides flexibility, it requires careful implementation and maintenance to ensure data consistency.

Immutable Views:

- In some cases, views may represent read-only or derived data that should not be modified directly. In such scenarios, the view definition can explicitly specify the view as non-updatable. This prevents users from attempting to update the view and ensures data integrity by enforcing a separation between the view and the underlying base tables.