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

Conclusion:

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

Ans Using views and triggers in databases provides numerous advantages:

Views:

1. Data Abstraction: Simplify data access by presenting a subset of information from one or more tables.

2. Data Security: Control access to sensitive data, offering a layer of security by limiting what users can see.

3. Performance Optimization: Improve query performance by storing frequently used complex operations.

4. Consistency: Ensure data consistency by centralizing logic and business rules.

5. Encapsulation: Hide complexity by encapsulating complex SQL queries, offering a simpler interface for users.

Triggers:

1. Enforce Data Integrity: Automatically enforce constraints on data to maintain integrity.

2. Audit Trail: Record changes made to the database for auditing purposes.

3. Complex Business Logic: Implement complex business logic not achievable with standard SQL operations.

4. Cascade Updates: Automatically propagate changes to related tables to maintain referential integrity.

5. Data Replication: Facilitate data replication between databases for consistency across distributed systems.

2. Explain different strategies to update views

Ans

Different strategies to update views include:

1. Simple Update: Directly updating the underlying tables that the view is based on. This strategy is straightforward but may not always be feasible if the view involves complex joins or aggregates.

2. Update Through Triggers: Employing INSTEAD OF triggers to intercept updates to the view and translate them into corresponding updates to the underlying tables. This allows for more flexibility and control over how updates are handled.

3. Updateable Views with Joins: Creating updateable views that involve simple joins and have a one-to-one correspondence with a single underlying table. Updates to such views can be propagated directly to the underlying table.

4. Partitioned Views: Using partitioned views to break down updates into smaller, more manageable pieces by dividing the data into separate tables or views based on specific criteria. Updates can then be applied to individual partitions as needed.

5. Materialized Views: Maintaining materialized views that store precomputed results of complex queries, allowing for efficient updates by refreshing the view periodically or incrementally.

6. Dynamic SQL: Dynamically generating SQL statements to update the underlying tables based on the current state of the view. This approach requires careful handling to ensure correctness and security. Each strategy has its advantages and limitations, and the choice depends on factors such as the complexity of the view, performance requirements, and the level of control needed over update