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:

- Create View Syntax:
  CREATE VIEW atm_transactions AS
  SELECT t_id, atm_id, account_id, transaction_type, status, t_time, t_date
  FROM transaction;

- Update View Syntax:
  CREATE OR REPLACE VIEW atm_transactions AS
  SELECT t_id, atm_id, account_id, transaction_type, status, t_time, t_date
  FROM transaction
  WHERE status = ‘completed’;

- Drop View Syntax:
  DROP VIEW atm_transactions;

- Create Trigger Syntax:

```
CREATE TRIGGER update_balance_after_transaction
AFTER INSERT ON transaction
FOR EACH ROW
BEGIN
   IF NEW.transaction_type = ‘debit’ THEN
      UPDATE account SET balance = balance – NEW.amount WHERE account_id
= NEW.account_id;
   ELSEIF NEW.transaction_type = ‘credit’ THEN
      UPDATE account SET balance = balance + NEW.amount WHERE account_id
= NEW.account_id;
   END IF;
END;
```

Conclusion:

1. Brief about the benefits for using views and triggers.
   Views and triggers offer several benefits in database management systems:
   Views:
   Data Abstraction: Views simplify complex queries or joins, presenting users with a consolidated virtual table. This abstraction shields users from underlying schema complexities, enhancing usability.
   Data Security: Views restrict access to sensitive data by exposing only necessary columns or rows based on user roles or permissions. This feature helps enforce data security principles.
   Performance Optimization: Views improve query performance by pre-computing and caching query results, particularly for frequently used or complex queries. They allow administrators to optimize query execution plans and reduce repetitive query overhead.
   Simplified Querying: Views encapsulate complex SQL logic into reusable components, promoting code reusability and maintainability. Centralizing common query logic reduces code duplication across applications.
   Logical Data Independence: Views abstract the physical database schema from application logic, enabling schema changes without impacting applications using the views. This enhances system flexibility and adaptability.
   Triggers:
   Data Integrity Enforcement: Triggers enforce data integrity constraints, such as referential integrity or business rules, by executing custom logic before or after specific database operations.
   Audit Trail: Triggers maintain an audit trail of database changes, logging information about modifications to sensitive data. This enhances accountability and facilitates compliance.
   Complex Business Logic: Triggers execute complex business logic or validation rules within the database, ensuring consistent application of rules across transactions.

Data Synchronization: Triggers synchronize data across multiple tables or databases, automatically propagating changes made to one table to related tables or databases. This ensures data consistency in distributed environments.
Event-Driven Processing: Triggers automate event-driven processing, triggering actions in response to specific database events without manual intervention. This enhances automation and efficiency.

2. Explain different strategies to update views.
   There are several strategies to update views, depending on the complexity of the view and the database management system (DBMS) being used. Here are some common strategies:
   Simple View Updates:
   For simple views that are based on a single table and contain all columns from that table, you can update the view directly as if it were a regular table. The DBMS will handle the update by modifying the underlying table.
   Updatable Views:
   Some views are explicitly defined as updatable by the DBMS. These views meet specific criteria, such as having a simple SELECT statement, not containing certain SQL constructs like GROUP BY or DISTINCT, and having unique key constraints defined. You can update these views directly, and the changes will be propagated to the underlying tables.
   Instead Of Triggers:
   Instead Of triggers allow you to intercept DML (Data Manipulation Language) operations (INSERT, UPDATE, DELETE) on a view and perform custom logic instead. With Instead Of triggers, you can define how the view should be updated, including modifying the underlying tables or rejecting the update altogether based on certain conditions.
   Manual Update of Underlying Tables:
   For more complex views that do not meet the criteria for updatable views or cannot be updated directly, you may need to update the underlying tables directly. In this case, you analyze the view definition to understand which tables are involved and perform the necessary updates on those tables.
   Materialized Views:
   Materialized views are physical copies of query results stored as tables. While they are not updated automatically, you can refresh them periodically to synchronize them with the underlying data. After refreshing, you can query and update the materialized view like a regular table.
   Recreating Views:
   In some cases, particularly if the underlying schema or data model has changed significantly, it may be simpler to drop and recreate the view with the updated definition rather than trying to update it directly.