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]

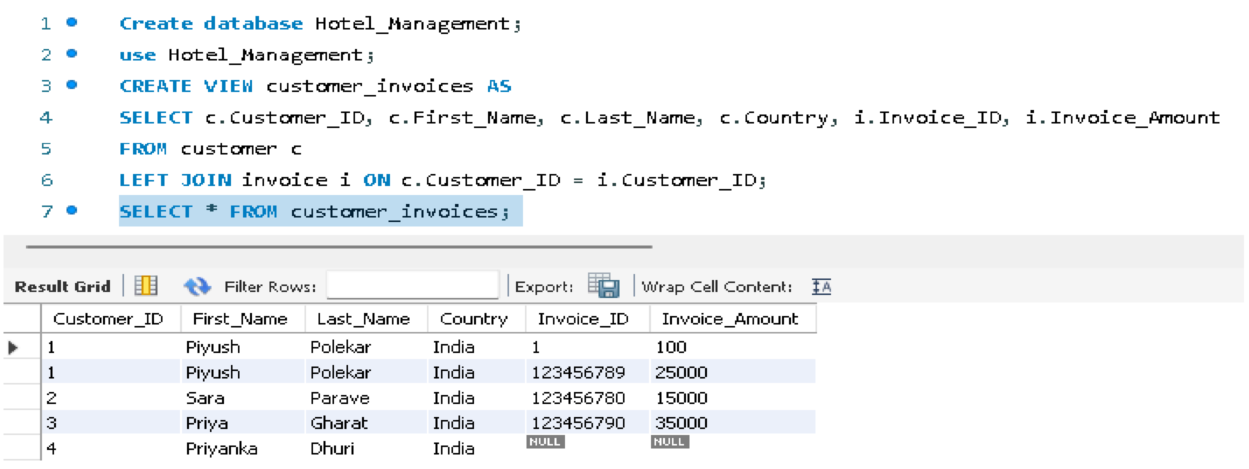
[trigger_body]

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:

VIEWS:-

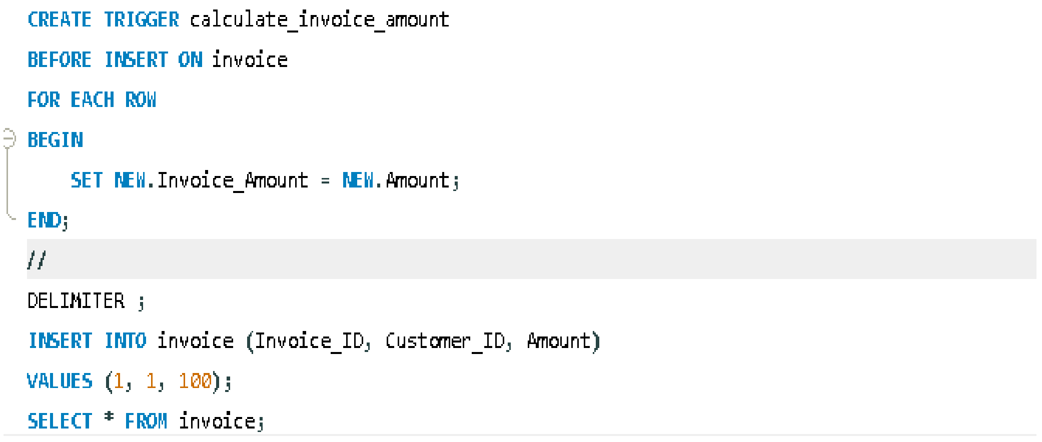


```sql
Create database Hotel_Management;
use Hotel_Management;
CREATE VIEW customer_invoices AS
SELECT c.Customer_ID, c.First_Name, c.Last_Name, c.Country, i.Invoice_ID, i.Invoice_Amount
FROM customer c
LEFT JOIN invoice i ON c.Customer_ID = i.Customer_ID;
SELECT * FROM customer_invoices;
```

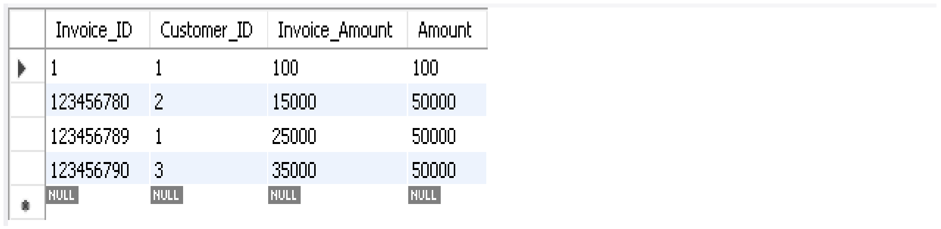
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_ID	First_Name	Last_Name	Country	Invoice_ID	Invoice_Amount
1	Piyush	Polekar	India	1	100
1	Piyush	Polekar	India	123456789	25000
2	Sara	Parave	India	123456780	15000
3	Priya	Gharat	India	123456790	35000
4	Priyanka	Dhuri	India	NULL	NULL

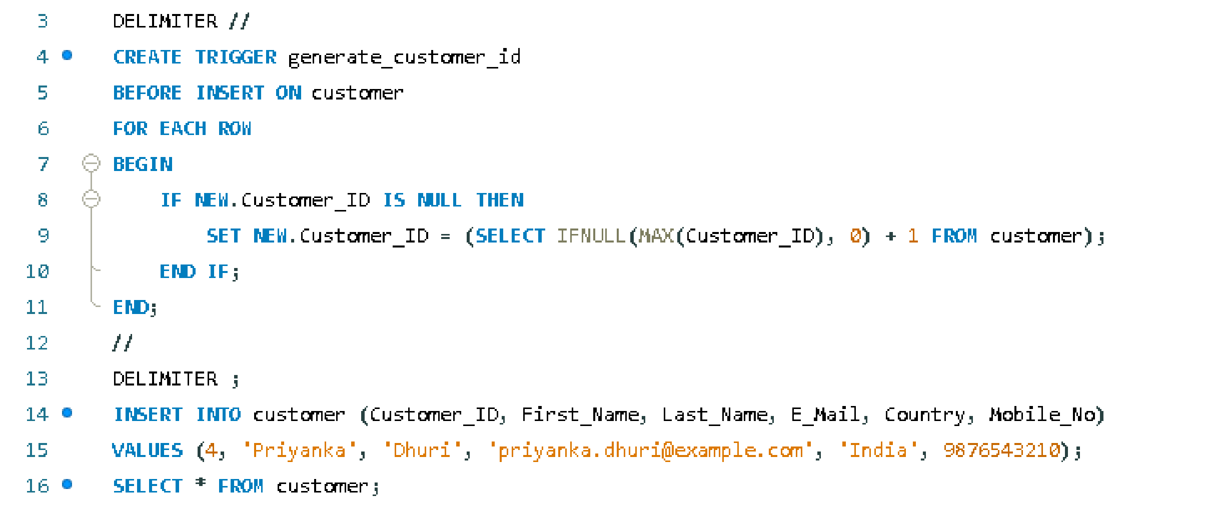
TRIGGERS:-



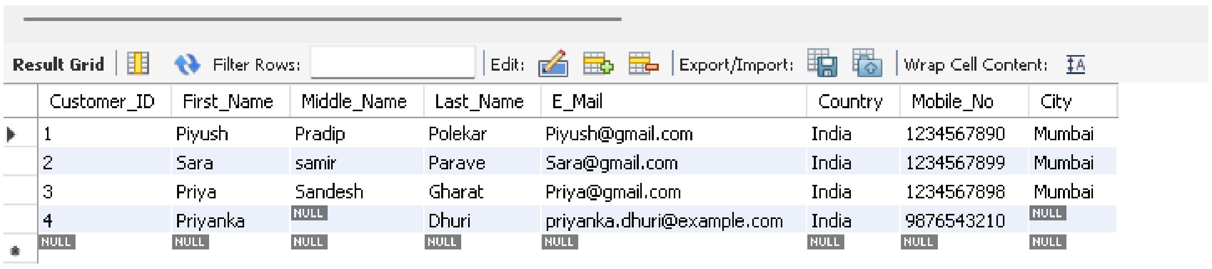
```
CREATE TRIGGER calculate_invoice_amount
BEFORE INSERT ON invoice
FOR EACH ROW
BEGIN
    SET NEW.Invoice_Amount = NEW.Amount;
END;
//
DELIMITER ;
INSERT INTO invoice (Invoice_ID, Customer_ID, Amount)
VALUES (1, 1, 100);
SELECT * FROM invoice;
```



	Invoice_ID	Customer_ID	Invoice_Amount	Amount
▶	1	1	100	100
	123456780	2	15000	50000
	123456789	1	25000	50000
	123456790	3	35000	50000
*	NULL	NULL	NULL	NULL



```
DELIMITER //
CREATE TRIGGER generate_customer_id
BEFORE INSERT ON customer
FOR EACH ROW
BEGIN
    IF NEW.Customer_ID IS NULL THEN
        SET NEW.Customer_ID = (SELECT IFNULL(MAX(Customer_ID), 0) + 1 FROM customer);
    END IF;
END;
//
DELIMITER ;
INSERT INTO customer (Customer_ID, First_Name, Last_Name, E_Mail, Country, Mobile_No)
VALUES (4, 'Priyanka', 'Dhuri', 'priyanka.dhuri@example.com', 'India', 9876543210);
SELECT * FROM customer;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	Customer_ID	First_Name	Middle_Name	Last_Name	E_Mail	Country	Mobile_No	City
▶	1	Piyush	Pradip	Polekar	Piyush@gmail.com	India	1234567890	Mumbai
	2	Sara	samir	Parave	Sara@gmail.com	India	1234567899	Mumbai
	3	Priya	Sandesh	Gharat	Priya@gmail.com	India	1234567898	Mumbai
	4	Priyanka	NULL	Dhuri	priyanka.dhuri@example.com	India	9876543210	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusion:

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

Views:

a) Simplified Data Access**: Views allow users to access and retrieve data from multiple tables using a single, simplified interface. This can enhance productivity by reducing the complexity of queries.

b) Data Security**: Views can be used to restrict access to specific columns or rows of data, providing a layer of security by hiding sensitive information from unauthorized users.

c) Data Abstraction**: Views abstract the underlying structure of the database, allowing changes to the database schema without affecting applications that rely on the views. This enhances flexibility and maintainability.

d) Performance Optimization**: Views can be precomputed and cached, improving query performance by reducing the need for repetitive joins or complex calculations.

Triggers:

a) Data Integrity Enforcement**: Triggers can enforce data integrity rules by automatically performing actions, such as validation checks or data modifications, before or after data manipulation operations (INSERT, UPDATE, DELETE).

b) Business Logic Implementation**: Triggers can implement complex business logic directly in the database, ensuring consistency and enforcing business rules across applications.

c) Auditing and Logging**: Triggers can be used to log changes made to the database, providing an audit trail of data modifications for compliance or troubleshooting purposes.

d) Data Synchronization**: Triggers can synchronize data between tables or databases, ensuring consistency and maintaining data integrity across distributed systems.

2) Explain different strategies to update views

a) Simple Views:
In some database systems, you can update a view if it is based on a single table and does not contain any aggregate functions or grouping.
The update is directly applied to the underlying table(s) that the view references.

b) Updatable Views:
Some views are explicitly designed to be updatable, allowing you to update data through the view itself.

These views must meet certain criteria, such as:
Contain all columns from a single base table.

Not include aggregate functions, GROUP BY clauses, or DISTINCT.
Not contain joins, subqueries in the SELECT list, or set operations (UNION, INTERSECT, EXCEPT).

c) Instead Of Triggers:
Instead Of triggers are used to enable updates on non-updatable views by intercepting the attempted update operations and providing custom logic to handle them.
These triggers can be defined to execute custom insert, update, or delete operations on the underlying tables when the view is modified.

d) Materialized Views:
Materialized views are physical copies of query results stored as tables.
They can be updated by refreshing the materialized view, which recalculates the data based on the underlying tables.
The refresh operation can be performed manually or scheduled to occur automatically at specified intervals.