|  |  |
| --- | --- |
| **Name** | Adwait S Purao |
| **UID no.** | 2021300101 |
| **Experiment No.** | 7 |

|  |  |
| --- | --- |
| **AIM:** | To create and query views in MySQL |
| **Program 1** | |
| **PROBLEM STATEMENT :** | To create various views and perform queries on them in MySQL |
| **Theory :** | **Views**  View is a data object which does not contain any data. Contents of the view are the resultant of a base table. They are operated just like base table but they don’t contain any data of their own. The difference between a view and a table is that views are definitions built on top of other tables (or views). If data is changed in the underlying table, the same change is reflected in the view. A view can be built on top of a single or multiple tables.  **Why views?**  Views can be effective copies of base tables.  Views can have column names and expressions.  You can use any clauses in views.  Views can be used in INSERT/UPDATE/DELETE.  Views can contain expressions in the select list.  Views can be views of views.  **Restrictions on View definition**  The SELECT statement cannot contain a subquery in the FROM clause.  The SELECT statement cannot refer to system or user variables.  Within a stored program, the definition cannot refer to program parameters or local variables.  The SELECT statement cannot refer to prepared statement parameters.  Any table or view referred to in the definition must exist.  The definition cannot refer to a TEMPORARY table, and you cannot create a TEMPORARY view.  Any tables named in the view definition must exist at definition time.  You cannot associate a trigger with a view.  Aliases for column names in the SELECT statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).  **SQL CREATE VIEW Statement**  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.  **CREATE VIEW Syntax**    Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.  **SQL Updating a View**  A view can be updated with the CREATE OR REPLACE VIEW statement.  **SQL CREATE OR REPLACE VIEW Syntax**    **SQL Dropping a View**  A view is deleted with the DROP VIEW statement.  **SQL DROP VIEW Syntax**    **SQL Inserting into a View**  We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View  **SQL Inserting into a View Syntax** |
| **Queries** | **Query 1: Creation of a view**  **1)**  **Statement: A view of customer table is created**  **Code:**  **CREATE VIEW cust\_view AS**  **SELECT C\_ID , Reservation\_no**  **FROM customer ;**  **Original Table:**    **Output:**    **2)**  **Statement: A view of Reservation table is created**  **Code:**  **CREATE VIEW reser\_view AS**  **SELECT C\_ID , Reservation\_no , R\_no**  **FROM customer ;**  **Original Table:**    **Output:**    **Query 2: Creating a view with natural join**  **Statement: A new view is created as a natural join of two views**  **Code:**  **create view cust1\_view**  **as select c\_id , reservation\_no,R\_no**  **from reservation natural join customer;**  **Original Table:**  **Reservation table**    **Customer table**    **Output:**    **Query 3: Cross join on views employee and hotel**  **Statement:A new view employ\_view is created by the cross join of employee and hotel table**  **Code:**  **create view employ\_view**  **as select e\_id,e\_name,e\_type**  **from employee cross join hotel\_info;**  **Original view:**  **Employee Table**    **Hotel table**    **Output:**    **Query 4: Dropping a view**  **Statement:**  **A view is completely deleted**  **Code:**  **drop view cust1\_view;**  **Original view:**    **Output:**    **Query 5: Order by in views**  **Statement:**  **Ordering the values in the view as per the price**  **Code:**  **CREATE VIEW room\_view AS**  **SELECT r\_no,R\_price,R\_type**  **FROM room**  **ORDER BY R\_price;**  **Original Table:**    **Output:**    **Query 6: Updating a view**  **Statement: A new view is created with the following parameters**  **Code:**  **CREATE OR REPLACE VIEW emp\_view AS**  **SELECT e\_name,e\_type, address, city ,e\_salary**  **FROM employee**  **WHERE E\_Salary>35000;**  **Original Table:(Employee Table)**    **Output:**    **Query 7: Deleting from a view**  **Statement: Deleting entry of Ranbir Kapoor from emp\_view table**  **Code:**  **DELETE FROM emp\_view**  **WHERE E\_NAME="Ranbir Kapoor";**  **Original Table:(Emp\_view table)**    **Output:**    **Query 8: Updating a view**  **Statement:**  **Updating the salary from 50000 to 100000**  **Code:**  **UPDATE emp\_view**  **SET e\_salary=100000**  **WHERE e\_name = 'Angelina Jolie';**  **Original Table:**    **Output:**    **Query 9: Querying hotel\_view**  **Statement:**  **Selecting hotels having vacancies more than 5**  **Code:**  **Creation hotel\_view**  **CREATE VIEW hotel\_view AS**  **SELECT h\_name,h\_vacancies, h\_address**  **FROM hotel\_info;**  **Query:**  **select \* from hotel\_view where h\_vacancies>7;**  **Original Table:(hotel\_view)**    **Output:**    **Query 10: (Querying room\_view)**  **Statement: Selecting room type as basic**  **Code:**  **select \* from room\_view where r\_type="Basic";**  **Original Table:**    **Output:** |
| **Conclusion**  **In this experiment we learnt that significance of views , we learnt that views can help the user if the same group of tables are accessed continuously . We learnt and implemented various functions on views like CREATE VIEW , REPLACE VIEW , DROP VIEW , INSERT INTO VIEW .** | |