EX NO:1 CREATE A DATABASE TABLE AND MANIPULATE DATA USING SQL DDL AND DML COMMANDS

AIM:

To write SQL query to create a database table and manipulate data using sql ddl and dml commands.

ALGORITHM:

1. CONNECT TO THE DATABASE SERVER.
2. CREATE A NEW DATABASE.
3. SWITCH TO THE NEWLY CREATED DATABASE.
4. DEFINE THE TABLE SCHEMA, INCLUDING COLUMN NAMES, DATA TYPES, AND CONSTRAINTS (PRIMARY KEY, UNIQUE, CHECK, NOT NULL).
5. CREATE THE TABLE USING THE CREATE TABLE SQL COMMAND.
6. INSERT ROWS INTO THE TABLE USING THE INSERT INTO SQL COMMAND.
7. UPDATE ROWS IN THE TABLE USING THE UPDATE SQL COMMAND.
8. DELETE ROWS FROM THE TABLE USING THE DELETE SQL COMMAND.
9. USE THE SELECT SQL COMMAND TO RETRIEVE DATA FROM THE TABLE.
10. CLOSE THE DATABASE CONNECTION.

RESULT:

Thus the SQL query to create a database table and manipulate data using sql ddl and dml commands has been written and executed successfully.

EX NO:2 CREATE A SET OF TABLES, ADD FOREIGN KEY CONSTRAINTS AND INCORPORATE REFERENTIAL INTEGRITY

AIM:

To write SQL query to create a set of tables, add foreign key constraints and incorporate referential integrity.

ALGORITHM:

1. Connect to the database server.
2. Create a new database.
3. Switch to the newly created database.
4. Define the table schema for each table, including column names, data types, and constraints.
5. Create each table using the CREATE TABLE SQL command.
6. Add foreign key constraints to the appropriate columns in each table using the ALTER TABLE SQL command.
7. Incorporate referential integrity by ensuring that foreign key values match the primary key values in the referenced table.
8. Close the database connection.

RESULT:

Thus the SQL query to create a set of tables, add foreign key constraints and incorporate referential integrity has been written and executed successfully.

EX NO:3 QUERY DATABASE TABLES USING WHERE CLAUSE CONDITIONS AND AGGREGATE FUNCTIONS

AIM:

To write SQL query to query database tables using where clause conditions and aggregate functions.

ALGORITHM:

1. Connect to the database server.
2. Select the database to be used.
3. Use the SELECT SQL command to query the database tables.
4. Specify the columns to be returned.
5. Use the WHERE clause to specify the condition(s) for selecting the rows to be returned.
6. Use the ORDER BY clause to sort the returned rows based on a specific column.
7. Use aggregate functions to perform calculations on selected columns.
8. Close the database connection.

RESULT:

Thus the SQL query to query database tables using where clause conditions and aggregate functions has been written and executed successfully.

EX NO:4 QUERY DATABASE TABLES USING SUBQUERIES AND SIMPLE JOIN OPERATIONS

AIM:

To write the query database tables using subqueries and simple join operations in SQL.

ALGORITHM:

1. Connect to the database server.
2. Select the database to be used.
3. Use the SELECT SQL command to query the database tables.
4. Specify the columns to be returned.
5. Use the WHERE clause to filter the rows based on a specific condition.
6. Use subqueries to perform additional calculations on selected columns.
7. Use simple join operations to combine data from multiple tables.
8. Close the database connection.

RESULT:

Thus the query database tables using subqueries and simple join operations in SQL has been written and executed successfully.

EX NO:5 QUERY DATABASE TABLES USING NATURAL, EQUI AND OUTER JOINS

AIM:

To write the query database tables using natural, equi and outer joins.

ALGORITHM:

1. Connect to the database server.
2. Select the database to be used.
3. Use the SELECT SQL command to query the database tables.
4. Specify the columns to be returned.
5. Use the FROM clause to specify the tables to be joined.
6. Use natural, equi, or outer joins to combine data from multiple tables.
7. Use the WHERE clause to filter the rows based on a specific condition.
8. Close the database connection.

RESULT:

Thus the query database tables using natural, equi and outer joins has been written and executed successfully.

EX NO:6 WRITE USER DEFINED FUNCTIONS AND STORED PROCEDURES IN SQL.

AIM;

To write the user defined functions and stored procedures in sql.

ALGORITHM:

1. Connect to the database server.
2. Select the database to be used.
3. Define the function or procedure using the CREATE FUNCTION or CREATE PROCEDURE SQL command.
4. Specify the input parameters and return value (for functions).
5. Write the SQL code to be executed in the function or procedure.
6. Save the function or procedure in the database.
7. Call the function or procedure using the EXECUTE SQL command.
8. Close the database connection.

RESULT:

Thus the user defined functions and stored procedures in SQL has been written and executed successfully.

EX NO:7 EXECUTE COMPLEX TRANSACTIONS AND REALIZE DCL AND TCL COMMANDS.

AIM:

To write the query to execute complex transactions and realize dcl and tcl commands.

ALGORITHM:

1. Connect to the database server.
2. Select the database to be used.
3. Begin a transaction using the BEGIN TRANSACTION SQL command.
4. Execute multiple SQL commands to modify the database.
5. Use the SAVEPOINT SQL command to create a savepoint within the transaction.
6. Use the ROLLBACK TO SAVEPOINT SQL command to rollback to a specific savepoint within the transaction.
7. Use the COMMIT SQL command to commit the transaction and make the changes permanent.
8. Use the ROLLBACK SQL command to rollback the entire transaction and undo all changes.
9. Use the GRANT SQL command to grant privileges to a user or group.
10. Use the REVOKE SQL command to revoke privileges from a user or group.
11. Close the database connection.

RESULT:

Thus the query to execute complex transactions and realize dcl and tcl commands has been written and executed successfully.

EX NO:8 WRITE SQL TRIGGERS FOR INSERT, DELETE, AND UPDATE OPERATIONS IN A DATABASE TABLE.

AIM:

To write sql triggers for insert, delete, and update operations in a database table..

ALGORITHM:

INSERT OPERATION:

1. Start a transaction to ensure data consistency.
2. Check if the data being inserted is valid and does not violate any constraints or rules defined for the table.
3. If the data is valid, insert the data into the table.
4. Commit the transaction to make the changes permanent.
5. If there is any error during the process, rollback the transaction to undo any changes made.

DELETE OPERATION:

1. Start a transaction to ensure data consistency.
2. Check if the data being deleted exists in the table.
3. If the data exists, delete it from the table.
4. Commit the transaction to make the changes permanent.
5. If there is any error during the process, rollback the transaction to undo any changes made.

UPDATE OPERATION:

1. Start a transaction to ensure data consistency.
2. Check if the data being updated exists in the table.
3. If the data exists, check if the new data being updated is valid and does not violate any constraints or rules defined for the table.
4. If the data is valid, update the data in the table.
5. Commit the transaction to make the changes permanent.
6. If there is any error during the process, rollback the transaction to undo any changes made.

RESULT:

Thus the SQL striggers for insert, delete, and update operations in a database table has been written and executed successfully.

EX NO: 9 Create View and index for database tables with a large number of records

AIM:

To write the query to create view and index for database tables with a large number of records.

ALGORITHM:

Creating a View:

1. Identify the query that retrieves the data you want to view.
2. Write the SQL statement that creates the view. For example: **CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;**
3. Execute the SQL statement to create the view.
4. Use the view in place of the original query in your application code.

Creating an Index:

1. Identify the column or columns that are frequently used in queries and have a large number of unique values.
2. Write the SQL statement that creates the index. For example: **CREATE INDEX index_name ON table_name (column1, column2);**
3. Execute the SQL statement to create the index.
4. Use the indexed columns in your queries to improve performance.

RESULT:

Thus the query to create view and index for database tables with a large number of records has been written and executed successfully.

EX NO:10 CREATE AN XML DATABASE AND VALIDATE IT USING XML SCHEMA.

AIM:

To create an XML database and validate it using XML schema.

ALGORITHM:

1. Define the data structure for your XML database using an XML schema. You can create an XML schema using a text editor, or a graphical tool such as XMLSpy or Oxygen XML Editor.
2. Save the XML schema file with a .xsd extension.
3. Create a new XML file and enter data based on the structure defined in the XML schema.
4. Save the XML file with a .xml extension.
5. Validate the XML file against the XML schema using a validation tool or XML editor that supports XML schema validation.
6. If there are any errors or warnings, correct them in the XML file and repeat the validation process until the file passes validation.
7. Once the XML file is validated, you can store it in your XML database.

RESULT:

Thus the Create an XML database and validate it using XML schema has been created and validated.

EX NO: 11 CREATE DOCUMENT, COLUMN AND GRAPH BASED DATA USING NOSQL DATABASE TOOLS

AIM:

To create document, column and graph based data using NOSQL database tools

ALGORITHM:

1. Choose a NoSQL database tool that supports the type of data model you need (document, column, or graph). Examples of popular NoSQL database tools include MongoDB, Cassandra, and Neo4j.
2. Install and configure the NoSQL database tool according to the instructions provided by the vendor. This typically involves setting up a database instance, defining security settings, and configuring the database for optimal performance.
3. For document-based data, create a database and collection within the NoSQL database tool. Define the structure of your documents using a schema or by allowing the database to create a flexible schema on-the-fly. Insert documents into the collection using the tool's API or a client application.
4. For column-based data, create a keyspace and column family within the NoSQL database tool. Define the structure of your columns using a schema or by allowing the database to create a flexible schema on-the-fly. Insert columns into the column family using the tool's API or a client application.
5. For graph-based data, create a graph within the NoSQL database tool. Define the structure of your nodes and edges using a schema or by allowing the database to create a flexible schema on-the-fly. Insert nodes and edges into the graph using the tool's API or a client application.
6. Create indexes and views to optimize queries against your data. NoSQL databases typically provide flexible indexing options that allow you to optimize queries based on the type of data you are working with.
7. Use the tool's API or client applications to retrieve and manipulate data stored in the NoSQL database. The API will vary depending on the specific NoSQL database tool you are using.
8. Periodically backup your NoSQL database to ensure data is protected in case of system failure or other issues.
9. Monitor performance of the NoSQL database tool to ensure it is performing optimally. NoSQL databases can be highly scalable and can handle large amounts of data, but it is important to monitor performance to ensure the database remains responsive as data volumes grow.

RESULT: Thus the create document, column and graph based data using NOSQL database tools has been created and executed successfully.

EX NO: 12 DEVELOP A SIMPLE GUI BASED DATABASE APPLICATION AND INCORPORATE ALL THE ABOVE-MENTIONED FEATURES

AIM:

To develop a simple GUI based database application and incorporate all the above-mentioned features.

ALGORITHM:

1. Choose a programming language and GUI framework to develop your application. Popular choices include Java with Swing or JavaFX, Python with PyQt or Tkinter, and C# with Windows Forms or WPF.
2. Choose a NoSQL database tool that supports the type of data model you need (document, column, or graph). Examples of popular NoSQL database tools include MongoDB for document-based data, Apache Cassandra for column-based data, and Neo4j for graph-based data.
3. Install and configure the NoSQL database tool according to the instructions provided by the vendor. This typically involves setting up a database instance, defining security settings, and configuring the database for optimal performance.
4. Design the GUI for your application using the GUI framework of your choice. Your GUI should allow users to view, add, update, and delete data stored in the NoSQL database.
5. Implement code to connect to the NoSQL database from your application. This code will vary depending on the specific NoSQL database tool you are using.
6. Implement code to retrieve and display data from the NoSQL database in the GUI. This code will also vary depending on the specific NoSQL database tool you are using and the type of data model (document, column, or graph-based) you are working with.
7. Implement code to add, update, and delete data in the NoSQL database from the GUI. This code will also vary depending on the specific NoSQL database tool you are using and the type of data model you are working with.
8. Implement indexing and view options in your application to optimize queries against the data stored in the NoSQL database. This will involve using the indexing features provided by your NoSQL database tool.
9. Test your application to ensure it is functioning properly and data is being stored and retrieved correctly from the NoSQL database.

By following these steps, you can develop a simple GUI-based database application that incorporates document, column, and graph-based data models. The specific steps and tools used will depend on the requirements of your project and the specific NoSQL database tool being used.

RESULT:

Thus to Develop a simple GUI based database application and incorporate all the above-mentioned features has been written and executed successfully.