SQL ASSIGNMENT (PART –2)

1. Explain the execution flow of SQL commands in details . Discuss why SQL commands follow a specific sequence and the purpose behind this execution order.

Sol : The Execution flow of SQL commands are :

* **FROM or JOIN:** Specifies the tables from which to retrieve data.
* **WHERE**: Filters the rows that meet the condition before grouping.
* **GROUP BY**: Groups rows that share a property.
* **HAVING**: Filters groups based on conditions, applied after grouping.
* **SELECT**: Specifies the columns to retrieve or calculate.
* **DISTINCT**: Removes duplicate rows from the result set.
* **ORDER BY**: Sorts the result set by specified columns.
* **LIMIT**: Specifies the maximum number of rows to return.
* **OFFSET**: Specifies how many rows to skip before starting to return rows.

SQL commands follow a specific sequence because of logical dependencies and performance considerations that ensure queries are processed correctly, efficiently. The purpose behind the SQL execution order is to ensure logical accuracy, data integrity, and query optimization. SQL is designed to handle large datasets systematically, and the specific sequence allows queries to be executed in a way that aligns with these goals.

2. Write Five SQL queries demonstrating Data Definition Language (DDL) operations.

Sol : Here are five SQL queries that are demonstrating DDL commands :-

* **CREATE :** This Command can be used to create Database or to create a Table.

-- For Creating Database---

CREATE DATABASE E\_Commerce;

USE E\_Commerce;

-- Creating Table Customers ---

CREATE TABLE Customers(CustomerID INT PRIMARY KEY,

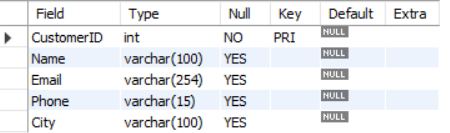
Name VARCHAR(100),

Email VARCHAR(254),

Phone VARCHAR(15),

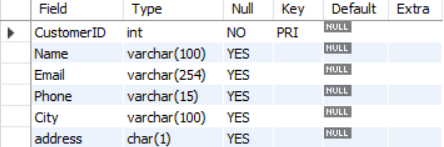
City VARCHAR(100)  );

DESC Customers;



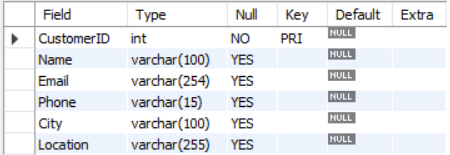
* **ALTER :** This Command is used for alteration or say modification. It can be used to add a column, to rename any existing column, to change the datatype of any column , to drop a column.
* ALTER table Customers add(address char);

Desc Customers;



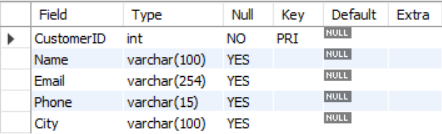
* ALTER TABLE Customers CHANGE address Location VARCHAR(255);

Desc Customers;



* ALTER TABLE Customers DROP COLUMN Location;

Desc Customers;



* **TRUNCATE :** This Command removes all records from a table, but does not destroy the table structure.

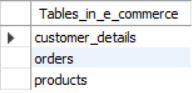
TRUNCATE table Customers;

* **DROP :** This Command can be used to completely removes all records of a table from database or removes database completely. This Command will also destroy the table structure.

DROP table Customers;

* **RENAME :** This Command is used to rename the name of any existing table in a database.

RENAME table Customers to Customer\_details;



3. Provide three SQL queries Illustrating Data Manipulation Language (DML) operations.

Sol : Performing Three SQL DML queries :-

* **INSERT :** This Command is used to insert data into tables or a new row.

-- Inserting Data into Customer Table --

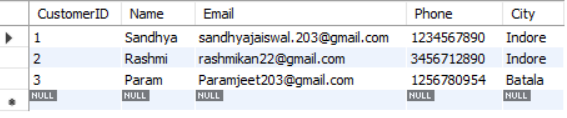
INSERT INTO Customers (CustomerID, Name, Email,Phone,City) VALUES

(1,'Sandhya', 'sandhyajaiswal.203@gmail.com','1234567890','Indore'),

(2,'Rashmi','rashmikan22@gmail.com','3456712890','Indore'),

(3,'Param','Paramjeet203@gmail.com','1256780954','Batala');

SELECT \* FROM Customers;

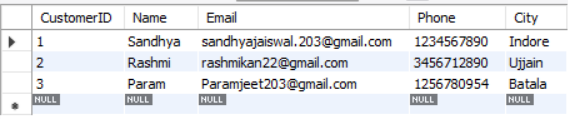


* **UPDATE :** This Command is used to update a existing row of a table.

--- Updating City ---

UPDATE Customers SET City = 'Ujjain' WHERE CustomerID = 2;

SELECT \* FROM Customers;



* **DELETE :** This Command is used to remove one or more records from the table.

DELETE FROM Customers WHERE CustomerID = 2;

4. Create two tables and demonstrate the application of various types of SQL joins including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.

Sol : Creating table **T1** and table **T2**

|  |  |
| --- | --- |
| T1 | T2 |
| 1 | 1 |
| 1 | 1 |
| 1 | 2 |
| 2 | 2 |
| 3 | 4 |
| 3 | NULL |
| 3 |  |

**INNER** **JOIN :** Fetches matching records.

|  |  |
| --- | --- |
| I | J |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 2 | 2 |

No. of inner joins are 8.

**LEFT** **JOIN :** Fetching matching records and fetching any additional records from left table that is not present in right table.

|  |  |
| --- | --- |
| L | J |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 2 | 2 |
| 3 | NULL |
| 3 | NULL |
| 3 | NULL |

No of records are 11.

**RIGHT** **JOIN :** Fetching matching records and fetching any additional records from right table that is not present in left table.

|  |  |
| --- | --- |
| R | J |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 2 | 2 |
| NULL | 4 |
| NULL | NULL |

No of records are 10.

**FULL OUTER** **JOIN :** Fetches records and any additional records from LEFT and RIGHT joins.

|  |  |
| --- | --- |
| F | J |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 2 | 2 |
| 3 |  |
| 3 |  |
| 3 |  |

No of Full join records are 13.

**NATURAL** **JOIN :**  Naturally like Inner Join with same column name but if column name is not same then it performs cross join.

Total Number of Records:-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IJ | LJ | RJ | FJ | NJ | CJ |
| 8 | 11 | 10 | 13 | 8 + 42 | 42 |

5. Compile a comprehensive list of best practices for writing efficient and maintainable SQL queries.

Sol : Following are some best practices we can use foe writing efficient and maintainable

SQL queries :

* Use indexes effectively.
* Avoid SELECT \* and retrieve only necessary columns.
* Optimize JOIN operations.
* Minimize the use of subqueries.
* Avoid redundant or unnecessary data retrieval.
* Utilize stored procedures.
* Consider partitioning and sharding.
* Normalize database tables.