PRACTICAL - 1 INTRODUCTION OF DATA BASE MANAGEMENT SYSTEM

Introduction of DBMS: - A Database Management System (DBMS) is a software application that facilitates the creation, organization, retrieval, and management of data in a database. It serves as an intermediary between the users and the database, providing an efficient and structured way to interact with and manipulate data. The primary goal of a DBMS is to ensure the integrity, security, and availability of data while providing a convenient and consistent interface for users and applications**.**

Software of DBMS: -

- **Oracle Database:** Developed by Oracle Corporation, Oracle Database is a robust and widely used relational database management system (RDBMS). It supports SQL and offers features such as high availability, scalability, and advanced security options.

- **Microsoft SQL Server:** Developed by Microsoft, SQL Server is an RDBMS that integrates well with Microsoft's suite of products. It supports Transact-SQL (T-SQL) and provides features such as business intelligence, data warehousing, and advanced analytics.

- **MySQL:** MySQL is an open-source relational database management system renowned for its reliability, performance, and ease of use. Released in 1995, it follows the relational model, organizing data into tables and providing efficient querying capabilities. With cross-platform compatibility, scalability options, and robust security features, MySQL accommodates both small and large-scale applications.

- **PostgreSQL:** PostgreSQL is a powerful, open-source object-relational database system. It is known for its extensibility, support for complex queries, and adherence to SQL standards. PostgreSQL is often used for large-scale and complex database applications.
- **SQLite:** SQLite is a lightweight, embedded database engine that does not require a separate server process. It is suitable for mobile and embedded applications due to its simplicity, small footprint, and ease of integration.
- **IBM Db2:** IBM Db2 is an enterprise-level relational database management system developed by IBM. It is known for its scalability, reliability, and support for various data types. Db2 is commonly used in large enterprises for business-critical applications.
- **MongoDB:** MongoDB is a NoSQL database that uses a document-oriented model. It is designed to handle large volumes of unstructured or semistructured data and is commonly used in scenarios where flexibility and scalability are crucial, such as in web and mobile applications.
- **Redis:** Redis is an open-source, in-memory data structure store that can be used as a database, cache, and message broker. It is known for its high performance and is often used in scenarios requiring fast data access, such as caching and real-time analytics.
- **Couchbase:** Couchbase is a NoSQL, distributed database system that is designed for interactive applications. It supports both key-value and document-oriented data models and is known for its scalability and high performance.
- **Amazon DynamoDB:** DynamoDB is a managed NoSQL database service provided by Amazon Web Services (AWS). It is designed for highperformance applications and is known for its automatic scaling, low-latency access, and seamless integration with other AWS services.

Role of DBMS

- **Data Organization and Storage:** DBMS helps organize and store large amounts of data in a structured manner. It uses tables, rows, and columns to store and retrieve information efficiently.

- **Data Retrieval:** Users can retrieve data from the database using queries. The DBMS provides a query language (e.g., SQL) that allows users to interact with the database and retrieve specific information based on their requirements.

- **Data Integrity and Security:** DBMS ensures data integrity by enforcing data constraints such as unique keys, foreign keys, and check constraints. It helps maintain the accuracy and consistency of data.
  Security features, including authentication and authorization, are implemented to control access to the database and ensure that only authorized users can perform specific operations.

- **Transaction Management:** DBMS supports transactions, which are sequences of one or more operations that are executed as a single unit. It ensures that transactions are either completed successfully or leaves the database in a consistent state if an error occurs.

- **Data Independence:** DBMS provides a layer of abstraction between the physical storage of data and the applications that interact with it. This allows changes in the database structure or organization without affecting the applications that use the data.

- **Backup and Recovery:** DBMS facilitates the creation of backup copies of the database to prevent data loss in case of hardware failures, software

errors, or other unforeseen events. It also supports recovery mechanisms to restore the database to a consistent state after a failure.

- **Data Dictionary Management:** DBMS maintains a data dictionary or metadata repository that contains information about the structure of the database, data types, relationships between tables, and other essential details. This information is valuable for both users and the DBMS itself.
- **Query Optimization:** DBMS optimizes queries to improve the efficiency of data retrieval. It analyzes the query execution plan and chooses the most efficient way to access and retrieve data from the database.

- **Scalability and Performance:** DBMS is designed to handle large volumes of data and provide efficient performance even as the database grows. It includes mechanisms for indexing, caching, and other optimizations to enhance performance.

Need for DBMS

The need for a Database Management System (DBMS) arises from the challenges associated with managing large volumes of data in an organized, efficient, and secure manner. Here are some key reasons why DBMS is essential:

- Data Organization and Management
- Data Security and Privacy
- Data Integrity and Consistency
- Concurrent Data Access
- Data Analysis and Reporting
- Scalability and Flexibility
- Cost-Effectiveness

- **Data Organization and Management:** One of the primary needs for a DBMS is data organization and management. DBMSs allow data to be stored in a structured manner, which helps in easier retrieval and analysis. A well-designed database schema enables faster access to information, reducing the time required to find relevant data. A DBMS also provides features like indexing and searching, which make it easier to locate specific data within the database. This allows organizations to manage their data more efficiently and effectively.

- **Data Security and Privacy:** DBMSs provide a robust security framework that ensures the confidentiality, integrity, and availability of data. They offer authentication and authorization features that control access to the database. DBMSs also provide encryption capabilities to protect sensitive data from unauthorized access. Moreover, DBMSs comply with various data privacy regulations such as the GDPR, HIPAA, and CCPA, ensuring that organizations can store and manage their data in compliance with legal requirements.

- **Data Integrity and Consistency:** Data integrity and consistency are crucial for any database. DBMSs provide mechanisms that ensure the accuracy and consistency of data. These mechanisms include constraints, triggers, and stored procedures that enforce data integrity rules.

- **Concurrent Data Access:** A DBMS provides a concurrent access mechanism that allows multiple users to access the same data simultaneously. This is especially important for organizations that require real-time data access. DBMSs use locking mechanisms to ensure that multiple users can access the same data without causing conflicts or data corruption.

- **Data Analysis and Reporting:** DBMSs provide tools that enable data analysis and reporting. These tools allow organizations to extract useful insights from their data, enabling better decision-making. DBMSs support various data analysis techniques such as OLAP, data mining, and machine learning.

- **Scalability and Flexibility:** DBMSs provide scalability and flexibility, enabling organizations to handle increasing amounts of data. DBMSs can be scaled horizontally by adding more servers or vertically by increasing the capacity of existing servers. This makes it easier for organizations to handle large amounts of data without compromising performance.

- **Cost-Effectiveness:** DBMSs are cost-effective compared to traditional file-based systems. They reduce storage costs by eliminating redundancies and optimizing data storage. They also reduce development costs by providing tools for database design, maintenance, and administration.

INTRODUCTION OF SQL

SQL (Structured Query Language) is a language to operate databases; it includes Database Creation, Database Deletion, Fetching Data Rows, Modifying & Deleting Data rows, etc.

SQL stands for Structured Query Language which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL was developed in the 1970s by IBM Computer Scientists and became a standard of the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987.

SQL is supported by most relational database management systems (RDBMS) such as MySQL, Oracle, SQL Server, and PostgreSQL. With SQL, you can perform a wide range of operations on a database, including creating tables and defining their structure, inserting, updating, and deleting data, querying data to retrieve specific information, and managing database constraints, indexes, and views.

The language consists of multiple components, including Data Definition Language (DDL) statements for creating and modifying database structures, Data Manipulation Language (DML) statements for performing operations on stored data, and Data Control Language (DCL) statements for managing database security and access controls.

SQL offers a powerful and flexible way to interact with databases, making it a fundamental skill for professionals working with data. It allows for efficient data retrieval and manipulation, enabling users to extract meaningful insights and support decision-making processes.

DDL (Data Definition Language)

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete

database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER**: This is used to alter the structure of the database.
- **TRUNCATE**: This is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT: This is used to add comments to the data dictionary. • **RENAME**: This is used to rename an object existing in the database.

DML (Data Manipulation Language)

The SQL commands deal with the manipulation of data present in the database belonging to DML or Data Manipulation Language and this includes most of the SQL statements. It is the part of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

- **INSERT**: It is used to insert data into a table.
- **UPDATE:** It is used to update existing data within a table.
- **DELETE:** It is used to delete records from a database table.
- **LOCK:** Table control concurrency.
- **CALL:** Call a PL/SQL or JAVA subprogram.
- **EXPLAIN PLAN:** It describes the access path to data.

DCL (Data Control Language)

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT:** This command gives users access privileges to the database.
- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.

TCL (Transaction Control Language)

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. Hence, the following TCL commands are used to control the execution of a transaction:

- **BEGIN:** Opens a Transaction.
- **COMMIT:** Commits a Transaction.
- **ROLLBACK:** Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT:** Sets a saving point within a transaction.

PRACTICAL-2

CREATING AND MANAGING TABLES: CREATE TABLE STATEMENT; REFERENCING ANOTHER USER'S TABLES; THE DEFAULT OPTION; DATA TYPES; ALTER TABLE STATEMENT; ADDING A COLUMN; MODIFYING A COLUMN; DROPPING A COLUMN; DROPPING A TABLE; TRUNCATING A TABLE.

Part A: WRITE SQL QUERIES CREATE, INSERT AND SELECT

- **Creating Table:**

```
mysql> use clg;
Database changed
mysql> create table friends(CRN int(8) primary key, NAME varchar(20), DATE_OF_BIRTH date, PHONE_NO int(10));
Query OK, 0 rows affected, 2 warnings (0.11 sec)
```

- **Inserting values in Table:**

```
mysql> insert into friends values('2221128','Nikhil Thakur','2000-12-12','1234567899');
Query OK, 1 row affected (0.01 sec)

mysql> insert into friends values('2221124','Chotuu','2012-12-12','1234567890');
Query OK, 1 row affected (0.07 sec)

mysql> insert into friends values('2221122','Vishal Rai','2002-10-10','1234567891');
Query OK, 1 row affected (0.00 sec)

mysql> insert into friends values('2221113','Susuant','2001-11-11','1234567892');
Query OK, 1 row affected (0.01 sec)

mysql> insert into friends values('2221123','Partap','2004-04-04','1234567893');
Query OK, 1 row affected (0.07 sec)
```

- **Selecting the table and taking output:**

```
mysql> select * from friends;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221122 | Vishal Rai   | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
5 rows in set (0.00 sec)
```

Part B: WRITE SQL QUERIES ALTER, UPDATE, DELETE AND DROP

- **ALTER TABLE:**

```
mysql> alter table friends add email varchar(50);
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from friends;
+---------+---------------+---------------+------------+-------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   | email |
+---------+---------------+---------------+------------+-------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 | NULL  |
| 2221122 | Vishal Rai    | 2002-10-10    | 1234567891 | NULL  |
| 2221123 | Partap        | 2004-04-04    | 1234567893 | NULL  |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 | NULL  |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 | NULL  |
+---------+---------------+---------------+------------+-------+
5 rows in set (0.00 sec)
```

- **MODIFY TABLE:**

```
mysql>
       alter table friends modify email varchar(100);
Query OK, 5 rows affected (0.14 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from friends;
+---------+---------------+---------------+------------+-------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   | email |
+---------+---------------+---------------+------------+-------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 | NULL  |
| 2221122 | Vishal Rai    | 2002-10-10    | 1234567891 | NULL  |
| 2221123 | Partap        | 2004-04-04    | 1234567893 | NULL  |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 | NULL  |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 | NULL  |
+---------+---------------+---------------+------------+-------+
5 rows in set (0.00 sec)
```

- **DROP TABLE:**

```
mysql> alter table friends drop column email;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from friends;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221122 | Vishal Rai   | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
5 rows in set (0.00 sec)
```

- **DELETE TABLE:**

```
mysql> delete from friends where CRN='2221128';
Query OK, 1 row affected (0.08 sec)

mysql> select * from friends;
+---------+------------+---------------+------------+
| CRN     | NAME       | DATE_OF_BIRTH | PHONE_NO   |
+---------+------------+---------------+------------+
| 2221113 | Susuant    | 2001-11-11    | 1234567892 |
| 2221122 | Vishal Rai | 2002-10-10    | 1234567891 |
| 2221123 | Partap     | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu     | 2012-12-12    | 1234567890 |
+---------+------------+---------------+------------+
4 rows in set (0.00 sec)
```

- **UPDATE TABLE:**

```
mysql> update friends set NAME='Vishal' where NAME='Vishal rai';
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
5 rows in set (0.00 sec)
```

PRACTICAL -3

BASIC SELECT STATEMENT; SELECTING ALL COLUMNS, SPECIFIC COLUMNS; USING ARITHMETIC OPERATORS; OPERATOR PRECEDENCE; USING PARENTHESIS; DEFINING A NULL VALUE; USING COLUMN ALIASES; CONCATENATION OPERATOR; ELIMINATING DUPLICATE ROWS; DISPLAYING TABLE STRUCTURE

- **SELECT STATEMENT**

(i) To select all columns:

```
mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
5 rows in set (0.00 sec)
```

(ii) To select specific columns:

```
mysql> select CRN,DATE_OF_BIRTH from friends;
+---------+---------------+
| CRN     | DATE_OF_BIRTH |
+---------+---------------+
| 2221113 | 2001-11-11    |
| 2221122 | 2002-10-10    |
| 2221123 | 2004-04-04    |
| 2221124 | 2012-12-12    |
| 2221128 | 2000-12-12    |
+---------+---------------+
5 rows in set (0.00 sec)
```

- **ARITHMETIC OPERATORS**

```
mysql> select 20/10 as ARITHMETIC_OPERATIONS;
+----------------------+
| ARITHMETIC_OPERATIONS |
+----------------------+
|               2.0000 |
+----------------------+
1 row in set (0.00 sec)

mysql> select 20%10 as ARITHMETIC_OPERATIONS;
+----------------------+
| ARITHMETIC_OPERATIONS |
+----------------------+
|                    0 |
+----------------------+
1 row in set (0.00 sec)

mysql> select 20+10 as ARITHMETIC_OPERATIONS;
+----------------------+
| ARITHMETIC_OPERATIONS |
+----------------------+
|                   30 |
+----------------------+
1 row in set (0.07 sec)

mysql> select 20-10 as ARITHMETIC_OPERATIONS;
+----------------------+
| ARITHMETIC_OPERATIONS |
+----------------------+
|                   10 |
+----------------------+
1 row in set (0.07 sec)

mysql> select 20*10 as ARITHMETIC_OPERATIONS;
+----------------------+
| ARITHMETIC_OPERATIONS |
+----------------------+
|                  200 |
+----------------------+
1 row in set (0.00 sec)
```

- **COLUMN ALIASES**

```
mysql> select CRN as class_roll_no,DATE_OF_BIRTH as janam_tithi from friends;
+---------------+-------------+
| class_roll_no | janam_tithi |
+---------------+-------------+
|       2221113 | 2001-11-11  |
|       2221122 | 2002-10-10  |
|       2221123 | 2004-04-04  |
|       2221124 | 2012-12-12  |
|       2221128 | 2000-12-12  |
+---------------+-------------+
5 rows in set (0.00 sec)
```

- **CONCATENATION OPERATOR**

```
mysql> select concat(CRN,NAME) as STUDENT,DATE_OF_BIRTH,PHONE_NO from friends;
+----------------------+---------------+------------+
| STUDENT              | DATE_OF_BIRTH | PHONE_NO   |
+----------------------+---------------+------------+
| 2221113Susuant       | 2001-11-11    | 1234567892 |
| 2221122Vishal        | 2002-10-10    | 1234567891 |
| 2221123Partap        | 2004-04-04    | 1234567893 |
| 2221124Chotuu        | 2012-12-12    | 1234567890 |
| 2221128Nikhil Thakur | 2000-12-12    | 1234567899 |
+----------------------+---------------+------------+
5 rows in set (0.00 sec)
```

- **ELIMINATING REDUNDANT DATA**

```
mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221126 | Vishal        | 2009-09-09    | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
6 rows in set (0.00 sec)

mysql> select distinct(NAME) as NAMES from friends;
+---------------+
| NAMES         |
+---------------+
| Susuant       |
| Vishal        |
| Partap        |
| Chotuu        |
| Nikhil Thakur |
+---------------+
5 rows in set (0.07 sec)
```

- **DISPLAY TABLE STRUCTURE**

```
mysql> DESC friends;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| CRN           | int         | NO   | PRI | NULL    |       |
| NAME          | varchar(20) | YES  |     | NULL    |       |
| DATE_OF_BIRTH | date        | YES  |     | NULL    |       |
| PHONE_NO      | int         | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

PRACTICAL-4

LIMITING ROWS USING A SELECTION; CHARATER STRINGS AND DATES; COMPARISION CONDITIONS; USING THE BETWEEN CONDITION; IN CONDITION; LIKE CONDITION; NULL CONDITION; LOGICAL CONDITIONS - AND, OR and NOT OPERATORS; RULES OF PRECEDENCE ORDER BY CLAUSE; SORTING – ASCENDING, DESCENDING ORDER.

- **LIMITING ROWS USING A SELECTION**
  a) **CHARACTER STRING**

```
mysql> select instr('NIKHIL THAKUR','T');
+----------------------------+
| instr('NIKHIL THAKUR','T') |
+----------------------------+
|                          8 |
+----------------------------+
1 row in set (0.00 sec)

mysql> select lower('NIKHIL THAKUR');
+------------------------+
| lower('NIKHIL THAKUR') |
+------------------------+
| nikhil thakur          |
+------------------------+
1 row in set (0.00 sec)

mysql> select length('NIKHIL THAKUR');
+--------------------------+
| length('NIKHIL THAKUR') |
+--------------------------+
|                       13 |
+--------------------------+
1 row in set (0.00 sec)

mysql> select trim('     NIKHIL THAKUR      ');
+-----------------------------------+
| trim('     NIKHIL THAKUR      ') |
+-----------------------------------+
| NIKHIL THAKUR                     |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> select substr('NIKHIL THAKUR',2,10);
+-----------------------------+
| substr('NIKHIL THAKUR',2,10) |
+-----------------------------+
| IKHIL THAK                  |
+-----------------------------+
1 row in set (0.00 sec)
```

b) DATES

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2024-02-26 20:45:31 |
+---------------------+
1 row in set (0.00 sec)

mysql> select sysdate();
+---------------------+
| sysdate()           |
+---------------------+
| 2024-02-26 20:45:34 |
+---------------------+
1 row in set (0.00 sec)

mysql> select dayofyear('2014-12-12');
+--------------------------+
| dayofyear('2014-12-12') |
+--------------------------+
|                      346 |
+--------------------------+
1 row in set (0.00 sec)

mysql> select curtime();
+-----------+
| curtime() |
+-----------+
| 20:46:00  |
+-----------+
1 row in set (0.00 sec)

mysql> select monthname('2014-12-12');
+--------------------------+
| monthname('2014-12-12') |
+--------------------------+
| December                 |
+--------------------------+
1 row in set (0.07 sec)
```

- **CONPARISION CONDITIONS**
  - **a) BETWEEN CONDITION**

```
mysql> select * from friends where DATE_OF_BIRTH BETWEEN '2002-01-01' AND '2005-01-01';
+---------+--------+---------------+------------+
| CRN     | NAME   | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------+---------------+------------+
| 2221122 | Vishal | 2002-10-10    | 1234567891 |
| 2221123 | Partap | 2004-04-04    | 1234567893 |
+---------+--------+---------------+------------+
2 rows in set (0.07 sec)
```

  - **b) IN CONDITION**

```
mysql> select NAME,PHONE_NO from friends where CRN IN('2221113','2221124');
+---------+------------+
| NAME    | PHONE_NO   |
+---------+------------+
| Susuant | 1234567892 |
| Chotuu  | 1234567890 |
+---------+------------+
2 rows in set (0.00 sec)
```

  - **c) LIKE CONDITION**

```
mysql> select * from friends where NAME like '_i%';
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221126 | Vishal        | 2009-09-09    | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
3 rows in set (0.07 sec)
```

d) NULL CONDITION

```
mysql> select * from friends;
+---------+--------------+--------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+--------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+--------------+------------+
6 rows in set (0.00 sec)

mysql> select * from friends where PHONE_NO is null;
+---------+--------+--------------+----------+
| CRN     | NAME   | DATE_OF_BIRTH | PHONE_NO |
+---------+--------+--------------+----------+
| 2221126 | Vishal | 2009-09-09    |     NULL |
+---------+--------+--------------+----------+
1 row in set (0.00 sec)
```

- **LOGICAL OPERATOR**

a) AND

```
mysql> select * from friends where PHONE_NO is null && NAME='Vishal';
+---------+--------+--------------+----------+
| CRN     | NAME   | DATE_OF_BIRTH | PHONE_NO |
+---------+--------+--------------+----------+
| 2221126 | Vishal | 2009-09-09    |     NULL |
+---------+--------+--------------+----------+
1 row in set, 1 warning (0.00 sec)
```

b) OR

```
mysql> select * from friends where NAME='Chotuu' || NAME='Partap';
+---------+--------+---------------+------------+
| CRN     | NAME   | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------+---------------+------------+
| 2221123 | Partap | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu | 2012-12-12    | 1234567890 |
+---------+--------+---------------+------------+
2 rows in set, 1 warning (0.00 sec)
```

c) NOT

```
mysql> select * from friends where!(NAME='Chotuu');
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
5 rows in set, 1 warning (0.00 sec)
```

- **ORDER BY CLAUSE**

```
mysql> select * from friends order by NAME desc;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
+---------+--------------+---------------+------------+
6 rows in set (0.00 sec)
```

- **SORTING**
    a) **ASCENDING ORDER**

```
mysql> select NAME from friends order by NAME asc;
+---------------+
| NAME          |
+---------------+
| Chotuu        |
| Nikhil Thakur |
| Partap        |
| Susuant       |
| Vishal        |
| Vishal        |
+---------------+
6 rows in set (0.00 sec)
```

b) **DESCENDING ORDER**

```
mysql> select NAME from friends order by NAME desc;
+---------------+
| NAME          |
+---------------+
| Vishal        |
| Vishal        |
| Susuant       |
| Partap        |
| Nikhil Thakur |
| Chotuu        |
+---------------+
6 rows in set (0.00 sec)
```

PRACTICAL- 5

MANIPULATING DATA : DATA MANIPULATION LANGUAGE; ADDING A NEW ROW TO A TABLE; INSERTING NEW ROWS, ROWS WITH NULL VALUES, SPECIFIC DATE VALUES, UPDATING ROWS IN A TABLE; UPDATING 2 COLUMNS; UPDATING ROWS BASED ON ANOTHER TABLE; REMOVING A ROW FROM DELETING ROWS FROM A TABLE; DELETING ROWS BASED ON ANOTHER TABLE.

- **ADDING A NEW ROW TO A TABLE**
  - **a) INSERTING NEW ROWS**

```
mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221126 | Vishal        | 2009-09-09    |       NULL |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
6 rows in set (0.01 sec)

mysql> insert into friends(CRN, NAME) VALUES('2221114','sabal gang');
Query OK, 1 row affected (0.01 sec)

mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang    | NULL          |       NULL |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221126 | Vishal        | 2009-09-09    |       NULL |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
7 rows in set (0.00 sec)
```

b) ROWS WITH NULL VALUES

```
mysql> insert into friends(CRN, NAME,DATE_OF_BIRTH,PHONE_NO) VALUES('2221127','varun',NULL,'1234567894');
Query OK, 1 row affected (0.00 sec)

mysql> select * from friends;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang   | NULL          |       NULL |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221127 | varun        | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
8 rows in set (0.00 sec)
```

- **UPDATING ROWS IN A TABLE**

```
mysql> update friends set PHONE_NO='1234567896' where PHONE_NO='1234567897';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from friends;
+---------+---------------+---------------+------------+
| CRN     | NAME          | DATE_OF_BIRTH | PHONE_NO   |
+---------+---------------+---------------+------------+
| 2221113 | Susuant       | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang    | 2003-10-10    | 1234567896 |
| 2221122 | Vishal        | 2002-10-10    | 1234567891 |
| 2221123 | Partap        | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu        | 2012-12-12    | 1234567890 |
| 2221126 | Vishal        | 2009-09-09    |       NULL |
| 2221127 | varun         | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+---------------+---------------+------------+
8 rows in set (0.00 sec)
```

- **UPDATING TWO COLUMNS**

```
mysql> update friends set DATE_OF_BIRTH='2003-10-10',PHONE_NO='1234567897' where CRN='2221114';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from friends;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang   | 2003-10-10    | 1234567897 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221127 | varun        | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
8 rows in set (0.00 sec)
```

- **UPDATING ROWS BASED ON ANOTHER TABLE**

```
mysql> select * from marks;
+---------+--------------+-------+
| crn     | name         | marks |
+---------+--------------+-------+
| 2221113 | susuant      |    68 |
| 2221122 | vishal       |    66 |
| 2221124 | Chotuu       |    68 |
| 2221128 | Nikhil Thakur |    66 |
+---------+--------------+-------+
4 rows in set (0.00 sec)

mysql> update friends INNER JOIN marks ON friends.CRN = marks.CRN set marks=marks+10 where marks=66;
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from marks;
+---------+--------------+-------+
| crn     | name         | marks |
+---------+--------------+-------+
| 2221113 | susuant      |    68 |
| 2221122 | vishal       |    76 |
| 2221124 | Chotuu       |    68 |
| 2221128 | Nikhil Thakur |    76 |
+---------+--------------+-------+
4 rows in set (0.00 sec)
```

- **DELETING ROWS FROM A TABLE**

```
mysql> select * from friends;
+---------+--------------+--------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+--------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang   | 2003-10-10    | 1234567896 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221126 | Vishal       | 2009-09-09    |       NULL |
| 2221127 | varun        | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+--------------+------------+
8 rows in set (0.00 sec)

mysql> delete from friends where CRN='2221126';
Query OK, 1 row affected (0.01 sec)

mysql> select * from friends;
+---------+--------------+--------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+--------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang   | 2003-10-10    | 1234567896 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221127 | varun        | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+--------------+------------+
7 rows in set (0.00 sec)
```

- **DELETING ROWS BASED ON ANOTHER TABLE**

```
mysql> delete friends, marks from friends inner join marks on friends.CRN=marks.crn where friends.CRN='2221127';
Query OK, 0 rows affected (0.00 sec)

mysql> select * from friends;
+---------+--------------+---------------+------------+
| CRN     | NAME         | DATE_OF_BIRTH | PHONE_NO   |
+---------+--------------+---------------+------------+
| 2221113 | Susuant      | 2001-11-11    | 1234567892 |
| 2221114 | sabal gang   | 2003-10-10    | 1234567896 |
| 2221122 | Vishal       | 2002-10-10    | 1234567891 |
| 2221123 | Partap       | 2004-04-04    | 1234567893 |
| 2221124 | Chotuu       | 2012-12-12    | 1234567890 |
| 2221127 | varun        | NULL          | 1234567894 |
| 2221128 | Nikhil Thakur | 2000-12-12    | 1234567899 |
+---------+--------------+---------------+------------+
7 rows in set (0.00 sec)
```

PRACTICAL NO 3: RESTRICTING AND SORTING DATA

AIM: Limiting rows using a selection; character strings and dates; comparison conditions; using the BETWEEN condition; IN condition; LIKE condition; NULL conditions; logical conditions-AND, OR and NOT operators; rules of precedence; ORDER BY clause; sorting –ascending, descending order.

A. LIMITING ROWS USING A SELECTION

STUDENTS

SID	Name	City	Age	Gender	Branch
2104482	Avlyn Kaur	Ludhiana	19	Female	IT
2104471	Anu grewal	Chandigarh	21	Female	Computer Science
2104488	Dashyamjit Singh	Ludhiana	19	Male	IT
2104364	Anmolvir Singh	Samrala	20	Male	Electronics
2104477	Arshnoor	Patiala	19	Female	Computer Science
2104465	Samiksha	Chandigarh	19	Female	Computer Science
2104478	Sumehar Gill	Ludhiana	20	Male	Electronics
2104443	Shashank Kumar	Patiala	20	Male	Electrical

- Restrict the rows returned by using the WHERE clause.

Syntax:

SELECT*|{[DISTINCT] column/expression[alias],…}

FROM table

[WHERE condition(s)];

EXAMPLE:

```
mysql> SELECT SID,Name,City,Age,Gender,Branch
    -> FROM student
    -> WHERE Age = 19;
```

SID	Name	City	Age	Gender	Branch
2104482	Avlyn Kaur	Ludhiana	19	Female	IT
2104488	Dashyamjit Singh	Ludhiana	19	Male	IT
2104477	Arshnoor	Patiala	19	Female	Computer Science
2104465	Samiksha	Chandigarh	19	Female	Computer Science

B. CHARACTER STRINGS AND DATES

. Character strings and date values are enclosed in single quotation marks.

. Character values are case sensitive, and date values are format sensitive.

. The default date format is DD-MON-RR.

```
mysql> SELECT Name,Age,City
    -> From student
    -> WHERE City = 'Ludhiana';
```

C. **Comparison Conditions**

Operator	Meaning
Between	Between two values(inclusive),
In	Match any of a list of values
Like	Match a character pattern
Is Null	Is a null value

➢ **Using the BETWEEN Condition**

Use the BETWEEN condition to display rows based on a range of values.

```
mysql> SELECT Name,Age
    -> From student
    -> WHERE Age BETWEEN 19 AND 20;
+------------------+------+
| Name             | Age  |
+------------------+------+
| Avlyn Kaur       |   19 |
| Dashyamjit Singh |   19 |
| Anmolvir Singh   |   20 |
| Arshnoor         |   19 |
| Samiksha         |   19 |
| Sumehar Gill     |   20 |
| Shashank Kumar   |   20 |
+------------------+------+
7 rows in set (0.00 sec)
```

➢ **Using the IN Condition**

Use the IN membership condition to test for values in a list.

```
mysql> SELECT SID,Name,City,Age
    -> FROM student
    -> WHERE Age IN(20,21);
+---------+----------------+------------+------+
| SID     | Name           | City       | Age  |
+---------+----------------+------------+------+
| 2104471 | Anu grewal     | Chandigarh |   21 |
| 2104364 | Anmolvir Singh | Samrala    |   20 |
| 2104478 | Sumehar Gill   | Ludhiana   |   20 |
| 2104443 | Shashank Kumar | Patiala    |   20 |
+---------+----------------+------------+------+
4 rows in set (0.00 sec)
```

➢ **Using the LIKE Condition**

- Use the LIKE condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - %denotes zero or many characters.
  - _denotes one character.
- You can combine pattern-matching characters.
- You can use the escape identifier to search for the actual % and _ symbol.

```
mysql> SELECT Name
    -> FROM student
    -> WHERE Name LIKE'S%';
+----------------+
| Name           |
+----------------+
| Samiksha       |
| Sumehar Gill   |
| Shashank Kumar |
+----------------+
3 rows in set (0.01 sec)
```

➢ **Using the NULL Conditions**

Test for nulls with the IS NULL operator.

```
mysql> SELECT Name, SID
    -> FROM student
    -> WHERE SID IS NULL;
Empty set (0.00 sec)
```

D. **Logical Conditions**

➢ Using the AND Operator

AND requires both conditions to be true.

```
mysql> SELECT SID,Name,Age,Branch
    -> FROM student
    -> WHERE Age = 19
    -> AND Branch LIKE'%IT%';
+---------+------------------+------+--------+
| SID     | Name             | Age  | Branch |
+---------+------------------+------+--------+
| 2104482 | Avlyn Kaur       |   19 | IT     |
| 2104488 | Dashyamjit Singh |   19 | IT     |
+---------+------------------+------+--------+
2 rows in set (0.00 sec)
```

➢ Using the OR Operator

OR requires either condition to be true.

```
mysql> SELECT SID,Name,Age,Branch
    -> FROM student
    -> WHERE Age = 19
    -> OR Branch LIKE'%IT%';
+---------+------------------+------+------------------+
| SID     | Name             | Age  | Branch           |
+---------+------------------+------+------------------+
| 2104482 | Avlyn Kaur       |   19 | IT               |
| 2104488 | Dashyamjit Singh |   19 | IT               |
| 2104477 | Arshnoor         |   19 | Computer Science |
| 2104465 | Samiksha         |   19 | Computer Science |
+---------+------------------+------+------------------+
4 rows in set (0.00 sec)
```

➢ Using the NOT Operator

It returns true if the following condition is false.

```
mysql> SELECT Name,Branch
    -> FROM student
    -> WHERE Branch
    -> NOT IN ('IT','Computer Science');
+----------------+-------------+
| Name           | Branch      |
+----------------+-------------+
| Anmolvir Singh | Electronics |
| Sumehar Gill   | Electronics |
| Shashank Kumar | Electrical  |
+----------------+-------------+
3 rows in set (0.00 sec)
```

E. **Rules of Precedence**

Order Evaluated Operator

1. Arithmetic Operators
2. Concatenation Operator
3. Comparison Conditions
4. Is [NOT] NULL,LIKE,[NOT] IN
5. [NOT] BETWEEN
6. NOT logical condition
7. AND logical condition
8. OR logical condition

- Override rules of precedence by using parenthesis.

```
mysql> SELECT SID,Name,Branch
    -> FROM student
    -> WHERE (Branch = 'IT'
    -> OR Branch = 'Computer Science')
    -> AND Age = 19;
+---------+------------------+------------------+
| SID     | Name             | Branch           |
+---------+------------------+------------------+
| 2104482 | Avlyn Kaur       | IT               |
| 2104488 | Dashyamjit Singh | IT               |
| 2104477 | Arshnoor         | Computer Science |
| 2104465 | Samiksha         | Computer Science |
+---------+------------------+------------------+
4 rows in set (0.00 sec)
```

F. **Order By Clause**

- Sort rows with the ORDER BY clause
- Ascending order is the default sort order.

```
mysql> SELECT SID,Name,City,Age
    -> FROM student
    -> ORDER BY Age;
+---------+------------------+------------+------+
| SID     | Name             | City       | Age  |
+---------+------------------+------------+------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 |
| 2104477 | Arshnoor         | Patiala    |   19 |
| 2104465 | Samiksha         | Chandigarh |   19 |
| 2104364 | Anmolvir Singh   | Samrala    |   20 |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 |
| 2104443 | Shashank Kumar   | Patiala    |   20 |
| 2104471 | Anu grewal       | Chandigarh |   21 |
+---------+------------------+------------+------+
8 rows in set (0.01 sec)
```

- **Sorting in Descending Order**

```
mysql> SELECT SID,Name,City,Age
    -> FROM student
    -> ORDER BY Age DESC;
+---------+------------------+------------+------+
| SID     | Name             | City       | Age  |
+---------+------------------+------------+------+
| 2104471 | Anu grewal       | Chandigarh |   21 |
| 2104364 | Anmolvir Singh   | Samrala    |   20 |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 |
| 2104443 | Shashank Kumar   | Patiala    |   20 |
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 |
| 2104477 | Arshnoor         | Patiala    |   19 |
| 2104465 | Samiksha         | Chandigarh |   19 |
+---------+------------------+------------+------+
8 rows in set (0.00 sec)
```

PRACTICAL NO 4: MANIPULATING DATA

AIM: Data manipulation language; adding a new row to a table; inserting-new rows, rows with NULL values, specific date values; updating rows in a table; updating two columns; updating rows based on another table; removing a row from a table deleting rows from a table; deleting rows based on another table.

A. **Data Manipulation Language:**

- A DML statement is executed when you:
  - Add new row to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A collection of DML statements that form a logical unit of work is called a Transaction.
- Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations:
  - Decrease the saving account.
  - Increase the checking account.
  - And record the transaction in the transaction journal.

B. **Adding a new row to a table**

- Add new rows to a table by using the insert statement.

Syntax:

INSERT INTO table [(column [, column…])]

VALUES (value [, value …]);

- Only one row is inserted at a time with this syntax.
- **Example:**

```
mysql> INSERT INTO student ( SID,Name,City,Age,Gender,Branch )
    -> VALUES      (2104365,'Ekamjot Singh','Ludhiana',20,'Male','Electronics');
Query OK, 1 row affected (0.03 sec)

mysql> SELECT*FROM student;
+---------+------------------+------------+------+--------+------------------+
| SID     | Name             | City       | Age  | Gender | Branch           |
+---------+------------------+------------+------+--------+------------------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 | Female | IT               |
| 2104471 | Anu grewal       | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT               |
| 2104364 | Anmolvir Singh   | Samrala    |   20 | Male   | Electronics      |
| 2104477 | Arshnoor         | Patiala    |   19 | Female | Computer Science |
| 2104465 | Samiksha         | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 | Male   | Electronics      |
| 2104443 | Shashank Kumar   | Patiala    |   20 | Male   | Electrical       |
| 2104365 | Ekamjot Singh    | Ludhiana   |   20 | Male   | Electronics      |
+---------+------------------+------------+------+--------+------------------+
9 rows in set (0.00 sec)
```

C. **Inserting rows with NULL Values**

- Implicit method: Omit the column from the column list.
- **Example:**

```
mysql> INSERT INTO student ( SID,Name,City)
    -> VALUES      (2104366,'Jaskaran Singh','Ludhiana');
Query OK, 1 row affected (0.01 sec)
```

- Explicit method: Specify the NULL keyword in the VALUES clause, specify the empty string (“) in the VALUES list for character strings and dates.
- **Example:**

```
mysql> INSERT INTO student
    -> VALUES      ( 2104445,'Harshveer Kaur','Chandigarh',NULL,NULL,NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT*FROM student;
+---------+------------------+------------+------+--------+------------------+
| SID     | Name             | City       | Age  | Gender | Branch           |
+---------+------------------+------------+------+--------+------------------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 | Female | IT               |
| 2104471 | Anu grewal       | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT               |
| 2104364 | Anmolvir Singh   | Samrala    |   20 | Male   | Electronics      |
| 2104477 | Arshnoor         | Patiala    |   19 | Female | Computer Science |
| 2104465 | Samiksha         | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 | Male   | Electronics      |
| 2104443 | Shashank Kumar   | Patiala    |   20 | Male   | Electrical       |
| 2104365 | Ekamjot Singh    | Ludhiana   |   20 | Male   | Electronics      |
| 2104366 | Jaskaran Singh   | Ludhiana   | NULL | NULL   | NULL             |
| 2104445 | Harshveer Kaur   | Chandigarh | NULL | NULL   | NULL             |
+---------+------------------+------------+------+--------+------------------+
11 rows in set (0.00 sec)
```

D. **Updating rows in a table:**

- Modify existing rows with the UPDATE statement.
- Specific row or rows are modified if you specify the WHERE clause.

- All rows in the table are modified if you omit the WHERE clause.
- **Example:**

```
mysql> UPDATE student
    -> SET Age = 20
    -> WHERE SID = 2104445;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT*FROM student;
+---------+------------------+------------+------+--------+------------------+
| SID     | Name             | City       | Age  | Gender | Branch           |
+---------+------------------+------------+------+--------+------------------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 | Female | IT               |
| 2104471 | Anu grewal       | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT               |
| 2104364 | Anmolvir Singh   | Samrala    |   20 | Male   | Electronics      |
| 2104477 | Arshnoor         | Patiala    |   19 | Female | Computer Science |
| 2104465 | Samiksha         | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 | Male   | Electronics      |
| 2104443 | Shashank Kumar   | Patiala    |   20 | Male   | Electrical       |
| 2104365 | Ekamjot Singh    | Ludhiana   |   20 | Male   | Electronics      |
| 2104366 | Jaskaran Singh   | Ludhiana   | NULL | NULL   | NULL             |
| 2104445 | Harshveer Kaur   | Chandigarh |   20 | NULL   | NULL             |
+---------+------------------+------------+------+--------+------------------+
11 rows in set (0.00 sec)
```

E. **Updating two columns:**

- **Syntax:**

```
UPDATE table_name
SET column1 = value1,
    column2 = value2,
    ...
    columnN = valueN
WHERE condition;
```

- **Example:**

```
mysql> UPDATE student
    -> SET City = 'Patiala',
    ->     Age  = ' 21'
    -> WHERE SID = 2104366;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT*FROM student;
+---------+-----------------+------------+------+--------+-----------------+
| SID     | Name            | City       | Age  | Gender | Branch          |
+---------+-----------------+------------+------+--------+-----------------+
| 2104482 | Avlyn Kaur      | Ludhiana   |   19 | Female | IT              |
| 2104471 | Anu grewal      | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT              |
| 2104364 | Anmolvir Singh  | Samrala    |   20 | Male   | Electronics     |
| 2104477 | Arshnoor        | Patiala    |   19 | Female | Computer Science |
| 2104465 | Samiksha        | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill    | Ludhiana   |   20 | Male   | Electronics     |
| 2104443 | Shashank Kumar  | Patiala    |   20 | Male   | Electrical      |
| 2104365 | Ekamjot Singh   | Ludhiana   |   20 | Male   | Electronics     |
| 2104366 | Jaskaran Singh  | Patiala    |   21 | NULL   | NULL            |
| 2104445 | Harshveer Kaur  | Chandigarh |   20 | NULL   | NULL            |
+---------+-----------------+------------+------+--------+-----------------+
11 rows in set (0.00 sec)
```

F. **Updating rows based on another table:**

- Use subqueries in UPDATE statements to update rows in a table based on value from another table.
- The given example updates the copy_student table based on the values from the student table.

```
mysql> UPDATE copy_student
    -> SET     Age = ( SELECT Age
    ->                 FROM student
    ->                 WHERE SID = 2104445 )
    -> WHERE   City = ( SELECT City
    ->                  FROM student
    ->                  WHERE SID = 2104366 );
Query OK, 4 rows affected (0.01 sec)
Rows matched: 6  Changed: 4  Warnings: 0
```

G. **Deleting rows from a table:**

- You can remove existing rows from a table by using the DELETE statement.
- **Syntax:**
  DELETE [FROM] table
  [WHERE condition];
- **Example:**

```
mysql> DELETE FROM student
    -> WHERE SID = '2104445';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT*FROM student;
+---------+------------------+------------+------+--------+------------------+
| SID     | Name             | City       | Age  | Gender | Branch           |
+---------+------------------+------------+------+--------+------------------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 | Female | IT               |
| 2104471 | Anu grewal       | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT               |
| 2104364 | Anmolvir Singh   | Samrala    |   20 | Male   | Electronics      |
| 2104477 | Arshnoor         | Patiala    |   19 | Female | Computer Science |
| 2104465 | Samiksha         | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 | Male   | Electronics      |
| 2104443 | Shashank Kumar   | Patiala    |   20 | Male   | Electrical       |
| 2104365 | Ekamjot Singh    | Ludhiana   |   20 | Male   | Electronics      |
| 2104366 | Jaskaran Singh   | Patiala    |   21 | NULL   | NULL             |
+---------+------------------+------------+------+--------+------------------+
10 rows in set (0.00 sec)
```

H. **Deleting rows based on another table:**

- Use subqueries in DELETE statement to remove rows from the another table based on values from another table.
- **Example:**

```
mysql> DELETE FROM copy_student
    -> WHERE SID =
    ->              (SELECT SID
    ->               FROM student
    ->               WHERE Name LIKE '%Jaskaran Singh%');
Query OK, 2 rows affected (0.01 sec)
```

```
mysql> SELECT*FROM copy_student;
+---------+------------------+------------+------+--------+------------------+
| SID     | Name             | City       | Age  | Gender | Branch           |
+---------+------------------+------------+------+--------+------------------+
| 2104482 | Avlyn Kaur       | Ludhiana   |   19 | Female | IT               |
| 2104471 | Anu grewal       | Chandigarh |   21 | Female | Computer Science |
| 2104488 | Dashyamjit Singh | Ludhiana   |   19 | Male   | IT               |
| 2104364 | Anmolvir Singh   | Samrala    |   20 | Male   | Electronics      |
| 2104477 | Arshnoor         | Patiala    |   20 | Female | Computer Science |
| 2104465 | Samiksha         | Chandigarh |   19 | Female | Computer Science |
| 2104478 | Sumehar Gill     | Ludhiana   |   20 | Male   | Electronics      |
| 2104443 | Shashank Kumar   | Patiala    |   20 | Male   | Electrical       |
| 2104365 | Ekamjot Singh    | Ludhiana   |   20 | Male   | Electronics      |
| 2104445 | Harshveer Kaur   | Chandigarh |   20 | NULL   | NULL             |
```

PRACTICAL NO 6: SINGLE ROW FUNCTION

Aim: Character functions - case manipulation and character manipulation functions; number functions, date functions; using arithmetic operators with dates; date functions, conversion functions.

Single Row Function: These functions operate on single rows only and return one result per row. There are different types of single-row functions.

A. **Lower() :** LOWER function converts all letters in the specified string to lowercase.

```
mysql> SELECT name,lower(name) FROM student;
+------------------+------------------+
| name             | lower(name)      |
+------------------+------------------+
| Avlyn Kaur       | avlyn kaur       |
| Anu grewal       | anu grewal       |
| Dashyamjit Singh | dashyamjit singh |
| Anmolvir Singh   | anmolvir singh   |
| Arshnoor         | arshnoor         |
| Samiksha         | samiksha         |
| Sumehar Gill     | sumehar gill     |
| Shashank Kumar   | shashank kumar   |
| Ekamjot Singh    | ekamjot singh    |
| Jaskaran Singh   | jaskaran singh   |
+------------------+------------------+
10 rows in set (0.00 sec)
```

B. **UPPER Function:** UPPER function converts all letters in the specified string to uppercase.

```
mysql>  SELECT name,UPPER(name) FROM student;
+------------------+------------------+
| name             | UPPER(name)      |
+------------------+------------------+
| Avlyn Kaur       | AVLYN KAUR       |
| Anu grewal       | ANU GREWAL       |
| Dashyamjit Singh | DASHYAMJIT SINGH |
| Anmolvir Singh   | ANMOLVIR SINGH   |
| Arshnoor         | ARSHNOOR         |
| Samiksha         | SAMIKSHA         |
| Sumehar Gill     | SUMEHAR GILL     |
| Shashank Kumar   | SHASHANK KUMAR   |
| Ekamjot Singh    | EKAMJOT SINGH    |
| Jaskaran Singh   | JASKARAN SINGH   |
+------------------+------------------+
10 rows in set (0.00 sec)
```

C. **L Trim:** LTRIM function removes all specified characters from the left-hand side of a string.

```
mysql> SELECT SID,LTRIM(Name) AS CleanedName
    -> FROM student;
+---------+-----------------+
| SID     | CleanedName     |
+---------+-----------------+
| 2104482 | Avlyn Kaur      |
| 2104471 | Anu grewal      |
| 2104488 | Dashyamjit Singh |
| 2104364 | Anmolvir Singh  |
| 2104477 | Arshnoor        |
| 2104465 | Samiksha        |
| 2104478 | Sumehar Gill    |
| 2104443 | Shashank Kumar  |
| 2104365 | Ekamjot Singh   |
| 2104366 | Jaskaran Singh  |
+---------+-----------------+
10 rows in set (0.00 sec)
```

D. **R Trim:** RTRIM function removes all specified characters from the right-hand side of a string.

```
mysql> SELECT SID,RTRIM(Name) AS CleanedName
    -> FROM student;
+---------+-----------------+
| SID     | CleanedName     |
+---------+-----------------+
| 2104482 | Avlyn Kaur      |
| 2104471 | Anu grewal      |
| 2104488 | Dashyamjit Singh |
| 2104364 | Anmolvir Singh  |
| 2104477 | Arshnoor        |
| 2104465 | Samiksha        |
| 2104478 | Sumehar Gill    |
| 2104443 | Shashank Kumar  |
| 2104365 | Ekamjot Singh   |
| 2104366 | Jaskaran Singh  |
+---------+-----------------+
10 rows in set (0.00 sec)
```

E. **L Pad:** left-pad a string with a specified character or characters to achieve a desired length.

```
mysql>  SELECT LPAD('Avlyn ','10','*') FROM student;
+-------------------------+
| LPAD('Avlyn ','10','*') |
+-------------------------+
| ****Avlyn               |
```

F. **R Pad:** The RPAD function is used to add a specified character (or a space) to the right side of a string until it reaches a desired length.

```
mysql> SELECT RPAD('Avlyn','10','*') FROM student;
+-------------------------+
| RPAD('Avlyn','10','*') |
+-------------------------+
| Avlyn*****              |
```

G. **SUBSTRING:** The SUBSTRING function extracts a substring (part of a string) from a given string.

```
mysql> SELECT SUBSTRING('Hello, World!', 8, 5) AS ExtractedString;
+-----------------+
| ExtractedString |
+-----------------+
| World           |
+-----------------+
1 row in set (0.00 sec)
```

H. **COUNT:** The COUNT() function returns the number of records returned by a select query.

```
mysql> SELECT COUNT(SID) AS Numberofstudents FROM student;
+-------------------+
| Numberofstudents  |
+-------------------+
|                10 |
+-------------------+
1 row in set (0.01 sec)
```

I. **ABS:** the ABS() function is used to return the absolute (positive) value of a number.

```
mysql> SELECT ABS(-243.5);
+-------------+
| ABS(-243.5) |
+-------------+
|       243.5 |
+-------------+
1 row in set (0.01 sec)
```

J. **MOD:** The MOD() function returns the remainder of a number divided by another number.

```
mysql> SELECT MOD(10, 2);
+------------+
| MOD(10, 2) |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)
```

K. **ROUND:** The ROUND() function rounds a number to a specified number of decimal places.

```
mysql> SELECT ROUND(135.375, 2);
+--------------------+
| ROUND(135.375, 2) |
+--------------------+
|             135.38 |
+--------------------+
1 row in set (0.00 sec)
```

L. **CEIL:** The CEIL() function returns the smallest integer value that is bigger than or equal to a number.

```
mysql> SELECT CEIL(25.75);
+-------------+
| CEIL(25.75) |
+-------------+
|          26 |
+-------------+
1 row in set (0.00 sec)
```

M. **FLOOR:** The FLOOR() function returns the largest integer value that is smaller than or equal to a number.

```
mysql> SELECT FLOOR(25.75);
+--------------+
| FLOOR(25.75) |
+--------------+
|           25 |
+--------------+
1 row in set (0.00 sec)
```

N. **POWER:** The POWER() function returns the value of a number raised to the power of another number.

```
mysql> SELECT POWER(4, 2);
+-------------+
| POWER(4, 2) |
+-------------+
|          16 |
+-------------+
1 row in set (0.00 sec)
```

O. **SQUAREROOT:** The SQRT() function returns the square root of a number.

```
mysql> SELECT SQRT(64);
+----------+
| SQRT(64) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

P. **ASCII:** The ASCII() function returns the ASCII value for the specific character.

```
mysql> SELECT Name, ASCII(Name) AS NumCodeofFirstChar
    -> FROM student;
+------------------+--------------------+
| Name             | NumCodeofFirstChar |
+------------------+--------------------+
| Avlyn Kaur       |                 65 |
| Anu grewal       |                 65 |
| Dashyamjit Singh |                 68 |
| Anmolvir Singh   |                 65 |
| Arshnoor         |                 65 |
| Samiksha         |                 83 |
| Sumehar Gill     |                 83 |
| Shashank Kumar   |                 83 |
| Ekamjot Singh    |                 69 |
| Jaskaran Singh   |                 74 |
+------------------+--------------------+
10 rows in set (0.00 sec)
```

Multiple Row Function: Multiple row functions work upon group of rows and return one result for the complete set of rows. They are also known as Group Functions.

A. **SUM:** The SUM() function calculates the sum of a set of values.

```
mysql> SELECT SUM(SID) AS TotalStudents FROM student;
+---------------+
| TotalStudents |
+---------------+
|      21044399 |
+---------------+
1 row in set (0.00 sec)
```

B. **MAX:** The MAX() function returns the maximum value in a set of values.

```
mysql> SELECT MAX(SID) AS LargestSID FROM student;
+------------+
| LargestSID |
+------------+
|    2104488 |
+------------+
1 row in set (0.00 sec)
```

C. **MIN:** The MIN() function returns the minimum value in a set of values.

```
mysql> SELECT MIN(SID) AS SmallestSID FROM student;
+-------------+
| SmallestSID |
+-------------+
|     2104364 |
+-------------+
1 row in set (0.00 sec)
```

D. **AVERAGE:** The AVG() function returns the average value of an expression.

```
mysql> SELECT AVG(SID) AS AverageSID FROM student;
+--------------+
| AverageSID   |
+--------------+
| 2104439.9000 |
+--------------+
1 row in set (0.00 sec)
```

PRACTICAL NO 7: DISPLAYING THE DATA FROM MULTIPLE TABLES

AIM: Cartesian products; different types of joins specific to the software package; SQL compliant joins

A. **INNER JOIN:** The INNER JOIN keyword selects records that have matching values in both tables.

```
mysql> SELECT student.SID, STUDENTS.Name
    -> FROM student
    -> INNER JOIN STUDENTS ON student.SID = STUDENTS.SID;
+---------+----------------+
| SID     | Name           |
+---------+----------------+
| 2104482 | Avlyn Kaur     |
| 2104471 | Anu grewal     |
| 2104364 | Anmolvir Singh |
+---------+----------------+
3 rows in set (0.00 sec)
```

B. **NATURAL JOIN:** Natural join is an SQL join operation that creates a join on the base of the common columns in the tables. To perform natural join there must be one common attribute(Column) between two tables.

```
mysql> SELECT STUDENTS.SID, Name, Branch
    -> FROM STUDENTS
    -> NATURAL JOIN student;
+---------+----------------+------------------+
| SID     | Name           | Branch           |
+---------+----------------+------------------+
| 2104482 | Avlyn Kaur     | IT               |
| 2104471 | Anu grewal     | Computer Science |
| 2104364 | Anmolvir Singh | Electronics      |
+---------+----------------+------------------+
3 rows in set (0.00 sec)
```

C. **LEFT JOIN:** The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

```
mysql> SELECT student.Name, STUDENTS.SID
    -> FROM student
    -> LEFT JOIN STUDENTS ON student.Branch = STUDENTS.Branch
    -> ORDER BY student.Name;
+------------------+---------+
| Name             | SID     |
+------------------+---------+
| Anmolvir Singh   | 2104364 |
| Anu grewal       | 2104471 |
| Arshnoor         | 2104471 |
| Avlyn Kaur       | 2104482 |
| Dashyamjit Singh | 2104482 |
| Ekamjot Singh    | 2104364 |
| Jaskaran Singh   |    NULL |
| Samiksha         | 2104471 |
| Shashank Kumar   |    NULL |
| Sumehar Gill     | 2104364 |
+------------------+---------+
10 rows in set (0.00 sec)
```

D. **RIGHT JOIN:** The RIGHT JOIN keyword returns all records from the right table (table2), and thematching records (if any) from the left table (table1).

```
mysql> SELECT STUDENTS.SID, student.Name
    -> FROM STUDENTS
    -> RIGHT JOIN student on STUDENTS.Branch = student.Branch
    -> ORDER BY STUDENTS.SID;
+---------+------------------+
| SID     | Name             |
+---------+------------------+
|    NULL | Shashank Kumar   |
|    NULL | Jaskaran Singh   |
| 2104364 | Anmolvir Singh   |
| 2104364 | Sumehar Gill     |
| 2104364 | Ekamjot Singh    |
| 2104471 | Anu grewal       |
| 2104471 | Arshnoor         |
| 2104471 | Samiksha         |
| 2104482 | Avlyn Kaur       |
| 2104482 | Dashyamjit Singh |
+---------+------------------+
10 rows in set (0.00 sec)
```

PRACTICAL NO 8: AGGREGATING DATA USING GROUP FUNCTIONS

AIM: Group functions for various statistical metrics; creating groups of data by GROUP BY clause; grouping by more than one column; excluding group results- HAVING Clause.

A. **Group Functions:** Unlike single-row functions, group functions operate on sets of rows to give one result per group. Group functions are also known as aggregate functions.

Types of Group Functions

•AVG

•COUNT

•MAX

•MIN

•SUM

Syntax:

SELECT [column,] group_function(column)

FROM table

[WHERE condition]

[GROUP BY column]

[ORDER BY colunın] ;

- **Using AVG and SUM Functions:**
  We can use AVG and SUM for numeric data.

```
mysql> SELECT AVG(SID), SUM(SID)
    -> FROM student;
+--------------+----------+
| AVG(SID)     | SUM(SID) |
+--------------+----------+
| 2104439.9000 | 21044399 |
+--------------+----------+
1 row in set (0.01 sec)
```

- **Using MIN and MAX Functions**

  We can use MIN and MAX for any datatype.

```
mysql> SELECT min(SID) , max(SID)
    -> FROM student;
+----------+----------+
| min(SID) | max(SID) |
+----------+----------+
|  2104364 |  2104488 |
+----------+----------+
1 row in set (0.01 sec)
```

- **Using the COUNT Function**

  COUNT(*) returns the number of rows in a table.

```
mysql> SELECT COUNT(*)
    -> FROM student;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
1 row in set (0.01 sec)
```

B. Creating Groups of Data: GROUP BY Clause

Divide rows in a table into smaller groups by using the GROUP BY clause.

Syntax:

SELECT column, group_function (column)

FROM table

[WHERE condition]

[GROUP BY group by]

[ORDER BY column];

Example:

```
mysql> SELECT Age, AVG(SID)
    -> FROM student
    -> GROUP BY Age;
+------+--------------+
| Age  | AVG(SID)     |
+------+--------------+
|   19 | 2104478.0000 |
|   21 | 2104418.5000 |
|   20 | 2104412.5000 |
+------+--------------+
3 rows in set (0.00 sec)
```

C. Grouping by More Than One Column

```
mysql> SELECT SID,Name,Branch
    -> FROM student
    -> ORDER BY SID;
+---------+------------------+------------------+
| SID     | Name             | Branch           |
+---------+------------------+------------------+
| 2104364 | Anmolvir Singh   | Electronics      |
| 2104365 | Ekamjot Singh    | Electronics      |
| 2104366 | Jaskaran Singh   | NULL             |
| 2104443 | Shashank Kumar   | Electrical       |
| 2104465 | Samiksha         | Computer Science |
| 2104471 | Anu grewal       | Computer Science |
| 2104477 | Arshnoor         | Computer Science |
| 2104478 | Sumehar Gill     | Electronics      |
| 2104482 | Avlyn Kaur       | IT               |
| 2104488 | Dashyamjit Singh | IT               |
+---------+------------------+------------------+
10 rows in set (0.00 sec)
```

D. Excluding Group Results: HAVING Clause

Use the HAVING clause to restrict groups

- Rows are grouped.

- The group function is applied.
- Groups matching the HAVING clause are displayed.

Syntax:

SELECT column, group_ function

FROM table

[WHERE condition]

[GROUP BY group_by_expression]

[HAVING group_condition]

[ORDER BY column];

Example:

```
mysql> SELECT SID , max(Age)
    -> FROM student
    -> GROUP BY SID
    -> HAVING max(Age) > 19;
+---------+----------+
| SID     | max(Age) |
+---------+----------+
| 2104471 |       21 |
| 2104364 |       20 |
| 2104478 |       20 |
| 2104443 |       20 |
| 2104365 |       20 |
| 2104366 |       21 |
+---------+----------+
6 rows in set (0.00 sec)
```

PRACTICAL NO 9: SUBQUERIES

AIM: Single-row subqueries; multiple-row subqueries; using group function in a subquery; HAVING clause with subqueries; usage of operators in multiple-row subqueries.

A. **Single-row Subqueries:** In this the subquery returns a Single row value or Multiple rows of values from the Subquery given after the 'WHERE' clause of the outer query.

```
mysql> SELECT SID,Name,City
    -> FROM student
    -> WHERE SID = (SELECT SID FROM student WHERE Name = 'Anu grewal');
+---------+-----------+-----------+
| SID     | Name      | City      |
+---------+-----------+-----------+
| 2104471 | Anu grewal | Chandigarh |
+---------+-----------+-----------+
1 row in set (0.01 sec)
```

B. **Multiple –row subqueries**: Returns one or more rows to the outer SQL statement using the IN, ANY, or ALL operator.

```
mysql> SELECT SID, Name, City, Branch
    -> FROM student
    -> WHERE SID IN (
    ->       SELECT SID
    ->       FROM student
    ->       WHERE City = 'Chandigarh'
    -> );
+---------+-----------+-----------+------------------+
| SID     | Name      | City      | Branch           |
+---------+-----------+-----------+------------------+
| 2104471 | Anu grewal | Chandigarh | Computer Science |
| 2104465 | Samiksha   | Chandigarh | Computer Science |
+---------+-----------+-----------+------------------+
2 rows in set (0.01 sec)
```

C. **Using Group Function in a Subquery:**

```
mysql> SELECT Name, SID, Branch
    -> FROM student
    -> WHERE SID = (SELECT MIN(SID)
    ->              FROM student);
+----------------+---------+-------------+
| Name           | SID     | Branch      |
+----------------+---------+-------------+
| Anmolvir Singh | 2104364 | Electronics |
+----------------+---------+-------------+
1 row in set (0.01 sec)
```

D. HAVING Clause with Subqueries:

```
mysql> SELECT SID, MIN(Age)
    -> FROM student
    -> GROUP BY SID
    -> HAVING MIN(Age) >
    ->                   (SELECT MIN(Age)
    ->                    FROM   student
    ->                    WHERE  SID = 2104482);
+---------+----------+
| SID     | MIN(Age) |
+---------+----------+
| 2104471 |       21 |
| 2104364 |       20 |
| 2104478 |       20 |
| 2104443 |       20 |
| 2104365 |       20 |
| 2104366 |       21 |
+---------+----------+
6 rows in set (0.00 sec)
```

PRACTICAL NO 10: CREATING VIEWS

AIM: Simple views and complex views; creating a view; retrieving data from view; querying a view; modifying a view ; removing a view; inline views.

A. Creating a view: We can create a view using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

B. Simple View: Simple view is view created on single table

Syntax:

Create view Viewname

as Select column_name1,Coumn_name2 from tablename.

Example:

```
mysql> Create view V_student
    -> as Select SID,Name from student;
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> select * from V_student;
+---------+-------------------+
| SID     | Name              |
+---------+-------------------+
| 2104482 | Avlyn Kaur        |
| 2104471 | Anu grewal        |
| 2104488 | Dashyamjit Singh  |
| 2104364 | Anmolvir Singh    |
| 2104477 | Arshnoor          |
| 2104465 | Samiksha          |
| 2104478 | Sumehar Gill      |
| 2104443 | Shashank Kumar    |
| 2104365 | Ekamjot Singh     |
| 2104366 | Jaskaran Singh    |
+---------+-------------------+
10 rows in set (0.01 sec)
```

C. **Complex View:** Complex view is created on using more than one tables.

```
mysql> CREATE VIEW studentBYCompanies AS
    -> SELECT student.city,Companies.address,student.Name
    -> FROM student, Companies
    -> WHERE student.City = Companies.address;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from studentBYCompanies;
+------------+------------+-------------------+
| city       | address    | Name              |
+------------+------------+-------------------+
| Ludhiana   | Ludhiana   | Avlyn Kaur        |
| Chandigarh | Chandigarh | Anu grewal        |
| Ludhiana   | Ludhiana   | Dashyamjit Singh  |
| Chandigarh | Chandigarh | Samiksha          |
| Ludhiana   | Ludhiana   | Sumehar Gill      |
| Ludhiana   | Ludhiana   | Ekamjot Singh     |
+------------+------------+-------------------+
6 rows in set (0.02 sec)
```

D. **Modifying a View:** To modify an existing view, we can use the ALTER VIEW statement.

```
mysql> ALTER VIEW studentBYCompanies
    -> AS
    -> SELECT student.city, Companies.address, student.Name
    -> FROM student
    -> INNER JOIN Companies
    -> ON student.City = Companies.address;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from studentBYCompanies;
+------------+------------+------------------+
| city       | address    | Name             |
+------------+------------+------------------+
| Ludhiana   | Ludhiana   | Avlyn Kaur       |
| Chandigarh | Chandigarh | Anu grewal       |
| Ludhiana   | Ludhiana   | Dashyamjit Singh |
| Chandigarh | Chandigarh | Samiksha         |
| Ludhiana   | Ludhiana   | Sumehar Gill     |
| Ludhiana   | Ludhiana   | Ekamjot Singh    |
+------------+------------+------------------+
6 rows in set (0.00 sec)
```

E. **Retrieving a View:** Use the SELECT statement to retrieve data from the view.

```
mysql> select * from studentBYCompanies;
+------------+------------+------------------+
| city       | address    | Name             |
+------------+------------+------------------+
| Ludhiana   | Ludhiana   | Avlyn Kaur       |
| Chandigarh | Chandigarh | Anu grewal       |
| Ludhiana   | Ludhiana   | Dashyamjit Singh |
| Chandigarh | Chandigarh | Samiksha         |
| Ludhiana   | Ludhiana   | Sumehar Gill     |
| Ludhiana   | Ludhiana   | Ekamjot Singh    |
+------------+------------+------------------+
6 rows in set (0.00 sec)
```

F. **Removing a View:** To remove a view in SQL, we can use the DROP VIEW statement.

```
mysql> DROP VIEW studentBYCompanies;
Query OK, 0 rows affected (0.01 sec)
```

This command will delete the view from the database.

G. **Inline views:** Inline views are particularly useful for simplifying complex queries without relying on **JOIN operations** or **subqueries**.

Syntax of Inline view:

```
SELECT column1, column2, ...
FROM (
   -- Your inline view query goes here
   SELECT ...
) AS alias_name;
```

Example:

```
mysql> SELECT MAX(SID) AS max_SID
    -> FROM (
    ->     SELECT SID
    ->     FROM student
    -> ) AS max_SID;
+---------+
| max_SID |
+---------+
| 2104488 |
+---------+
1 row in set (0.00 sec)
```

MINI PROJECT

AIM: Creating a table, inserting the values, updating the table by using no of queries, deleting the columns when not required .illustrate the use of joins. Mandatory to use having clause in database**.**

A. **Creating a table:** To create a new table in a database, we use the CREATE TABLE statement.

```
mysql> CREATE TABLE Employees (
    ->     EmployeeID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     Department VARCHAR(50),
    ->     Salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

B. **Inserting the Values:** Once the table is created, we can insert data into it using the INSERT INTO statement.

```
mysql> INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)
    -> VALUES
    ->     (1, 'John', 'Doe', 'HR', 60000.00),
    ->     (2, 'Jane', 'Smith', 'IT', 75000.00),
    ->     (3, 'Alice', 'Johnson', 'Finance', 90000.00);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

C. **Updating the table by using no of queries:** We can modify existing data using the UPDATE statement.

```
mysql> UPDATE Employees
    -> SET Salary = 65000.00
    -> WHERE EmployeeID = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM EMPLOYEES;
+------------+-----------+----------+------------+----------+
| EmployeeID | FirstName | LastName | Department | Salary   |
+------------+-----------+----------+------------+----------+
|          1 | John      | Doe      | HR         | 65000.00 |
|          2 | Jane      | Smith    | IT         | 75000.00 |
|          3 | Alice     | Johnson  | Finance    | 90000.00 |
+------------+-----------+----------+------------+----------+
3 rows in set (0.01 sec)
```

D. **Deleting the columns when not required:** Suppose we no longer need the "Department" column. We can remove it using the ALTER TABLE statement:

```
mysql> ALTER TABLE Employees
    -> DROP COLUMN Department;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from employees;
+------------+-----------+----------+----------+
| EmployeeID | FirstName | LastName | Salary   |
+------------+-----------+----------+----------+
|          1 | John      | Doe      | 65000.00 |
|          2 | Jane      | Smith    | 75000.00 |
|          3 | Alice     | Johnson  | 90000.00 |
+------------+-----------+----------+----------+
3 rows in set (0.00 sec)
```

E. **Illustrate the use of joins:** Joins allow us to combine data from multiple tables.

```
mysql> SELECT e.FirstName, e.LastName, p.ProjectName
    -> FROM Employees e
    -> INNER JOIN Projects p ON e.Department = p.Department;
```

F. **Having clause in database:** The HAVING clause is used with aggregate functions (e.g., SUM, COUNT, etc.).

```
mysql> SELECT EmployeeID, AVG(Salary) AS AvgSalary
    -> FROM Employees
    -> GROUP BY EmployeeID
    -> HAVING AVG(Salary) > 70000.00;
+------------+--------------+
| EmployeeID | AvgSalary    |
+------------+--------------+
|          2 | 75000.000000 |
|          3 | 90000.000000 |
+------------+--------------+
2 rows in set (0.02 sec)
```