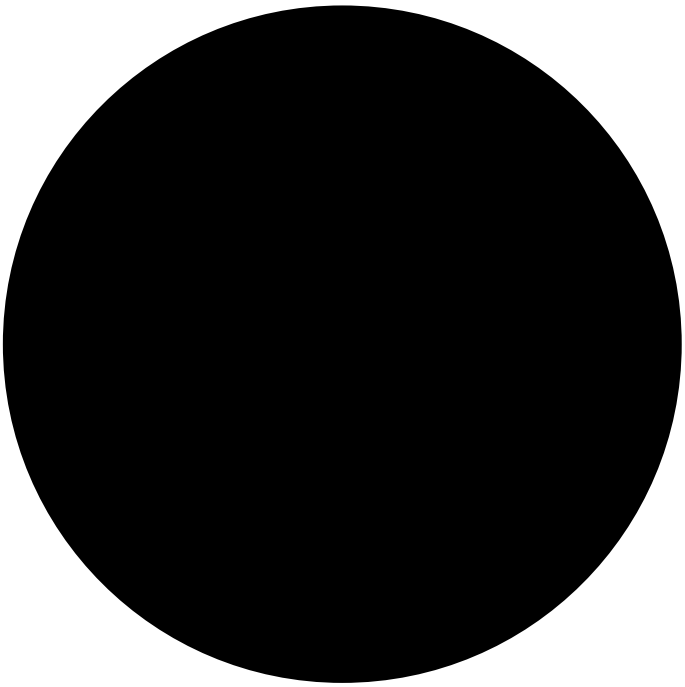
.....

.....

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Database Management Systems

Laboratory Manual

Subject Code: CO-202

SUBMITTED TO:

SUBMITTED BY:

jsklfjdkljfio

(23/CS/5….5)

INDEX

S. No	Aim	Date	Signature
1.	To study about ER diagram	16/01/2025	
2.	To study DDL commands in SQL	23/01/2025	
3.	To implement DML command in SQL	30/01/2025	
4.	To implement Set operations and Clauses like WHERE, GROUP BY, ORDER BY, HAVING in SQL	06/02/2025	
5.	To implement various single row and multi-row functions in SQL	13/02/2025	
6.	Implement various operations on Joins	27/03/2025	
7.	Write a program to implement sub-queries in SQL	10/04/2025	
8.	Write a program to implement views in MySQL	10/04/2025	

Contents

1. To study about ER Diagram.

Object:

To understand and design an Entity-Relationship (ER) diagram for a database system.

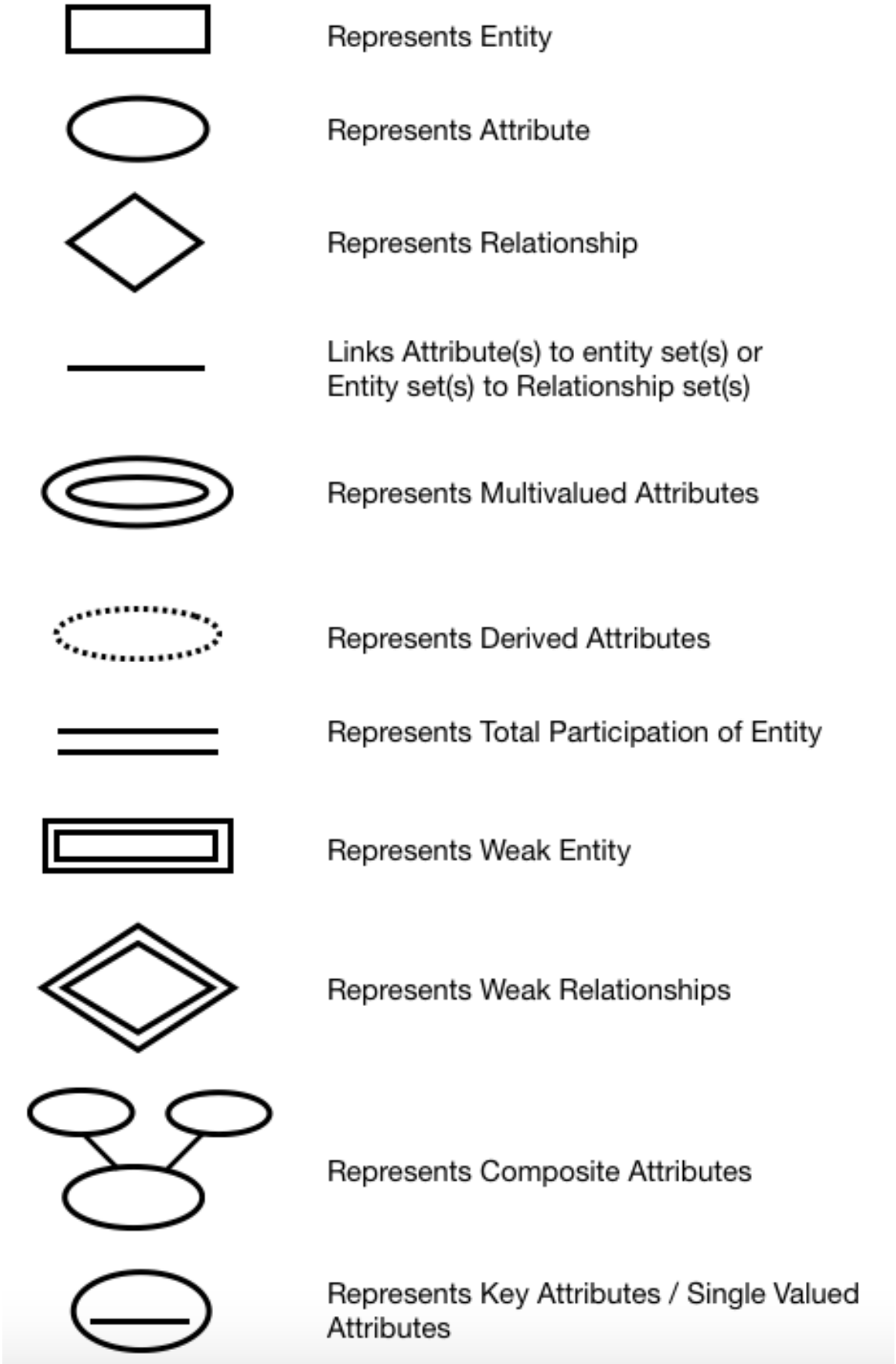
Theory:

An Entity-Relationship (ER) Diagram is a visual representation of data that describes how data is related to each other. It is used in the initial stages of database design to capture the system's data requirements. ER diagrams are the foundation for building relational databases and help in designing a logical structure before implementation.

Key components include:

- **Entities**: Objects or things in the system that have a distinct existence.
- **Attributes**: Characteristics or properties of entities.
- **Primary Key**: A unique attribute used to identify each entity instance uniquely.
- **Relationships**: Associations between entities.
- **Cardinality**: Defines how many instances of one entity relate to instances of another entity.

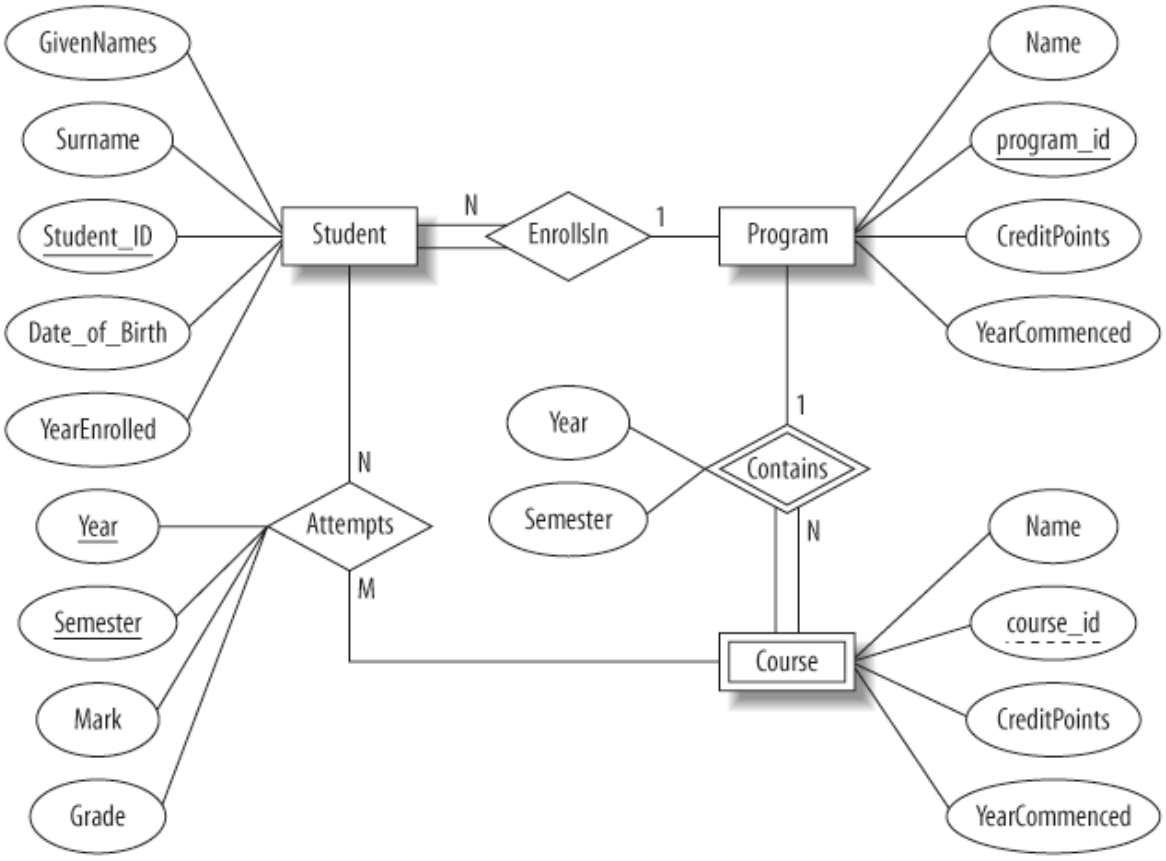
Symbols Used in ER diagram:



Example Case Study:

This database stores information about students, courses, and the degree programs they are enrolled in.

- **Programs**: Each program has a name, unique ID, total credit points, and year of commencement. A program contains one or more courses.
- **Courses**: Each course has a name, unique ID, credit points, and the year it commenced. Courses are assigned to specific years and semesters within a program.
- **Students**: Each student has one or more given names, a surname, student ID, date of birth, and the year they enrolled. A student must be enrolled in one program.
- **Course Attempts**: When a student takes a course, the year and semester of the attempt are recorded. If completed, a grade and mark are also stored.



Conclusion:

ER diagrams are essential for database design. They help identify entities, attributes, and relationships, providing a clear roadmap for creating the database schema. Using ER diagrams ensures better communication between developers and stakeholders.

2. To study DDL commands in SQL.

Object:

To study and implement Data Definition Language (DDL) commands in SQL.

Software Used: MYSQL 5.0

Theory:

DDL commands are used to define and modify database structures. Common DDL commands include:

- **CREATE**: Create a new table
  - **Syntax**:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

- **ALTER**: Modify existing table structure
  - **Syntax**:

```
ALTER TABLE table_name
[ADD | DROP | MODIFY] column_name datatype;
```

- **DROP**: Delete table
  - **Syntax**:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

- **TRUNCATE**: Remove all records (but not structure)
  - **Syntax**:

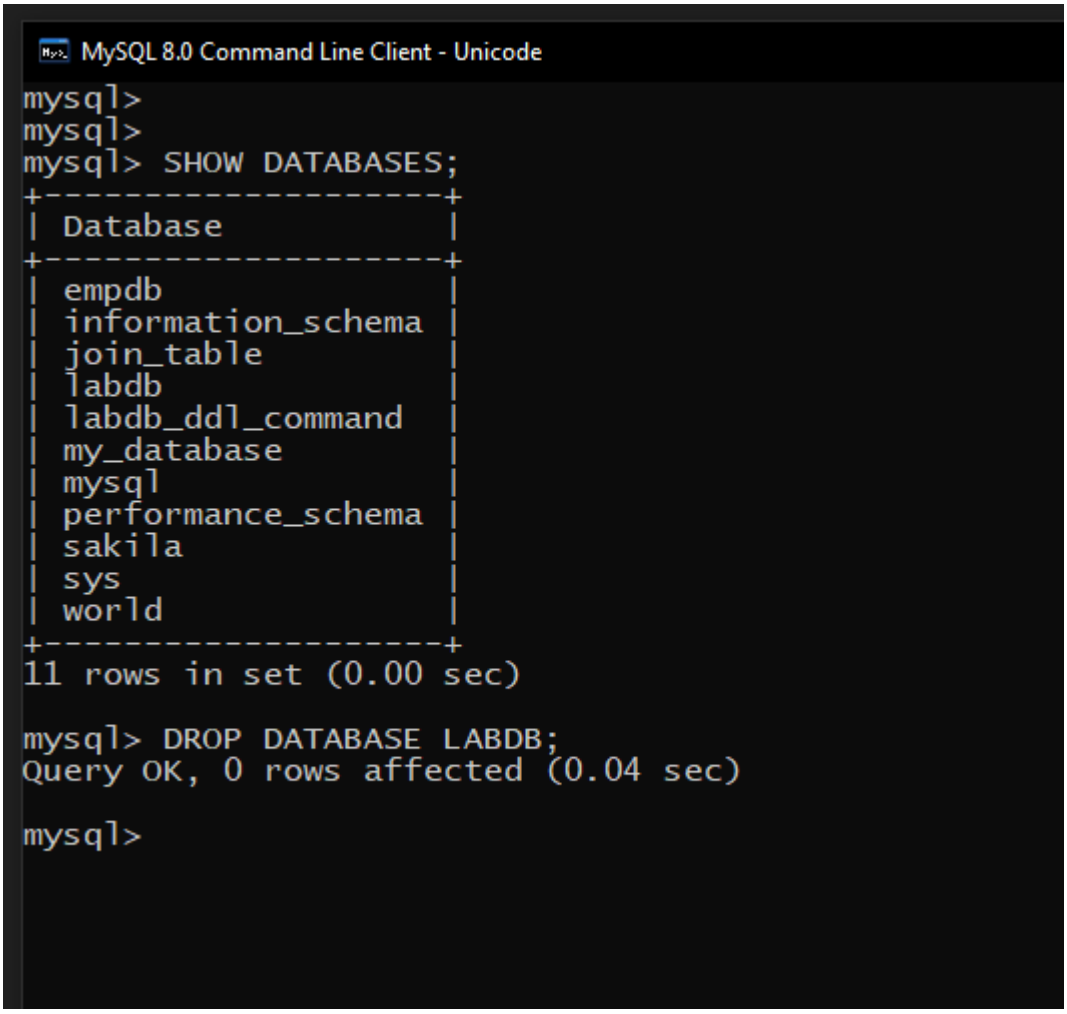
```
TRUNCATE TABLE table_name;
```

Implementation:

```
mysql>  CREATE DATABASE LabDB;
Query OK, 1 row affected (0.01 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| empdb              |
| information_schema |
| join_table         |
| labdb              |
| labdb_ddl_command  |
| my_database        |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
11 rows in set (0.00 sec)

mysql> USE LABDB;
Database changed
mysql>
```

```
mysql> USE LABDB;
Database changed
mysql>  CREATE TABLE Students (
    ->  StudentID INT PRIMARY KEY AUTO_INCREMENT,
    ->  Name VARCHAR(50) NOT NULL,
    ->  Age INT,
    ->  Course VARCHAR(50),
    ->  EnrollmentDate DATE
    ->  );
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_labdb |
+----------------+
| students       |
+----------------+
1 row in set (0.00 sec)

mysql> DESC STUDENTS;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| StudentID      | int         | NO   | PRI | NULL    | auto_increment |
| Name           | varchar(50) | NO   |     | NULL    |                |
| Age            | int         | YES  |     | NULL    |                |
| Course         | varchar(50) | YES  |     | NULL    |                |
| EnrollmentDate | date        | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> DESC STUDENTS;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| StudentID      | int         | NO   | PRI | NULL    | auto_increment |
| Name           | varchar(50) | NO   |     | NULL    |                |
| Age            | int         | YES  |     | NULL    |                |
| Course         | varchar(50) | YES  |     | NULL    |                |
| EnrollmentDate | date        | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql>  ALTER TABLE Students ADD Email VARCHAR(100);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC STUDENTS;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| StudentID      | int          | NO   | PRI | NULL    | auto_increment |
| Name           | varchar(50)  | NO   |     | NULL    |                |
| Age            | int          | YES  |     | NULL    |                |
| Course         | varchar(50)  | YES  |     | NULL    |                |
| EnrollmentDate | date         | YES  |     | NULL    |                |
| Email          | varchar(100) | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql>
```



Conclusion:

DDL commands help define and control the structure of a database. Understanding these commands is crucial for designing, maintaining, and modifying database schemas efficiently.

3. To implement DML commands in SQL.

Object:

To implement Data Manipulation Language (DML) commands to insert, update, and delete data in a database.

Software Used: MYSQL 5.0

Theory:

DML is used to interact with the data stored in relational databases. It allows modification and retrieval of data.

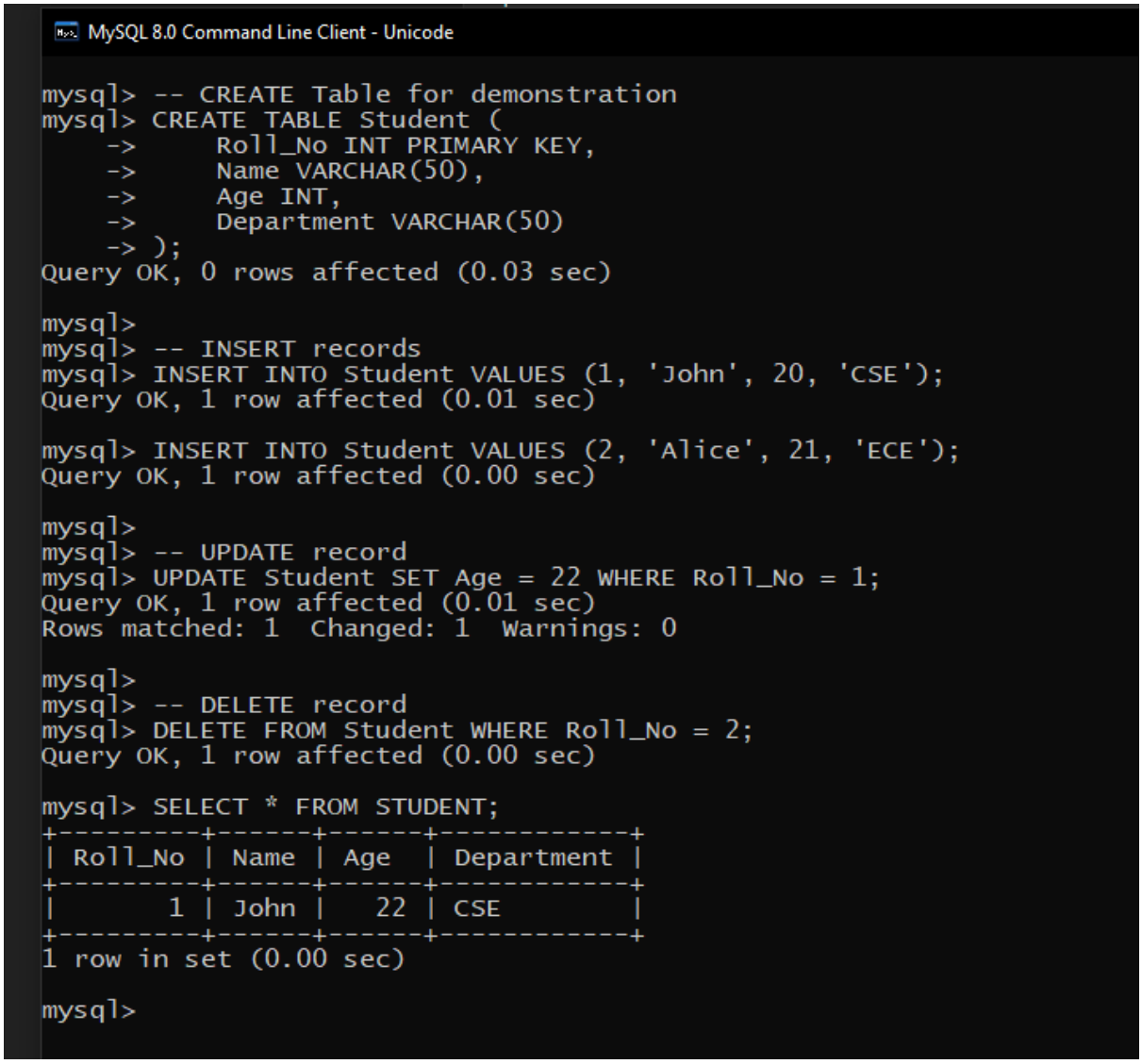
Common DML Commands:

- **INSERT** – Adds new rows to a table.
- **UPDATE** – Modifies existing rows.
- **DELETE** – Removes data from a table.

Implementation:

```
-- CREATE Table for demonstration
CREATE TABLE Student (
    Roll_No INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Department VARCHAR(50)
);

-- INSERT records
INSERT INTO Student VALUES (1, 'John', 20, 'CSE');
INSERT INTO Student VALUES (2, 'Alice', 21, 'ECE');

-- UPDATE record
UPDATE Student SET Age = 22 WHERE Roll_No = 1;

-- DELETE record
DELETE FROM Student WHERE Roll_No = 2;
```

Result:



Conclusion:

DML commands allow manipulation of records in the database, enabling dynamic data handling.

4. To implement set operations and SQL clauses like WHERE, GROUP BY, ORDER BY, and HAVING.

Object:
To implement set operations and SQL clauses like WHERE, GROUP BY, ORDER BY, and HAVING.

Software Used: MYSQL 5.0

Theory:
SQL provides various clauses and set operations to enhance data querying capabilities.

❖ **SQL Clauses:**

- **WHERE**: Filters rows based on condition
- **GROUP BY**: Groups rows with the same values
- **ORDER BY**: Sorts result
- **HAVING**: Applies condition to groups

❖ **Set Operations:**

- **UNION** – Combines results of two SELECT statements (removes duplicates).
- **UNION ALL** – Includes duplicates.
- **INTERSECT** – Returns common rows (not supported in all DBMSs).
- **EXCEPT/MINUS** – Returns rows from the first SELECT not found in the second.

Implementation:

```
-- Creating Employee table
CREATE TABLE Employee (
    ID INT,
    Name VARCHAR(50),
    Department VARCHAR(30),
    Salary INT
);


INSERT INTO Employee VALUES
(1, 'Alice', 'HR', 40000),
(2, 'Bob', 'IT', 60000),
(3, 'Charlie', 'HR', 45000),
(4, 'David', 'IT', 70000);
```

-- WHERE clause

```
SELECT * FROM Employee WHERE Department = 'IT';
```

```
mysql> -- WHERE clause
mysql> SELECT * FROM Employee WHERE Department = 'IT';
+------+-------+------------+--------+
| ID   | Name  | Department | Salary |
+------+-------+------------+--------+
|    2 | Bob   | IT         |  60000 |
|    4 | David | IT         |  70000 |
+------+-------+------------+--------+
2 rows in set (0.00 sec)
```

-- GROUP BY and HAVING

```
SELECT Department, COUNT(*) AS NumEmployees, AVG(Salary) AS AvgSalary
FROM Employee
GROUP BY Department
HAVING AVG(Salary) > 45000;
```

```
mysql> -- GROUP BY and HAVING
mysql> SELECT Department, COUNT(*) AS NumEmployees, AVG(Salary) AS AvgSalary
    -> FROM Employee
    -> GROUP BY Department
    -> HAVING AVG(Salary) > 45000;
+------------+--------------+------------+
| Department | NumEmployees | AvgSalary  |
+------------+--------------+------------+
| IT         |            2 | 65000.0000 |
+------------+--------------+------------+
1 row in set (0.01 sec)
```

-- ORDER BY

```
SELECT * FROM Employee ORDER BY Salary DESC;
```

```
mysql> -- ORDER BY
mysql> SELECT * FROM Employee ORDER BY Salary DESC;
+------+---------+------------+--------+
| ID   | Name    | Department | Salary |
+------+---------+------------+--------+
|    4 | David   | IT         |  70000 |
|    2 | Bob     | IT         |  60000 |
|    3 | Charlie | HR         |  45000 |
|    1 | Alice   | HR         |  40000 |
+------+---------+------------+--------+
4 rows in set (0.00 sec)
```

```
-- Set Operation: UNION
CREATE TABLE FormerEmployees (
    Name VARCHAR(50)
);

INSERT INTO FormerEmployees VALUES ('Eve'), ('Bob');

SELECT Name FROM Employee
UNION
SELECT Name FROM FormerEmployees;
```

```
mysql> -- Set Operation: UNION
mysql> CREATE TABLE FormerEmployees (
    ->     Name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> INSERT INTO FormerEmployees VALUES ('Eve'), ('Bob');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> SELECT Name FROM Employee
    -> UNION
    -> SELECT Name FROM FormerEmployees;
+---------+
| Name    |
+---------+
| Alice   |
| Bob     |
| Charlie |
| David   |
| Eve     |
+---------+
5 rows in set (0.00 sec)
```

Conclusion:

Clauses like **WHERE**, **GROUP BY**, **ORDER BY**, and **HAVING** make SQL queries powerful and flexible. Set operations allow combining and comparing datasets across tables.

5. To implement various Single row and Multiple-row function in SQL.

Object:

To understand and implement single-row and multi-row functions in SQL for data transformation and analysis.

Software Used: MYSQL 5.0

Theory:

❖ **Single Row Functions:**

These functions operate on single rows and return one result per row. Types include:

- String Functions: `UPPER()`, `LOWER()`, `LENGTH()`, `CONCAT()`, `SUBSTR()`, `INSTR()`
- Numeric Functions: `ROUND()`, `CEIL()`, `FLOOR()`, `MOD()`
- Date Functions: `SYSDATE()`, `NOW()`, `DATEDIFF()`, `CURDATE()`
- Conversion Functions: `CAST()`, `CONVERT()`

❖ **Multi-row (Group) Functions:**

These operate on multiple rows and return a single result. Examples:

- `SUM()`, `AVG()`, `MIN()`, `MAX()`, `COUNT()`

Implementation & Result:

```
-- Creating Employee table
CREATE TABLE Employee (
    ID INT,
    Name VARCHAR(50),
    Salary INT,
    JoinDate DATE
);

INSERT INTO Employee VALUES
(1, 'Alice', 50000, '2021-06-10'),
(2, 'Bob', 60000, '2020-01-15'),
(3, 'Charlie', 45000, '2022-03-01');
```

-- Single Row Functions

```
SELECT UPPER(Name) AS UpperName FROM Employee;
```

```
mysql> -- Single Row Functions
mysql> SELECT UPPER(Name) AS UpperName FROM Employee;
+-----------+
| UpperName |
+-----------+
| ALICE     |
| BOB       |
| CHARLIE   |
| DAVID     |
+-----------+
4 rows in set (0.00 sec)
```

```
SELECT LENGTH(Name) AS NameLength FROM Employee;
```

```
mysql> SELECT LENGTH(Name) AS NameLength FROM Employee;
+------------+
| NameLength |
+------------+
|          5 |
|          3 |
|          7 |
|          5 |
+------------+
4 rows in set (0.00 sec)
```

```
SELECT ROUND(Salary/12, 2) AS MonthlySalary FROM Employee;
```

```
mysql> SELECT ROUND(Salary/12, 2) AS MonthlySalary FROM Employee;
+---------------+
| MonthlySalary |
+---------------+
|       3333.33 |
|       5000.00 |
|       3750.00 |
|       5833.33 |
+---------------+
4 rows in set (0.01 sec)
```

```
SELECT YEAR(JoinDate) AS JoinYear FROM Employee;
```

```
-- Multi-row Functions
SELECT COUNT(*) AS TotalEmployees FROM Employee;
SELECT AVG(Salary) AS AverageSalary FROM Employee;
SELECT MAX(Salary) AS MaxSalary FROM Employee;
```

```
mysql> -- Multi-row Functions
mysql> SELECT COUNT(*) AS TotalEmployees FROM Employee;
+----------------+
| TotalEmployees |
+----------------+
|              4 |
+----------------+
1 row in set (0.01 sec)

mysql> SELECT AVG(Salary) AS AverageSalary FROM Employee;
+---------------+
| AverageSalary |
+---------------+
|    53750.0000 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT MAX(Salary) AS MaxSalary FROM Employee;
+-----------+
| MaxSalary |
+-----------+
|     70000 |
+-----------+
1 row in set (0.00 sec)
```

Conclusion:

Single and multi-row functions are powerful tools for data transformation and aggregation, enhancing the querying capabilities of SQL.

6. To Implement Various Operations on Joins.

Object:

To understand and implement different types of SQL joins to retrieve data from multiple tables based on logical relationships.

Software Used: MYSQL 5.0

Theory:

Types of Joins:

- **INNER JOIN**: Returns matching rows from both tables.
  - **Syntax**:

```
SELECT A.*, B.*
FROM TableA A
INNER JOIN TableB B
ON A.common_column = B.common_column;
```

- **LEFT JOIN**: Returns all rows from the left table and matched rows from the right table.
  - **Syntax**:

```
SELECT A.*, B.*
FROM TableA A
LEFT JOIN TableB B
ON A.common_column = B.common_column;
```

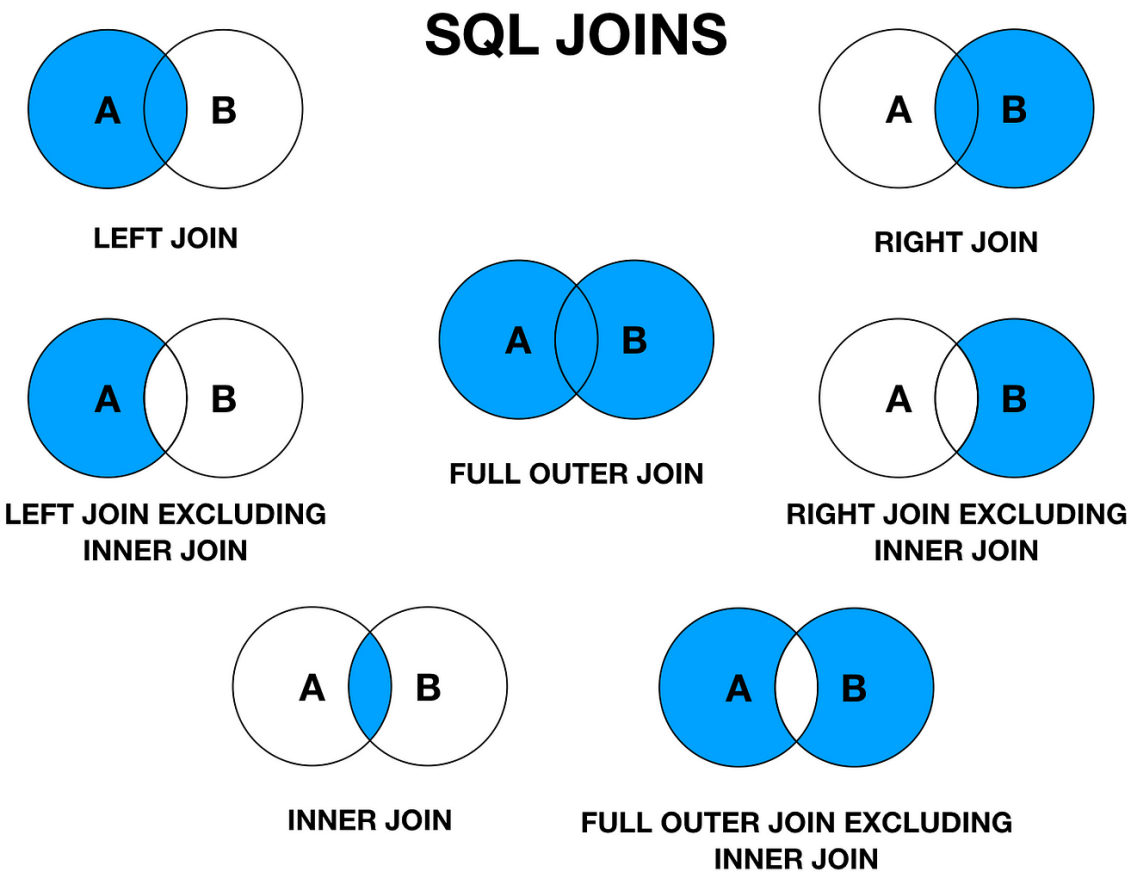
- **RIGHT JOIN**: Returns all rows from the right table and matched rows from the left table.
  - **Syntax**:

```
SELECT A.*, B.*
FROM TableA A
RIGHT JOIN TableB B
ON A.common_column = B.common_column;
```

- **FULL OUTER JOIN**: Returns all rows when there is a match in one of the tables (not supported in MySQL directly).
  - **Syntax**:

```
SELECT A.*, B.*
FROM TableA A
FULL OUTER JOIN TableB B
ON A.common_column = B.common_column;
```

- **LEFT JOIN EXCLUDING INNER JOIN**: Returns only the unmatched rows from the left table.
    - **Syntax**:
      ```
      SELECT A.*
      FROM TableA A
      LEFT JOIN TableB B
      ON A.common_column = B.common_column
      WHERE B.common_column IS NULL;
      ```
- **RIGHT JOIN EXCLUDING INNER JOIN**: Returns only the unmatched rows from the right table.
    - **Syntax**:
      ```
      SELECT B.*
      FROM TableA A
      RIGHT JOIN TableB B
      ON A.common_column = B.common_column
      WHERE A.common_column IS NULL;
      ```
- **FULL OUTER JOIN EXCLUDING INNER JOIN**: Returns only the unmatched rows from both tables.
    - **Syntax**:
      ```
      SELECT A.*, B.*
      FROM TableA A
      FULL OUTER JOIN TableB B
      ON A.common_column = B.common_column
      WHERE A.common_column IS NULL OR B.common_column IS NULL;
      ```



Implementation & Result:

```
-- 1. Creating tables
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    DeptID INT
);

CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50)
);

-- 2. Inserting data into Employees
INSERT INTO Employees (EmpID, Name, DeptID) VALUES
(1, 'Alice', 101),
(2, 'Bob', 102),
(3, 'Charlie', NULL),
(4, 'David', 103),
(5, 'Eva', 105);

-- 3. Inserting data into Departments
INSERT INTO Departments (DeptID, DeptName) VALUES
(101, 'HR'),
(102, 'Finance'),
(103, 'Engineering'),
(104, 'Marketing');

-- 4. INNER JOIN
SELECT E.Name, D.DeptName
FROM Employees E
INNER JOIN Departments D
ON E.DeptID = D.DeptID;
```

```
-- 5. LEFT JOIN
SELECT E.Name, D.DeptName
FROM Employees E
LEFT JOIN Departments D
ON E.DeptID = D.DeptID;

-- 6. RIGHT JOIN
SELECT E.Name, D.DeptName
FROM Employees E
RIGHT JOIN Departments D
ON E.DeptID = D.DeptID;

-- 7. FULL OUTER JOIN (for MySQL use UNION of LEFT and RIGHT)
SELECT E.Name, D.DeptName
FROM Employees E
LEFT JOIN Departments D
ON E.DeptID = D.DeptID
UNION
SELECT E.Name, D.DeptName
FROM Employees E
RIGHT JOIN Departments D
ON E.DeptID = D.DeptID;

-- 8. LEFT JOIN EXCLUDING INNER JOIN
SELECT E.Name
FROM Employees E
LEFT JOIN Departments D
ON E.DeptID = D.DeptID
WHERE D.DeptID IS NULL;

-- 9. RIGHT JOIN EXCLUDING INNER JOIN
```

```
SELECT D.DeptName
FROM Employees E
RIGHT JOIN Departments D
ON E.DeptID = D.DeptID
WHERE E.DeptID IS NULL;

-- 10. FULL OUTER JOIN EXCLUDING INNER JOIN
SELECT E.Name, D.DeptName
FROM Employees E
LEFT JOIN Departments D
ON E.DeptID = D.DeptID
WHERE D.DeptID IS NULL
UNION
SELECT E.Name, D.DeptName
FROM Employees E
RIGHT JOIN Departments D
ON E.DeptID = D.DeptID
WHERE E.DeptID IS NULL;
```

```
mysql> -- Creating Employee table
mysql> CREATE TABLE Employee (
    ->     ID INT,
    ->     Name VARCHAR(50),
    ->     Salary INT,
    ->     JoinDate DATE
    -> );
ERROR 1050 (42S01): Table 'employee' already exists
mysql>
mysql> INSERT INTO Employee VALUES
    -> (1, 'Alice', 50000, '2021-06-10'),
    -> (2, 'Bob', 60000, '2020-01-15'),
    -> (3, 'Charlie', 45000, '2022-03-01');
ERROR 1265 (01000): Data truncated for column 'Salary' at row 1
mysql>
mysql> -- Single Row Functions
mysql> SELECT UPPER(Name) AS UpperName FROM Employee;
+-----------+
| UpperName |
+-----------+
| ALICE     |
| BOB       |
| CHARLIE   |
| DAVID     |
+-----------+
4 rows in set (0.00 sec)

mysql> SELECT LENGTH(Name) AS NameLength FROM Employee;
+------------+
| NameLength |
+------------+
|          5 |
|          3 |
|          7 |
|          5 |
+------------+
4 rows in set (0.00 sec)

mysql> SELECT ROUND(Salary/12, 2) AS MonthlySalary FROM Employee;
+---------------+
| MonthlySalary |
+---------------+
|       3333.33 |
|       5000.00 |
|       3750.00 |
|       5833.33 |
+---------------+
4 rows in set (0.01 sec)
```

```
mysql> SELECT YEAR(JoinDate) AS JoinYear FROM Employee;
ERROR 1054 (42S22): Unknown column 'JoinDate' in 'field list'
mysql>
mysql> -- Multi-row Functions
mysql> SELECT COUNT(*) AS TotalEmployees FROM Employee;
+----------------+
| TotalEmployees |
+----------------+
|              4 |
+----------------+
1 row in set (0.01 sec)

mysql> SELECT AVG(Salary) AS AverageSalary FROM Employee;
+---------------+
| AverageSalary |
+---------------+
|    53750.0000 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT MAX(Salary) AS MaxSalary FROM Employee;
+-----------+
| MaxSalary |
+-----------+
|     70000 |
+-----------+
1 row in set (0.00 sec)

mysql> -- 1. Creating tables
mysql> CREATE TABLE Employees (
    ->     EmpID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     DeptID INT
    -> );
Query OK, 0 rows affected (0.26 sec)

mysql>
mysql> CREATE TABLE Departments (
    ->     DeptID INT PRIMARY KEY,
    ->     DeptName VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> -- 2. Inserting data into Employees
mysql> INSERT INTO Employees (EmpID, Name, DeptID) VALUES
    -> (1, 'Alice', 101),
    -> (2, 'Bob', 102),
    -> (3, 'Charlie', NULL),
    -> (4, 'David', 103),
    -> (5, 'Eva', 105);
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
mysql> -- 3. Inserting data into Departments
mysql> INSERT INTO Departments (DeptID, DeptName) VALUES
    -> (101, 'HR'),
    -> (102, 'Finance'),
    -> (103, 'Engineering'),
    -> (104, 'Marketing');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> -- 4. INNER JOIN
mysql> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> INNER JOIN Departments D
    -> ON E.DeptID = D.DeptID;
+-------+-------------+
| Name  | DeptName    |
+-------+-------------+
| Alice | HR          |
| Bob   | Finance     |
| David | Engineering |
+-------+-------------+
3 rows in set (0.02 sec)

mysql>
mysql> -- 5. LEFT JOIN
mysql> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> LEFT JOIN Departments D
    -> ON E.DeptID = D.DeptID;
+---------+-------------+
| Name    | DeptName    |
+---------+-------------+
| Alice   | HR          |
| Bob     | Finance     |
| Charlie | NULL        |
| David   | Engineering |
| Eva     | NULL        |
+---------+-------------+
5 rows in set (0.00 sec)

mysql>
mysql> -- 6. RIGHT JOIN
mysql> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> RIGHT JOIN Departments D
    -> ON E.DeptID = D.DeptID;
+-------+-------------+
| Name  | DeptName    |
+-------+-------------+
| Alice | HR          |
| Bob   | Finance     |
| David | Engineering |
| NULL  | Marketing   |
+-------+-------------+
4 rows in set (0.00 sec)
```

```
mysql> -- 7. FULL OUTER JOIN (for MySQL use UNION of LEFT and RIGHT)
mysql> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> LEFT JOIN Departments D
    -> ON E.DeptID = D.DeptID
    -> UNION
    -> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> RIGHT JOIN Departments D
    -> ON E.DeptID = D.DeptID;
+---------+-------------+
| Name    | DeptName    |
+---------+-------------+
| Alice   | HR          |
| Bob     | Finance     |
| Charlie | NULL        |
| David   | Engineering |
| Eva     | NULL        |
| NULL    | Marketing   |
+---------+-------------+
6 rows in set (0.01 sec)

mysql>
mysql> -- 8. LEFT JOIN EXCLUDING INNER JOIN
mysql> SELECT E.Name
    -> FROM Employees E
    -> LEFT JOIN Departments D
    -> ON E.DeptID = D.DeptID
    -> WHERE D.DeptID IS NULL;
+---------+
| Name    |
+---------+
| Charlie |
| Eva     |
+---------+
2 rows in set (0.00 sec)

mysql>
mysql> -- 9. RIGHT JOIN EXCLUDING INNER JOIN
mysql> SELECT D.DeptName
    -> FROM Employees E
    -> RIGHT JOIN Departments D
    -> ON E.DeptID = D.DeptID
    -> WHERE E.DeptID IS NULL;
+-----------+
| DeptName  |
+-----------+
| Marketing |
+-----------+
1 row in set (0.00 sec)
```

```
mysql> -- 10. FULL OUTER JOIN EXCLUDING INNER JOIN
mysql> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> LEFT JOIN Departments D
    -> ON E.DeptID = D.DeptID
    -> WHERE D.DeptID IS NULL
    -> UNION
    -> SELECT E.Name, D.DeptName
    -> FROM Employees E
    -> RIGHT JOIN Departments D
    -> ON E.DeptID = D.DeptID
    -> WHERE E.DeptID IS NULL;
+---------+-----------+
| Name    | DeptName  |
+---------+-----------+
| Charlie | NULL      |
| Eva     | NULL      |
| NULL    | Marketing |
+---------+-----------+
3 rows in set (0.01 sec)
```

Conclusion:

This experiment demonstrated how SQL JOIN operations can be used to fetch combined data from multiple tables based on related keys. Understanding these joins is crucial for real-world database queries involving relationships between entities.

7. Write a Program to Implement Subqueries in SQL.

Object:

To implement and understand subqueries (nested queries) in SQL for complex data retrieval.

Software Used: MYSQL 5.0

Theory:

A subquery is a query nested inside another query. It can return single or multiple rows and may be used with operators like **`IN`**, **`ANY`**, **`ALL`**, **`EXISTS`**.

Types of Subqueries:

- Single Row Subquery
- Multiple Row Subquery
- Correlated Subquery – references outer query column.

Table for Implementation:

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Marks INT
);

CREATE TABLE Subjects (
    SubjectID INT PRIMARY KEY,
    Name VARCHAR(50),
    MaxMarks INT
);
```

Data Insertion:

```
INSERT INTO Students VALUES
(1, 'Alice', 20, 85),
(2, 'Bob', 21, 78),
(3, 'Charlie', 22, 90),
(4, 'David', 20, 60),
(5, 'Eva', 21, 92);

INSERT INTO Subjects VALUES
```

```
(101, 'Maths', 100),
(102, 'Physics', 100),
(103, 'Chemistry', 100);
```

Implementation:

- **Subquery using IN**

```
SELECT Name FROM Students
WHERE Marks IN (SELECT Marks FROM Students WHERE Marks > 80);
```

- **Subquery using ANY**

```
SELECT Name FROM Students
WHERE Marks > ANY (SELECT Marks FROM Students WHERE Age = 20);
```

- **Subquery using ALL**

```
SELECT Name FROM Students
WHERE Marks > ALL (SELECT Marks FROM Students WHERE Age = 22);
```

- **Subquery using EXISTS**

```
SELECT Name FROM Students S
WHERE EXISTS (SELECT * FROM Subjects WHERE MaxMarks = 100);
```

- **Nested Subquery**

```
SELECT Name FROM Students
WHERE Marks = (SELECT MAX(Marks) FROM Students);
```

- **Correlated Subquery**

```
SELECT Name, Marks FROM Students S1
WHERE Marks > (SELECT AVG(Marks) FROM Students S2 WHERE S1.Age =
S2.Age);
```

Result:

```
mysql> CREATE TABLE Students (
    ->     StudentID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     Age INT,
    ->     Marks INT
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> CREATE TABLE Subjects (
    ->     SubjectID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     MaxMarks INT
    -> );
ERROR 1050 (42S01): Table 'subjects' already exists
mysql> INSERT INTO Students VALUES
    -> (1, 'Alice', 20, 85),
    -> (2, 'Bob', 21, 78),
    -> (3, 'Charlie', 22, 90),
    -> (4, 'David', 20, 60),
    -> (5, 'Eva', 21, 92);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO Subjects VALUES
    -> (101, 'Maths', 100),
    -> (102, 'Physics', 100),
    -> (103, 'Chemistry', 100);
ERROR 1062 (23000): Duplicate entry '101' for key 'subjects.PRIMARY'
mysql> SELECT Name FROM Students
    -> WHERE Marks IN (SELECT Marks FROM Students WHERE Marks > 80);
+---------+
| Name    |
+---------+
| Alice   |
| Charlie |
| Eva     |
+---------+
3 rows in set (0.01 sec)

mysql> SELECT Name FROM Students
    -> WHERE Marks > ANY (SELECT Marks FROM Students WHERE Age = 20);
+---------+
| Name    |
+---------+
| Alice   |
| Bob     |
| Charlie |
| Eva     |
+---------+
4 rows in set (0.01 sec)
```

```
mysql> SELECT Name FROM Students
    -> WHERE Marks > ALL (SELECT Marks FROM Students WHERE Age = 22);
+------+
| Name |
+------+
| Eva  |
+------+
1 row in set (0.00 sec)

mysql> SELECT Name FROM Students S
    -> WHERE EXISTS (SELECT * FROM Subjects WHERE MaxMarks = 100);
+---------+
| Name    |
+---------+
| Alice   |
| Bob     |
| Charlie |
| David   |
| Eva     |
+---------+
5 rows in set (0.00 sec)

mysql> SELECT Name FROM Students
    -> WHERE Marks = (SELECT MAX(Marks) FROM Students);
+------+
| Name |
+------+
| Eva  |
+------+
1 row in set (0.00 sec)

mysql> SELECT Name, Marks FROM Students S1
    -> WHERE Marks > (SELECT AVG(Marks) FROM Students S2 WHERE S1.Age = S2.Age);
+-------+-------+
| Name  | Marks |
+-------+-------+
| Alice |    85 |
| Eva   |    92 |
+-------+-------+
2 rows in set (0.00 sec)
```

Conclusion:

This experiment successfully demonstrates how subqueries in SQL can be used to fetch data based on dynamic conditions. Subqueries are powerful tools for writing complex and efficient queries.

8. To study about ER Diagram.

Object:

To understand and implement Views in MySQL using **CREATE VIEW**, **UPDATE VIEW**, **DROP VIEW**, and query operations on views.

Software Used: MYSQL 5.0

Theory:

A **View** is a **virtual table** based on the result of a SQL query. It contains rows and columns just like a real table, but it does not store data physically. Views are used to:

- Simplify complex queries
- Enhance security by restricting access to specific rows/columns
- Present data in a specific format

❖ Key Points:

- Views can be created using one or more tables.
- You can **query**, **update**, or delete from views (with limitations).
- A view does **not store data**, it dynamically fetches it.

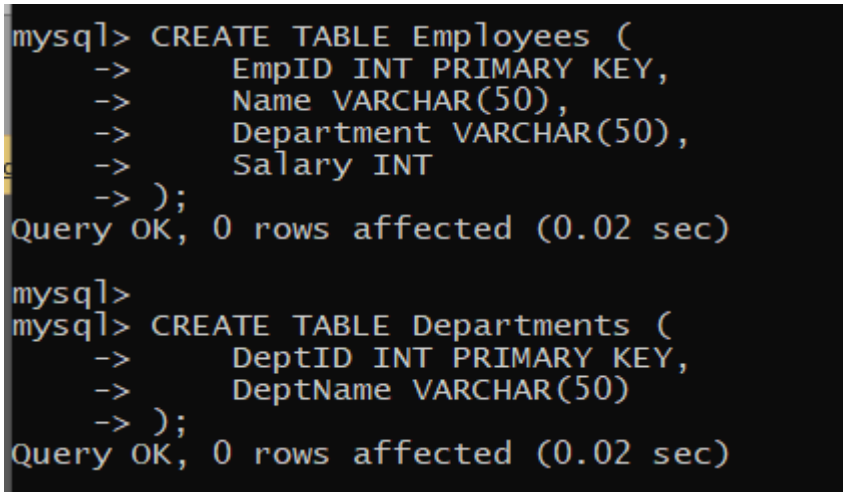
Syntax:

```
-- Create View
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;

-- Update View (re-create with OR REPLACE)
CREATE OR REPLACE VIEW view_name AS
SELECT ...;

-- Drop View
DROP VIEW view_name;

-- Query a View
SELECT * FROM view_name;
```

Implementation:

❖ Create Base Tables:

```
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Department VARCHAR(50),
    Salary INT
);

CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50)
);
```



❖ Insert Sample Data:

```
INSERT INTO Employees VALUES
(1, 'Alice', 'IT', 75000),
(2, 'Bob', 'HR', 65000),
(3, 'Charlie', 'IT', 80000),
(4, 'David', 'Finance', 72000),
(5, 'Eva', 'HR', 67000);

INSERT INTO Departments VALUES
(1, 'IT'),
```

```
(2, 'HR'),
(3, 'Finance');
```

```
mysql> INSERT INTO Departments VALUES
    -> (1, 'IT'),
    -> (2, 'HR'),
    -> (3, 'Finance');
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE VIEW IT_Employees AS
    -> SELECT EmpID, Name, Salary
    -> FROM Employees
    -> WHERE Department = 'IT';
Query OK, 0 rows affected (0.04 sec)
```

- **Create Simple View (IT Employees Only):**

```
CREATE VIEW IT_Employees AS
SELECT EmpID, Name, Salary
FROM Employees
WHERE Department = 'IT';
```

```
mysql> CREATE VIEW IT_Employees AS
    -> SELECT EmpID, Name, Salary
    -> FROM Employees
    -> WHERE Department = 'IT';
Query OK, 0 rows affected (0.04 sec)
```

- **Query the View:**

```
SELECT * FROM IT_Employees;
```

```
mysql> SELECT * FROM IT_Employees;
+-------+---------+--------+
| EmpID | Name    | Salary |
+-------+---------+--------+
|     1 | Alice   |  75000 |
|     3 | Charlie |  80000 |
+-------+---------+--------+
2 rows in set (0.01 sec)
```

- **Create a View with Join:**

```
CREATE VIEW Employee_Department AS
SELECT E.EmpID, E.Name, E.Department, D.DeptName
FROM Employees E
JOIN Departments D ON E.Department = D.DeptName;
```

```
mysql> CREATE VIEW Employee_Department AS
    -> SELECT E.EmpID, E.Name, E.Department, D.DeptName
    -> FROM Employees E
    -> JOIN Departments D ON E.Department = D.DeptName;
Query OK, 0 rows affected (0.01 sec)
```

❖ **Create View Showing High Salary Employees:**

```
CREATE VIEW High_Salary AS

SELECT * FROM Employees

WHERE Salary > 70000;
```

```
mysql> CREATE VIEW High_Salary AS
    -> SELECT * FROM Employees
    -> WHERE Salary > 70000;
Query OK, 0 rows affected (0.01 sec)
```

❖ **Update Data Through View (Only if View is updatable):**

```
UPDATE High_Salary

SET Salary = 90000

WHERE EmpID = 1;
```

```
mysql> UPDATE High_Salary
    -> SET Salary = 90000
    -> WHERE EmpID = 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

❖ **Drop a View:**

```
UPDATE High_Salary

SET Salary = 90000

WHERE EmpID = 1;
```

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

mysql>
```

Conclusion:

This experiment successfully implements the concept of Views in MySQL. Views are powerful tools that allow users to manage complexity, enforce security, and provide customized data access without modifying the underlying tables.