Exercise 1: Database Creation and Data Definition Operations

AIM:

To create a database and perform data definition operations such as creating tables, defining primary and foreign keys using SQL commands in MySQL

Procedure:

Step 1: Open MySQL Command Line Interface

Step 2: Create a new database using the `CREATE DATABASE` command.

Step 3: Select the database using the `USE` command.

Step 4: Create the `Departments` table with `DepartmentID` as the primary key.

Step 5: Create the `Students` table with a foreign key reference to `Departments`.

Step 6: Use `SHOW TABLES` and `DESCRIBE` to verify the structure of both tables.

Step 7: Execute all commands sequentially and check output

```
-- Step 1: Create the Database
CREATE DATABASE UniversityDB;

-- Step 2: Use the Database
USE UniversityDB;

-- Step 3: Create Departments Table
CREATE TABLE Departments (
    DepartmentID INT AUTO_INCREMENT PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL
);

-- Step 4: Create Students Table
CREATE TABLE Students (
    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DOB DATE,
    Gender ENUM('Male', 'Female', 'Other'),
```

```
    Email VARCHAR(100) UNIQUE,
    ContactNumber VARCHAR(15),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

-- Step 5: Verify Tables
SHOW TABLES;
DESCRIBE Students;
DESCRIBE Departments;
```

```
mysql> CREATE DATABASE UniversityDB;
Query OK, 1 row affected

mysql> USE UniversityDB;
Database changed

mysql> SHOW TABLES;
+--------------------+
| Tables_in_UniversityDB |
+--------------------+
| Departments        |
| Students           |
+--------------------+

mysql> DESCRIBE Students;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| StudentID     | int          | NO   | PRI | NULL    | auto_increment |
| FirstName     | varchar(50)  | YES  |     | NULL    |                |
| LastName      | varchar(50)  | YES  |     | NULL    |                |
| DOB           | date         | YES  |     | NULL    |                |
| Gender        | enum(...)    | YES  |     | NULL    |                |
| Email         | varchar(100) | YES  | UNI | NULL    |                |
| ContactNumber | varchar(15)  | YES  |     | NULL    |                |
| DepartmentID  | int          | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+


mysql> DESCRIBE Departments;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| DepartmentID  | int          | NO   | PRI | NULL    | auto_increment |
| DepartmentName| varchar(100) | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+

mysql> SHOW COLUMNS FROM Departments LIKE 'DepartmentID';
```

Output:

```
+--------------+---------+------+-----+---------+----------------+
| Field        | Type    | Null | Key | Default | Extra          |
+--------------+---------+------+-----+---------+----------------+
| DepartmentID | int     | NO   | PRI | NULL    | auto_increment |
+--------------+---------+------+-----+---------+----------------+
```

Insert Sample Data into `Departments` Table

```
INSERT INTO Departments (DepartmentName) VALUES
('Computer Science'),
('Mathematics'),
('Physics'),
('Chemistry'),
('Statistics')
```

Insert Sample Data into `Students` Table

```
INSERT INTO Students (FirstName, LastName, DOB, Gender, Email, ContactNumber,
DepartmentID) VALUES

 ('Abi', 'K', '1969-11-20', 'Female', 'abi.karthi@gmail.com', '9876543211', 101),

('Sree', 'T', '2004-06-11', 'Female', 'sree.t@au.edu', '9876543212', 103),

('David', 'Brown', '2000-08-10', 'Male', 'david.brown@example.com', '9876543213',
104),

('Kavitha', 'R', '1978-12-05', 'Female', 'rgkavitha@yahoo.com', '9876543214', 102),

('Kannan', 'T R T', '1959-04-13', 'Male', 'kannan@example.com', '9876543215', 101);
```

Select First 5 Rows from `Students` Table

```
SELECT * FROM Students LIMIT 5; mysql> SELECT * FROM Students LIMIT 5;
+-----------+-----------+----------+-----------+--------+---------------------------+---------------+--------------+
| StudentID | FirstName | LastName | DOB       | Gender | Email                     | ContactNumber | DepartmentID |
+-----------+-----------+----------+-----------+--------+---------------------------+---------------+--------------+
|         1 | Kohul     | T        | 2000-05-14 | Male   | kohul.t@tn.gov            | 9876543210    | 2            |
|         2 | Abi       | K        | 1969-11-20 | Female | abi.karthi@gmail.com      | 9876543211    | 1            |
|         3 | Sree      | T        | 2004-06-11 | Female | sree.t@au.edu             | 9876543212    | 3            |
|         4 | David     | Brown    | 2000-08-10 | Male   | david.brown@example.com   | 9876543213    | 4            |
|         5 | Kavitha   | R        | 1978-12-05 | Female | rgkavitha@yahoo.com       | 9876543214    | 2            |
+-----------+-----------+----------+-----------+--------+---------------------------+---------------+--------------+
5 rows in set (0.00 sec)
```

Result

1. Created a database named UniversityDB
2. Created two tables (Departments and Students) with proper primary and foreign key relationships
3. Inserted sample data into both tables
4. Verified the table structures and data

Exercise 2: Database using Basic Queries, Join, Update, Delete Operations and Aggregate Functions

AIM

To perform basic queries, joins, update, delete operations, and aggregate functions using SQL commands in MySQL.

PROCEDURE

Step 1: Open MySQL Command Line Interface.
Step 2: Select the database **UniversityDB** using the `USE` command.
Step 3: Execute the following SQL queries:

- **Basic Queries**: Select all records, filter with conditions.
- **Join Operations**: Perform inner join and left join.
- **Aggregate Functions**: Use `COUNT`, `AVG`, and `GROUP BY`.
- **Update Operations**: Modify records using `UPDATE`.
- **Delete Operations**: Remove records using `DELETE`.

Step 4: Verify the outputs after each query.
Step 5: Record the results.

INPUT & OUTPUT

1. Basic Queries

```
-- Select all students
SELECT * FROM Students;

-- Select specific columns
SELECT StudentID, FirstName, LastName, Email FROM Students;

-- Students in Computer Science department (DepartmentID = 1)
SELECT * FROM Students WHERE DepartmentID = 1;

-- Female students
SELECT * FROM Students WHERE Gender = 'Female';

-- Students born after 2000
SELECT * FROM Students WHERE DOB > '2000-01-01';
```

Sample Output:

StudentID	FirstName	LastName	DOB	Gender	Email	ContactNumber	DepartmentID
2	Abi	K	1969-11-20	Female	abi.karthi@gmail.com	9876543211	1
3	Sree	T	2004-06-11	Female	sree.t@au.edu	9876543212	3

2. Join Operations

```
-- Inner Join
SELECT s.StudentID, s.FirstName, s.LastName, d.DepartmentName
FROM Students s
INNER JOIN Departments d ON s.DepartmentID = d.DepartmentID;

-- Left Join
SELECT d.DepartmentName, s.FirstName, s.LastName
FROM Departments d
LEFT JOIN Students s ON d.DepartmentID = s.DepartmentID;
```

Sample Output (Inner Join):

StudentID	FirstName	LastName	DepartmentName
2	Abi	K	Computer Science
3	Sree	T	Physics

3. Aggregate Functions

```
-- Count students by department
SELECT d.DepartmentName, COUNT(s.StudentID) AS StudentCount
FROM Departments d
LEFT JOIN Students s ON d.DepartmentID = s.DepartmentID
GROUP BY d.DepartmentName;

-- Count students by gender
SELECT Gender, COUNT(*) AS Count
FROM Students
GROUP BY Gender;

-- Average age of students
SELECT AVG(YEAR(CURDATE()) - YEAR(DOB)) AS AverageAge
FROM Students;
```

Sample Output (Count by Gender):

```
Gender Count

Male   2

Female 3
```

4. Update Operations

```
-- Update email
UPDATE Students
SET Email = 'new.email@example.com'
WHERE StudentID = 1;

-- Change department
UPDATE Students
SET DepartmentID = 2
WHERE StudentID = 3;

-- Update multiple fields
UPDATE Students
SET ContactNumber = '9998887777', Email = 'updated.email@example.com'
WHERE StudentID = 5;
```

Output:

```
Query OK, 1 row affected
```

5. Delete Operations

```
-- Delete a student
DELETE FROM Students WHERE StudentID = 6;
```

Output:

```
Query OK, 1 row affected
```

RESULT

1. Executed **basic queries** to filter and retrieve student data.
2. Performed **inner join and left join** between Students and Departments.
3. Applied **aggregate functions** (COUNT, AVG) with `GROUP BY`.
4. Successfully **updated student records** using `UPDATE`.
5. Deleted a record from the Students table using `DELETE`.

1. BASIC QUERIES

Select all students:

```sql
SELECT * FROM Students;
```

Select specific columns:

```sql
SELECT StudentID, FirstName, LastName, Email FROM Students;
```

Filter with WHERE clause:

```sql
-- Students in Computer Science department (DepartmentID = 1)
SELECT * FROM Students WHERE DepartmentID = 1;

-- Female students
SELECT * FROM Students WHERE Gender = 'Female';

-- Students born after 2000
SELECT * FROM Students WHERE DOB > '2000-01-01';
```

2. JOIN OPERATIONS

Inner Join to get student names with department names:

```sql
SELECT s.StudentID, s.FirstName, s.LastName, d.DepartmentName
FROM Students s
INNER JOIN Departments d ON s.DepartmentID = d.DepartmentID;
```

Left Join to show all departments even if they have no students:

sql

```sql
SELECT d.DepartmentName, s.FirstName, s.LastName
FROM Departments d
LEFT JOIN Students s ON d.DepartmentID = s.DepartmentID;
```

3. AGGREGATE FUNCTIONS

Count students by department:

sql

```sql
SELECT d.DepartmentName, COUNT(s.StudentID) AS StudentCount
FROM Departments d
LEFT JOIN Students s ON d.DepartmentID = s.DepartmentID
GROUP BY d.DepartmentName;
```

Count students by gender:

sql

```sql
SELECT Gender, COUNT(*) AS Count
FROM Students
GROUP BY Gender;
```

Find average age of students:

sql

```sql
SELECT AVG(YEAR(CURDATE()) - YEAR(DOB)) AS AverageAge
FROM Students;
```

4. UPDATE OPERATIONS

Update a student's email:

sql

```sql
UPDATE Students
SET Email = 'new.email@example.com'
WHERE StudentID = 1;
```

Change a student's department:

sql

```sql
UPDATE Students
SET DepartmentID = 2
WHERE StudentID = 3;
```

Update multiple fields:

sql

```sql
UPDATE Students
SET ContactNumber = '9998887777', Email = 'updated.email@example.com'
WHERE StudentID = 5;
```

5. DELETE OPERATIONS

Delete a student record:

sql

```sql
DELETE FROM Students WHERE StudentID = 6;
```

6. ADVANCED QUERIES

Find students with specific pattern in name:

sql

```sql
-- Students whose first name starts with 'K'
SELECT * FROM Students WHERE FirstName LIKE 'K%';
```

Students with specific birth year:

sql

```sql
SELECT * FROM Students WHERE YEAR(DOB) = 2000;
```

Order students by last name:

sql

```sql
SELECT * FROM Students ORDER BY LastName;
```

Order by multiple criteria:

sql

```sql
SELECT * FROM Students ORDER BY DepartmentID, LastName, FirstName;
```

7. COMPLEX DATA MANIPULATION

Add a new column to Students table:

sql

```sql
ALTER TABLE Students ADD COLUMN EnrollmentDate DATE;
```

Update the new column with values:

sql

```sql
UPDATE Students SET EnrollmentDate = '2023-09-01' WHERE StudentID = 1;
UPDATE Students SET EnrollmentDate = '2022-08-15' WHERE StudentID = 2;
```

```sql
UPDATE Students SET EnrollmentDate = '2023-09-01' WHERE StudentID = 3;
UPDATE Students SET EnrollmentDate = '2021-07-20' WHERE StudentID = 4;
UPDATE Students SET EnrollmentDate = '2020-06-10' WHERE StudentID = 5;
```

Create a view for frequently used query:

sql

```sql
CREATE VIEW StudentDetails AS
SELECT s.StudentID, s.FirstName, s.LastName, s.DOB, s.Gender, s.Email,
       s.ContactNumber, d.DepartmentName, s.EnrollmentDate
FROM Students s
INNER JOIN Departments d ON s.DepartmentID = d.DepartmentID;
```

Query the view:

sql

```sql
SELECT * FROM StudentDetails;
```

Find students by enrollment year:

sql

```sql
SELECT * FROM StudentDetails WHERE YEAR(EnrollmentDate) = 2023;
```

8. TRANSACTION MANAGEMENT

Using transactions for multiple operations:

sql

```sql
START TRANSACTION;

-- Update student record
UPDATE Students SET ContactNumber = '1112223333' WHERE StudentID = 1;

-- Insert a new student
INSERT INTO Students (FirstName, LastName, DOB, Gender, Email, ContactNumber,
DepartmentID, EnrollmentDate)
VALUES ('New', 'Student', '2001-03-15', 'Male', 'new.student@example.com',
'4445556666', 2, '2023-09-01');

-- Commit the transaction
COMMIT;
```

9. DATA VALIDATION QUERIES

Find duplicate emails:

sql

```sql
SELECT Email, COUNT(*)
FROM Students
GROUP BY Email
HAVING COUNT(*) > 1;
```

Find students with invalid department references:

sql

```sql
SELECT s.*
FROM Students s
LEFT JOIN Departments d ON s.DepartmentID = d.DepartmentID
WHERE d.DepartmentID IS NULL;
```

10. PERFORMANCE OPTIMIZATION

Create indexes for frequently queried columns:

sql

```sql
CREATE INDEX idx_students_department ON Students(DepartmentID);
CREATE INDEX idx_students_email ON Students(Email);
CREATE INDEX idx_students_dob ON Students(DOB);
```

These operations cover a wide range of SQL functionalities from basic queries to more complex data manipulation. You can modify these examples based on your specific needs or explore additional SQL features as you become more comfortable with the database operations