Student–Society Database

Entering and Exiting MySQL

Enter MySQL:

```
sudo mysql -u root -p
```

Exit MySQL:

```
exit;
```

Database Schema Creation

Create Database

```
CREATE DATABASE CollegeDB;
USE CollegeDB;
```

Explanation:

- **CREATE DATABASE** creates a new database.
- **USE CollegeDB** selects it so all commands work inside it.

Create Tables

```
CREATE TABLE STUDENT(
    RollNo CHAR(6) PRIMARY KEY,
    StudentName VARCHAR(20),
    Course VARCHAR(10),
    DOB DATE
);

CREATE TABLE SOCIETY(
    SocID CHAR(6) PRIMARY KEY,
    SocName VARCHAR(20),
```

```
    MentorName VARCHAR(15),
    TotalSeats INT UNSIGNED
);

CREATE TABLE ENROLLMENT(
    RollNo CHAR(6),
    SID CHAR(6),
    DateOfEnrollment DATE,
    FOREIGN KEY (RollNo) REFERENCES STUDENT(RollNo),
    FOREIGN KEY (SID) REFERENCES SOCIETY(SocID)
);
```

Explanation:

- **Primary keys** uniquely identify each row.
- **Foreign keys** link ENROLLMENT to STUDENT and SOCIETY.

Insert Sample Entries into Tables

Insert Records into STUDENT

```
INSERT INTO STUDENT VALUES
('S101', 'Ananya', 'computerscience', '2002-05-11'),
('S102', 'Aarav', 'chemistry', '2001-12-20'),
('S103', 'Megha', 'physics', '2003-03-15'),
('S104', 'Zoya', 'computerscience', '2000-11-02'),
('S105', 'Xian', 'chemistry', '2002-01-30');
```

Insert Records into SOCIETY

```
INSERT INTO SOCIETY VALUES
('S01', 'Debating', 'Gupta', 20),
('S02', 'Dancing', 'Mehta', 30),
('S03', 'Sashakt', 'Kumar', 25),
('NSS', 'Sports', 'Amit Gupta', 40);
```

Insert Records into ENROLLMENT

```
INSERT INTO ENROLLMENT VALUES
('S101', 'S01', '2024-01-10'),
('S102', 'S01', '2024-01-12'),
```

```
('S103', 'S02', '2024-02-05'),
('S101', 'S03', '2024-02-15'),
('S104', 'S02', '2024-03-01'),
('S105', 'NSS', '2024-03-10');
```

Explanation:

- These sample entries allow all SELECT, JOIN, UPDATE, and GROUP BY queries to run properly.
- Foreign key order is maintained: STUDENT → SOCIETY → ENROLLMENT.

SQL Queries With Explanations

1. Retrieve names of students enrolled in any society

```
SELECT StudentName
FROM STUDENT
WHERE RollNo IN (SELECT RollNo FROM ENROLLMENT);
```

Explanation:

- **FROM STUDENT** – we need student names.
- **WHERE RollNo IN (...)** – selects only students who enrolled.
- **Subquery** retrieves all RollNo from ENROLLMENT.

2. Retrieve all society names

```
SELECT SocName FROM SOCIETY;
```

Explanation:

- Directly fetches society names from SOCIETY table.

3. Student names starting with 'A'

```
SELECT StudentName
FROM STUDENT
WHERE StudentName LIKE 'A%';
```

Explanation:

- **LIKE 'A%'** means names beginning with A.

4. Students studying Computer Science or Chemistry

```
SELECT *
FROM STUDENT
WHERE Course IN ('computerscience','chemistry');
```

Explanation:

- **IN** lets us match multiple course names.

5. Roll numbers starting with X or Z and ending with 1

```
SELECT StudentName
FROM STUDENT
WHERE (RollNo LIKE 'X%' OR RollNo LIKE 'Z%')
AND RollNo LIKE '%1';
```

Explanation:

- First condition matches prefix (X or Z).
- Second condition ensures it ends with 1.

6. Societies with more than N seats

```
SELECT *
FROM SOCIETY
WHERE TotalSeats > N;
```

Explanation:

- Filters based on seat capacity greater than user input.

7. Update mentor name

```
UPDATE SOCIETY
SET MentorName='NewMentor'
WHERE SocID='S01';
```

Explanation:

- **SET** updates a column.
- **WHERE** ensures only one society is modified.

8. Societies with more than 5 enrolled students

```
SELECT SID, COUNT(*) AS Total
FROM ENROLLMENT
GROUP BY SID
HAVING COUNT(*) > 5;
```

Explanation:

- **GROUP BY SID** groups all enrollments per society.
- **HAVING** is used because COUNT() is an aggregate function.

9. Youngest student in society 'NSS'

```
SELECT StudentName
FROM STUDENT
WHERE RollNo IN (
    SELECT RollNo FROM ENROLLMENT WHERE SID='NSS'
)
ORDER BY DOB DESC
LIMIT 1;
```

Explanation:

- Latest DOB = youngest student.
- Subquery filters only students in society NSS.

10. Most popular society

```
SELECT SID, COUNT(*) AS Total
FROM ENROLLMENT
GROUP BY SID
ORDER BY Total DESC
LIMIT 1;
```

Explanation:

- Highest count = most enrolled society.

11. Two least popular societies

```
SELECT SID, COUNT(*) AS Total
FROM ENROLLMENT
GROUP BY SID
ORDER BY Total ASC
LIMIT 2;
```

Explanation:

- Sorted ascending to find lowest enrolled.

12. Students NOT enrolled in any society

```
SELECT StudentName
FROM STUDENT
WHERE RollNo NOT IN (SELECT RollNo FROM ENROLLMENT);
```

Explanation:

- NOT IN ensures we filter non-enrolled students.

13. Students enrolled in at least two societies

```
SELECT RollNo, COUNT(*)
FROM ENROLLMENT
GROUP BY RollNo
HAVING COUNT(*) >= 2;
```

Explanation:

- Groups by student and checks count 2.

14. Societies with maximum enrollment

```
SELECT SID
FROM ENROLLMENT
GROUP BY SID
HAVING COUNT(*) = (
    SELECT MAX(cnt)
    FROM (SELECT COUNT(*) AS cnt FROM ENROLLMENT GROUP BY SID) AS x
);
```

Explanation:

- Finds the highest enrollment count.
- Matches societies with that count.

15. Students + societies paired

```
SELECT S.StudentName, Y.SocName
FROM STUDENT S
JOIN ENROLLMENT E ON S.RollNo = E.RollNo
JOIN SOCIETY Y ON E.SID = Y.SocID;
```

Explanation:

- JOIN links student → enrollment → society.

16. Students in Debating, Dancing, or Sashakt

```
SELECT StudentName
FROM STUDENT
WHERE RollNo IN (
    SELECT RollNo FROM ENROLLMENT
    WHERE SID IN (
        SELECT SocID FROM SOCIETY
        WHERE SocName IN ('Debating','Dancing','Sashakt')
    )
);
```

Explanation:

- Nested IN finds societies by name, then students enrolled.

17. Societies whose mentor has 'Gupta'

```
SELECT SocName
FROM SOCIETY
WHERE MentorName LIKE '%Gupta%';
```

Explanation:

- LIKE with % matches any position.

18. Societies where enrolled students = 10% of seats

```
SELECT S.SocName
FROM SOCIETY S
JOIN ENROLLMENT E ON S.SocID = E.SID
GROUP BY S.SocID
HAVING COUNT(*) = 0.10 * S.TotalSeats;
```

Explanation:

- HAVING lets us compare aggregates with TotalSeats.

19. Vacant seats

```
SELECT S.SocName,
S.TotalSeats - COUNT(E.RollNo) AS VacantSeats
FROM SOCIETY S
LEFT JOIN ENROLLMENT E ON S.SocID = E.SID
GROUP BY S.SocID;
```

Explanation:

- LEFT JOIN ensures societies with 0 enrollments still appear.

20. Increase TotalSeats by 10%

```
UPDATE SOCIETY
SET TotalSeats = TotalSeats * 1.10;
```

Explanation:

- Multiplies existing seats by 1.10.

21. Add FeesPaid column

```
ALTER TABLE ENROLLMENT
ADD FeesPaid ENUM('Yes','No') DEFAULT 'No';
```

Explanation:

- Adds a new field to store payment info.

22. Update enrollment dates

```
UPDATE ENROLLMENT SET DateOfEnrollment='2025-01-01' WHERE SID='S01';
UPDATE ENROLLMENT SET DateOfEnrollment=CURDATE() WHERE SID='S02';
UPDATE ENROLLMENT SET DateOfEnrollment='2024-01-01' WHERE SID='S03';
```

Explanation:

- Updates using WHERE conditions for different societies.

23. Create a view

```
CREATE VIEW SocietyEnrollmentView AS
SELECT S.SocName, COUNT(E.RollNo) AS TotalEnrolled
FROM SOCIETY S
LEFT JOIN ENROLLMENT E ON S.SocID = E.SID
GROUP BY S.SocID;
```

Explanation:

- View acts like a stored result of the query.

24. Students enrolled in all societies

```
SELECT RollNo
FROM ENROLLMENT
GROUP BY RollNo
HAVING COUNT(DISTINCT SID) = (SELECT COUNT(*) FROM SOCIETY);
```

Explanation:

- Compares number of societies a student joined with total societies.

25. Count societies with more than 5 students

```
SELECT COUNT(*)
FROM (
    SELECT SID FROM ENROLLMENT
    GROUP BY SID
    HAVING COUNT(*) > 5
) AS t;
```

Explanation:

- Uses a subquery to find societies first, then counts them.

26. Add MobileNumber column

```
ALTER TABLE STUDENT
ADD MobileNumber VARCHAR(10) DEFAULT 'NA';
```

Explanation:

- ALTER TABLE adds new column with default value.

27. Students older than 20 years

```
SELECT COUNT(*)
FROM STUDENT
WHERE TIMESTAMPDIFF(YEAR, DOB, CURDATE()) > 20;
```

Explanation:

- TIMESTAMPDIFF calculates age from DOB.

28. Students born in 2001 and enrolled

```
SELECT StudentName
FROM STUDENT
WHERE YEAR(DOB)=2001
AND RollNo IN (SELECT RollNo FROM ENROLLMENT);
```

Explanation:

- YEAR(DOB) extracts year of birth.

29. Society names starting with S, ending with t, and at least 7 students

```
SELECT SocName
FROM SOCIETY
WHERE SocName LIKE 'S%t'
AND SocID IN (
    SELECT SID FROM ENROLLMENT
    GROUP BY SID
    HAVING COUNT(*) >= 7
);
```

Explanation:

- LIKE 'S%t' checks pattern.
- Subquery filters societies with 7 enrollments.

30. Full capacity report

```
SELECT S.SocName, S.MentorName, S.TotalSeats AS TotalCapacity,
COUNT(E.RollNo) AS TotalEnrolled,
S.TotalSeats - COUNT(E.RollNo) AS UnfilledSeats
```

```
FROM SOCIETY S
LEFT JOIN ENROLLMENT E ON S.SocID = E.SID
GROUP BY S.SocID;
```

Explanation:

- Shows total seats, enrolled students, and vacant seats.

II. Database Administration Commands

Create User

```
CREATE USER 'bob'@'localhost' IDENTIFIED BY 'bob123';
```

Explanation:

- Creates a new MySQL user.

Create Role

```
CREATE ROLE 'developer';
```

Explanation:

- Role groups permissions for easy management.

Grant Privileges

```
GRANT SELECT, INSERT, UPDATE ON CollegeDB.* TO 'developer';
GRANT 'developer' TO 'bob'@'localhost';
```

Explanation:

- Grants permissions to role.
- Assigns role to user.

Revoke Privileges

```
REVOKE INSERT ON CollegeDB.* FROM 'developer';
```

Explanation:

- Removes only the INSERT permission.

Create Index

```
CREATE INDEX idx_student_name ON STUDENT(StudentName);
```

Explanation:

- Index improves search speed on StudentName.