1. **Creation of tables**
2. **Create Students Table**

CREATE DATABASE StudentManagement; USE StudentManagement;

-- Creating Students Table CREATE TABLE Students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_nameVARCHAR(50) NOT NULL, last\_nameVARCHAR(50) NOT NULL,

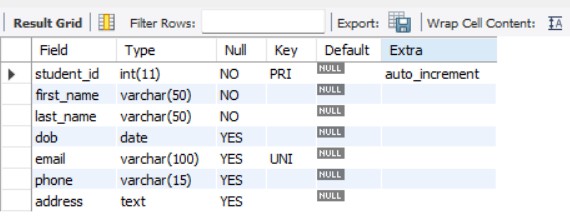
dob DATE,

email VARCHAR(100) UNIQUE, phone VARCHAR(15),

address TEXT

);

select \* from students; desc students; **Output:**



1. **Create Course Table**

-- Creating Courses Table CREATE TABLE Courses (

course\_id INT PRIMARY KEY AUTO\_INCREMENT,

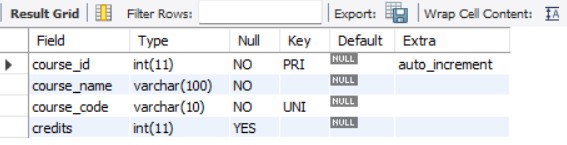
course\_nameVARCHAR(100) NOT NULL, course\_codeVARCHAR(10) UNIQUE NOT NULL,

credits INT CHECK (credits BETWEEN 1 AND 6)

);

select \* from Courses; desc Courses;

**Output:**

****

1. **Create Faculty Table**

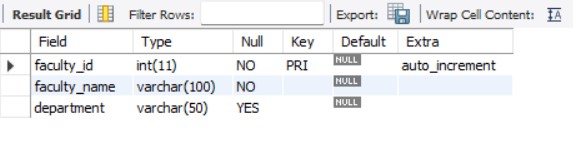
-- Creating Faculty Table CREATE TABLE Faculty (

faculty\_id INT PRIMARY KEY AUTO\_INCREMENT,

faculty\_nameVARCHAR(100) NOT NULL, department VARCHAR(50)

);

select \* from Faculty; desc Faculty; **Output:**



1. **Applying integrity constraints to tables.**

-- Creating Enrollment Table (Many-to-Many Relationship) CREATE TABLE Enrollment (

enrollment\_id INT PRIMARY KEY AUTO\_INCREMENT, student\_id INT,

course\_id INT,

enrollment\_date DATETIME DEFAULT CURRENT\_TIMESTAMP, grade CHAR(2),

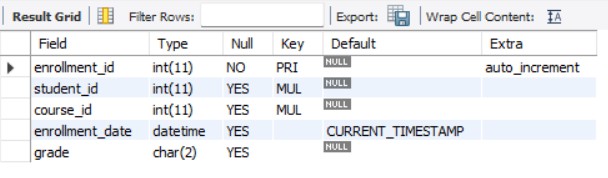
FOREIGN KEY (student\_id) REFERENCES Students(student\_id) ON DELETE CASCADE,

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id) ON DELETE CASCADE

);

select \* from Enrollment; descEnrollment;

**Output:**



1. **Application of INSERT, DELETE & UPDATE commands.**

**INSERT Command**

**Adding Data into the students Table**

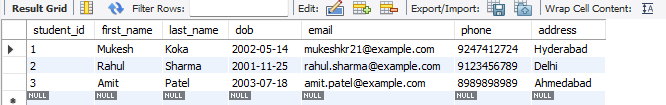
INSERT INTO students (first\_name, last\_name, dob, email, phone, address) VALUES

('Ajay', 'Koka', '2002-05-14', ['ajay,koka@Gmail.com',](mailto:%27ajay%2Ckoka@Gmail.com) '9247412724', 'Hyderabad'),

('Rahul', 'Sharma', '2001-11-25', ['Rahul.sharma@example.com',](mailto:%27Rahul.sharma@example.com) '9123456789', 'Delhi'),

('Amit', 'Patel', '2003-07-18', ['amit.patel@example.com',](mailto:%27amit.patel@example.com) '8989898989', 'Ahmedabad');

**Output:**

****

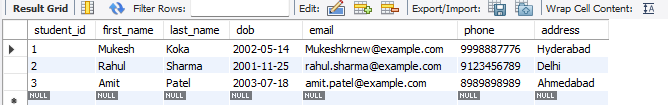
**UPDATE Command**

**Modifying a Student’s Email and Phone Number**

UPDATE students

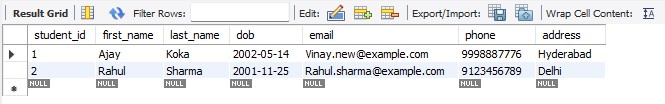
SET email = ['Vinay.new@example.com',](mailto:%27Vinay.new@example.com) phone = '9998887776' WHERE student\_id = 1;

**Output:**

****

**DELETE Command Removing a Student Record** DELETE FROM students WHERE student\_id = 3;

**Output:**

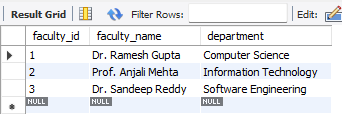


**Inserting into Faculty Table**

INSERT INTO Faculty (faculty\_name, department) VALUES ('Dr. Ramesh Gupta', 'Computer Science'),

('Prof. Anjali Mehta', 'Information Technology'), ('Dr. Sandeep Reddy', 'Software Engineering'); select \* from Faculty;

**Output:**

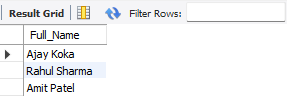
****

1. **Applying built-in functions.**
2. **String Functions**

**Concatenation of First and Last Name**

SELECT CONCAT(first\_name, ' ', last\_name) AS Full\_Name FROM Students;

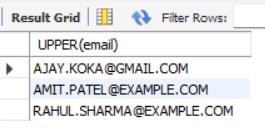
**Output:**

****

**Convert Email to Uppercase**

SELECT UPPER(email) FROM Students;

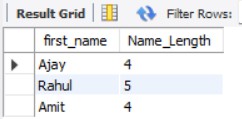
**Output:**

****

**Length of Student's Name**

SELECT first\_name, LENGTH(first\_name) AS Name\_Length FROM Students;

**Output:**

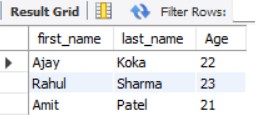
****

1. **Date Functions**

**Finding Age of Students**

SELECT first\_name, last\_name, TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS Age FROM Students;

**Output:**

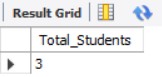
****

1. **Aggregate Functions**

**Count the Total Number of Students**

SELECT COUNT(\*) AS Total\_Students FROM Students;

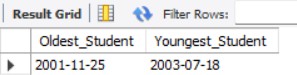
**Output:**

****

**Find the Earliest and Latest Date of Birth**

SELECT MIN(dob) AS Oldest\_Student, MAX(dob) AS Youngest\_Student FROM Students;

**Output:**

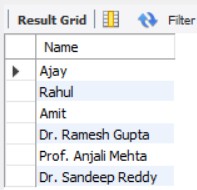
****

1. **Queries Using Set Operators**
2. **UNION: Combining Students and Faculty Names**

SELECT first\_name AS Name FROM Students UNION

SELECT faculty\_name FROM Faculty;

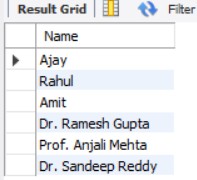
**Output:**

****

1. **UNION ALL: Keeping Duplicate Names** SELECT first\_name AS Name FROM Students UNION ALL

SELECT faculty\_name FROM Faculty;

**Output:**

****

1. **Queries using various types of joins.**

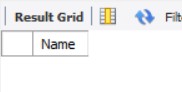
**INNER JOIN to find common names between Students and Faculty.**

SELECT first\_name AS Name FROM Students

INNER JOIN Faculty

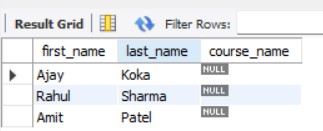
ON Students.first\_name = Faculty.faculty\_name;

**Output:**

****

**LEFT JOIN to get all Students and their Courses, Even If Not Enrolled.** SELECT Students.first\_name, Students.last\_name, Courses.course\_name FROM Students

LEFT JOIN Enrollment ON Students.student\_id = Enrollment.student\_id LEFT JOIN Courses ON Enrollment.course\_id = Courses.course\_id; **Output:**



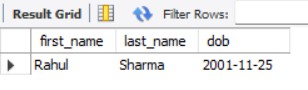
1. **Selecting data using subqueries.**

**Find the Oldest Student (Using MIN in a Subquery)**

SELECT first\_name, last\_name, dob FROM Students

WHERE dob = (SELECT MIN(dob) FROM Students);

**Output:**

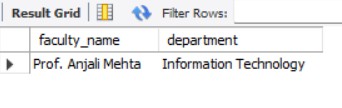
****

**Find Faculty Members Belonging to the Largest Department (Using MAX)**

SELECT faculty\_name, department FROM Faculty

WHERE department = (SELECT department FROM Faculty GROUP BY department ORDER BY COUNT(\*) DESC LIMIT 1);

**Output:**

****

1. **Problems related to database management.**
2. **Handling Orphan Records in the Enrollment Table**

**Problem:**

If a student is deleted from the Students table, their enrollments in the Enrollment table become orphaned (pointing to a non-existent student).

**Solution:**

You already have ON DELETE CASCADE in the Enrollment table, which automatically deletes enrollments when a student is removed.

To verify this behavior:

DELETE FROM Students WHERE student\_id = 2;

SELECT \* FROM Enrollment; -- Check if related records are deleted

1. **Preventing Duplicate Student Entries**

**Problem:**

The **Students** table allows NULL values in the **email** column. However, we want to ensure that every student must have an email address.

How can we modify the table to prevent NULL values in the email column?

**Solution:**

We can use the ALTER TABLE statement to modify the email column and set it as NOT NULL, ensuring that every student has a valid email.

ALTER TABLE Students MODIFY email VARCHAR(100) NOT NULL;

1. **Handling Null Values in Student Contact Information**

**Problem:**

Some students might not provide an email or phone number, leading to incomplete data.

**Solution:**

Use NOT NULL constraints and provide default values if needed.

ALTER TABLE Students MODIFY email VARCHAR(100) NOT NULL;

ALTER TABLE Students MODIFY phone VARCHAR(15) NOT NULL DEFAULT 'Not

Provided';

1. **Finding Students Without Enrollment (Data Consistency)**

**Problem:**

Some students might exist in the Students table but have **never enrolled** in any course.

**Solution:**

Use a LEFT JOIN to find such students.

SELECT s.student\_id, s.first\_name, s.last\_name FROM Students s

LEFT JOIN Enrollment e ON s.student\_id = e.student\_id

WHERE e.student\_id IS NULL;

1. **Identifying Courses Without Enrollments**

**Problem:**

Some courses might exist in the Courses table but have **no students enrolled**.

**Solution:**

Use NOT EXISTS to find such courses.

SELECT c.course\_id, c.course\_name FROM Courses c

WHERE NOT EXISTS (SELECT 1 FROM Enrollment e WHERE e.course\_id = c.course\_id);

1. **Normalization Issue – Storing Repetitive Faculty Data rses Without Enrollments**

**Problem:**

If the same faculty name appears multiple times in different departments, it leads to **data redundancy**.

**Solution:**

Create a Departments table and use department\_id as a foreign key in the Faculty table.

CREATE TABLE Departments (

department\_id INT PRIMARY KEY AUTO\_INCREMENT, department\_name VARCHAR(100) UNIQUE NOT NULL);

ALTER TABLE Faculty ADD COLUMN department\_id INT;

ALTER TABLE Faculty ADD CONSTRAINT fk\_department FOREIGN KEY (department\_id) REFERENCES Departments(department\_id);

1. **Handling Case Sensitivity in Searches**

**Problem:**

If users search for a student with SELECT \* FROM Students WHERE first\_name = 'ajay';, it may not return results if names are stored with capitalized letters.

**Solution:**

Use LOWER() or COLLATE for case-insensitive searches.

SELECT \* FROM Students WHERE LOWER(first\_name) = LOWER('ajay');

1. **Checking for Duplicate Faculty Names**

**Problem:**

Two faculty members with the **same name but different departments** may cause confusion.

**Solution:**

Find duplicate names using GROUP BY.

SELECT faculty\_name, COUNT(\*) AS count FROM Faculty

GROUP BY faculty\_name

HAVING COUNT(\*) > 1;

1. **Improving Query Performance Using Indexes**

**Problem:**

If the database grows, queries may slow down due to full table scans.

**Solution:**

Add indexes on frequently searched columns.

CREATE INDEX idx\_student\_name ON Students(first\_name, last\_name);

CREATE INDEX idx\_course\_code ON Courses(course\_code);

1. **Backing Up the Database Regularly**

**Problem:**

Data loss due to accidental deletions or system failures.

**Solution:**

Use MySQL’s backup feature (mysqldump) to export data.

mysqldump -u root -p StudentManagement > backup.sql