**Lab 1: Creating a Simple Database**

**Exercise**

Create a database named SchoolDB.

**Solution**

CREATE DATABASE SchoolDB;

**Lab 2: Creating a Table**

**Exercise**

Create a table Students with columns: StudentID (Primary Key), Name, Age, Class, Email.

**Solution**

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

Class VARCHAR(50),

Email VARCHAR(100)

);

**Lab 3: Adding Constraints**

**Exercise**

Modify the Students table to ensure:

* Email is unique.
* Age cannot be NULL.

**Solution**

ALTER TABLE Students

ADD CONSTRAINT unique\_email UNIQUE (Email),

MODIFY Age INT NOT NULL;

**Lab 4: Creating Relationships**

**Exercise**

Create a Courses table and link it to Students using a foreign key StudentID.

**Solution**

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(100),

StudentID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID)

);

**Lab 5: Normalization (1NF, 2NF, 3NF)**

**Exercise**

Given the unnormalized table:

| OrderID | Product | Quantity | Price | CustomerName | Address |

Normalize it to 3NF.

**Solution**

* **1NF:** Split repeating groups.
* **2NF:** Create separate Orders and Customers tables.
* **3NF:** Remove transitive dependency by creating a Products table.

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Address VARCHAR(255)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Price DECIMAL(10,2)

);

CREATE TABLE OrderDetails (

OrderID INT,

ProductID INT,

Quantity INT,

PRIMARY KEY (OrderID, ProductID),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

**Lab 6: One-to-Many Relationship**

**Exercise**

Create a Teachers table and establish a one-to-many relationship with Courses.

**Solution**

CREATE TABLE Teachers (

TeacherID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

ALTER TABLE Courses ADD COLUMN TeacherID INT;

ALTER TABLE Courses ADD FOREIGN KEY (TeacherID) REFERENCES Teachers(TeacherID);

**Lab 7: Many-to-Many Relationship**

**Exercise**

Implement a many-to-many relationship between Students and Courses.

**Solution**

CREATE TABLE StudentCourses (

StudentID INT,

CourseID INT,

PRIMARY KEY (StudentID, CourseID),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

**Lab 8: Inserting Data**

**Exercise**

Insert sample data into the Students and Courses tables.

**Solution**

INSERT INTO Students (StudentID, Name, Age, Class, Email)

VALUES (1, 'Alice', 20, 'CS101', 'alice@example.com');

INSERT INTO Courses (CourseID, CourseName, StudentID)

VALUES (101, 'Database Management', 1);

**Lab 9: Updating Data**

**Exercise**

Update Alice's age to 21.

**Solution**

UPDATE Students SET Age = 21 WHERE Name = 'Alice';

**Lab 10: Deleting Data**

**Exercise**

Delete the course Database Management from the Courses table.

**Solution**

DELETE FROM Courses WHERE CourseName = 'Database Management';

**Lab 11: Indexing**

**Exercise**

Create an index on the Email column in the Students table.

**Solution**

CREATE INDEX idx\_email ON Students(Email);

**Lab 12: Using Views**

**Exercise**

Create a view that shows student names and their enrolled courses.

**Solution**

CREATE VIEW StudentCoursesView AS

SELECT s.Name, c.CourseName

FROM Students s

JOIN StudentCourses sc ON s.StudentID = sc.StudentID

JOIN Courses c ON sc.CourseID = c.CourseID;

**Lab 13: Using Stored Procedures**

**Exercise**

Write a stored procedure to get student details by ID.

**Solution**

DELIMITER //

CREATE PROCEDURE GetStudentDetails(IN student\_id INT)

BEGIN

SELECT \* FROM Students WHERE StudentID = student\_id;

END //

DELIMITER ;

**Lab 14: Using Triggers**

**Exercise**

Create a trigger that prevents deletion of students with registered courses.

**Solution**

DELIMITER //

CREATE TRIGGER prevent\_student\_deletion

BEFORE DELETE ON Students

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM StudentCourses WHERE StudentID = OLD.StudentID) > 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Cannot delete student with registered courses';

END IF;

END //

DELIMITER ;

**Lab 15: Using Transactions**

**Exercise**

Write a transaction to enroll a student in a course.

**Solution**

START TRANSACTION;

INSERT INTO StudentCourses (StudentID, CourseID) VALUES (1, 101);

COMMIT;

**Lab 16: Using Joins**

**Exercise**

Fetch all students and their enrolled courses.

**Solution**

SELECT s.Name, c.CourseName

FROM Students s

JOIN StudentCourses sc ON s.StudentID = sc.StudentID

JOIN Courses c ON sc.CourseID = c.CourseID;

**Lab 17: Using Subqueries**

**Exercise**

Find students who are not enrolled in any course.

**Solution**

SELECT Name FROM Students

WHERE StudentID NOT IN (SELECT StudentID FROM StudentCourses);

**Lab 18: Using Group By**

**Exercise**

Find the count of students in each class.

**Solution**

SELECT Class, COUNT(\*) AS StudentCount

FROM Students

GROUP BY Class;

**Lab 19: Using Having**

**Exercise**

Find classes with more than 2 students.

**Solution**

SELECT Class, COUNT(\*) AS StudentCount

FROM Students

GROUP BY Class

HAVING COUNT(\*) > 2;

**Lab 20: Using Window Functions**

**Exercise**

Rank students by age in descending order.

**Solution**

SELECT Name, Age, RANK() OVER (ORDER BY Age DESC) AS Rank

FROM Students;

**Case Study 1: E-Commerce Database Design**

**Scenario**

A company wants to track products, customers, and orders. Each order contains multiple products, and each customer can place multiple orders.

**Solution**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100) UNIQUE

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Name VARCHAR(100),

Price DECIMAL(10,2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

OrderID INT,

ProductID INT,

Quantity INT,

PRIMARY KEY (OrderID, ProductID),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

**Case Study 2: Library Management System**

**Scenario**

A library needs to track books, authors, and borrowers. A book can have multiple authors, and borrowers can borrow multiple books.

**Solution**

CREATE TABLE Authors (

AuthorID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Books (

BookID INT PRIMARY KEY,

Title VARCHAR(100)

);

CREATE TABLE BookAuthors (

BookID INT,

AuthorID INT,

PRIMARY KEY (BookID, AuthorID),

FOREIGN KEY (BookID) REFERENCES Books(BookID),

FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)

);

CREATE TABLE Borrowers (

BorrowerID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Borrowings (

BorrowID INT PRIMARY KEY,

BorrowerID INT,

BookID INT,

BorrowDate DATE,

ReturnDate DATE,

FOREIGN KEY (BorrowerID) REFERENCES Borrowers(BorrowerID),

FOREIGN KEY (BookID) REFERENCES Books(BookID)

);

**Case Study 3: Hospital Management System**

**Scenario**

A hospital manages patients, doctors, and appointments.

**Solution**

CREATE TABLE Patients (

PatientID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

Contact VARCHAR(50)

);

CREATE TABLE Doctors (

DoctorID INT PRIMARY KEY,

Name VARCHAR(100),

Specialty VARCHAR(100)

);

CREATE TABLE Appointments (

AppointmentID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

AppointmentDate DATETIME,

FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),

FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)

);

**Case Study 4: Employee Management System**

**Scenario**

A company wants to manage employees, departments, and salaries.

**Solution**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

DepartmentID INT,

Salary DECIMAL(10,2),

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

**Case Study 5: Online Learning Platform**

**Scenario**

An education platform tracks courses, instructors, and students.

**Solution**

CREATE TABLE Instructors (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID)

);

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Enrollments (

StudentID INT,

CourseID INT,

PRIMARY KEY (StudentID, CourseID),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);