1. **Design any database with at least 3 entities and relationships between them. Draw suitable ER/EER diagram for the system.**

**Answer :**

**📘 Entity Attributes:**

**Book**

* BookID (PK)
* Title
* Author
* ISBN

**Member**

* MemberID (PK)
* Name
* Email

**Loan**

* LoanID (PK)
* MemberID (FK)
* BookID (FK)
* LoanDate
* ReturnDate

🛠 SQL Commands to Create Tables:

-- Step 1: Create the database

CREATE DATABASE LibraryDB;

-- Step 2: Use the database

USE LibraryDB;

-- Step 3: Create the Book table

CREATE TABLE Book (

BookID INT PRIMARY KEY,

Title VARCHAR(100),

Author VARCHAR(100),

ISBN VARCHAR(20)

);

-- Step 4: Create the Member table

CREATE TABLE Member (

MemberID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100)

);

-- Step 5: Create the Loan table with foreign keys

CREATE TABLE Loan (

LoanID INT PRIMARY KEY,

MemberID INT,

BookID INT,

LoanDate DATE,

ReturnDate DATE,

FOREIGN KEY (MemberID) REFERENCES Member(MemberID),

FOREIGN KEY (BookID) REFERENCES Book(BookID)

);

-- Step 6: Show all tables in the database

SHOW TABLES;

-- Step 7: Show table structures

DESCRIBE Book;

DESCRIBE Member;

DESCRIBE Loan;

1. **Design and implement a database (for assignment no 1) using DDL statements and apply normalization on them.**

**Ans :**

## ✅ Step 1: ****Database Design**** (with Normalization)

### Entities:

1. **Student**
2. **Course**
3. **Instructor**
4. **Enrollment** (relationship between Student and Course)

### 🔍 Normalization Steps:

We will assume this is in **3rd Normal Form (3NF)**:

* No repeating groups (1NF ✅)
* All non-key attributes are fully functionally dependent on the primary key (2NF ✅)
* No transitive dependencies (3NF ✅)

### 📦 Tables & Attributes:

#### Student

* StudentID (PK)
* Name
* Email

#### Instructor

* InstructorID (PK)
* Name
* Department

#### Course

* CourseID (PK)
* Title
* InstructorID (FK → Instructor)

#### Enrollment

* EnrollmentID (PK)
* StudentID (FK → Student)
* CourseID (FK → Course)
* EnrollmentDate

✅ Step 2: **SQL DDL Statements (Normalized Design)**

-- Step 1: Create the database

CREATE DATABASE StudentCourseDB;

-- Step 2: Use the database

USE StudentCourseDB;

-- Step 3: Create the Student table

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100)

);

-- Step 4: Create the Instructor table

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

-- Step 5: Create the Course table

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

-- Step 6: Create the Enrollment table

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

-- Step 7: Insert sample data into Student table

INSERT INTO Student (StudentID, Name, Email) VALUES

(1, 'Alice Smith', 'alice.smith@example.com'),

(2, 'Bob Johnson', 'bob.johnson@example.com'),

(3, 'Charlie Brown', 'charlie.brown@example.com'),

(4, 'David Williams', 'david.williams@example.com'),

(5, 'Eve Davis', 'eve.davis@example.com');

-- Step 8: Insert sample data into Instructor table

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. John Doe', 'Computer Science'),

(2, 'Prof. Jane Green', 'Mathematics'),

(3, 'Dr. Michael Black', 'Physics');

-- Step 9: Insert sample data into Course table

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

-- Step 10: Insert sample data into Enrollment table

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, 1, '2025-04-01'),

(2, 1, 2, '2025-04-02'),

(3, 2, 3, '2025-04-03'),

(4, 3, 4, '2025-04-04'),

(5, 4, 5, '2025-04-05');

-- Step 11: Show all tables in the database

SHOW TABLES;

-- Step 12: Describe table structure for each table

DESCRIBE Student;

DESCRIBE Instructor;

DESCRIBE Course;

DESCRIBE Enrollment;

-- Step 13: Verify data by selecting all records from each table

SELECT \* FROM Student;

SELECT \* FROM Instructor;

SELECT \* FROM Course;

SELECT \* FROM Enrollment;

1. **. Create Table with primary key and foreign key constraints. a. Alter table with add n modify b. Drop table**

**Ans :**

**1. Create Tables with Primary and Foreign Key Constraints**

In this example, we will create a **Student** table, an **Instructor** table, and a **Course** table, where:

* Student will have a **Primary Key** on StudentID.
* Course will have a **Foreign Key** on InstructorID (referencing the Instructor table).
* Enrollment will be a **junction table** with **Foreign Keys** to both Student and Course.

**-- Step 1: Create the database**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table (using Indian names)**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table (using Indian names)**

INSERT INTO Student (StudentID, Name, Email) VALUES

(1, 'Amit Patel', 'amit.patel@example.com'),

(2, 'Priya Sharma', 'priya.sharma@example.com'),

(3, 'Ravi Kumar', 'ravi.kumar@example.com'),

(4, 'Neha Singh', 'neha.singh@example.com'),

(5, 'Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, 1, '2025-04-01'),

(2, 2, 2, '2025-04-02'),

(3, 3, 3, '2025-04-03'),

(4, 4, 4, '2025-04-04'),

(5, 5, 5, '2025-04-05');

**-- Step 11: Add a PhoneNumber column to Student table**

ALTER TABLE Student

ADD COLUMN PhoneNumber VARCHAR(15);

**-- Step 12: Modify the Email column in Student table to allow longer email addresses**

ALTER TABLE Student

MODIFY COLUMN Email VARCHAR(150);

**-- Step 13: Drop the Enrollment table (removes the table and its data)**

DROP TABLE Enrollment;

**-- Step 14: View all tables in the database**

SHOW TABLES;

**-- Step 15: Describe the structure of each table to verify their structure**

DESCRIBE Instructor;

DESCRIBE Student;

DESCRIBE Course;

DESCRIBE Enrollment;

**-- Step 16: View all data from each table to verify the inserted data**

SELECT \* FROM Instructor;

SELECT \* FROM Student;

SELECT \* FROM Course;

SELECT \* FROM Enrollment;

1. **Perform following SQL queries on the database created in assignment 1.**

**• Implementation of relational operators in SQL**

**• Boolean operators and pattern matching**

**• Arithmetic operations and built in functions**

**• Group functions**

**• Processing Date and Time functions**

**• Complex queries and set operators**

**Ans :**

**-- Step 1: Create the database**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table (using Indian names)**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table (using Indian names)**

INSERT INTO Student (StudentID, Name, Email) VALUES

(1, 'Amit Patel', 'amit.patel@example.com'),

(2, 'Priya Sharma', 'priya.sharma@example.com'),

(3, 'Ravi Kumar', 'ravi.kumar@example.com'),

(4, 'Neha Singh', 'neha.singh@example.com'),

(5, 'Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, 1, '2025-04-01'),

(2, 2, 2, '2025-04-02'),

(3, 3, 3, '2025-04-03'),

(4, 4, 4, '2025-04-04'),

(5, 5, 5, '2025-04-05');

**-- -- Step 11: Relational Operators**

-- Find all students whose names start with 'A' and whose StudentID is greater than 2

SELECT \* FROM Student

WHERE Name LIKE 'A%' AND StudentID > 2;

**-- Step 12: Boolean Operators and Pattern Matching**

-- Find students who either belong to the "Computer Science" department or have an email address containing "example.com"

SELECT \* FROM Student

WHERE Email LIKE '%example.com%' OR StudentID IN (SELECT StudentID FROM Enrollment WHERE CourseID = 1);

**-- Step 13: Arithmetic Operations and Built-in Functions**

-- Calculate the total number of students enrolled in each course and display their names in uppercase.

SELECT C.Title AS CourseTitle, COUNT(E.StudentID) AS TotalStudents,

UPPER(S.Name) AS StudentName

FROM Course C

JOIN Enrollment E ON C.CourseID = E.CourseID

JOIN Student S ON E.StudentID = S.StudentID

GROUP BY C.CourseID, C.Title, S.Name;

**-- Step 14: Group Functions**

-- Find the average enrollment count per course.

SELECT C.Title AS CourseTitle, AVG(TotalStudents) AS AvgEnrollment

FROM Course C

JOIN (SELECT CourseID, COUNT(StudentID) AS TotalStudents FROM Enrollment GROUP BY CourseID) AS E

ON C.CourseID = E.CourseID

GROUP BY C.Title;

**-- Step 15: Processing Date and Time Functions**

-- Find all students who enrolled after '2025-04-03'.

SELECT S.Name, E.EnrollmentDate

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

WHERE E.EnrollmentDate > '2025-04-03';

-- Calculate the number of days between the enrollment date and today's date for each student.

SELECT S.Name, DATEDIFF(CURDATE(), E.EnrollmentDate) AS DaysSinceEnrollment

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID;

**-- Step 16: Complex Queries and Set Operators**

-- Find the names of students who are enrolled in both 'Introduction to Programming' and 'Data Structures' (using INTERSECT).

SELECT S.Name

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID

WHERE C.Title = 'Introduction to Programming'

INTERSECT

SELECT S.Name

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID

WHERE C.Title = 'Data Structures';

-- Find all students who are enrolled in either 'Introduction to Programming' or 'Data Structures' (using UNION).

SELECT S.Name

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID

WHERE C.Title = 'Introduction to Programming'

UNION

SELECT S.Name

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID

WHERE C.Title = 'Data Structures';

**Explanation of the Code**

1. **Database Creation & Table Definitions**:
   * We create a new database IndianCollegeDB.
   * We define the tables Instructor, Student, Course, and Enrollment, with their respective primary keys and foreign key relationships.
2. **Data Insertion**:
   * Sample data is inserted into the tables. The Instructor table contains information about instructors, the Student table contains student details, the Course table contains course details, and the Enrollment table tracks which students are enrolled in which courses.
3. **SQL Operations**:
   * **Relational Operators**: We use relational operators (LIKE, >, etc.) to filter students whose names start with 'A' and whose StudentID is greater than 2.
   * **Boolean Operators and Pattern Matching**: We combine conditions with OR and use LIKE to find students whose email contains "example.com".
   * **Arithmetic Operations and Built-in Functions**: We calculate the total number of students per course and display their names in uppercase.
   * **Group Functions**: We calculate the average number of students enrolled in each course.
   * **Date and Time Functions**: We find students who enrolled after a specific date and calculate the days since enrollment using DATEDIFF().
   * **Complex Queries and Set Operators**: We use INTERSECT to find students enrolled in both 'Introduction to Programming' and 'Data Structures', and UNION to find students enrolled in either course.

**5.Execute DDL/DML statements which demonstrate the use of views. Update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.**

**Ans :**

**-- Step 1: Create the database**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table**

INSERT INTO Student (StudentID, Name, Email) VALUES

(1, 'Amit Patel', 'amit.patel@example.com'),

(2, 'Priya Sharma', 'priya.sharma@example.com'),

(3, 'Ravi Kumar', 'ravi.kumar@example.com'),

(4, 'Neha Singh', 'neha.singh@example.com'),

(5, 'Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, 1, '2025-04-01'),

(2, 2, 2, '2025-04-02'),

(3, 3, 3, '2025-04-03'),

(4, 4, 4, '2025-04-04'),

(5, 5, 5, '2025-04-05');

**-- Step 11: Create a simple view that selects student names and emails**

CREATE VIEW StudentView AS

SELECT Name, Email

FROM Student;

**-- Query the StudentView to show all students and their emails**

SELECT \* FROM StudentView;

**-- Step 12: Create a view that joins Student, Enrollment, and Course tables**

CREATE VIEW StudentCourseView AS

SELECT S.Name AS StudentName, C.Title AS CourseTitle, E.EnrollmentDate

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID;

**-- Query the StudentCourseView to show student names, course titles, and enrollment** dates

SELECT \* FROM StudentCourseView;

**-- Step 13: Update through the view on the Student table (StudentView)**

UPDATE StudentView

SET Email = 'amit.newemail@example.com'

WHERE Name = 'Amit Patel';

**-- Verify the update in the base Student table**

SELECT \* FROM Student WHERE Name = 'Amit Patel';

**-- Step 14: Create a view with aggregate function (non-updatable view)**

CREATE VIEW CourseEnrollmentCount AS

SELECT C.Title AS CourseTitle, COUNT(E.StudentID) AS StudentCount

FROM Course C

LEFT JOIN Enrollment E ON C.CourseID = E.CourseID

GROUP BY C.Title;

**-- Query the CourseEnrollmentCount view (non-updatable view)**

SELECT \* FROM CourseEnrollmentCount;

**-- Try updating through a non-updatable view (this will fail)**

**-- Uncomment the following line to see the error**

UPDATE CourseEnrollmentCount

SET StudentCount = 10

WHERE CourseTitle = 'Introduction to Programming';

**-- Step 15: Drop the StudentCourseView if no longer needed**

DROP VIEW StudentCourseView;

**-- Step 16: Drop the database if no longer needed (optional)**

DROP DATABASE IndianCollegeDB;

**-- ------------- Commands to Show Desired Output ------------**

**-- Show the data in the StudentView (Names and Emails)**

SELECT \* FROM StudentView;

**-- Show the data in the StudentCourseView (Student Name, Course Title, Enrollment Date)**

SELECT \* FROM StudentCourseView;

**-- Show the data in the CourseEnrollmentCount (Course Title, Student Count)**

SELECT \* FROM CourseEnrollmentCount;

**-- After updating through the view, show the updated email for Amit Patel**

SELECT \* FROM Student WHERE Name = 'Amit Patel';

**-- Show the error when trying to update a non-updatable view**

**-- Uncomment the following lines to see the error**

UPDATE CourseEnrollmentCount

SET StudentCount = 10

WHERE CourseTitle = 'Introduction to Programming';

**6. Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use.**

**Ans :**

### 1. ****Stored Procedure****: A stored procedure will be created to insert a new student into the Student table. It will take the student's name and email as parameters.

### 2. ****Function****: A function will be created to calculate and return the total number of students enrolled in a specific course. It will take a CourseID as input and return the count of students enrolled in that course.

**Complete PL/SQL Code:**

**-- Step 1: Create the database**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the created database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY AUTO\_INCREMENT,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table**

INSERT INTO Student (Name, Email) VALUES

('Amit Patel', 'amit.patel@example.com'),

('Priya Sharma', 'priya.sharma@example.com'),

('Ravi Kumar', 'ravi.kumar@example.com'),

('Neha Singh', 'neha.singh@example.com'),

('Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, '2025-04-01'),

(2, 2, '2025-04-02'),

(3, 3, '2025-04-03'),

(4, 4, '2025-04-04'),

(5, 5, '2025-04-05');

**-- Step 11: Create a Stored Procedure to Add a New Student**

CREATE OR REPLACE PROCEDURE AddNewStudent(

p\_StudentName IN VARCHAR(100),

p\_Email IN VARCHAR(100)

)

IS

BEGIN

-- Insert a new student into the Student table

INSERT INTO Student (Name, Email)

VALUES (p\_StudentName, p\_Email);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Student ' || p\_StudentName || ' has been added successfully.');

END;

/

**-- Step 12: Create a Function to Get Total Enrolled Students for a Given Course**

CREATE OR REPLACE FUNCTION GetTotalEnrolledStudents(

p\_CourseID IN INT

)

RETURN INT

IS

total\_enrollment INT;

BEGIN

-- Count the number of students enrolled in the given course

SELECT COUNT(\*)

INTO total\_enrollment

FROM Enrollment

WHERE CourseID = p\_CourseID;

RETURN total\_enrollment;

END;

/

**-- Step 13: Execute the Stored Procedure to Add a New Student**

EXEC AddNewStudent('Suresh Verma', 'suresh.verma@example.com');

**-- Step 14: Call the Function to Get the Total Number of Students Enrolled in Course 1**

DECLARE

v\_TotalStudents INT;

BEGIN

**-- Get the total number of students enrolled in Course 1**

v\_TotalStudents := GetTotalEnrolledStudents(1);

DBMS\_OUTPUT.PUT\_LINE('Total students enrolled in Course 1: ' || v\_TotalStudents);

END;

/

**-- Step 15: Show the List of All Students**

SELECT \* FROM Student;

**-- Step 16: Show the List of All Enrollments (Student-Course Pairings)**

SELECT S.Name AS StudentName, C.Title AS CourseTitle, E.EnrollmentDate

FROM Student S

JOIN Enrollment E ON S.StudentID = E.StudentID

JOIN Course C ON E.CourseID = C.CourseID;

**-- Step 17: Show the Total Number of Students Enrolled in Each Course**

SELECT C.Title AS CourseTitle, COUNT(E.StudentID) AS TotalEnrolled

FROM Course C

LEFT JOIN Enrollment E ON C.CourseID = E.CourseID

GROUP BY C.Title;

**7.** **Write and execute suitable database triggers .Consider row level and statement level triggers.**

**Ans :**

.  **Row-Level Trigger**:

* We will create a row-level trigger that automatically updates the EnrollmentDate in the Enrollment table whenever a new enrollment is added or an existing enrollment is updated. For simplicity, the trigger will set the EnrollmentDate to the current date (SYSDATE) when a row is inserted or updated.

 **Statement-Level Trigger**:

* We will create a statement-level trigger that prevents any deletion from the Student table if a student is enrolled in any course. The trigger will check if the student is enrolled in any course by looking at the Enrollment table. If the student is enrolled, the deletion will be canceled, and an error message will be displayed.

**Complete SQL Code with Triggers**:

**-- Step 1: Create the database (if not already created)**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the created database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY AUTO\_INCREMENT,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table**

INSERT INTO Student (Name, Email) VALUES

('Amit Patel', 'amit.patel@example.com'),

('Priya Sharma', 'priya.sharma@example.com'),

('Ravi Kumar', 'ravi.kumar@example.com'),

('Neha Singh', 'neha.singh@example.com'),

('Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, '2025-04-01'),

(2, 2, '2025-04-02'),

(3, 3, '2025-04-03'),

(4, 4, '2025-04-04'),

(5, 5, '2025-04-05');

**-- Step 11: Create a Row-Level Trigger to automatically update EnrollmentDate**

CREATE OR REPLACE TRIGGER trg\_update\_enrollment\_date

AFTER INSERT OR UPDATE ON Enrollment

FOR EACH ROW

BEGIN

**-- Set the EnrollmentDate to the current date whenever a new enrollment is inserted or updated**

:NEW.EnrollmentDate := SYSDATE;

DBMS\_OUTPUT.PUT\_LINE('Enrollment Date set to ' || TO\_CHAR(SYSDATE, 'YYYY-MM-DD') || ' for EnrollmentID ' || :NEW.EnrollmentID);

END;

/

**-- Step 12: Create a Statement-Level Trigger to prevent deletion of students enrolled in any course**

CREATE OR REPLACE TRIGGER trg\_prevent\_delete\_student

BEFORE DELETE ON Student

FOR EACH ROW

DECLARE

v\_enrollment\_count INT;

BEGIN

-- Check if the student is enrolled in any course by looking in the Enrollment table

SELECT COUNT(\*) INTO v\_enrollment\_count

FROM Enrollment

WHERE StudentID = :OLD.StudentID;

IF v\_enrollment\_count > 0 THEN

**-- If the student is enrolled in any course, prevent the deletion**

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete student. This student is enrolled in one or more courses.');

END IF;

END;

/

**-- Step 13: Execute the Row-Level Trigger (Example: Insert a new enrollment)**

INSERT INTO Enrollment (StudentID, CourseID) VALUES (2, 3);

**-- This will trigger the row-level trigger and automatically set the EnrollmentDate to the current date**

**-- Step 14: Execute the Statement-Level Trigger (Example: Try to delete a student enrolled in a course)**

DELETE FROM Student WHERE StudentID = 2;

**-- This will trigger the statement-level trigger, and the deletion will be prevented if the student is enrolled in any course**

**-- Step 15: Show the data from the Enrollment table to check EnrollmentDate after the trigger**

SELECT \* FROM Enrollment;

**-- Step 16: Show the data from the Student table (no deletion should happen for enrolled students)**

SELECT \* FROM Student;

**8. Write a PL/SQL block to implement all types of cursor.**

**Ans :**

**-- Step 1: Create the database (if not already created)**

CREATE DATABASE IndianCollegeDB;

**-- Step 2: Use the created database**

USE IndianCollegeDB;

**-- Step 3: Create the Instructor table with Primary Key**

CREATE TABLE Instructor (

InstructorID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100)

);

**-- Step 4: Create the Student table with Primary Key**

CREATE TABLE Student (

StudentID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Email VARCHAR(100)

);

**-- Step 5: Create the Course table with Foreign Key reference to Instructor table**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

Title VARCHAR(100),

InstructorID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

);

**-- Step 6: Create the Enrollment table (junction table) with Foreign Keys**

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY AUTO\_INCREMENT,

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

);

**-- Step 7: Insert sample data into Instructor table**

INSERT INTO Instructor (InstructorID, Name, Department) VALUES

(1, 'Dr. Rajesh Kumar', 'Computer Science'),

(2, 'Prof. Meena Agarwal', 'Mathematics'),

(3, 'Dr. Arvind Sharma', 'Physics');

**-- Step 8: Insert sample data into Student table**

INSERT INTO Student (Name, Email) VALUES

('Amit Patel', 'amit.patel@example.com'),

('Priya Sharma', 'priya.sharma@example.com'),

('Ravi Kumar', 'ravi.kumar@example.com'),

('Neha Singh', 'neha.singh@example.com'),

('Vikram Joshi', 'vikram.joshi@example.com');

**-- Step 9: Insert sample data into Course table**

INSERT INTO Course (CourseID, Title, InstructorID) VALUES

(1, 'Introduction to Programming', 1),

(2, 'Data Structures', 1),

(3, 'Calculus I', 2),

(4, 'Quantum Mechanics', 3),

(5, 'Linear Algebra', 2);

**-- Step 10: Insert sample data into Enrollment table**

INSERT INTO Enrollment (StudentID, CourseID, EnrollmentDate) VALUES

(1, 1, '2025-04-01'),

(2, 2, '2025-04-02'),

(3, 3, '2025-04-03'),

(4, 4, '2025-04-04'),

(5, 5, '2025-04-05');

**-- Step 11: PL/SQL block demonstrating different types of cursors**

DECLARE

**-- 1. Implicit Cursor example: Using SELECT INTO**

v\_StudentName VARCHAR2(100);

v\_StudentEmail VARCHAR2(100);

**-- 2. Explicit Cursor example: Declaring an explicit cursor**

CURSOR c\_enrollments IS

SELECT e.EnrollmentID, s.Name AS StudentName, c.Title AS CourseTitle

FROM Enrollment e

JOIN Student s ON e.StudentID = s.StudentID

JOIN Course c ON e.CourseID = c.CourseID;

**-- 3. Cursor FOR Loop example: Using a FOR loop for an implicit cursor**

v\_EnrollmentCount INT := 0;

**-- 4. Parameterized Cursor example: A cursor that accepts a parameter**

CURSOR c\_student\_enrollment(p\_StudentID INT) IS

SELECT c.Title

FROM Enrollment e

JOIN Course c ON e.CourseID = c.CourseID

WHERE e.StudentID = p\_StudentID;

BEGIN

**-- 1. Implicit Cursor: Using SELECT INTO to fetch student info**

SELECT Name, Email

INTO v\_StudentName, v\_StudentEmail

FROM Student

WHERE StudentID = 1;

**-- Display the student information**

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_StudentName);

DBMS\_OUTPUT.PUT\_LINE('Student Email: ' || v\_StudentEmail);

**-- 2. Explicit Cursor: Opening and fetching data from explicit cursor**

OPEN c\_enrollments;

LOOP

FETCH c\_enrollments INTO v\_StudentName, v\_EnrollmentCount;

EXIT WHEN c\_enrollments%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Enrollment ID: ' || v\_EnrollmentCount || ', Student: ' || v\_StudentName);

END LOOP;

CLOSE c\_enrollments;

**-- 3. Cursor FOR Loop: A more concise way to handle cursors**

FOR rec IN (SELECT e.EnrollmentID, s.Name AS StudentName, c.Title AS CourseTitle

FROM Enrollment e

JOIN Student s ON e.StudentID = s.StudentID

JOIN Course c ON e.CourseID = c.CourseID) LOOP

DBMS\_OUTPUT.PUT\_LINE('Enrollment ID: ' || rec.EnrollmentID ||

', Student: ' || rec.StudentName ||

', Course: ' || rec.CourseTitle);

END LOOP;

**-- 4. Parameterized Cursor: Fetching courses for a specific student**

OPEN c\_student\_enrollment(2); -- Passing StudentID 2 as a parameter

FOR rec IN c\_student\_enrollment LOOP

DBMS\_OUTPUT.PUT\_LINE('Student 2 is enrolled in: ' || rec.Title);

END LOOP;

CLOSE c\_student\_enrollment;

END;

/

**-- Step 12: Show the output using SELECT statements**

-- Show all enrollments

SELECT \* FROM Enrollment;

-- Show all students

SELECT \* FROM Student;

**Explanation of the Code:**

1. **Implicit Cursor**:
   * **What it does**: This is automatically created by Oracle when performing a SELECT INTO statement. It implicitly handles the result of the query.
   * **Example**: The block first uses an implicit cursor to select the Name and Email of the student with StudentID = 1 and stores these values into variables v\_StudentName and v\_StudentEmail.
2. **Explicit Cursor**:
   * **What it does**: An explicit cursor is declared by the user, and the programmer manually opens, fetches, and closes it.
   * **Example**: The c\_enrollments cursor is explicitly declared to fetch student enrollment details. The cursor is opened and then fetched inside a loop, displaying enrollment details.
3. **Cursor FOR Loop**:
   * **What it does**: This is a simplified version of an explicit cursor. It automatically handles the opening, fetching, and closing of the cursor for you.
   * **Example**: The FOR rec IN (...) loop processes the SELECT statement, which retrieves the student enrollments, and automatically closes the cursor once the loop completes.
4. **Parameterized Cursor**:
   * **What it does**: A cursor that takes a parameter as input, which is used to filter the results dynamically.
   * **Example**: The c\_student\_enrollment cursor is parameterized with p\_StudentID, and the cursor is opened with the parameter value 2 to fetch courses for student 2. It processes the result set in a FOR loop and displays the courses.