Learning Management System (LMS).

1) Use Case Specification Template: Populate the template with your project details and expected functionalities.

2) Create DDL (Data Definition Language): Define the database schema.

3) Create ER Diagram (Entity-Relationship Diagram): Visual representation of the database.

4) Extract Data: Example queries for data extraction.

5) Prepare and Upload Document: Compile everything into a document and upload it to GitHub.

---

1) Use Case Specification :

This is attached in a separate Excel file.

---

2) Create DDL (Data Definition Language):

Table 1 : Role

Create Table Role:

```
CREATE TABLE Role (
    RoleID INT PRIMARY KEY IDENTITY(1,1),
    RoleName VARCHAR(50) NOT NULL UNIQUE
);
```

Insert Some values in Table Role :

```
INSERT INTO Role (RoleName) VALUES ('Admin'), ('User');
```

Table 2 : Users

Create Table Users:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY IDENTITY(1,1),
    Username VARCHAR(50) NOT NULL UNIQUE,
    PasswordHash VARBINARY(MAX) NOT NULL,
    FullName VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    RoleID INT NOT NULL,
    FOREIGN KEY (RoleID) REFERENCES Role(RoleID)
);
```

Insert Some Values in Table Users:

```
INSERT INTO Users (Username, PasswordHash, FullName, Email, RoleID) VALUES
('dhanapal', 0x1234567890ABCDEF, 'Dhanapal', 'dhanapal.m@payoda.com', 1),
('siddarth', 0x1234567890ABCDEF, 'Siddarth', 'siddarth.s@payoda.com', 2),
('prathik', 0x1234567890ABCDEF, 'Prathik', 'prathik.b@payoda.com', 2);
```

Table 3: Course

Create Table Course:

```
CREATE TABLE Course (
    CourseID INT PRIMARY KEY IDENTITY(1,1),
    CourseName VARCHAR(100) NOT NULL UNIQUE,
    Description TEXT,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL
);
```

Insert Some Values in Table Course:

```
INSERT INTO Course (CourseName, Description, StartDate, EndDate) VALUES
('Python Basics', 'Introduction to Python programming.', '2024-09-01', '2024-12-01'),
('JavaScript Essentials', 'Learn core JavaScript concepts.', '2024-10-01', '2024-12-15'),
('Java Fundamentals', 'Basics of Java programming.', '2024-08-15', '2024-11-30'),
('Ruby on Rails', 'Introduction to web development with Ruby on Rails.', '2024-11-01', '2025-01-15'),
('C# Basics', 'Learn the basics of C# programming.', '2024-09-15', '2024-12-15');
```

Table 4 : Enrollment

Create Table Enrollment:

```
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY IDENTITY(1,1),
    UserID INT NOT NULL,
    CourseID INT NOT NULL,
    EnrollmentDate DATE NOT NULL,
    Status VARCHAR(50) DEFAULT 'Enrolled',
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    UNIQUE (UserID, CourseID)
);
```

Insert Some Values in Table Enrollment:

```
INSERT INTO Enrollment (UserID, CourseID, EnrollmentDate, Status) VALUES
(2, 1, '2024-09-01', 'Enrolled'), -- UserID 2 (Siddarth) enrolls in CourseID 1 (Python Basics)
(3, 2, '2024-10-01', 'Enrolled'), -- UserID 3 (Prathik) enrolls in CourseID 2 (JavaScript Essentials)
(2, 3, '2024-08-15', 'Enrolled'), -- UserID 2 (Siddarth) enrolls in CourseID 3 (Java Fundamentals)
(3, 4, '2024-11-01', 'Enrolled'); -- UserID 3 (Prathik) enrolls in CourseID 4 (Ruby on Rails)
```

Table 5: Assessment

Create Table Assessment:

```
CREATE TABLE Assessment (
    AssessmentID INT PRIMARY KEY IDENTITY(1,1),
    CourseID INT NOT NULL,
    AssessmentName VARCHAR(100) NOT NULL,
    AssessmentDate DATE NOT NULL,
    MaxScore INT NOT NULL,
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

Insert Some Values in Table Assessment:

```
INSERT INTO Assessment (CourseID, AssessmentName, AssessmentDate, MaxScore)
 VALUES
(1, 'Python Basics Quiz', '2024-11-15', 100), -- Assessment for CourseID 1 (Python Basics)
(2, 'JavaScript Project', '2024-12-01', 150), -- Assessment for CourseID 2 (JavaScript Essentials)
(3, 'Java Fundamentals Test', '2024-10-15', 120), -- Assessment for CourseID 3 (Java Fundamentals)
(4, 'Ruby on Rails Assignment', '2024-12-15', 200); -- Assessment for CourseID 4 (Ruby on Rails)
```

Table 6: Result

Create Table Result:

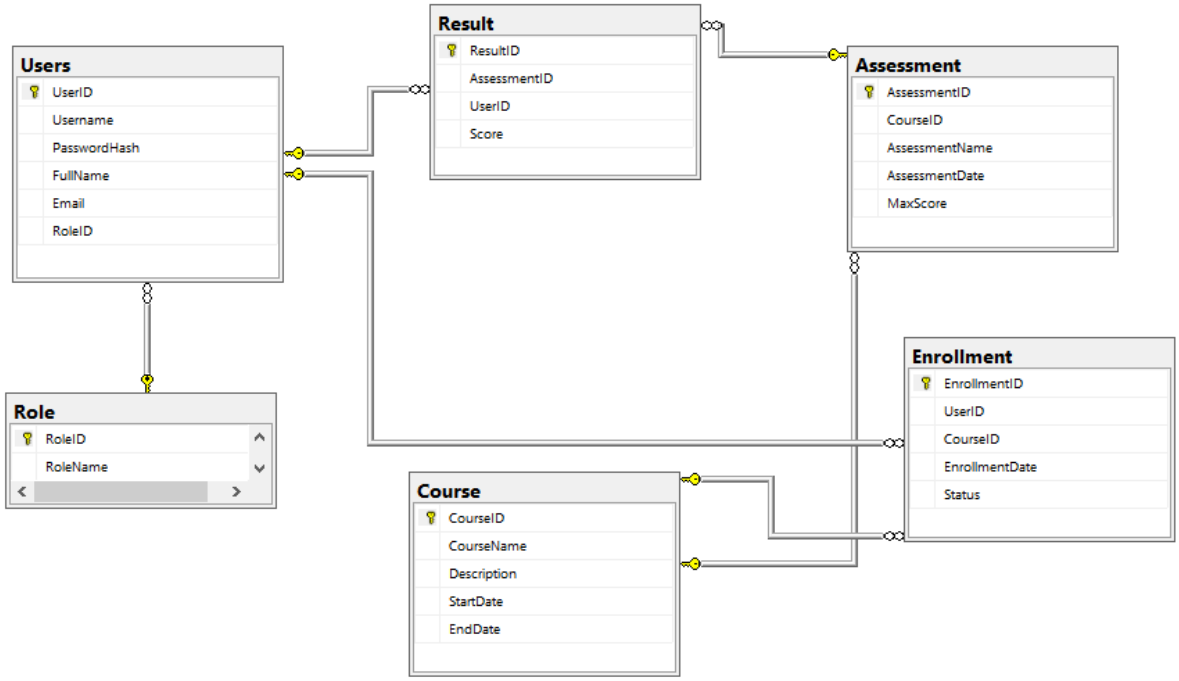
```
CREATE TABLE Result (
    ResultID INT PRIMARY KEY IDENTITY(1,1),
    AssessmentID INT NOT NULL,
    UserID INT NOT NULL,
    Score INT NOT NULL,
    FOREIGN KEY (AssessmentID) REFERENCES Assessment(AssessmentID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    UNIQUE (AssessmentID, UserID)
);
```

Insert Some Values in Table Result:

```
INSERT INTO Result (AssessmentID, UserID, Score) VALUES
(1, 2, 85), -- AssessmentID 1 (Python Basics Quiz) by UserID 2 (Siddarth)
(2, 3, 140), -- AssessmentID 2 (JavaScript Project) by UserID 3 (Prathik)
(3, 2, 110), -- AssessmentID 3 (Java Fundamentals Test) by UserID 2 (Siddarth)
(4, 3, 190); -- AssessmentID 4 (Ruby on Rails Assignment) by UserID 3 (Prathik)
```

---

3) Create ER Diagram (Entity-Relationship Diagram):



---

4) Extract Data:

Retrieve All Records from a Table

- **Query Task:** Select all records from the `Users` table.

  SELECT * FROM Users;

	UserID	Username	PasswordHash	FullName	Email	RoleID
1	2	dhanapal	0x1234567890ABCDEF	Dhanapal	dhanapal.m@payoda.com	1
2	3	siddarth	0x1234567890ABCDEF	Siddarth	siddarth.s@payoda.com	2
3	4	prathik	0x1234567890ABCDEF	Prathik	prathik.b@payoda.com	2

Filter Records Based on a Condition

- **Query Task:** Select all enrollments from the `Enrollment` table where the enrollment date is after January 1, 2023.

SELECT * FROM Enrollment WHERE EnrollmentDate > '2023-01-01';

	EnrollmentID	UserID	CourseID	EnrollmentDate	Status
1	1	2	1	2024-09-01	Enrolled
2	2	3	2	2024-10-01	Enrolled
3	3	2	3	2024-08-15	Enrolled
4	4	3	4	2024-11-01	Enrolled

Join Two Tables

- **Query Task:** Retrieve the names of students along with their enrollment IDs from the `Users` and `Enrollment` tables

  SELECT s.FullName, e.EnrollmentID

  FROM Student s

  INNER JOIN Enrollment e ON s.StudentID = e.StudentID

	FullName	EnrollmentID
1	Dhanapal	1
2	Dhanapal	3
3	Siddarth	2
4	Siddarth	4

Aggregate Data Using Group By

- **Query Task:** Find the total number of enrollments for each course.

  SELECT CourseID, COUNT(*) AS TotalEnrollments

  FROM Enrollment

  GROUP BY CourseID;

	CourseID	TotalEnrollments
1	1	1
2	2	1
3	3	1
4	4	1

Filter Groups Using HAVING

- **Query Task:** Retrieve the course IDs and their total number of enrollments, but only for courses that have more than 5 enrollments.

  SELECT CourseID, COUNT(*) AS TotalEnrollments

  FROM Enrollment

  GROUP BY CourseID

  HAVING COUNT(*) >= 1;

	CourseID	TotalEnrollments
1	1	1
2	2	1
3	3	1
4	4	1

Order Results Using ORDER BY

- **Query Task:** Select all courses from the `Course` table and order them by start date in descending order.

  SELECT * FROM Course ORDER BY StartDate DESC;

	CourseID	CourseName	Description	StartDate	EndDate
1	4	Ruby on Rails	Introduction to web development with Ruby on Ra...	2024-11-01	2025-01-15
2	2	JavaScript Essentials	Learn core JavaScript concepts.	2024-10-01	2024-12-15
3	5	C# Basics	Learn the basics of C# programming.	2024-09-15	2024-12-15
4	1	Python Basics	Introduction to Python programming.	2024-09-01	2024-12-01
5	3	Java Fundamentals	Basics of Java programming.	2024-08-15	2024-11-30

Retrieve Data with a Subquery

- **Query Task:** Find the names of students who have scored more than 90 in any assessment.

```
SELECT FullName
FROM Student
WHERE StudentID IN (
  SELECT StudentID
  FROM Result
  WHERE Score > 90
);
```

	FullName
1	Dhanapal
2	Siddarth

Use CASE Statements

- **Query Task:** Retrieve assessment results along with a column that indicates if the score is 'High' (>=90), 'Medium' (>=70 and <90), or 'Low' (<70).

```
SELECT StudentID, AssessmentID, Score,
  CASE
    WHEN Score >= 90 THEN 'High'
    WHEN Score >= 70 THEN 'Medium'
    ELSE 'Low'
```

```
    END AS ScoreCategory
FROM Result;
```

	UserID	AssessmentID	Score	ScoreCategory
1	2	1	85	Medium
2	3	2	140	High
3	2	3	110	High
4	3	4	190	High
