Q.5) **Online Education Platform Management System Database**

The Online Education Platform Management System is designed to manage courses, students, instructors, enrollments, assessments, and payments efficiently. This database solution supports the enrollment of students in various courses, facilitates instructor-student interactions, tracks assessments, and processes payments while ensuring the smooth operation of an online learning environment.

### 1. Database Tables Design

**Table: Students Table**

**This table stores information about students enrolled in the platform.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| student\_id | INT (PK) | Unique identifier for each student |
| first\_name | VARCHAR(50) | Student's first name |
| last\_name | VARCHAR(50) | Student's last name |
| email | VARCHAR(100) | Student's email address |
| phone\_number | VARCHAR(15) | Student's contact number |
| date\_of\_birth | DATE | Student's date of birth |

**Table: Instructors Table**

**This table stores information about the instructors teaching courses.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| instructor\_id | INT (PK) | Unique identifier for each instructor |
| first\_name | VARCHAR(50) | Instructor's first name |
| last\_name | VARCHAR(50) | Instructor's last name |
| email | VARCHAR(100) | Instructor's email address |
| expertise\_area | VARCHAR(100) | Area of expertise of the instructor (e.g., Mathematics) |

**Table: Courses Table**

**This table contains details about the courses available on the platform.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| course\_id | INT (PK) | Unique identifier for each course |
| course\_name | VARCHAR(100) | Name of the course |
| course\_description | TEXT | Description of the course content |
| instructor\_id | INT (FK) | References Instructors(instructor\_id) |
| prerequisite\_id | INT (FK) | References another course that serves as a prerequisite (nullable) |
| max\_seats | INT | Maximum number of students that can enroll |

**Table: Enrollments Table**

**This table tracks which students are enrolled in which courses.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| enrollment\_id | INT (PK) | Unique identifier for each enrollment |
| student\_id | INT (FK) | References Students(student\_id) |
| course\_id | INT (FK) | References Courses(course\_id) |
| enrollment\_date | DATE | Date when the student enrolled in the course |

**Table: Assessments Table**

**This table stores information about assessments given in the courses.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| assessment\_id | INT (PK) | Unique identifier for each assessment |
| course\_id | INT (FK) | References Courses(course\_id) |
| assessment\_name | VARCHAR(100) | Name of the assessment (e.g., Midterm, Final Exam) |
| assessment\_date | DATE | Date when the assessment is scheduled |

**Table: Payments Table**

**This table manages payments made by students for courses.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| payment\_id | INT (PK) | Unique identifier for each payment |
| student\_id | INT (FK) | References Students(student\_id) |
| course\_id | INT (FK) | References Courses(course\_id) |
| payment\_date | DATE | Date when payment was made |
| amount\_paid | DECIMAL(10,2) | Amount paid for the course |
| payment\_status | VARCHAR(20) | Status of payment (e.g., 'Paid', 'Pending') |

### 2. Constraints for Referential Integrity

### ** Foreign Keys:**

### **student\_id in Enrollments references Students(student\_id).**

### **course\_id in Enrollments, Payments, and Assessments references Courses(course\_id).**

### **instructor\_id in Courses references Instructors(instructor\_id).**

### **prerequisite\_id in Courses references Courses(course\_id) (nullable).**

### ** Primary Keys:**

### **Each table has a primary key for unique identification of rows.**

### ** Check Constraints:**

### **Ensure that max\_seats in Courses is positive.**

### **Enforce that the payment\_status field contains valid values (e.g., 'Paid', 'Pending').**

### 3. Stored Procedures

#### a. Enroll Student in Course

#### **This procedure registers a student in a course, ensuring that there are available seats and that the student meets the prerequisites.**

#### CREATE PROCEDURE EnrollStudent(IN studentId INT, IN courseId INT)

#### BEGIN

#### DECLARE availableSeats INT;

#### DECLARE prerequisiteMet BOOLEAN;

#### -- Check if there are available seats

#### SELECT max\_seats - COUNT(\*) INTO availableSeats

#### FROM Enrollments

#### WHERE course\_id = courseId;

#### -- Check if the course has available seats

#### IF availableSeats > 0 THEN

#### -- Check if prerequisites are met

#### SELECT prerequisite\_id INTO prerequisiteMet FROM Courses WHERE course\_id = courseId;

#### IF prerequisiteMet IS NULL OR prerequisiteMet IN (SELECT course\_id FROM Enrollments WHERE student\_id = studentId) THEN

#### -- Enroll student if prerequisites are met

#### INSERT INTO Enrollments (student\_id, course\_id, enrollment\_date)

#### VALUES (studentId, courseId, NOW());

#### ELSE

#### SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Prerequisite not met for the course';

#### END IF;

#### ELSE

#### SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No available seats in the course';

#### END IF;

#### END;

#### b. Record Assessment for a Course

**This procedure registers an assessment for a specific course.**

#### CREATE PROCEDURE RecordAssessment(IN courseId INT, IN assessmentName VARCHAR(100), IN assessmentDate DATE)

#### BEGIN

#### INSERT INTO Assessments (course\_id, assessment\_name, assessment\_date)

#### VALUES (courseId, assessmentName, assessmentDate);

#### END;

#### c. Comment on Post

#### **This procedure processes a payment made by a student for a course.**

#### CREATE PROCEDURE ProcessPayment(IN studentId INT, IN courseId INT, IN amount DECIMAL(10,2))

### BEGIN

### DECLARE courseFee DECIMAL(10,2);

### -- Get course fee

### SELECT price INTO courseFee FROM Courses WHERE course\_id = courseId;

### -- Ensure the amount paid is equal to the course fee

### IF amount = courseFee THEN

### INSERT INTO Payments (student\_id, course\_id, payment\_date, amount\_paid, payment\_status)

### VALUES (studentId, courseId, NOW(), amount, 'Paid');

### ELSE

### SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Payment amount does not match course fee';

### END IF;

### END;

### 4. Triggers

#### a. Trigger to Update Course Seat Availability

**This trigger ensures that available seats in a course are updated each time a new enrollment occurs.**

#### CREATE TRIGGER AfterEnrollStudent

#### AFTER INSERT ON Enrollments

#### FOR EACH ROW

#### BEGIN

#### UPDATE Courses

#### SET max\_seats = max\_seats - 1

#### WHERE course\_id = NEW.course\_id;

#### END;

#### b. Trigger for Assessment Reminder

**This trigger sends an automatic reminder when an assessment is approaching.**

#### CREATE TRIGGER AssessmentReminder

#### BEFORE INSERT ON Assessments

#### FOR EACH ROW

#### BEGIN

#### DECLARE reminderDate DATE;

#### -- Set reminder to 3 days before the assessment date

#### SET reminderDate = DATE\_SUB(NEW.assessment\_date, INTERVAL 3 DAY);

#### -- Send reminder notification to student (this can be extended to an actual notification system)

#### INSERT INTO Notifications (student\_id, message, notification\_date)

#### VALUES (NEW.student\_id, CONCAT('Reminder: Upcoming assessment for ', NEW.assessment\_name), reminderDate);

#### END;

### 5. SQL Queries for Reports

#### a. Course Completion Rates Report

#### SELECT c.course\_name,

#### COUNT(e.student\_id) AS total\_enrolled,

#### COUNT(a.student\_id) AS total\_completed

#### FROM Courses c

#### LEFT JOIN Enrollments e ON c.course\_id = e.course\_id

#### LEFT JOIN Assessments a ON e.student\_id = a.student\_id

#### GROUP BY c.course\_id;

#### b. Most Popular Courses Report

#### SELECT c.course\_name, COUNT(e.student\_id) AS total\_enrolled

#### FROM Courses c

#### JOIN Enrollments e ON c.course\_id = e.course\_id

#### GROUP BY c.course\_id

#### ORDER BY total\_enrolled DESC;

#### C. Instructor Effectiveness Report

SELECT i.first\_name, i.last\_name, COUNT(a.student\_id) AS total\_assessments\_passed

FROM Instructors i

JOIN Courses c ON i.instructor\_id = c.instructor\_id

JOIN Assessments a ON c.course\_id = a.course\_id

GROUP BY i.instructor\_id;

**D. Revenue per Course**

SELECT c.course\_name, SUM(p.amount\_paid) AS total\_revenue

FROM Courses c

JOIN Payments p ON c.course\_id = p.course\_id

GROUP BY c.course\_id;

**Conclusion**

**The Online Education Platform Management System provides a comprehensive database structure to efficiently manage courses, students, instructors, assessments, and payments. It ensures proper enrollment, manages prerequisites, and tracks student progress through assessments. The stored procedures automate critical processes such as enrollment, assessment recording, and payment processing, while the triggers ensure real-time updates to course availability and reminders for assessments. The system also supports reporting on course completion, instructor effectiveness, and revenue generation, enabling platform administrators to make informed decisions and enhance user experience.**