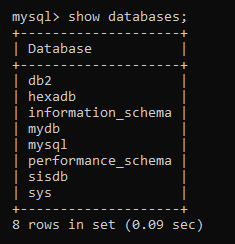
Task 1: Database design

1. **Create the database named “SISDB”.**



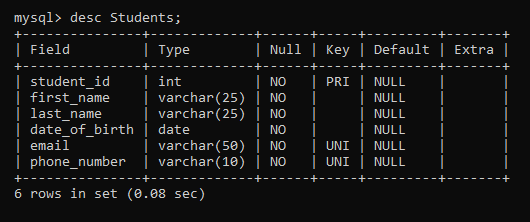
Database named “SISDB” is created.



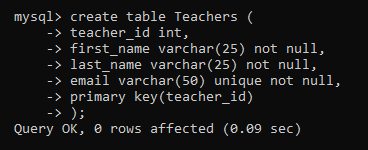
1. **Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.**
2. **Students**
3. **Courses**
4. **Enrollments**
5. **Teacher**
6. **Payments**

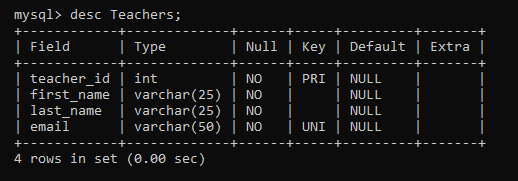
**Students Table**

****

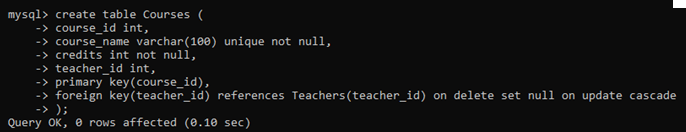
****

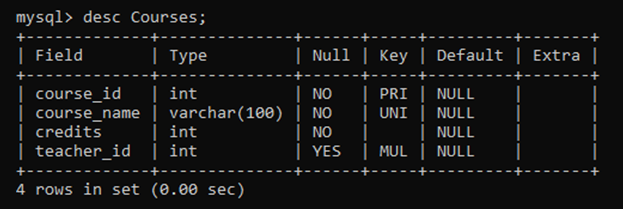
**Teachers Table**

****

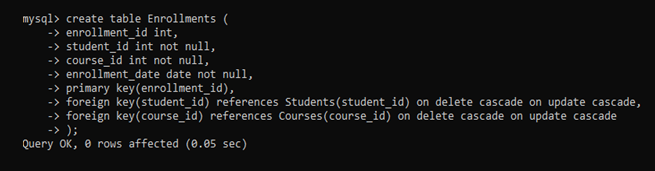
****

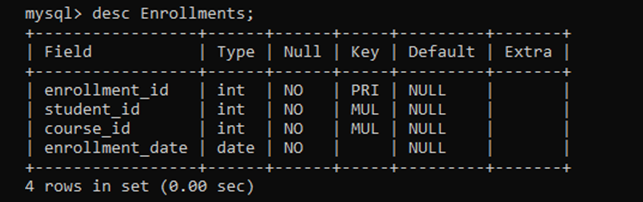
**Courses Table**

****

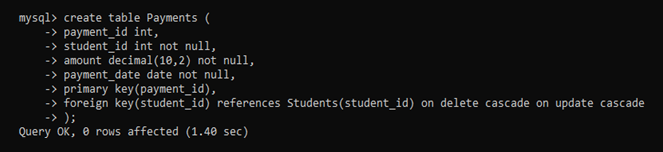
****

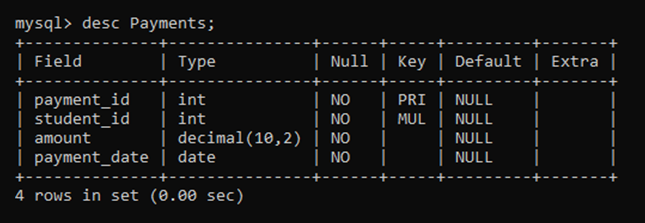
**Enrollments Table**

****

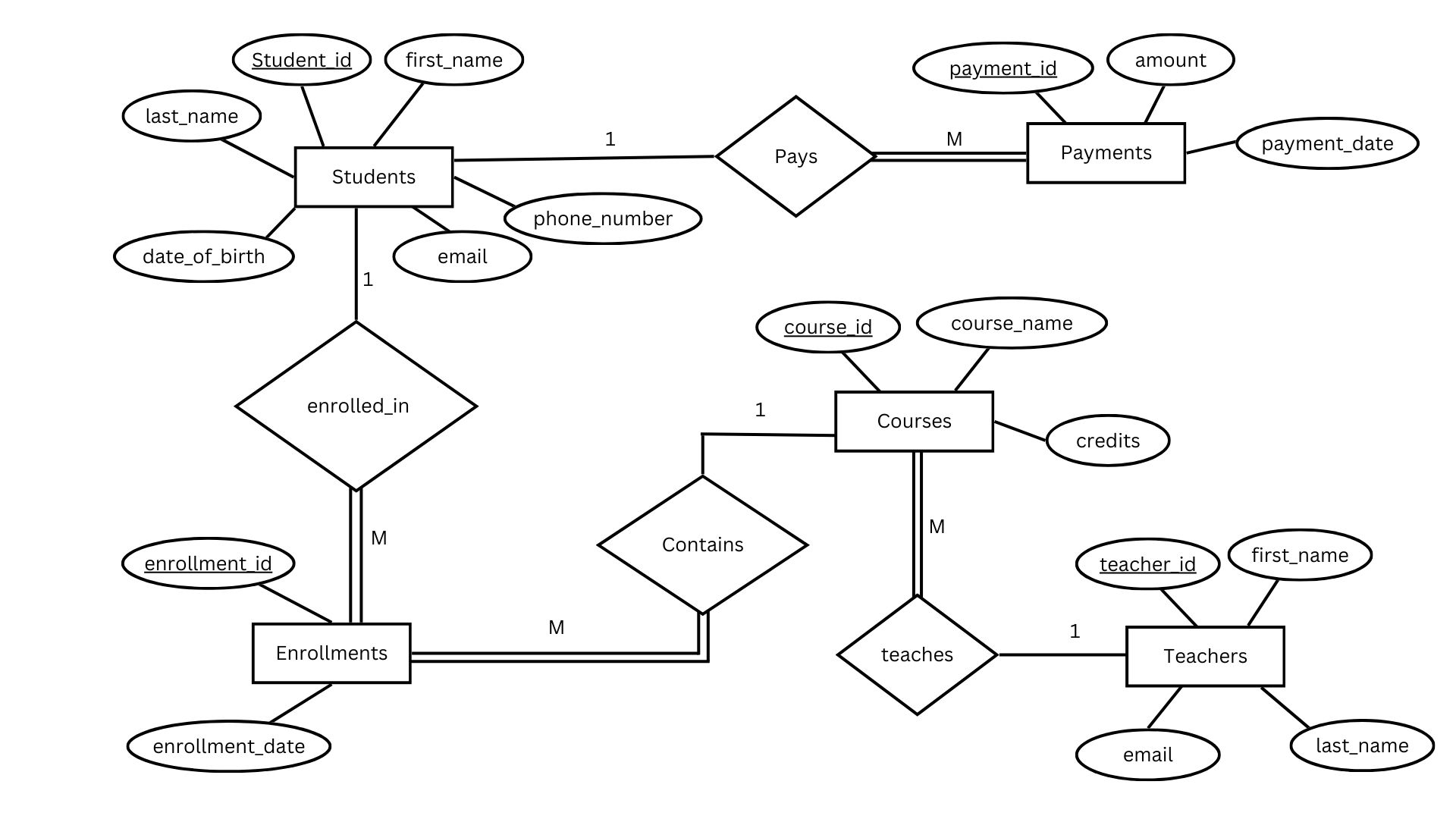
****

**Payments Table**

****

****

1. **Create an ERD (Entity Relationship Diagram) for the database.**

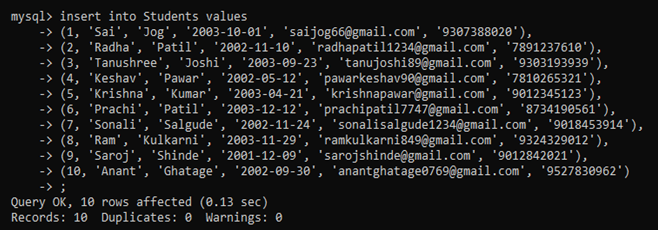
****

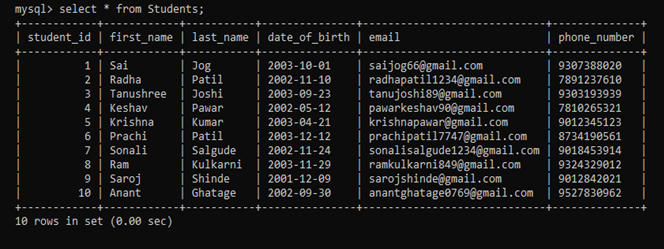
1. **Create appropriate Primary Key and Foreign Key constraints for referential integrity.**

**Already done.**

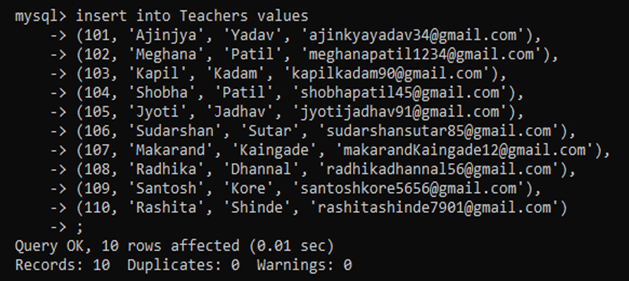
1. **Insert at least 10 sample records into each of the following tables.**
2. **Students**
3. **Courses**
4. **Enrollments**
5. **Teacher**
6. **Payments**

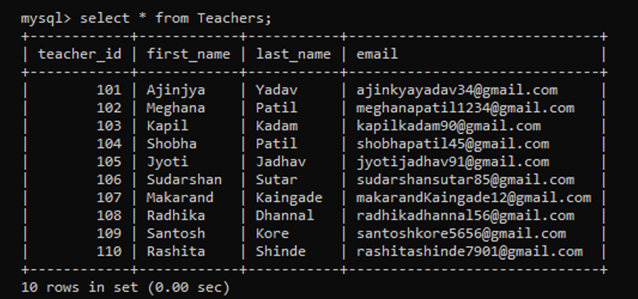
**Students Table:**

****

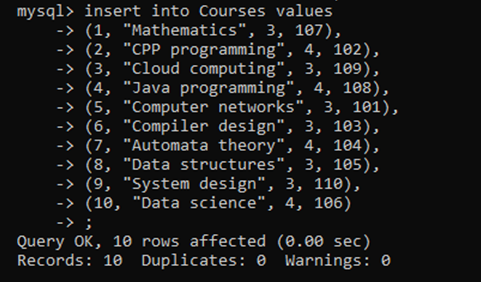
****

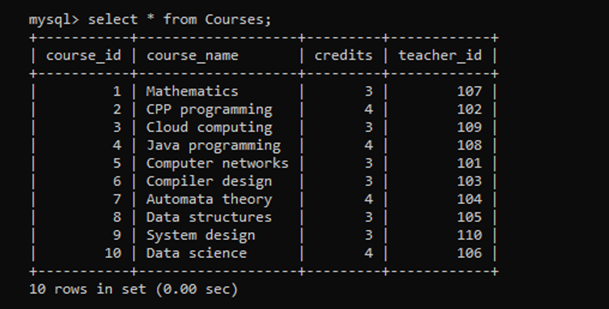
**Teachers Table:**

****

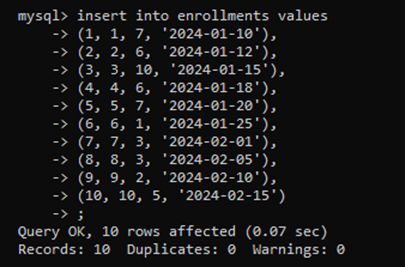
****

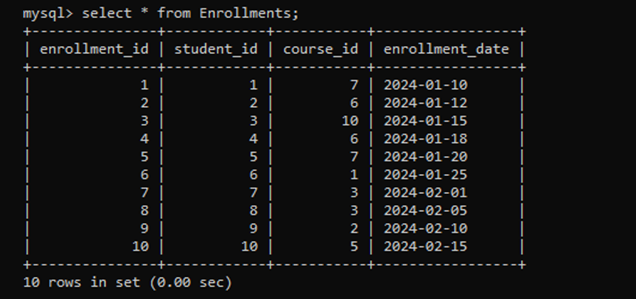
**Courses Table:**

****

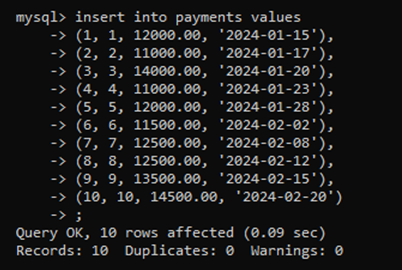
****

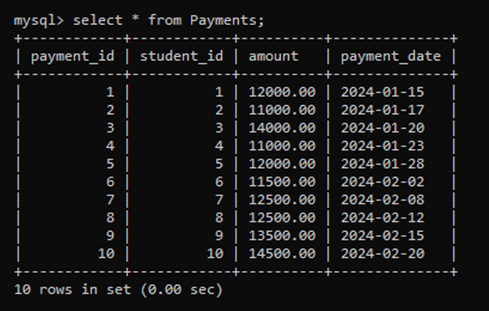
**Enrollments Table:**

****

****

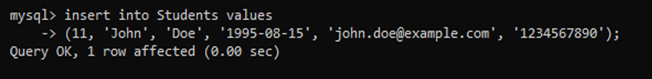
**Payments Table:**

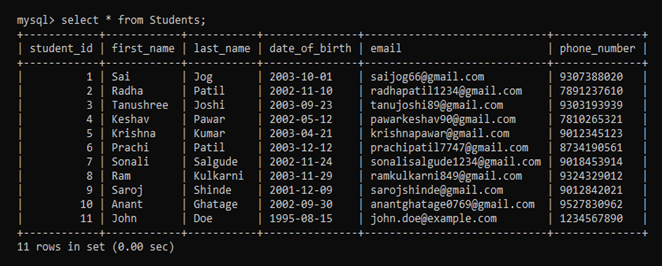
****

****

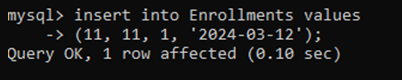
Task 2: Select, Where, Between, AND, LIKE

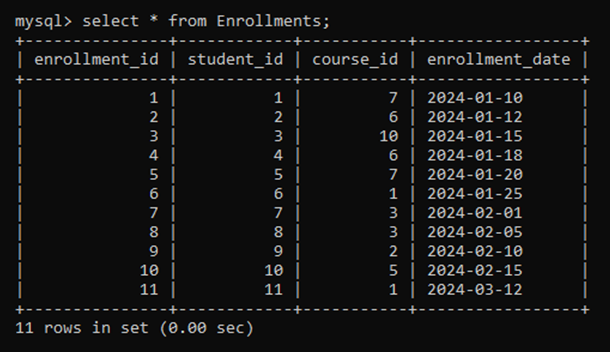
1. **Write an SQL query to insert a new student into the "Students" table with the following details:**
2. **First Name: John**
3. **Last Name: Doe**
4. **Date of Birth: 1995-08-15**
5. **Email: john.doe@example.com**
6. **Phone Number: 1234567890**

****

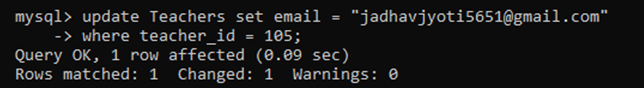
****

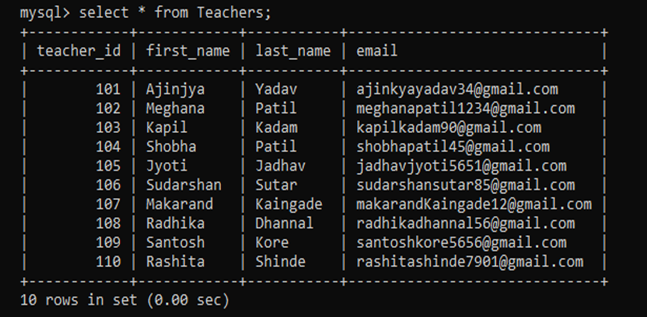
1. **Write an SQL query to enroll a student in a course. Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date.**

****

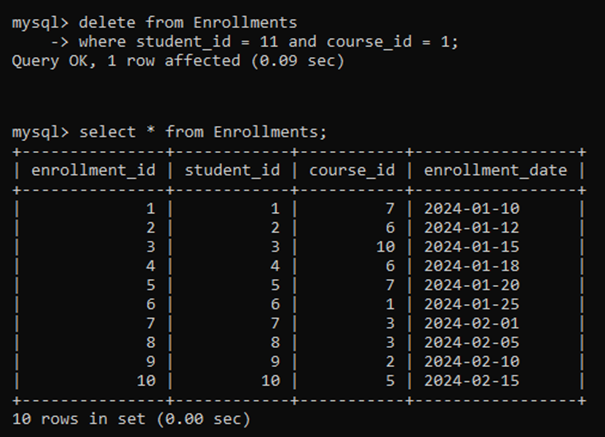
****

1. **Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.**

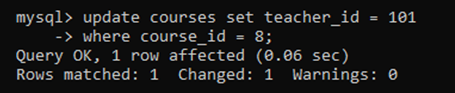
****

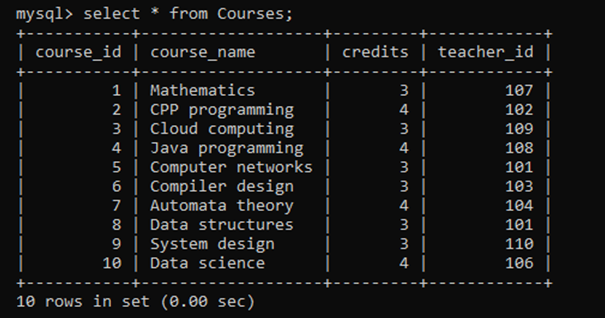
****

1. **Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.**

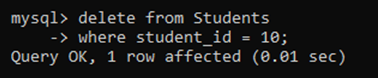
****

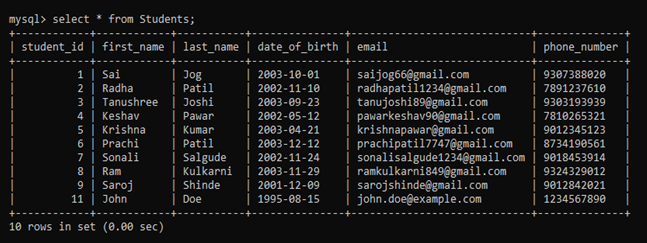
1. **Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.**

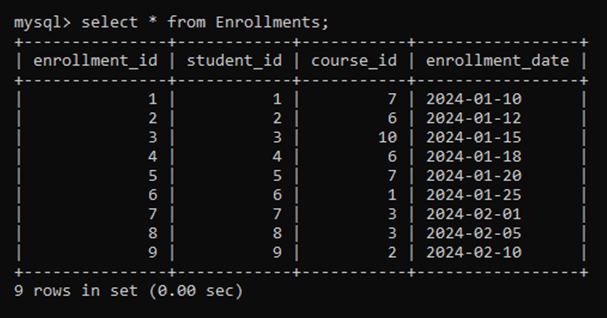
****

****

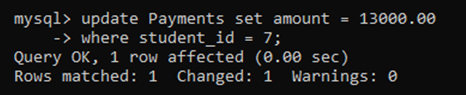
1. **Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.**

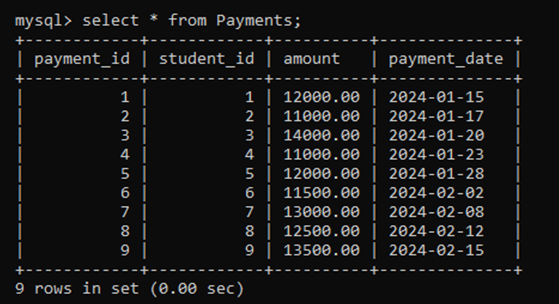
****

****

****

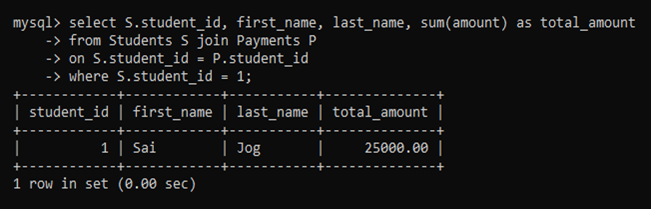
1. **Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.**

****

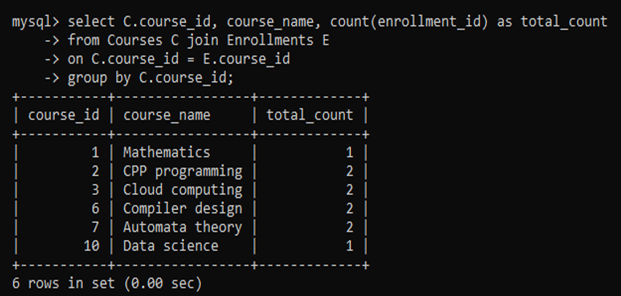
****

Task 3. Aggregate functions, Having, Order By, GroupBy and Joins

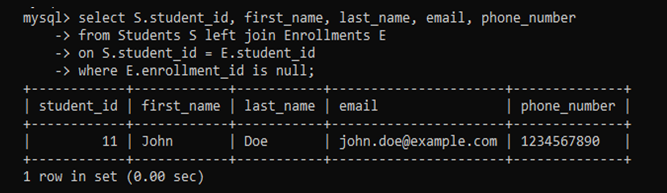
1. **Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.**

****

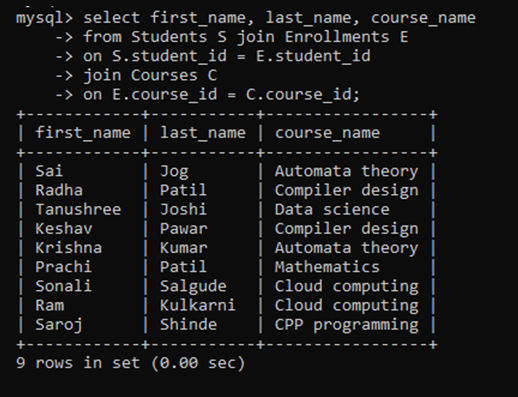
1. **Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.**

****

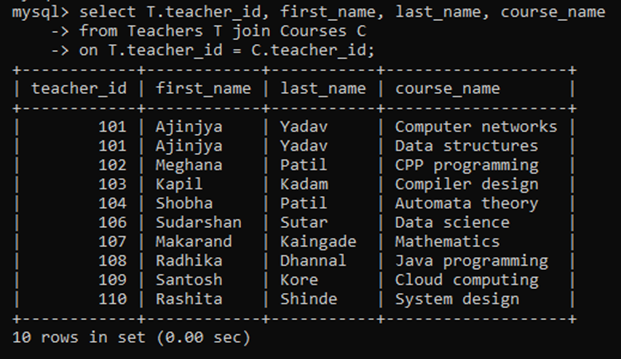
1. **Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.**

****

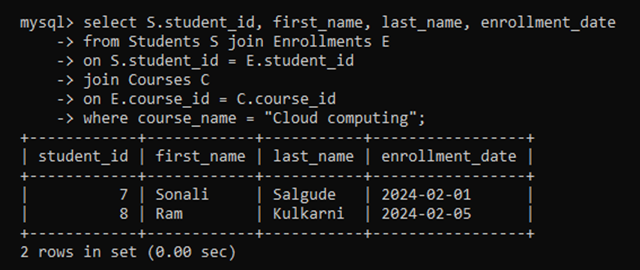
1. **Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.**

****

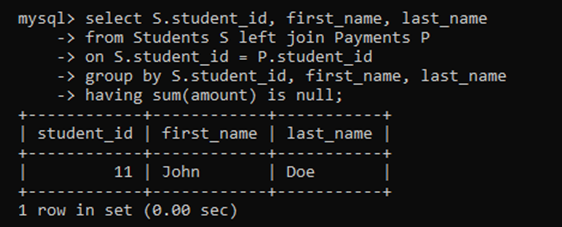
1. **Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.**

****

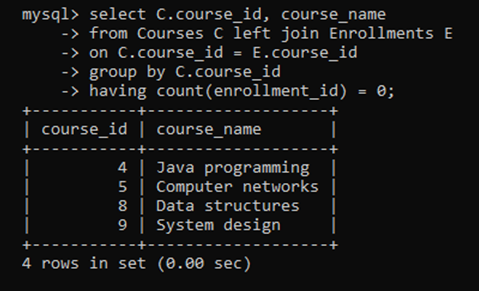
1. **Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables.**

****

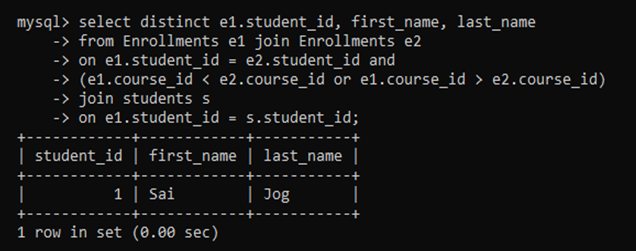
1. **Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.**

****

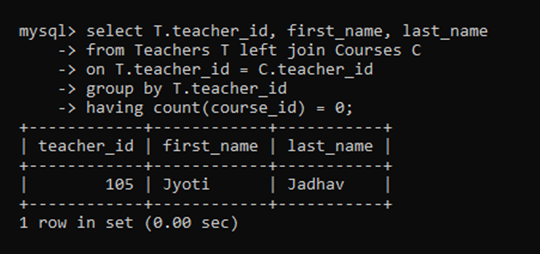
1. **Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.**

****

1. **Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.**

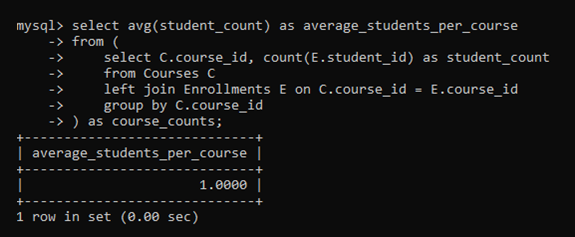
****

1. **Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Courses" table and filter for teachers with NULL course assignments.**

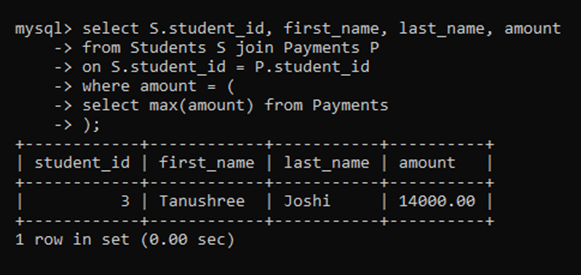
****

Task 4. Subquery and its type

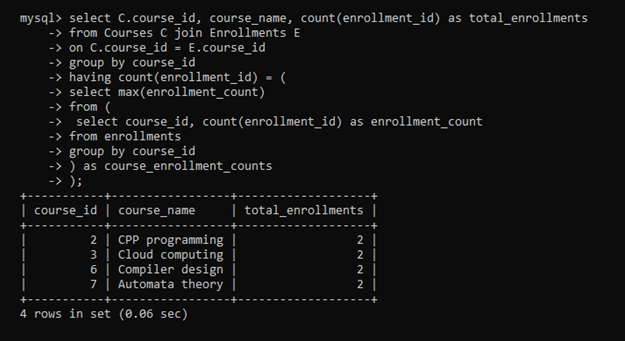
1. **Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.**

****

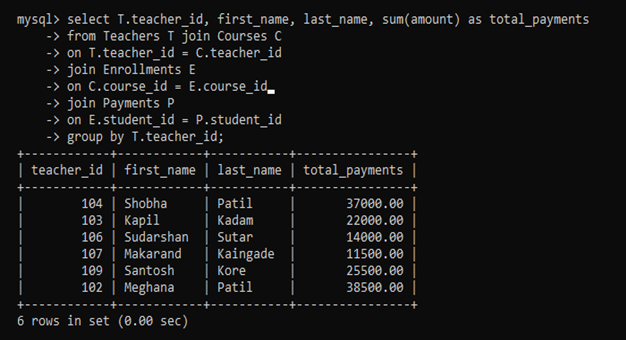
1. **Identify the student(s) who made the highest payment. Use a subquery to find the maximum payment amount and then retrieve the student(s) associated with that amount.**

****

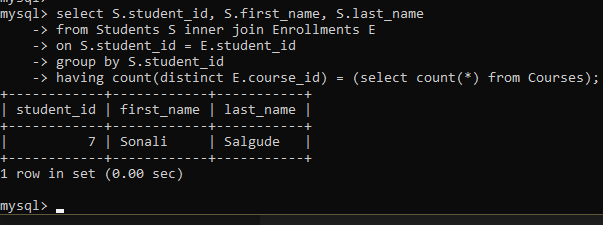
1. **Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.**

****

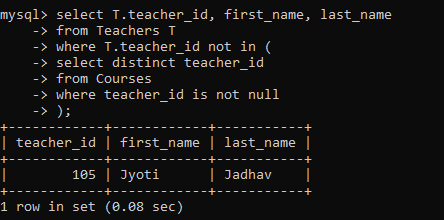
1. **Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.**

****

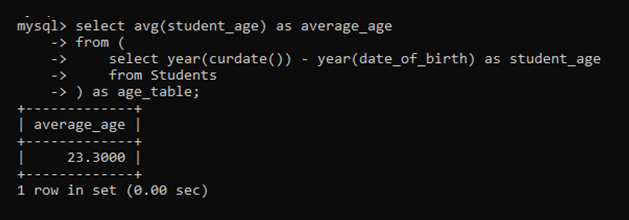
1. **Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.**

****

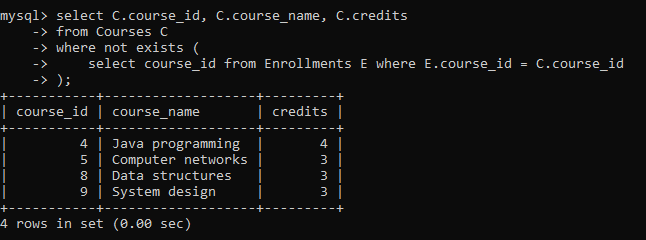
1. **Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.**

****

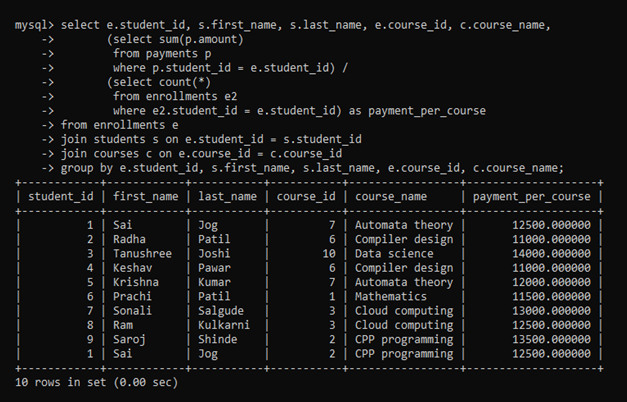
1. **Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.**

****

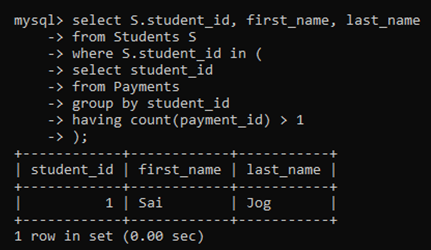
1. **Identify courses with no enrollments. Use subqueries to find courses without enrollment records.**

****

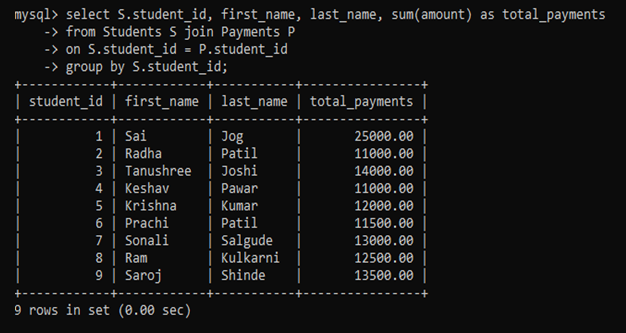
1. **Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.**

****

1. **Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.**

****

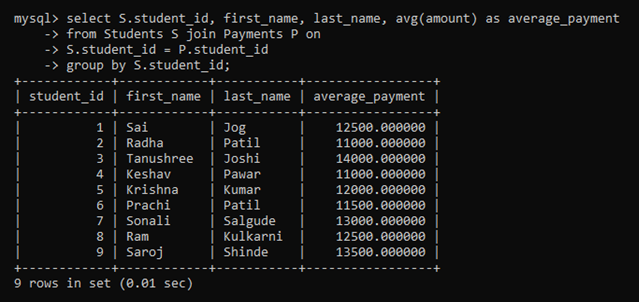
1. **Write an SQL query to calculate the total payments made by each student. Join the "Students" table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.**

****

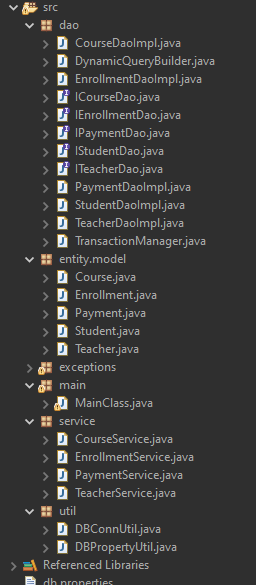
1. **Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.**

****

1. **Calculate the average payment amount made by students. Use JOIN operations between the "Students" table and the "Payments" table and GROUP BY to calculate the average.**

****

**Project Directory Structure:**

****

**Task 1: Define Classes with attributes**

**Task 2: Add constructors**

**Task 3: Implement Methods**

**Task 5: Implement the collections and update the constructor**

**Course.java**

package entity.model;

import java.util.List;

import service.EnrollmentService;

import exceptions.InvalidCourseDataException;

import java.util.ArrayList;

public class Course {

private int courseID;

private String courseName;

private String courseCode;

private int credits;

private Teacher instructorName;

private List<Enrollment> enrollments;

public Course() {

this.enrollments = new ArrayList<>();

}

public Course(int courseID, String courseName, int credits, String courseCode, Teacher instructorName) {

this();

this.courseID = courseID;

this.courseName = courseName;

this.credits = credits;

this.courseCode = courseCode;

this.instructorName = instructorName;

}

public void addEnrollment(Enrollment e) {

enrollments.add(e);

}

public List<Enrollment> getEnrollmentsList() {

return enrollments;

}

public void assignTeacher(Teacher teacher) throws exceptions.InvalidTeacherDataException {

if (teacher == null || teacher.getFirstName() == null) {

throw new exceptions.InvalidTeacherDataException("Invalid teacher data provided.");

}

this.instructorName = teacher;

System.***out***.println("✅ Teacher assigned to course " + courseName);

}

public void updateCourseInfo(String courseCode, String courseName, String instructor)

throws InvalidCourseDataException {

if (courseCode == null || courseName == null) {

throw new InvalidCourseDataException("Course code and name must not be null.");

}

this.courseCode = courseCode;

this.courseName = courseName;

}

public void displayCourseInfo() {

System.***out***.println("Course ID: " + courseID);

System.***out***.println("Name : " + courseName);

System.***out***.println("Code : " + courseCode);

System.***out***.println("Instructor: " + (instructorName != null ? instructorName.getFirstName() : "None"));

}

public List<Enrollment> getEnrollments() {

return new EnrollmentService().getEnrollmentsByCourse(courseID);

}

public Teacher getTeacher() {

return instructorName;

}

public int getCourseID() {

return courseID;

}

public void setCourseID(int courseID) {

this.courseID = courseID;

}

public String getCourseName() {

return courseName;

}

public void setCourseName(String courseName) {

this.courseName = courseName;

}

public String getCourseCode() {

return courseCode;

}

public void setCourseCode(String courseCode) {

this.courseCode = courseCode;

}

public Teacher getInstructorName() {

return instructorName;

}

public void setInstructorName(Teacher instructorName) {

this.instructorName = instructorName;

}

public int getCredits() {

return credits;

}

public void setCredits(int credits) {

this.credits = credits;

}

}

**Enrollment.java**

package entity.model;

import java.util.Date;

public class Enrollment {

private int enrollmentID;

private Student student;

private Course course;

private Date enrollmentDate;

public Enrollment() {

super();

}

public Enrollment(int enrollmentID, Student student, Course course, Date enrollmentDate) {

this.enrollmentID = enrollmentID;

this.student = student;

this.course = course;

this.enrollmentDate = enrollmentDate;

}

public Student getStudent() {

return student;

}

public Course getCourse() {

return course;

}

public int getEnrollmentID() {

return enrollmentID;

}

public void setEnrollmentID(int enrollmentID) {

this.enrollmentID = enrollmentID;

}

public void setStudent(Student student) {

this.student = student;

}

public void setCourse(Course course) {

this.course = course;

}

public Date getEnrollmentDate() {

return enrollmentDate;

}

public void setEnrollmentDate(Date enrollmentDate) {

this.enrollmentDate = enrollmentDate;

}

}

**Payment.java**

package entity.model;

import java.util.Date;

import exceptions.PaymentValidationException;

public class Payment {

private int paymentID;

private Student student;

private double amount;

private Date date;

public Payment() {

super();

}

public Payment(int paymentID, Student student, double amount, java.util.Date date) {

this.paymentID = paymentID;

this.student = student;

this.amount = amount;

this.date = date;

}

public Student getStudent() {

return student;

}

public double getPaymentAmount() {

return amount;

}

public Date getPaymentDate() {

return date;

}

public int getPaymentID() {

return paymentID;

}

public void setPaymentID(int paymentID) {

this.paymentID = paymentID;

}

public void setStudent(Student student) {

this.student = student;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) throws PaymentValidationException {

if (amount <= 0) throw new PaymentValidationException("Amount must be greater than 0.");

this.amount = amount;

}

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

}

**Student.java**

package entity.model;

import exceptions.InvalidStudentDataException;

import service.EnrollmentService;

import service.PaymentService;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

public class Student {

private int studentID;

private String firstName;

private String lastName;

private Date dateOfBirth;

private String email;

private String phoneNumber;

private List<Enrollment> enrollments;

private List<Payment> payments;

public Student() {

this.enrollments = new ArrayList<>();

this.payments = new ArrayList<>();

}

public Student(int studentID, String firstName, String lastName, Date dateOfBirth, String email, String phoneNumber) {

this();

this.studentID = studentID;

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.email = email;

this.phoneNumber = phoneNumber;

}

public void addEnrollment(Enrollment e) {

if (e != null && !enrollments.contains(e)) {

enrollments.add(e);

}

}

public void addPayment(Payment p) {

if (p != null && !payments.contains(p)) {

payments.add(p);

}

}

public List<Enrollment> getEnrollments() {

return enrollments;

}

public List<Payment> getPayments() {

return payments;

}

public void enrollInCourse(Course course) {

new EnrollmentService().enrollExistingStudent(this, course);

}

public void updateStudentInfo(String firstName, String lastName, Date dateOfBirth, String email, String phoneNumber) throws InvalidStudentDataException {

if (firstName == null || email == null) {

throw new InvalidStudentDataException("Name and email must not be null.");

}

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.email = email;

this.phoneNumber = phoneNumber;

}

public void makePayment(double amount, Date paymentDate) {

new PaymentService().recordStudentPayment(this.studentID, amount, paymentDate);

}

public void displayStudentInfo() {

System.***out***.println("Student ID: " + studentID);

System.***out***.println("Name : " + firstName + " " + lastName);

System.***out***.println("DOB : " + dateOfBirth);

System.***out***.println("Email : " + email);

System.***out***.println("Phone : " + phoneNumber);

}

public List<Course> getEnrolledCourses() {

return new EnrollmentService().getCoursesForStudent(studentID);

}

public List<Payment> getPaymentHistory() {

return new PaymentService().getPaymentsForStudent(studentID);

}

// Getters and Setters

public int getStudentID() {

return studentID;

}

public void setStudentID(int studentID) {

this.studentID = studentID;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

}

**Teacher.java**

package entity.model;

import exceptions.InvalidTeacherDataException;

import java.util.ArrayList;

import java.util.List;

public class Teacher {

private int teacherID;

private String firstName;

private String lastName;

private String email;

private List<Course> assignedCourses;

public Teacher() {

this.assignedCourses = new ArrayList<>();

}

public Teacher(int teacherID, String firstName, String lastName, String email) {

this();

this.teacherID = teacherID;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

}

// 🔄 Use method name "assignCourse" as expected

public void assignCourse(Course course) {

if (course != null && !assignedCourses.contains(course)) {

assignedCourses.add(course);

}

}

public List<Course> getAssignedCourses() {

return assignedCourses;

}

public void updateTeacherInfo(String name, String email, String expertise) throws InvalidTeacherDataException {

if (name == null || email == null) {

throw new InvalidTeacherDataException("Teacher name and email are required.");

}

String[] split = name.split(" ");

this.firstName = split[0];

this.lastName = split.length > 1 ? split[1] : "";

this.email = email;

}

public void displayTeacherInfo() {

System.***out***.println("Teacher ID : " + teacherID);

System.***out***.println("Name : " + firstName + " " + lastName);

System.***out***.println("Email : " + email);

}

// Getters and Setters

public int getTeacherID() {

return teacherID;

}

public void setTeacherID(int teacherID) {

this.teacherID = teacherID;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**Task 4: Exceptions handling and Custom Exceptions**

Implementing custom exceptions allows you to define and throw exceptions tailored to specific

situations or business logic requirements.

**CourseNotFoundException.java**

package exceptions;

public class CourseNotFoundException extends RuntimeException {

public CourseNotFoundException(String message) {

super(message);

}

}

**DuplicateEnrollmentException.java**

package exceptions;

public class DuplicateEnrollmentException extends RuntimeException {

public DuplicateEnrollmentException(String message) {

super(message);

}

}

**InsufficentFundsException.java**

package exceptions;

public class InsufficientFundsException extends RuntimeException {

public InsufficientFundsException(String message) {

super(message);

}

}

**InvalidCourseDataException.java**

package exceptions;

public class InvalidCourseDataException extends RuntimeException {

public InvalidCourseDataException(String message) {

super(message);

}

}

**InvalidEnrollmentDataException.java**

package exceptions;

public class InvalidEnrollmentDataException extends RuntimeException {

public InvalidEnrollmentDataException(String message) {

super(message);

}

}

**InvalidStudentDataException.java**

package exceptions;

public class InvalidStudentDataException extends RuntimeException {

public InvalidStudentDataException(String message) {

super(message);

}

}

**InvalidTeacherDataException.java**

package exceptions;

public class InvalidTeacherDataException extends RuntimeException {

public InvalidTeacherDataException(String message) {

super(message);

}

}

**PaymentValidationException.java**

package exceptions;

public class PaymentValidationException extends RuntimeException {

public PaymentValidationException(String message) {

super(message);

}

}

**StudentNotFoundException.java**

package exceptions;

public class StudentNotFoundException extends RuntimeException {

public StudentNotFoundException(String message) {

super(message);

}

}

**TeacherNotFoundException.java**

package exceptions;

public class TeacherNotFoundException extends RuntimeException {

public TeacherNotFoundException(String message) {

super(message);

}

}

**Task 7: Database Connectivity**

**1) Database Initialization:**

**DBPropertyUtil.java**

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class DBPropertyUtil {

public static String getConnectionString(String propertyFileName) {

Properties props = new Properties();

try (FileInputStream fis = new FileInputStream(propertyFileName)) {

props.load(fis);

String host = props.getProperty("host");

String port = props.getProperty("port");

String dbname = props.getProperty("dbname");

String username = props.getProperty("username");

String password = props.getProperty("password");

// Construct JDBC MySQL URL

return "jdbc:mysql://" + host + ":" + port + "/" + dbname +

"?user=" + username + "&password=" + password;

} catch (IOException e) {

System.***out***.println("Error reading properties file: " + e.getMessage());

}

return null;

}

}

**DBConnUtil.java**

package util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnUtil {

public static Connection getConnection(String connectionString) {

Connection conn = null;

try {

Class.*forName*("com.mysql.cj.jdbc.Driver"); // Load MySQL driver

conn = DriverManager.*getConnection*(connectionString);

System.***out***.println("Database connection established.");

} catch (ClassNotFoundException e) {

System.***out***.println("MySQL JDBC Driver not found: " + e.getMessage());

} catch (SQLException e) {

System.***out***.println("SQL Exception: " + e.getMessage());

}

return conn;

}

}

**2) Data Retrieval and data insertion and updating:**

**IStudentDao.java**

package dao;

import entity.model.Student;

import exceptions.StudentNotFoundException;

import exceptions.InvalidStudentDataException;

import java.util.List;

public interface IStudentDao {

void insertStudent(Student student) throws InvalidStudentDataException;

Student getStudentById(int id) throws StudentNotFoundException;

List<Student> getAllStudents();

void updateStudent(Student student) throws StudentNotFoundException, InvalidStudentDataException;

}

**StudentDaoImpl.java**

package dao;

import entity.model.Student;

import exceptions.InvalidStudentDataException;

import exceptions.StudentNotFoundException;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class StudentDaoImpl implements IStudentDao {

private Connection getConnection() {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

*@Override*

public void insertStudent(Student student) throws InvalidStudentDataException {

if (student.getFirstName() == null || student.getEmail() == null || student.getDateOfBirth() == null) {

throw new InvalidStudentDataException("Student data is incomplete.");

}

String sql = "INSERT INTO students (first\_name, last\_name, date\_of\_birth, email, phone\_number) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)) {

stmt.setString(1, student.getFirstName());

stmt.setString(2, student.getLastName());

stmt.setDate(3, new java.sql.Date(student.getDateOfBirth().getTime()));

stmt.setString(4, student.getEmail());

stmt.setString(5, student.getPhoneNumber());

stmt.executeUpdate();

ResultSet rs = stmt.getGeneratedKeys();

if (rs.next()) {

student.setStudentID(rs.getInt(1));

}

System.***out***.println("Student inserted successfully.");

} catch (SQLException e) {

throw new InvalidStudentDataException("❌ Error inserting student: " + e.getMessage());

}

}

*@Override*

public Student getStudentById(int id) throws StudentNotFoundException {

String sql = "SELECT \* FROM students WHERE student\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Student(

rs.getInt("student\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

} else {

throw new StudentNotFoundException("Student with ID " + id + " not found.");

}

} catch (SQLException e) {

throw new StudentNotFoundException("Error fetching student: " + e.getMessage());

}

}

*@Override*

public List<Student> getAllStudents() {

List<Student> students = new ArrayList<>();

String sql = "SELECT \* FROM students";

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

students.add(new Student(

rs.getInt("student\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

));

}

} catch (SQLException e) {

System.***out***.println("Error retrieving students: " + e.getMessage());

}

return students;

}

*@Override*

public void updateStudent(Student student) throws StudentNotFoundException, InvalidStudentDataException {

if (student.getStudentID() <= 0 || student.getEmail() == null) {

throw new InvalidStudentDataException("Invalid or missing student data.");

}

String sql = "UPDATE students SET first\_name=?, last\_name=?, date\_of\_birth=?, email=?, phone\_number=? WHERE student\_id=?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, student.getFirstName());

stmt.setString(2, student.getLastName());

stmt.setDate(3, new java.sql.Date(student.getDateOfBirth().getTime()));

stmt.setString(4, student.getEmail());

stmt.setString(5, student.getPhoneNumber());

stmt.setInt(6, student.getStudentID());

int rows = stmt.executeUpdate();

if (rows == 0) throw new StudentNotFoundException("Student not found for update.");

System.***out***.println("Student updated successfully.");

} catch (SQLException e) {

throw new InvalidStudentDataException("❌ Error updating student: " + e.getMessage());

}

}

}

**ITeacherDao.java**

package dao;

import entity.model.Teacher;

import exceptions.InvalidTeacherDataException;

import exceptions.TeacherNotFoundException;

import java.util.List;

public interface ITeacherDao {

void insertTeacher(Teacher teacher) throws InvalidTeacherDataException;

Teacher getTeacherById(int id) throws TeacherNotFoundException;

List<Teacher> getAllTeachers();

void updateTeacher(Teacher teacher) throws InvalidTeacherDataException, TeacherNotFoundException;

}

**TeacherDaoImpl.java**package dao;

import entity.model.Teacher;

import exceptions.InvalidTeacherDataException;

import exceptions.TeacherNotFoundException;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class TeacherDaoImpl implements ITeacherDao {

private Connection getConnection() {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

*@Override*

public void insertTeacher(Teacher teacher) throws InvalidTeacherDataException {

if (teacher.getFirstName() == null || teacher.getEmail() == null) {

throw new InvalidTeacherDataException("Teacher data is incomplete.");

}

String sql = "INSERT INTO teachers (first\_name, last\_name, email) VALUES (?, ?, ?)";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)) {

stmt.setString(1, teacher.getFirstName());

stmt.setString(2, teacher.getLastName());

stmt.setString(3, teacher.getEmail());

stmt.executeUpdate();

ResultSet rs = stmt.getGeneratedKeys();

if (rs.next()) {

teacher.setTeacherID(rs.getInt(1));

}

System.***out***.println("Teacher inserted successfully.");

} catch (SQLException e) {

throw new InvalidTeacherDataException("❌ Error inserting teacher: " + e.getMessage());

}

}

*@Override*

public Teacher getTeacherById(int id) throws TeacherNotFoundException {

String sql = "SELECT \* FROM teachers WHERE teacher\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Teacher(

rs.getInt("teacher\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email")

);

} else {

throw new TeacherNotFoundException("Teacher with ID " + id + " not found.");

}

} catch (SQLException e) {

throw new TeacherNotFoundException("❌ Error fetching teacher: " + e.getMessage());

}

}

*@Override*

public List<Teacher> getAllTeachers() {

List<Teacher> teachers = new ArrayList<>();

String sql = "SELECT \* FROM teachers";

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

teachers.add(new Teacher(

rs.getInt("teacher\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email")

));

}

} catch (SQLException e) {

System.***out***.println("Error retrieving teachers: " + e.getMessage());

}

return teachers;

}

*@Override*

public void updateTeacher(Teacher teacher) throws InvalidTeacherDataException, TeacherNotFoundException {

if (teacher.getTeacherID() <= 0 || teacher.getEmail() == null) {

throw new InvalidTeacherDataException("Teacher data is invalid or incomplete.");

}

String sql = "UPDATE teachers SET first\_name=?, last\_name=?, email=? WHERE teacher\_id=?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, teacher.getFirstName());

stmt.setString(2, teacher.getLastName());

stmt.setString(3, teacher.getEmail());

stmt.setInt(4, teacher.getTeacherID());

int rows = stmt.executeUpdate();

if (rows == 0) throw new TeacherNotFoundException("Teacher not found for update.");

System.***out***.println("Teacher updated successfully.");

} catch (SQLException e) {

throw new InvalidTeacherDataException("❌ Error updating teacher: " + e.getMessage());

}

}

}

**ICourseDao.java**

package dao;

import entity.model.Course;

import exceptions.CourseNotFoundException;

import exceptions.InvalidCourseDataException;

import java.util.List;

public interface ICourseDao {

void insertCourse(Course course) throws InvalidCourseDataException;

Course getCourseById(int id) throws CourseNotFoundException;

List<Course> getAllCourses();

void updateCourse(Course course) throws CourseNotFoundException, InvalidCourseDataException;

}

**CourseDaoImpl.java**

package dao;

import entity.model.Course;

import entity.model.Teacher;

import util.DBConnUtil;

import util.DBPropertyUtil;

import exceptions.\*;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class CourseDaoImpl implements ICourseDao {

private Connection getConnection() {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

*@Override*

public void insertCourse(Course course) throws InvalidCourseDataException {

if (course.getCourseName() == null || course.getCourseName().isBlank()

|| course.getCourseCode() == null || course.getCourseCode().isBlank()

|| course.getCredits() <= 0) {

throw new InvalidCourseDataException("Course data is incomplete or invalid.");

}

String sql = "INSERT INTO courses (course\_name, course\_code, credits, teacher\_id) VALUES (?, ?, ?, ?)";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)) {

stmt.setString(1, course.getCourseName());

stmt.setString(2, course.getCourseCode());

stmt.setInt(3, course.getCredits());

if (course.getInstructorName() != null) {

stmt.setInt(4, course.getInstructorName().getTeacherID());

} else {

stmt.setNull(4, Types.***INTEGER***);

}

stmt.executeUpdate();

ResultSet keys = stmt.getGeneratedKeys();

if (keys.next()) {

course.setCourseID(keys.getInt(1));

}

if (course.getInstructorName() != null) {

course.getInstructorName().assignCourse(course);

}

System.***out***.println("Course inserted successfully.");

} catch (SQLException e) {

throw new InvalidCourseDataException("❌ Error inserting course: " + e.getMessage());

}

}

*@Override*

public Course getCourseById(int id) throws CourseNotFoundException {

String sql = "SELECT \* FROM courses c LEFT JOIN teachers t ON c.teacher\_id = t.teacher\_id WHERE c.course\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

Teacher teacher = null;

if (rs.getInt("teacher\_id") != 0) {

teacher = new Teacher(

rs.getInt("teacher\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email")

);

}

Course course = new Course(

rs.getInt("course\_id"),

rs.getString("course\_name"),

rs.getInt("credits"),

rs.getString("course\_code"),

teacher

);

if (teacher != null) {

teacher.assignCourse(course);

}

return course;

} else {

throw new CourseNotFoundException("Course with ID " + id + " not found.");

}

} catch (SQLException e) {

throw new CourseNotFoundException("Database error while retrieving course: " + e.getMessage());

}

}

*@Override*

public List<Course> getAllCourses() {

List<Course> courses = new ArrayList<>();

String sql = "SELECT \* FROM courses c LEFT JOIN teachers t ON c.teacher\_id = t.teacher\_id";

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Teacher teacher = null;

if (rs.getInt("teacher\_id") != 0) {

teacher = new Teacher(

rs.getInt("teacher\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email")

);

}

Course course = new Course(

rs.getInt("course\_id"),

rs.getString("course\_name"),

rs.getInt("credits"),

rs.getString("course\_code"),

teacher

);

if (teacher != null) {

teacher.assignCourse(course);

}

courses.add(course);

}

} catch (SQLException e) {

System.***out***.println("Error retrieving courses: " + e.getMessage());

}

return courses;

}

*@Override*

public void updateCourse(Course course) throws CourseNotFoundException, InvalidCourseDataException {

if (course.getCourseName() == null || course.getCourseCode() == null || course.getCredits() <= 0) {

throw new InvalidCourseDataException("Course data is incomplete or invalid.");

}

String sql = "UPDATE courses SET course\_name = ?, course\_code = ?, credits = ?, teacher\_id = ? WHERE course\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, course.getCourseName());

stmt.setString(2, course.getCourseCode());

stmt.setInt(3, course.getCredits());

if (course.getInstructorName() != null) {

stmt.setInt(4, course.getInstructorName().getTeacherID());

} else {

stmt.setNull(4, Types.***INTEGER***);

}

stmt.setInt(5, course.getCourseID());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated == 0) {

throw new CourseNotFoundException("Course with ID " + course.getCourseID() + " not found.");

}

if (course.getInstructorName() != null) {

course.getInstructorName().assignCourse(course);

}

System.***out***.println("Course updated successfully.");

} catch (SQLException e) {

throw new InvalidCourseDataException("❌ Error updating course: " + e.getMessage());

}

}

}

**IEnrollmentDao.java**

package dao;

import entity.model.Enrollment;

import exceptions.DuplicateEnrollmentException;

import exceptions.InvalidEnrollmentDataException;

import java.util.List;

public interface IEnrollmentDao {

void insertEnrollment(Enrollment enrollment) throws DuplicateEnrollmentException, InvalidEnrollmentDataException;

Enrollment getEnrollmentById(int id);

List<Enrollment> getAllEnrollments();

void updateEnrollment(Enrollment enrollment) throws InvalidEnrollmentDataException;

}

**EnrollmentDaoImpl.java**package dao;

import entity.model.Course;

import entity.model.Enrollment;

import entity.model.Student;

import entity.model.Teacher;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import exceptions.\*;

public class EnrollmentDaoImpl implements IEnrollmentDao {

private Connection getConnection() {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

*@Override*

public void insertEnrollment(Enrollment enrollment) throws DuplicateEnrollmentException, InvalidEnrollmentDataException {

if (enrollment.getStudent() == null || enrollment.getCourse() == null || enrollment.getEnrollmentDate() == null) {

throw new InvalidEnrollmentDataException("Enrollment data is incomplete.");

}

// Check if student is already enrolled

for (Enrollment existing : getAllEnrollments()) {

if (existing.getStudent().getStudentID() == enrollment.getStudent().getStudentID() &&

existing.getCourse().getCourseID() == enrollment.getCourse().getCourseID()) {

throw new DuplicateEnrollmentException("Student already enrolled in this course.");

}

}

String sql = "INSERT INTO enrollments (student\_id, course\_id, enrollment\_date) VALUES (?, ?, ?)";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)) {

stmt.setInt(1, enrollment.getStudent().getStudentID());

stmt.setInt(2, enrollment.getCourse().getCourseID());

stmt.setDate(3, new java.sql.Date(enrollment.getEnrollmentDate().getTime()));

stmt.executeUpdate();

ResultSet keys = stmt.getGeneratedKeys();

if (keys.next()) {

enrollment.setEnrollmentID(keys.getInt(1));

}

System.***out***.println("Enrollment inserted successfully.");

} catch (SQLException e) {

throw new InvalidEnrollmentDataException("Error inserting enrollment: " + e.getMessage());

}

}

*@Override*

public Enrollment getEnrollmentById(int id) {

String sql = """

SELECT e.\*,

s.first\_name AS s\_fname, s.last\_name AS s\_lname, s.date\_of\_birth, s.email AS s\_email, s.phone\_number,

c.course\_id, c.course\_name, c.course\_code, c.credits,

t.teacher\_id, t.first\_name AS t\_fname, t.last\_name AS t\_lname, t.email AS t\_email

FROM enrollments e

JOIN students s ON e.student\_id = s.student\_id

JOIN courses c ON e.course\_id = c.course\_id

LEFT JOIN teachers t ON c.teacher\_id = t.teacher\_id

WHERE e.enrollment\_id = ?

""";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

Student student = new Student(

rs.getInt("student\_id"),

rs.getString("s\_fname"),

rs.getString("s\_lname"),

rs.getDate("date\_of\_birth"),

rs.getString("s\_email"),

rs.getString("phone\_number")

);

Teacher teacher = new Teacher(

rs.getInt("teacher\_id"),

rs.getString("t\_fname"),

rs.getString("t\_lname"),

rs.getString("t\_email")

);

Course course = new Course(

rs.getInt("course\_id"),

rs.getString("course\_name"),

rs.getInt("credits"),

rs.getString("course\_code"),

teacher

);

return new Enrollment(

rs.getInt("enrollment\_id"),

student,

course,

rs.getDate("enrollment\_date")

);

}

} catch (SQLException e) {

System.***out***.println("Error fetching enrollment: " + e.getMessage());

}

return null;

}

*@Override*

public List<Enrollment> getAllEnrollments() {

List<Enrollment> enrollments = new ArrayList<>();

String sql = """

SELECT e.\*,

s.first\_name AS s\_fname, s.last\_name AS s\_lname, s.date\_of\_birth, s.email AS s\_email, s.phone\_number,

c.course\_id, c.course\_name, c.course\_code, c.credits,

t.teacher\_id, t.first\_name AS t\_fname, t.last\_name AS t\_lname, t.email AS t\_email

FROM enrollments e

JOIN students s ON e.student\_id = s.student\_id

JOIN courses c ON e.course\_id = c.course\_id

LEFT JOIN teachers t ON c.teacher\_id = t.teacher\_id

""";

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Student student = new Student(

rs.getInt("student\_id"),

rs.getString("s\_fname"),

rs.getString("s\_lname"),

rs.getDate("date\_of\_birth"),

rs.getString("s\_email"),

rs.getString("phone\_number")

);

Teacher teacher = new Teacher(

rs.getInt("teacher\_id"),

rs.getString("t\_fname"),

rs.getString("t\_lname"),

rs.getString("t\_email")

);

Course course = new Course(

rs.getInt("course\_id"),

rs.getString("course\_name"),

rs.getInt("credits"),

rs.getString("course\_code"),

teacher

);

Enrollment enrollment = new Enrollment(

rs.getInt("enrollment\_id"),

student,

course,

rs.getDate("enrollment\_date")

);

enrollments.add(enrollment);

}

} catch (SQLException e) {

System.***out***.println("Error retrieving enrollments: " + e.getMessage());

}

return enrollments;

}

*@Override*

public void updateEnrollment(Enrollment enrollment) throws InvalidEnrollmentDataException {

if (enrollment.getStudent() == null || enrollment.getCourse() == null || enrollment.getEnrollmentDate() == null) {

throw new InvalidEnrollmentDataException("Enrollment data is incomplete.");

}

String sql = "UPDATE enrollments SET student\_id = ?, course\_id = ?, enrollment\_date = ? WHERE enrollment\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, enrollment.getStudent().getStudentID());

stmt.setInt(2, enrollment.getCourse().getCourseID());

stmt.setDate(3, new java.sql.Date(enrollment.getEnrollmentDate().getTime()));

stmt.setInt(4, enrollment.getEnrollmentID());

stmt.executeUpdate();

System.***out***.println("Enrollment updated successfully.");

} catch (SQLException e) {

throw new InvalidEnrollmentDataException("Error updating enrollment: " + e.getMessage());

}

}

}

**IPaymentDao.java**

package dao;

import entity.model.Payment;

import exceptions.PaymentValidationException;

import java.util.List;

public interface IPaymentDao {

void insertPayment(Payment payment) throws PaymentValidationException;

Payment getPaymentById(int id);

List<Payment> getAllPayments();

void updatePayment(Payment payment) throws PaymentValidationException;

}

**PaymentDaoImpl.java**

package dao;

import entity.model.Payment;

import entity.model.Student;

import exceptions.PaymentValidationException;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class PaymentDaoImpl implements IPaymentDao {

private Connection getConnection() {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

*@Override*

public void insertPayment(Payment payment) throws PaymentValidationException {

if (payment.getStudent() == null || payment.getAmount() <= 0 || payment.getDate() == null) {

if(payment.getStudent() == null) {

System.***out***.println("1");

}

if(payment.getAmount() <= 0) {

System.***out***.println("2");

}

if(payment.getDate() == null) {

System.***out***.println("3");

}

throw new PaymentValidationException("Invalid payment data.");

}

String sql = "INSERT INTO payments (student\_id, amount, payment\_date) VALUES (?, ?, ?)";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)) {

stmt.setInt(1, payment.getStudent().getStudentID());

stmt.setDouble(2, payment.getAmount());

stmt.setDate(3, new java.sql.Date(payment.getDate().getTime()));

stmt.executeUpdate();

ResultSet rs = stmt.getGeneratedKeys();

if (rs.next()) {

payment.setPaymentID(rs.getInt(1));

}

System.***out***.println("Payment inserted successfully.");

} catch (SQLException e) {

throw new PaymentValidationException("❌ Error inserting payment: " + e.getMessage());

}

}

*@Override*

public Payment getPaymentById(int id) {

String sql = "SELECT \* FROM payments p JOIN students s ON p.student\_id = s.student\_id WHERE p.payment\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

Student student = new Student(

rs.getInt("student\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

return new Payment(

rs.getInt("payment\_id"),

student,

rs.getDouble("amount"),

rs.getDate("payment\_date")

);

}

} catch (SQLException e) {

System.***out***.println("Error fetching payment: " + e.getMessage());

}

return null;

}

*@Override*

public List<Payment> getAllPayments() {

List<Payment> payments = new ArrayList<>();

String sql = "SELECT \* FROM payments p JOIN students s ON p.student\_id = s.student\_id";

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Student student = new Student(

rs.getInt("student\_id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

Payment payment = new Payment(

rs.getInt("payment\_id"),

student,

rs.getDouble("amount"),

rs.getDate("payment\_date")

);

payments.add(payment);

}

} catch (SQLException e) {

System.***out***.println("Error retrieving payments: " + e.getMessage());

}

return payments;

}

*@Override*

public void updatePayment(Payment payment) throws PaymentValidationException {

if (payment.getStudent() == null || payment.getAmount() <= 0 || payment.getDate() == null) {

throw new PaymentValidationException("Invalid payment data.");

}

String sql = "UPDATE payments SET student\_id = ?, amount = ?, payment\_date = ? WHERE payment\_id = ?";

try (Connection conn = getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, payment.getStudent().getStudentID());

stmt.setDouble(2, payment.getAmount());

stmt.setDate(3, new java.sql.Date(payment.getDate().getTime()));

stmt.setInt(4, payment.getPaymentID());

stmt.executeUpdate();

System.***out***.println("Payment updated successfully.");

} catch (SQLException e) {

throw new PaymentValidationException("❌ Error updating payment: " + e.getMessage());

}

}

}

**3) Transaction manager:**

Implement methods for handling database transactions when enrolling students, assigning teachers, or recording payments. Transactions should be atomic and maintain data integrity. Use database transactions to ensure that multiple related operations either all succeed or all fail. Implement error handling and rollback mechanisms in case of transaction failures.

package dao;

import entity.model.\*;

import java.util.Date;

import java.util.Scanner;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class TransactionManager {

private Connection getConnection() throws SQLException {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

public boolean enrollStudent(Enrollment enrollment) {

String sql = "INSERT INTO enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date) VALUES (?, ?, ?, ?)";

try (Connection conn = getConnection()) {

conn.setAutoCommit(false);

try (PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, enrollment.getEnrollmentID());

stmt.setInt(2, enrollment.getStudent().getStudentID());

stmt.setInt(3, enrollment.getCourse().getCourseID());

stmt.setDate(4, new java.sql.Date(enrollment.getEnrollmentDate().getTime()));

stmt.executeUpdate();

conn.commit();

System.***out***.println("✅ Enrollment successful and committed.");

return true;

} catch (SQLException e) {

conn.rollback();

System.***out***.println("❌ Enrollment failed. Rolled back. " + e.getMessage());

}

} catch (SQLException e) {

System.***out***.println("DB connection error: " + e.getMessage());

}

return false;

}

public boolean assignTeacherToCourse(Course course) {

String sql = "UPDATE courses SET teacher\_id = ? WHERE course\_id = ?";

try (Connection conn = getConnection()) {

conn.setAutoCommit(false);

try (PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, course.getInstructorName().getTeacherID());

stmt.setInt(2, course.getCourseID());

stmt.executeUpdate();

conn.commit();

System.***out***.println("✅ Teacher assigned and committed.");

return true;

} catch (SQLException e) {

conn.rollback();

System.***out***.println("❌ Teacher assignment failed. Rolled back. " + e.getMessage());

}

} catch (SQLException e) {

System.***out***.println("DB connection error: " + e.getMessage());

}

return false;

}

public boolean recordPayment(Payment payment) {

String sql = "INSERT INTO payments (payment\_id, student\_id, amount, payment\_date) VALUES (?, ?, ?, ?)";

try (Connection conn = getConnection()) {

conn.setAutoCommit(false);

try (PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, payment.getPaymentID());

stmt.setInt(2, payment.getStudent().getStudentID());

stmt.setDouble(3, payment.getAmount());

stmt.setDate(4, new java.sql.Date(payment.getDate().getTime()));

stmt.executeUpdate();

conn.commit();

System.***out***.println("✅ Payment recorded and committed.");

return true;

} catch (SQLException e) {

conn.rollback();

System.***out***.println("❌ Payment failed. Rolled back. " + e.getMessage());

}

} catch (SQLException e) {

System.***out***.println("DB connection error: " + e.getMessage());

}

return false;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

TransactionManager manager = new TransactionManager();

// Student input

System.***out***.print("Enter Student ID: ");

int studentId = scanner.nextInt();

Student student = new Student();

student.setStudentID(studentId);

// Course input

System.***out***.print("Enter Course ID: ");

int courseId = scanner.nextInt();

Course course = new Course();

course.setCourseID(courseId);

// Teacher input

System.***out***.print("Enter Teacher ID to assign to Course: ");

int teacherId = scanner.nextInt();

Teacher teacher = new Teacher();

teacher.setTeacherID(teacherId);

course.setInstructorName(teacher);

// Enrollment input

System.***out***.print("Enter Enrollment ID: ");

int enrollmentId = scanner.nextInt();

Enrollment enrollment = new Enrollment();

enrollment.setEnrollmentID(enrollmentId);

enrollment.setStudent(student);

enrollment.setCourse(course);

enrollment.setEnrollmentDate(new Date());

System.***out***.println("\n🔄 Testing enrollStudent...");

manager.enrollStudent(enrollment);

// Assign teacher to course

System.***out***.println("\n🔄 Testing assignTeacherToCourse...");

manager.assignTeacherToCourse(course);

// Payment input

System.***out***.print("\nEnter Payment ID: ");

int paymentId = scanner.nextInt();

System.***out***.print("Enter Payment Amount: ");

double amount = scanner.nextDouble();

Payment payment = new Payment();

payment.setPaymentID(paymentId);

payment.setStudent(student);

payment.setAmount(amount);

payment.setDate(new Date());

System.***out***.println("\n🔄 Testing recordPayment...");

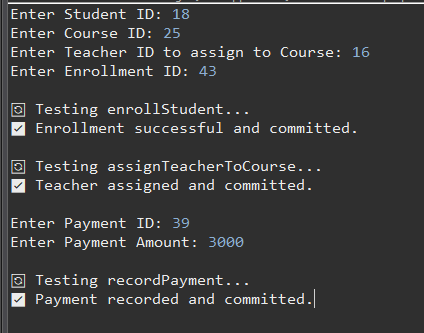
manager.recordPayment(payment);

scanner.close();

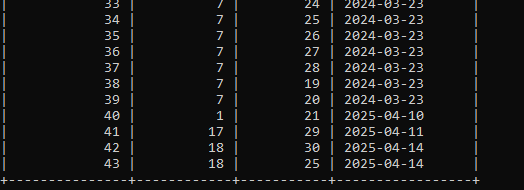
}

}

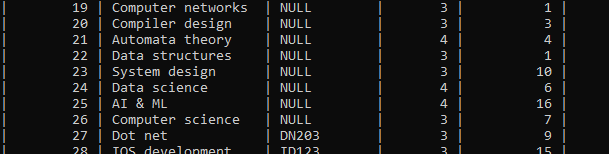
**Output (Transaction manager) :**

****

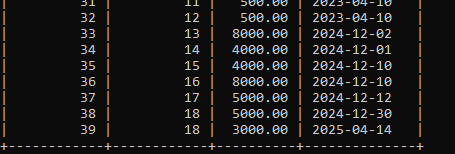
**Enrollment is successfully done:**

****

**Course is successfully assigned to a teacher:**

****

**Successfully made a payment:**

****

**4) Dynamic query builder:**

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to retrieve specific data from the database. Users should be able to specify columns, conditions, and sorting criteria. Create a query builder method that dynamically generates SQL queries based on user input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

package dao;

import java.util.Scanner;

import util.DBConnUtil;

import util.DBPropertyUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class DynamicQueryBuilder {

private Connection getConnection() throws SQLException {

String connStr = DBPropertyUtil.*getConnectionString*("db.properties");

return DBConnUtil.*getConnection*(connStr);

}

public void runDynamicQuery(String table, List<String> columns, String condition, String orderBy) {

StringBuilder query = new StringBuilder("SELECT ");

if (columns == null || columns.isEmpty()) {

query.append("\*");

} else {

query.append(String.*join*(", ", columns));

}

query.append(" FROM ").append(table);

if (condition != null && !condition.trim().isEmpty()) {

query.append(" WHERE ").append(condition);

}

if (orderBy != null && !orderBy.trim().isEmpty()) {

query.append(" ORDER BY ").append(orderBy);

}

System.***out***.println("Executing: " + query);

try (Connection conn = getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query.toString())) {

ResultSetMetaData metaData = rs.getMetaData();

int colCount = metaData.getColumnCount();

System.***out***.println("\n📋 Query Results:");

while (rs.next()) {

for (int i = 1; i <= colCount; i++) {

System.***out***.print(metaData.getColumnName(i) + ": " + rs.getString(i) + " | ");

}

System.***out***.println();

}

} catch (SQLException e) {

System.***out***.println("Error running dynamic query: " + e.getMessage());

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

DynamicQueryBuilder queryBuilder = new DynamicQueryBuilder();

System.***out***.println("🔍 Dynamic SQL Query Builder");

// Table name

System.***out***.print("Enter table name: ");

String table = scanner.nextLine();

// Columns input

System.***out***.print("Enter columns to select (comma-separated or leave empty for all): ");

String columnInput = scanner.nextLine();

List<String> columns;

if (columnInput.trim().isEmpty()) {

columns = new ArrayList<>();

} else {

columns = Arrays.*asList*(columnInput.split("\\s\*,\\s\*")); // trims spaces too

}

// Condition

System.***out***.print("Enter WHERE condition (or leave empty): ");

String condition = scanner.nextLine();

// Order By

System.***out***.print("Enter ORDER BY column(s) (or leave empty): ");

String orderBy = scanner.nextLine();

// Execute query

System.***out***.println("\n🛠 Executing dynamic query...\n");

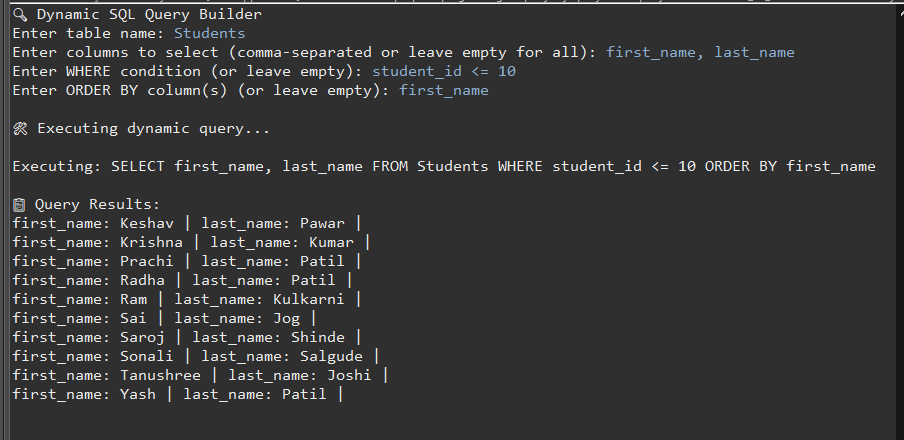
queryBuilder.runDynamicQuery(table, columns, condition, orderBy);

scanner.close();

}

}

**Output (Dynamic query builder) :**

****

**MainModule.java**

package main;

import dao.CourseDaoImpl;

import dao.StudentDaoImpl;

import dao.TeacherDaoImpl;

import entity.model.\*;

import service.EnrollmentService;

import service.PaymentService;

import java.sql.Date;

import java.util.List;

import java.util.Scanner;

public class MainModule {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

StudentDaoImpl studentDao = new StudentDaoImpl();

TeacherDaoImpl teacherDao = new TeacherDaoImpl();

CourseDaoImpl courseDao = new CourseDaoImpl();

EnrollmentService enrollmentService = new EnrollmentService();

PaymentService paymentService = new PaymentService();

System.out.println("Welcome to Student Information System");

int choice;

do {

System.out.println("\n========= MENU =========");

System.out.println("1. Register Student");

System.out.println("2. Register Teacher");

System.out.println("3. Create Course & Assign Teacher");

System.out.println("4. Enroll Student in a Course");

System.out.println("5. Make Payment");

System.out.println("6. View Student Info");

System.out.println("7. View Enrolled Courses");

System.out.println("8. View Payment History");

System.out.println("9. View Teacher Info");

System.out.println("10. Enrollment Report for Course");

System.out.println("0. Exit");

System.out.print("Enter choice: ");

choice = sc.nextInt();

sc.nextLine(); // consume newline

switch (choice) {

case 1 -> {

System.out.println("\nRegister Student:");

System.out.print("First Name: ");

String fname = sc.nextLine();

System.out.print("Last Name: ");

String lname = sc.nextLine();

System.out.print("DOB (yyyy-mm-dd): ");

Date dob = Date.valueOf(sc.nextLine());

System.out.print("Email: ");

String email = sc.nextLine();

System.out.print("Phone: ");

String phone = sc.nextLine();

Student s = new Student();

s.updateStudentInfo(fname, lname, dob, email, phone);

studentDao.insertStudent(s);

}

case 2 -> {

System.out.println("\nRegister Teacher:");

System.out.print("Full Name: ");

String fullName = sc.nextLine();

String[] names = fullName.split(" ");

String tFname = names[0];

String tLname = names.length > 1 ? names[1] : "";

System.out.print("Email: ");

String tEmail = sc.nextLine();

System.out.print("Expertise: ");

String expertise = sc.nextLine();

Teacher t = new Teacher();

t.updateTeacherInfo(tFname + " " + tLname, tEmail, expertise);

teacherDao.insertTeacher(t);

}

case 3 -> {

System.out.println("\nCreate Course:");

System.out.print("Course Name: ");

String courseName = sc.nextLine();

System.out.print("Course Code: ");

String courseCode = sc.nextLine();

System.out.print("Credits: ");

int credits = sc.nextInt();

sc.nextLine();

System.out.print("Assign Existing Teacher ID: ");

int teacherId = sc.nextInt();

sc.nextLine();

Teacher t = teacherDao.getTeacherById(teacherId);

if (t == null) {

System.out.println("Teacher not found.");

break;

}

Course c = new Course();

c.setCourseName(courseName);

c.setCourseCode(courseCode);

c.setCredits(credits);

c.setInstructorName(t);

courseDao.insertCourse(c);

}

case 4 -> {

System.out.println("\nEnrolll Student in a Course");

System.out.print("Student ID: ");

int studentId = sc.nextInt();

sc.nextLine();

System.out.print("Course Name: ");

String cname = sc.nextLine();

Student st = studentDao.getStudentById(studentId);

if (st == null) {

System.out.println("Student not found.");

break;

}

List<Course> allCourses = courseDao.getAllCourses();

for (Course course : allCourses) {

if (course.getCourseName().equalsIgnoreCase(cname)) {

st.enrollInCourse(course);

break;

}

}

}

case 5 -> {

System.out.println("\nMake a Payment:");

System.out.print("Student ID: ");

int sid = sc.nextInt();

System.out.print("Amount: ");

double amt = sc.nextDouble();

sc.nextLine();

System.out.print("Payment Date (yyyy-mm-dd): ");

Date payDate = Date.valueOf(sc.nextLine());

Student stud = studentDao.getStudentById(sid);

if (stud == null) {

System.out.println("Student not found.");

break;

}

stud.makePayment(amt, payDate);

}

case 6 -> {

System.out.print("\nEnter Student ID to view info: ");

int sid = sc.nextInt();

Student s = studentDao.getStudentById(sid);

if (s != null) s.displayStudentInfo();

else System.out.println("Student not found.");

}

case 7 -> {

System.out.print("\nEnter Student ID: ");

int sid = sc.nextInt();

Student s = studentDao.getStudentById(sid);

if (s != null) {

List<Course> enrolled = s.getEnrolledCourses();

for (Course c : enrolled) {

c.displayCourseInfo();

System.out.println("-------------------");

}

} else System.out.println("Student not found.");

}

case 8 -> {

System.out.print("\nEnter Student ID: ");

int sid = sc.nextInt();

Student s = studentDao.getStudentById(sid);

if (s != null) {

List<Payment> p = s.getPaymentHistory();

for (Payment pay : p) {

System.out.println("Paid ₹" + pay.getPaymentAmount() + " on " + pay.getPaymentDate());

}

} else System.out.println("Student not found.");

}

case 9 -> {

System.out.print("\nEnter Teacher ID: ");

int tid = sc.nextInt();

Teacher t = teacherDao.getTeacherById(tid);

if (t != null) t.displayTeacherInfo();

else System.out.println("Teacher not found.");

}

case 10 -> {

System.out.print("\nEnter Course Name for Report: ");

String cname = sc.nextLine();

enrollmentService.generateEnrollmentReport(cname);

}

case 0 -> System.out.println("\nExiting...");

default -> System.out.println("Invalid option. Try again.");

}

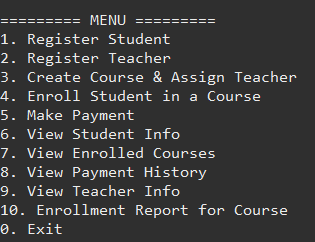
} while (choice != 0);

sc.close();

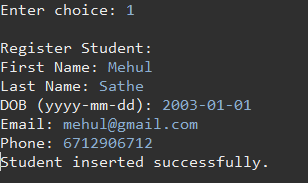
}

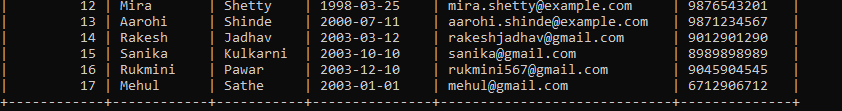
}

**Output:**

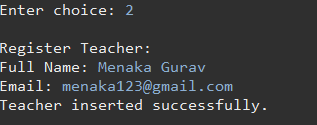
****

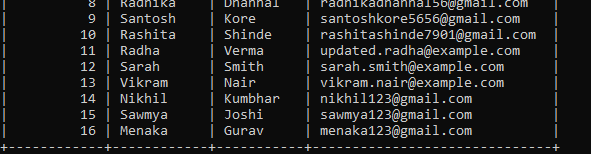
**Choice 1: Register student**

****

****

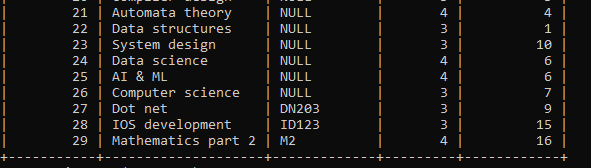
**Choice 2: Register teacher**

****

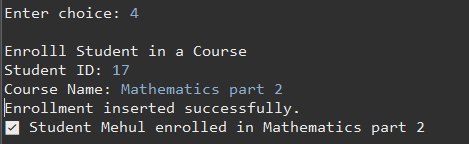
****

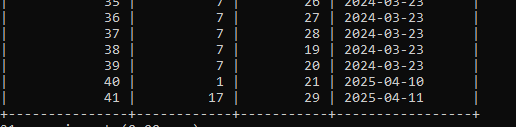
**Choice 3: Create course and assign teacher**

****

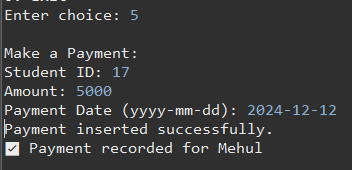
****

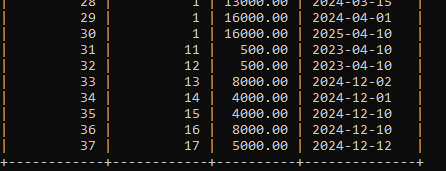
**Choice 4: Enroll student in a course**

****

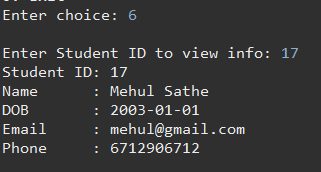
****

**Choice 5: Make payment**

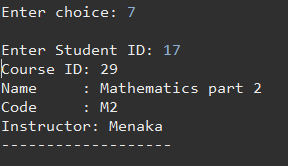
****

****

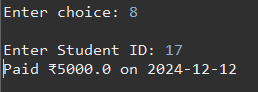
**Choice 6: View student info**

****

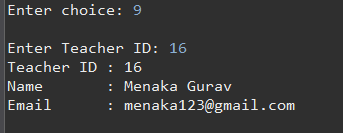
**Choice 7: View enrolled courses**

****

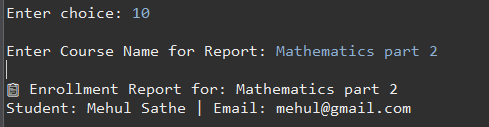
**Choice 8: View payment history**

****

**Choice 9: View teacher info**

****

**Choice 10: Enrollment report for course**

****