AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

ADVANCE DATABASE MANAGEMENT SYSTEM

FALL 2024-25

Project Name

Course Registration Management System

Section: B
Group: 03
Submission Date: 03/ 02/ 2025

Supervised By

JUENA AHMED NOSHIN
Assistant Professor, Faculty, Department of Computer Science

Submitted By

ID	Name	Topic Name	Contribution
22-49415-3	S S Zobaer Ahmed	Backend construction, AI development and Integration, Documentation, Query Writing	25%
21-45834-3	Morshed Al-Jaber Bishal	Database Specialist, Design, Query Writing	25%
21-45837-3	SANVIRAJ AYNUL SIAM	Documentation, Design, Query writing	25%
21-45836-3	SHARUP PAUL	Query Writing, Relational Algebra	25%

Table of Contents

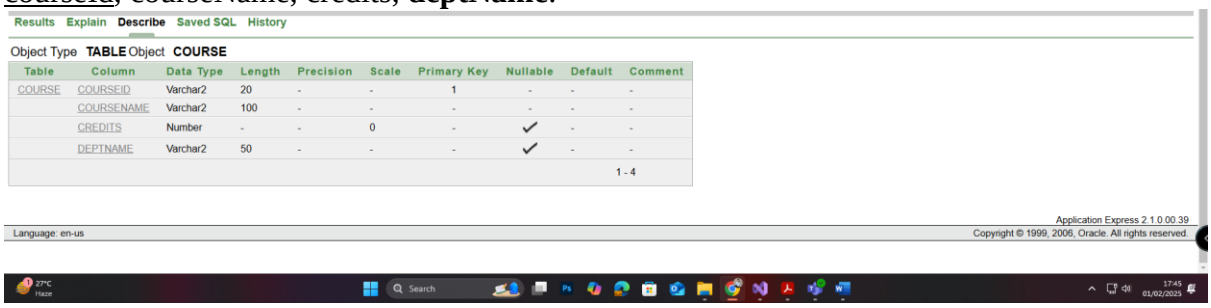
Project Updates

Changes in Tables and ERD

In the course table we remove an attribute or column named semesterOffered and also add a column named DeptName.

<u>Previous Course Table:</u>
<u>courseId</u>, courseName, semesterOffered, credits.

<u>Updated Course Table:</u>
<u>courseId</u>, courseName, credits, **deptName**.



Results Explain Describe Saved SQL History

Object Type TABLE Object COURSE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
COURSE	COURSEID	Varchar2	20	-	-	1	-	-	-
	COURSENAME	Varchar2	100	-	-	-	-	-	-
	CREDITS	Number	-	-	0	-	✓	-	-
	DEPTNAME	Varchar2	50	-	-	-	✓	-	-

1 - 4

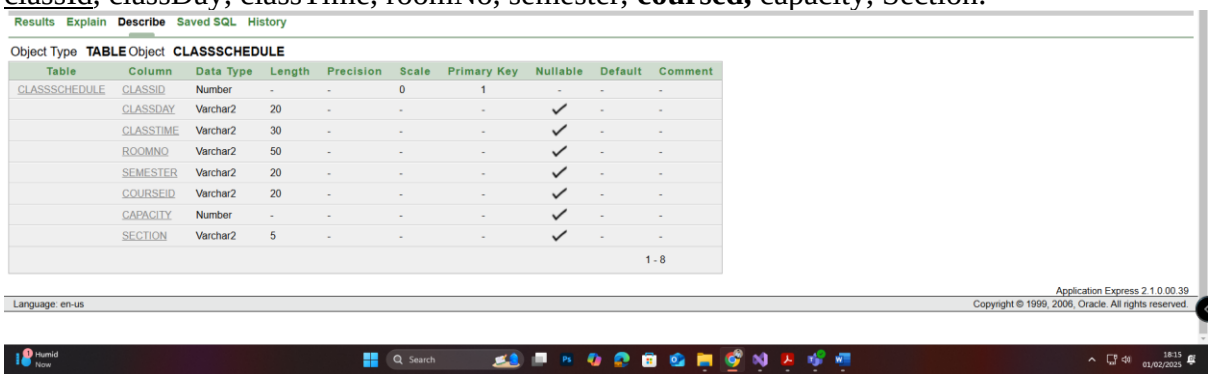
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

In ClassSchedule table we will add two new attributes named Capacity and Section

<u>Previous ClassSchedule Table:</u>
<u>classId</u>, classDay, classTime, roomNo, semester, **courseId**.

<u>Updated ClassSchedule Table:</u>

<u>classId</u>, classDay, classTime, roomNo, semester, **coursed,** capacity, Section.



Results Explain Describe Saved SQL History

Object Type TABLE Object CLASSSCHEDULE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CLASSSCHEDULE	CLASSID	Number	-	-	0	1	-	-	-
	CLASSDAY	Varchar2	20	-	-	-	✓	-	-
	CLASSTIME	Varchar2	30	-	-	-	✓	-	-
	ROOMNO	Varchar2	50	-	-	-	✓	-	-
	SEMESTER	Varchar2	20	-	-	-	✓	-	-
	COURSEID	Varchar2	20	-	-	-	✓	-	-
	CAPACITY	Number	-	-	-	-	✓	-	-
	SECTION	Varchar2	5	-	-	-	✓	-	-

1 - 8

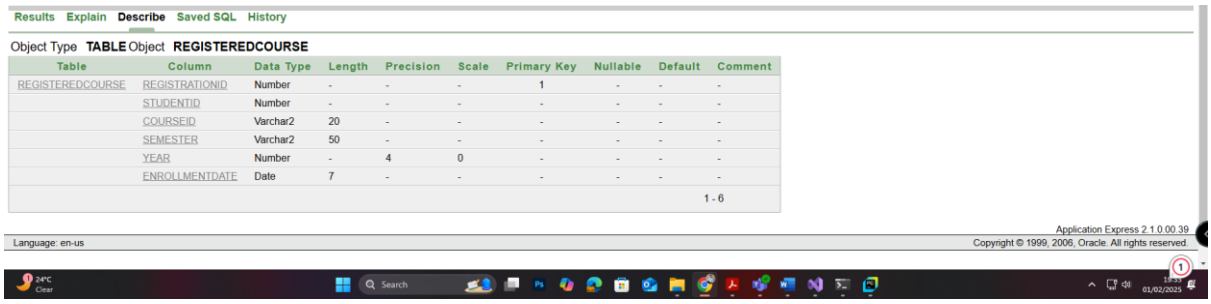
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

We will add two new Entities or Tables

1. **RegisteredCourse:**

```
CREATE TABLE RegisteredCourse (
   registrationId NUMBER PRIMARY KEY,
   studentId NUMBER NOT NULL,
   courseId VARCHAR2(20) NOT NULL,
   semester VARCHAR2(50) NOT NULL,
   year NUMBER(4) NOT NULL,
```

```
   enrollmentDate DATE NOT NULL,
   CONSTRAINT fk_student FOREIGN KEY (studentId) REFERENCES Student(studentId)
ON DELETE CASCADE,
   CONSTRAINT fk_course FOREIGN KEY (courseId) REFERENCES Courses(courseId)
ON DELETE CASCADE
);
```



Results Explain Describe Saved SQL History

Object Type TABLE Object REGISTEREDCOURSE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
REGISTEREDCOURSE	REGISTRATIONID	Number	-	-	-	1	-	-	-
	STUDENTID	Number	-	-	-	-	-	-	-
	COURSEID	Varchar2	20	-	-	-	-	-	-
	SEMESTER	Varchar2	50	-	-	-	-	-	-
	YEAR	Number	-	4	0	-	-	-	-
	ENROLLMENTDATE	Date	7	-	-	-	-	-	-

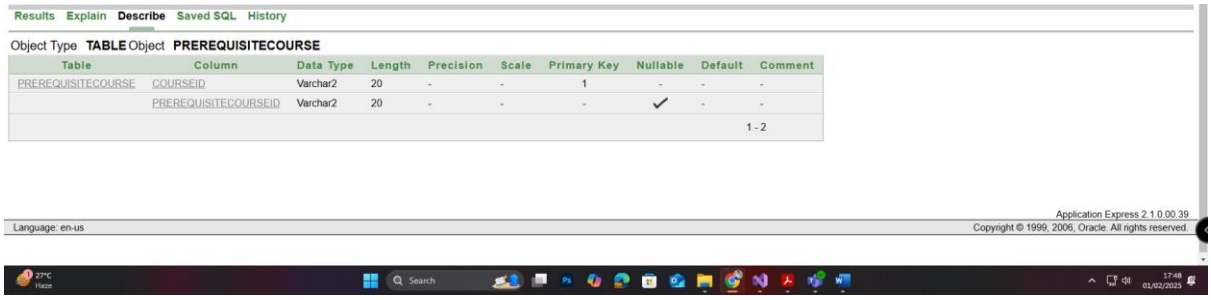
1 - 6

Language: en-us

Application Express 2.1.0.00.39

2. **PreRequisiteCourse:**

```
CREATE TABLE PreRequisiteCourse (
   courseId VARCHAR2(20) NOT NULL,
   preRequisiteCourseID VARCHAR2(20),
   PRIMARY KEY (courseId),
   FOREIGN KEY (courseId) REFERENCES Course(courseId) ON DELETE CASCADE,
   FOREIGN KEY (preRequisiteCourseID) REFERENCES Course(courseId) ON DELETE
CASCADE
);
```



Results Explain Describe Saved SQL History

Object Type TABLE Object PREREQUISITECOURSE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
PREREQUISITECOURSE	COURSEID	Varchar2	20	-	-	1	-	-	-
	PREREQUISITECOURSEID	Varchar2	20	-	-	-	✓	-	-

1 - 2

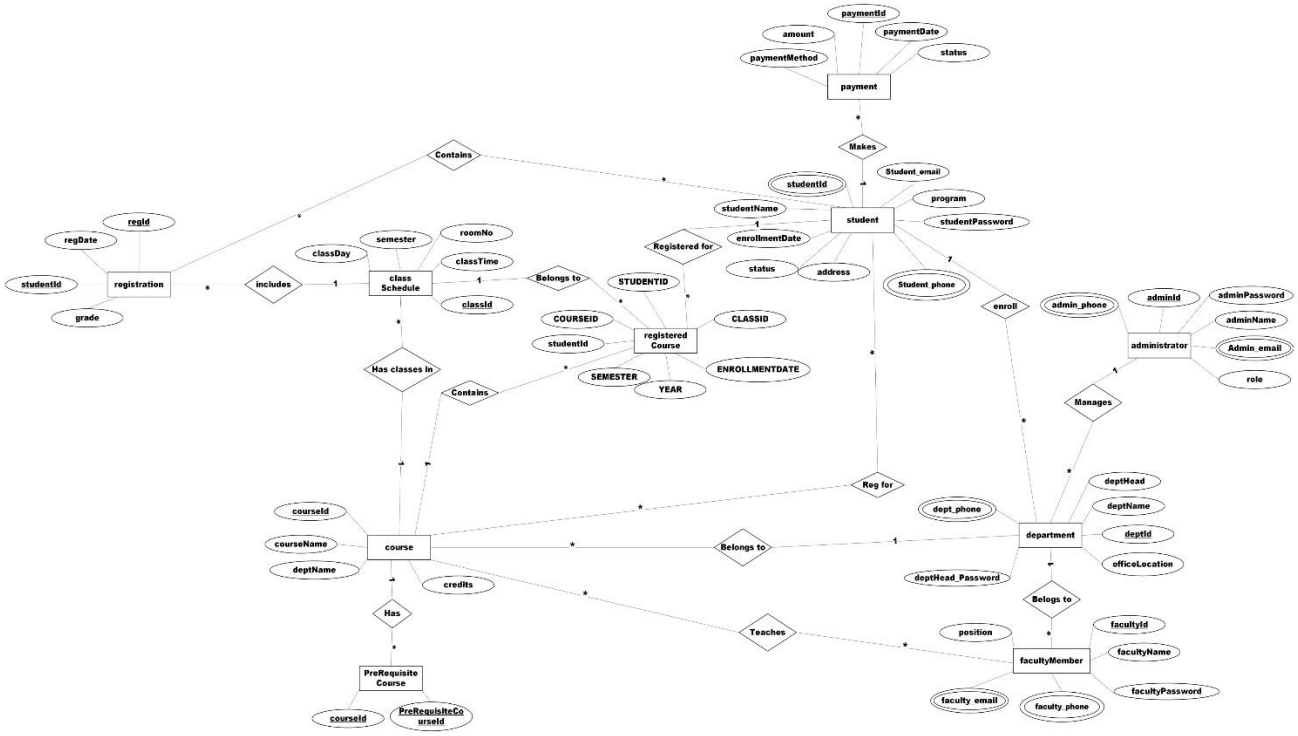
Language: en-us

Application Express 2.1.0.00.39

Update on User Interface

We have made slight improvements to our User Interface! Dive into the **User Interface** section to explore the fresh design and improved user experience.

Updated ERD



Click to view : [ER Diagram CRMS]

Updates to Query Writing

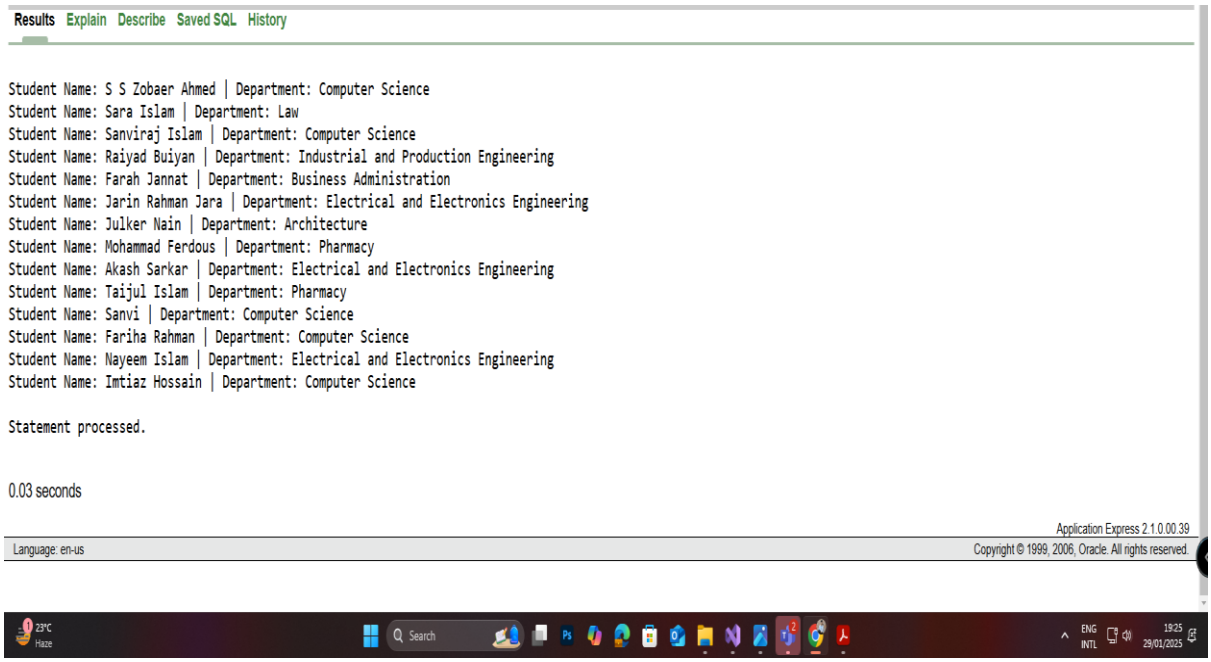
Joining With PL/SQL:

1. Write a query to list students and their department names using PL/SQL

Query:

```
DECLARE
  CURSOR student_cursor IS
    SELECT s.studentName, d.deptName
    FROM Student s
    JOIN Department d ON s.deptId = d.deptId;
  v_studentName Student.studentName%TYPE;
  v_deptName Department.deptName%TYPE;
BEGIN
  OPEN student_cursor;
  LOOP
    FETCH student_cursor INTO v_studentName, v_deptName;
    EXIT WHEN student_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Student Name: ' || v_studentName || ' | Department: ' ||
v_deptName);
  END LOOP;
  CLOSE student_cursor;
```

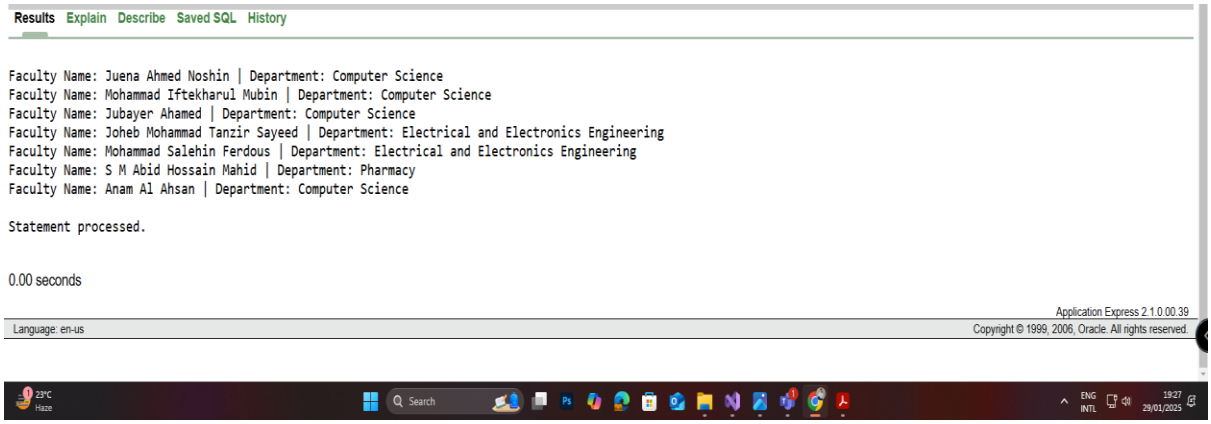
```
END;
/
```



2. Write a query to list faculty members and their respective department names using PL/SQL

Query:

```
DECLARE
  CURSOR faculty_cursor IS
    SELECT f.facultyName, d.deptName
    FROM FacultyMember f
    JOIN Department d ON f.deptId = d.deptId;
  v_facultyName FacultyMember.facultyName%TYPE;
  v_deptName Department.deptName%TYPE;
BEGIN
  OPEN faculty_cursor;
  LOOP
    FETCH faculty_cursor INTO v_facultyName, v_deptName;
    EXIT WHEN faculty_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Faculty Name: ' || v_facultyName || ' | Department: ' ||
v_deptName);
  END LOOP;
  CLOSE faculty_cursor;
END;
/
```



Project Proposal

Background of the problem

Course registration is an important process for students in universities. It helps them enroll in the right courses so they can complete their degrees on time. However, many students face difficulties during registration, which can lead to **missing out on courses** and **delays in graduation**.

Main Problems Students Face:

1. **Limited Seats in Courses:** Some students can't get into the courses they need because seats fill up too fast.
2. **No Immediate Help:** Academic advisors are often too busy to help every student during registration.
3. **Confusion About Course Selection:** Many students, especially first-year students, struggle to choose the right courses.
4. **Time Conflicts:** Sometimes, two or more chosen courses have overlapping class times.
5. **Slow Decision-Making:** If students can't decide quickly, they may lose their chance to enroll in available courses.
6. **System Issues:** The registration website may slow down or crash when too many students use it at the same time.

The Biggest Problem:
Some students **fail to register for any courses** due to these issues, which affects their academic progress and causes stress

Solution to the problem

Proposed Solution: Virtual Advisor Chatbot using AI

To solve this issue, we plan to add a **Virtual Advisor Chatbot** powered by **Artificial**

Intelligence (AI). This chatbot will use **Natural Language Processing (NLP)** with the **OpenAI API** to help students register for courses smoothly.

How the Virtual Advisor Chatbot Will Help:

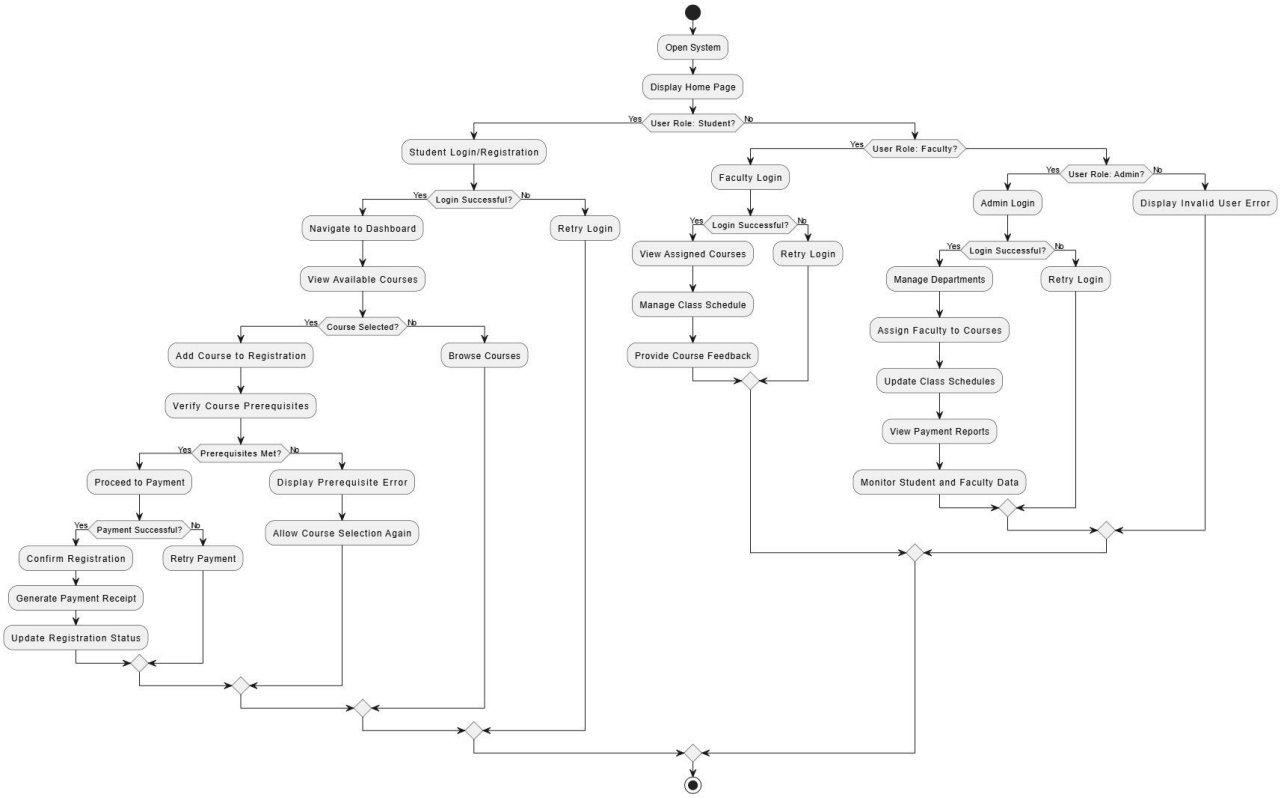
1. **Suggesting Courses:** It will recommend the best courses based on a student's academic progress and available options.
2. **Finding Alternatives:** If a course is full, the chatbot will suggest similar courses.
3. **Checking Requirements:** It will make sure students meet the prerequisites for a course before they enroll.
4. **Avoiding Time Conflicts:** It will help students choose courses that don't overlap in schedule.
5. **Available Anytime:** Unlike human advisors, the chatbot will be available **24/7** to assist students.
6. **Notifying About Seat Availability:** If a course is full, it will alert students when a seat opens up.
7. **Answering Questions:** The chatbot will provide information about course rules, credit limits, and registration deadlines.
8. **Helping Universities Plan Better:** The AI can predict which courses will be in high demand so universities can **offer more seats** in those courses.

Benefits of This Solution:

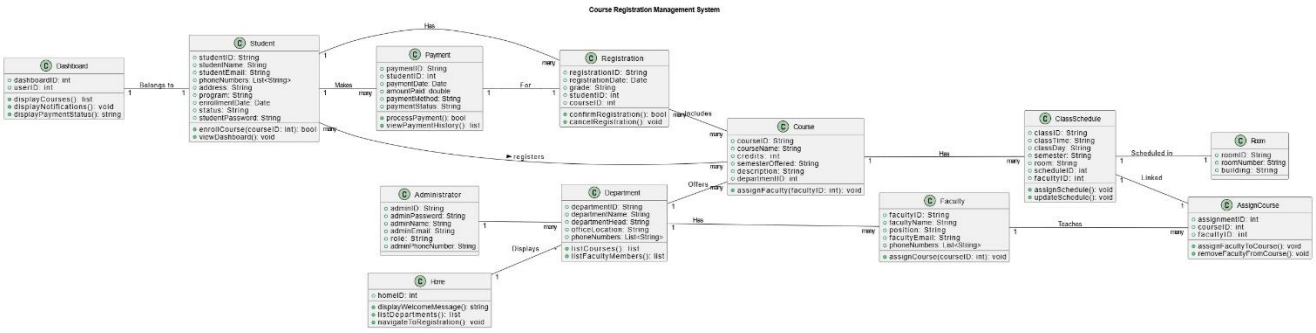
1. **More Students Get Registered:** The chatbot will help students quickly find available **courses** so they don't miss out.
2. **Less Pressure on Human Advisors:** Since the chatbot answers common questions, advisors can focus on more complex issues.
3. **Faster Decision-Making:** Students won't have to waste time searching for courses manually.
4. **Better Student Experience:** A smooth and stress-free registration process
5. **Smarter Course Planning:** Universities will know in advance which courses need more seats.

Diagrams

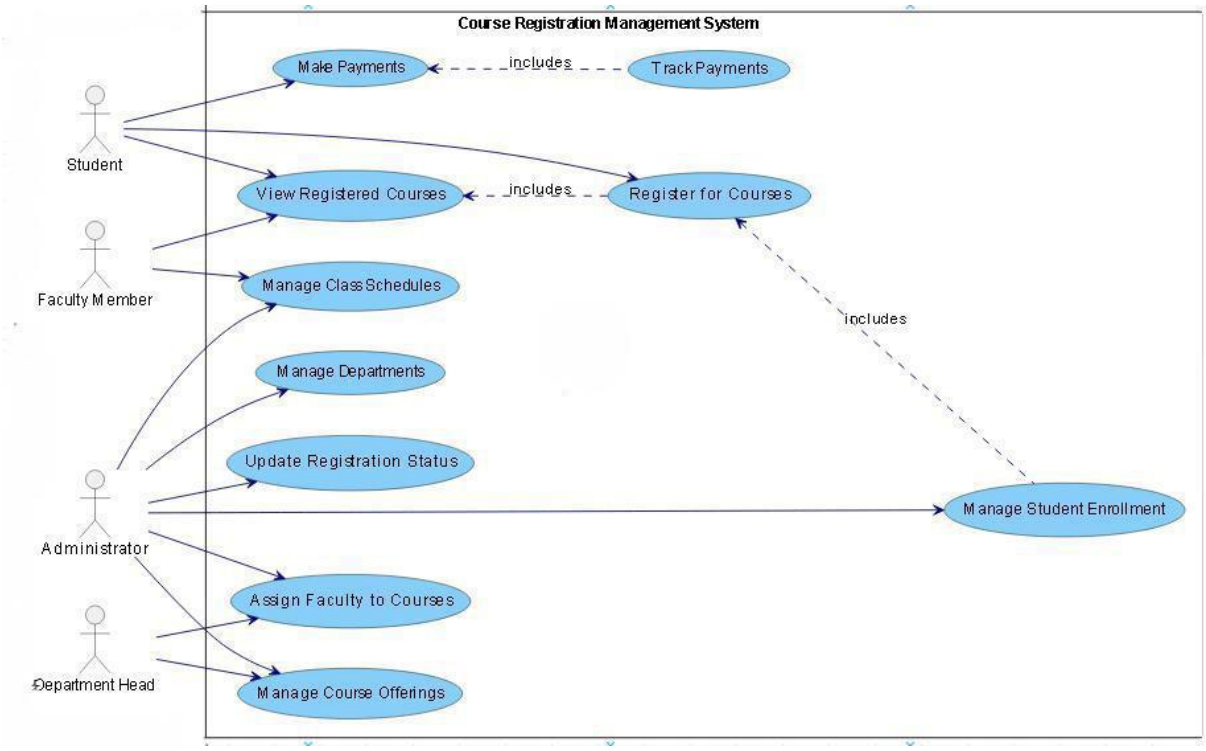
Activity Diagram:



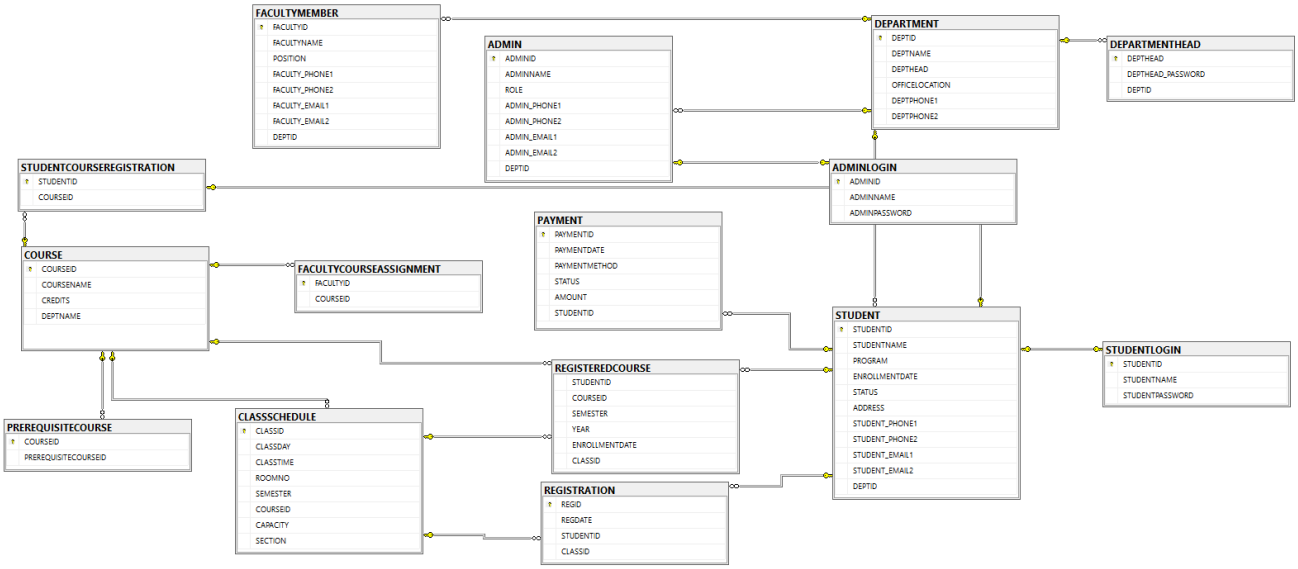
Class Diagram:



Use Case Diagram:



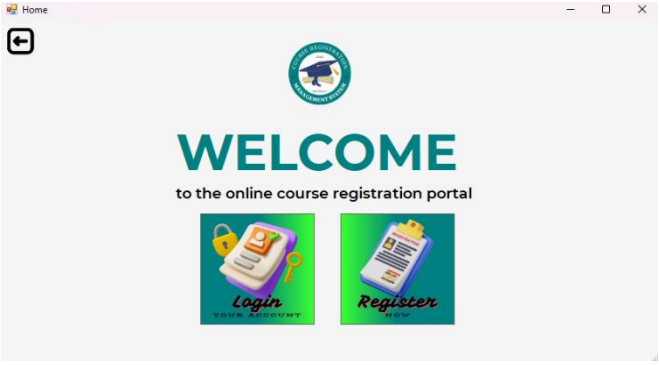
Schema Diagram

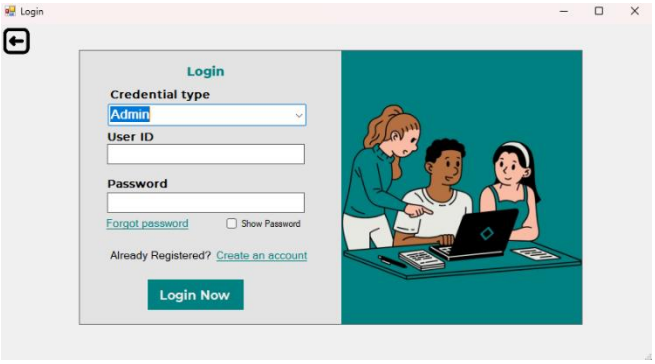


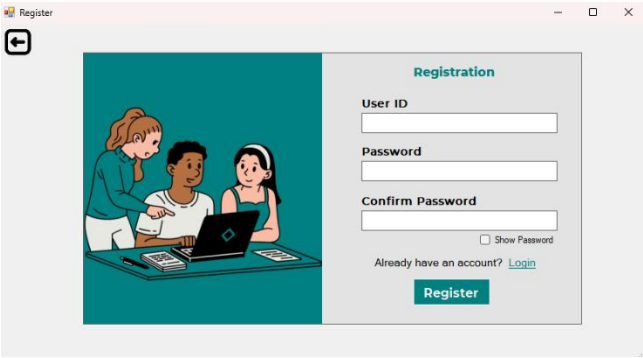
User Interface

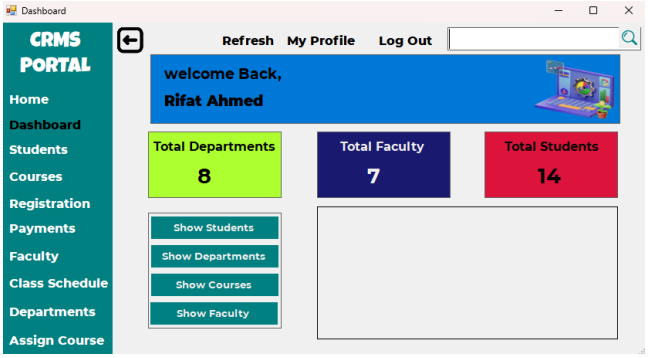
Interface Design

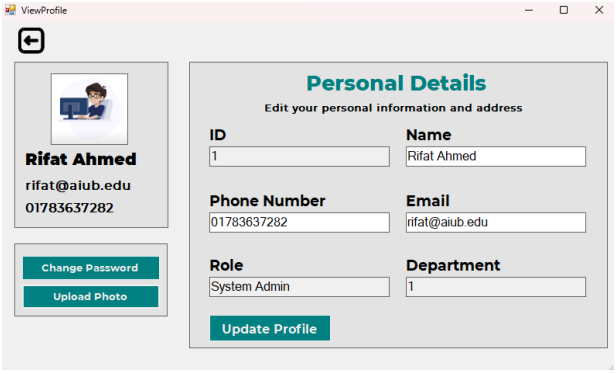
Admin Panel

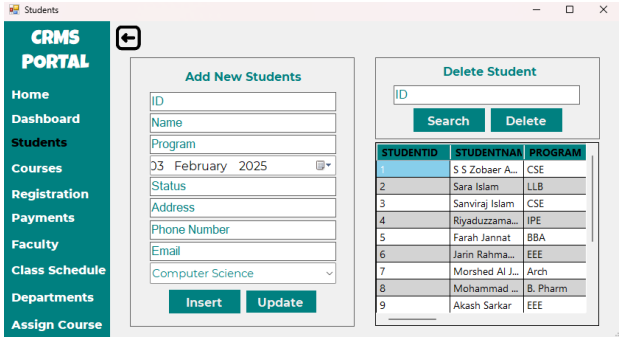


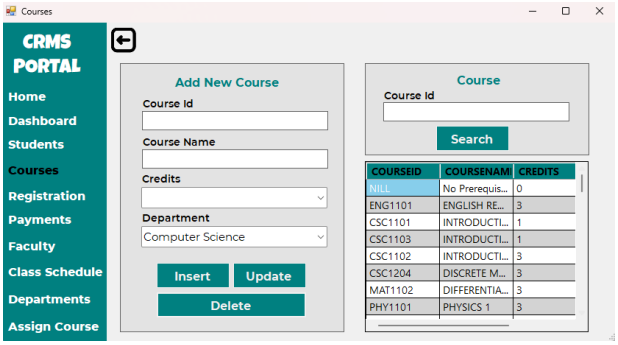


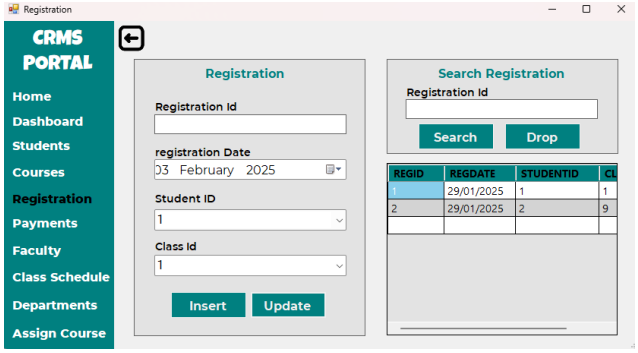


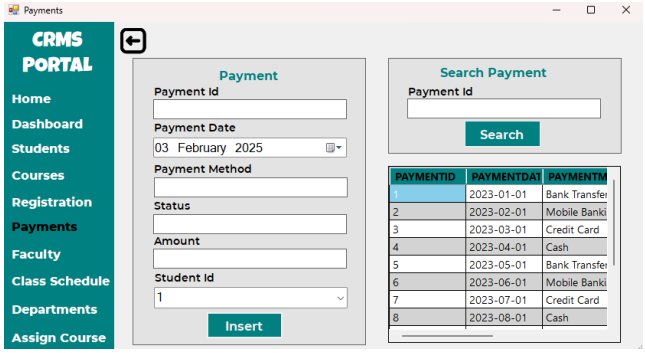


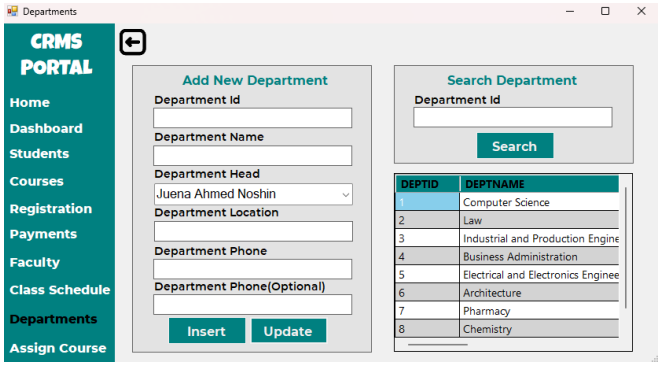




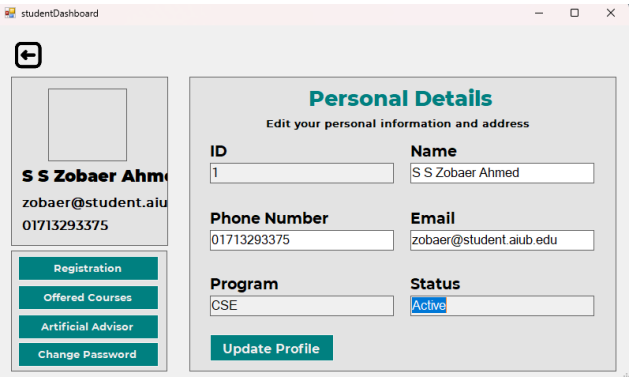


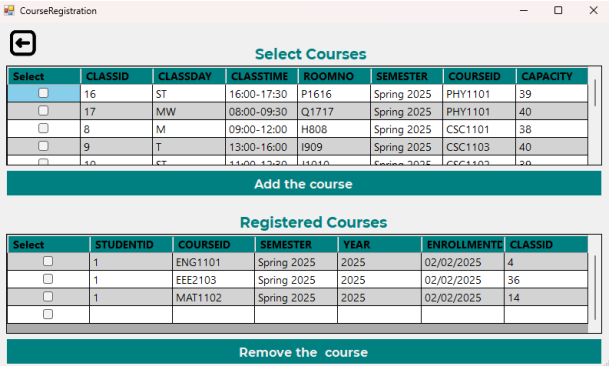


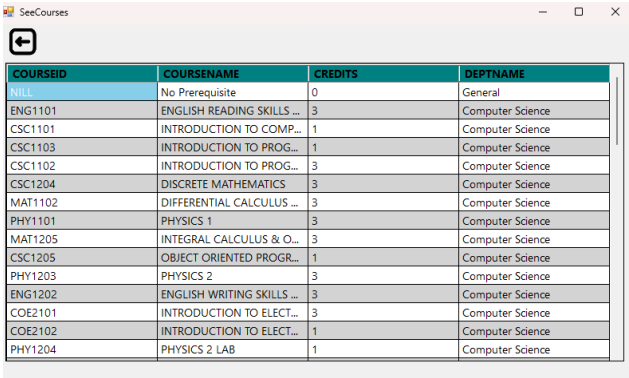




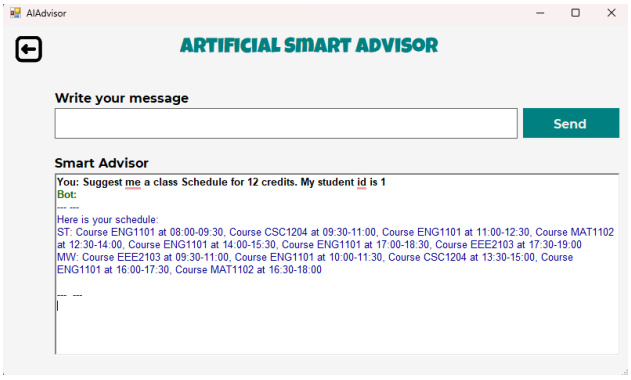
Student Panel







COURSEID	COURSENAME	CREDITS	DEPTNAME
NILL	No Prerequisite	0	General
ENG1101	ENGLISH READING SKILLS ...	3	Computer Science
CSC1101	INTRODUCTION TO COMP...	1	Computer Science
CSC1103	INTRODUCTION TO PROG...	1	Computer Science
CSC1102	INTRODUCTION TO PROG...	3	Computer Science
CSC1204	DISCRETE MATHEMATICS	3	Computer Science
MAT1102	DIFFERENTIAL CALCULUS ...	3	Computer Science
PHY1101	PHYSICS 1	3	Computer Science
MAT1205	INTEGRAL CALCULUS & O...	3	Computer Science
CSC1205	OBJECT ORIENTED PROGR...	1	Computer Science
PHY1203	PHYSICS 2	3	Computer Science
ENG1202	ENGLISH WRITING SKILLS ...	3	Computer Science
COE2101	INTRODUCTION TO ELECT...	3	Computer Science
COE2102	INTRODUCTION TO ELECT...	1	Computer Science
PHY1204	PHYSICS 2 LAB	1	Computer Science

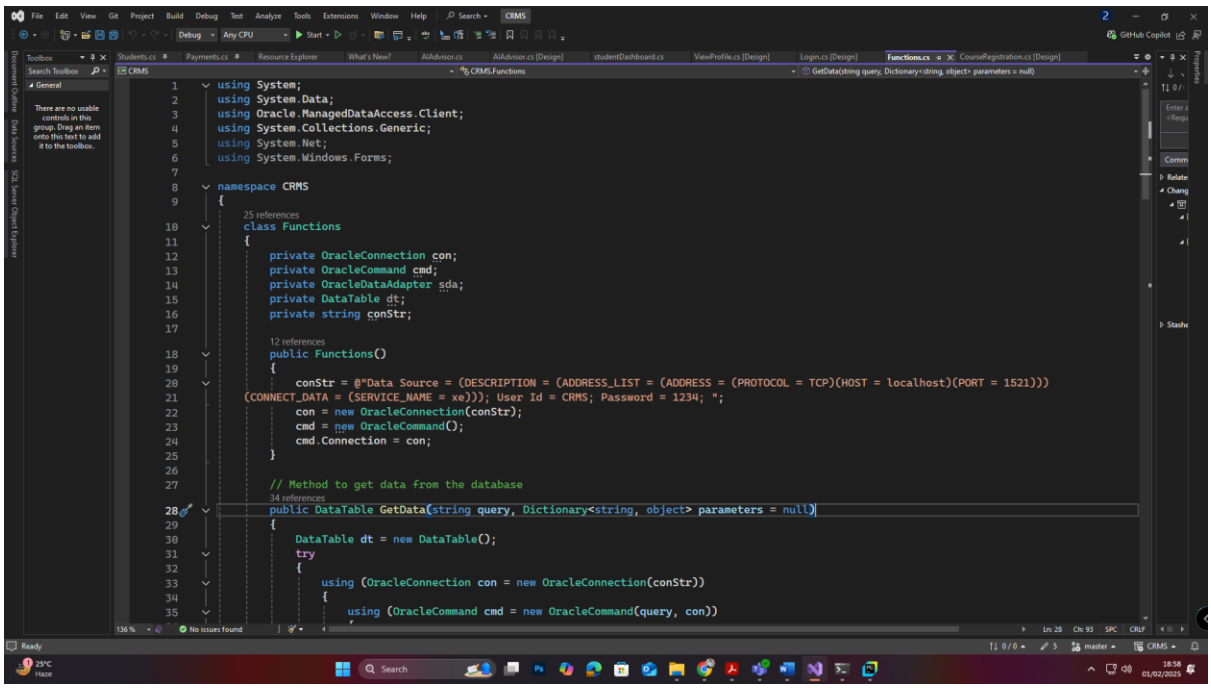


Final term interface design code

Find out the code here: Course Registration Management System

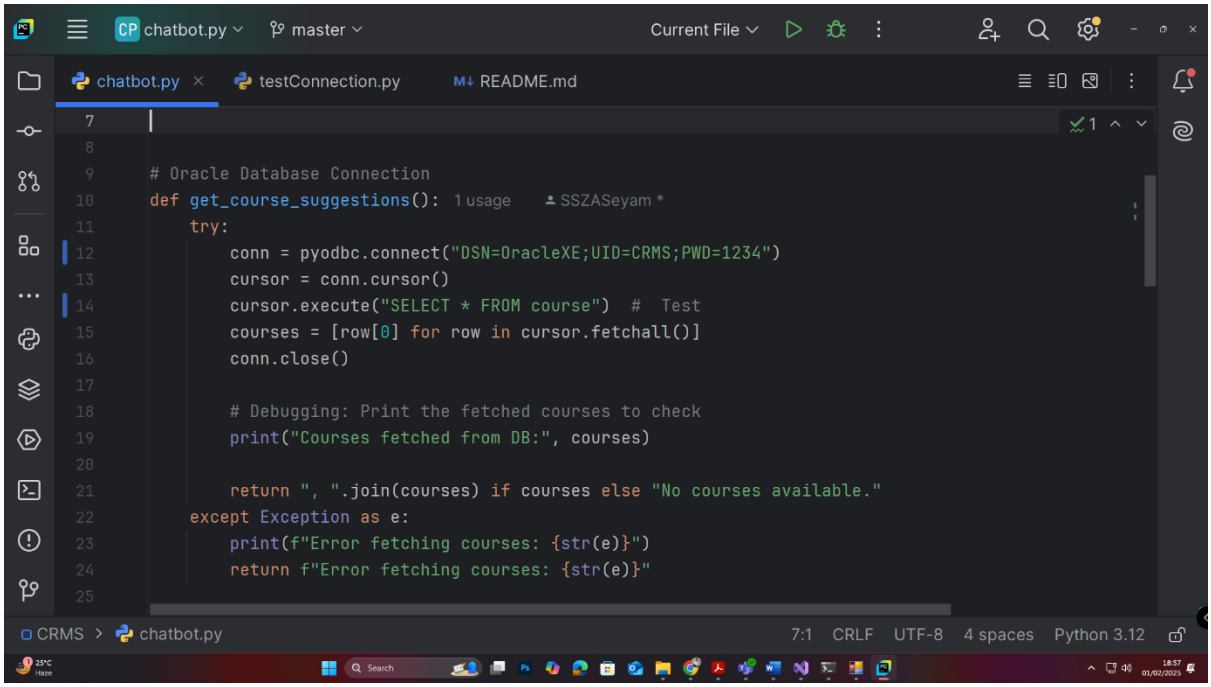
Database Connection

Connection in C#



```
using System;
using System.Data;
using Oracle.ManagedDataAccess.Client;
using System.Collections.Generic;
using System.Net;
using System.Windows.Forms;

namespace CRMS
{
    25 references
    class Functions
    {
        private OracleConnection con;
        private OracleCommand cmd;
        private OracleDataAdapter sda;
        private DataTable dt;
        private string conStr;

        12 references
        public Functions()
        {
            conStr = @"Data Source = (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521)))
    (CONNECT_DATA = (SERVICE_NAME = xe))); User Id = CRMS; Password = 1234; ";
            con = new OracleConnection(conStr);
            cmd = new OracleCommand();
            cmd.Connection = con;
        }

        // Method to get data from the database
        34 references
        public DataTable GetData(string query, Dictionary<string, object> parameters = null)
        {
            DataTable dt = new DataTable();
            try
            {
                using (OracleConnection con = new OracleConnection(conStr))
                {
                    using (OracleCommand cmd = new OracleCommand(query, con))
```

Connection in Python



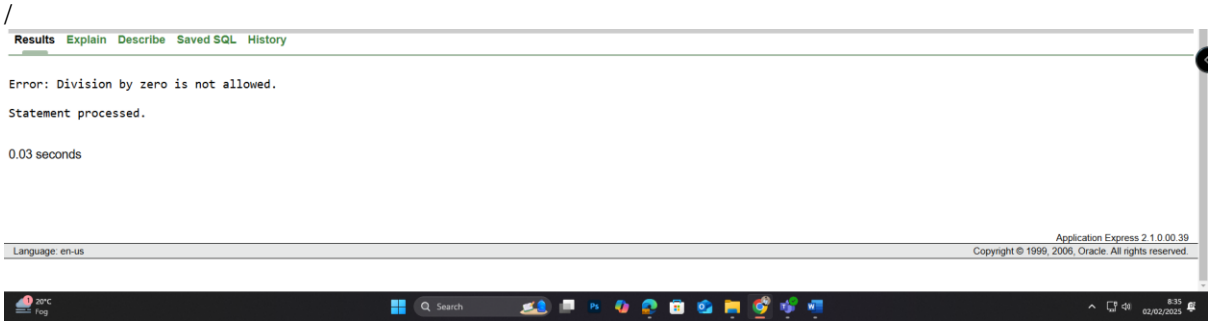
Query Writing

Exception handling

Question 1: Write a PL/SQL block to handle division by zero.

Answer:

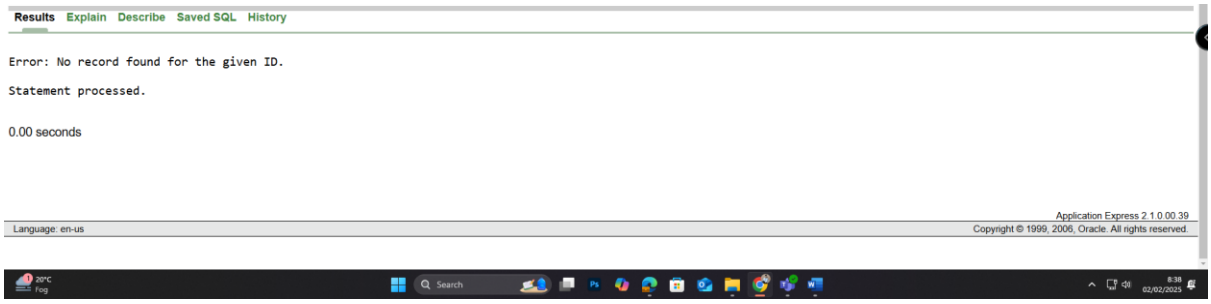
```
DECLARE
   num1 NUMBER := 10;
   num2 NUMBER := 0;
   result NUMBER;
BEGIN
   result := num1 / num2;
   DBMS_OUTPUT.PUT_LINE('Result: ' || result);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Error: Division by zero is not allowed.');
END;
/
```



Question 2: Write a PL/SQL block to handle a NO_DATA_FOUND exception when querying an empty table.

Answer:

```
DECLARE
   deptName VARCHAR2(100);
BEGIN
   SELECT DEPTNAME INTO deptName FROM DEPARTMENT WHERE DEPTID = 0;
   DBMS_OUTPUT.PUT_LINE('Name: ' || deptName);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: No record found for the given ID.');
END;
/
```

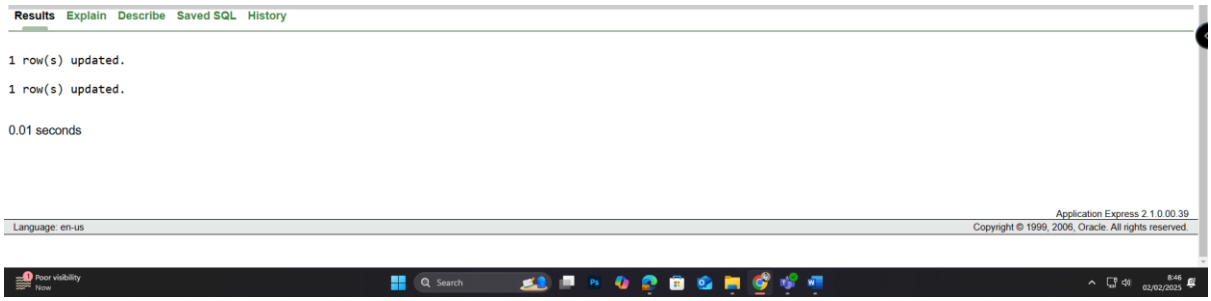


Implicit Locking

Question 1: Write a PL/SQL block to demonstrate implicit locking from student table using an UPDATE statement.

Answer:

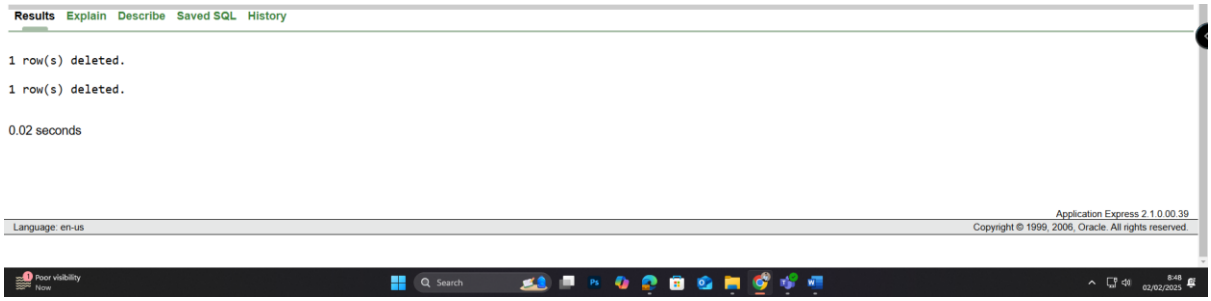
```
DECLARE
   v_rows NUMBER;
BEGIN
   UPDATE student
   SET studentname = 'Sanvi Raj'
   WHERE studentid = 7;

   v_rows := SQL%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE(v_rows || ' row(s) updated.');
END;
/
```



Question 2: Write a PL/SQL block to demonstrate implicit locking from faculty member table using DELETE.

Answer:

```
DECLARE
   v_rows NUMBER;
BEGIN
   DELETE FROM facultyMember
   WHERE facultyid = 5;

   v_rows := SQL%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE(v_rows || ' row(s) deleted.');
END;
/
```

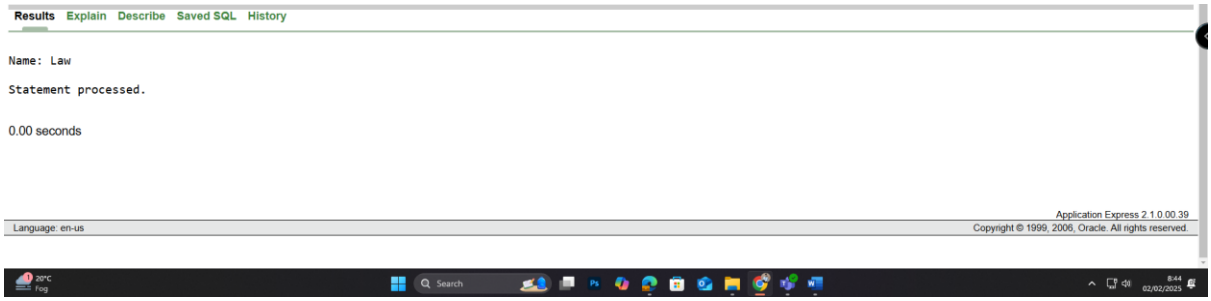


Explicit Locking

Question 1: Write a PL/SQL block to demonstrate explicit locking on Department table using SELECT FOR UPDATE .

Answer:

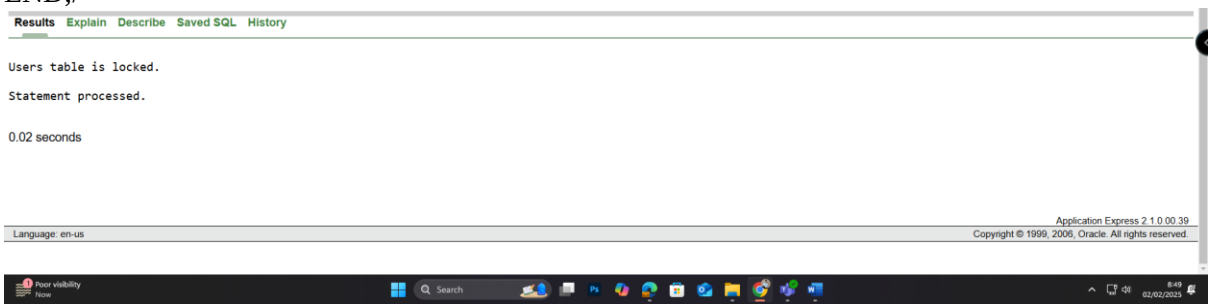
```
DECLARE
   dept_Name VARCHAR2(50);
BEGIN
   SELECT deptname
   INTO dept_Name
   FROM Department
   WHERE Deptid = 2
   FOR UPDATE;

   DBMS_OUTPUT.PUT_LINE('Name: ' || dept_Name);
END;
/
```



Question 2: Write a PL/SQL block to demonstrate the use of LOCK TABLE for explicit locking.

Answer:

```
BEGIN
   LOCK TABLE registeredcourse IN EXCLUSIVE MODE;
   DBMS_OUTPUT.PUT_LINE('Users table is locked.');
END;/
```



Relational Algebra

1. **Retrieve all students from the "Computer Science" department**

   Relational Algebra:
   σ DEPTNAME='Computer Science'(STUDENT)\sigma_{\text{DEPTNAME} = 'Computer Science'} (\text{STUDENT})σDEPTNAME='ComputerScience'(STUDENT)
   Explanation:
   This selection (σ) retrieves all students whose DEPTNAME is "Computer Science" from the STUDENT table.

2. **Retrieve student names and emails who are registered for a course**

   Relational Algebra:
   πSTUDENTNAME, STUDENT_EMAIL1(REGISTEREDCOURSE⋈STUDENT)
   Explanation:
   This natural join (⋈) combines REGISTEREDCOURSE and STUDENT on STUDENTID.
   The projection (π) retrieves only STUDENTNAME and STUDENT_EMAIL1.

3. **Find all students who have registered for at least two different courses**

   Relational Algebra:
   Π STUDENTID (REGISTEREDCOURSE ⋈ ρC1(REGISTEREDCOURSE)) – π STUDENTID(REGISTEREDCOURSE)
   Explanation:
   This self-join (⋈) finds students who have more than one course registered.
   The set difference (-) ensures only students with at least two courses are selected.

4. **Find students who have registered for "Introduction to Database"**

   Relational Algebra:
   Π STUDENTNAME (σ COURSENAME = 'Introduction to Database '(COURSE) ⋈ REGISTEREDCOURSE ⋈ STUDENT)

Explanation:
The **selection (σ)** retrieves only rows where COURSENAME = 'Introduction to database'.
The **join (⋈)** ensures we get students registered for this course.
The **projection (π)** returns only STUDENTNAME.

5. **Find all students who have registered for a course or taken a prerequisite course**

Relational Algebra:
Π STUDENTID(REGISTEREDCOURSE)∪ π STUDENTID
(PREREQUISITECOURSE⋈REGISTEREDCOURSE)
Explanation:
The union (∪) retrieves students who have either registered for a course or have completed at least one prerequisite course.
The join (⋈) ensures we fetch prerequisite-related students from REGISTEREDCOURSE.