DATABASE SYSTEMS
Student Registration System
Using PL/SQL and JDBC

Submitted by

1. Krishianjan Lanka
2. Chhavi Khatri
3. Anjani Praneet Meruvu
4. Chakri Venkat Katam

State Univeristy of New york at Binghamton
INFO 532- Database Systems

Under the guidance of
Professor Hafiz M Ali

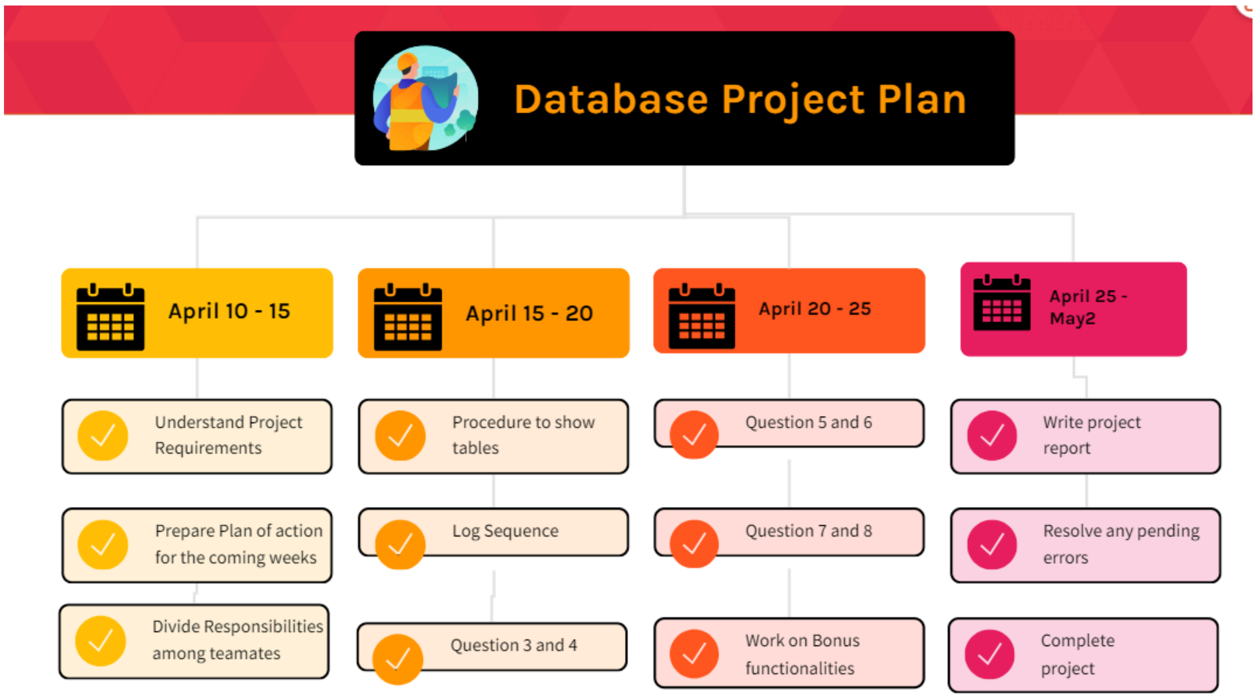
S

1. **Signed Statement**

"I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating, I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment and my grade will be reduced by one level (e.g., from A to A- or from B+ to B) for my first offense, and that I will receive a grade of "F" for the course for any additional offense of any kind"

- Anjani Praneet Meruvu
- Krishianjan Lanka
- Chakri Venkat Katam
- Chhavi Khatri

2. **Plans for the project**



3. Meetings

Meeting 1

- Understood the requirements for the project and planned on how to proceed further
- Installed the required IDE and understood JDBC connection
- Understood the given Demo files

Meeting 2

- Started and executed sequence to generate values for log#
- Started a procedure show_students and implemented

Meeting 3

- Started a procedure to show students ListStudentsInClass which shows the list of all the students in the class. Incase there is no class it shows class is invalid
- We created a procedure to get prerequisites. Including both direct and Indirect ones

Meeting 4

- We created a procedure to enroll students and designed and implemented the EnrollStudentIntoClass logic during a project meeting
- Implemented cases like
  'The classid is invalid'
  'Cannot enroll into a class from a previous semester'

‘The class is already full”

Meeting 5

- Created procedure DropStudentFromClass and implemented cases like
  ‘ Invalid B#: The student does not exist or is not a graduate student.'
  ‘'Not a Graduate Student: This student is not a graduate student.’

Meeting 6

- Created Procedure to Delete student from student table
  If the B# is valid it deletes the student from table
- Created Triggers to add to log table
  - LogStudentDeletion
  - LogEnrollment
  - LogDrop

Meeting 7

- Created package student_mngt and included all the procedures into the package
- Tested if the procedures are working on CLI

Meeting 8

- Started menu driven interface on java and JDBC connected established
- Design for the interface is finished

Meeting 9

- Tested the overall functionality of the project through the interface. There were errors with enrolling through the front end which we have resolved.

4. Roles and Responsibilities

Log Sequence	Chakri
Procedure to show tables	Krishi
Procedure to show class details	Praneet
Procedure to show class details	Praneet
Procedure to return all the prerequisite courses	Krishi
Procedure to enroll students	ALL
Procedure to drop students	ALL
Procedure to delete students	ALL
Triggers for Log tables	Praneet
Creating java menu driven interface	Chhavi , Krishi
Design of the interface	Chhavi , Krishi
Testing and Debugging	Chakri
Documentation	Chakri

5. Self Assessment

The phrase that describes our team is **“worked really well**

together”

6. Explanation of Objects Created

Procedures and Functions

1. ***ListStudentsInClass:***
   Objective: Lists all the students enrolled in a specified class
   Usage: Takes a class ID as input and outputs student details if the class exists.

2. ***GetPrerequisites:***
   Objective: Retrieves and displays the prerequisites for a given course.
   Usage: Accepts department code and course number as parameters and lists direct and indirect prerequisites.

3. ***GetIndirectPrerequisites***:
   Objective: This procedure retrieves all indirect prerequisites for a given course within a department. It's designed to help understand the dependency chain of courses that are prerequisite to a specific course but not directly listed as such.

   Usage: The procedure is called with two parameters: ipre_dept_code (department code of the course) and ipre_course# (course number). It recursively explores and displays all the courses that are indirectly required before one can enroll in the specified course. This is useful for academic advisors and students planning their course schedules, ensuring they meet all necessary prerequisites.

4. ***EnrollStudentIntoClass:***
   Objective: This procedure handles the enrollment of a graduate

student into a specified class, provided several conditions are met such as prerequisites satisfaction, class availability, and the student's enrollment limit for the semester.
Usage: If you'll try to enroll a student, then he/she will be taken if input parameters g_B#_param (id of student) and classid_param. This process confirms the student's validity and class. This ensures that a student is a graduate, verifies the availability of the class with its prerequisites, and registers the student. Handy in administrative functions like student management systems where enrollment conditions are strictly enforced.

5. ***DropStudentFromClass:***
Objective: This process supports the process of removing a graduate student from a class. It makes sure that the student actually exists, is taking the course, and has other courses within the same semester before allowing the dropping operation.

Usage: The procedure requires the student ID (g_B#_param) and class ID (classid_param) as inputs to process the student's removal from the class. It is particularly used in scenarios where maintaining academic load balance and verifying conditions before dropping a class are critical.

6. ***DeleteStudent:***
Objective: This procedure is designed to remove a student from the database based on the provided student ID. It ensures that only valid student IDs are processed, preventing accidental deletions and providing clear feedback on the outcome of the operation.

Usage: To delete a student, the procedure requires the student's

ID (B#_param) as an input. It is particularly useful in administrative functions like maintaining up-to-date records in a student management system, where students who have graduated or withdrawn are removed from active records.

7. <u>Sequence:</u>

ListStudentsInClass:
Objective: Provides a unique sequence number for logging operations in the logs table.

Usage: This sequence starts at 1000 and increments by 1 for each new log entry, ensuring each log entry has a unique identifier, which is critical for maintaining an organized and searchable logging system.

8. <u>Triggers</u>*:*

Triggers

8.1. LogEnrollment :
Objective: Capture and log every new enrollment in table `g_enrollments` into table `logs`. Log the operation together with the user, timestamp, and key values associated with the enrollment.

Usage: Auto-invoked after an insert operation on `g_enrollments`, useful in audit trails and tracking changes in enrollment data.

8.2. LogDrop

Objective: Capture and log every "drop" operation (deletion from \`g_enrollments\`) into the \`logs\` table. This will include the logging of the user, time of operation, and details of the dropped enrollment.

Usage: Triggered after the delete operation of \`g_enrollments\`. This will help you retain history of all class drop activity that would help you in administrative tracking and audits.

8.3. UpdateClassSizeOnDrop

Objective: It automatically decrements the count of class size in the classes table whenever there's one student dropped from the class.

Usage: A delete operation on g_enrollments will need this trigger to be invoked so that the class size is always up-to-date with the current number of enrollments.

8.4. PreventFullClassEnrollmentTrigger

Objective: Prevents the enrollment of students into a class that has reached its capacity limit.

Usage: Executes before an insert on g_enrollments. It checks if the class is full and raises an application error if an enrollment attempt violates the capacity constraint.

8.5. UpdateClassSizeTrigger

Objective: Ensures the class size in the classes table is updated to reflect new enrollments by incrementing the current class size.

Usage: Activated after an insert operation on g_enrollments. Keeps the class size up-to-date following new student enrollments.

SQL Package:

```
CREATE OR REPLACE PACKAGE student_mngt
IS
   PROCEDURE ListStudentsInClass(classid_param IN CHAR);
   PROCEDURE GetPrerequisites(dept_code_param IN VARCHAR2, course#_param IN NUMBER);
   PROCEDURE GetIndirectPrerequisites(ipre_dept_code IN VARCHAR2, ipre_course# IN NUMBER);
   PROCEDURE EnrollStudentIntoClass(g_B#_param IN CHAR, classid_param IN CHAR);
   PROCEDURE DropStudentFromClass(g_B#_param IN CHAR,classid_param IN CHAR);
   PROCEDURE DeleteStudent(B#_param IN CHAR);
END student_mngt;
/

CREATE OR REPLACE PROCEDURE SHOW_STUDENTS
IS
   CURSOR student_cursor IS
      SELECT B#, first_name, last_name, st_level, gpa, email, bdate
      FROM students;
   student_record student_cursor%ROWTYPE;
BEGIN

   OPEN student_cursor;

   LOOP
      FETCH student_cursor INTO student_record;
      EXIT WHEN student_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Student ID: ' || student_record.B#);
      DBMS_OUTPUT.PUT_LINE('First Name: ' || student_record.first_name);
      DBMS_OUTPUT.PUT_LINE('Last Name: ' || student_record.last_name);
      DBMS_OUTPUT.PUT_LINE('Student Level: ' || student_record.st_level);
```

```
        DBMS_OUTPUT.PUT_LINE('GPA: ' || student_record.gpa);
        DBMS_OUTPUT.PUT_LINE('Email: ' || student_record.email);
        DBMS_OUTPUT.PUT_LINE('Birth Date: ' || student_record.bdate);
        DBMS_OUTPUT.PUT_LINE('----------------------------');
    END LOOP;

    CLOSE student_cursor;
END;
/

CREATE OR REPLACE PROCEDURE ListStudentsInClass(classid_param IN CHAR) IS
  v_class_exists NUMBER;
BEGIN
  -- if classid exists
  SELECT COUNT(*) INTO v_class_exists
  FROM classes
  WHERE classid = classid_param;

  IF v_class_exists = 0 THEN
    DBMS_OUTPUT.PUT_LINE('The classid is invalid.');
  ELSE

    FOR student_record IN (
      SELECT s.B#, s.first_name, s.last_name
      FROM students s
      JOIN g_enrollments g ON s.B# = g.g_B#
      WHERE g.classid = classid_param
    )
    LOOP
      DBMS_OUTPUT.PUT_LINE('Student ID: ' || student_record.B#);
      DBMS_OUTPUT.PUT_LINE('First Name: ' || student_record.first_name);
      DBMS_OUTPUT.PUT_LINE('Last Name: ' || student_record.last_name);
      DBMS_OUTPUT.PUT_LINE('----------------------------');
    END LOOP;
  END IF;
END ListStudentsInClass;
/


CREATE OR REPLACE PROCEDURE ListStudentsInClass(classid_param IN CHAR) IS
  v_class_exists NUMBER;
BEGIN
  -- if classid exists
  SELECT COUNT(*) INTO v_class_exists
  FROM classes
  WHERE classid = classid_param;

  IF v_class_exists = 0 THEN
    DBMS_OUTPUT.PUT_LINE('The classid is invalid.');
  ELSE

    FOR student_record IN (
      SELECT s.B#, s.first_name, s.last_name
      FROM students s
      JOIN g_enrollments g ON s.B# = g.g_B#
      WHERE g.classid = classid_param
    )
    LOOP
      DBMS_OUTPUT.PUT_LINE('Student ID: ' || student_record.B#);
      DBMS_OUTPUT.PUT_LINE('First Name: ' || student_record.first_name);
      DBMS_OUTPUT.PUT_LINE('Last Name: ' || student_record.last_name);
      DBMS_OUTPUT.PUT_LINE('----------------------------');
    END LOOP;
  END IF;
END ListStudentsInClass;
/



CREATE OR REPLACE PROCEDURE GetPrerequisites(
  dept_code_param IN VARCHAR2,
  course#_param IN NUMBER
) IS
  v_course_exists NUMBER;


  CURSOR DirectPrerequisitesCursor IS
    SELECT pre_dept_code, pre_course#
    FROM prerequisites
    WHERE dept_code = dept_code_param
    AND course# = course#_param;


  CURSOR IndirectPrerequisitesCursor(
    ipre_dept_code VARCHAR2,
    ipre_course# NUMBER
  ) IS
    SELECT pre_dept_code, pre_course#
    FROM prerequisites
    WHERE dept_code = ipre_dept_code
    AND course# = ipre_course#;


  v_depth NUMBER := 0;


  PROCEDURE GetIndirectPrerequisites(
    ipre_dept_code IN VARCHAR2,
    ipre_course# IN NUMBER
  ) IS
  BEGIN

    v_depth := v_depth + 1;


    FOR indirect_prereq IN (
      SELECT pre_dept_code, pre_course#
      FROM prerequisites
      WHERE dept_code = ipre_dept_code
      AND course# = ipre_course#
    )
    LOOP

      DBMS_OUTPUT.PUT_LINE(RPAD(' ', v_depth*2, ' ') || indirect_prereq.pre_dept_code ||
indirect_prereq.pre_course#);

      GetIndirectPrerequisites(indirect_prereq.pre_dept_code, indirect_prereq.pre_course#);
    END LOOP;


    v_depth := v_depth - 1;
  END GetIndirectPrerequisites;

BEGIN

  SELECT COUNT(*) INTO v_course_exists
  FROM courses
  WHERE dept_code = dept_code_param
  AND course# = course#_param;

  IF v_course_exists = 0 THEN
    DBMS_OUTPUT.PUT_LINE(dept_code_param || course#_param || ' does not exist.');
  ELSE
    -- Display direct prerequisites
    DBMS_OUTPUT.PUT_LINE('Direct Prerequisites:');
    FOR direct_prereq IN DirectPrerequisitesCursor LOOP
      DBMS_OUTPUT.PUT_LINE(direct_prereq.pre_dept_code || direct_prereq.pre_course#);

      -- Call GetIndirectPrerequisites for indirect prerequisites
      GetIndirectPrerequisites(direct_prereq.pre_dept_code, direct_prereq.pre_course#);
    END LOOP;
  END IF;
END GetPrerequisites;
/


CREATE OR REPLACE PROCEDURE EnrollStudentIntoClass(
    g_B#_param IN CHAR,
    classid_param IN CHAR
) IS
    v_student_exists NUMBER;
    v_is_grad_student VARCHAR2(10);
    v_class_exists NUMBER;
    v_class_semester VARCHAR2(20);
    v_class_size NUMBER;
    v_class_limit NUMBER;
    v_student_enrollments NUMBER;
    v_student_semester_classes NUMBER;
BEGIN
    SELECT COUNT(*), st_level
    INTO v_student_exists, v_is_grad_student
    FROM students
    WHERE B# = g_B#_param
      AND st_level IN ('master', 'PhD')
    GROUP BY st_level;

    IF v_student_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Invalid B#: The student does not exist or is not a graduate student.');
    END IF;

    IF v_is_grad_student IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Not a Graduate Student: This student is not a graduate student.');
    END IF;

    SELECT COUNT(*), semester, limit, class_size
    INTO v_class_exists, v_class_semester, v_class_limit, v_class_size
    FROM classes
    WHERE classid = classid_param
    GROUP BY semester, limit, class_size;

    IF v_class_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Invalid Class ID: The specified class does not exist.');
    END IF;

    IF v_class_size >= v_class_limit THEN
        RAISE_APPLICATION_ERROR(-20005, 'Class Full: The class is already full.');
    END IF;

    SELECT COUNT(*)
    INTO v_student_enrollments
    FROM g_enrollments
    WHERE g_B# = g_B#_param
      AND classid = classid_param;

    IF v_student_enrollments > 0 THEN
        RAISE_APPLICATION_ERROR(-20006, 'Already Enrolled: The student is already enrolled in this class.');
    END IF;

    SELECT COUNT(*)
    INTO v_student_semester_classes
    FROM g_enrollments ge
    JOIN classes c ON ge.classid = c.classid
    WHERE ge.g_B# = g_B#_param
      -- Remove the 'Spring 2021' semester condition
      AND c.year = EXTRACT(YEAR FROM SYSDATE)
    GROUP BY ge.g_B#;

    IF v_student_semester_classes >= 5 THEN
        RAISE_APPLICATION_ERROR(-20007, 'Exceeded Limit: Students cannot be enrolled in more than five classes
in a semester.');
    END IF;

    -- If all checks pass, proceed with enrollment
    INSERT INTO g_enrollments (g_B#, classid)
    VALUES (g_B#_param, classid_param);
    DBMS_OUTPUT.PUT_LINE('Enrollment successful.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END EnrollStudentIntoClass;
/

CREATE OR REPLACE PROCEDURE DropStudentFromClass(
    g_B#_param IN CHAR,
    classid_param IN CHAR
) IS
    v_student_exists NUMBER;
    v_is_grad_student VARCHAR2(10);
    v_class_exists NUMBER;
    v_student_enrolled NUMBER;
    v_other_classes NUMBER;
BEGIN
    -- Check if the student exists and is a graduate student
    SELECT COUNT(*), st_level
    INTO v_student_exists, v_is_grad_student
    FROM students
    WHERE B# = g_B#_param
      AND st_level IN ('master', 'PhD')
    GROUP BY st_level;

    IF v_student_exists = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Invalid B#: The student does not exist or is not a graduate student.');
        RETURN;
    END IF;

    IF v_is_grad_student IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('Not a Graduate Student: This student is not a graduate student.');
        RETURN;
    END IF;

    -- Check if the class exists
    SELECT COUNT(*)
    INTO v_class_exists
    FROM classes
    WHERE classid = classid_param;

    IF v_class_exists = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Invalid Class ID: The specified class does not exist.');
        RETURN;
    END IF;

    -- Check if the student is enrolled in the class
    SELECT COUNT(*)
    INTO v_student_enrolled
    FROM g_enrollments
    WHERE g_B# = g_B#_param
      AND classid = classid_param;

    IF v_student_enrolled = 0 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Not Enrolled: The student is not enrolled in this class.');
        RETURN;
    END IF;

    -- Check if the student is enrolled in other classes for the current semester
    SELECT COUNT(*)
    INTO v_other_classes
    FROM g_enrollments ge
    WHERE ge.g_B# = g_B#_param
      AND ge.classid <> classid_param;

    IF v_other_classes = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Cannot Drop: This is the only class for this student in the current semester.');
        RETURN;
    END IF;

    -- Remove the student from the class
    DELETE FROM g_enrollments
    WHERE g_B# = g_B#_param
      AND classid = classid_param;

    DBMS_OUTPUT.PUT_LINE('Student dropped from the class successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred while dropping the student from the class.');
END DropStudentFromClass;
/


CREATE OR REPLACE PROCEDURE DeleteStudent(
    B#_param IN CHAR
) IS
BEGIN

    DELETE FROM students
    WHERE B# = B#_param;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('The B# is invalid.');
        RETURN;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Student deleted successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END DeleteStudent;
/
CREATE OR REPLACE TRIGGER DeleteEnrollmentsOnStudentDelete
BEFORE DELETE ON students
FOR EACH ROW
BEGIN
    DELETE FROM g_enrollments
    WHERE g_B# = :OLD.B#;
```

```
END;
/


CREATE OR REPLACE TRIGGER LogStudentDeletion
AFTER DELETE ON students
FOR EACH ROW
BEGIN
   INSERT INTO logs (log#, user_name, op_time, table_name, operation, tuple_keyvalue)
   VALUES (
      log_sequence.NEXTVAL, USER, SYSDATE, 'Students', 'DELETE', :OLD.B#
   );
END;
/



CREATE OR REPLACE TRIGGER LogEnrollment
AFTER INSERT ON g_enrollments
FOR EACH ROW
BEGIN
   INSERT INTO logs (log#, user_name, op_time, table_name, operation, tuple_keyvalue)
   VALUES (
      log_sequence.NEXTVAL, USER, SYSDATE, 'G_Enrollments', 'INSERT', :NEW.g_B# || ',' || :NEW.classid
   );
END;
/


CREATE OR REPLACE TRIGGER LogDrop
AFTER DELETE ON g_enrollments
FOR EACH ROW
BEGIN
   INSERT INTO logs (log#, user_name, op_time, table_name, operation, tuple_keyvalue)
   VALUES (
      log_sequence.NEXTVAL, USER, SYSDATE, 'G_Enrollments', 'DELETE', :OLD.g_B# || ',' || :OLD.classid
   );
END;
```

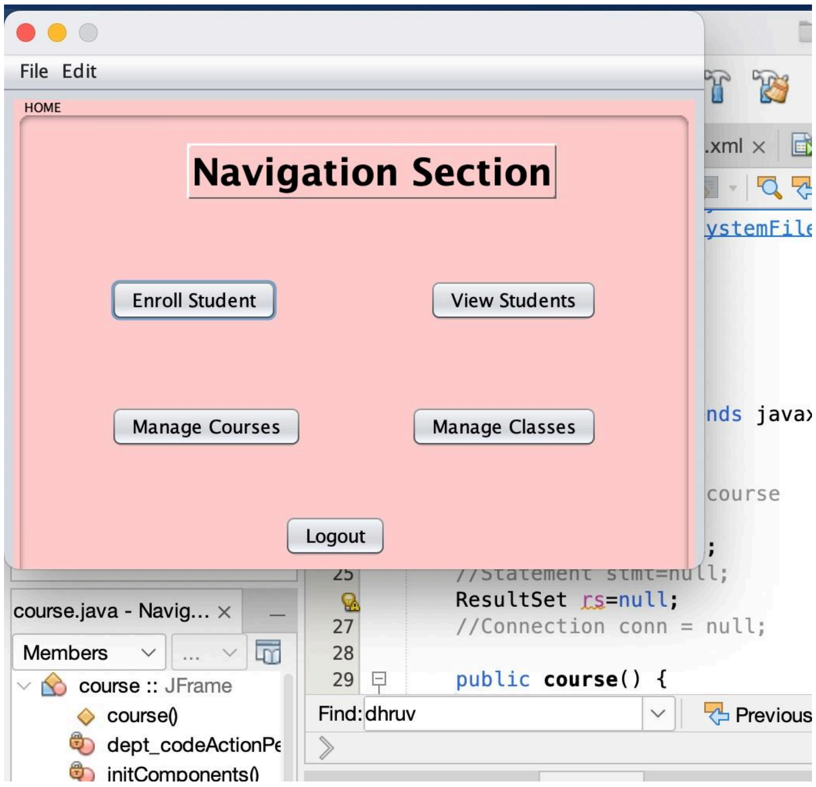
JDBC Connection :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;

public class databaseConnection {
   static String DB_URL = "jdbc:oracle:thin:@castor.cc.binghamton.edu:1521:ACAD111"; // Update the
database URL
```

```
    static String USER = "klanka"; // Update the username as per your Harvey account starting
    static String PASS = "im clearing my pwd user can enter there password based on there user name "; //
Update the password
    static String JDBC_Driver = "oracle.jdbc.driver.OracleDriver";

    public static Connection connection() {
        try {
            Class.forName(JDBC_Driver);
            System.out.println("Connected");
            return DriverManager.getConnection(DB_URL, USER, PASS);
        } catch (ClassNotFoundException | SQLException e) {
            JOptionPane.showMessageDialog(null, e);
            return null;
        }
    }
}
```

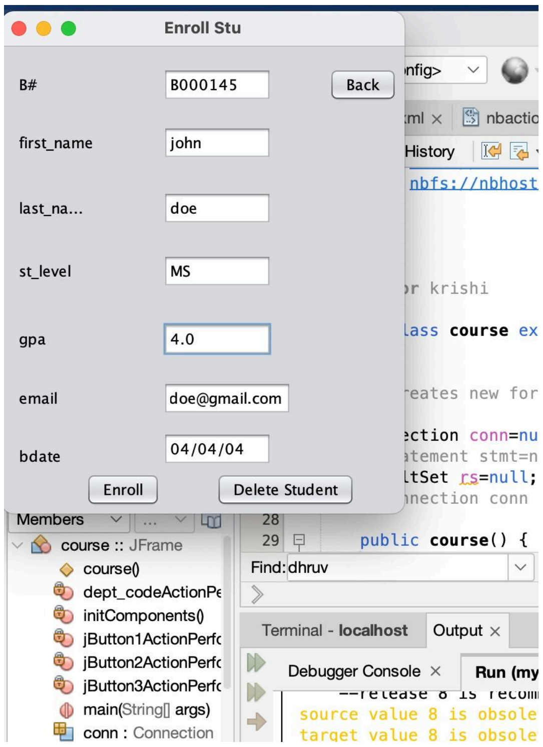
Screenshots of the working Project :

1. Navigation section (home)



```
    //Statement stmt=null;
    ResultSet rs=null;
    //Connection conn = null;

    public course() {
```

2. Enroll Student ;

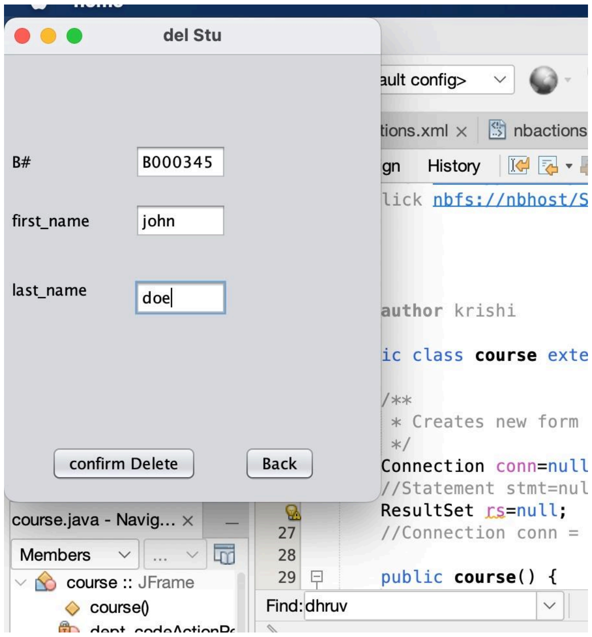


B#	B000145
first_name	john
last_na...	doe
st_level	MS
gpa	4.0
email	doe@gmail.com
bdate	04/04/04

```
nbfs://nbhost
or krishi
lass course ex
reates new for
ection conn=nu
atement stmt=n
ltSet rs=null;
nnection conn
public course() {
```

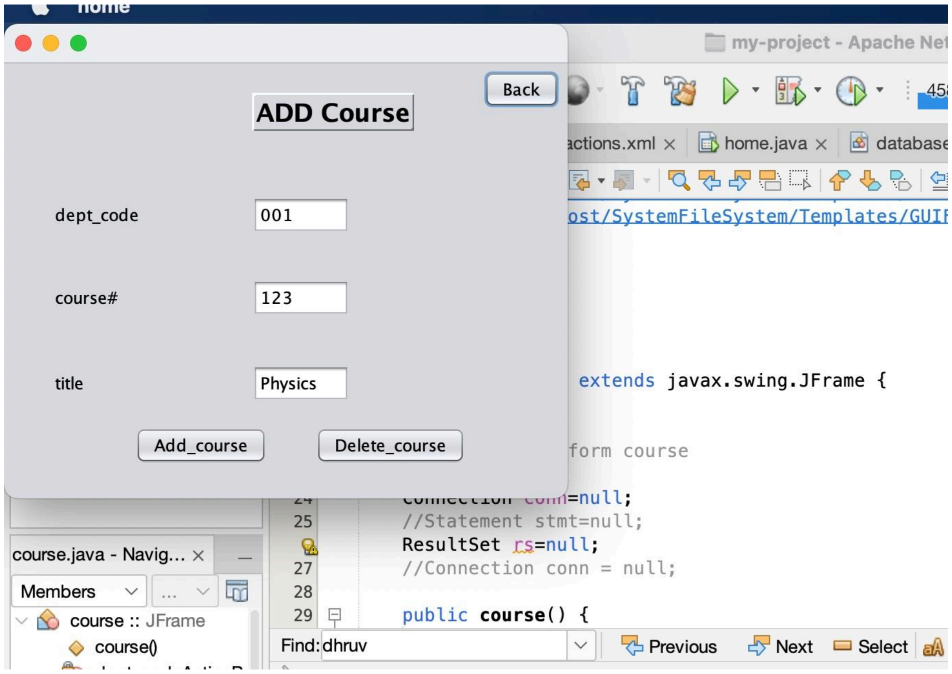
```
--release 8 is recomm
source value 8 is obsole
target value 8 is obsole
```

3. Delete Student ;



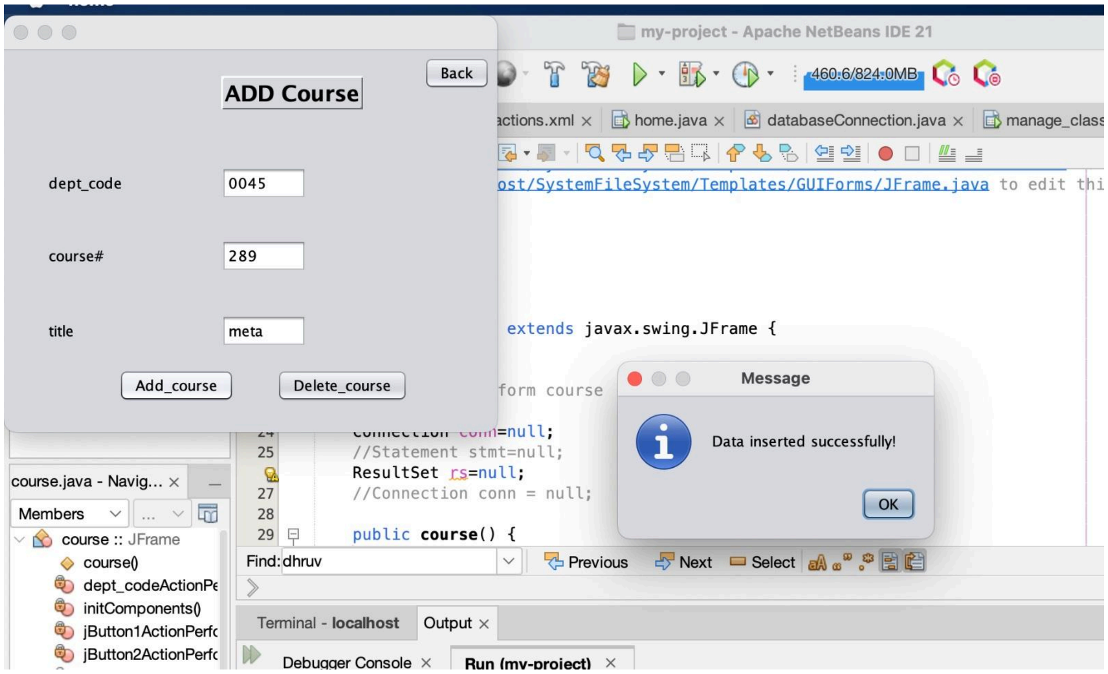
```
lick nbfs://nbhost/S

author krishi

ic class course exte

/**
 * Creates new form
 */
Connection conn=null
//Statement stmt=nul
ResultSet rs=null;
//Connection conn =

public course() {
```

4. Add Course ;



```
ost/SystemFileSystem/Templates/GUI

extends javax.swing.JFrame {

form course

Connection conn=null;
//Statement stmt=null;
ResultSet rs=null;
//Connection conn = null;

public course() {
```

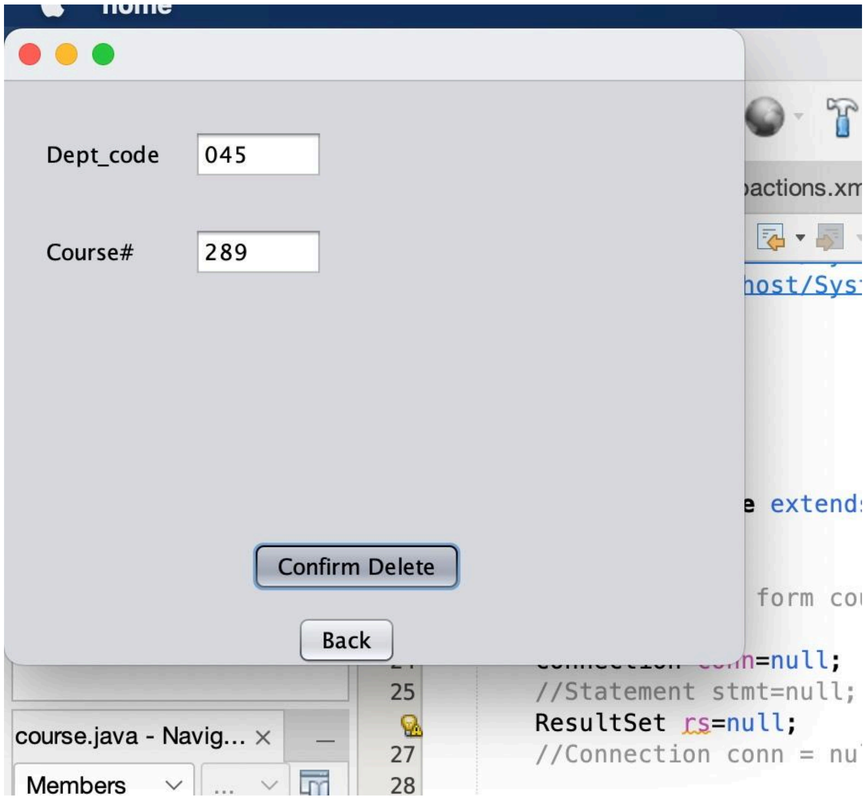
4. Courses Added ;



Data inserted successfully!

```
ost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit thi
 extends javax.swing.JFrame {
form course
        Connection conn=null;
        //Statement stmt=null;
        ResultSet rs=null;
        //Connection conn = null;

        public course() {
```

5. Delete Courses:

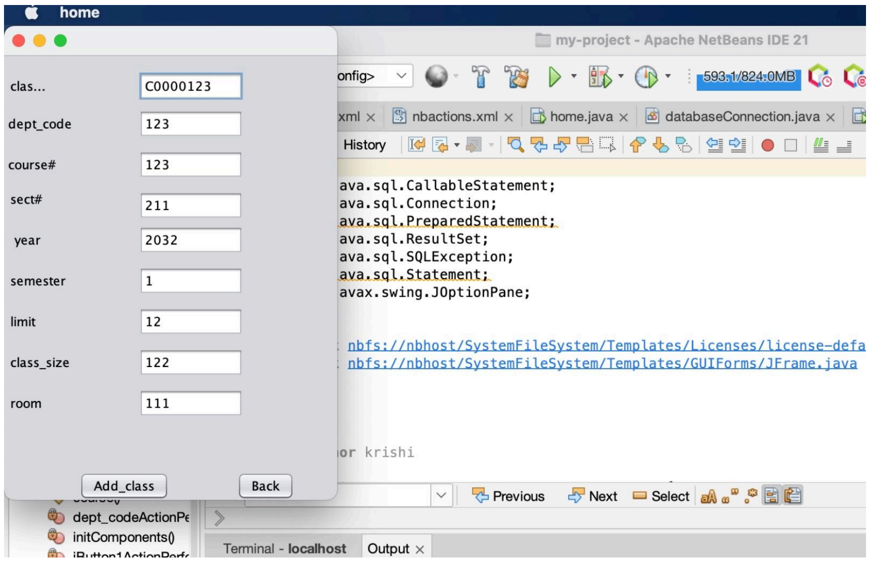


Dept_code 045

Course# 289

```
//Statement stmt=null;
ResultSet rs=null;
//Connection conn = nu
```

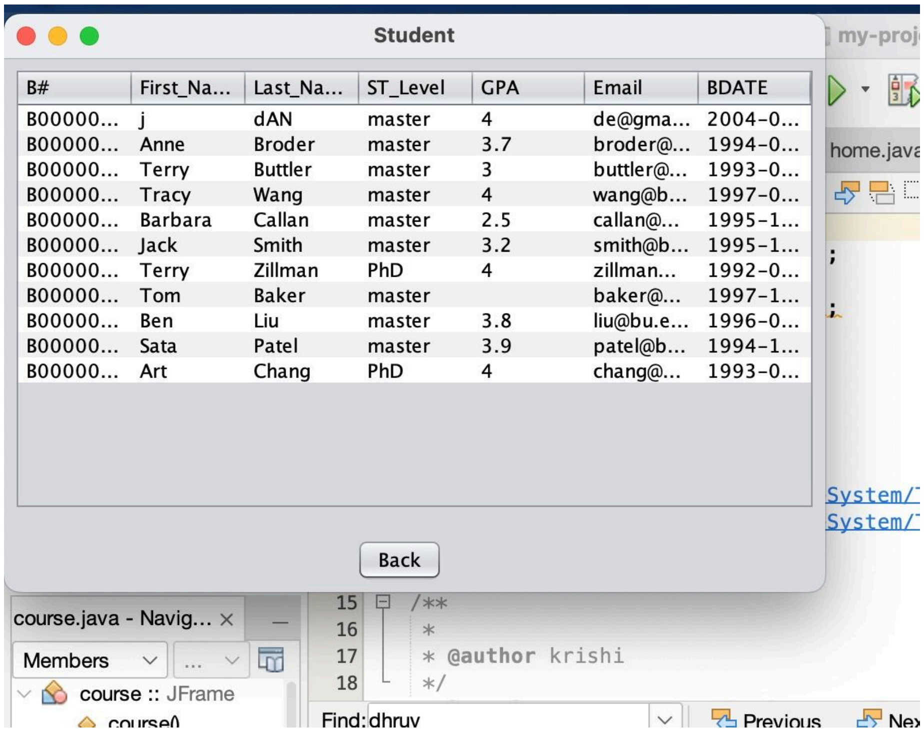
6. Add Classes :



clas...	C0000123
dept_code	123
course#	123
sect#	211
year	2032
semester	1
limit	12
class_size	122
room	111

```
ava.sql.CallableStatement;
ava.sql.Connection;
ava.sql.PreparedStatement;
ava.sql.ResultSet;
ava.sql.SQLException;
ava.sql.Statement;
avax.swing.JOptionPane;

nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-defa
nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java

or krishi
```

7. Show students list



B#	First_Na...	Last_Na...	ST_Level	GPA	Email	BDATE
B00000...	j	dAN	master	4	de@gma...	2004-0...
B00000...	Anne	Broder	master	3.7	broder@...	1994-0...
B00000...	Terry	Buttler	master	3	buttler@...	1993-0...
B00000...	Tracy	Wang	master	4	wang@b...	1997-0...
B00000...	Barbara	Callan	master	2.5	callan@...	1995-1...
B00000...	Jack	Smith	master	3.2	smith@b...	1995-1...
B00000...	Terry	Zillman	PhD	4	zillman...	1992-0...
B00000...	Tom	Baker	master		baker@...	1997-1...
B00000...	Ben	Liu	master	3.8	liu@bu.e...	1996-0...
B00000...	Sata	Patel	master	3.9	patel@b...	1994-1...
B00000...	Art	Chang	PhD	4	chang@...	1993-0...

```
/**
 *
 * @author krishi
 */
```