MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
Database Systems Lab (CSE 2262)

Student Academic Data management

SUBMITTED BY:
Anurag Nayak (210905016) Roll no:3 Sec: A
Adruti Onam (210905190) Roll no: 34 Sec: A

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.
April 2023

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
12/05/2023

CERTIFICATE

This is to certify that the project titled **Student Academic Data management** is a record of the bonafide work done by **Anurag Nayak(Reg.No.210905016), Adruti Onam(Reg.No 2109050190)**, submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Prof. Anup Bhat B, Associate Professor, CSE Dept.**

2. **Mr. Govardhan Hegde, Assistant Professor Selection Grade, CSE Dept.**

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

In today's digital age, universities and educational institutions are increasingly relying on web-based systems to manage their administrative processes. One such process is the recording and management of student registration and grade information for courses. The task of managing this information manually can be a cumbersome and error-prone process, which is why the need for an efficient and reliable system to handle this task is crucial. In this context, this project aims to design and implement a web-based system that can record student registration and grade information for courses at a university. This system will allow for the easy and accurate recording of marks given to each student in each assignment or exam of a course, and will also compute a weighted sum of marks to get the total course marks. Moreover, the system will be designed to allow for the addition of new assignments or exams at any time, without any predefined limit. Finally, the system will support grading, enabling cutoffs to be specified for various grades. The proposed system will streamline the recording and management of student registration and grade information, making the process more efficient, accurate, and reliable. Additionally, the web-based system will enable authorized users, such as instructors and administrative staff, to access the recorded information from anywhere with an internet connection. This will provide a significant advantage over traditional paper-based systems, which often require physical access to files and records. Furthermore, the system will incorporate security features to ensure the privacy and confidentiality of student information, as well as backup and recovery mechanisms to prevent data loss. Overall, the proposed web-based system will offer a robust, flexible, and user-friendly solution for managing student registration and grade information for courses at a university.

CHAPTER 2: PROBLEM STATEMENT & OBJECTIVES

Problem Statement: Create and execute a system on the web that can maintain records of student registration and grades for I Courses offered by a University. Also enabling the storage of information on course performance, including marks achieved by each student for every assignment and exam in the course. The System should be able to compute the total course marks by assignment weights to the marks achieved and add new exams and assignments as needed. Additionally, the system should facilitate grading by allowing the definition of cut-offs for different grades.

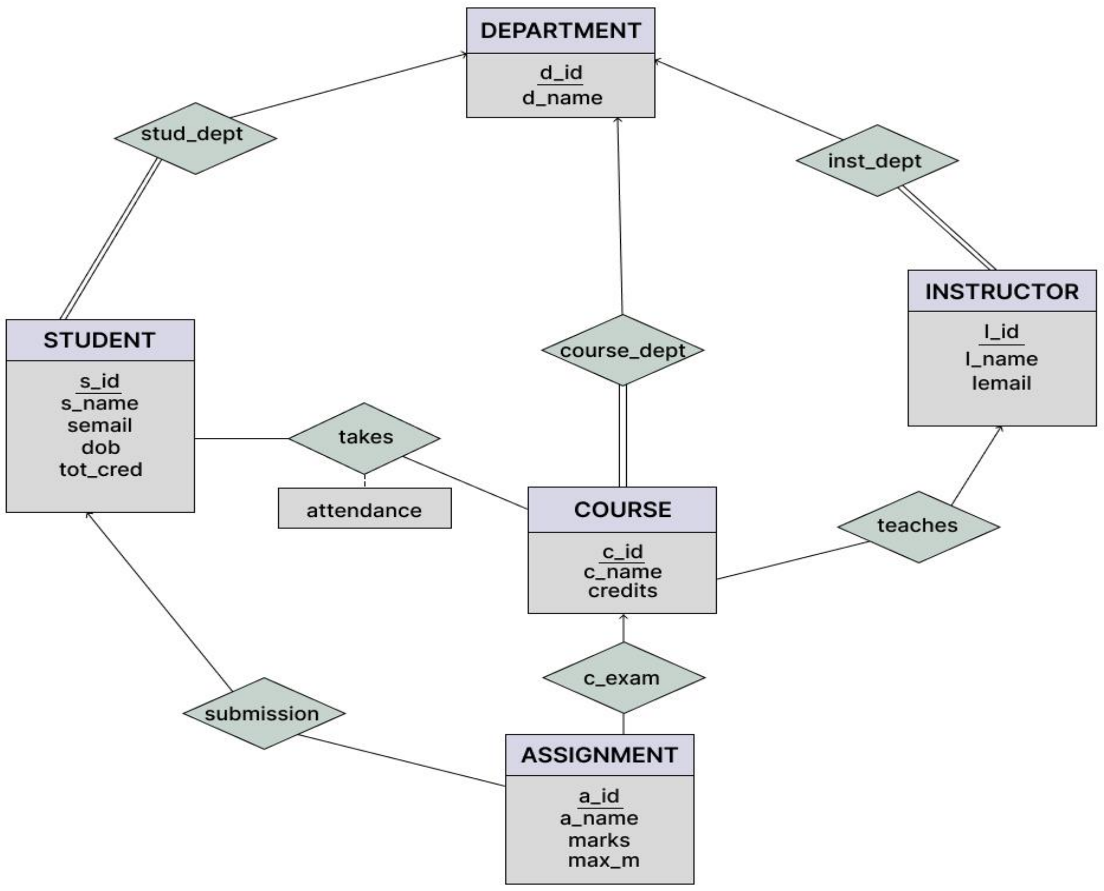
Objectives: The objective of this project is to design and implement a web-based system that allows for the efficient and accurate recording and management of student registration and grade information for courses at a university. The system will enable the recording of marks given to each student in each assignment or exam of a course, and compute a weighted sum of marks to get the total course marks. The system will also support grading, permitting cutoffs to be specified for various grades, and will be designed with security, backup, and recovery features to ensure the privacy and confidentiality of student information. In addition to the above, the objective of this project is to provide an easily accessible and user-friendly system that can be used by authorized personnel, such as instructors and administrative staff. The system will be designed to be scalable and flexible, allowing for the addition of new assignments or exams as needed, and to provide reliable and accurate information to aid decision-making processes.

CHAPTER 3: METHODOLOGY

1.Identifying the entities involved in the system: The main entities involved in the system are to be student, department, instructor, course, assignment, and grades.
2.Determining the relationships between entities: The relationships between the entities can be many-to-many, many-to-one and one-to-one. For example, a student can take multiple courses, and a course can have multiple students. Similarly, a course can have multiple assignments, and an assignment can be assigned to one course.
3.Define the attributes for each entity: For students, attributes may include student ID, name, address, email, phone number, and major. For courses, attributes may include course ID, title, instructor, and department. For assignments, attributes may include assignment ID, description, due date, and total points. For grades, attributes may include student ID, course ID, assignment ID, and grade.
4.Reducing the above Entity-Relationship to Normalised form: Using first normal form we replaced the name attribute by first name and last name and eliminated the name attribute to make it atomic. In the given schema, each table has a single candidate key and no non-trivial functional dependencies, making it already in 3NF. Hence, further normalization to BCNF is not required.
5.Creating a database schema: Using the identified entities and their attributes, we created a database schema that includes tables for each entity and columns for each attribute. Make sure to define primary and foreign keys to establish relationships between tables.
6.Implementation of the schema in Oracle: Using Oracle's SQL language to create the tables and define their relationships.
7.Populating the database: Insert data into the tables to represent students, courses, assignments, instructors and departments.
8.Creating queries to retrieve data: Use SQL to create queries that allow users to retrieve data from the database. For example, a query might retrieve a student's grades for a specific course, or a query might display all assignments for a particular course.

9.Creating PL/SQL commands and Triggers: Using PL/SQL and trigger commands to implement complex queries and displaying the necessary data from the given table and managing errors, exceptions and logs.
10.Testing the system: Test the system thoroughly to ensure that it performs as expected and that all data is accurate and complete. Maintenance of the system: Regularly backup the database and perform maintenance tasks to ensure the system continues to function properly over time.

ER Diagram:



<u>NORMALISATION</u>

The given database consists of six tables: Department, Student, Instructor, Course, Takes, and Assignment. Using first normal form we replaced the name attribute by first name and last name and eliminated the name attribute to make it atomic. To reduce the given database using Boyce Codd Normal Form (BCNF):
These tables are already in BCNF as the only candidate key
is their primary key, and there are no non-trivial functional dependencies between their attributes.

1. Department Table: d_id as Primary Key
2. Student Table: s_id as Primary Key
3. Instructor Table: I_id as Primary Key
4. Takes Table: s_id and c_id Primary Key
5. Course Table: c_id as Primary Key
6. Assignment Table: a_id as Primary Key

BCNF is a higher level of normalization, which ensures that a table has no non-trivial functional dependencies of attributes on anything other than a candidate key. If there are any such dependencies, the table is split into multiple tables to achieve BCNF.

In the given schema, each table has a single candidate key and no non-trivial functional dependencies, making it already in 3NF. Hence, further normalization to BCNF is not required.

Therefore the normalised tables are:
department (<u>d_id</u>,d_name)
student (<u>s_id</u>,s_fname, s_lname,semail,dob,tot_cred,d_id(foreign key))
instructor(I_id(primary key),I_name,Iemail,d_id(foreign key))
course(<u>c_id</u>,c_name,credits,d_id(foreign key),I_id(foreign key))
takes(<u>s_id</u>(foreign key),<u>c_id</u>(foreign key),attendance)
assignment(<u>a_id</u>,a_name,marks,c_id(foreign key),s_id(foreign key),max_m)

DML/DDL COMMANDS

1. DDL commands:

```
CREATE TABLE department (
  d_id NUMBER(5) PRIMARY KEY,
  d_name VARCHAR2(50)
);

CREATE TABLE student (
  s_id NUMBER(5) PRIMARY KEY,
  s_name VARCHAR2(50),
  semail VARCHAR2(50),
  dob DATE,
  tot_cred NUMBER(5),
  d_id NUMBER(5),
  FOREIGN KEY (d_id) REFERENCES department(d_id)
);

CREATE TABLE instructor (
  i_id NUMBER(5) PRIMARY KEY,
  i_name VARCHAR2(50),
  iemail VARCHAR2(50),
  d_id NUMBER(5),
  FOREIGN KEY (d_id) REFERENCES department(d_id)
);

CREATE TABLE course (
  c_id NUMBER(5) PRIMARY KEY,
  c_name VARCHAR2(50),
  credits NUMBER(5),
  d_id NUMBER(5),
  i_id NUMBER(5),
  FOREIGN KEY (d_id) REFERENCES department(d_id),
  FOREIGN KEY (i_id) REFERENCES instructor(i_id)
);

CREATE TABLE takes (
  s_id NUMBER(5),
  c_id NUMBER(5),
  attendance NUMBER(5),
  primary key(s_id,c_id),
  FOREIGN KEY (s_id) REFERENCES student(s_id) on delete cascade,
  FOREIGN KEY (c_id) REFERENCES course(c_id)
);

CREATE TABLE assignment (
  a_id NUMBER(5) PRIMARY KEY,
  a_name VARCHAR2(50),
  marks NUMBER(5),
  c_id NUMBER(5),
  s_id NUMBER(5),
  max_m NUMBER(5),
  FOREIGN KEY (c_id) REFERENCES course(c_id),
  FOREIGN KEY (s_id) REFERENCES student(s_id) on delete cascade
);
```

2. Complex Queries:

```
--1.Retrieve the name and email of all students who have not attended a course
in department 1:

SELECT s_name, semail
FROM student
WHERE s_id NOT IN (
    SELECT s_id FROM takes
    JOIN course ON takes.c_id = course.c_id
    WHERE course.d_id =1
);

--2.Retrieve the name of all students who have attended a course taught by all
instructors:

SELECT s_name
FROM student
WHERE NOT EXISTS (
    SELECT I_id FROM instructor
    WHERE NOT EXISTS (
        SELECT course.c_id FROM course
        JOIN takes ON course.c_id = takes.c_id
        WHERE takes.s_id = student.s_id AND course.I_id = instructor.I_id
    )
);

--3.Retrieve the name and email of all students who have taken courses in all
departments:

SELECT s_name, semail
FROM student
WHERE NOT EXISTS (
    SELECT d_id FROM department
    WHERE NOT EXISTS (
        SELECT course.c_id FROM course
        JOIN takes ON course.c_id = takes.c_id
        WHERE takes.s_id = student.s_id AND course.d_id = department.d_id
    )
);

--4.Retrieve the name and email of all instructors who have taught at least
--one course with a total number of credits greater than 4 in department 1,
--sorted by the total number of credits:

SELECT instructor.I_name, instructor.Iemail, SUM(course.credits) as
total_credits
FROM instructor
JOIN course ON instructor.I_id = course.I_id
WHERE course.d_id = 1
GROUP BY instructor.I_id, instructor.I_name, instructor.Iemail
HAVING SUM(course.credits) > 4
ORDER BY total_credits DESC;

--5.List the names of all instructors who have taught at least one course outside
their department:

SELECT DISTINCT i.I_name
FROM instructor i
JOIN course c ON i.I_id = c.I_id
WHERE c.d_id <> i.d_id;

--6.Query for finding the number of assignments given by each student
```

```
SELECT s.s_name, COUNT(a.a_id) AS num_assignments
FROM student s
LEFT JOIN assignment a ON s.s_id = a.s_id
GROUP BY s.s_id, s.s_name;

--7.Query to find the department with the highest total number of credits
offered:

SELECT d.d_name, SUM(c.credits) as total_credits
FROM department d
JOIN course c ON d.d_id = c.d_id
GROUP BY d.d_name
ORDER BY total_credits DESC
FETCH FIRST 1 ROWS ONLY;
```

3. <u>PL/SQL commands:</u>

```
--1. PL/SQL block that deletes the records of students with attendance less
than 75 from the takes table:
set serveroutput on
DECLARE
  v_attendance takes.attendance%TYPE;
BEGIN
  FOR rec IN (SELECT * FROM takes)
  LOOP
    SELECT attendance INTO v_attendance
    FROM takes
    WHERE s_id = rec.s_id AND c_id = rec.c_id;

    IF v_attendance < 75 THEN
      DELETE FROM takes
      WHERE s_id = rec.s_id AND c_id = rec.c_id;
    END IF;
  END LOOP;
  COMMIT;
END;
/
```

```
-- 2. PL/SQL procedure that takes a student ID as a parameter and displays all
--the details related to that student across all tables:
set serveroutput on
CREATE OR REPLACE PROCEDURE
GET_STUDENT_DETAILS(p_s_id IN NUMBER) AS
   v_s_id student.s_id%TYPE;
   v_s_name student.s_name%TYPE;
   v_semail student.semail%TYPE;
   v_dob student.dob%TYPE;
   v_tot_cred student.tot_cred%TYPE;
   v_d_name department.d_name%TYPE;
   v_c_id course.c_id%TYPE;
   v_c_name course.c_name%TYPE;
   v_credits course.credits%TYPE;
   v_i_name instructor.i_name%TYPE;
   v_attendance takes.attendance%TYPE;
   v_a_name assignment.a_name%TYPE;
   v_marks assignment.marks%TYPE;
   v_max_m assignment.max_m%TYPE;

   CURSOR C IS
      SELECT s.s_id, s.s_name, s.semail, s.dob, s.tot_cred, d.d_name, t.c_id,
c.c_name,
            c.credits, i.i_name, t.attendance, a.a_name, a.marks, a.max_m
      FROM student s
      JOIN takes t ON s.s_id = t.s_id
      JOIN course c ON t.c_id = c.c_id
      JOIN instructor i ON c.i_id = i.i_id
      JOIN department d ON s.d_id = d.d_id
      LEFT JOIN assignment a ON t.c_id = a.c_id AND t.s_id = a.s_id AND
a.a_name = 'Assignment 1'
      WHERE s.s_id = p_s_id;
BEGIN
   OPEN C;
   LOOP
      FETCH C INTO v_s_id, v_s_name, v_semail, v_dob, v_tot_cred,
v_d_name, v_c_id, v_c_name,
               v_credits, v_i_name, v_attendance, v_a_name, v_marks,
v_max_m;
      EXIT WHEN C%NOTFOUND;
```

```
        DBMS_OUTPUT.PUT_LINE('Student ID: ' || v_s_id);
        DBMS_OUTPUT.PUT_LINE('Student Name: ' || v_s_name);
        DBMS_OUTPUT.PUT_LINE('Student Email: ' || v_semail);
        DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || v_dob);
        DBMS_OUTPUT.PUT_LINE('Total Credits: ' || v_tot_cred);
        DBMS_OUTPUT.PUT_LINE('Department Name: ' || v_d_name);
        DBMS_OUTPUT.PUT_LINE('Course ID: ' || v_c_id);
        DBMS_OUTPUT.PUT_LINE('Course Name: ' || v_c_name);
        DBMS_OUTPUT.PUT_LINE('Course Credits: ' || v_credits);
        DBMS_OUTPUT.PUT_LINE('Instructor Name: ' || v_i_name);
        DBMS_OUTPUT.PUT_LINE('Attendance: ' || v_attendance);
        DBMS_OUTPUT.PUT_LINE('Assignment Name: ' || v_a_name);
        DBMS_OUTPUT.PUT_LINE('Marks: ' || v_marks);
        DBMS_OUTPUT.PUT_LINE('Maximum Marks: ' || v_max_m);
        DBMS_OUTPUT.PUT_LINE('---------------------------------------');
    END LOOP;
    CLOSE C;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No data found for the given student ID');
END;
/



BEGIN
    GET_STUDENT_DETAILS('&s_id');
END;
/

--3. PL/SQL command to find the grading of a given student based on their
total marks:
set serveroutput on
DECLARE
  v_s_id    student.s_id%TYPE := '&s_id';
  v_total_marks NUMBER := 0;
  v_grading VARCHAR2(10);
BEGIN

  SELECT SUM(a.marks * c.credits / s.tot_cred) INTO v_total_marks
  FROM assignment a
  JOIN course c ON a.c_id = c.c_id
  JOIN student s ON a.s_id = s.s_id
  WHERE a.s_id = v_s_id;


  IF v_total_marks >= 90 THEN
    v_grading := 'A';
  ELSIF v_total_marks >= 80 THEN
    v_grading := 'B';
  ELSIF v_total_marks >= 70 THEN
    v_grading := 'C';
  ELSIF v_total_marks >= 60 THEN
    v_grading := 'D';
  ELSE
    v_grading := 'F';
  END IF;

  DBMS_OUTPUT.PUT_LINE('Grading for student ' || v_s_id || ': ' ||
v_grading);
END;
/

--4.PL/SQL for finding the total course marks using the computation
of(weighted) sum of marks using procedures
set serveroutput on

CREATE OR REPLACE PROCEDURE calculate_total_marks (p_s_id  IN
NUMBER, p_total_marks OUT NUMBER)
AS
   v_total_marks NUMBER := 0;
   v_c_id course.c_id%TYPE;
   v_marks assignment.marks%TYPE;
   v_credits course.credits%TYPE;
   v_total_cred student.tot_cred%TYPE;

   CURSOR C IS
      SELECT c.c_id, a.marks, c.credits, s.tot_cred
      FROM takes t
      JOIN course c ON t.c_id = c.c_id
      JOIN assignment a ON t.s_id = a.s_id AND t.c_id = a.c_id
      JOIN student s ON t.s_id = s.s_id
```

```
        WHERE t.s_id = p_s_id AND t.attendance >= 75;
BEGIN
    OPEN C;
    LOOP
        FETCH C INTO v_c_id, v_marks, v_credits, v_total_cred;
        EXIT WHEN C%NOTFOUND;

        v_total_marks := v_total_marks + (v_marks * (v_credits / v_total_cred));
    END LOOP;
    CLOSE C;

    p_total_marks := v_total_marks;
END;
/


DECLARE
    v_total_marks NUMBER;
BEGIN
    calculate_total_marks('&s_id', v_total_marks);
    DBMS_OUTPUT.PUT_LINE('Total marks: ' || v_total_marks);
END;
/
```

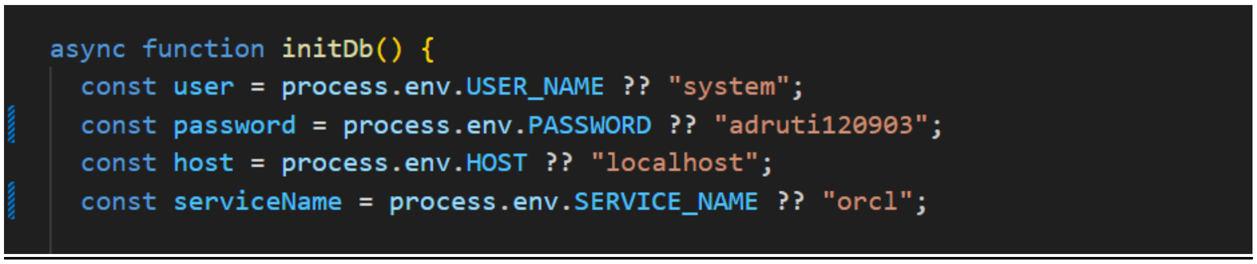
4. Triggers

```
--1. trigger that raises an error when a student tries to take up more than 6
courses:

CREATE OR REPLACE TRIGGER max_courses_trigger
BEFORE INSERT ON takes
FOR EACH ROW
DECLARE
    v_num_courses NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_num_courses
    FROM takes
    WHERE s_id = :new.s_id;
    IF v_num_courses >= 6 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Maximum limit of 6 courses
exceeded for this student');
    END IF;
END;
/

--2. trigger that shows an error message when attempting to insert an
assignment with a score greater than 100 and a max_m value greater than 100:
set serveroutput on
CREATE OR REPLACE TRIGGER check_assignment_score
BEFORE INSERT ON assignment
FOR EACH ROW
BEGIN
  IF :NEW.marks > 100 OR :NEW.max_m > 100 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Error: Assignment score or
max_m value cannot exceed 100');
  END IF;
END;
/

--3. trigger that captures student changes and inserts them into the
student_change table:

set serveroutput on
drop table student_change;
CREATE table student_change(op varchar(10) ,entry_date date, s_id
NUMBER(5), s_name VARCHAR2(50), semail VARCHAR2(50),
dob DATE, tot_cred NUMBER(5), d_id NUMBER(5));


CREATE OR REPLACE TRIGGER student_change_trigger
BEFORE INSERT OR UPDATE OR DELETE ON student
FOR EACH ROW
DECLARE
    v_op VARCHAR2(10);
BEGIN
    IF INSERTING THEN
        v_op := 'INSERT';
        INSERT INTO student_change(op,entry_date, s_id, s_name, semail, dob,
tot_cred, d_id)
```

```
    VALUES(v_op, SYSDATE, :NEW.s_id, :NEW.s_name, :NEW.semail,
:NEW.dob, :NEW.tot_cred, :NEW.d_id);
    ELSIF UPDATING THEN
        v_op := 'UPDATE';
        INSERT INTO student_change(op,entry_date, s_id, s_name, semail, dob,
tot_cred, d_id)
    VALUES(v_op, SYSDATE, :OLD.s_id, :OLD.s_name, :OLD.semail,
:OLD.dob, :OLD.tot_cred, :OLD.d_id);
    ELSIF DELETING THEN
        v_op := 'DELETE';
        INSERT INTO student_change(op,entry_date, s_id, s_name, semail, dob,
tot_cred, d_id)
    VALUES(v_op, SYSDATE, :OLD.s_id, :OLD.s_name, :OLD.semail,
:OLD.dob, :OLD.tot_cred, :OLD.d_id);
    END IF;
END;
/


INSERT INTO student (s_id, s_name, semail, dob, tot_cred, d_id) VALUES
(11, 'gg', 'gg@email.com', TO_DATE('1990-10-10', 'yyyy-mm-dd'), 28, 5);

--4. trigger that will insert a new row into the takes table
--with c_id=1 whenever a new student is added to the student table:

CREATE OR REPLACE TRIGGER add_default_course_to_student
AFTER INSERT ON student
FOR EACH ROW
BEGIN
  INSERT INTO takes (s_id, c_id, attendance)
  VALUES (:NEW.s_id, 1, NULL);
END;
/
```

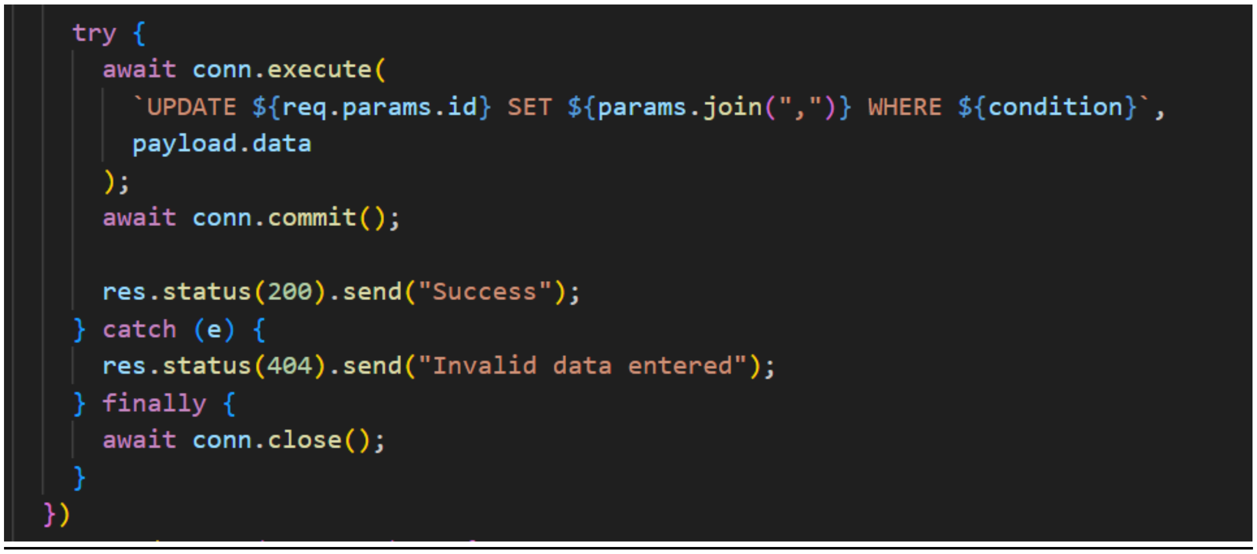
CHAPTER 4: RESULTS & SNAPSHOTS

1. DB Connection



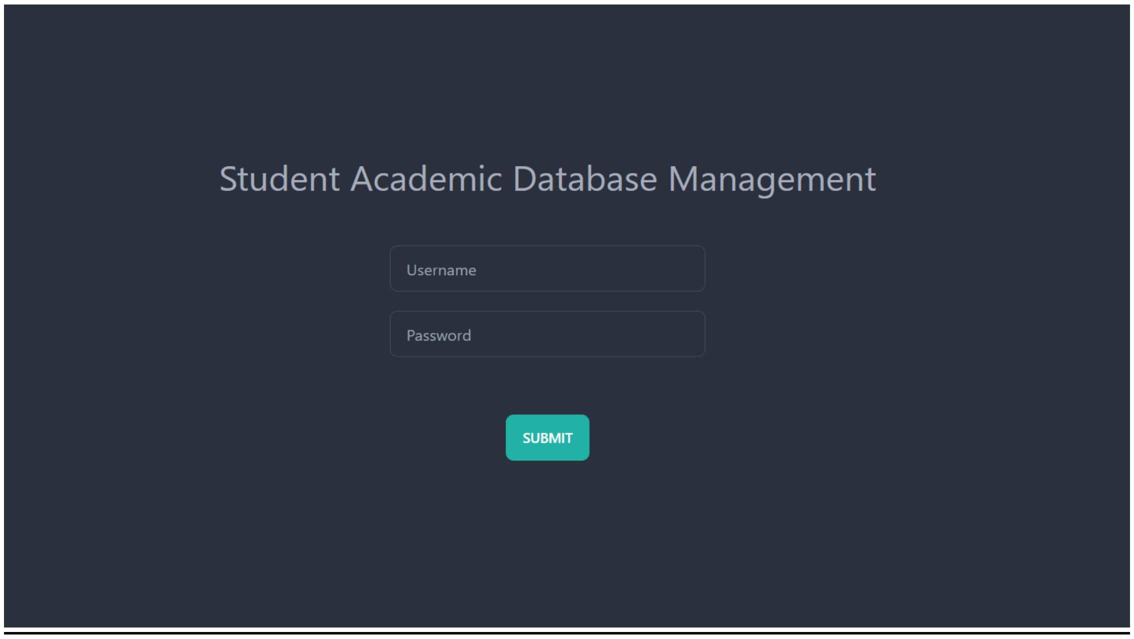
```
async function initDb() {
  const user = process.env.USER_NAME ?? "system";
  const password = process.env.PASSWORD ?? "adruti120903";
  const host = process.env.HOST ?? "localhost";
  const serviceName = process.env.SERVICE_NAME ?? "orcl";
```

2. Insertion/Updating/Deletion in Website

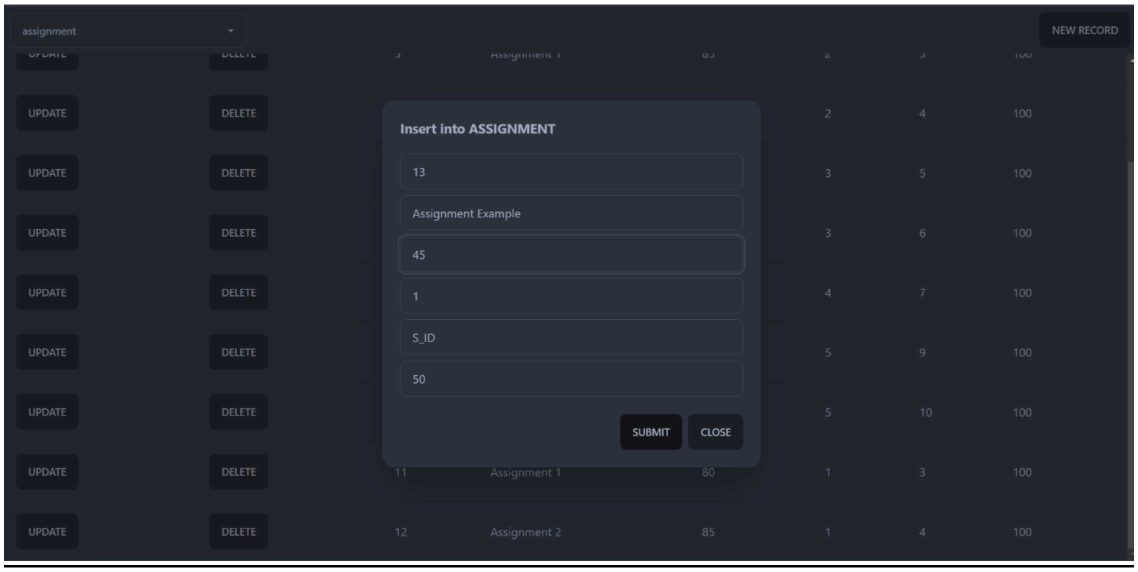


```
  try {
    await conn.execute(
      `UPDATE ${req.params.id} SET ${params.join(",")} WHERE ${condition}`,
      payload.data
    );
    await conn.commit();

    res.status(200).send("Success");
  } catch (e) {
    res.status(404).send("Invalid data entered");
  } finally {
    await conn.close();
  }
})
```

3. Login Page for Administrator


Student Academic Database Management

4. Insertion of Assignment


Insert into ASSIGNMENT

13

Assignment Example

45

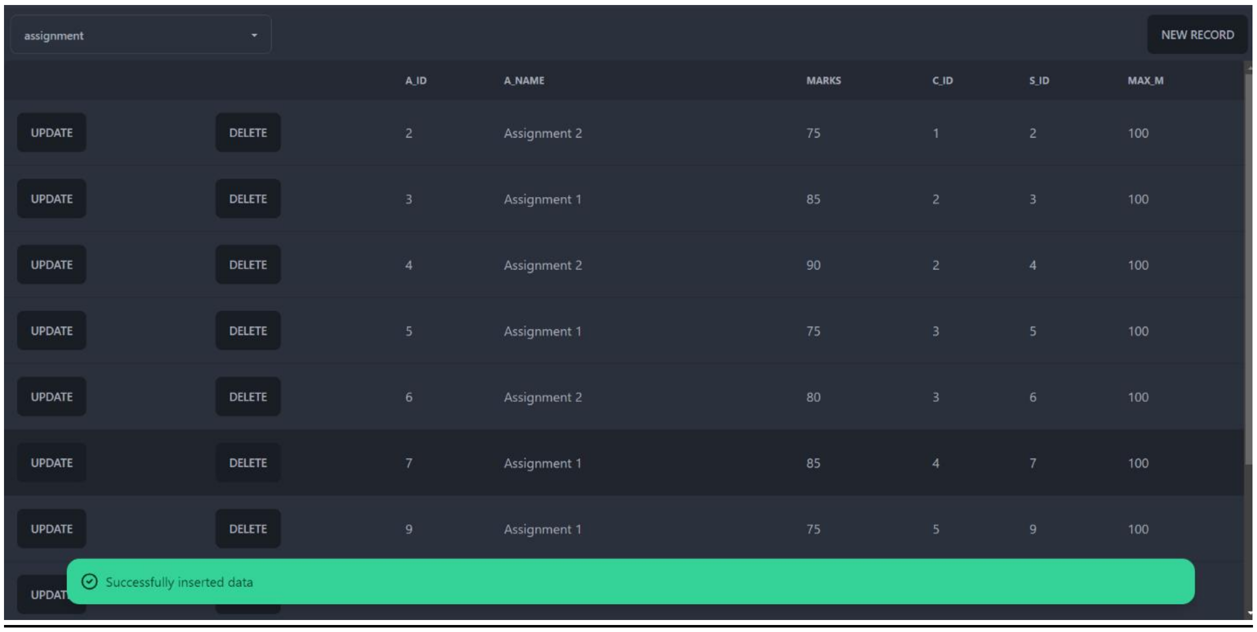
1

S_ID

50

UPDATE	DELETE	11	Assignment 1	80	1	3	100
UPDATE	DELETE	12	Assignment 2	85	1	4	100

5. Pop Up-Message being displayed on successful insertion



		A_ID	A_NAME	MARKS	C_ID	S_ID	MAX_M
UPDATE	DELETE	2	Assignment 2	75	1	2	100
UPDATE	DELETE	3	Assignment 1	85	2	3	100
UPDATE	DELETE	4	Assignment 2	90	2	4	100
UPDATE	DELETE	5	Assignment 1	75	3	5	100
UPDATE	DELETE	6	Assignment 2	80	3	6	100
UPDATE	DELETE	7	Assignment 1	85	4	7	100
UPDATE	DELETE	9	Assignment 1	75	5	9	100

Successfully inserted data

CHAPTER 5: CONCLUSIONS

In conclusion, the implementation of both the Student Academic Record System and the system for recording student registration and grade information have proven to be valuable tools for managing academic records and supporting student success. Both systems offer features such as easy registration of courses, updating and managing student information, and flexible tracking of student academic progress. The Student Academic Record System has the added benefit of providing easy access to grades for each course a student is enrolled in, while the system allows for the addition of assignments and exams as needed. Additionally, the grading feature in the web-based system allows for the specification of cutoffs for various grades, simplifying the process of calculating and assigning final grades for each student. These systems have helped to reduce paperwork and manual data entry, save time, and provide accurate and up-to-date information for both students and faculty members. Overall, both systems are valuable resources for universities and colleges to improve their administrative processes and provide better services to their students.

CHATER 6: LIMITATIONS & FUTURE WORK

Limitations:

Data security: The system will need to handle sensitive information, such as student grades and personal information, which will need to be protected from unauthorized access.
Scalability: The system will need to handle a large number of users and courses, which may require scaling up the infrastructure.
Performance: The system will need to respond quickly to user requests, which may require optimizing queries and database design.
User interface design: The system will need to be easy to use and navigate, which may require designing a user-friendly interface.
Data accuracy: The system will need to ensure that data is entered correctly and consistently, which may require implementing data validation and error checking.

Future Work:

Integration with learning management systems: The web system could be integrated with popular learning management systems (LMS) such as Canvas or Blackboard, allowing students and faculty to access all course information and resources from a single platform.
Analytics and reporting: The system could be enhanced with analytics and reporting features, allowing administrators and faculty to track student performance and identify areas for improvement.
Mobile optimization: The system could be optimized for mobile devices, allowing students and faculty to access course information and resources from their smartphones or tablets.
Automated grading: The system could be enhanced with features to automatically grade assignments and exams, reducing the workload on faculty and increasing consistency in grading.

CHAPTER 7: REFERENCES

- https://www.geeksforgeeks.org/
- https://stackoverflow.com
- www.tutorialspoint.com
- https://www.javatpoint.com/oracle-procedure
- “DATABASE SYSTEM CONCEPTS SIXTH EDITION “ by Abraham Silberschatz Yale University, Henry F. Korth Lehigh University, S. Sudarshan Indian Institute of Technology,Bombay.