MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

DATABASE SYSTEMS MINI PROJECT

STUDENT MANAGEMENT SYSTEM

PREPARED BY

SOUMYA SAHU
210905196
KUSHALA SARADA A V
210905189

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal 09/05/2023

CERTIFICATE

This is to certify that the project titled STUDENT MANAGEMENT SYSTEM is a record of the bonafide work done by SOUMYA SAHU (Reg No.210905196) andKUSHALA SARADA A V(Reg No.210905189) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology(BTech.) 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.Dr. Anup Bhat B., Assistant Professor, CSE Dept.

2.MR. Govardhan Hegde ,Assistant Professor Selection Grade,CSE Dept.

CONTENTS

1. INTRODUCTION

1.1 ABSTRACT

This project aims to design and implement a system that permits recording of course
performance information—
specifically, the marks given to each student in each assignment or exam of a course
computation of a (weighted) sum of marks to get the total course marks.
The number of assignments/exams which will not be predefined; that is, more assignments/exams can be added at any time.
This system should also support grading, permitting cutoffs to be specified for various grades.

2. PROBLEM STATEMENT

This project would require mySql to create a database which will help students to keep a tab on the following details
Their marks
The assignments that are due

Their grades and the cutoffs.

3. METHODOLOGY

3.1 SCHEMA REDUCTION

• The first step is to analyse the existing schema to identify the tables that can be reduced or eliminated.
• Normalise the schema to remove redundant data and ensure that each table is in the most appropriate normal form.
• Construct an ER diagram and reduce the entities and entity relationships obtained from the ER diagram to a schema.

STRONG ENTITIES:-
-Students
-Course
-Assignments

WEAK ENTITIES:-
-Grade

RELATIONSHIP SETS:-
-Enrolls In:- One student can enrol in many courses and one course can be taken by many students. Hence,it's a many to many relationship.
-Submits:-One student can submit many assignments and one assignment can be submitted by many students.Hence,it's a many to many relationship.
-Has-One course has many assignments and one assignment can be associated with many courses.Hence,it's a many to many relationship.

REDUCED SCHEMA:-

1.Students(Student_id, phone_number, name, email);
2.Course(Course_id, course_name, Department_name, Credits);
3.Assignments(Assignment_id, total_marks, Assignment_name, marks);
4.Enrolls In(student_id, course_id);
5.Submits(student_id, Assignment_id);
6.Has(Assignment_ID, Course_id);
7.Grade(upperLimit, LowerLimit, Pass/fail, GradeName, Assignment_id);

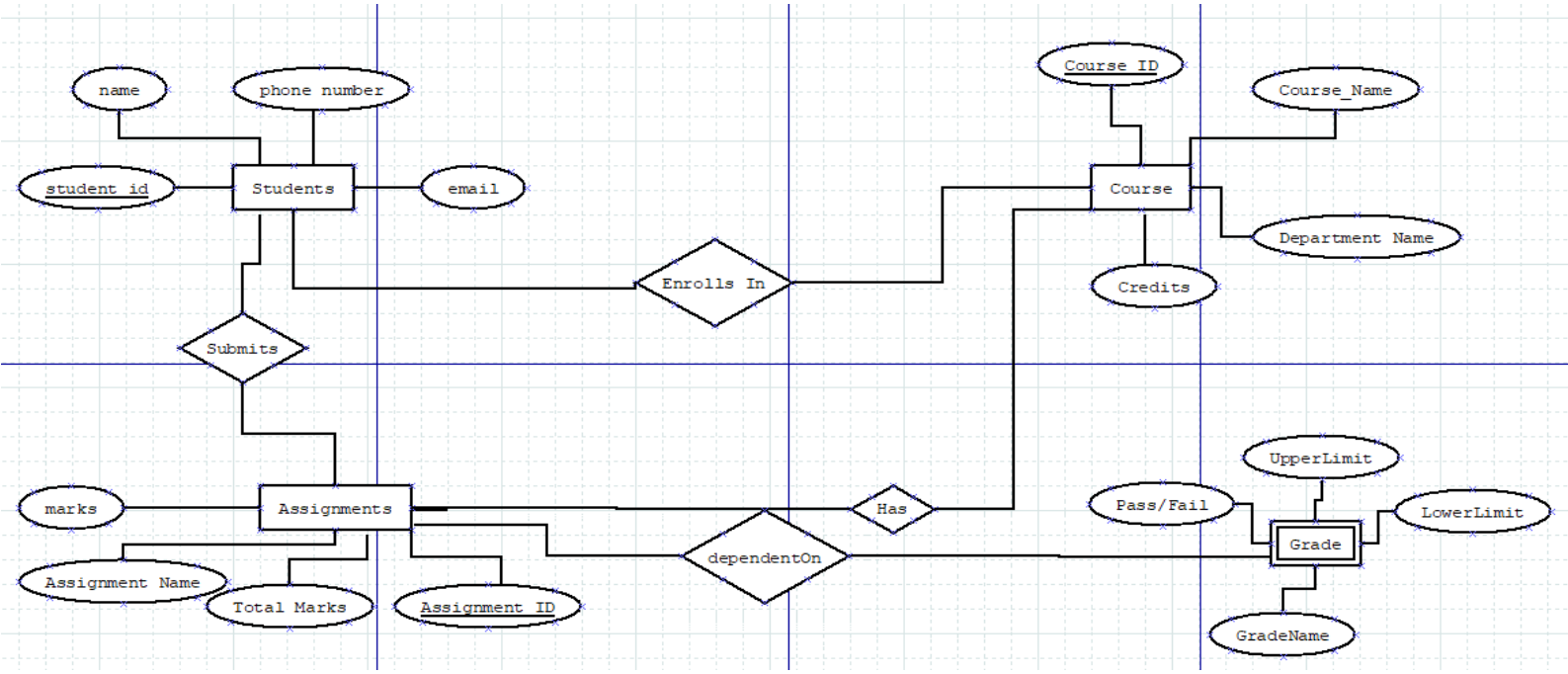
FUNCTIONAL DEPENDENCIES:-
Student_id → name,email,phoneNumber;
course_id→course_name, Department_name,credits;
Assignment_id→Assignment_name;
course_id→→ Assignment_id;

NORMALIZATION

-All tables that are present are in first normal form i.e it is atomic in nature.
-We have decomposed the Assignment table into Assignment(assignment_id,assignment_name) and marks_assignment(assignment_id,marks,student_id) using boyce codd normal form decomposition using the functional dependency Assignment_id→Assignment_name .

-We have decomposed the Course table into course(Course_id, course_name, Department_name, Credits) and course_assignment(assignment_id,course_id) using 4th normal form decomposition with the functional dependency course_id→→ Assignment_id.

ER DIAGRAM-


name
phone number
student_id
Students
email
Course_ID
Course_Name
Course
Department Name
Credits
Enrolls In
Submits
marks
Assignments
Has
dependentOn
Assignment Name
Total Marks
Assignment_ID
UpperLimit
Pass/Fail
LowerLimit
Grade
GradeName

CREATE TABLE AND INSERT TABLE COMMANDS

```
CREATE TABLE Students (
  student_id INT PRIMARY KEY,
  name VARCHAR2(50) NOT NULL,
  email VARCHAR2(50) UNIQUE,
  phone_number VARCHAR2(20)
);


CREATE TABLE course(
  course_id varchar(20) PRIMARY KEY,
  title VARCHAR2(50) NOT NULL,
  credit INT,
  dept_name varchar(50)
);

CREATE TABLE failed_students(
            student_id INT,
            course_id varchar(20),
            CONSTRAINT fk_XYZ
    FOREIGN KEY (student_id)
    REFERENCES Students(student_id),
    FOREIGN KEY (course_id)
    REFERENCES course(course_id)
);

CREATE TABLE assignment(
  assignment_id INT primary key,
  assignment_name varchar(50),
  total_marks int
);
```

```
CREATE TABLE marks_assignment(
  assignment_id INT,
  marks INT,
  student_id INT,
  CONSTRAINT fk_student1
    FOREIGN KEY (student_id)
    REFERENCES Students(student_id),
  CONSTRAINT fk_assignment4
    FOREIGN KEY (assignment_id)
    REFERENCES assignment(assignment_id)
);


CREATE TABLE grade(
  assignment_id INT,
  grade_name varchar(20) NOT NULL,
  fail_pass VARCHAR(10),
  upperLimit number,
  lowerLimit number,
  CONSTRAINT fk_assignment3
    FOREIGN KEY (assignment_id)
    REFERENCES assignment(assignment_id)
);

CREATE TABLE course_student(
  student_id INT,
  course_id varchar(20),
  CONSTRAINT fk_student2
    FOREIGN KEY (student_id)
    REFERENCES Students(student_id),
 CONSTRAINT fk_course1
    FOREIGN KEY (course_id)
```

```
    REFERENCES course(course_id)
);


CREATE TABLE course_assignment(
  assignment_id INT,
  course_id varchar(20),
  CONSTRAINT fk_course2
    FOREIGN KEY (course_id)
    REFERENCES course(course_id),
  CONSTRAINT fk_assignment2
    FOREIGN KEY (assignment_id)
    REFERENCES assignment(assignment_id)
);


insert into Students values(200,'Souyma
Sahu','sahu@gmail.com',1234567890);
insert into Students values(202,'Kushala
Aravapalli','kushala@gmail.com',3214567890);
insert into Students values(204,'Anu
Agarwal','anuu@gmail.com',1234569870);
insert into Students values(206,'gaurav
gupta','gaurav@gmail.com',1236547890);
insert into Students
values(208,'Sujeet','suji@gmail.com',4321567890);
insert into Students
values(210,'Mermaid','maid@gmail.com',1234587609);
insert into Students values(212,'yogi
sharma','sharma@gmail.com',1234567089);
insert into Students values(214,'Aditya
sharma','adii@gmail.com',4365888089);
```

```
insert into course values('cs-101','dmbs',3,'comp-Sci');
insert into course values('cs-102','ES',4,'comp-Sci');
insert into course values('cs-103','DAA',4,'comp-Sci');
insert into course values('cs-104','COA',4,'comp-Sci');
insert into course values('cs-105','DSA',4,'comp-Sci');
insert into course values('ece-101','EMW',3,'electronics');
insert into course values('ece-102','LIC',4,'electronics');
insert into course values('ece-103','M4',3,'electronics');
insert into course values('bio-101','DSD',3,'biology');
insert into course values('bio-102','DSW',4,'biology');

insert into course_student values(202,'cs-101');
insert into course_student values(202,'cs-102');
insert into course_student values(202,'cs-103');
insert into course_student values(204,'cs-101');
insert into course_student values(206,'cs-101');
insert into course_student values(208,'cs-101');
insert into course_student values(210,'cs-102');
insert into course_student values(204,'cs-103');
insert into course_student values(206,'ece-101');
insert into course_student values(208,'ece-102');
insert into course_student values(208,'ece-103');
insert into course_student values(210,'bio-101');
insert into course_student values(212,'bio-102');
insert into course_student values(214,'bio-102');
insert into course_student values(216,'cs-101');


insert into assignment values(1,'Bio_fisac1',5);
insert into assignment values(2,'Bio_fisac2',5);
insert into assignment values(3,'cse_misac1',15);
```

```
insert into assignment values(4,'cse_fisac1',50);
insert into assignment values(5,'cse_fisac2',50);
insert into assignment values(6,'ece_fisac1',5);
insert into assignment values(7,'ece_fisac2',5);
insert into assignment values(8,'Bio_misac3',15);
insert into assignment values(9,'Bio_endsem',50);
insert into assignment values(10,'cse1_endsem',50);

insert into course_assignment values(1,'bio-101');
insert into course_assignment values(2,'bio-102');
insert into course_assignment values(3,'cs-101');
insert into course_assignment values(4,'cs-102');
insert into course_assignment values(5,'cs-103');
insert into course_assignment values(6,'ece-101');
insert into course_assignment values(7,'ece-101');
insert into course_assignment values(8,'bio-102');
insert into course_assignment values(9,'bio-102');
insert into course_assignment values(10,'cs-101');



insert into marks_assignment values(1,2,210);
insert into marks_assignment values(2,5,212);
insert into marks_assignment values(2,4,214);
insert into marks_assignment values(3,15,204);
insert into marks_assignment values(3,11,208);
insert into marks_assignment values(3,12,202);
insert into marks_assignment values(3,7,216);
insert into marks_assignment values(4,41,204);
insert into marks_assignment values(5,20,204);
insert into marks_assignment values(10,35,208);
insert into marks_assignment values(10,40,206);
insert into marks_assignment values(10,26,204);
```

```
insert into marks_assignment values(8,14,214);
insert into marks_assignment values(8,7,212);


insert into grade values(1,'A','P',5,5);
insert into grade values(1,'B','P',4,3);
insert into grade values(1,'C','P',2,1);
insert into grade values(1,'F','F',1,0);

insert into grade values(2,'A','P',5,5);
insert into grade values(2,'B','P',4,3);
insert into grade values(2,'C','P',2,1);
insert into grade values(2,'F','F',1,0);

insert into grade values(3,'A','P',15,12);
insert into grade values(3,'B','P',11,8);
insert into grade values(3,'C','P',7,4);
insert into grade values(3,'F','F',3,0);

insert into grade values(4,'A','P',50,40);
insert into grade values(4,'B','P',39,35);
insert into grade values(4,'C','P',34,30);
insert into grade values(4,'D','P',29,25);
insert into grade values(4,'E','P',24,18);
insert into grade values(4,'F','F',17,0);

insert into grade values(5,'A','P',50,40);
insert into grade values(5,'B','P',39,35);
insert into grade values(5,'C','P',34,30);
insert into grade values(5,'D','P',29,25);
insert into grade values(5,'E','P',24,18);
insert into grade values(5,'F','F',17,0);
```

```
insert into grade values(6,'A','P',5,5);
insert into grade values(6,'B','P',4,3);
insert into grade values(6,'C','P',2,1);
insert into grade values(6,'F','F',1,0);

insert into grade values(7,'A','P',5,5);
insert into grade values(7,'B','P',4,3);
insert into grade values(7,'C','P',2,1);
insert into grade values(7,'F','F',1,0);
```

QUERIES

q1.list all the students who registered in a particular course.

```
set serveroutput on
DECLARE
  c_id varchar(20);
  CURSOR student_courses IS
    SELECT DISTINCT student_id, name, email, phone_number
    FROM Students natural join course_student where c_id =
course_id;
BEGIN
  c_id:='&course_id';
  FOR sc IN student_courses LOOP
    DBMS_OUTPUT.PUT_LINE(sc.student_id || ', ' || sc.name || ', ' ||
sc.email || ', ' || sc.phone_number);
  END LOOP;
END;
/[1]
```

[1]

```
SQL> @ "C:\Users\Sahus\OneDrive\Desktop\PROJECT\q2.sql"
SP2-0310: unable to open file "C:\Users\Sahus\OneDrive\Desktop\PROJ
ECT\q2.sql"
SQL> @ "C:\Users\Sahus\OneDrive\Desktop\PROJECT\q2.txt"
Enter value for course_id: cs-101
old   7:   c_id:='&course_id';
new   7:   c_id:='cs-101';
202, Kushala Aravapalli, kushala@gmail.com, 3214567890
204, Anu Agarwal, anuu@gmail.com, 1234569870
206, gaurav gupta, gaurav@gmail.com, 1236547890
208, Sujeet, suji@gmail.com, 4321567890
216, ishan sharma, ishan@gmail.com, 43548089
```

q2. retrieve all assignment id,assignment name given by a particular student

```
set serveroutput on
DECLARE
  xstudent_id Students.student_id%TYPE := 204;
BEGIN
  FOR assignment_rec IN (
    SELECT a.assignment_id, a.assignment_name
    FROM assignment a
    JOIN marks_assignment ma ON a.assignment_id =
ma.assignment_id
    WHERE ma.student_id = xstudent_id
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('Assignment ID: ' ||
assignment_rec.assignment_id || ', Name: ' ||
assignment_rec.assignment_name);
  END LOOP;
END;
/
```

```
SQL> @ "C:\Users\Sahus\OneDrive\Desktop\PROJECT\q3.txt"
Enter value for student_id: 204
old   2:   xstudent_id Students.student_id%TYPE := '&student_id';
new   2:   xstudent_id Students.student_id%TYPE := '204';
Assignment ID: 3, Name: cse_misac1
Assignment ID: 4, Name: cse_fisac1
Assignment ID: 5, Name: cse_fisac2
Assignment ID: 10, Name: cse1_endsem

PL/SQL procedure successfully completed.
```

q3. PL/SQL query to find the number of assignments done for each course

```
DECLARE
  CURSOR c_courses IS SELECT course_id FROM course;
  ccourse_id course.course_id%TYPE;
  num_assignments INTEGER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Course ID | Number of
Assignments');
  FOR course_rec IN c_courses LOOP
    ccourse_id := course_rec.course_id;
    SELECT COUNT(*) INTO num_assignments FROM
course_assignment WHERE course_id = ccourse_id;
    DBMS_OUTPUT.PUT_LINE(ccourse_id || ' | ' ||
num_assignments);
  END LOOP;
END;
/
```

```
SQL> @ "C:\Users\Sahus\OneDrive\Desktop\PROJECT\q4.txt"
Course ID | Number of Assignments
cs-101 | 2
cs-102 | 1
cs-103 | 1
cs-104 | 0
cs-105 | 0
ece-101 | 2
ece-102 | 0
ece-103 | 0
bio-101 | 1
bio-102 | 3
```

q4. find top 3 students for given course_id.

```
SET SERVEROUTPUT ON
DECLARE
  max_students NUMBER(3) := 3;
  course_id_to_check course.course_id%TYPE := '&course_id';
BEGIN
            FOR student_rec IN (
              SELECT s.student_id, s.name, SUM(ma.marks) AS total_marks
              FROM Students s
              JOIN marks_assignment ma ON s.student_id = ma.student_id
              JOIN assignment a ON ma.assignment_id = a.assignment_id
              JOIN course_assignment ca ON a.assignment_id =
          ca.assignment_id
              JOIN course c ON ca.course_id = c.course_id
              WHERE c.course_id = course_id_to_check
              GROUP BY s.student_id, s.name
              ORDER BY total_marks DESC
            )
            LOOP
              DBMS_OUTPUT.PUT_LINE('ID: ' || student_rec.student_id || '
          Name: ' || student_rec.name || ', Total Marks: ' ||
          student_rec.total_marks);
              max_students := max_students - 1;
              IF (max_students < 1) THEN
                EXIT;
```

```
    END IF;
  END LOOP;
END;
/
```

```
SQL> @ "C:\Users\Sahus\OneDrive\Desktop\PROJECT\q5.txt"
Enter value for course_id: cs-101
old   3:   course_id_to_check course.course_id%TYPE := '&course_id'
;
new   3:   course_id_to_check course.course_id%TYPE := 'cs-101';
ID: 208 Name: Sujeet, Total Marks: 46
ID: 204 Name: Anu Agarwal, Total Marks: 41
ID: 206 Name: gaurav gupta, Total Marks: 40

PL/SQL procedure successfully completed.
```

q5.Display Student ID who scored highest for a given assignment?

```
SET SERVEROUTPUT ON
DECLARE
  l_assignment_id assignment.assignment_id%TYPE :=
'&assignment_id';
  l_student_id marks_assignment.student_id%TYPE;
  l_highest_marks marks_assignment.marks%TYPE;

  CURSOR c_students IS
    SELECT student_id, marks
    FROM marks_assignment
    WHERE assignment_id = l_assignment_id
    ORDER BY marks DESC;

BEGIN
  OPEN c_students;
  FETCH c_students INTO l_student_id, l_highest_marks;

  IF c_students%FOUND THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Student ID: ' || l_student_id || ',
Highest Marks: ' || l_highest_marks);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No student found for the given
assignment.');
```

input-
Enter value for assignment_id: 10

output-
Student ID: 206, Highest Marks: 40

q6. Display all courses available using pl/sql

```
declare
  c_title course.title%TYPE;
  c_id course.course_id%TYPE;
begin
  for c in (select course_id, title from course)
  loop
    c_id := c.course_id;
    c_title := c.title;
    DBMS_OUTPUT.PUT_LINE('Course ID: ' || c_id || ', Title: ' ||
c_title);
  end loop;
END;
/
```

output-
Course ID: cs-101, Title: DBMS
Course ID: cs-102, Title: ES
Course ID: cs-103, Title: DAA
Course ID: cs-104, Title: COA
Course ID: cs-105, Title: DSA
Course ID: ece-101, Title: EMW

Course ID: ece-102, Title: LIC
Course ID: ece-103, Title: M4
Course ID: bio-101, Title: DSD
Course ID: bio-102, Title: DSW

q7. trigger to check the marks entered can't be zero and can't be more than the total marks.

```
set serveroutput on;
CREATE OR REPLACE TRIGGER trg_check_marks_assignment
BEFORE INSERT OR UPDATE ON marks_assignment
FOR EACH ROW
DECLARE
  c_total_marks assignment.total_marks%TYPE;
BEGIN
  SELECT total_marks
  INTO c_total_marks
  FROM assignment
  WHERE assignment_id = :NEW.assignment_id;

  IF :NEW.marks <0 OR :NEW.marks > c_total_marks THEN
     RAISE_APPLICATION_ERROR(-20001, 'Invalid marks. Marks should be greater than 0 and not exceed the total marks of the assignment.');
  END IF;
END;
/
```

```
SQL> insert into marks_assignment values(10,-6,204);
insert into marks_assignment values(10,-6,204)
            *
ERROR at line 1:
ORA-20001: Invalid marks. Marks should be greater than 0 and not exceed the
total marks of the assignment.
ORA-06512: at "SYSTEM.TRG_CHECK_MARKS_ASSIGNMENT", line 10
ORA-04088: error during execution of trigger
'SYSTEM.TRG_CHECK_MARKS_ASSIGNMENT'
```

q8.retrieve the list most popular course_id with highest intake?

```
declare
  max_count number;
  popular_course varchar(20);
begin
  select max(count) into max_count from (
    select count(*) AS count from course_student
    group by course_id
  );

  select course_id into popular_course from (
    select course_id, count(*) count from course_student
    group by course_id
    having count(*) = max_count
  );

  DBMS_OUTPUT.PUT_LINE('Most Popular Course: ' ||
popular_course || ' (Intake: ' || max_count || ')');
END;
/
```

```
Most Popular Course: cs-101 (Intake: 5)

PL/SQL procedure successfully completed.
```

q9. retrieves the average marks of a specific course for a particular assignment.

```
SET SERVEROUTPUT ON
DECLARE
  l_course_id course.course_id%TYPE := '&course_id';
  l_assignment_id assignment.assignment_id%TYPE :=
'&assignment_id';
```

```
  l_average_marks NUMBER;
BEGIN
  SELECT AVG(marks)
  INTO l_average_marks
  FROM marks_assignment ma
  JOIN assignment a ON ma.assignment_id = a.assignment_id
  JOIN course_assignment ca ON a.assignment_id =
ca.assignment_id
  JOIN course c ON ca.course_id = c.course_id
  WHERE c.course_id = l_course_id
  AND a.assignment_id = l_assignment_id;

  DBMS_OUTPUT.PUT_LINE('Average marks for Course ' ||
l_course_id || ', Assignment ' || l_assignment_id || ': ' ||
l_average_marks);
END;
/
```

Output-
Average marks for Course cs-101, Assignment 3: 11.25

10.Delete all those students whose grade is 'F' for a particular course.

```
SET SERVEROUTPUT ON
DECLARE
  course_id_in VARCHAR2(20) := 'cs-101';
  grade_cutoff_a NUMBER := '&grade_A'; -- Minimum percentage
for grade A
  grade_cutoff_b NUMBER := '&grade_B'; -- Minimum percentage
for grade B
  grade_cutoff_c NUMBER := '&grade_C'; -- Minimum percentage
for grade C
```

```
  grade_cutoff_d NUMBER := '&grade_D'; -- Minimum percentage
for grade D
  grade_cutoff_e NUMBER := '&grade_E'; -- Minimum percentage
for grade E
  percentage_marks NUMBER;
  grade VARCHAR2(1);

BEGIN
  FOR student_rec IN (
    SELECT student_id, SUM(marks) AS total_marks_obtained,
           SUM(total_marks) AS total_marks_possible
    FROM course_student natural join marks_assignment natural
join assignment natural join course_assignment
    WHERE course_id = course_id_in
    GROUP BY student_id
  ) LOOP
        -- Calculate the percentage of marks obtained by the student
        percentage_marks := (student_rec.total_marks_obtained /
student_rec.total_marks_possible) * 100;

        -- Determine the grade based on the percentage of marks
        IF percentage_marks >= grade_cutoff_a THEN
          grade := 'A';
        ELSIF percentage_marks >= grade_cutoff_b THEN
          grade := 'B';
        ELSIF percentage_marks >= grade_cutoff_c THEN
          grade := 'C';
        ELSIF percentage_marks >= grade_cutoff_d THEN
          grade := 'D';
        ELSIF percentage_marks >= grade_cutoff_e THEN
          grade := 'E';
        ELSE
          -- Delete student records from marks_assignment and
course_student
```

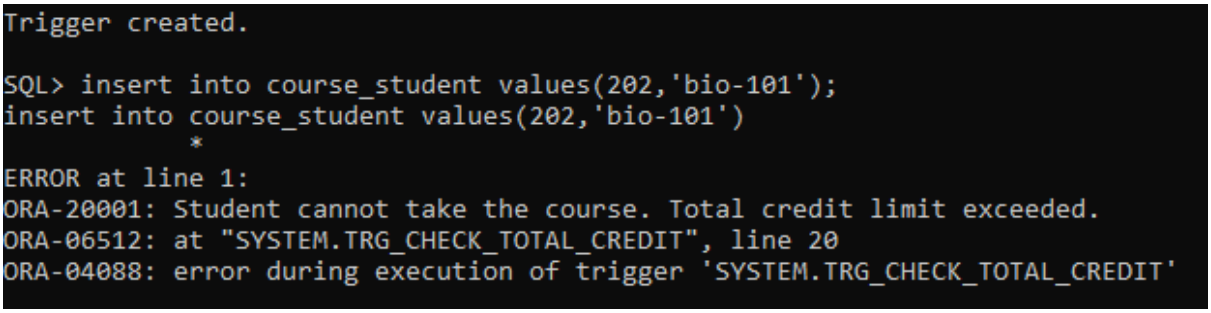
```
        DELETE FROM marks_assignment WHERE student_id =
student_rec.student_id;
        DELETE FROM course_student WHERE student_id =
student_rec.student_id;

        -- Insert student_id into new table failed_students
        INSERT INTO failed_students VALUES
(student_rec.student_id,course_id_in);

        DBMS_OUTPUT.PUT_LINE('Student with ID ' ||
student_rec.student_id || ' has been deleted due to low
percentage.');

        -- Skip to the next student record
        CONTINUE;
      END IF;

      -- Display the student details and grade
      DBMS_OUTPUT.PUT_LINE('Student ID: ' ||
student_rec.student_id);
      DBMS_OUTPUT.PUT_LINE('Percentage: ' ||
percentage_marks || '%');
      DBMS_OUTPUT.PUT_LINE('Grade: ' || grade);
      DBMS_OUTPUT.PUT_LINE('-------------------------');
  END LOOP;
END;
/
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\q13.txt
Enter value for grade_a: 40
old   3:   grade_cutoff_a NUMBER := '&grade_A'; -- Minimum percentage for grade A
new   3:   grade_cutoff_a NUMBER := '40'; -- Minimum percentage for grade A
Enter value for grade_b: 30
old   4:   grade_cutoff_b NUMBER := '&grade_B'; -- Minimum percentage for grade B
new   4:   grade_cutoff_b NUMBER := '30'; -- Minimum percentage for grade B
Enter value for grade_c: 20
old   5:   grade_cutoff_c NUMBER := '&grade_C'; -- Minimum percentage for grade C
new   5:   grade_cutoff_c NUMBER := '20'; -- Minimum percentage for grade C
Enter value for grade_d: 10
old   6:   grade_cutoff_d NUMBER := '&grade_D'; -- Minimum percentage for grade D
new   6:   grade_cutoff_d NUMBER := '10'; -- Minimum percentage for grade D
Enter value for grade_e: 5
old   7:   grade_cutoff_e NUMBER := '&grade_E'; -- Minimum percentage for grade E
new   7:   grade_cutoff_e NUMBER := '5'; -- Minimum percentage for grade E
Student ID: 202
Percentage: 80%
Grade: A
-------------------------
Student ID: 204
Percentage: 63.07692307692307692307692307692307692308%
Grade: A
-------------------------
Student ID: 206
Percentage: 80%
Grade: A
-------------------------
Student ID: 208
Percentage: 70.76923076923076923076923076923076923077%
Grade: A
-------------------------
Student ID: 216
Percentage: 46.66666666666666666666666666666666666667%
Grade: A
-------------------------

PL/SQL procedure successfully completed.
```

Q11)Trigger for inserting row in course_student if total credit of the student is credit<=14

```
CREATE OR REPLACE TRIGGER trg_check_total_credit
BEFORE INSERT ON course_student
FOR EACH ROW
DECLARE
  total_credit NUMBER := 0;
  course_credit NUMBER :=0;
BEGIN
  -- Calculate the total credit of the student
  SELECT COALESCE(SUM(c.credit), 0)
  INTO total_credit
  FROM course_student cs
  JOIN course c ON cs.course_id = c.course_id
```

```
  WHERE cs.student_id = :NEW.student_id;

  -- Get the credit for the new course
  SELECT credit
  INTO course_credit
  FROM course
  WHERE course_id = :NEW.course_id;

  -- Check if the total credit exceeds 15
  IF total_credit + course_credit > 14 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Student cannot take
the course. Total credit limit exceeded.');
  END IF;
END;
/
```



Q12)write a trigger for name of the student is a valid name containing only alphabet

```
create or replace trigger check_name
before insert or update on Students
for each row
begin
  IF LENGTH(TRIM(TRANSLATE(:NEW.name,
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
Z', ' '))) > 0 THEN
    RAISE_APPLICATION_ERROR(-20100, 'Name must contain
only alphabets');
  END IF;
```

END;
/

```
Trigger created.

SQL> insert into Students values(250,'Aditya s.12','afdsa@gmail.com',436588802342);
insert into Students values(250,'Aditya s.12','afdsa@gmail.com',436588802342)
            *
ERROR at line 1:
ORA-20100: Name must contain only alphabets
ORA-06512: at "SYSTEM.CHECK_NAME", line 3
ORA-04088: error during execution of trigger 'SYSTEM.CHECK_NAME'
```

Q13)write a row trigger that records date and time for a student when they register for the course and the student can't drop the course before 2 months.

```
drop trigger trg_insert_course_student;
drop trigger trg_delete_course_student;
drop table student_joining_date;

create table student_joining_date (
  student_id int,
  course_id varchar(20),
  joining_date date
);

create or replace trigger trg_insert_course_student
after insert on course_student
for each row
declare
  joining_date date;
begin
  select SYSDATE into joining_date from dual;
  insert into student_joining_date (student_id, course_id,
joining_date)
  values(:NEW.student_id, :NEW.course_id, joining_date);
END;
```

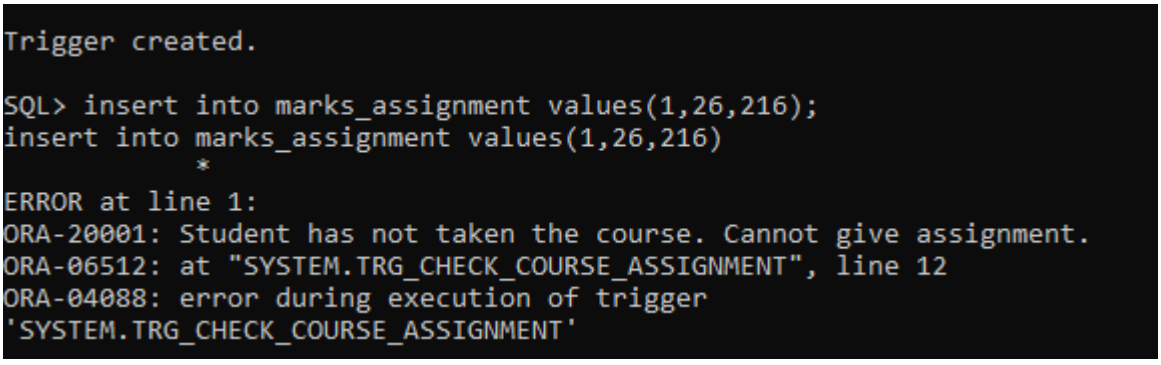
```
/

create or replace trigger trg_delete_course_student
before delete on course_student
for each row
declare

  joining_date date;
begin
  select joining_date into joining_date from student_joining_date
  where student_id = :OLD.student_id AND course_id =
:OLD.course_id;
  IF joining_date + INTERVAL '2' MONTH >= SYSDATE then
    RAISE_APPLICATION_ERROR(-20100, 'Cannot drop course
within 2 months of joining');
  END IF;
END;
/


--insert into course_student values(202,'bio-102');
--delete from course_student where student_id=202 and
course_id='bio-102';
```

```
SQL> insert into course_student values(202,'bio-102');

1 row created.

SQL> delete from course_student where student_id=202 and course_id='bio-102';
delete from course_student where student_id=202 and course_id='bio-102'
            *
ERROR at line 1:
ORA-20100: Cannot drop course within 2 months of joining
ORA-06512: at "SYSTEM.TRG_DELETE_COURSE_STUDENT", line 8
ORA-04088: error during execution of trigger 'SYSTEM.TRG_DELETE_COURSE_STUDENT'
```

Q14)trigger to check the student who hasn't take the course, can't given any assignment from that particular course

```
SET SERVEROUTPUT ON
```

```
CREATE OR REPLACE TRIGGER trg_check_course_assignment
BEFORE INSERT ON marks_assignment
FOR EACH ROW
DECLARE
  course_exists NUMBER := 0;
BEGIN
  -- Check if the course exists in course_student table for the student
  SELECT COUNT(*) INTO course_exists
  FROM course_student
  WHERE student_id = :NEW.student_id
    AND course_id = (SELECT course_id FROM course_assignment
WHERE assignment_id = :NEW.assignment_id);

  -- If course doesn't exist, raise an application error
  IF course_exists = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Student has not taken the
course. Cannot give assignment.');
  END IF;
END;
/
```



q15.trigger for the same assignment can't be given twice.

```
create or replace trigger trg_twice_assignment
before insert on marks_assignment
for each row
declare
```

```
  c integer;
begin
  select count(*) into c
  from marks_assignment
  where student_id = :NEW.student_id
    and assignment_id = :NEW.assignment_id;

  if c > 0 then
    RAISE_APPLICATION_ERROR(-20001, 'Student has already
taken this assignment.');
  END IF;
END;
/
```

```
Trigger created.

SQL> insert into marks_assignment values(5,20,204);
insert into marks_assignment values(5,20,204)
            *
ERROR at line 1:
ORA-20001: Student has already taken this assignment.
ORA-06512: at "SYSTEM.TRG_MARKS_ASSIGNMENT", line 10
ORA-04088: error during execution of trigger 'SYSTEM.TRG_MARKS_ASSIGNMENT'
```

q16.PL/SQL code that adds an assignment for a given course_id,prompts the user to enter the assignment name ,the marks for the assignment and grade cutoff?

```
SET SERVEROUTPUT ON
DECLARE
  c_id course.course_id%TYPE := '&course_id';
  ass_id assignment.assignment_id%TYPE;
  ass_name assignment.assignment_name%TYPE :=
'&Assignment_name';
  tot_marks assignment.total_marks%TYPE := '&Total_Marks';
  grade_au number := '&grade_au';
  grade_bu number := '&grade_bu';
  grade_cu number := '&grade_cu';
  grade_du number := '&grade_du';
  grade_eu number := '&grade_eu';
```

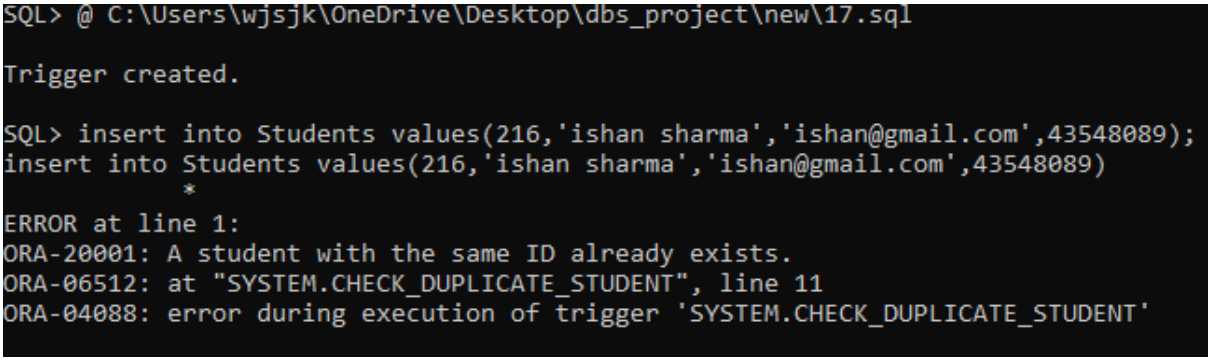
```
  grade_fu number := '&grade_fu';
  grade_al number := '&grade_al';
  grade_bl number := '&grade_bl';
  grade_cl number := '&grade_cl';
  grade_dl number := '&grade_dl';
  grade_el number := '&grade_el';
  grade_fl number := '&grade_fl';
  g_ap varchar(2) := 'P';
  g_ff varchar(2) := 'F';
BEGIN

  -- Retrieve the next available assignment_id for the given
course_id
  SELECT MAX(assignment_id) + 1 INTO ass_id
  FROM assignment;

  -- Insert the new assignment into the assignment table
  INSERT INTO assignment VALUES (ass_id,
ass_name,tot_marks);
  INSERT INTO course_assignment VALUES (ass_id,c_id);

  -- Insert the grade table values for the assignment
  INSERT INTO grade values (ass_id,'A',g_ap, grade_au, grade_al);
  INSERT INTO grade values (ass_id,'B',g_ap, grade_bu, grade_bl);
  INSERT INTO grade values(ass_id,'C',g_ap, grade_cu, grade_cl);
  INSERT INTO grade values (ass_id,'D',g_ap, grade_du, grade_dl);
  INSERT INTO grade values(ass_id,'E',g_ap, grade_eu, grade_el);
  INSERT INTO grade values(ass_id,'F',g_ff, grade_fu, grade_fl);

  DBMS_OUTPUT.PUT_LINE('Assignment added successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\16.sql
Enter value for course_id: cs-101
old   2:   c_id course.course_id%TYPE := '&course_id';
new   2:   c_id course.course_id%TYPE := 'cs-101';
Enter value for assignment_name: cse-misac2
old   4:   ass_name assignment.assignment_name%TYPE := '&Assignment_name';
new   4:   ass_name assignment.assignment_name%TYPE := 'cse-misac2';
Enter value for total_marks: 15
old   5:   tot_marks assignment.total_marks%TYPE := '&Total_Marks';
new   5:   tot_marks assignment.total_marks%TYPE := '15';
Enter value for grade_au: 15
old   6:   grade_au number := '&grade_au';
new   6:   grade_au number := '15';
Enter value for grade_bu: 11
old   7:   grade_bu number := '&grade_bu';
new   7:   grade_bu number := '11';
Enter value for grade_cu: 8
old   8:   grade_cu number := '&grade_cu';
new   8:   grade_cu number := '8';
Enter value for grade_du: 5
old   9:   grade_du number := '&grade_du';
new   9:   grade_du number := '5';
Enter value for grade_eu: 3
old  10:   grade_eu number := '&grade_eu';
new  10:   grade_eu number := '3';
Enter value for grade_fu: 0
old  11:   grade_fu number := '&grade_fu';
new  11:   grade_fu number := '0';
Enter value for grade_al: 12
old  12:   grade_al number := '&grade_al';
new  12:   grade_al number := '12';
Enter value for grade_bl: 9
old  13:   grade_bl number := '&grade_bl';
new  13:   grade_bl number := '9';
Enter value for grade_cl: 6
old  14:   grade_cl number := '&grade_cl';
new  14:   grade_cl number := '6';
Enter value for grade_dl: 4
old  15:   grade_dl number := '&grade_dl';
new  15:   grade_dl number := '4';
Enter value for grade_el: 1
old  16:   grade_el number := '&grade_el';
new  16:   grade_el number := '1';
Enter value for grade_fl: 0
old  17:   grade_fl number := '&grade_fl';
new  17:   grade_fl number := '0';
Assignment added successfully.

PL/SQL procedure successfully completed.
```

q17. To enforce the constraint that the same student cannot enrol again in the student table.

set serveroutput on
CREATE OR REPLACE TRIGGER check_duplicate_student
BEFORE INSERT ON students
FOR EACH ROW

```
DECLARE
   duplicate_count NUMBER;
BEGIN
   -- Check if the new student already exists in the table
   SELECT COUNT(*) INTO duplicate_count
   FROM studentS
   WHERE student_id = :NEW.student_id;

   -- If a duplicate student is found, raise an exception
   IF duplicate_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'A student with the
same ID already exists.');
   END IF;
END;
/
```



q18. To display the details of a course based on the entered course_id

```
SELECT *
FROM course
WHERE course_id = 'cs-101';
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\18.sql

COURSE_ID            TITLE
-------------------- --------------------------------------------------
    CREDIT DEPT_NAME
---------- --------------------------------------------------
cs-101               dmbs
         3 comp-Sci
```

q19. display the assignments that are due for a particular student.

```
SELECT a.assignment_id, a.assignment_name
FROM assignment a
JOIN course_assignment ca ON a.assignment_id =
ca.assignment_id
JOIN course_student cs ON ca.course_id = cs.course_id
LEFT JOIN marks_assignment ma ON a.assignment_id =
ma.assignment_id AND cs.student_id = ma.student_id
WHERE cs.student_id = 202
  AND ma.assignment_id IS NULL;
```

```
ASSIGNMENT_ID ASSIGNMENT_NAME
------------- --------------------------------------------------

           10 cse1_endsem
            4 cse_fisac1
            5 cse_fisac2
```

q20.To display the names of assignments for a specific course_id

```
SELECT a.assignment_name
FROM assignment a,course_assignment ca,course c
WHERE a.assignment_id = ca.assignment_id and c.course_id =
'cs-101' and ca.course_id = c.course_id;
```

```
ASSIGNMENT_NAME
--------------------------------------------------
cse_misac1
cse1_endsem
3
cse-misac2
```

q21.To retrieve all assignments given by a particular student with a known student_id and for a given course_id

```
SELECT a.assignment_id, a.assignment_name
FROM assignment a,course_assignment ca,course_student
cs,marks_assignment ma
WHERE a.assignment_id = ca.assignment_id and ca.course_id =
cs.course_id and a.assignment_id = ma.assignment_id and
cs.student_id = ma.student_id and cs.student_id = 204 and
ca.course_id = 'cs-101';
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\21.sql

ASSIGNMENT_ID ASSIGNMENT_NAME
------------- --------------------------------------------------
            3 cse_misac1
           10 cse1_endsem
```

Q22.get a list of courses enrolled by a particular student.

```
SELECT c.title, c.credit, c.dept_name
FROM course c
JOIN course_student cs ON c.course_id = cs.course_id
WHERE cs.student_id = 202;
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\22.sql

TITLE                                              CREDIT
-------------------------------------------------- ----------
DEPT_NAME
--------------------------------------------------
dmbs                                                    3
comp-Sci

ES                                                      4
comp-Sci

DAA                                                     4
comp-Sci


TITLE                                              CREDIT
-------------------------------------------------- ----------
DEPT_NAME
--------------------------------------------------
DSW                                                     4
biology
```

q23)list of all courses that have at least one student who has obtained A grade in all assignments.

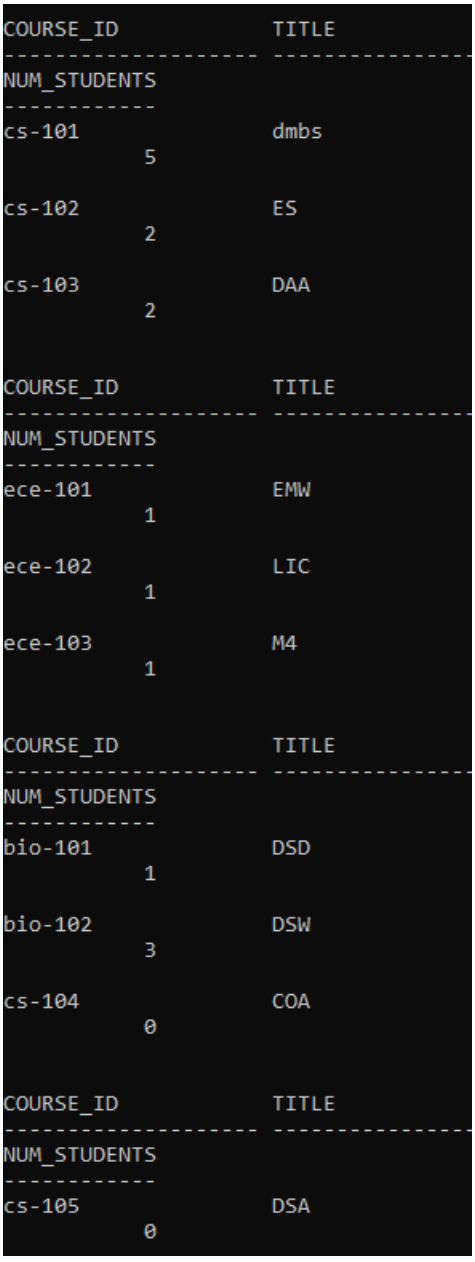
```
SELECT cs.course_id, c.title
FROM course_student cs
JOIN course c ON cs.course_id = c.course_id
WHERE cs.course_id NOT IN (
    SELECT cs.course_id
    FROM course_student cs
    JOIN marks_assignment ma ON cs.student_id = ma.student_id
    JOIN grade g ON ma.assignment_id = g.assignment_id
    WHERE g.grade_name <> 'A'
)
GROUP BY cs.course_id, c.title;
```

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\23.sql

COURSE_ID            TITLE
-------------------- --------------------------------------------------
ece-101              EMW
```

q24)Get the list of all courses along with the number of students enrolled in each course:

```
SELECT c.course_id, c.title, COUNT(cs.student_id) num_students
FROM course c
```

LEFT JOIN course_student cs ON c.course_id = cs.course_id
GROUP BY c.course_id, c.title;



```
COURSE_ID           TITLE
------------------- ---------------
NUM_STUDENTS
------------
cs-101              dmbs
          5

cs-102              ES
          2

cs-103              DAA
          2


COURSE_ID           TITLE
------------------- ---------------
NUM_STUDENTS
------------
ece-101             EMW
          1

ece-102             LIC
          1

ece-103             M4
          1


COURSE_ID           TITLE
------------------- ---------------
NUM_STUDENTS
------------
bio-101             DSD
          1

bio-102             DSW
          3

cs-104              COA
          0


COURSE_ID           TITLE
------------------- ---------------
NUM_STUDENTS
------------
cs-105              DSA
          0
```

q25)list of all students in a course.

SELECT s.student_id, s.name
FROM Students s
JOIN course_student cs ON s.student_id = cs.student_id
JOIN course c ON cs.course_id = c.course_id
WHERE c.title = 'dbms';

```
SQL> @ C:\Users\wjsjk\OneDrive\Desktop\dbs_project\new\25.sql

STUDENT_ID NAME
---------- --------------------------------------------------
       202 Kushala Aravapalli
       204 Anu Agarwal
       206 gaurav gupta
       208 Sujeet
       216 ishan sharma
```

5.CONCLUSION

In conclusion, the Student Database Management System is an essential tool for managing and tracking the academic performance of students. With its ability to record and compute marks, as well as support grading, the system provides a centralised database that enables teachers and administrators to monitor student progress effectively. Furthermore, the system's flexibility in accommodating new assignments/exams and cutoffs for different grades makes it a valuable asset for educational institutions. Overall, the Student Database Management System streamlines the process of managing student academic data and helps ensure that students receive the support and resources they need to succeed.

Goals achieved by the STUDENT DATABASEmanagement project:

• The Student Database Management System is a valuable tool for recording and organising course performance information.
• The system allows for the recording of marks for each student in every assignment or exam of a course.
• It can compute the total course marks by summing up the weighted marks of all assignments/exams.
• The system is flexible and allows for the addition of new assignments/exams at any time.
• The system supports grading and allows for the specification of cutoffs for various grades.

• Overall, the Student Database Management System is a useful tool for both students and instructors in managing and tracking academic progress.

6. LIMITATIONS AND FUTURE WORK

6.1 LIMITATIONS

Here are some potential limitations for the Student Database Management System project:

• Data security: The system may face security challenges if it is not implemented with adequate security measures in place. There may be the risk of unauthorised access or data breaches if the system is not secured properly.

• Scalability: Although the system is designed to be flexible and support the addition of new assignments/exams at any time, it may face scalability challenges if it is required to handle a large amount of data. The system may become slow or unresponsive if it is not designed to handle a high volume of data.

• Technical expertise: The system requires technical expertise to be installed and maintained, and may not be easy to use for non-technical users. This may pose a challenge for instructors or students who are not familiar with database management systems.

• Compatibility: The system may have compatibility issues with some operating systems or devices, making it inaccessible to some

users. This may limit the number of students or instructors who can use the system effectively.

• Cost: The implementation of the system may require a significant investment of time and resources. There may be licensing fees or other costs associated with acquiring and implementing the necessary hardware and software.

6.2 FUTURE WORK

• Predictive analytics: The system can be enhanced to include predictive analytics to help instructors identify students who are at risk of falling behind or failing a course, and take proactive steps to provide additional support to these students.

• Mobile access: The system can be adapted to support mobile devices, making it easier for students and instructors to access the system on the go.

• Advanced reporting: The system can be enhanced to provide more detailed and customised reporting capabilities, allowing instructors to analyse student performance data in greater depth.

• Integration with student information systems: The system can be integrated with existing student information systems, providing a more comprehensive view of student performance and progress throughout their academic career.

7. REFERENCES

W3Schools - https://www.w3schools.com/sql/
Stack Overflow - https://stackoverflow.com/questions/tagged/sql
TEXTBOOK -SILBERSCHATZ
SQL Language Quick Reference (by oracle)Oracle Database SQL Language Quick Reference