Project Report
On
Course Registration System

Team Members –

Anuraag Motiwale(asmotiwa@ncsu.edu)
Parag Nakhwa(psnakhwa@ncsu.edu)
Abhishek Singh(aksingh5@ncsu.edu)
Shriyansh Yadav(scyadav@ncsu.edu)

The report contains four sections:

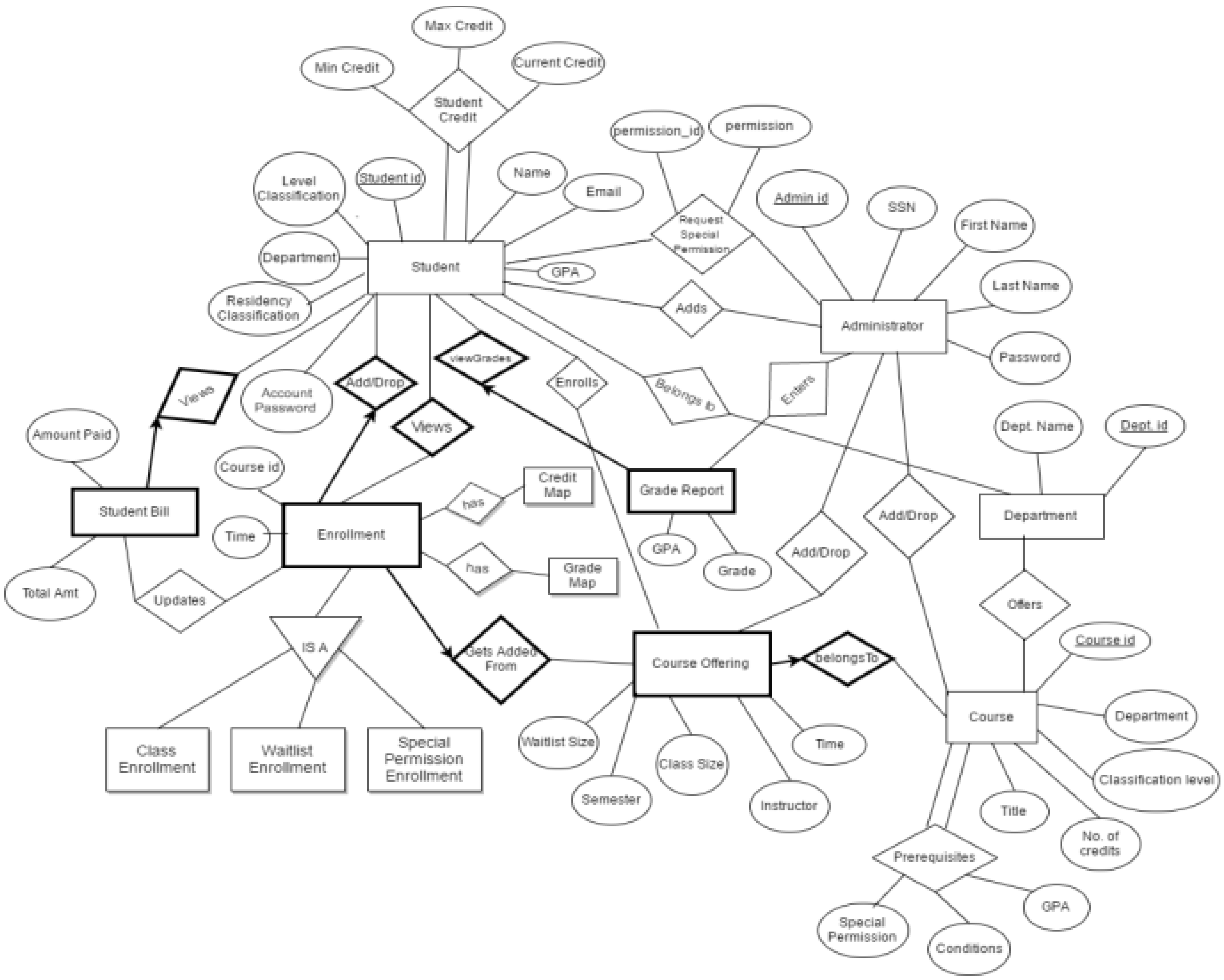
Problem Statement:

The purpose of Course Registration System is to provide a fully functional Database Management System for educational institutions to automate and handle all end-to-end relevant functionalities in a comprehensive manner.

The goal of the project is to cover all core workflows of an educational institution which includes the processes relevant to Student account and Administrator account. All the nuances related to admin privileges i.e Administrator account like adding a student, adding courses, adding a course offering, assigning grades to students, handling special permissions for enrolling in course along with those associated to Student accounts, i.e. functions like course registration, dropping courses, viewing grades, viewing schedule and viewing and paying bills are fabricated in the system in a unified manner.

Section 1. Entity Relationship Model:

1.1 ER Model Diagram



In our ER model diagram:

1. We have a IS-A relationship for Enrollment, which can be further specialized into, class enrollment, waitlist enrollment, special permission enrollment.

2. The ER diagram also shows that course offering is a weak entity which is dependent on student and course. Also, student bill is a weak entity dependent on Student. The weak entities observe total participation in the relationship with their owning entities; course_offering observes total participation in the relationship with course and student_bill observes the total participation in the relationship with student.

3. The prerequisite and student credit concern have been handled as relationships among the participating entities.

4. The listing of entity and relationships, supported by their description and list of key, represented in the ER model, can be seen with much more explanation in the relational model.

Section.2 Relational Model:

2.1 Entities and Functional Dependencies

1. **Student:**

Student(**student_id: number**, username: string, password: string, First_name: string, Last_name: string, email: string, phone: string, department: string, level_class: string, residency_class: string, GPA: number, Dateofbirth: date)

CREATE TABLE Student(Student_id INT, username VARCHAR(20) UNIQUE, password VARCHAR(50) NOT NULL, First_Name VARCHAR(20) NOT NULL, Last_Name VARCHAR(20) NOT NULL, email VARCHAR(30) NOT NULL, phone VARCHAR(10) NOT NULL, department VARCHAR(10) NOT NULL, level_class VARCHAR(20) NOT NULL, residency_class VARCHAR(20) NOT NULL, GPA DECIMAL(3,2) NOT NULL, Dateofbirth DATE NOT NULL, CONSTRAINT student_pk PRIMARY KEY(Student_id), CONSTRAINT level_c CHECK(level_class in ('Undergraduate', 'Graduate')), CONSTRAINT residency CHECK(residency_class in ('In-State', 'Out-State', 'International')));

Description: Contains all the personal information about a student.

Primary key: student_id

Functional Dependency:
student_id -> username, password, First_name, Last_name, email, phone, department, level_class, residency_class, GPA, Dateofbirth

Constraints: Username should be UNIQUE. All the attributes namely, password, First_name, Last_name, email, phone, department, level_class, residency_class, GPA, Dateofbirth, are governed by NOT NULL constraint.

2. **Course:**

Course(course_id: string, course name: string, department: string, level class: string, gpa_req: number, pre_req_courses: string, spcl_approval_req: string, variable_credit: string, credits: number)

CREATE TABLE COURSES(COURSE_ID VARCHAR(6), COURSE_NAME VARCHAR(100) NOT NULL, DEPT_NAME VARCHAR(10) NOT NULL, LEVEL_CLASS VARCHAR(15) NOT NULL, GPA_REQ NUMERIC, PRE_REQ_COURSES VARCHAR(50), SPCL_APPROVAL_REQ VARCHAR(3) NOT NULL,VARIABLE_CREDIT VARCHAR(10) NOT NULL, CREDITS INT NOT NULL, CONSTRAINT COURSE_PK PRIMARY KEY(COURSE_ID), CONSTRAINT LEVEL_CLASS_CHECK CHECK(LEVEL_CLASS IN ('Undergraduate', 'Graduate')));

Description: Contains the list of all courses and its basic information i.e. whether it requires special permission, has any prerequisites, GPA requirement, variable credit.

Primary key: course_id

Functional Dependency:
course_id -> (course name, department, level class, gpa_req, pre_req_courses, spcl_approval_req, variable_credit, credits)

Constraints: Attributes namely, course name, department, level class, spcl_approval_req, variable_credit, credits are governed by NOT NULL constraint.

3. **Administrator:**

Admins(employee_id: number, username: string, password: string, First_name: string, Last_name: string, Dateofbirth: date)

CREATE TABLE Admins(Employee_id INT, username VARCHAR(20) UNIQUE, password VARCHAR(50) NOT NULL, SSN VARCHAR(7) NOT NULL, First_Name VARCHAR(20) NOT NULL, Last_Name VARCHAR(20) NOT NULL, Dateofbirth DATE NOT NULL, CONSTRAINT admin_pk PRIMARY KEY(Employee_id));

Description: Contains all the personal information about a admin.

Primary key: employee_id

Functional Dependency:
employee_id -> username, password, First_name, Last_name, Dateofbirth

Constraints: Username should be UNIQUE. All the attributes namely, password, First_name, Last_name, Dateofbirth, are governed by NOT NULL constraint.

4. **Course Offering :**

Course_Offering(course_id: number, **faculty_name: string, semester: string,** days_of_week: string, start_time: time, end_time: time, class_size: number, number_of_enrolled: number, waitlist_size: number)

CREATE TABLE COURSE_OFFERING(COURSE_ID VARCHAR(6), FACULTY_NAME VARCHAR(50), SEMESTER VARCHAR(10) NOT NULL, DAYS_OF_WEEK VARCHAR(20) NOT NULL, START_TIME interval day (0) to second(0) NOT NULL, END_TIME interval day (0) to second(0) NOT NULL, CLASS_SIZE INT NOT NULL, NUMBER_OF_ENROLLED INT NOT NULL, WAITLIST_SIZE INT NOT NULL, CONSTRAINT COURSE_OFFERING_PK PRIMARY KEY (COURSE_ID, FACULTY_NAME, SEMESTER), CONSTRAINT SEMESTER_CHECK CHECK(SEMESTER IN ('Fall','Spring','Summer')));

Description: Contains the information about course offering for every semester.

Primary key: course_id, semester, faculty_name

Functional Dependency:
course_id, semester, faculty_name -> days_of_week, start_time, end_time, class_size, number_of_enrolled, waitlist_size

Constraints: (course_id, semester, faculty_name) together identify a unique record. Attributes namely, days_of_week, start_time, end_time, class_size, number_of_enrolled, waitlist_size are governed by NOT NULL constraint.

5. **Variable credit courses:**

Variable_Credit_Courses(course_id: string, min_credits: number, max_credits: number)

CREATE TABLE VARIABLE_CREDIT_COURSES(COURSE_ID VARCHAR(6), MIN_CREDIT INT NOT NULL, MAX_CREDIT INT NOT NULL, CONSTRAINT variable_pk PRIMARY KEY (COURSE_ID));

Description: Contains the information about course_id that requires variable credit features i.e. minimum and maximum credits for the range.

Primary key: course_id

Functional Dependency:
course_id -> min_credits, max_credits

Constraints: Attributes namely, min_credits and, max_credits are governed by NOT NULL constraint.

6. Creditmap:
Creditmap(level_class: string, residency_class: string, min_credits: number, max_credits number, Cost_per_credit: number);

CREATE TABLE CREDITMAP(level_class VARCHAR(20), residency_class VARCHAR(20), min_credits INT,max_credits INT, Cost_per_credit INT);

Description: Contains the information about credit structure i.e. credit cost and minimum and maximum credit requirements, based on the level of class and level of residency.

Primary key: level_class, residency_class

Functional Dependency:
level_class, residency_class -> min_credits, max_credits number, Cost_per_credit

Constraints: (level_class, residency_class) together identify a unique record.

7. Grademap:
Grademap(letter_grade: string, number_grade: number)

CREATE TABLE GRADEMAP(LETTER_GRADE VARCHAR(2), NUMBER_GRADE DECIMAL(3,2));

Description: Contains the information about course offering for every semester.

Primary key: letter_grade

Functional Dependency:
letter_grade -> number_grade:

Constraints: letter_grade uniquely identifies a number grade.

2.2 Weak Entities:

1. Student bill:

Studentbill(student_id: number, total_amount:number, amount_paid: number);

CREATE TABLE STUDENTBILL(Student_id INT, Total_Amount INT DEFAULT 0, Amount_Paid INT DEFAULT 0, CONSTRAINT bill_fk FOREIGN KEY (Student_id) REFERENCES STUDENT(Student_id) ON DELETE CASCADE);

Description: Contains the information about student bill i.e. amount paid and total amount.

Foreign key constraint:
student_id foreign key of Student table.

Functional Dependency:
student_id -> total_amount, amount_paid

Constraints: (total_amount, amount_paid) have a default value 0 when a new student is created. But, as soon as the student enrolls into courses, based upon his credit and using the information in grademap, and creditmap table, the values for total_amount, amount_paid are updated.

<u>2.3 Specialization:</u>

1. Enrollment:

Enrollment is specialized as General enrollment and waitlist enrollment.

1. **Gen_Enrollment(student_id: number**, **course_id: string, faculty: string, semester: string**, letter grade: string,credit: number)

CREATE TABLE GEN_ENROLLMENT(STUDENT_ID INT, COURSE_ID VARCHAR(10), FACULTY VARCHAR(50), SEMESTER VARCHAR(10), LETTER_GRADE VARCHAR(5), CREDIT INT, CONSTRAINT enrollment_pk PRIMARY KEY (Student_id, Course_id, FACULTY, Semester));

Description: Contains the information about confirmed enrollment of students in various courses.

Primary key: student_id, course_id, semester, faculty_name

Functional Dependency:
student_id, course_id, semester, faculty_name -> letter grade, credit

Constraints: (student_id, course_id, semester, faculty_name) together identify a unique record.

2. **Waitlist (student_id: number**, **course_id: string, faculty: string, semester: string**, waitlist_number: number,credit: number)

CREATE TABLE WAITLIST(STUDENT_ID INT, COURSE_ID VARCHAR(10), FACULTY VARCHAR(50), SEMESTER VARCHAR(10), WAITLIST_NUMBER INT, CREDIT INT,
CONSTRAINT waitlist_pk PRIMARY KEY (Student_id, Course_id, FACULTY, Semester));

Description: Contains the information about waitlisted students in various courses.

Primary key: student_id**,** course_id, semester, faculty_name

Functional Dependency:
student_id, course_id, semester, faculty_name -> waitlist_number, credit

Constraints: (student_id, course_id, semester, faculty_name) together identify a unique record.

3. Special_permission:

Special_permission(create_time: time, student_id: number, course_id: string, faculty: string, semester: string, approval_status: string, admin_name: string)

```
CREATE TABLE SPECIAL_PERMISSION( CREATE_TIME TIMESTAMP,
STUDENT_ID INT, COURSE_ID VARCHAR(10), FACULTY VARCHAR(50),
SEMESTER VARCHAR(10), APPROVAL_STATUS VARCHAR(20),
ADMIN_NAME VARCHAR(50), DATE_OF_APPROVAL DATE,
CONSTRAINT special_permission_pk PRIMARY KEY (Student_id, Course_id,
FACULTY, Semester));
```

Description: Contains the information about enrollment of students in various courses that require special_permission.

Primary key: student_id, course_id, semester, faculty_name

Functional Dependency:
student_id, course_id, semester, faculty_name -> approval_status, admin_name

Constraints: (student_id, course_id, semester, faculty_name) together identify a unique record.

2.4 Relationships:

1. StudentCredit:

StudentCredit(**student_id: number**, min_credit: number, max_credit: number, current_credit: number, tot_credit: number)

CREATE TABLE STUDENTCREDIT(Student_id INT, min_credit INT DEFAULT 0, max_credit INT DEFAULT 0, current_credit INT DEFAULT 0, tot_credit INT DEFAULT 0, CONSTRAINT credit_fk FOREIGN KEY (Student_id) REFERENCES STUDENT(Student_id) ON DELETE CASCADE);

Description: Contains the information about credits of students i.e. minimum credits, maximum credits, current credits, as well as total credits.

Foreign key:
Student_id is foreign key from student table.

Functional Dependency:
Student_id -> min_credit, max_credit, current_credit, tot_credit

Constraints:. The default values for min_credit, max_credit, current_credit, tot_credit is 0 for a newly enrolled student, but is updated as and when the enrollment procedure goes on.

2. Prerequisite:

Prerequisite(course_id: string, prerequisite_id: string, letter_grade: string, number_grade: number)

CREATE TABLE PREREQUISITE(COURSE_ID VARCHAR(10), PREREQUISITE_ID VARCHAR(10), LETTER_GRADE VARCHAR(5), CONSTRAINT prereq_pk PRIMARY KEY(COURSE_ID, PREREQUISITE_ID));

Description: Contains the information about prerequisite courses required for a course as well as any basic grade requirement for the particular prerequisite.

Primary key: course_id, prerequisite_id

Functional Dependency:
course_id, prerequisite_id -> letter_grade

Constraints:. **(**course_id, prerequisite_id) together identify a unique record.

2.5 Data Validation Constraints:

1. In the course offering table, the currently enrolled students should not be greater than the class size.

CONSTRAINT class_size_constraint CHECK(NUMBER_OF_ENROLLED <= CLASS_SIZE),

2. In the course offering table, the currently waitlisted students should not be greater than the waitlist size.

CONSTRAINT waitlist_size_constraint CHECK(WAITLISTED <= WAITLISTED)

3. In the student table, the residency level should only be one of the following:
CONSTRAINT residency CHECK(residency_class in ('In-State', 'Out-State', 'International'))

4. In the course table, the classification level should only be one of the following:
CONSTRAINT LEVEL_CLASS_CHECK CHECK(LEVEL_CLASS IN ('Undergraduate', 'Graduate'))

5. The student's current credit limit is also ensured to be less than the total credit limit.
CONSTRAINT credit_limit_constraint CHECK (max_credit >= current_credit)

6. The phone number entered for the student should be of length 10 is also ensured.
CONSTRAINT phone_length CHECK (LENGTH(phone) = 10)

2.6 Triggers

The triggers were implemented to make the task of the application easier by updating tables on insert, delete or update on a table. There were times when triggers had to be applied on tables on which delete operation is performed and the same table had to be updated. This was the case when a student moves from waitlist to enrolled and the waitlist number of other waitlisted students had to be updated. When this happens, a mutating trigger exception is thrown and a trigger does not work. This led us to move some of the functionality if the trigger inside the application code.

Following are some of the triggers implemented in this project:
1. This trigger is fired when a student enrolls in a course. It updates the current credit of the student and also updates the student bill considering the cost per credit of the student. Also the number of currently enrolled students gets updated.

```
CREATE OR REPLACE TRIGGER enrollment_trigger
AFTER INSERT ON ENROLLMENT
FOR EACH ROW
DECLARE TEMP1 INT;
BEGIN
UPDATE STUDENTCREDIT SET CURRENT_CREDIT =
CURRENT_CREDIT + :new.credit
  where STUDENT_ID = :new.STUDENT_ID;
SELECT CREDIT_COST INTO TEMP1 FROM STUDENTCREDIT WHERE
STUDENT_ID = :new.STUDENT_ID;
UPDATE STUDENTBILL SET TOTAL_AMOUNT =TOTAL_AMOUNT +
(TEMP1*:new.credit)
                                          WHERE STUDENT_ID = :new.STUDENT_ID;
UPDATE COURSE_OFFERING SET NUMBER_OF_ENROLLED =
NUMBER_OF_ENROLLED + 1
        where course_id = :new.course_id and Semester = 'SPRING'
                                                 and Faculty_Name = :new.FACULTY;
```

```
END;
/
```

2. This trigger is fired when a student drops a course. The trigger updates the number of currently enrolled students, updates the current credits of the students and also updates the bill of the student who has dropped the course.

```
CREATE OR REPLACE TRIGGER enrollment_delete_triger
AFTER DELETE ON ENROLLMENT
FOR EACH ROW
DECLARE
TEMP_COST INT;
BEGIN
UPDATE COURSE_OFFERING SET NUMBER_OF_ENROLLED =
NUMBER_OF_ENROLLED - 1
    WHERE course_id = :old.course_id and Semester = 'SPRING';
UPDATE STUDENTCREDIT SET CURRENT_CREDIT =
CURRENT_CREDIT - :old.CREDIT
    WHERE STUDENT_ID = :old.STUDENT_ID;
SELECT CREDIT_COST INTO TEMP_COST FROM STUDENTCREDIT
WHERE STUDENT_ID = :old.student_id;
UPDATE STUDENTBILL SET TOTAL_AMOUNT = TOTAL_AMOUNT -
(TEMP_COST * :old.CREDIT)
    WHERE STUDENT_ID = :old.STUDENT_ID;
END;
/
```

3. This trigger is fired when the student is being removed from the waitlist. Either he/she drops the course from waitlist or are moved to enrollment. This trigger updates the waitlist count for the course offering. This trigger also checks if the deadline has passed away or not.
If the deadline has passed, then it deletes all the entries from the table as a clean up process.

```
CREATE OR REPLACE TRIGGER waitlist_delete_trigger
```

```
AFTER DELETE ON WAITLIST
FOR EACH ROW
DECLARE TEMP_STATUS INT;
BEGIN
  SELECT STATUS INTO TEMP_STATUS FROM DEADLINE_ENFORCED;
  IF (TEMP_STATUS = 0) THEN
  UPDATE COURSE_OFFERING SET WAITLISTED = WAITLISTED - 1
where course_id = :old.course_id and Semester = 'SPRING';
  END IF;
  IF( TEMP_STATUS = 1 )THEN
  UPDATE COURSE_OFFERING SET WAITLISTED = 0;
  END IF;
END;
/
```

4. This trigger is fired when a student moves in the waitlist. It updates the waitlist count for that particular course offering for which the student has moved to waitlist.

```
CREATE OR REPLACE TRIGGER waitlist_trigger
AFTER INSERT ON WAITLIST
FOR EACH ROW
BEGIN
UPDATE COURSE_OFFERING SET WAITLISTED = WAITLISTED + 1
where course_id = :new.course_id and Semester = 'SPRING';
END;
/
```

2.7 Procedures:

1. We have implemented a procedure to calculate the gpa for a student when the admin enters the grades for the student.

```
create or replace PROCEDURE calgpa(sid INT)
AS
gpa1 DECIMAL(3,2);
BEGIN

SELECT SUM(EE.CREDIT * GM.NUMBER_GRADE)/SUM(EE.CREDIT)
INTO gpa1 FROM ENROLLMENT EE, GRADEMAP GM
WHERE GM.LETTER_GRADE = EE.LETTER_GRADE and EE.STUDENT_ID
= sid;

UPDATE STUDENT SET GPA = gpa1 WHERE STUDENT_ID = sid;
END;
```

Section 3. Constraints:

3.1 Database handled constraints

1. The credit limit for newly enrolled student is set after a new student is enrolled based on the level classification and residency level. This has been ensured by creating a trigger to set the credit limits of the newly enrolled student based on the attributes.
2. The cost per credit is also correctly set for newly enrolled student and this is also ensured by defining a trigger to do so.
3. The bill of the student gets updated automatically after he/she enrolls for the course based on the course credits and cost per credit. We have used a trigger to update the student bill when he/she enrolls in new course.
4. When a student enrolls in a course or gets added in wait list, the count for number of students in class and in waitlist are updated simultaneously using a trigger.
5. When the admin assigns the grade to the student for a particular course, the GPA gets updated. We have used a procedure to calculate the GPA of the student whenever a grade is entered.
6. Ensuring that the classification level and residency level entered in the table for a student is valid by using the 'CHECK' constraint to avoid confusion.
7. Ensuring that the phone number entered is valid by using 'CHECK' constraint.
8. When a student record is deleted, the subsequent entries of that student from tables like- 'STUDENTCREDIT' and 'STUDENTBILL' also gets deleted by defining a 'ON DELETE CASCADE' on the tables who have 'Student_ID' as their foreign keys.
9. When a student from waitlist enrolled for the course, the waitlist number of other students in the wait list gets updated automatically using a trigger.
10. The student also cannot enroll in the course which he/she has already enrolled in. This is ensured by defining a primary key which ensures that a student can only enroll once for a course.
11. The current enrolled credit cannot be greater than the maximum credit limit for the student. This is ensured by using the 'CHECK' constraint which

always checks whether the current credit limit is less than or equal to the maximum credit limit.

12. After deadline is enforced, the waitlist is cleaned up using a trigger to delete the waitlist table and also remove the students from enrollment who have not yet paid the fees.

3.2 Application handled constraints

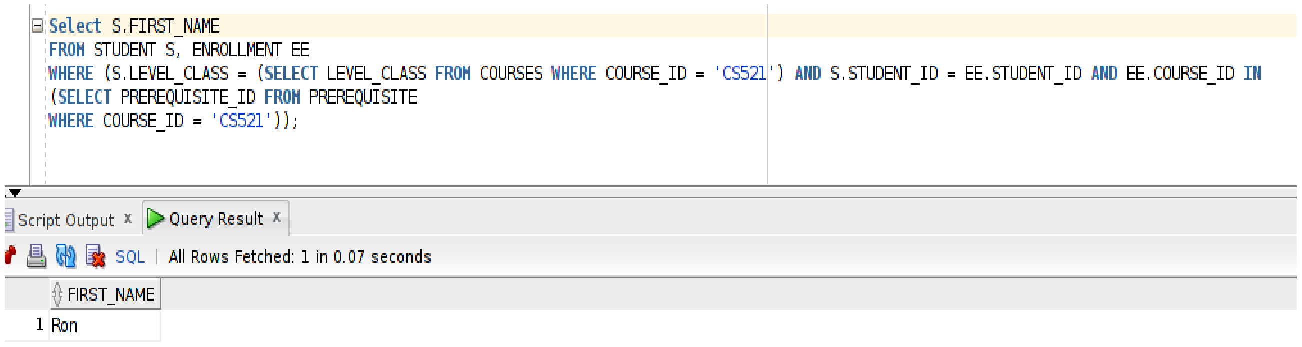
1. The student cannot access the pages of admin and vice-versa.
2. The student cannot change his/her residency level and classification level as it can affect the credit limits and cost per credits for the student.
3. The student is able to view the courses to enroll which are of the same level classification as that of the student.
4. The application has been designed to minimise the errors due to entering the values.
5. On enforcing the deadline, the student is not allowed to enroll or drop the courses.

Section 4: Report Queries

1. Find names of all students that are eligible to enroll for CSC 521 in Spring semester.(Based on prerequisites)

Solution:

Select S.FIRST_NAME
FROM STUDENT S, ENROLLMENT EE
WHERE (S.LEVEL_CLASS = (SELECT LEVEL_CLASS FROM COURSES
WHERE COURSE_ID = 'CS521') AND S.STUDENT_ID = EE.STUDENT_ID
AND EE.COURSE_ID IN
(SELECT PREREQUISITE_ID FROM PREREQUISITE
WHERE COURSE_ID = 'CS521'));



```
Select S.FIRST_NAME
FROM STUDENT S, ENROLLMENT EE
WHERE (S.LEVEL_CLASS = (SELECT LEVEL_CLASS FROM COURSES WHERE COURSE_ID = 'CS521') AND S.STUDENT_ID = EE.STUDENT_ID AND EE.COURSE_ID IN
(SELECT PREREQUISITE_ID FROM PREREQUISITE
WHERE COURSE_ID = 'CS521'));
```

```
All Rows Fetched: 1 in 0.07 seconds

  FIRST_NAME
1 Ron
```

2. Give list of students whose GPA decreased after the current semester

Solution:

```
SELECT SS.FIRST_NAME FROM STUDENT SS WHERE (
(SELECT SUM(EE.CREDIT * GM.NUMBER_GRADE)/SUM(EE.CREDIT)
FROM ENROLLMENT EE, GRADEMAP GM
WHERE GM.LETTER_GRADE = EE.LETTER_GRADE and EE.STUDENT_ID
= SS.STUDENT_ID)
-
(SELECT SUM(EE.CREDIT * GM.NUMBER_GRADE)/SUM(EE.CREDIT)
FROM ENROLLMENT EE, GRADEMAP GM
WHERE GM.LETTER_GRADE = EE.LETTER_GRADE and EE.STUDENT_ID
= SS.STUDENT_ID and EE.SEMESTER='SPRING')
)>0;
```

After assigning grades to the students, we ran this query and got the result as follows.

After Fall semester -

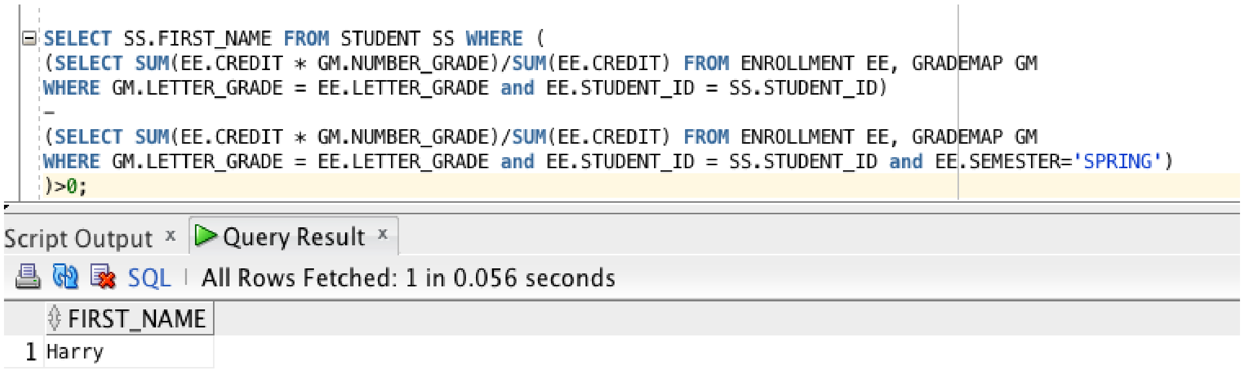
GPA for Harry - 4

GPA for Hermione - 3.33

After assigning grades for the spring courses-

GPA for Harry - 3

GPA for Hermione - 3.55



```
SELECT SS.FIRST_NAME FROM STUDENT SS WHERE (
(SELECT SUM(EE.CREDIT * GM.NUMBER_GRADE)/SUM(EE.CREDIT) FROM ENROLLMENT EE, GRADEMAP GM
WHERE GM.LETTER_GRADE = EE.LETTER_GRADE and EE.STUDENT_ID = SS.STUDENT_ID)
-
(SELECT SUM(EE.CREDIT * GM.NUMBER_GRADE)/SUM(EE.CREDIT) FROM ENROLLMENT EE, GRADEMAP GM
WHERE GM.LETTER_GRADE = EE.LETTER_GRADE and EE.STUDENT_ID = SS.STUDENT_ID and EE.SEMESTER='SPRING')
)>0;
```

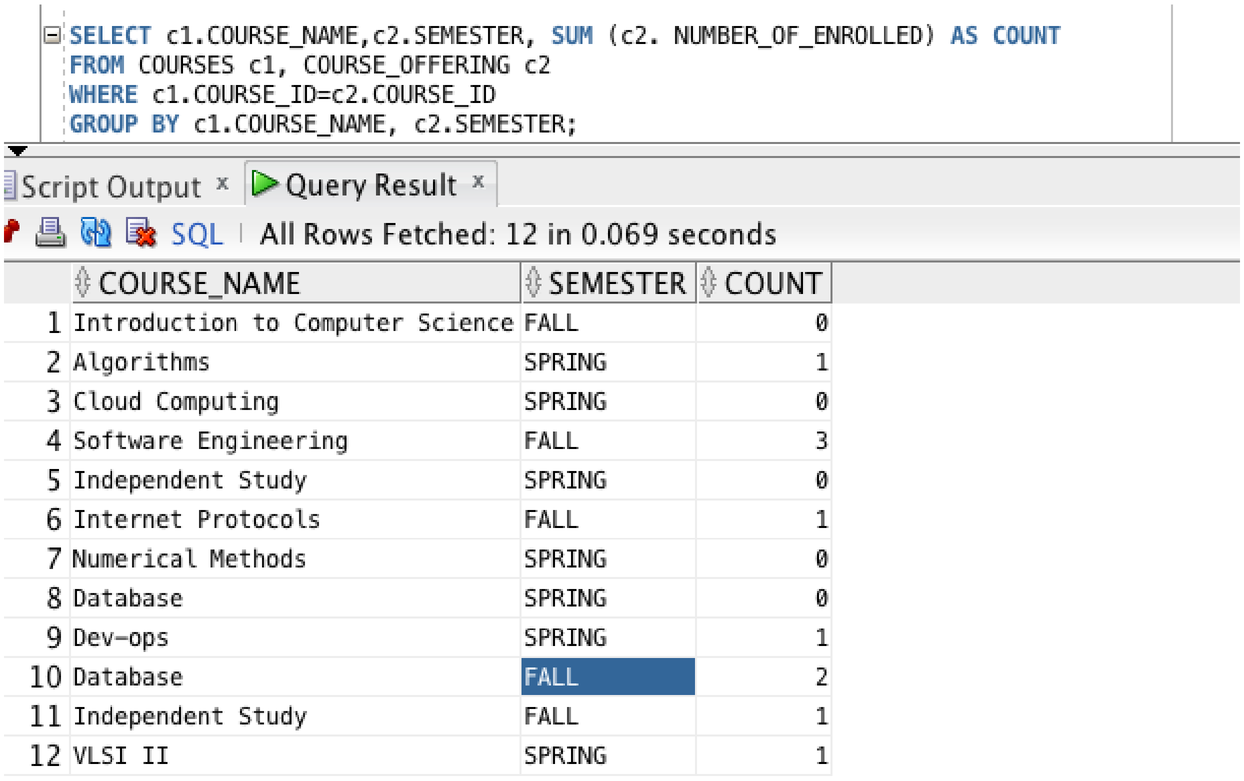
All Rows Fetched: 1 in 0.056 seconds

	FIRST_NAME
1	Harry

3. List all courses and number of students enrolled per semester

Solution:

SELECT c1.COURSE_NAME,c2.SEMESTER, SUM (c2. NUMBER _OF_ ENROLLED) AS COUNT
FROM COURSES c1, COURSE_OFFERING c2
WHERE c1.COURSE_ID=c2.COURSE_ID
GROUP BY c1.COURSE_NAME, c2.SEMESTER;



```
SELECT c1.COURSE_NAME,c2.SEMESTER, SUM (c2. NUMBER_OF_ENROLLED) AS COUNT
FROM COURSES c1, COURSE_OFFERING c2
WHERE c1.COURSE_ID=c2.COURSE_ID
GROUP BY c1.COURSE_NAME, c2.SEMESTER;
```

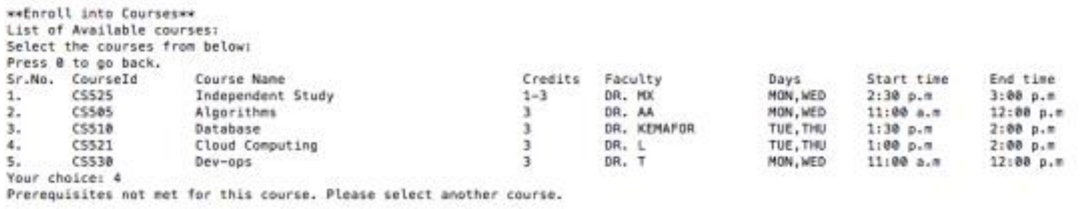
All Rows Fetched: 12 in 0.069 seconds

	COURSE_NAME	SEMESTER	COUNT
1	Introduction to Computer Science	FALL	0
2	Algorithms	SPRING	1
3	Cloud Computing	SPRING	0
4	Software Engineering	FALL	3
5	Independent Study	SPRING	0
6	Internet Protocols	FALL	1
7	Numerical Methods	SPRING	0
8	Database	SPRING	0
9	Dev-ops	SPRING	1
10	Database	FALL	2
11	Independent Study	FALL	1
12	VLSI II	SPRING	1

SCREENSHOTS

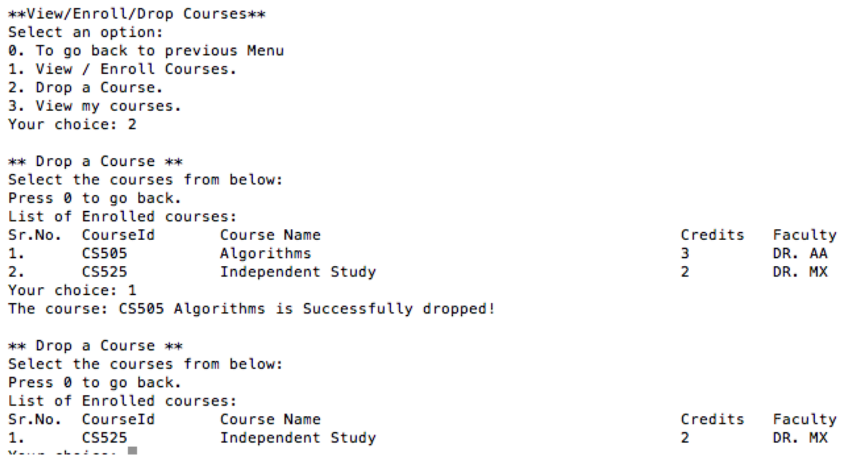
A few screenshots of the application.

Student view/enroll courses



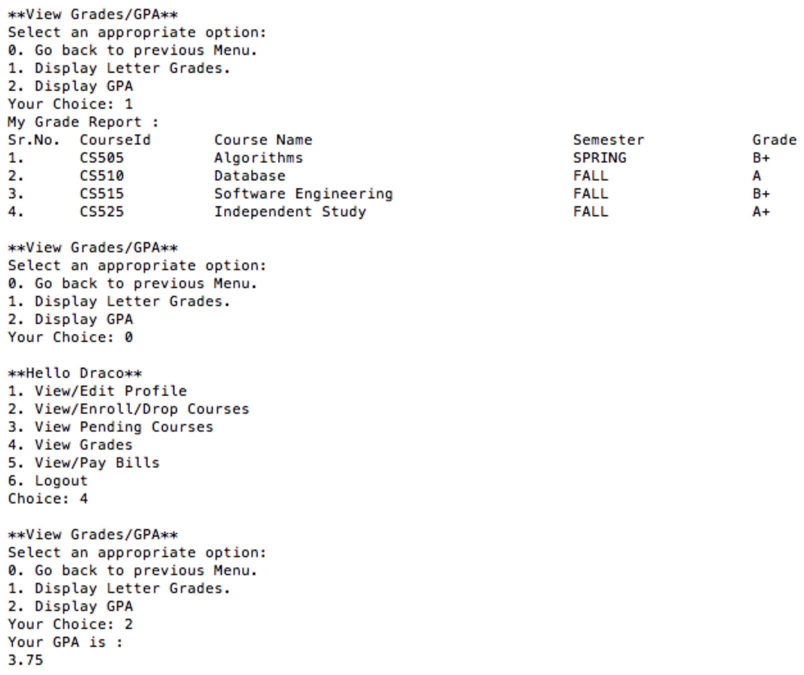
```
**Enroll into Courses**
List of Available courses:
Select the courses from below:
Press 0 to go back.
Sr.No.  CourseId      Course Name                          Credits   Faculty          Days       Start time    End time
1.      CS525         Independent Study                    1-3       DR. MX           MON,WED    2:30 p.m      3:00 p.m
2.      CS505         Algorithms                           3         DR. AA           MON,WED    11:00 a.m     12:00 p.m
3.      CS510         Database                             3         DR. KEMAFOR      TUE,THU    1:30 p.m      2:00 p.m
4.      CS521         Cloud Computing                      3         DR. L            TUE,THU    1:00 p.m      2:00 p.m
5.      CS530         Dev-ops                              3         DR. T            MON,WED    11:00 a.m     12:00 p.m
Your choice: 4
Prerequisites not met for this course. Please select another course.
```

Student drop courses



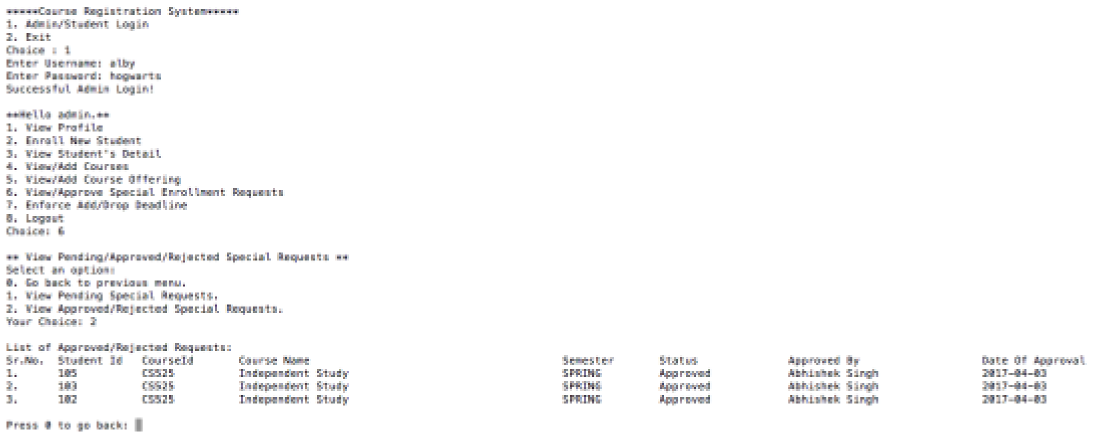
```
**View/Enroll/Drop Courses**
Select an option:
0. To go back to previous Menu
1. View / Enroll Courses.
2. Drop a Course.
3. View my courses.
Your choice: 2

** Drop a Course **
Select the courses from below:
Press 0 to go back.
List of Enrolled courses:
Sr.No.  CourseId      Course Name                                      Credits   Faculty
1.      CS505         Algorithms                                       3         DR. AA
2.      CS525         Independent Study                                2         DR. MX
Your choice: 1
The course: CS505 Algorithms is Successfully dropped!

** Drop a Course **
Select the courses from below:
Press 0 to go back.
List of Enrolled courses:
Sr.No.  CourseId      Course Name                                      Credits   Faculty
1.      CS525         Independent Study                                2         DR. MX
```

Admin assigns grades



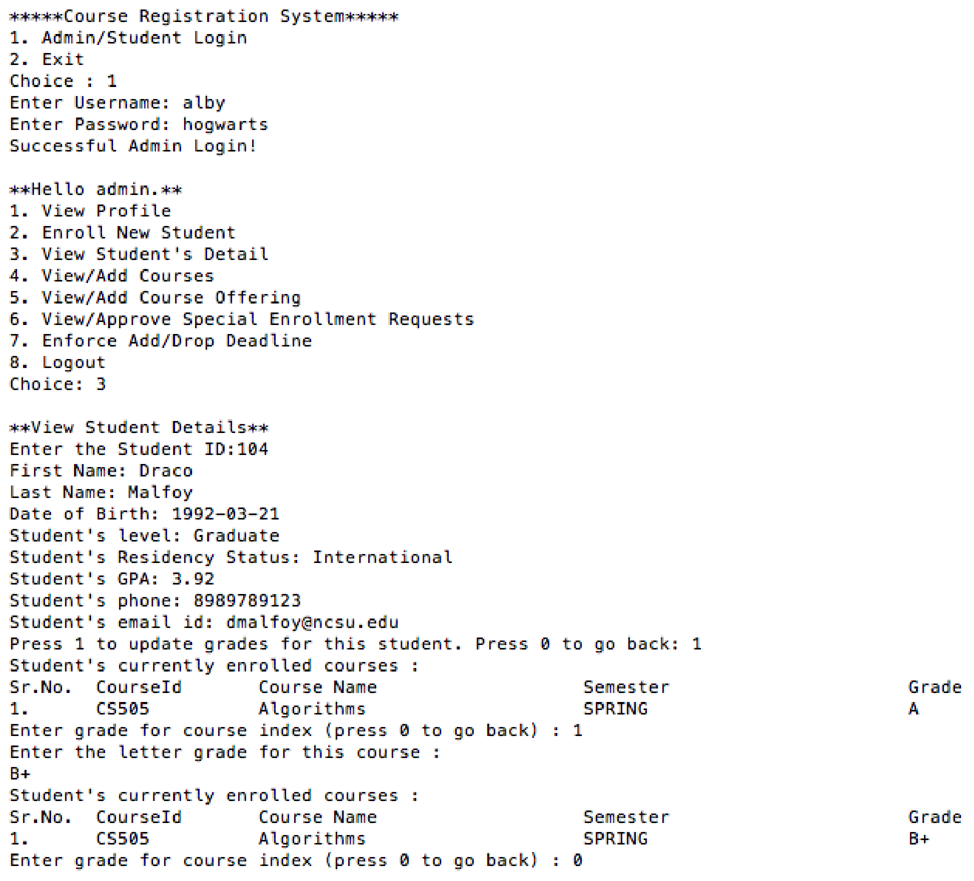
```
**View Grades/GPA**
Select an appropriate option:
0. Go back to previous Menu.
1. Display Letter Grades.
2. Display GPA
Your Choice: 1
My Grade Report :
Sr.No.  CourseId        Course Name                             Semester            Grade
1.      CS505           Algorithms                              SPRING              B+
2.      CS510           Database                                FALL                A
3.      CS515           Software Engineering                    FALL                B+
4.      CS525           Independent Study                       FALL                A+

**View Grades/GPA**
Select an appropriate option:
0. Go back to previous Menu.
1. Display Letter Grades.
2. Display GPA
Your Choice: 0

**Hello Draco**
1. View/Edit Profile
2. View/Enroll/Drop Courses
3. View Pending Courses
4. View Grades
5. View/Pay Bills
6. Logout
Choice: 4

**View Grades/GPA**
Select an appropriate option:
0. Go back to previous Menu.
1. Display Letter Grades.
2. Display GPA
Your Choice: 2
Your GPA is :
3.75
```

Admin views special requests



```
*****Course Registration System*****
1. Admin/Student Login
2. Exit
Choice : 1
Enter Username: alby
Enter Password: hogwarts
Successful Admin Login!

**Hello admin.**
1. View Profile
2. Enroll New Student
3. View Student's Detail
4. View/Add Courses
5. View/Add Course Offering
6. View/Approve Special Enrollment Requests
7. Enforce Add/Drop Deadline
8. Logout
Choice: 6

** View Pending/Approved/Rejected Special Requests **
Select an option:
0. Go back to previous menu.
1. View Pending Special Requests.
2. View Approved/Rejected Special Requests.
Your Choice: 2

List of Approved/Rejected Requests:
Sr.No.  Student Id  CourseId     Course Name                           Semester     Status        Approved By             Date Of Approval
1.      105         CS525        Independent Study                     SPRING       Approved      Abhishek Singh          2017-04-03
2.      103         CS525        Independent Study                     SPRING       Approved      Abhishek Singh          2017-04-03
3.      102         CS525        Independent Study                     SPRING       Approved      Abhishek Singh          2017-04-03

Press 0 to go back: 
```

Student views grades



```
*****Course Registration System*****
1. Admin/Student Login
2. Exit
Choice : 1
Enter Username: alby
Enter Password: hogwarts
Successful Admin Login!

**Hello admin.**
1. View Profile
2. Enroll New Student
3. View Student's Detail
4. View/Add Courses
5. View/Add Course Offering
6. View/Approve Special Enrollment Requests
7. Enforce Add/Drop Deadline
8. Logout
Choice: 3

**View Student Details**
Enter the Student ID:104
First Name: Draco
Last Name: Malfoy
Date of Birth: 1992-03-21
Student's level: Graduate
Student's Residency Status: International
Student's GPA: 3.92
Student's phone: 8989789123
Student's email id: dmalfoy@ncsu.edu
Press 1 to update grades for this student. Press 0 to go back: 1
Student's currently enrolled courses :
Sr.No.  CourseId       Course Name                  Semester                     Grade
1.      CS505          Algorithms                   SPRING                       A
Enter grade for course index (press 0 to go back) : 1
Enter the letter grade for this course :
B+
Student's currently enrolled courses :
Sr.No.  CourseId       Course Name                  Semester                     Grade
1.      CS505          Algorithms                   SPRING                       B+
Enter grade for course index (press 0 to go back) : 0
```