|  |
| --- |
| PES UNIVERSITY |
| COURSE MANAGEMENT SYSTEM |
| DBMS MINI PROJECT |

|  |
| --- |
| Deepank Girish  Pes1201800395 |

CONTENTS

[PROBLEM STATEMENT 2](#_Toc41380244)

[ER MODEL 2](#_Toc41380245)

[RELATIONAL MODEL 3](#_Toc41380246)

[NORMALIZATION 6](#_Toc41380247)

[LOSSLESS JOIN PROPERTY 8](#_Toc41380248)

[TABLES WITH CHECK CONSTRAINTS 9](#_Toc41380249)

[REFERENTIAL INTEGRITY CONSTRAINTS 10](#_Toc41380250)

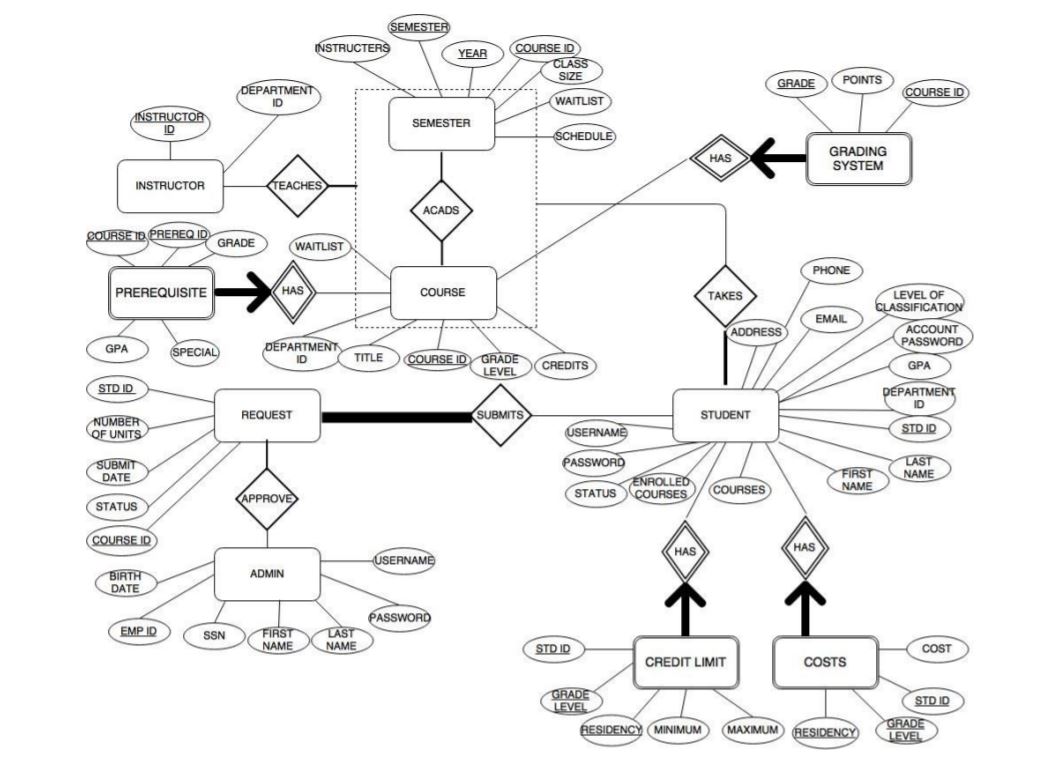
[TRIGGERS 11](#_Toc41380251)

[QUERIES 11](#_Toc41380252)

# PROBLEM STATEMENT

The purpose of the *Course Registration System* is to provide a fully functional database management system for educational institutions to automate and comprehensively handle all end-to-end relevant functionalities. The goal of the project is to cover all core workflows of an educational institution which includes the processes relevant to the student account and administrator account. All the nuances related to admin privileges i.e. administrator account like adding a student, adding courses, 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.

# ER MODEL

****

*Fig.1 ER Diagram Illustration*

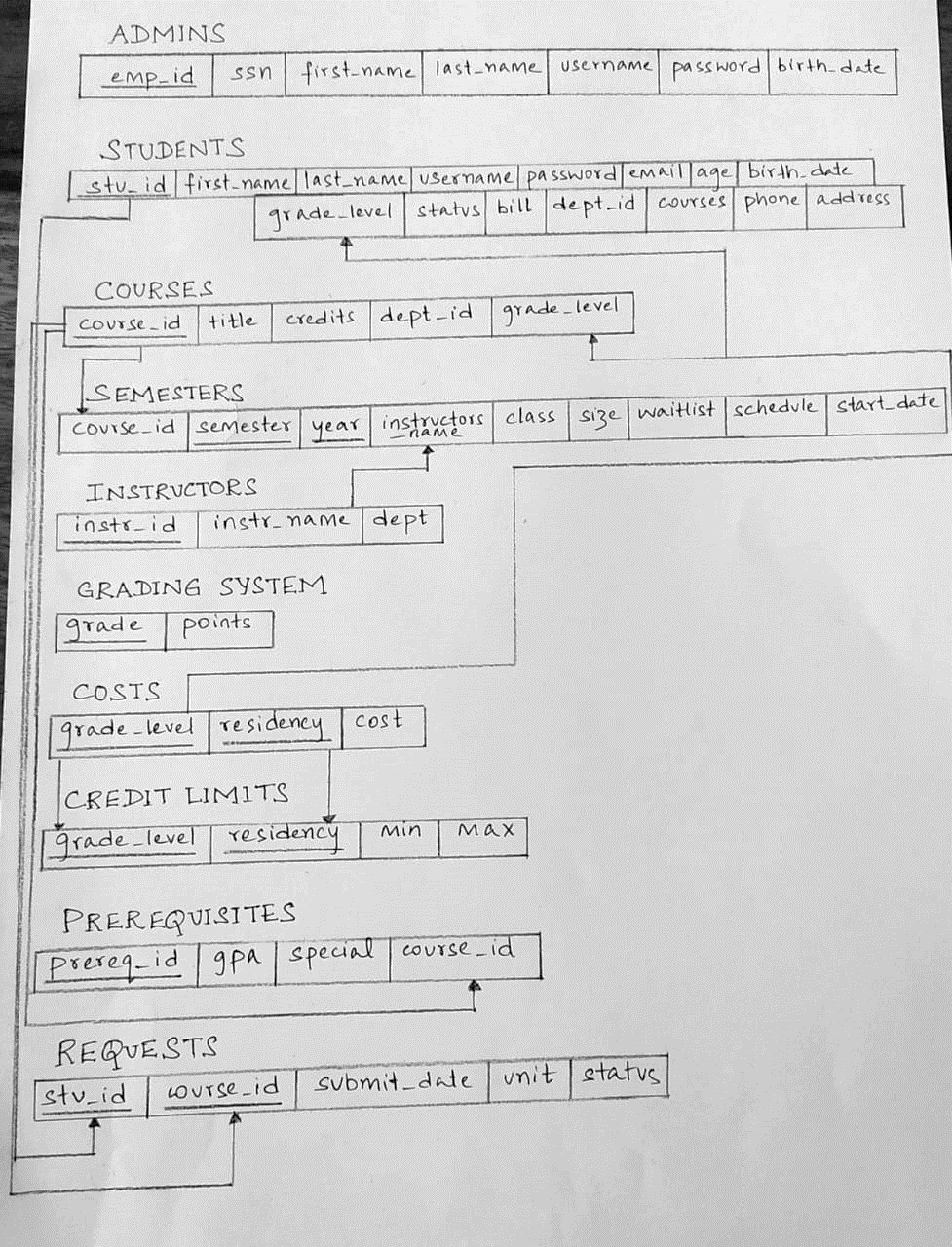
*Entities*: Admin, Request, Student, Course, Prerequisite, Semester, Instructor, Costs, Credit Limit, Grading System

*Aggregation Relationship*: Course and Semester

*Binary Relationships*: Admin-Request, Student-Request, Student-[Course/Info/Semester], Instructor-[Course/Info/Semester]

*Weak Entities*: Student: Costs, Credit Limit; Course: Prerequisite, Grading System

# RELATIONAL MODEL

****

*Fig.2 Schema Diagram*

**ADMINISTRATOR**: This table contains data of all the administrators. The admin will approve requests submitted by the students for adding a course in their time table. The admin is also responsible for adding courses and student’s entries in the university database.

The functional dependencies involved are:

* ADMIN.EMP\_ID -> ADMIN.SSN, ADMIN.FIRST\_NAME, ADMIN.LAST\_NAME, ADMIN.USERNAME
* ADMIN.SSN -> ADMIN.FIRST\_NAME, ADMIN.LAST\_NAME, ADMIN.PASSWORD, ADMIN.USERNAME
* ADMIN.USERNAME -> ADMIN.FIRST\_NAME, ADMIN.LAST\_NAME, ADMIN.PASSWORD, ADMIN.EMP\_ID, ADMIN.SSN

*Candidate key:* {EMP\_ID, SSN}

*Primary key:* EMP\_ID

*Alternate key:* SSN

**COURSES:** This table has the data for all the courses taught in a particular semester. The data involves the course title, course id, and the number of credits of a particular course. Only the admin has the authority to edit this table. The students will only be able to view it and add courses for their semester term. Some of the courses come with a prerequisite, relational entity of which is a weak entity to the course relational entity.

The functional dependencies involved are:

* COURSE.COURSE\_ID -> COURSE.TITLE, COURSE.CREDITS, CREDITS.DEPT\_ID, CREDITS.GRADE\_LEVEL
* COURSE.DEPT\_ID -> COURSE.TITLE

*Primary key:* COURSE\_ID

**INSTRUCTOR:** This table includes the list of all the instructors teaching a course in a particular semester. Every instructor is from a particular department of the university**.**

The functional dependencies involved are:

* INSTRUCTOR.INSTR\_ID -> INSTRUCTOR.INSTR\_NAME, INSTRUCTOR.DEPT

*Primary key:* INSTR\_ID

**GRADING SYSTEM:** This table maps the grades with the points, which will help in calculating the GPA of the student. Since there will be no grades for a course if the course itself is not there, this is a weak entity for the course table.

The functional dependencies involved are:

* GRADINGSYSTEM.GRADE -> GRADESYSTEM.POINTS

*Primary key:* INSTR\_ID

**STUDENTS:** The student table has the data for all the students taking courses at the university**.**

The functional dependencies involved are:

* STUDENT.STD\_ID -> STUDENT.FIRST\_NAME, STUDENT.LAST\_NAME, STUDENT.DEPT\_ID, STUDENT.PASSWORD, STUDENT.GRADE\_LEVEL
* STUDENT.EMAIL -> STUDENT.STD\_ID, STUDENT.FIRST\_NAME, STUDENT.LAST\_NAME, STUDENT.DEPT\_ID, STUDENT.PHONE, STUDENT.USERNAME
* STUDENT.PHONE -> STUDENT.STD\_ID, STUDENT.FIRST\_NAME, STUDENT.LAST\_NAME, STUDENT.DEPT\_ID
* STUDENT.USERNAME -> STUDENT.STD\_ID, STUDENT.FIRST\_NAME, STUDENT.LAST\_NAME, STUDENT.DEPT\_ID, STUDENT.PASSWORD, STUDENT.PHONE

*Candidate key:* {STD\_ID, PHONE}

*Primary key:* STD\_ID

*Alternate key:* PHONE

**CREDIT LIMITS:** This table helps in assigning maximum and minimum credits to a student, depending upon his/her residency status and grade level.

The functional dependencies involved are:

* CREDITLIMIT.GRADE\_LEVEL -> CREDITLIMIT.MIN, CREDITLIMIT.MAX
* CREDITLIMIT.RESIDENCY -> CREDITLIMIT.MIN, CREDITLIMIT.MAX

*Primary key:* {GRADE\_LEVEL, RESIDENCY}

**COSTS:** This table stores the bill for each student if any is there.

The functional dependencies involved are:

* COSTS.GRADE\_LEVEL -> COSTS.COST
* COSTS.RESIDENCY -> COSTS.COST

*Primary key:* {GRADE\_LEVEL, RESIDENCY}

**PREREQUISITES:** This table shows the prerequisites of a course. It also includes the special permission required from the department for a student to enroll in a course.

The functional dependencies involved are:

* PREREQUISITES.PREREQ\_ID -> PREREQUISITES.GPA, PREREQUISITES.SPECIAL, PREREQUISITES.COURSE\_ID
* PREREQUISITES.COURSE\_ID -> PREREQUISITES.GPA

*Primary key:* PREREQ\_ID

*Foreign key:* COURSE\_ID

**REQUESTS:** This table involves the request the students will send to the administrator to get a course added in their schedule.

The functional dependencies involved are:

* REQUESTS.STUD\_ID -> REQUESTS.SUBMITDATE, REQUESTS.UNIT, REQUESTS.STATUS
* REQUESTS.COURSE\_ID -> REQUESTS.SUBMITDATE, REQUESTS.UNIT, REQUESTS.STATUS

*Primary key:* {STUD\_ID, COURSE\_ID}

**SEMESTER:** This table lists the schedule for an entire semester, including the schedule of the classes, which instructor is teaching which course, and many more.

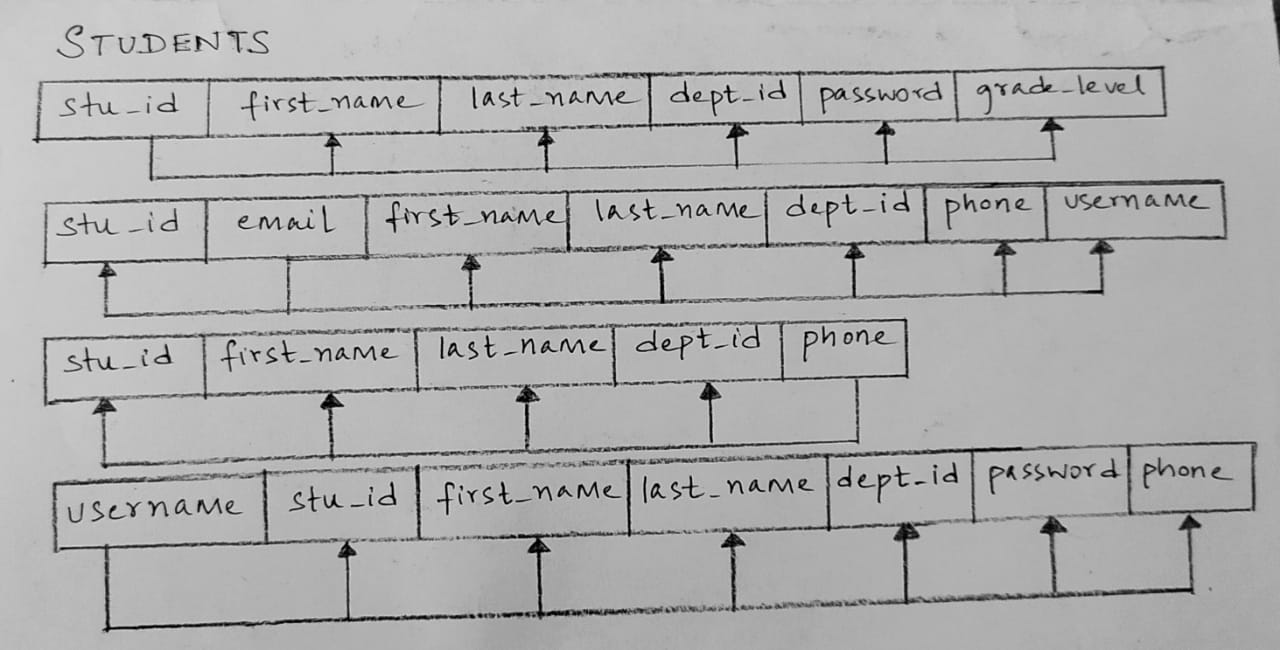
The functional dependencies involved are:

* SEMESTERS.CLASSSIZE -> SEMESTERS.WAITLISTS
* SEMESTERS.SEMESTER -> SEMESTER.SCHEDULE, SEMESTERS.STARTDATE
* SEMESTERS.YEAR -> SEMESTER.SCHEDULE, SEMESTERS.STARTDATE

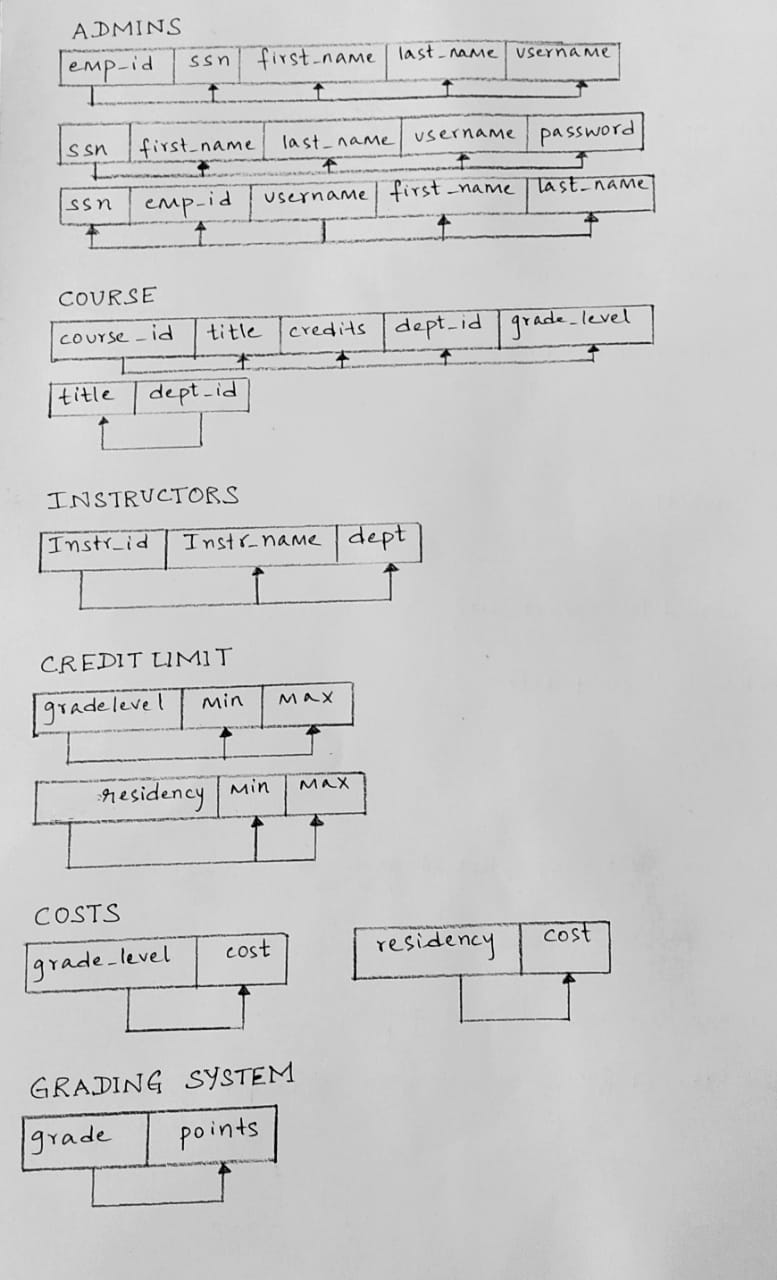
*Primary key:* {SEMESTER, YEAR}

*Foreign key:* COURSE\_ID

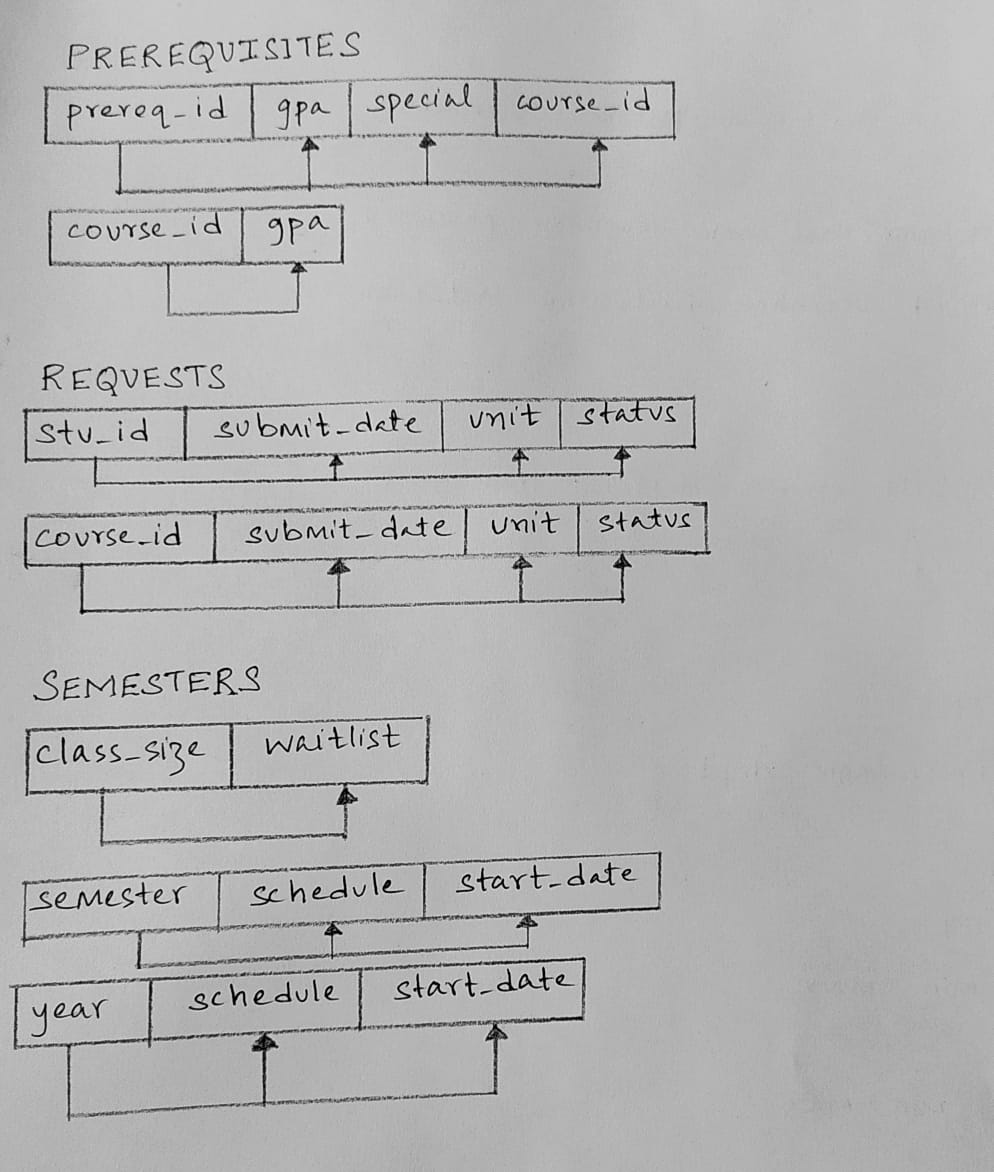
# NORMALIZATION

****

*Fig.3 Normalization (1)*



*Fig.4 Normalization (2)*

****

*Fig.5 Normalization (3)*

# LOSSLESS JOIN PROPERTY

For testing lossless or nonadditive join property, consider the table Prerequisites.

Below we have the initial matrix *S,*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **prereq\_id** | **gpa** | **special** | **course\_id** |
| **R1** | b11 | b12 | b13 | b14 |
| **R2** | b21 | b22 | b23 | b24 |

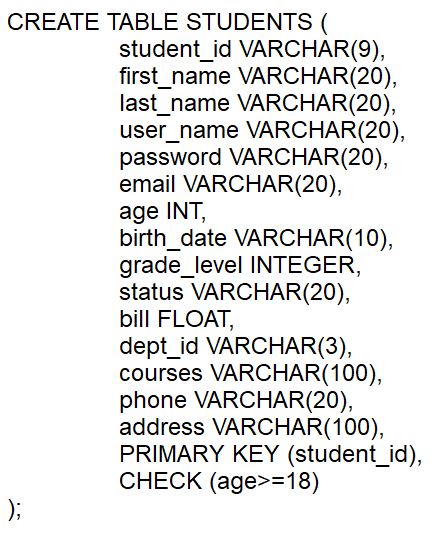
After decomposition into R1 and R2 we have,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **prereq\_id** | **gpa** | **special** | **course\_id** |
| **R1** | a1 | a2 | a3 | a4 |
| **R2** | b21 | a2 | b23 | a1 |

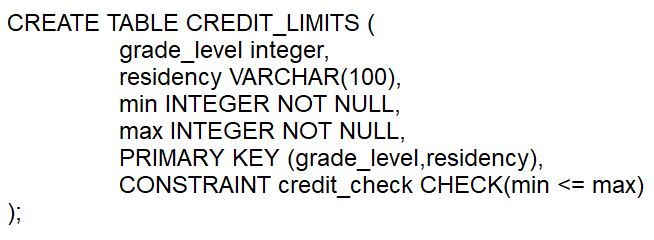
Here, the second row is entirely made of *a* symbol and also *R1 ∩ R2 -> R2*. Therefore, the above decomposition has the nonadditive join property.

# TABLES WITH CHECK CONSTRAINTS

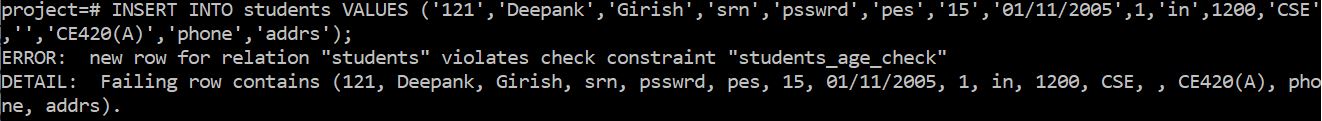
The tables ‘Students’ and ‘Credit Limits’ were created with check constraints. In table students, records of students whose age is less than 18 cannot be entered. In Credit Limits, minimum credit must always be less than maximum credit.



*Fig.6 Table Student*



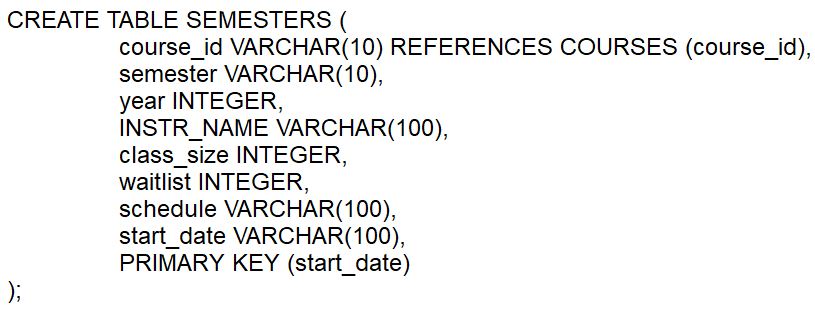
*Fig.7 Table Credit Limits*



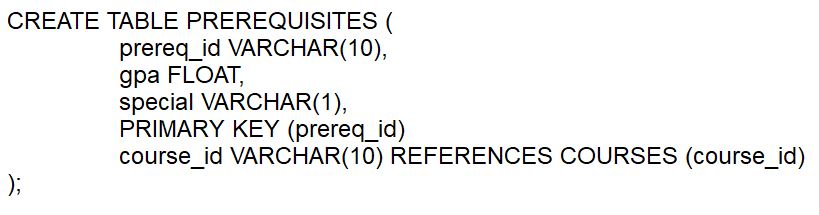
*Fig.8 Check Constraint Illustration*

# REFERENTIAL INTEGRITY CONSTRAINTS

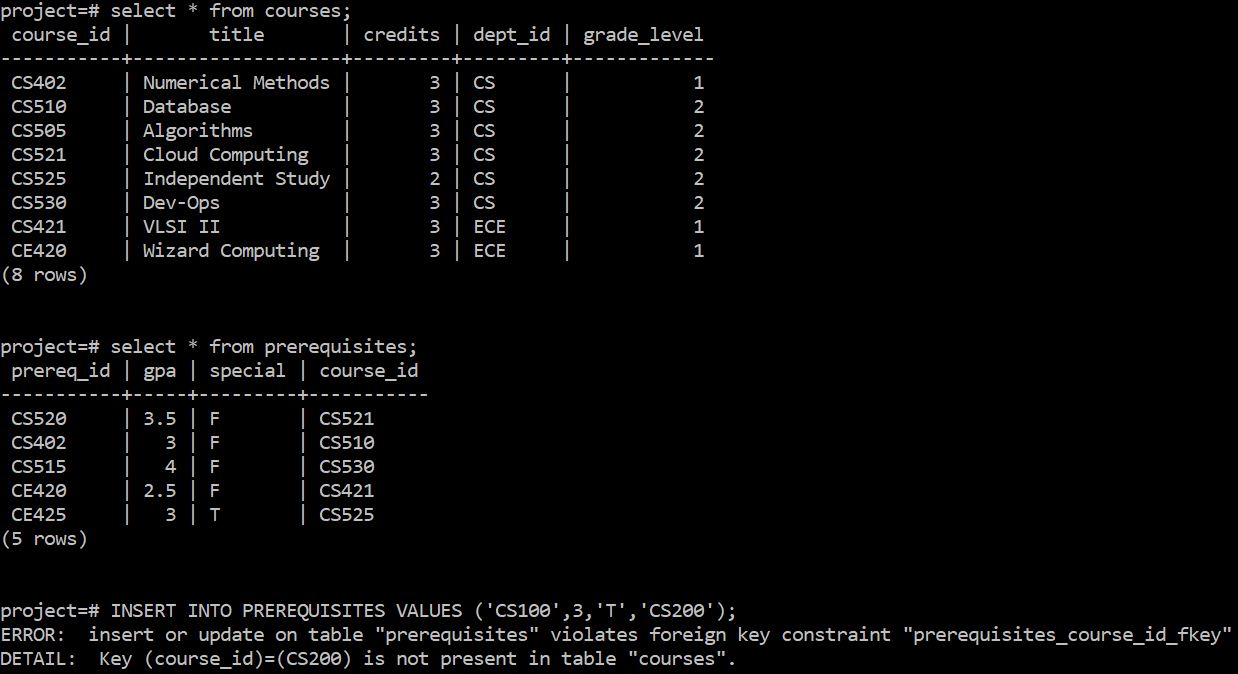
The referential integrity constraints were made use of in table ‘Prerequisites’ and ‘Semesters’. In Prerequisites, *course\_id* (COURSES table) is a foreign key. Similarly, *course\_id* is also the foreign key in Semesters.



*Fig.9 Table Semesters*



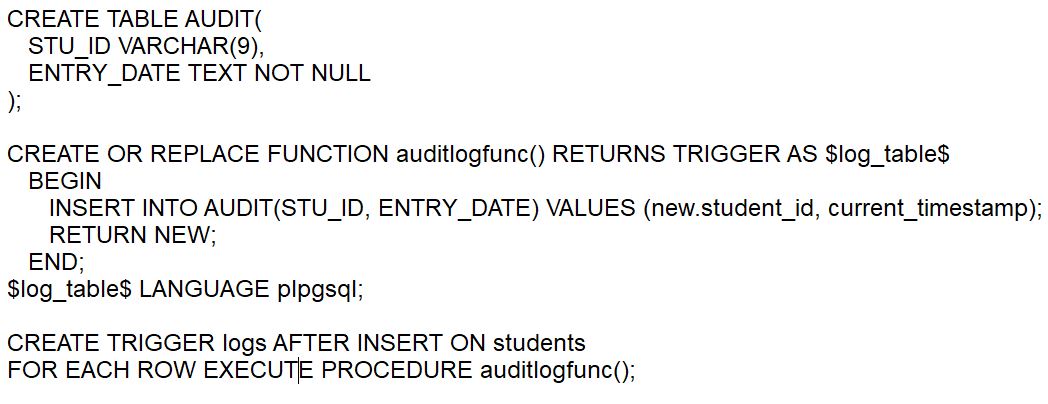
*Fig.9 Table Semesters*



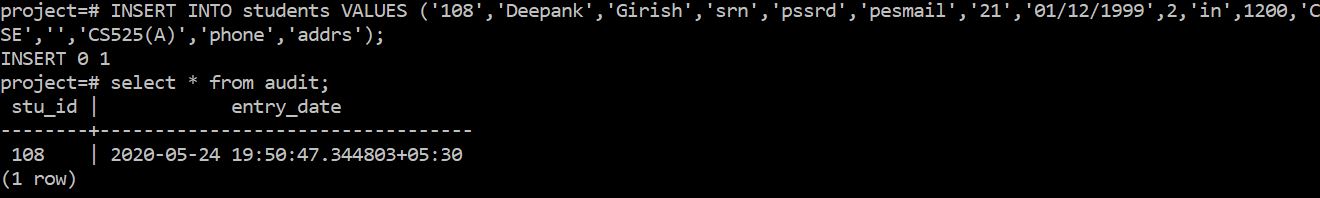
*Fig.9 Referential Integrity Constraints Illustration*

# TRIGGERS

In our model, a trigger gets fired whenever records are entered in table Students. This acts as an audit trial. Therefore, log messages are inserted into a new table ‘Audit’ using a procedure named *auditlogfunc* which we have defined. The audit table contains the added student id and corresponding insertion timestamp.



*Fig.10 Trigger Definition*

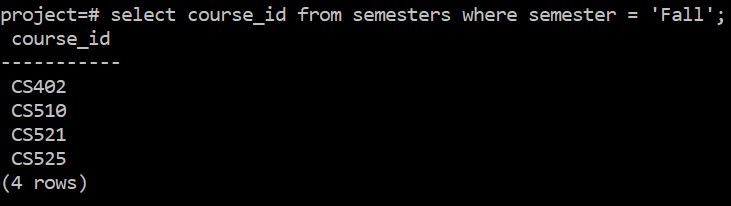


*Fig.10 Trigger Illustration*

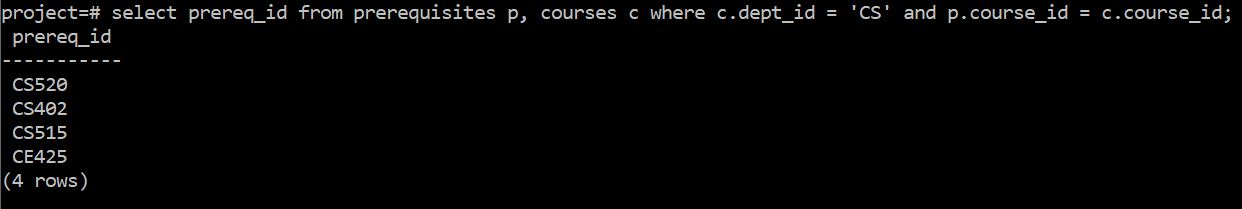
# QUERIES

The following three queries were implemented:

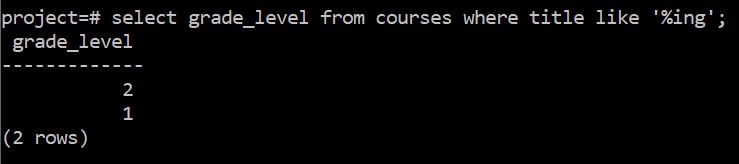
1. Display the IDs of the courses that are taught in the Fall semester.
2. List all the IDs of the prerequisite courses being offered by the CS department.
3. Show the grade levels of all the courses whose title ends with ‘-ing’.



*Fig.11 Query (1)*



*Fig.12 Query (2)*



*Fig.13 Query (3)*