PES UNIVERSITY

COURSE MANAGEMENT SYSTEM

DBMS MINI PROJECT

DEEPANK GIRISH
PES1201800395

CONTENTS

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

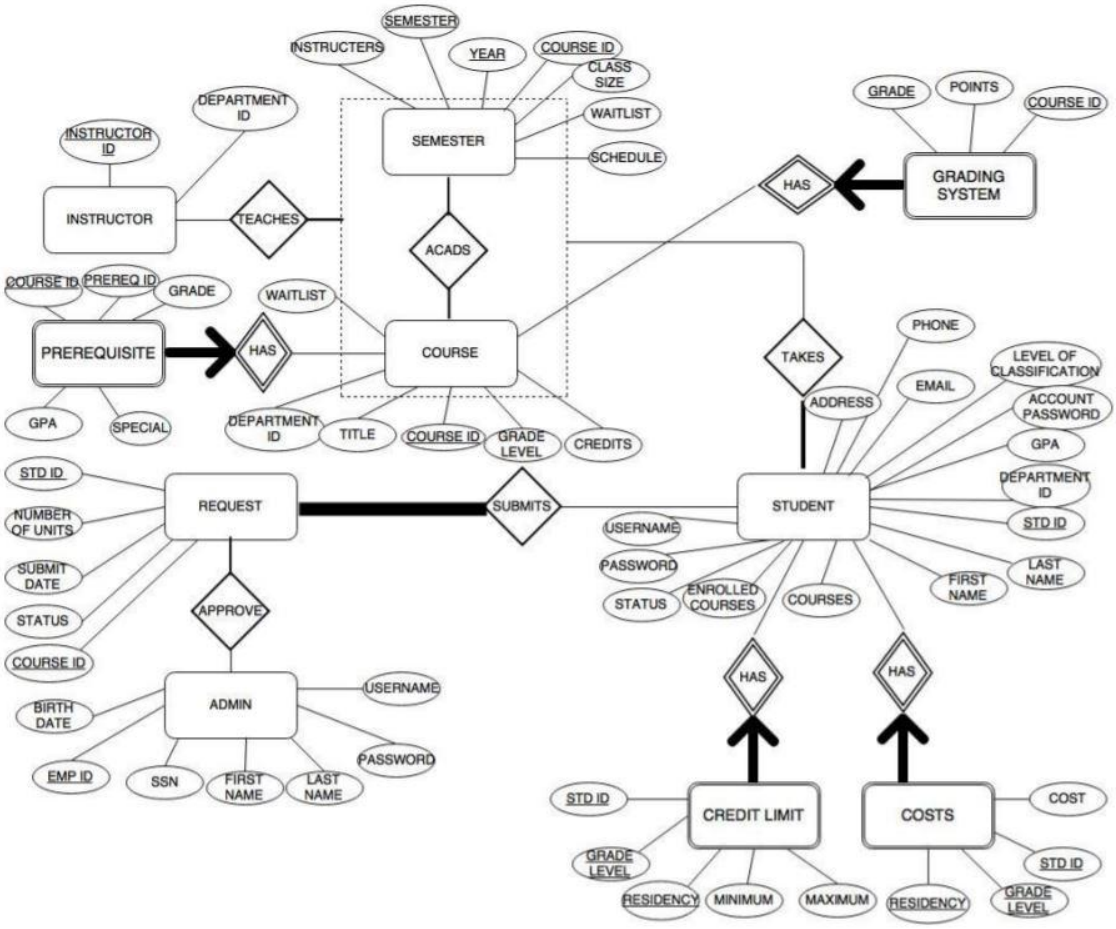

Fig.1 ER Diagram Illustration

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

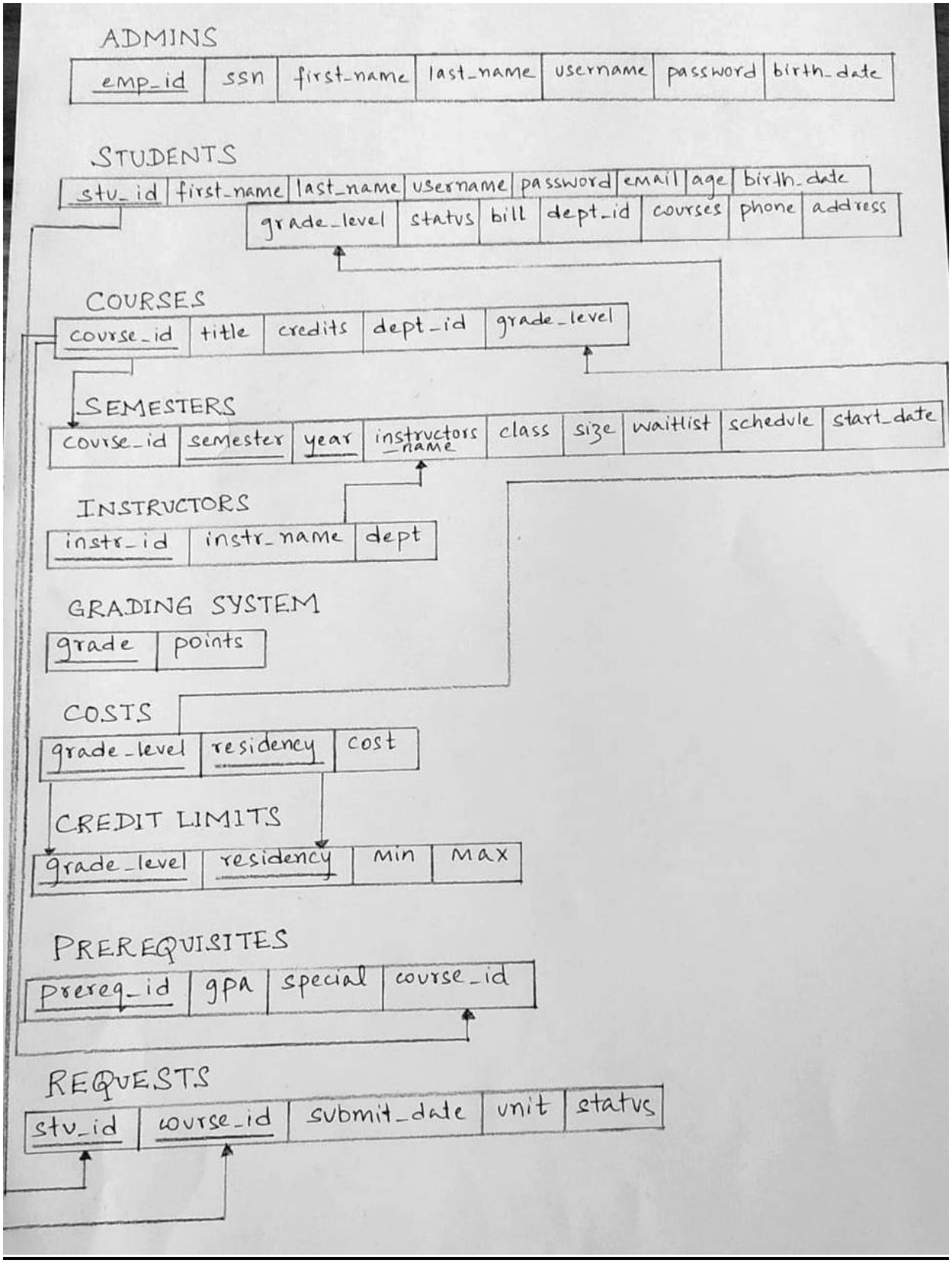

Fig.2 Schema Diagram

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

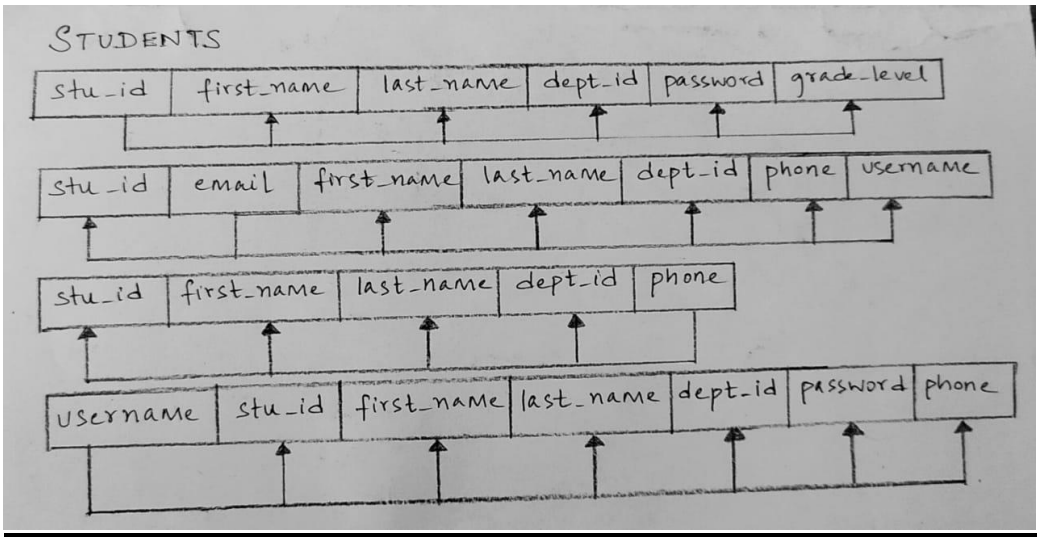

Fig.3 Normalization (1)

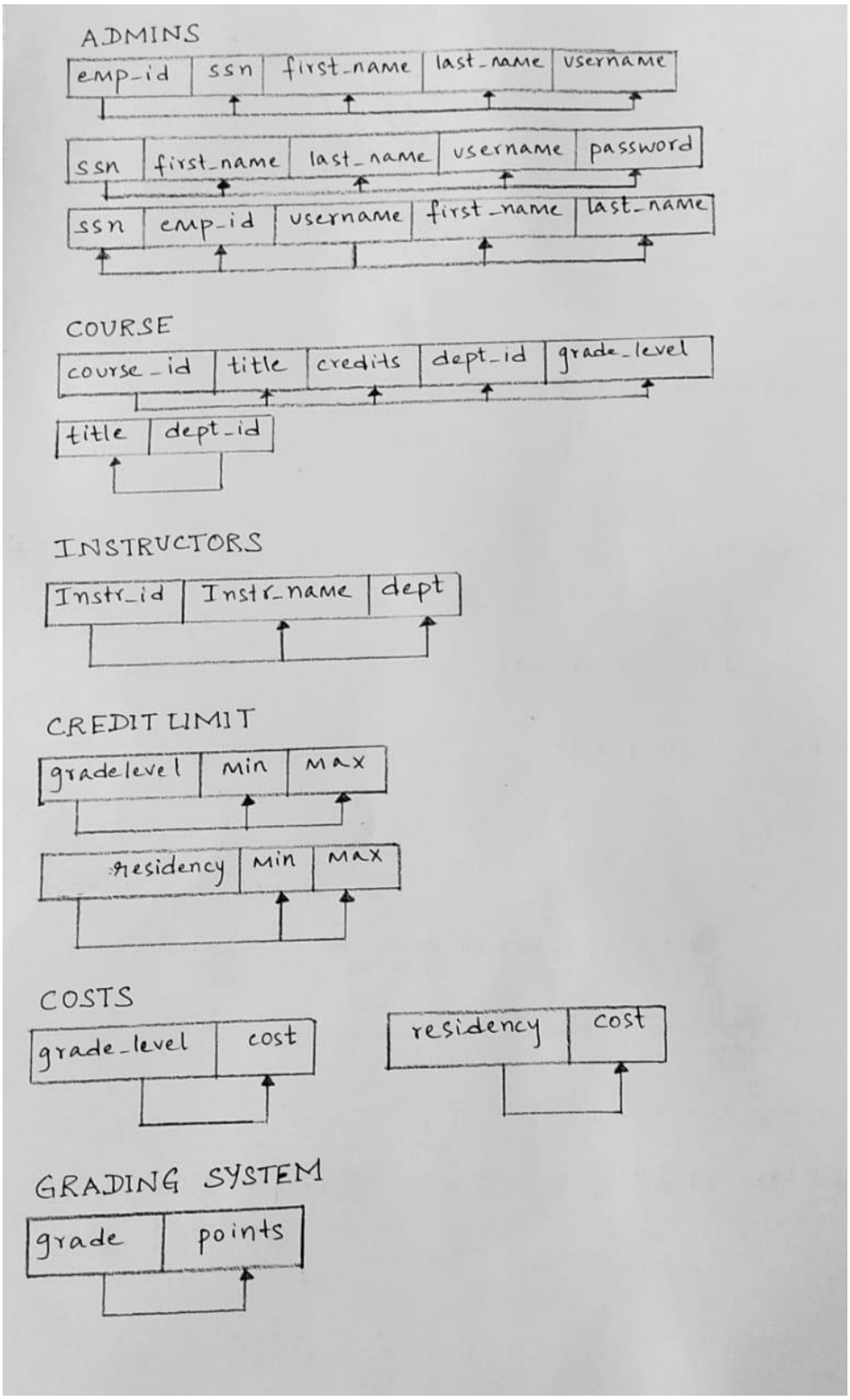

Fig.4 Normalization (2)

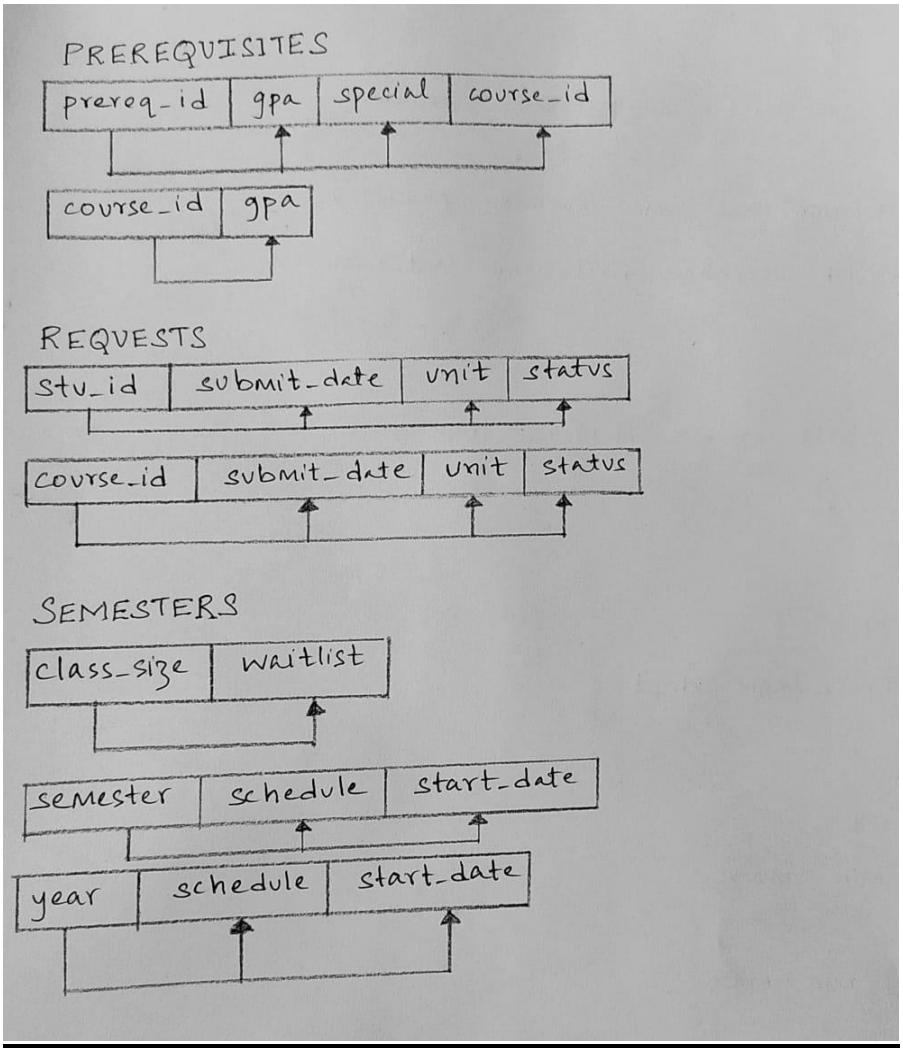

Fig.5 Normalization (3)

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**
R_1	b_{11}	b_{12}	b_{13}	b_{14}
R_2	b_{21}	b_{22}	b_{23}	b_{24}

After decomposition into R_1 and R_2 we have,

	prereq_id	**gpa**	**special**	**course_id**
R_1	a_1	a_2	a_3	a_4
R_2	b_{21}	a_2	b_{23}	a_1

Here, the second row is entirely made of *a* symbol and also $R_1 \cap R_2 \rightarrow R_2$. 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.

```
CREATE TABLE STUDENTS (
        student_id VARCHAR(9),
        first_name VARCHAR(20),
        last_name VARCHAR(20),
        user_name VARCHAR(20),
        password VARCHAR(20),
        email VARCHAR(20),
        age INT,
        birth_date VARCHAR(10),
        grade_level INTEGER,
        status VARCHAR(20),
        bill FLOAT,
        dept_id VARCHAR(3),
        courses VARCHAR(100),
        phone VARCHAR(20),
        address VARCHAR(100),
        PRIMARY KEY (student_id),
        CHECK (age>=18)
);
```

Fig.6 Table Student

```
CREATE TABLE CREDIT_LIMITS (
        grade_level integer,
        residency VARCHAR(100),
        min INTEGER NOT NULL,
        max INTEGER NOT NULL,
        PRIMARY KEY (grade_level,residency),
        CONSTRAINT credit_check CHECK(min <= max)
);
```

Fig.7 Table Credit Limits

```
project=# INSERT INTO students VALUES ('121','Deepank','Girish','srn','psswrd','pes','15','01/11/2005',1,'in',1200,'CSE'
,'','CE420(A)','phone','addrs');
ERROR:  new row for relation "students" violates check constraint "students_age_check"
DETAIL:  Failing row contains (121, Deepank, Girish, srn, psswrd, pes, 15, 01/11/2005, 1, in, 1200, CSE, , CE420(A), pho
ne, addrs).
```

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.

```
CREATE TABLE SEMESTERS (
            course_id VARCHAR(10) REFERENCES COURSES (course_id),
            semester VARCHAR(10),
            year INTEGER,
            INSTR_NAME VARCHAR(100),
            class_size INTEGER,
            waitlist INTEGER,
            schedule VARCHAR(100),
            start_date VARCHAR(100),
            PRIMARY KEY (start_date)
);
```

Fig.9 Table Semesters

```
CREATE TABLE PREREQUISITES (
            prereq_id VARCHAR(10),
            gpa FLOAT,
            special VARCHAR(1),
            PRIMARY KEY (prereq_id)
            course_id VARCHAR(10) REFERENCES COURSES (course_id)
);
```

Fig.9 Table Semesters

```
project=# select * from courses;
 course_id |       title       | credits | dept_id | grade_level
-----------+-------------------+---------+---------+-------------
 CS402     | Numerical Methods |       3 | CS      |           1
 CS510     | Database          |       3 | CS      |           2
 CS505     | Algorithms        |       3 | CS      |           2
 CS521     | Cloud Computing   |       3 | CS      |           2
 CS525     | Independent Study |       2 | CS      |           2
 CS530     | Dev-Ops           |       3 | CS      |           2
 CS421     | VLSI II           |       3 | ECE     |           1
 CE420     | Wizard Computing  |       3 | ECE     |           1
(8 rows)


project=# select * from prerequisites;
 prereq_id | gpa | special | course_id
-----------+-----+---------+-----------
 CS520     | 3.5 | F       | CS521
 CS402     |   3 | F       | CS510
 CS515     |   4 | F       | CS530
 CE420     | 2.5 | F       | CS421
 CE425     |   3 | T       | CS525
(5 rows)


project=# INSERT INTO PREREQUISITES VALUES ('CS100',3,'T','CS200');
ERROR:  insert or update on table "prerequisites" violates foreign key constraint "prerequisites_course_id_fkey"
DETAIL:  Key (course_id)=(CS200) is not present in table "courses".
```

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.

```
CREATE TABLE AUDIT(
   STU_ID VARCHAR(9),
   ENTRY_DATE TEXT NOT NULL
);

CREATE OR REPLACE FUNCTION auditlogfunc() RETURNS TRIGGER AS $log_table$
   BEGIN
      INSERT INTO AUDIT(STU_ID, ENTRY_DATE) VALUES (new.student_id, current_timestamp);
      RETURN NEW;
   END;
$log_table$ LANGUAGE plpgsql;

CREATE TRIGGER logs AFTER INSERT ON students
FOR EACH ROW EXECUTE PROCEDURE auditlogfunc();
```

Fig.10 Trigger Definition

```
project=# INSERT INTO students VALUES ('108','Deepank','Girish','srn','pssrd','pesmail','21','01/12/1999',2,'in',1200,'C
SE','','CS525(A)','phone','addrs');
INSERT 0 1
project=# select * from audit;
 stu_id |           entry_date
--------+-------------------------------
 108    | 2020-05-24 19:50:47.344803+05:30
(1 row)
```

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’.

```
project=# select course_id from semesters where semester = 'Fall';
 course_id
-----------
 CS402
 CS510
 CS521
 CS525
(4 rows)
```

Fig.11 Query (1)

```
project=# select prereq_id from prerequisites p, courses c where c.dept_id = 'CS' and p.course_id = c.course_id;
 prereq_id
-----------
 CS520
 CS402
 CS515
 CE425
(4 rows)
```

Fig.12 Query (2)

```
project=# select grade_level from courses where title like '%ing';
 grade_level
-------------
           2
           1
(2 rows)
```

Fig.13 Query (3)