DBMS Project

College Admission System

INTRODUCTION

The project is the simulation of the admission system or an allocation system. The project is based on Node JS server technology with XAMPP's Mysql database server. The technologies used in the website are

FRONTEND TECHNOLOGY

- HTML
- CSS
- JavaScript
- Bootstrap (for styling)
- Tox Progress (for progress bars)

BACKEND TECHNOLOGY

- Node JS
- EJS (templating language)
- Express (for web server)

INSTALL INSTRUCTIONS

- In Order for this project to run Node needs to be installed. The downloads page is here. Install it.
- In the collegeAdmission folder, there is a sql folder in which there is a sql file.

- Create a new database with the name collegeAdmission and import the above mentioned file.
- XAMPP server is required.
- In the collegeAdmission folder, open the command prompt in that folder and run the command **npm install.** This will install all the files needed for this project.
- In the same command prompt, run another command **npm run start**. This will start the server
- In the browser go to http://localhost:3000/
- The project is ready to test.

DESCRIPTION

The project is for two kinds of users.

- Students
- Admin

For the students, the tasks they can do are

- Register themselves as users of the page
- Login with their credentials
- Upload their image
- Add their academics information
- View their Rank
- Add their preferred choices
- Get their allotment order if application is accepted

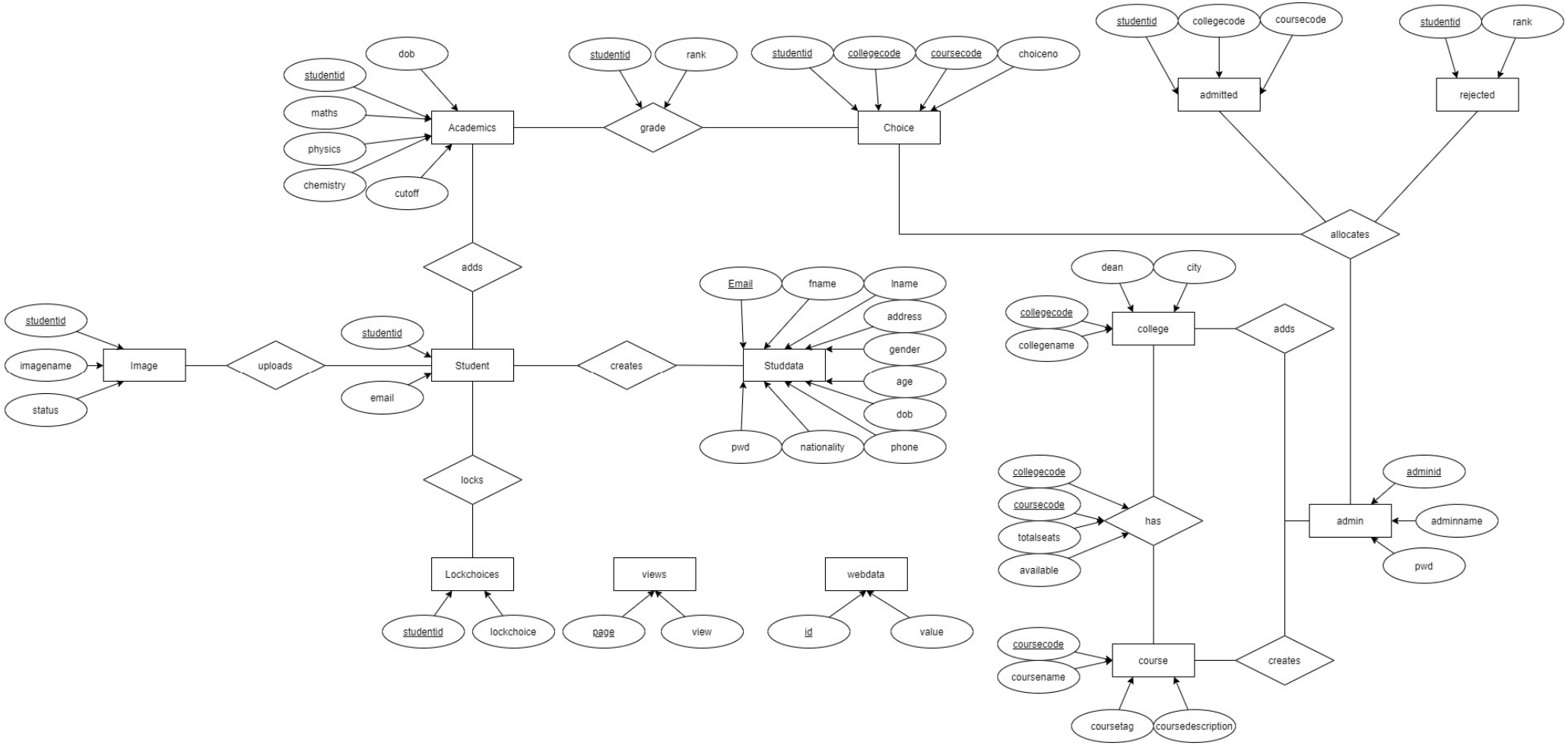
For the admin, the tasks he/she can do are

- Login with their credentials
- Monitor Views, Students, Colleges ,Courses
- Search and Delete a Student
- Add, Search and Delete a College
- Add, Search and Delete a Course
- Add and Delete a Course from a College
- Allocate seats based on students' ranks
- Reset all the data in the database

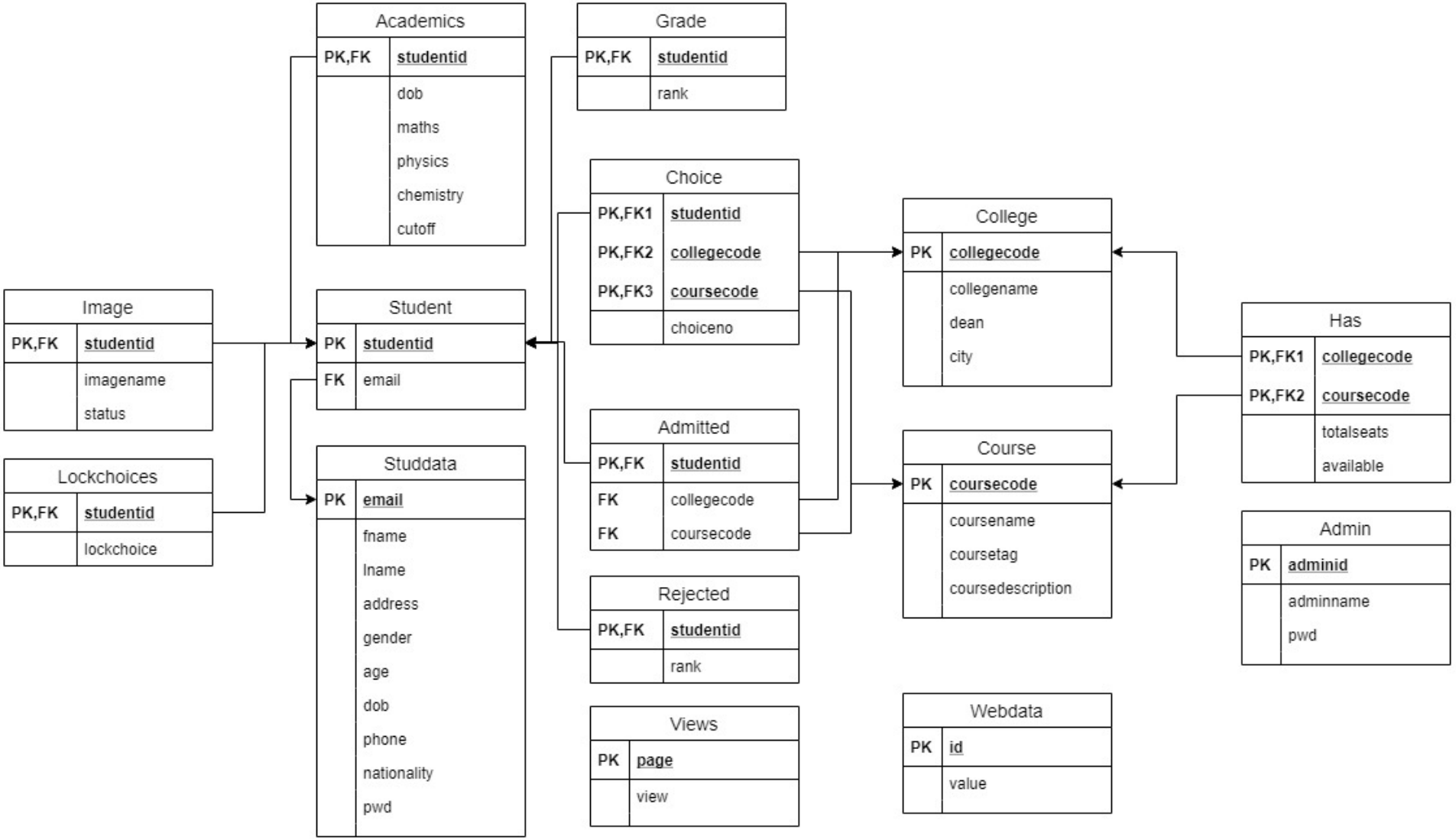
ADMIN LOGIN

Admin Name: admin
Password: admin

ER MODEL


studentid
collegecode
coursecode
admitted
studentid
rank
rejected
dob
studentid
maths
physics
chemistry
cutoff
Academics
studentid
rank
grade
studentid
collegecode
coursecode
choiceno
Choice
allocates
adds
dean
city
collegecode
collegename
college
adds
Email
fname
lname
address
gender
age
dob
pwd
nationality
phone
Studdata
studentid
imagename
status
Image
uploads
studentid
email
Student
creates
locks
collegecode
coursecode
totalseats
available
has
adminid
adminname
pwd
admin
Lockchoices
studentid
lockchoice
views
page
view
webdata
id
value
coursecode
coursename
course
creates
coursetag
coursedescription

RELATIONAL SCHEMA



TABLES DESCRIPTION

NORMALIZATION

All the tables are normalized upto 3NF.

1. STUDENT

Attributes	Data type
studentid	INT(11) AUTO INCREMENT
email	VARCHAR(100)

TABLE CREATE QUERY

```
CREATE TABLE `student` (
 studentid int(11) NOT NULL AUTO_INCREMENT,
 email varchar(100) DEFAULT NULL,
PRIMARY KEY (studentid),
FOREIGN KEY(email) REFERENCES studdata(email)
);
```

2.STUDDATA

Attributes	Data Type
email	VARCHAR(100)
fname	TINYTEXT
lname	VARCHAR(20)
address	VARCHAR(40)
gender	CHAR(1)
age	INT(11)
dob	DATE
phone	BIGINT(20)
nationality	TINYTEXT
pwd	LONGTEXT

TABLE CREATE QUERY

```
CREATE TABLE `studdata` (
 `email` varchar(100) NOT NULL,
 `fname` tinytext DEFAULT NULL,
 `lname` varchar(20) DEFAULT NULL,
 `address` varchar(40) DEFAULT NULL,
 `gender` char(1) DEFAULT NULL,
 `age` int(11) DEFAULT NULL,
 `dob` date DEFAULT NULL,
 `phone` bigint(20) DEFAULT NULL,
 `nationality` tinytext DEFAULT NULL,
 `pwd` longtext DEFAULT NULL,
 PRIMARY KEY (`email`));
```

3. IMAGE

Attributes	Data Types
studentid	INT(11)
imagename	LONGTEXT
status	INT(1)

TABLE CREATE QUERY

```
CREATE TABLE `image` (
 `studentid` int(11) NOT NULL,
 `imagename` longtext DEFAULT NULL,
 `status` int(1) DEFAULT NULL,
 PRIMARY KEY (`studentid`));
```

4. ACADEMICS

Attributes	Data Types
studentid	INT(11)
dob	DATE
maths	INT(3)
physics	INT(3)
chemistry	INT(3)
cutoff	DECIMAL(5,2)

TABLE CREATE QUERY

```
CREATE TABLE `academics` (
 `studentid` int(11) NOT NULL,
 `dob` date DEFAULT NULL,
 `maths` int(3) DEFAULT NULL,
 `physics` int(3) DEFAULT NULL,
 `chemistry` int(3) DEFAULT NULL,
 `cutoff` decimal(5,2) DEFAULT NULL,
 PRIMARY KEY (`studentid`),
 FOREIGN KEY (`studentid`) REFERENCES `student` (`studentid`));
```

5. LOCKCHOICES

Attributes	Data Types
studentid	INT(11)
lockchoice	INT(2)

TABLE CREATE QUERY

```
CREATE TABLE `lockchoices` (
 `studentid` int(11) NOT NULL,
 `lockchoice` int(2) DEFAULT 0,
 PRIMARY KEY (`studentid`),
 FOREIGN KEY (`studentid`) REFERENCES `student` (`studentid`));
```

6. CHOICE

Attributes	Data Types
studentid	INT(11)
collegecode	CHAR(6)
coursecode	CHAR(6)
choiceno	INT(11)

TABLE CREATE QUERY

```
CREATE TABLE `choice` (
 `studentid` int(11) NOT NULL,
 `collegecode` char(6) NOT NULL,
 `coursecode` char(6) NOT NULL,
 `choiceno` int(11) DEFAULT NULL,
 PRIMARY KEY (`studentid`,`collegecode`,`coursecode`),
 KEY `collegecode` (`collegecode`),
 KEY `coursecode` (`coursecode`),
 FOREIGN KEY (`studentid`) REFERENCES `student` (`studentid`),
 FOREIGN KEY (`collegecode`) REFERENCES `college` (`collegecode`),
 FOREIGN KEY (`coursecode`) REFERENCES `course` (`coursecode`));
```

7.ADMITTED

Attributes	Data Types
studentid	INT(11)
collegecode	CHAR(6)
coursecode	CHAR(6)

TABLE CREATE QUERY

```
CREATE TABLE `admitted` (
 `studentid` int(11) NOT NULL,
 `collegecode` char(6) DEFAULT NULL,
 `coursecode` char(6) DEFAULT NULL,
 PRIMARY KEY (`studentid`),
 KEY `collegecode` (`collegecode`),
 KEY `coursecode` (`coursecode`),
 FOREIGN KEY (`studentid`) REFERENCES `student` (`studentid`),
 FOREIGN KEY (`collegecode`) REFERENCES `college` (`collegecode`),
 FOREIGN KEY (`coursecode`) REFERENCES `course` (`coursecode`));
```

8.REJECTED

Attributes	Data Types
studentid	INT(11)
rank	BIGINT(20)

TABLE CREATE QUERY

```
CREATE TABLE `rejected` (
 `studentid` int(11) NOT NULL,
 `rank` bigint(20) NOT NULL,
 PRIMARY KEY (`studentid`),
 FOREIGN KEY (`studentid`) REFERENCES `student` (`studentid`));
```

9.COLLEGE

Attributes	Data Types
collegecode	CHAR(6)
collegename	VARCHAR(50)
dean	VARCHAR(30)
city	VARCHAR(30)

TABLE CREATE QUERY

```
CREATE TABLE `college` (
 `collegecode` char(6) NOT NULL,
 `collegename` varchar(50) DEFAULT NULL,
 `dean` varchar(30) DEFAULT NULL,
 `city` varchar(30) DEFAULT NULL,
 PRIMARY KEY (`collegecode`));
```

10. COURSE

Attributes	Data Types
coursecode	CHAR(6)
coursename	VARCHAR(50)
coursetag	VARCHAR(4)
coursedescription	LONGTEXT

TABLE CREATE QUERY

```
CREATE TABLE `course` (
 `coursecode` char(6) NOT NULL,
 `coursename` varchar(50) DEFAULT NULL,
 `coursetag` varchar(4) DEFAULT NULL,
 `coursedescription` longtext DEFAULT NULL,
 PRIMARY KEY (`coursecode`));
```

11.HAS

Attributes	Data Types
collegecode	CHAR(6)
coursecode	CHAR(6)
totalseats	INT(11)
available	INT(11)

TABLE CREATE QUERY

```
CREATE TABLE `has` (
 `collegecode` char(6) NOT NULL,
 `coursecode` char(6) NOT NULL,
 `totalseats` int(11) DEFAULT NULL,
 `available` int(11) DEFAULT NULL,
 PRIMARY KEY (`collegecode`,`coursecode`),
 FOREIGN KEY (`collegecode`) REFERENCES `college` (`collegecode`),
 FOREIGN KEY (`coursecode`) REFERENCES `course` (`coursecode`));
```

12.ADMIN

Attributes	Data Types
adminid	INT(11) AUTO INCREMENT
adminname	VARCHAR(20)
pwd	LONGTEXT

TABLE CREATE QUERY

```
CREATE TABLE `admin` (
 `adminid` int(11) NOT NULL AUTO_INCREMENT,
 `adminname` varchar(20) DEFAULT NULL,
 `pwd` longtext DEFAULT NULL,
 PRIMARY KEY (`adminid`));
```

The following tables are for website design and is not for relation purposes. It stores data for the website

13.Views

Attributes	Data Types
page	VARCHAR(20)
view	INT(11)

TABLE CREATE QUERY

```
CREATE TABLE `views` (
 `page` varchar(20) NOT NULL,
 `view` int(11) DEFAULT NULL,
 PRIMARY KEY (`page`));
```

14.WEBDATA

Attributes	Data Types
id	VARCHAR(20)
value	INT(11)

TABLE CREATE QUERY

```
CREATE TABLE `webdata` (
 `id` varchar(20) NOT NULL,
 `value` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`));
```

VIEWS DESCRIPTION

1.GRADE

Attributes	Data Types
studentid	INT(11)
rank	BIGINT(21)

VIEW CREATE QUERY

```
CREATE VIEW grade AS
select academics.studentid AS studentid,
rank() over
    ( order by academics.cutoff desc,
    academics.maths desc,
    academics.physics desc,
    academics.chemistry desc,
    academics.dob desc) AS `rank`
from academics;
```

STORED PROCEDURES DESCRIPTION

1.Allocation

The procedure code

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `allocation`()
    MODIFIES SQL DATA
    DETERMINISTIC
    SQL SECURITY INVOKER
BEGIN

--declare all the necessary variables
     DECLARE college_code CHAR(6);
     DECLARE course_code CHAR(6);
     DECLARE student_id INT(11);
     DECLARE available_seats INT(11);
      DECLARE admitted_count INT(11);
     DECLARE is_done INTEGER DEFAULT 0;

--declare the rank_student cursor
     DECLARE rank_student CURSOR FOR
         SELECT grade.studentid
              ,choice.collegecode
              ,choice.coursecode
         FROM grade,choice
         WHERE grade.studentid=choice.studentid
         ORDER BY grade.rank,choice.choiceno;
```

```
--declare not found handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_done = 1;
--declare transaction fail handler
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
     ROLLBACK;
    END;
--open cursor
    OPEN rank_student;
--start transaction
START TRANSACTION;
--set all lockchoice to 1
     UPDATE lockchoices SET lockchoice=1;
--start loop
     allocation:LOOP
--fetching from cursor
     FETCH rank_student INTO
             student_id,
             college_code,
             course_code;
--check for not found
        IF is_done=1 THEN
          LEAVE allocation;
        END IF;
--get available seats for that choice
        SELECT has.available
            INTO available_seats
            FROM has
            WHERE has.collegecode=college_code AND
                has.coursecode=course_code;
```

```
--check for available seats
        IF available_seats=0 THEN
          ITERATE allocation;
        END IF;
--get student is already admitted
        SELECT COUNT(admitted.studentid)
            INTO admitted_count
            FROM admitted
            WHERE admitted.studentid=student_id;
--check if already that student admitted
        IF admitted_count>0 THEN
          ITERATE allocation;
        END IF;
--insert that choice into admitted
        INSERT INTO admitted VALUES
          (student_id,college_code,course_code);
-end loop
    END LOOP allocation;
--close cursor
    CLOSE rank_student;
--end transaction
COMMIT;
END$$
DELIMITER ;
```

Description

This procedure uses a cursor to get choices ordered by rank and by choiceno. Then it fetches each row and check for available seats for each choice. If available choice is greater than 0, it checks for if student is already admitted or not. If not admitted, the student's choice will be allocated to him by inserting it in the allocation table. This procedure also uses transaction because all these tasks needed to be done without interruption.

2.Rejected_adder

The procedure code

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `rejected_adder`()
    MODIFIES SQL DATA
BEGIN
--declare necessary files
    DECLARE student_id INT(11);
    DECLARE rank BIGINT;
    DECLARE is_done INTEGER DEFAULT 0;
--declare cursor cursorrejected
    DECLARE cursorrejected CURSOR FOR
     SELECT studentid,rank
         FROM student,grade
         WHERE student.studentid=grade.studentid
         AND studentid NOT IN
         (SELECT studentid FROM admitted);
--declare not found handler
     DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_done = 1;

--open cursor
```

```
    OPEN cursorrejected;
--start loop
    rejection:LOOP
--fetch from cursor
         FETCH cursorrejected INTO student_id,rank;
--check for not found
        IF is_done=1 THEN
         LEAVE rejection;
        END IF;
--insert into rejected table
        INSERT INTO rejected VALUES(student_id,rank);
--end loop
    END LOOP rejection;
--close cursor
    CLOSE cursorrejected;
END$$
DELIMITER ;
```

Description

This procedure uses a cursor that gets all students whose all choices were rejected. Then the procedure fetch from cursor and insert the row into rejected table. This table will be used to identify rejected students. This procedure will be called by allocation procedure

3.Delete_student

The procedure code

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `delete_student`(IN `id`
INT(11))
    MODIFIES SQL DATA
    SQL SECURITY INVOKER
BEGIN
     DECLARE email_delete VARCHAR(100);
    SELECT student.email INTO email_delete FROM student WHERE
     student.studentid=id;
     DELETE FROM academics WHERE studentid=id;
    DELETE FROM image WHERE studentid=id;
    DELETE FROM choice WHERE studentid=id;
    DELETE FROM admitted WHERE studentid=id;
    DELETE FROM rejected WHERE studentid=id;
    DELETE FROM lockchoices WHERE studentid=id;
    DELETE FROM student WHERE studentid=id;
    DELETE FROM studdata WHERE email=email_delete;
END
DELIMITER ;
```

Description

This procedure gets an id as an input argument. Then it deletes the row with that id from all the tables such as academics, image, studdata, choice, admitted, rejected, lockchoices and student.

4.Reset

The procedure code

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `reset`
   (IN `college_delete` INT(2))
    NO SQL
BEGIN
    DELETE FROM academics;
    DELETE FROM admitted;
    DELETE FROM choice;
    UPDATE has SET available=totalseats;
    DELETE FROM image;
    DELETE FROM lockchoices;
    DELETE FROM rejected;
    UPDATE views SET view='0';
    UPDATE webdata SET value='0';
    DELETE FROM student;
    DELETE FROM studdata;
    IF college_delete=1 THEN
        DELETE FROM has;
     DELETE FROM college;
        DELETE FROM course;
    END IF;
END$$
DELIMITER ;
```

Description

This procedure takes a college_delete as input. It truncates and updates all the tables back to its original values. If college_delete was 1, then it will truncate all college related tables as well.

FUNCTIONS DESCRIPTION

1.College_details

The function code is

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION
`college_details`(`course_code` CHAR(6)) RETURNS longtext CHARSET utf8
    READS SQL DATA
BEGIN
--declare all the necessary variables
    DECLARE return_val LONGTEXT DEFAULT '';
    DECLARE college_name VARCHAR(50);
    DECLARE college_code CHAR(6);
    DECLARE is_done INTEGER DEFAULT 0;
--declare cursor for college_info
    DECLARE college_info CURSOR FOR
     SELECT college.collegecode,college.collegename
        FROM college,has
        WHERE has.coursecode=course_code AND
        has.collegecode=college.collegecode;
--declare not found handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_done = 1;
```

```
--open cursor
    OPEN college_info;
--start loop
    college:LOOP
--fetch cursor
     FETCH college_info INTO
          college_code,college_name;
--check for not found
        IF is_done=1 THEN
         LEAVE college;
        END IF;
--concat in a format like CLG001:A College
        SELECT CONCAT(return_val,college_code,':',
                          college_name,';') INTO return_val;
--end loop
    END LOOP college;
--close cursor
    CLOSE college_info;
--return value
    RETURN return_val;
END$$
DELIMITER ;
```

Description

This function takes a course_code as an input and uses a cursor to find all the course available colleges details and produces a string that contains all the course available colleges' collegecode and collegename in a uniform format. This function returns that string

2.Course_details

The function code is

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION
`course_details`(`college_code` CHAR(6)) RETURNS longtext CHARSET utf8
    READS SQL DATA
BEGIN
--declare all necessary variables
    DECLARE return_val LONGTEXT DEFAULT '';
    DECLARE course_name VARCHAR(50);
    DECLARE course_tag VARCHAR(4);
    DECLARE is_done INTEGER DEFAULT 0;
--declare cursor course_info
    DECLARE course_info CURSOR FOR
     SELECT course.coursetag,course.coursename
         FROM course,has
         WHERE has.collegecode=college_code AND
           has.coursecode=course.coursecode;
--declare not found handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_done = 1;
--open cursor
    OPEN course_info;
--start loop
    course:LOOP
--fetch from cursor
     FETCH course_info INTO
          course_tag,course_name;
```

```
--check for not found
        IF is_done=1 THEN
          LEAVE course;
        END IF;
--concat the result in a format CSE:Computer Science Engineering
        SELECT CONCAT(return_val,course_tag,':',
                            course_name,';') INTO return_val;
--end loop
    END LOOP course;
--close cursor
    CLOSE course_info;
--return value
    RETURN return_val;
END$$
DELIMITER ;
```

TRIGGERS DESCRIPTION

1.student_init

The trigger code was

```
CREATE TRIGGER `student_init`
AFTER INSERT ON `student`
FOR EACH ROW
BEGIN
    INSERT INTO image(studentid,imagename,status)
    VALUES(NEW.studentid,'profiledefault.jpg','0');
    INSERT INTO lockchoices(studentid,lockchoice)
    VALUES(NEW.studentid,'0');
END
```

Description

This trigger get triggered after an insert on student table. This table inserts new rows in tables image and lockchoices for each new student. The image table stores the default profile picture name and lockchoices shows that the choices were not locked initially.

2.Available_insert

The trigger code is

```
CREATE TRIGGER `available_insert`
AFTER INSERT ON `admitted`
FOR EACH ROW
BEGIN
        UPDATE has
        SET available = available-1
        WHERE   collegecode = new.collegecode
        AND coursecode = new.coursecode;
END
```

Description

This trigger gets triggered when an insert is happened on admitted table. When a row is inserted, the available seats in has table needs to be decreased so that the available seats can go as far as 0. After that, no more students can be added for that course.

3.Available_delete

The trigger code is

```
CREATE TRIGGER `available_delete`
AFTER DELETE ON `admitted`
FOR EACH ROW
BEGIN
        UPDATE has
        SET available = available+1
        WHERE   collegecode = old.collegecode
        AND coursecode = old.coursecode;
END
```

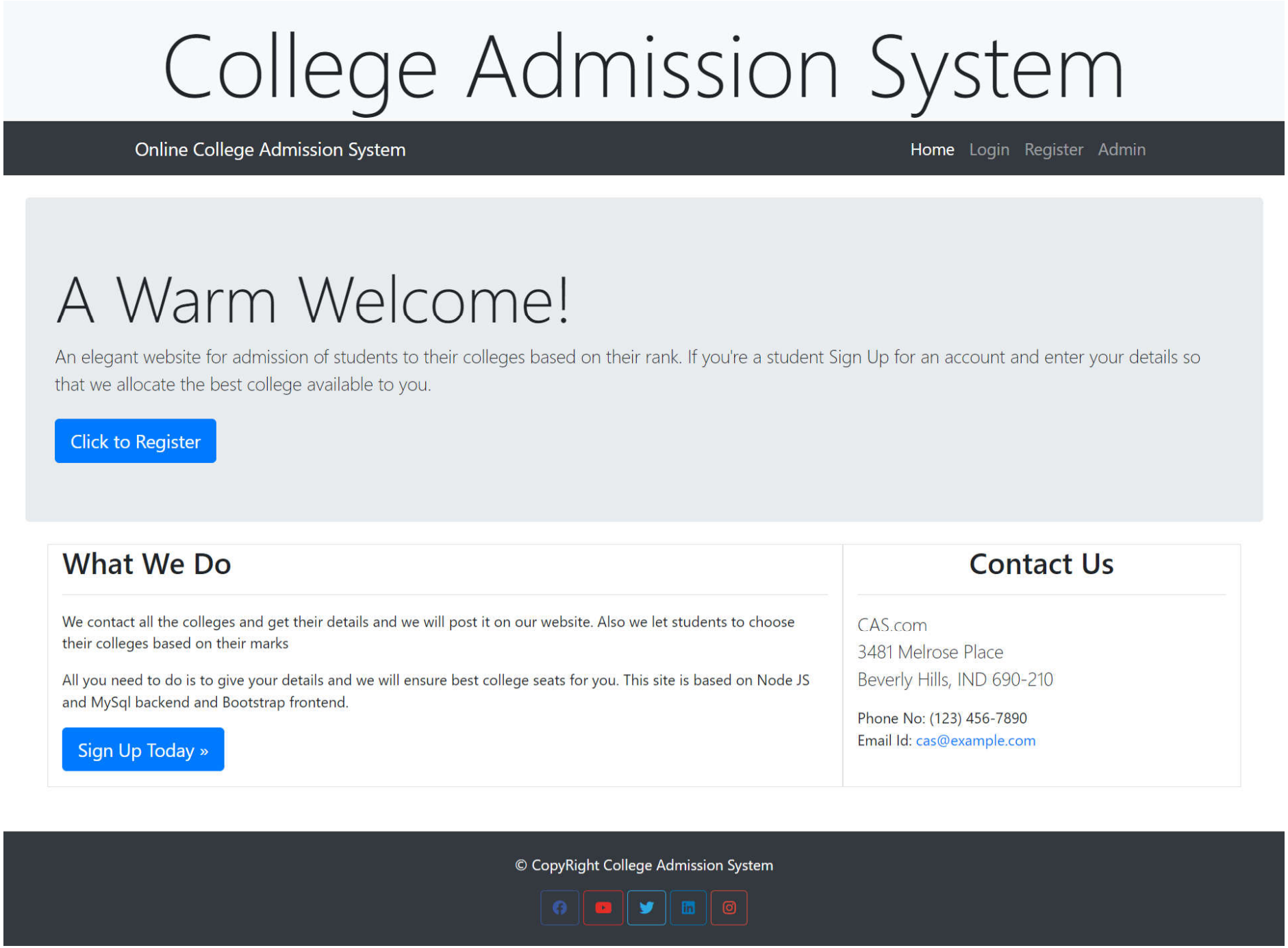
Description

This trigger gets triggered when there is an delete on admitted table. If a row is deleted, then the available seats for that particular course needs to be increased. This trigger ensures that property so that new students can be added to this course.

WEBSITE DESCRIPTION

Each webpage description is given as a screen shot of the page, the function of the page and the queries used in the pages.

HOME PA


College Admission System
Online College Admission System
Home Login Register Admin
A Warm Welcome!
An elegant website for admission of students to their colleges based on their rank. If you're a student Sign Up for an account and enter your details so that we allocate the best college available to you.
Click to Register
What We Do
We contact all the colleges and get their details and we will post it on our website. Also we let students to choose their colleges based on their marks
All you need to do is to give your details and we will ensure best college seats for you. This site is based on Node JS and MySql backend and Bootstrap frontend.
Sign Up Today »
Contact Us
CAS.com
3481 Melrose Place
Beverly Hills, IND 690-210
Phone No: (123) 456-7890
Email Id: cas@example.com
© CopyRight College Admission System

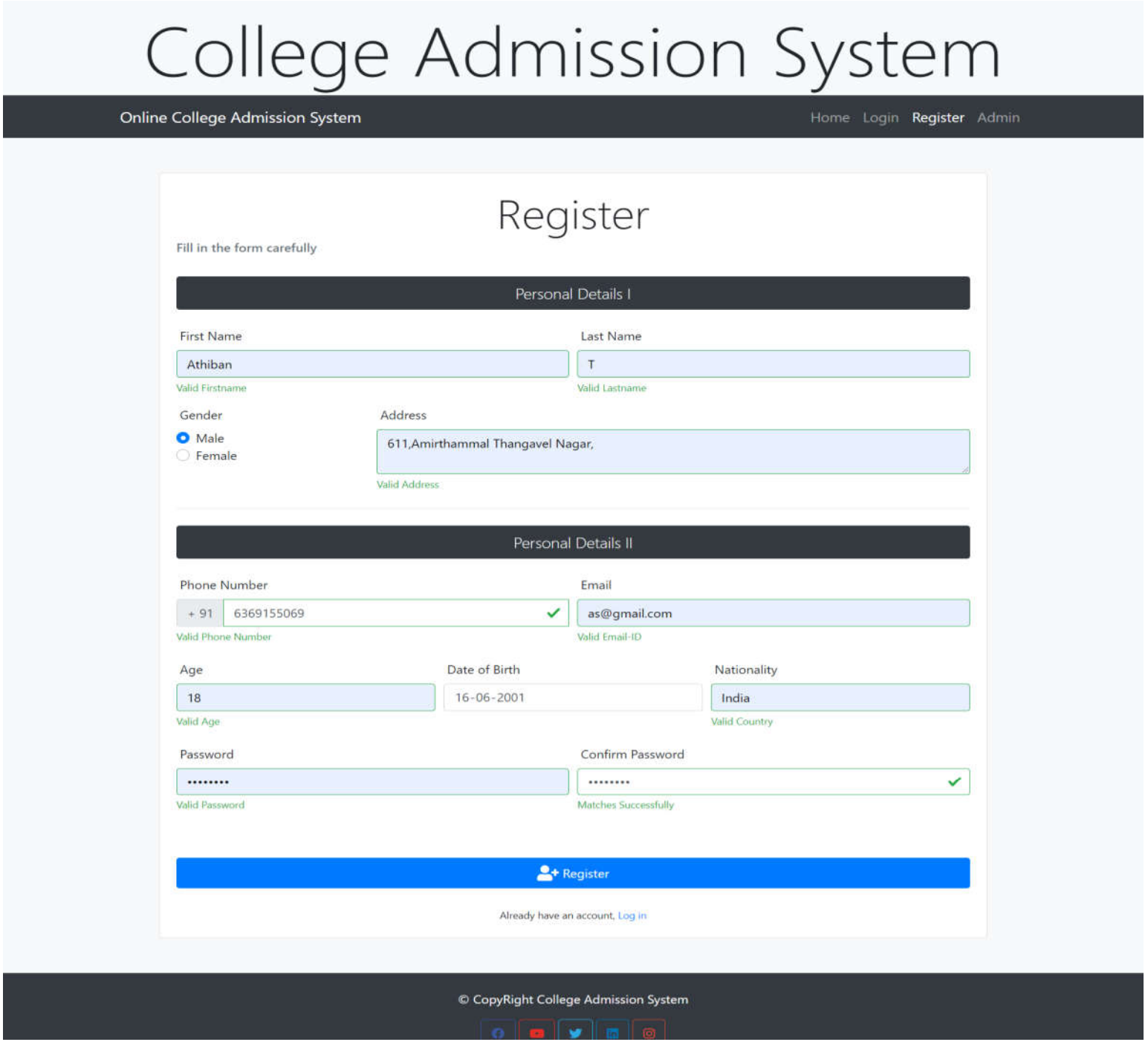
PAGE DESCRIPTION

This page is kind of an introduction to the website and it has all the necessary links in the navigation bar for the users. The links available are home, login, register and admin. The home page is the current page. The login page is for the students who have registered. The register page is for the students who need to register as a student for this page. The admin page is for the page admin to login.

Query Format

? defines values from the form or input

REGISTER PAGE



PAGE DESCRIPTION

This page is for the students who need to register themselves as the user of this page. It takes all the necessary information and creates an account for that student.

QUERIES USED

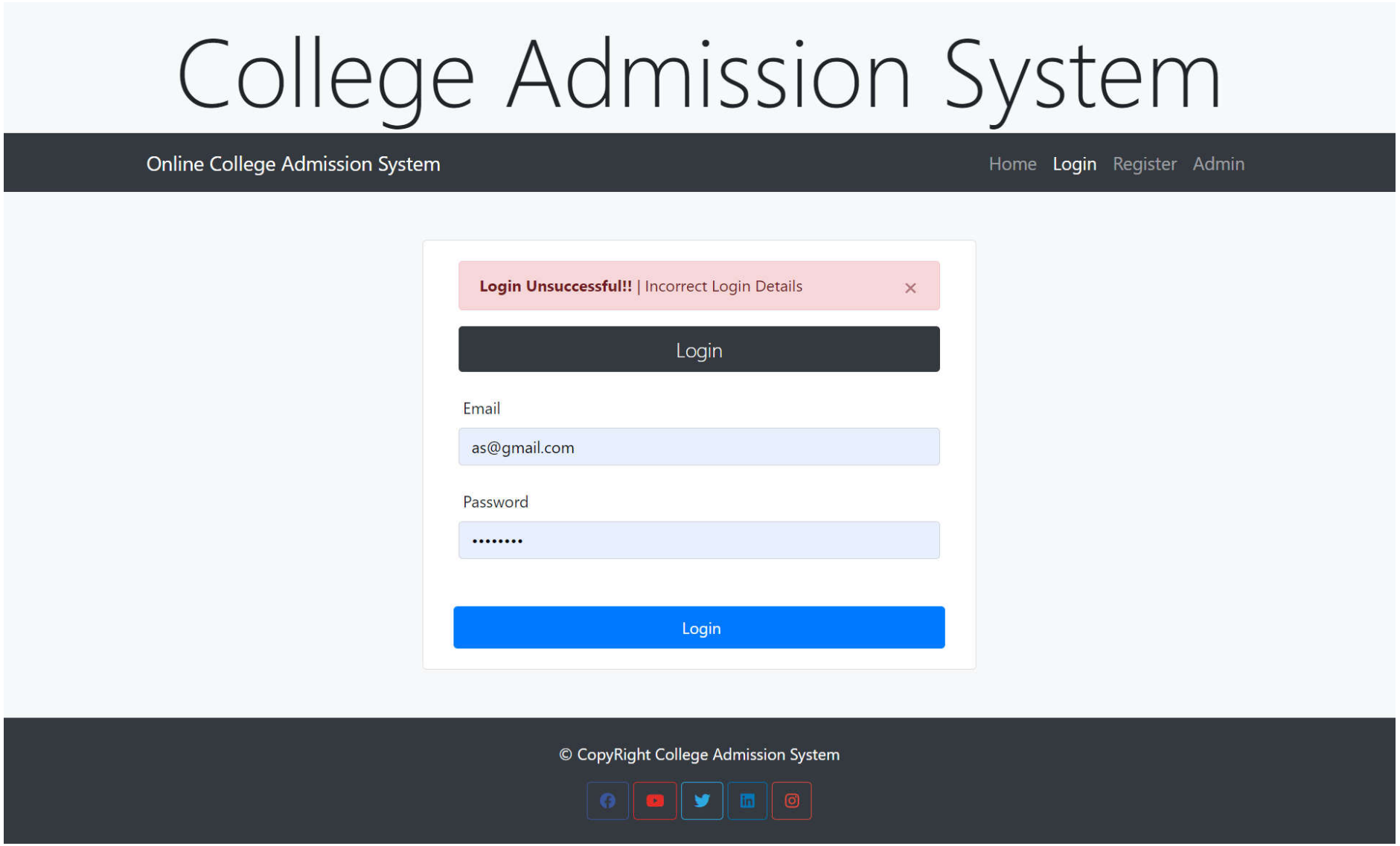
1.Check for an existing account with that email

```
SELECT * FROM studdata,student
WHERE studdata.email=
     (select email FROM student WHERE studentid=? or email=?);
```

2.Register a student by inserting

```
INSERT INTO studdata
     (email,fname,lname,address,gender,age,dob,nationality,pwd)
     VALUES(?,?,?,?,?,?,?,?,?,?);
INSERT INTO student(email) VALUES (?);
```

LOGIN PAGE


College Admission System
Online College Admission System
Home
Login
Register
Admin
Login Unsuccessful!! | Incorrect Login Details
Login
Email
as@gmail.com
Password
Login
© CopyRight College Admission System

PAGE DESCRIPTION

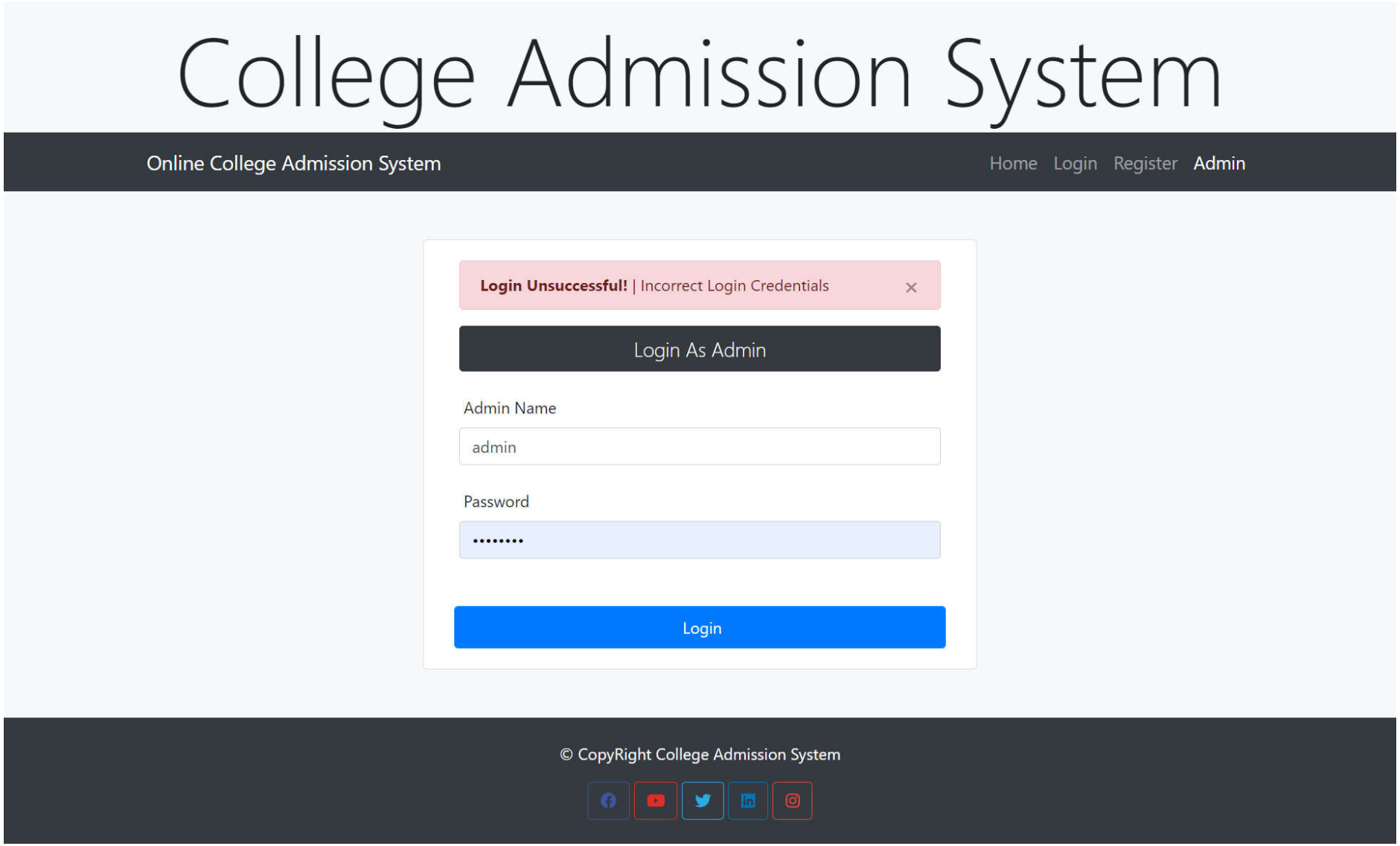
This page gets the email and password for registered user and check for account availability and redirect them to the students home page.

QUERIES USED

1.Check for account availability

```
SELECT * FROM studdata,student
WHERE studdata.email=
(select email FROM student WHERE studentid=? or email=?);
```

ADMIN PAGE


College Admission System
Online College Admission System
Home Login Register Admin
Login Unsuccessful! | Incorrect Login Credentials
Login As Admin
Admin Name
admin
Password
Login
© CopyRight College Admission System

PAGE DESCRIPTION

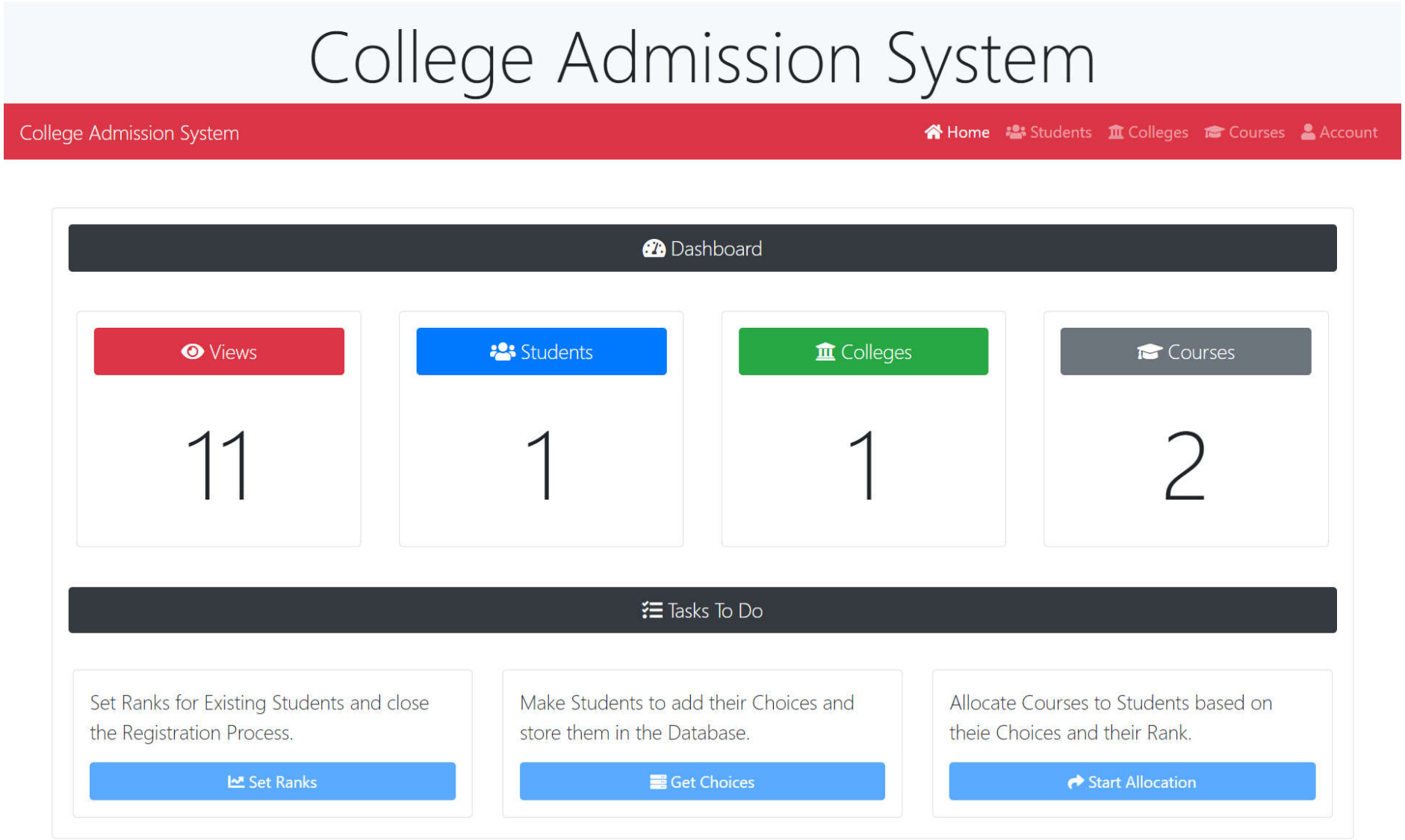
This page takes the admin name and password as input and checks for an admin account with that name and password and redirects to the admin home page.

QUERIES USED

1.Get an account with that name and password

```
SELECT * FROM admin WHERE adminname=? AND pwd=?;
```

ADMIN HOME PAGE



PAGE DESCRIPTION

This page shows all the counts of different tables and also sets the content of the webdata table.It also resets the database and allocates seats.

QUERIES USED

1.Get the sum of views table

```
SELECT SUM(view) AS count FROM views
```

2.Get the count of students

```
SELECT COUNT(studentid) AS count FROM student;
```

3.Get the count of colleges

```
SELECT COUNT(collegecode) AS count FROM college;
```

4.Get the count of courses

```
SELECT COUNT(coursecode) AS count FROM course;
```

5.Update Webdata Table Content in tasks to-do

```
UPDATE webdata SET value=1 WHERE id=?;
```

6.Reset the database

```
CALL reset(?);
```

7.Allocates Seats to students

```
CALL allocation();
```

SEARCH STUDENT PAGE

College Admission System

College Admission System | Home | Students | Colleges | Courses | Account

Search For a particular Student

Type The Student's First Name

ath

Athiban T ,

Student ID	1	**Address**	611,Amirthammal Thangavel NagarOrikkai.
First Name	Athiban	**Gender**	Male
Last Name	T	**Date Of Birth**	2000-12-11
Email ID	as@gmail.com	**Phone Number**	6369155069
Age	18	**Nationality**	India
Maths	198	**Chemistry**	200
Physics	197	**Cut-Off**	198.75

PAGE DESCRIPTION

In this page, if you type student's first name you get their profile if it is found. Even there is multiple students with the same name it shows all the students profile.

QUERIES USED

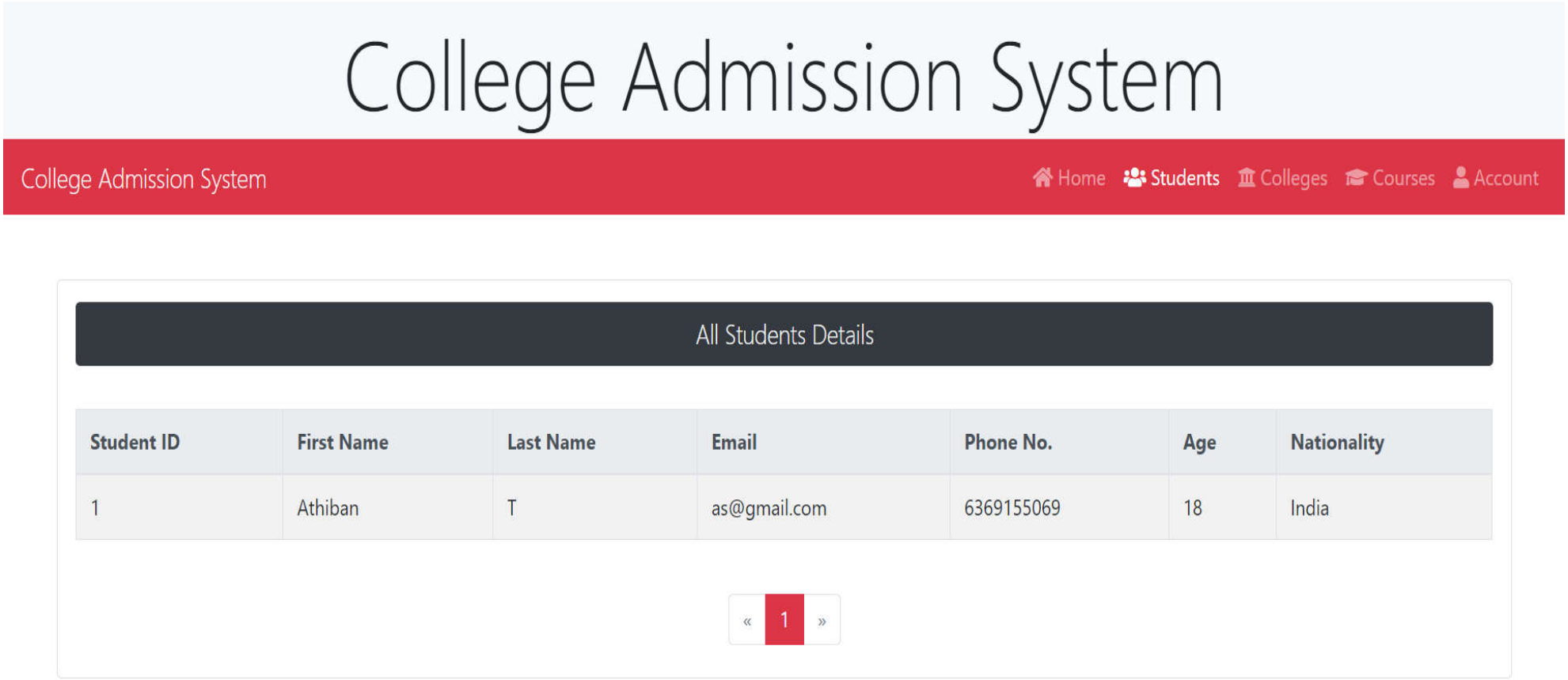
1.Get student data with the part of first name

```
SELECT *,LOWER(fname),INSTR(LOWER(fname),?) FROM student,studdata
     WHERE INSTR(LOWER(fname),?)>0
     AND student.email=studdata.email
     ORDER BY INSTR(LOWER(fname),?);
```

2.Get academics data for that student

```
SELECT * FROM academics WHERE studentid=?;
```

ALL STUDENTS PAGE



PAGE DESCRIPTION

This page shows the details of all the registered students 5 results at a time. Depending on the page numbers the results will vary and the results are ordered by studentid.

QUERIES USED

1.Get the students 5 at a time

```
SELECT *
FROM student,studdata
WHERE student.email=studdata.email
ORDER BY studentid LIMIT ?,5;
```

DELETE STUDENT PAGE

College Admission System

College Admission System | Home | Students | Colleges | Courses | Account

Delete Student By their ID

1 | Delete

Athiban T ,

Are You Sure You Want To Delete This Profile? | Yes | No

Student ID	1	**Address**	611,Amirthammal Thangavel NagarOrikkai.
First Name	Athiban	**Gender**	Male
Last Name	T	**Date Of Birth**	2000-12-11
Email ID	as@gmail.com	**Phone Number**	6369155069
Age	18	**Nationality**	India
Maths	198	**Chemistry**	200
Physics	197	**Cut-Off**	198.75

PAGE DESCRIPTION

This page get a student id as input and show the students profile with two options. If yes is clicked, the student will be removed. If no is clicked, new form gets displayed

QUERIES USED

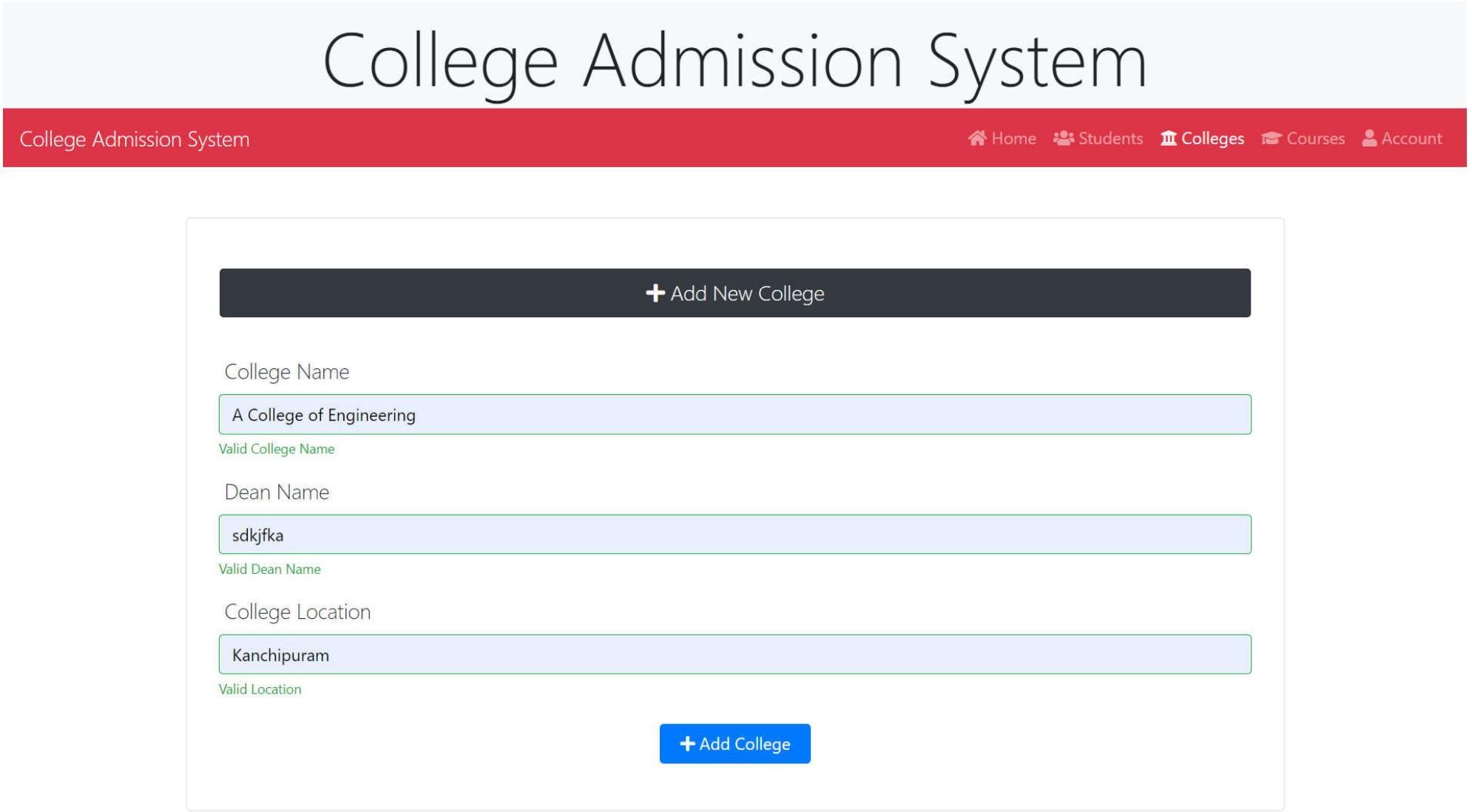
1.Get student profile by id

```
SELECT * FROM student,studdata WHERE studentid=? AND student.email=stu
ddata.email;
```

2.Delete Student Data

```
CALL delete_student(?);
```

ADD COLLEGE PAGE



PAGE DESCRIPTION

This page gets all the information needed to add the college to the colleges table. It checks for the entered strings length before inserting.

QUERIES USED

1.Check for the college already inserted

```
SELECT COUNT(collegecode) FROM college WHERE collegename=?;
```

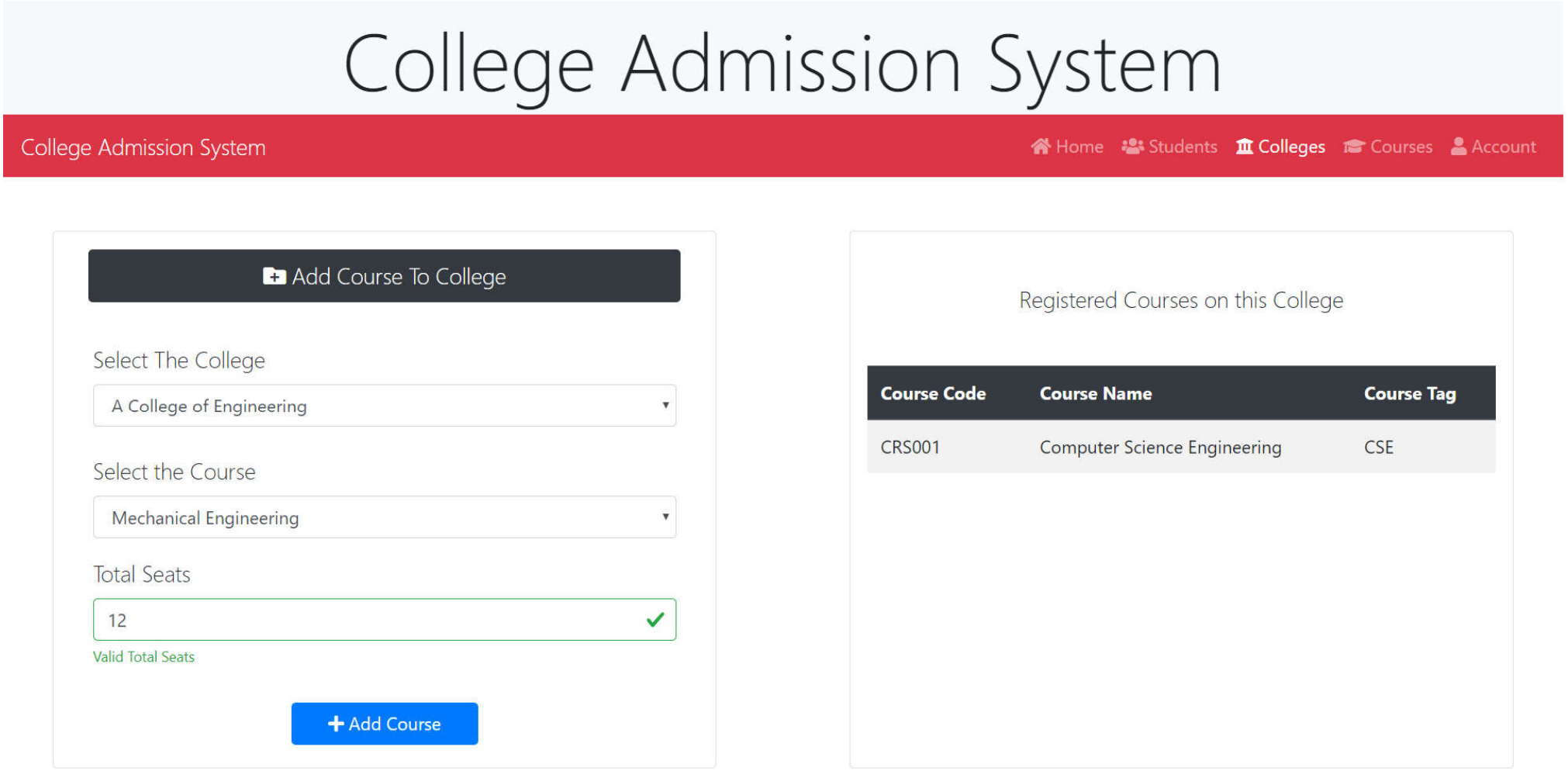
2.Insert college into the table

```
INSERT INTO college VALUES(?,?,?,?);
```

3.Find the number of colleges

```
SELECT COUNT(collegecode) AS count FROM college;
```

ADD COURSE TO COLLEGE PAGE



PAGE DESCRIPTION

This page has two cards. On the left card, if you select a college the courses that are not in that college. If you enter the total seats then the course will be added to the college. On the right card, the courses that are currently available in the college are shown.

QUERIES USED

1.Get all the colleges

```
SELECT * FROM COLLEGES
```

2.Find the courses available on that college

```
SELECT course.coursecode,coursename,coursetag,coursedescription
FROM has,course
WHERE has.collegecode=? AND has.coursecode=course.coursecode
```

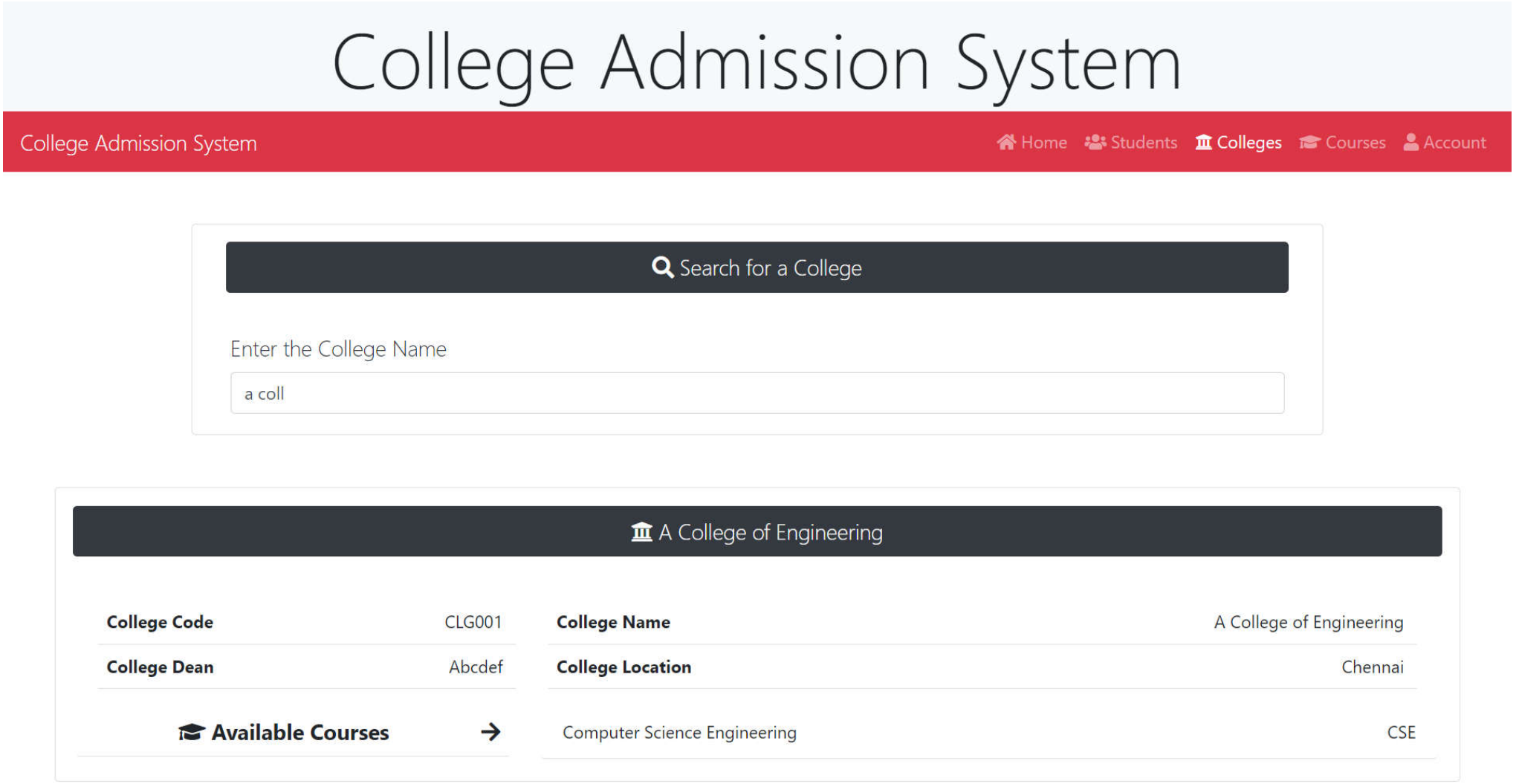
3.Find the remaining courses not in that college

```
SELECT course.coursecode,coursename,coursetag,coursedescription
FROM course
WHERE course.coursecode NOT IN
     (SELECT coursecode FROM has WHERE collegecode=?);
```

4.Add the course into the college

```
INSERT INTO has VALUES(?,?,?,?);
```

SEARCH COLLEGE PAGE



PAGE DESCRIPTION

In this page if you type the college name then the college details will be shown along the courses available on the college. If there are multiple colleges then all colleges will be shown.

QUERIES USED

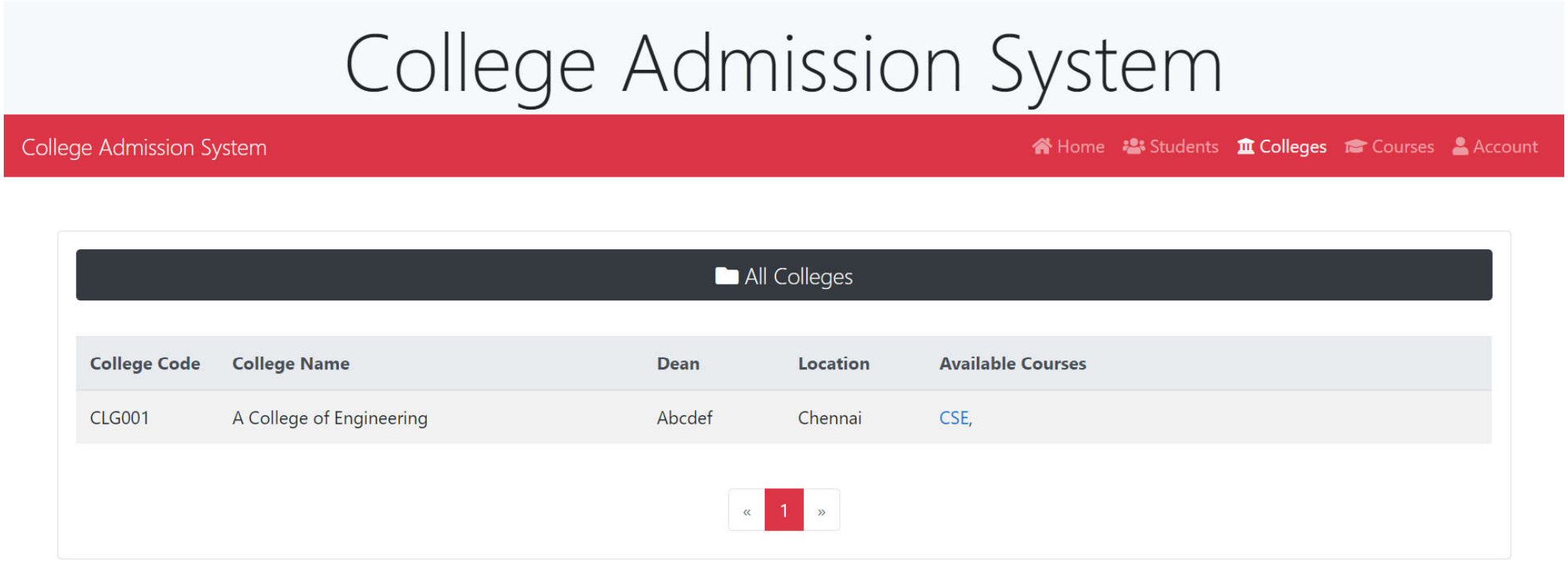
1.Find the college by using part of college name string

```
SELECT *,LOWER(collegename),INSTR(LOWER(collegename),?)
FROM college
WHERE INSTR(LOWER(collegename),?)>0
ORDER BY INSTR(LOWER(collegename),?);
```

2.Find available courses for a college

```
SELECT course.coursecode,coursename,coursetag,coursedescription
FROM has,course
WHERE has.collegecode=? AND has.coursecode=course.coursecode;
```

ALL COLLEGES PAGE



PAGE DESCRIPTION

This page shows all the colleges 5 results per page along with their available courses.

QUERIES USED

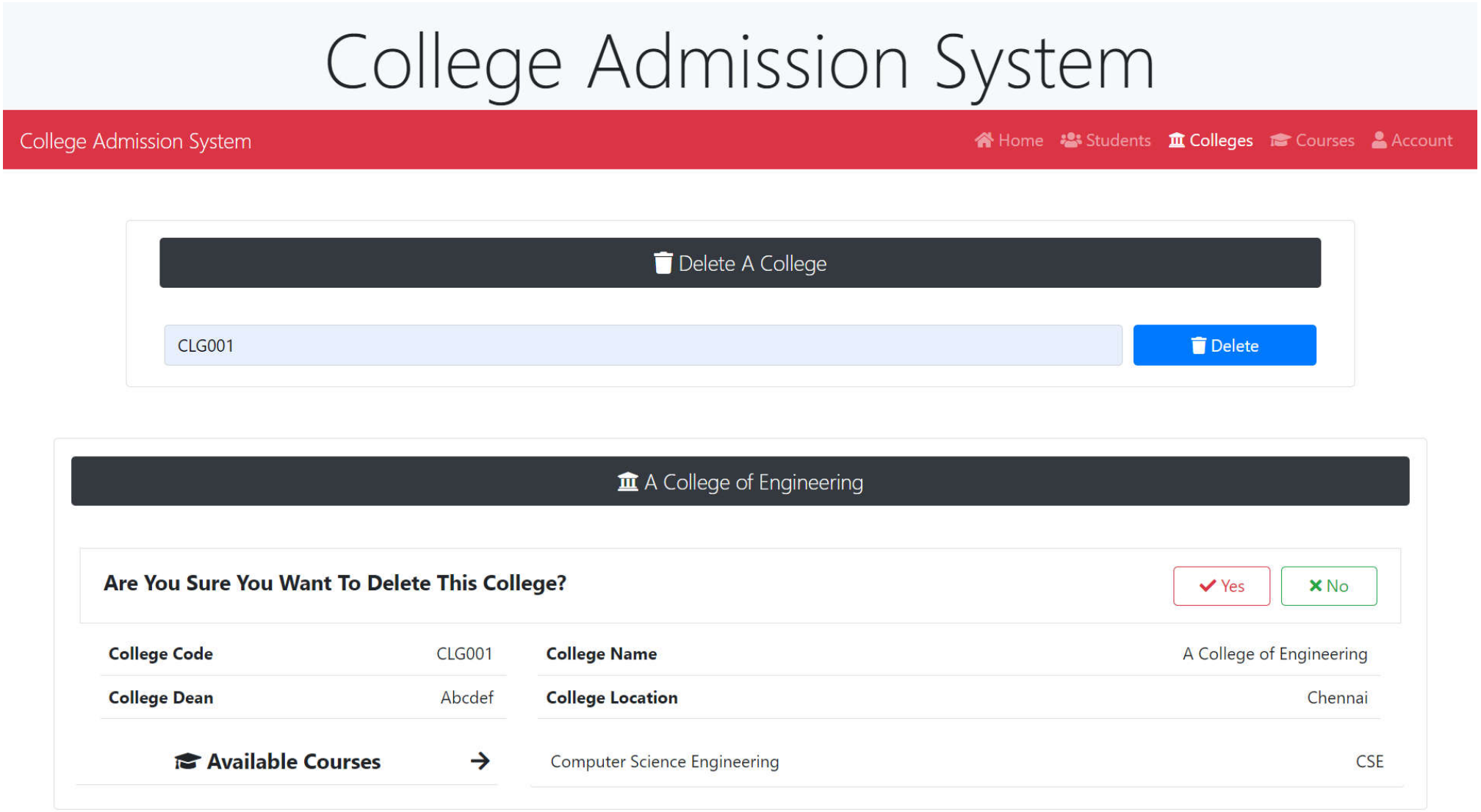
1.Get all colleges by limit and offset

```
SELECT * FROM college ORDER BY collegecode LIMIT ?,5;
```

2.Get available courses for the college

```
SELECT course_details(?) AS course_string;
```

DELETE COLLEGE PAGE



PAGE DESCRIPTION

This page gets collegecode as input, then the page shows the college details with two options. If yes is clicked the college will be deleted and all the courses associated with the college is also deleted from the has table. If no is clicked then the page is ready for another college deletion.

QUERIES USED

1. Get college by collegecode

```
SELECT * FROM college WHERE collegecode=?;
```

2. Find available courses from a College

```
SELECT course.coursecode,coursename,coursetag,coursedescription
FROM has,course
WHERE has.collegecode=? AND has.coursecode=course.coursecode;
```

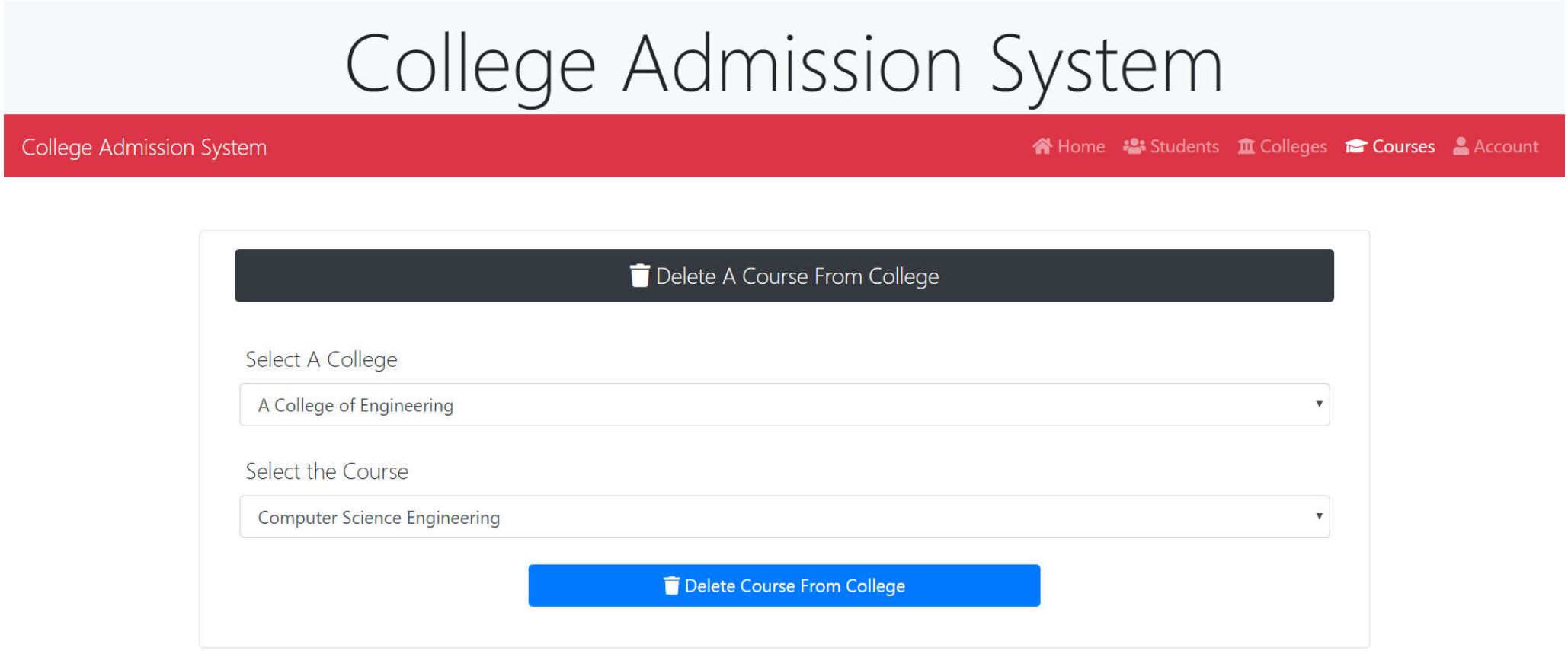
3.Delete college from has table

```
DELETE FROM has WHERE collegecode=?;
```

4. Delete college from college table

```
DELETE FROM college WHERE collegecode=?;
```

DELETE COURSE FROM COLLEGE PAGE



PAGE DESCRIPTION

This page gets a college and the course as input and deletes that course from that college.

QUERIES USED

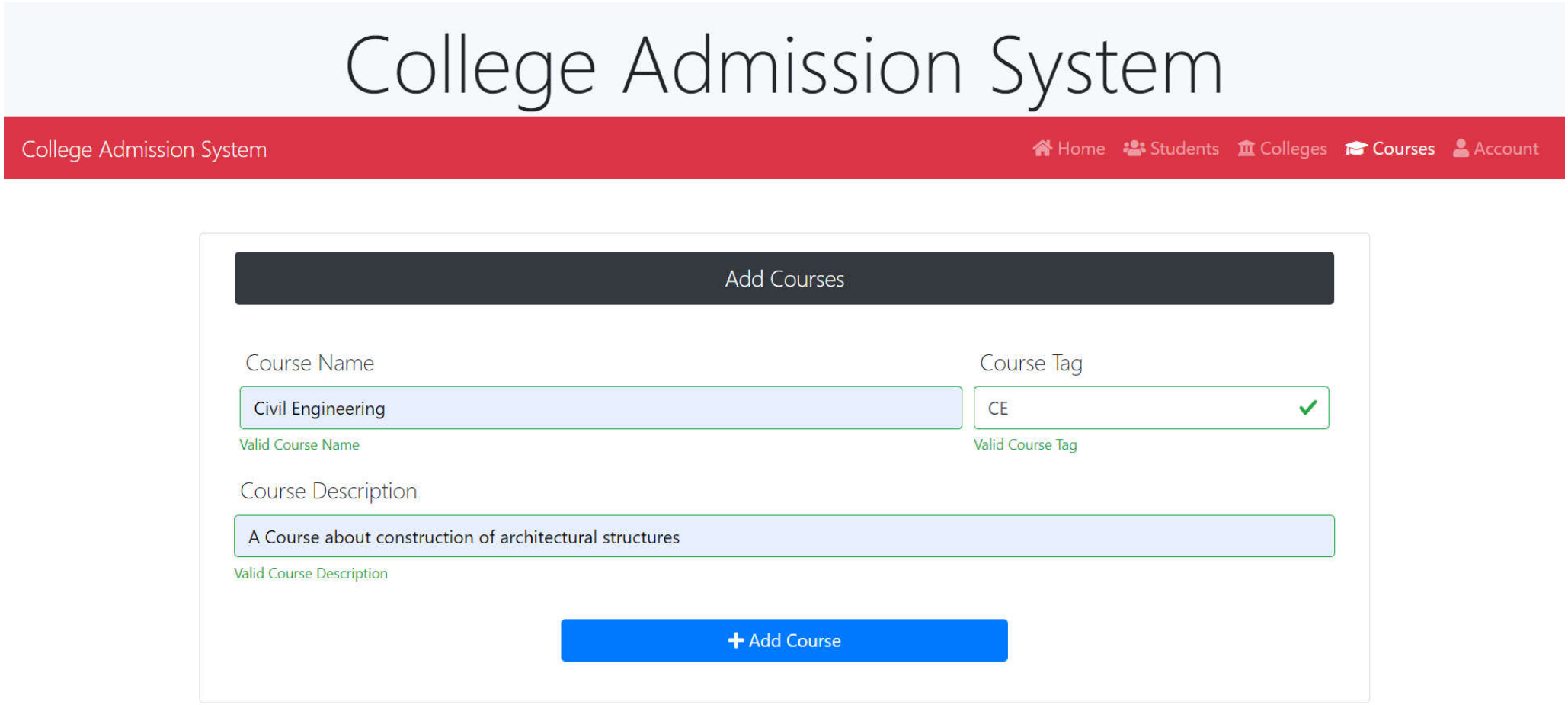
1.Get all colleges

```
SELECT * FROM COLLEGES;
```

2.Get courses available on that college

```
SELECT course.coursecode,coursename,coursetag,coursedescription
FROM course
WHERE course.coursecode NOT IN
     (SELECT coursecode FROM has WHERE collegecode=?);
```

ADD COURSE PAGE



PAGE DESCRIPTION

This page gets the course details and add course to the courses table. This gets the name, tag, description and adds it.

QUERIES USED

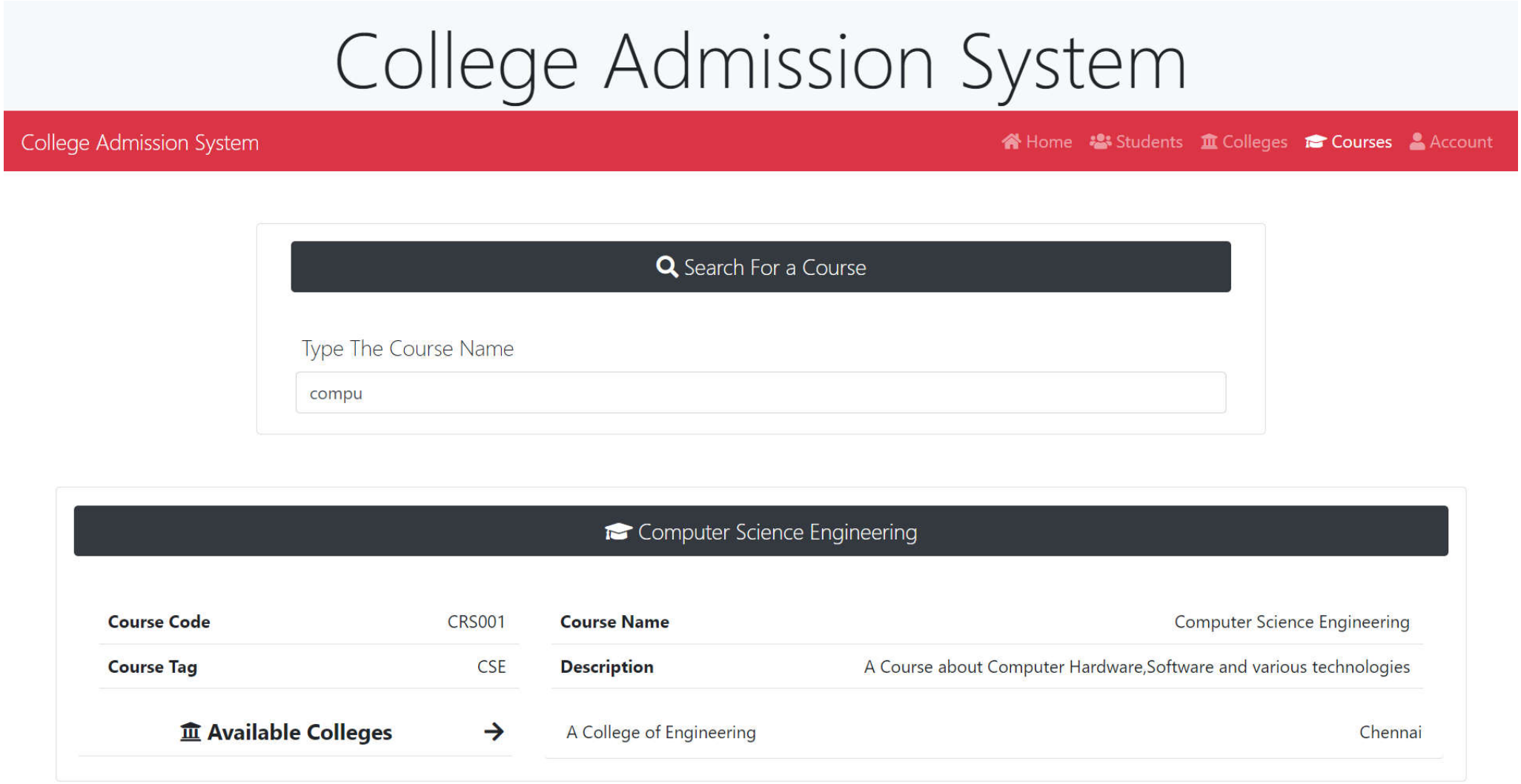
1.Check for already available course

```
SELECT COUNT(coursecode) AS count FROM course WHERE coursetag=?;
```

2.Insert course into the courses table

```
INSERT INTO course VALUES(?,?,?,?);
```

SEARCH COURSE PAGE



PAGE DESCRIPTION

This page gets course name as being typed, it gets the course with that name along with the colleges in which the course is available. If multiple courses are available, then all courses are shown.

QUERIES USED

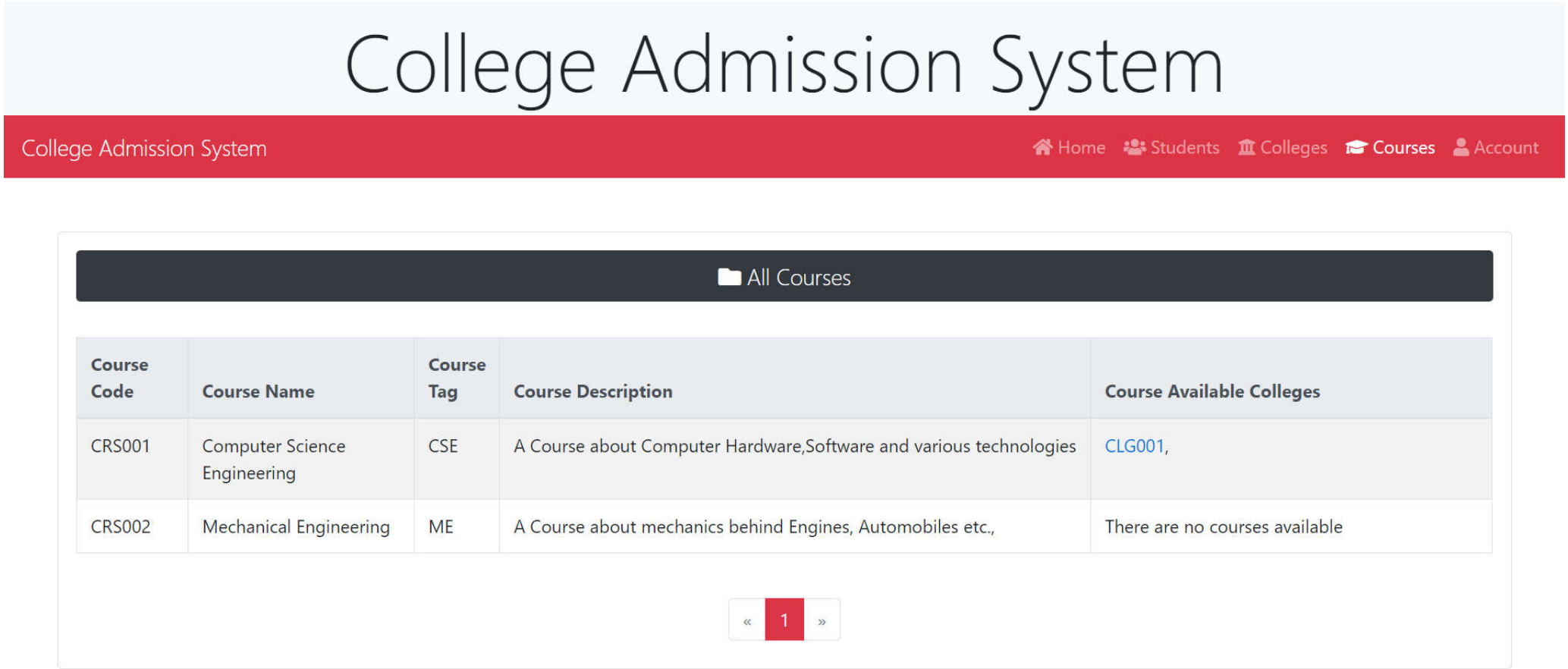
1.Get the courses with the part of the course name string

```
SELECT *,LOWER(coursename),INSTR(LOWER(coursename),?)
FROM course
WHERE INSTR(LOWER(coursename),?)>0
ORDER BY INSTR(LOWER(coursename),?);
```

2. Find the colleges with that course

```
SELECT collegename,dean,city
FROM college,has
WHERE has.coursecode=? AND has.collegecode=college.collegecode;
```

ALL COURSES PAGE



PAGE DESCRIPTION

This page shows all the courses along with the colleges that course is available. It shows 5 results per page and based on the page number the courses list is ordered by coursecode.

QUERIES USED

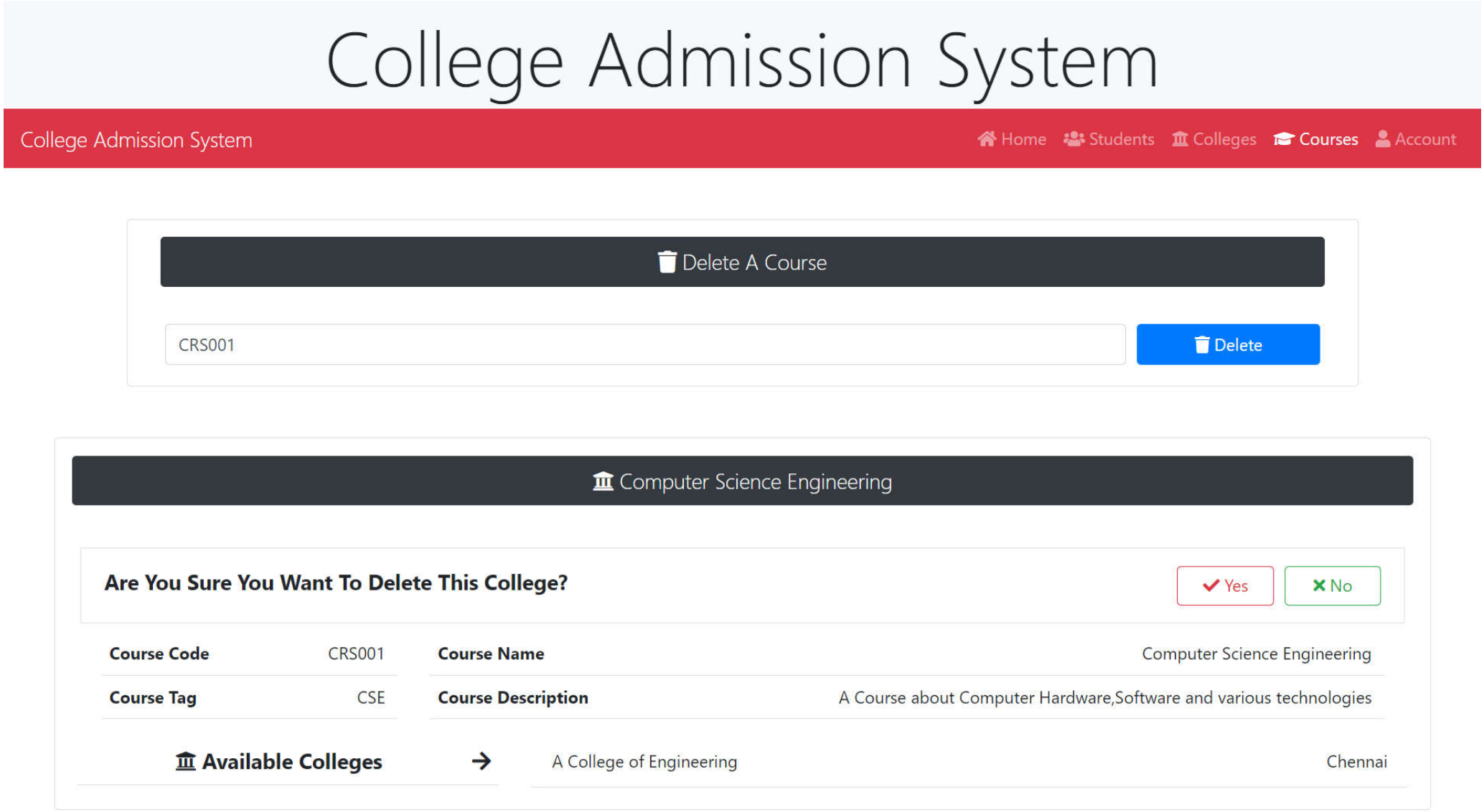
1.Get all the courses with limit 5 results

```
SELECT * FROM course ORDER BY coursecode LIMIT ?,5;
```

2.Get the available colleges for that course

```
SELECT course_details(?) AS course_string;
```

DELETE COURSE PAGE



PAGE DESCRIPTION

This page gets a coursecode as input and gets the course details along with the available colleges for that course with two options. If yes is clicked, then the course will be deleted and all the colleges with that course is also deleted. If no is clicked, the page is ready for new course deletion.

QUERIES USED

1.Get a course by its coursecode

```
SELECT * FROM course WHERE coursecode=?;
```

2.Get available colleges for that course

```
SELECT collegename,dean,city
FROM college,has
WHERE has.coursecode=? AND has.collegecode=college.collegecode;
```

3.Delete course from has table

```
DELETE FROM has WHERE coursecode=?;
```

4.Delete course from course table

```
DELETE FROM course WHERE coursecode=?;
```

STUDENTS HOME PAGE

College Admission System

College Admission System

Home Academics Rank Choices

What You Need To Do Now?

- Enter Your Marks in Academics tab
- Your Rank Will be determined. Please Wait
- Go to Rank Tab and Check Your Rank
- Enter Your Choices under Choices Tab
- **Get Your Allocation Details in Home Tab**

Hello Athiban T ,

Student ID	1	**Address**	611,Amirthammal Thangavel NagarOrikkai.
First Name	Athiban	**Gender**	Male
Last Name	T	**Date Of Birth**	2000-12-11
Email ID	as@gmail.com	**Phone Number**	6369155069
Age	18	**Nationality**	India

PAGE DESCRIPTION

This page gets a profile picture and uploads it to the server. It also shows all the student details. It shows the tasks to do now list and it also shows whether the application got accepted or rejected.

QUERIES USED

1.Find if image already inserted

```
SELECT imagename,status FROM image WHERE studentid=?
```

2.Insert student profile image

```
UPDATE image SET imagename=?,status=1 WHERE studentid=?
```

3.Get student profile

```
SELECT * FROM studdata,student
WHERE studdata.email=(select email FROM student WHERE studentid=?)
```

4.Get tasks list value from webdata table

```
SELECT value FROM webdata WHERE id=?
```

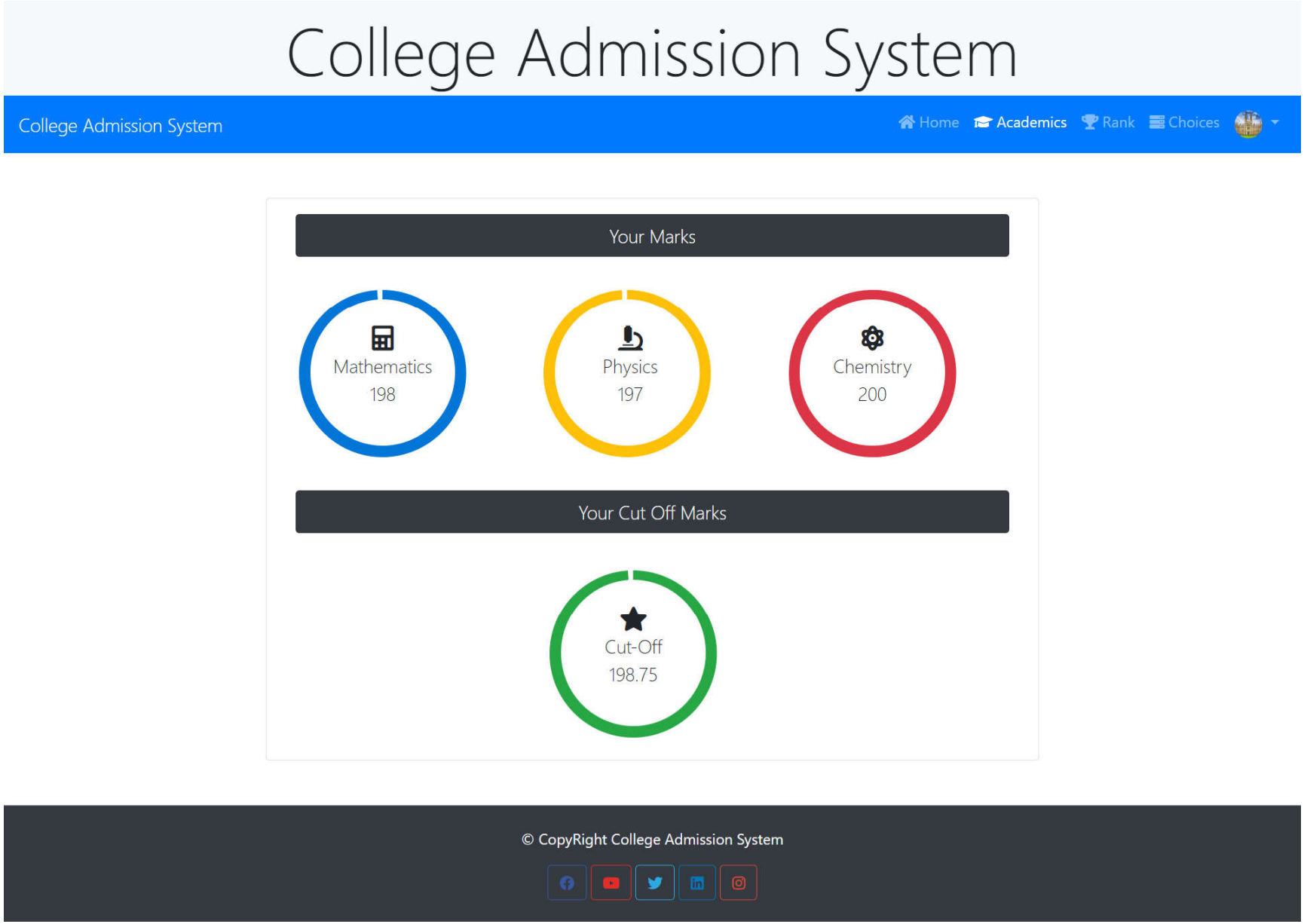
5.Check whether admitted or rejected

```
SELECT * FROM rejected WHERE studentid=?
```

6.If admitted, Get admitted Data

```
SELECT student.studentid,college.collegename,course.coursename
     FROM student,studdata,course,college,admitted
     WHERE student.studentid=admitted.studentid
     AND college.collegecode=admitted.collegecode
     AND course.coursecode=admitted.coursecode
     AND student.email=studdata.email
     AND student.studentid=?;
```

ACADEMICS PAGE



PAGE DESCRIPTION

This page gets the marks details and adds it into the database. It calculates the cutoff and shows it in a circular progress bar fashion. This marks helps in calculating the rank.

QUERIES USED

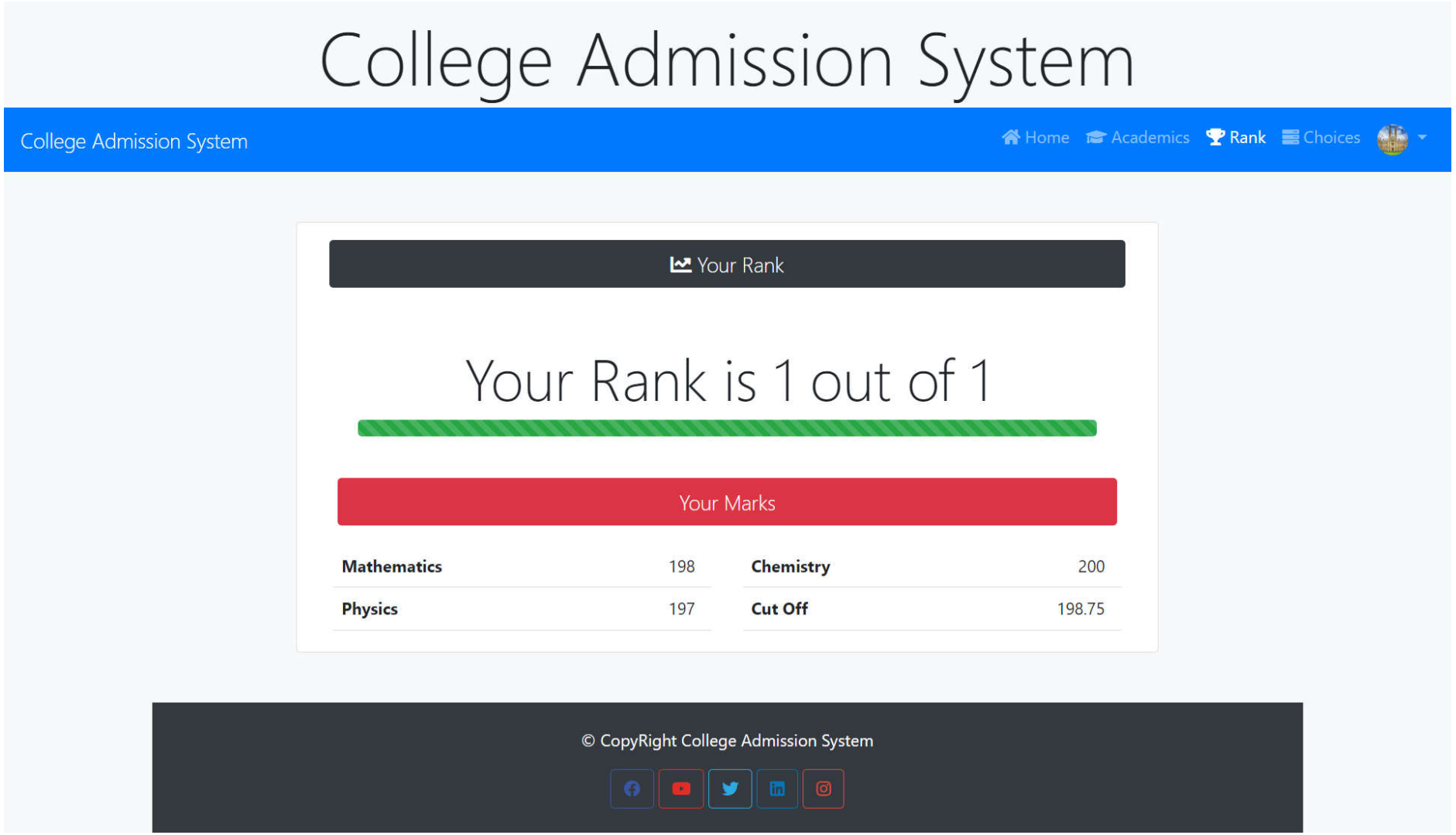
1.Check if marks already entered

```
SELECT * FROM academics WHERE studentid=?;
```

2.Insert marks to the table

```
INSERT INTO academics VALUE(?,?,?,?,?,?);
```

RANK PAGE


College Admission System
College Admission System
Home
Academics
Rank
Choices
Your Rank
Your Rank is 1 out of 1
Your Marks
Mathematics 198
Chemistry 200
Physics 197
Cut Off 198.75
© CopyRight College Admission System

PAGE DESCRIPTION

This page shows the rank of the student among all other students along with some academics information.

QUERIES USED

1.Check whether to show rank or not

```
SELECT value FROM webdata WHERE id='showRank';
```

2.Get count of all students in the table

```
SELECT COUNT(studentid) AS count FROM student;
```

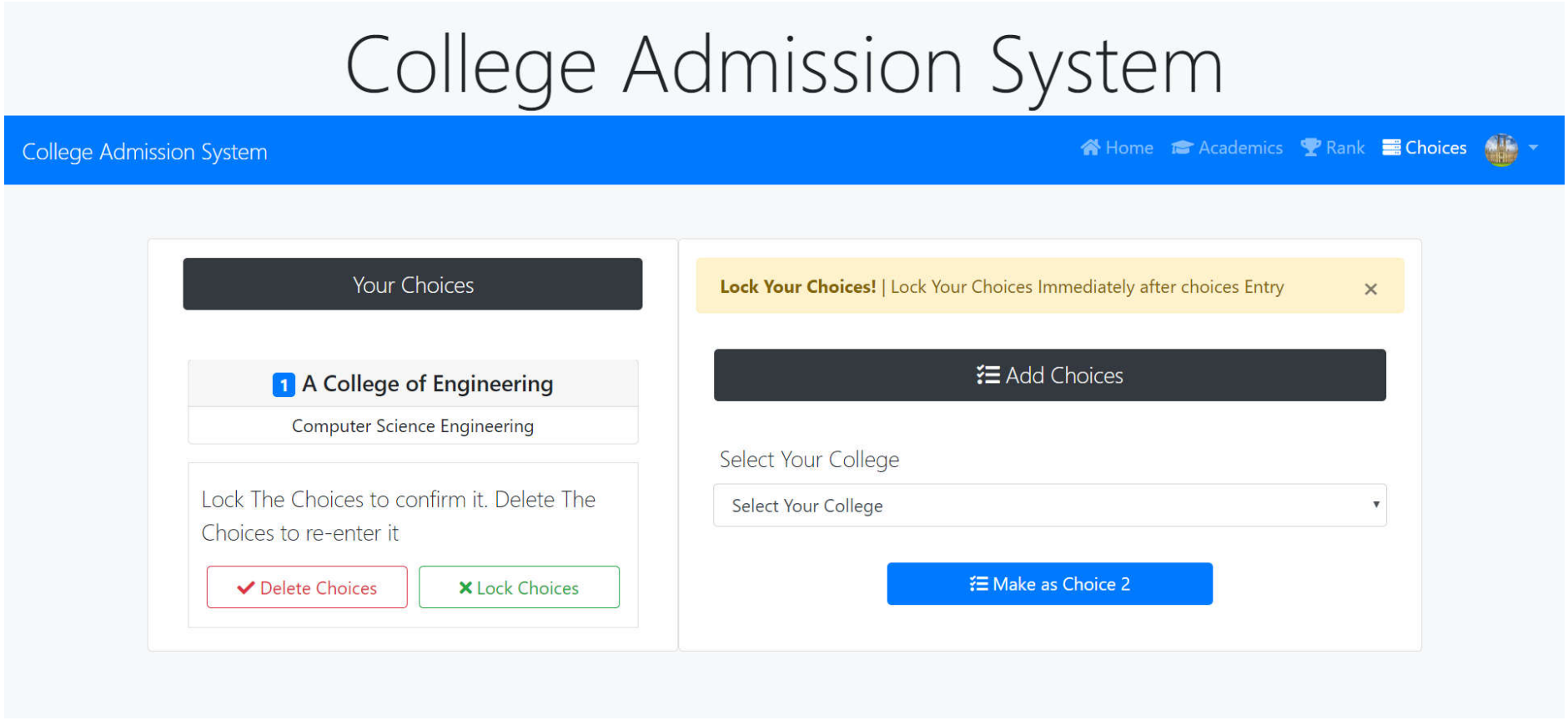
3.Get academics information

```
SELECT * FROM academics WHERE studentid=?;
```

4.Get rank information

```
SELECT * FROM grade WHERE studentid=?;
```

CHOICES PAGE



PAGE DESCRIPTION

This page lets you add choices according to a order. On the left card it shows all the current choices and options delete choices and lock them. On the right card, it asks you the college and course and adds it as the next choice.

QUERIES USED

1.Get Choices for the student

```
SELECT college.collegename,course.coursename,choice.choiceno
FROM college,course,choice
WHERE course.coursecode=choice.coursecode
AND college.collegecode=choice.collegecode
AND choice.studentid=?;
```

2.Get count of choices for the student

```
SELECT COUNT(*) AS count FROM choice WHERE studentid=?;
```

3.Get all colleges in the table

```
SELECT college.collegecode,college.collegename
FROM has,college
WHERE has.collegecode=college.collegecode;
```

4.Get all courses in a college that are not added as choices

```
SELECT course.coursecode,course.coursename
FROM has,course
WHERE has.collegecode=? AND has.coursecode=course.coursecode
AND has.coursecode NOT IN
(SELECT coursecode FROM choice WHERE collegecode=? AND studentid=?);
```

5.Insert new choice

```
INSERT INTO choice VALUES (?,?,?,?);
```

6.Delete Choices

```
DELETE FROM choice WHERE studentid=?
```

7.Lock Choices

```
UPDATE lockchoices SET lockchoice=1 WHERE studentid=?
```

8.Check whether to get choices or not

```
SELECT value FROM webdata WHERE id='getChoices';
```

9.Check whether the choice is locked or not

```
SELECT lockchoice FROM lockchoices WHERE studentid=?
```