Bilkent University

Department of Computer Engineering

GROUP

Course Enrollment System

Design Report

Abdullah Mahir Özer 21300616

Mert Sezer 21320246

Group 41

CS 353 / 1

Database System

Fall 2018

26 November 2018

1. PROJECT DESCRIPTION **7**

2. REVISED E/R MODEL **7**

2.1. Changes Made to The Model 7

2.2. Updated E/R Diagram 8

3. Relational Schemas **9**

3.1. STUDENT entity 9

Relational Model: 9

Functional Dependencies: 9

Candidate Keys: 9

Primary Key: 9

Normal Form: 9

Table Definition: 9

3.2 Course entity 10

Relational Model: 10

Functional Dependencies: 10

Candidate Keys: 10

Normal Form: 10

Table Definition: 10

3.3 Student_Course Relation 11

Relational Model: 11

Functional Dependencies: 11

Candidate Keys: 11

Normal Form: 11

Table Definition: 11

3.4. Course_Requirements Relation 11

Relational Model: 11

Functional Dependencies: 12

Candidate Keys: 12

Normal Form: 12

Table Definition: 12

3.5 check_TA Relation 12

Relational Model: 12

Functional Dependencies: 12

Candidate Keys: 12

Normal Form: 13

Table Definition: 13

3.6 check_inst Relation 13

Relational Model: 13

Functional Dependencies: 13

Candidate Keys: 13

Normal Form: 13

Table Definition: 13
3.7. Instructor_Assignment Relation 14
Relational Model: 14
Functional Dependencies: 14
Candidate Keys: 14
Normal Form: 14
Table Definition: 14
3.8. Classroom entity 15
Relational Model: 15
Functional Dependencies: 15
Candidate Keys: 15
Normal Form: 15
Table Definition: 15
3.9. Section entity 15
Relational Model: 15
Functional Dependencies: 15
Candidate Keys: 15
Normal Form: 16
Table Definition: 16
3.10. Enroll relation 16
Relational Model: 16
Functional Dependencies: 16
Candidate Keys: 16
Normal Form: 17
Table Definition: 17
3.11. sec_Class relation 17
Relational Model: 17
Functional Dependencies: 17
Candidate Keys: 17
Normal Form: 17
Table Definition: 17
3.12. Time_Slot entity 18
Relational Model: 18
Functional Dependencies: 18
Candidate Keys: 18
Normal Form: 18
Table Definition: 18
3.13. sec_time_slot relation 18
Relational Model: 18
Functional Dependencies: 18
Candidate Keys: 18
Normal Form: 19

Table Definition: 19
3.14. prereq relation 19
Relational Model: 19
Functional Dependencies: 19
Candidate Keys: 19
Normal Form: 19
Table Definition: 19
3.15. Instructor entity 20
Relational Model: 20
Functional Dependencies: 20
Candidate Keys: 20
Normal Form: 20
Table Definition: 20
3.16. TA entity 21
Relational Model: 21
Functional Dependencies: 21
Candidate Keys: 21
Normal Form: 21
Table Definition: 21
3.17. SMS entity 21
Relational Model: 21
Functional Dependencies: 21
Candidate Keys: 22
Normal Form: 22
Table Definition: 22
3.18. TA_Assignment entity 22
Relational Model: 22
Functional Dependencies: 22
Candidate Keys: 22
Normal Form: 22
Table Definition: 22
3.19. Instructor_Assignment entity 23
Relational Model: 23
Functional Dependencies: 23
Candidate Keys: 23
Normal Form: 23
Table Definition: 23
3.20. Lab entity 23
Relational Model: 23
Functional Dependencies: 23
Candidate Keys: 23
Normal Form: 24

Table Definition: 24
3.21. Project entity 24
Relational Model: 24
Functional Dependencies: 24
Candidate Keys: 24
Normal Form: 24
Table Definition: 24
3.22. Homework entity 25
Relational Model: 25
Functional Dependencies: 25
Candidate Keys: 25
Normal Form: 25
Table Definition: 25
3.23. advisor relation 25
Relational Model: 25
Functional Dependencies: 26
Candidate Keys: 26
Normal Form: 26
Table Definition: 26
3.24. send_SMS relation 26
Relational Model: 26
Functional Dependencies: 26
Candidate Keys: 26
Normal Form: 26
Table Definition: 27
3.25. Assignment entity 27
Relational Model: 27
Functional Dependencies: 27
Candidate Keys: 27
Normal Form: 27
Table Definition: 27
3.26. Withdraw relation 28
Relational Model: 28
Functional Dependencies: 28
Candidate Keys: 28
Normal Form: 28
Table Definition: 28
3.27. submit relation 29
Relational Model: 29
Functional Dependencies: 29
Candidate Keys: 29
Normal Form: 29

Table Definition: 29
3.28. Student_Section Relation 29
Relational Model: 29
Functional Dependencies: 29
Candidate Keys: 29
Normal Form: 30
Table Definition: 30
3.29. Assignment_of Relation 30
Relational Model: 30
Functional Dependencies: 30
Candidate Keys: 30
Normal Form: 30
Table Definition: 30
3.30. Exam entity 31
Relational Model: 31
Functional Dependencies: 31
Candidate Keys: 31
Normal Form: 31
Table Definition: 31
3.31. Course_Section relation 31
Relational Model: 31
Functional Dependencies: 32
Candidate Keys: 32
Normal Form: 32
Table Definition: 32
3.32 Person entity
3.33 Payment entity
3.34 Student_Payment relation
3.35 Instructor_Course relation

4. FUNCTIONAL COMPONENTS 32
4.1. USE CASES/SCENARIOS 32
Students 32
Academic Personnel 32
4.2. ALGORITHMS 33
Course Registration 33
4.3. DATA STRUCTURES 33
5.1 Login Page 33
5.2 Enroll in courses (Selecting sections) 35
5.3 Show Instructors 37
5.4 Course Search 39
5.4.1 Course Search by Department 39

5.4.2 Course Search by courseName 39

6. Advanced Database Components 39

6.1 Views 39

6.2 TRIGGERS 32

6.3 CONSTRAINTS 42

7. Implementation Plan 42

8. Website 42

1. PROJECT DESCRIPTION

We aim to create a course enrollment system similar to STARS system of Bilkent University. The system will let students enroll for courses, see available sections, courses and instructors. It will also let instructors offer courses. Students, instructors, courses, lecture hours will be stored in the database and the system will work by manipulating this database.

2. REVISED E/R MODEL

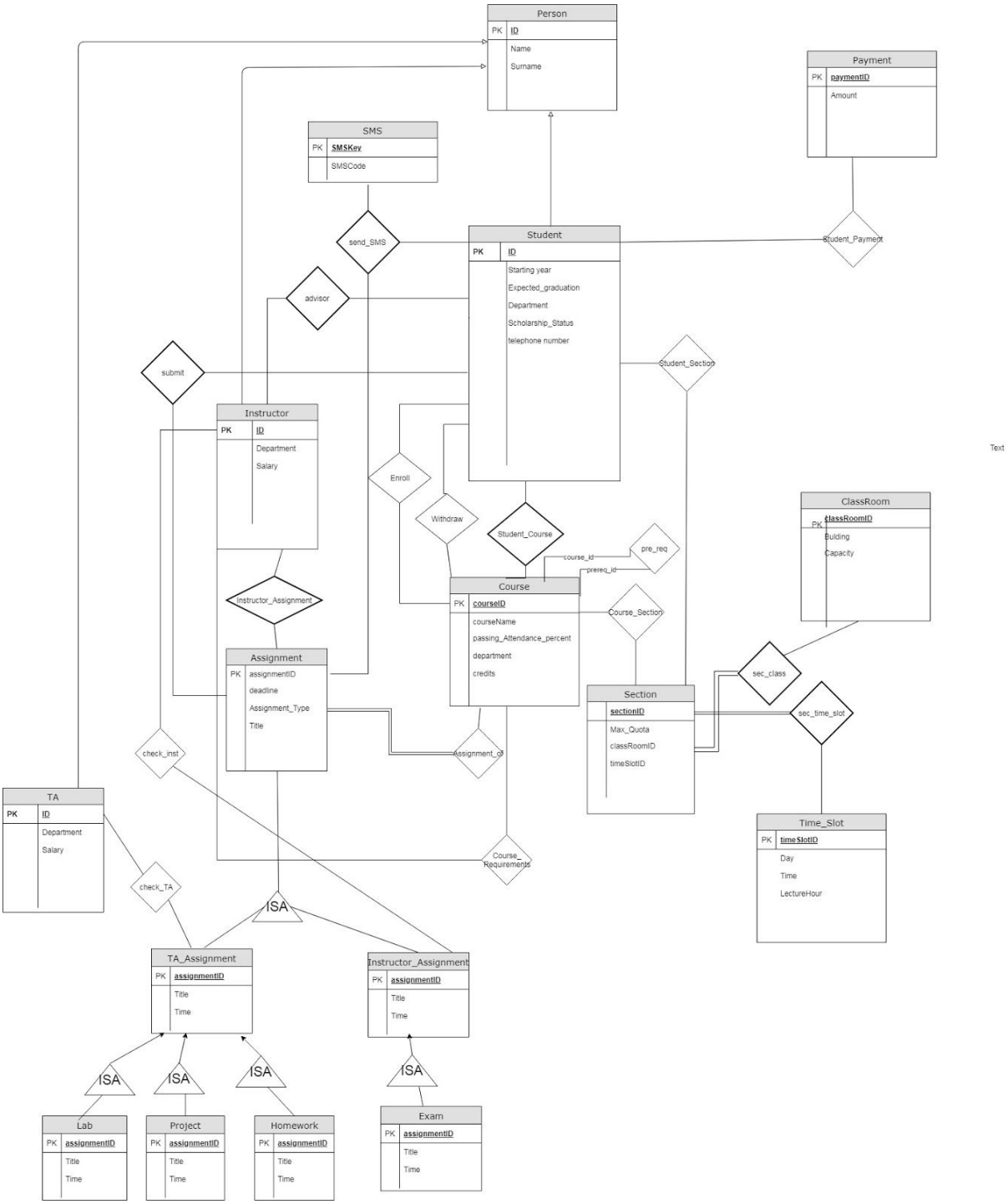
2.1. Changes Made to The Model

- Payments attribute was deleted from the student table. A new table called ‘Payment’ was made and a new relation Student_Payment was created. This relation holds every payment made by a particular student.
- ‘Teaches relation was renamed to Instructor_Course’.
- passingGrade, passingAttendancePercentage attributes were removed from the Course table. A new relation called ‘Course_Requirements’ was created. This relation holds the passing grade, min FZ grades, letter grades and minimum passing attendance for each course. This relation is also connected to the instructor since the instructor defines the passing and FZ grades.
- Classroom table had 2 primary keys, that was fixed.
- Schedule of the student is held in Student_Section relation. An update to the schedule also updates the Student_Course relation where all the curriculum of the student is held. It updates the currentlyTaking boolean to true when a course is added to the schedule, and to false when a course is removed.
- TA_Assignment entities and Instructor_Assignment entities are added; because instructors and TA(also added in the revised diagram)’s have to check different

types of assignments. Also check_inst and check_TA relationships are added for that purpose.

- Person entity is added and note that TA's, Instructor's and Student's are Person and therefore we deleted the attributes name, surname from all 3 subentities of Person. These 3 subentities are taking name and surname as a foreign key from Person.

2.2. Updated E/R Diagram



3. Relational Schemas

3.1. STUDENT entity

Relational Model:

STUDENT(ID, starting_year, expected_graduation, department, scholarship_status, payments, curriculum, GPA, telephone_number, password, current_login_at, last_login_at, gender, birth_date)

Functional Dependencies:

ID -> password

Candidate Keys:

{ (ID, telephone_number) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE student
(
        ID                  UNIQUE numeric(8,0) AUTO_INCREMENT,
        name varchar(20),
        surname varchar(20),
        starting_year       date,
        expected_graduationdate,
        department          varchar(32) NOT NULL,
        scholarship_status  varchar(32),
        GPA                 numeric(1,2),
        telephone_number    numeric(11,0),
        password            varchar(32),
```

```
    current_login_at    datetime,
    last_login_at       datetime,
    gender              boolean,
    birth_date          date,
    PRIMARY KEY(ID),
    FOREIGN KEY (name) REFERENCES Person(name),
    FOREIGN KEY (surname) REFERENCES Person(surname)
);
```

3.2 Course entity

Relational Model:

Course(courseName, passingAttendance_percent, credits)

Functional Dependencies:

courseID → courseName

Candidate Keys:

{courseID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Course(
    courseID                numeric(4,0) AUTO_INCREMENT,
    passing_Attendance_percent  numeric varchar(2),
    department varchar(4),
    courseName              varchar(32),
    PRIMARY KEY(courseID));
```

3.3 Student_Course Relation

Relational Model:

Student_Course(ID, courseID)

FOREIGN KEY (ID) references Student(ID)

FOREIGN KEY (courseID) references Course(courseID)

Functional Dependencies:

None

Candidate Keys:

{ID, courseID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Student_Course(
        ID numeric(8,0),
        courseID numeric(4,0),
        FOREIGN KEY (ID) references Student(ID),
        FOREIGN KEY (courseID) references Course(courseID)  );
```

3.4. Course_Requirements Relation

Relational Model:

Course_Requirements(courseID, ID, passingGrade, minAttendance)

FOREIGN KEY courseID references Course(courseID)

FOREIGN KEY ID references Student(ID)

Functional Dependencies:

courseID → passingGrade

courseID → minAttendance

Candidate Keys:

{courseID, ID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Course_Requirements(
        courseID            numeric(4,0) AUTO_INCREMENT,
        ID          numeric(8,0),
        passingGrade        numeric(1,2),
        minAttendance       numeric(1,2),
        FOREIGN KEY(courseID) REFERENCES Course(courseID),
        FOREIGN KEY(ID) REFERENCES Student(ID)) ;
```

3.5 check_TA Relation

Designer Comment: A TA can check special types of assignments that are labs, projects and homeworks that are TA_assignments.

Relational Model:

check_TA (ID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{ID, assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE check_TA
(
        ID              numeric (8,0),
        assignmentID        numeric (6,0)),
        PRIMARY KEY(ID, assignmentID),
        FOREIGN KEY (ID) REFERENCES Instructor(ID),
        FOREIGN KEY (assignmentID) REFERENCES Assignment(assignmentIID) );
```

3.6 check_inst Relation

Designer Comment: An instruct
or can check a special type of assignment that is exam that is a instructor_assignment.

Relational Model:

check_inst (ID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{ID, assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE check_inst
(
        ID              numeric (8,0),
        assignmentID        numeric (6,0))
```

PRIMARY KEY(ID, assignmentID),

FOREIGN KEY (ID) REFERENCES Instructor(ID),

FOREIGN KEY (assignmentID) REFERENCES Assignment(assignmentID));

3.7. Instructor_Assignment Relation

Designer comment:
This relation only exists do indicate which instructor has read this assignment. This information is useful when there are objections to a grade.

Relational Model:

Instructor_Assignment(ID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{ID, assignmentID}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Instructor_Assignment

(

ID numeric (8,0),

assignmentID numeric (6,0)),

PRIMARY KEY(ID, assignmentID),

FOREIGN KEY (ID) REFERENCES Instructor(ID),

FOREIGN KEY (assignmentID) REFERENCES Assignment(assignmentID));

3.8. Classroom entity

Relational Model:

Classroom(classRoomID, Building, Capacity)

Functional Dependencies:

classRoomID → capacity

Candidate Keys:

{classRoomID, building}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Classroom(

classRoomID numeric(3,0),

Building varchar(20),

Capacity numeric(3,0),

PRIMARY KEY(classRoomID));

3.9. Section entity

Relational Model:

Section(courseID, sectionID, Max_Quota, ClassRoomID, Time_SlotID)

foreign key courseID references Course(courseID)

Functional Dependencies:

courseID → Max_Quota

Candidate Keys:

{courseID, sectionID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Section(
        courseID  varchar(3),
        sectionID varchar(2),
        classroomID varchar(6),
        Time_SlotID  DATE,
        Max_Quota          numeric(6,0),
        PRIMARY KEY(courseID, sectionID),
        FOREIGN KEY(classroomID) REFERENCES ClassRoom,
        FOREIGN KEY(Time_SlotID) REFERENCES TimeSlot,
        FOREIGN KEY(courseID) REFERENCES Course;
```

3.10. Enroll relation

Relational Model:

enroll(ID, courseID)

foreign key ID references Student(ID)

foreign key courseID references Course(courseID)

Functional Dependencies:

None

Candidate Keys:

{ID, courseID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE enroll(
        ID numeric(8,0),
        courseID numeric(4,0),
```

FOREIGN KEY ID references Student,

FOREIGN KEY courseID references Course);

3.11. sec_Class relation

Relational Model:

sec_class(sectionID, classRoomID)

Foreign Key classRoomID references ClassRoom(classRoomID)

Foreign Key sectionID references Section(sectionID)

Functional Dependencies:

None

Candidate Keys:

{sectionID, classRoomID}

Normal Form:

BCNF

Table Definition:

CREATE TABLE sec_class(

timeSlotID DATETIME,

classRoomID varchar(10),

FOREIGN KEY timeSlotID references Time_Slot,

FOREIGN KEY classRoomID references ClassRoom);

3.12. Time_Slot entity

Relational Model:

Time_Slot(timeSlotID, Day, Time, LectureHour)

Functional Dependencies:

None

Candidate Keys:

{timeSlotID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Time_Slot(
        timeslotID              numeric(6,0) AUTO_INCREMENT,
        Day             DATETIME,
        Time            DATETIME,
        LectureHour   DATETIME,
        PRIMARY KEY(timeSlotID));
```

3.13. sec_time_slot relation

Relational Model:

sec_time_slot(sectionID, timeSlotID)

Functional Dependencies:

None

Candidate Keys:

{sectionID, timeSlotID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE sec_time_slot(
```

sectionID varchar(2),

timeSlotID DATE,

FOREIGN KEY(section_ID) REFERENCES Section,

FOREIGN KEY(timeSlotID) REFERENCES Time_Slot);

3.14. prereq relation

Relational Model:

prereq(courseID,prereqID)

Functional Dependencies:

None

Candidate Keys:

{courseID, prereqID}

Normal Form:

BCNF

Table Definition:

CREATE TABLE prereq(

courseID numeric(4,0),

prereqID numeric(4,0),

FOREIGN KEY courseID references Course(courseID),

FOREIGN KEY prereqID references Course(courseID) ;

3.15. Instructor entity

Relational Model:

Instructor(ID, department, salary)

Functional Dependencies:

ID -> name, surname, department, salary

Candidate Keys:

{ID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Instructor(
        ID        numeric(8,0),
        name              varchar(20),
        surname         varchar(20),
        department    varchar(100),
        salary  INT,
        PRIMARY KEY (ID),
        FOREIGN KEY name REFERENCES Person(name),
        FOREIGN KEY surname REFERENCES Person(surname) );
```

3.16. TA entity

Relational Model:

TA(ID, department, salary)

Functional Dependencies:

ID → name, surname, department, salary

Candidate Keys:

{ID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE TA(
        ID              numeric(8,0),
        name            varchar(30),
        surname         varchar(30),
        department      varchar(100),
        salary  INT,
        PRIMARY KEY (ID),
        FOREIGN KEY name REFERENCES Person(name),
        FOREIGN KEY  surname REFERENCES Person(surname));
```

3.17. SMS entity

Relational Model:

SMS(SmsKey, SMS_Code)

Functional Dependencies:

None

Candidate Keys:

{SmsKey}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE SmsKey(
        SMSKey              numeric(6,0) AUTO_INCREMENT,
        SMS_code            INT,
        PRIMARY KEY(SMSKey)) ;
```

3.18. TA_Assignment entity

Relational Model:

TA_Assignment(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{ (assignmentID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE TA_Assignment(
        assignmentID        numeric(6,0) AUTO_INCREMENT,
        Title           varchar(32),
        Time            DATE,
        PRIMARY KEY(assignmentID));
```

3.19. Instructor_Assignment entity

Relational Model:

Instructor_Assignment(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Instructor_Assignment(
        assignmentID        numeric(6,0) AUTO_INCREMENT,
        Title               varchar(32),
        Time                DATE,
        PRIMARY KEY(assignmentID));
```

3.20. Lab entity

Relational Model:

Lab(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Lab(
        assignmentID        numeric(6,0) AUTO_INCREMENT,
        Title               varchar(32),
        Time                DATE,
        PRIMARY KEY(assignmentID));
```

3.21. Project entity

Relational Model:

Project(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Project(
        assignmentID        numeric(6,0) AUTO_INCREMENT,
```

```
    Title           varchar(32),
    Time            DATE,
    PRIMARY KEY(assignmentID));
```

3.22. Homework entity

Relational Model:

Homework(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Homework(
    assignmentID        numeric(6,0) AUTO_INCREMENT,
    Title           varchar(32),
    Time            DATE,
    PRIMARY KEY(assignmentID));
```

3.23. advisor relation

Relational Model:

advisor(ID, StudentID,)

Functional Dependencies:

None

Candidate Keys:

{ID, StudentID}

Normal Form:

BCNF

Table Definition:

CREATE TABLE advisor(

ID numeric(8,0),

StudentID numeric(8,0),

FOREIGN KEY StudentID references Student(ID),

FOREIGN KEY ID references Instructor(ID));

3.24. send_SMS relation

Relational Model:

send_SMS(SMSKey, ID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{SMSKey, ID, assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE send_SMS(
        SMSKey              numeric(6,0),
        ID          numeric(8,0),
        assigmentID         numeric(6,0),
        FOREIGN KEY SMSKey references SMS,
        FOREIGN KEY ID references Student,
        FOREIGN KEY assignmentID references Assignment) ;
```

3.25. Assignment entity

Designer Comment: AssignmentID won't consist of only 1 numeric character so that it can be a primary key. Also Assignment_Type can be one of TA_Assignment like lab, project, homework or one of Instructor_Assignment like exam.

Relational Model:

Assignment(<u>assignmentID</u>, deadline, Assignment_Type, Title)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Assignment(
        assignmentID        numeric(6,0),
        courseID            numeric(4,0),
        deadline            DATETIME,
```

```
    Assignment_Type varchar(25),
    PRIMARY KEY(assignmentID),
    FOREIGN KEY courseID references Course);
```

3.26. Withdraw relation

Relational Model:

Withdraw(ID, courseID)

foreign key ID references Student(ID)

foreign key courseID references Course(courseID)

Functional Dependencies:

None

Candidate Keys:

{ID, courseID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Withdraw(
    ID numeric(8,0),
    courseID numeric(4,0),
    FOREIGN KEY ID references Student,
    FOREIGN KEY courseID references Course );
```

3.27. submit relation

Relational Model:

submit(ID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{ID, assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE submit(
        ID              numeric(8,0),
        assignmentID    numeric(6,0),
        FOREIGN KEY ID references Student
        FOREIGN KEY assignmentID references Assignment);
```

3.28. Student_Section Relation

Relational Model:

Student_Section (courseID, sectionID)

foreign key ID references Student(ID)

foreign key sectionID references Section(sectionID)

Functional Dependencies:

None

Candidate Keys:

{ID, sectionID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Student_Section(
        ID numeric(8,0),
        sectionID numeric(2,0),
        FOREIGN KEY ID references Student,
        FOREIGN KEY sectionID references Section );
```

3.29. Assignment_of Relation

Relational Model:

Assignment_of(courseID, assignmentID)

Functional Dependencies:

None

Candidate Keys:

{courseID, assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Assignment_of(
        courseID                numeric(4,0),
```

```
    assignmentID        numeric(6,0),
    FOREIGN KEY courseID references Course,
    FOREIGN KEY assignmentID references Assignment );
```

3.30. Exam entity

Relational Model:

Exam(assignmentID, Title, Time)

Functional Dependencies:

None

Candidate Keys:

{assignmentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Exam(
    assignmentID        numeric(6,0) AUTO_INCREMENT,
    Title           varchar(32),
    Time            DATE,
    PRIMARY KEY(assignmentID));
```

3.31. Course_Section relation

Relational Model:

Course_Section(courseID, sectionID)

Foreign Key courseID references Course(courseID)

Foreign Key sectionID references Section(sectionID)

Functional Dependencies:

None

Candidate Keys:

{sectionID, courseID}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Course_Section(

courseID varchar(3),

sectionID varchar(2),

FOREIGN KEY courseID references Course,

FOREIGN KEY sectionID references Section);

3.32. Person

Relational Model:

Person(ID, name, surname)

Functional Dependencies:

None

Candidate Keys:

{ID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Person(
        ID numeric(8,0),
        name varchar(20),
        surname varchar(20) );
```

3.33. Payment entity

Relational Model:

Payment(paymentID, amount)

Functional Dependencies:

None

Candidate Keys:

{paymentID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Payment(
        paymentID numeric(10,0),
        amount  INT);
```

3.34. Student_Payment relation

Relational Model:

Student_Payment(paymentID, ID)

Functional Dependencies:

None

Candidate Keys:

{paymentID, ID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Student_Payment(
        ID numeric(8,0),
        paymentID numeric(10, 0),
        FOREIGN KEY (ID) REFERENCES Student,
        FOREIGN KEY (paymentID) REFERENCES Payment
        );
```

3.35. Instructor_Course Relation

Relational Model:

Instructor_Course(ID, courseID)

Functional Dependencies:

None

Candidate Keys:

{ID, courseID}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Instructor_Course
(
        ID                  numeric (8,0),
        courseID                    numeric (4,0)),
        PRIMARY KEY(ID, courseID),
        FOREIGN KEY (ID) REFERENCES Instructor(ID),
        FOREIGN KEY (courseID) REFERENCES Assignment(courseID)       );
```

4. FUNCTIONAL COMPONENTS

4.1. USE CASES/SCENARIOS

Students

Students are able to :

- Log in to the accounts created for them.
- Add, change or drop courses.
- Make payments for certain services.
- View their grades and attendances for the courses they are taking.
- See the upcoming exams and the time left.
- See and compare offerings which are sent from relevant professionals.

Instructor

Instructors are able to:

- Log in to the accounts created for them.
- Grade exams and assignments of students and courses they are responsible of.

4.2. ALGORITHMS

Course Registration

Students cannot register to the courses if they have not passed the prerequisite courses. The system will only offer the option to add a course if the student is eligible to it. In order to add a course, the student needs to choose a section. A section can only be taken by as many students as the capacity of that section's classroom. Even if there are free seats available, students can still not register to the course if the section's timeslot conflicts with the student's schedule, unless the student is trying to change sections and the conflict is with the section that is to be dropped at the end of the change. A student can have no more that one section of the same course concurrently. Students can drop all courses they have added.

4.3. DATA STRUCTURES

In the database, relational schemas are created using three types of variables.

These are numeric types, text types and date.

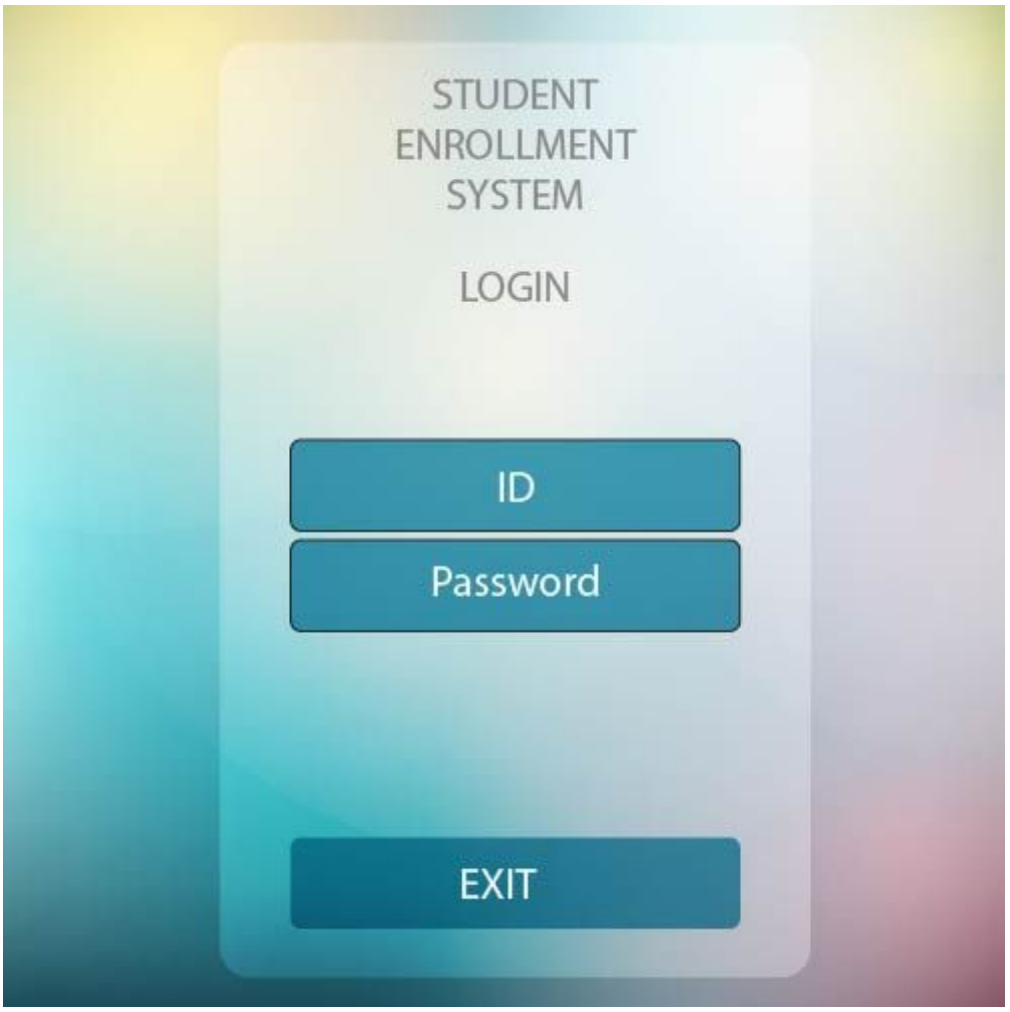
Numeric types are used for keeping numeric data such as IDs, cost etc.. Numeric types are created as INT as standard numeric type.

Information that require text types such as name, city names, passwords etc., are created as VARCHAR as standard.

Information that require date types such as login date, start date etc., are created as DATE and DATETIME as standard.

5. User Interface Design and Corresponding SQL Statements

5.1 Login Page



Inputs: @ID, @password

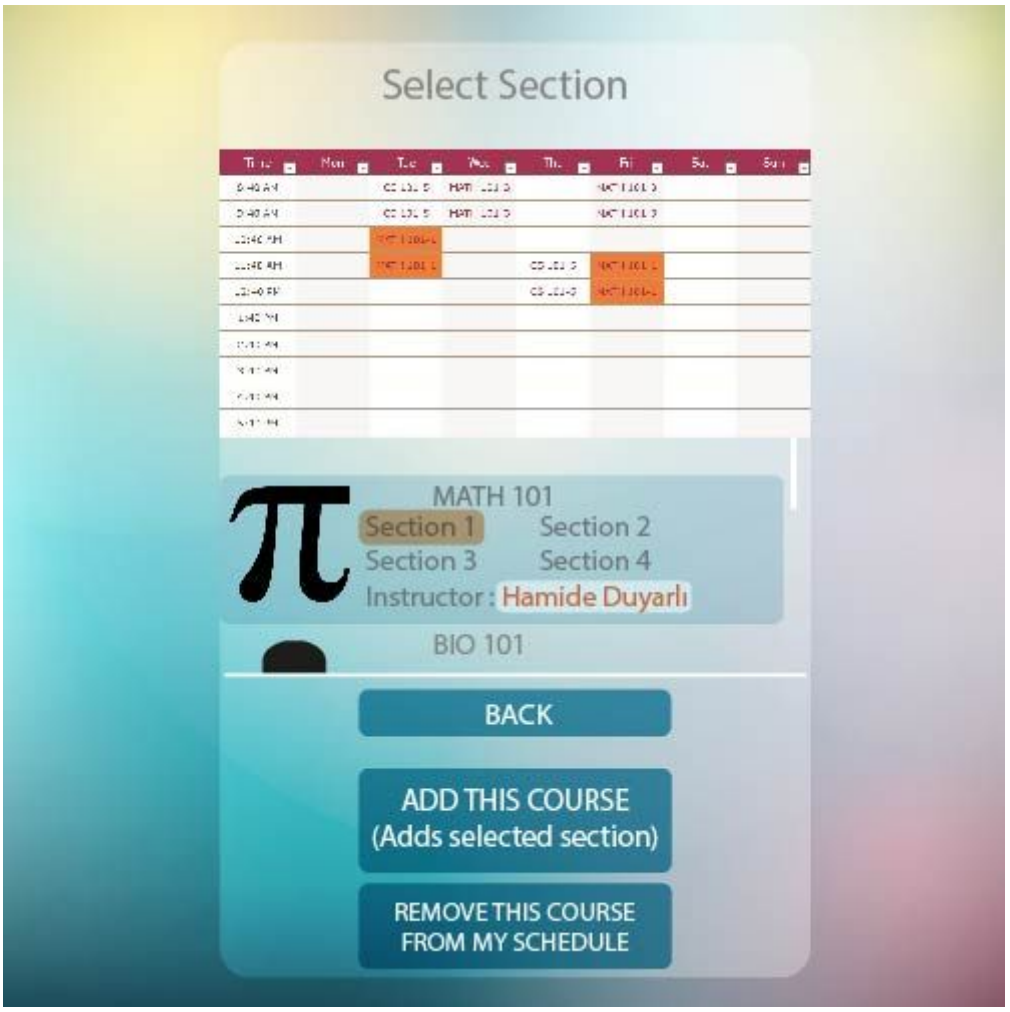
Process:

Users trying to access the program are greeted with the log-in page. Entering a valid ID and a matching password logs the person in to their account.

SQL Statement :

```
SELECT *
FROM Person
WHERE ID = @ID AND password = @password;
```

5.2 Enroll in courses (Selecting sections)



Inputs: @SectionSelection, @IDofThisStudent(This input is set when logging in.)

Process: The courses that are required in the student's department are listed. The student can select a course. When a course is selected, the sections that are corresponding to a course are listed near(to the right) the name of the course. The student can select a section and see the changes in their weekly schedule. The student also sees the instructor name change when he/she changes the section selection.

SQL Statements:

Schedule By Using Student_Section relation:

```
SELECT Section.sectionID, Course.courseID, Course.courseName
FROM Student_Section
INNER JOIN Section
ON Section.sectionID = Student_Section.sectionID
```

```
INNER JOIN Sec_Time_Slot
ON Sec_Time_Slot.sectionID = Section.sectionID
INNER JOIN Time_Slot
ON Time_Slot.timeSlotID = Sec_Time_Slot.timeSlotID
INNER JOIN Course_Section
ON Section.sectionID = Course_Section.sectionID
INNER JOIN Course
ON Course.courseID = Course_Section.courseID
WHERE Student.ID = @IDofThisStudent
GROUP BY timeSlotID
/*Another query for viewing schedule using DISTINCT instead of GROUP BY*/
CREATE VIEW schedule(sectionOnDay, courseIDOnDay, courseNameOnDay)
AS SELECT DISTINCT TimeSlot.timeSlotID, Section.sectionID, Course.courseID,
Course.courseName
FROM Student_Section
INNER JOIN Section
ON Section.sectionID = Student_Section.sectionID
INNER JOIN Sec_Time_Slot
ON Sec_Time_Slot.sectionID = Section.sectionID
INNER JOIN Time_Slot
ON Time_Slot.timeSlotID = Sec_Time_Slot.timeSlotID
INNER JOIN Course_Section
ON Section.sectionID = Course_Section.sectionID
INNER JOIN Course
ON Course.courseID = Course_Section.courseID
WHERE Student.ID = @IDofThisStudent
```

Viewing schedule By Using Student_Course relation:

```
CREATE VIEW schedule(sectionOnDay, courseIDOnDay, courseNameOnDay)
AS SELECT sectionID, courseID, courseName
FROM Student
INNER JOIN Student_Course
```

```
ON (Student.ID = Student_Course.ID AND
INNER JOIN Course
ON Course.courseID = Student_Course.courseID
INNER JOIN
Course_Section
ON Course_Section.courseID = Student_Course.courseID
INNER JOIN Section
ON Course_Section.sectionID = Section.sectionID
INNER JOIN sec_time_slot
ON Section.sectionID = sec_time_slot.sectionID
INNER JOIN time_slot
ON sec_time_slot.time_slot_id = time_slot.time_slot_id
WHERE
Course.currentlyTaking = 1
INSERT INTO
```

Adding course to schedule:

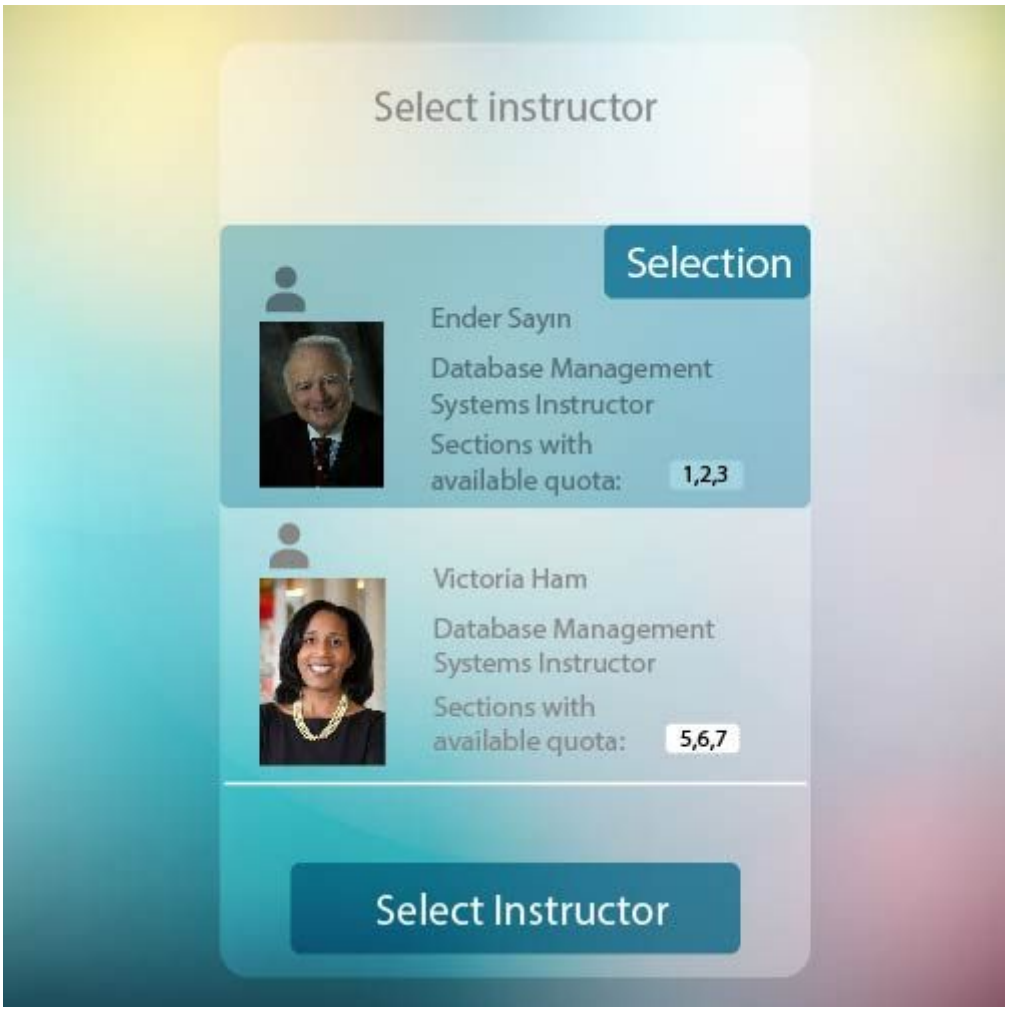
```
SELECT *
FROM Section
Where Section.sectionID = @SectionSelection
INSERT INTO Student_Section
VALUES(@ID, @sectionID);
```

Updating curriculum after a course is added to schedule(Student_Course):

```
SELECT*
FROM Course_Section

UPDATE Student_Course
SET Course.currentlyTaking = 1
WHERE @SelectedSection.sectionID = Course_Section.sectionID
AND  Course_Section.courseID = Student_Course.courseID
```

5.3 Show Instructors



Inputs: @department, @selectedInstructor

Process:This interface is separate from the add section interface. Here instructors are given more importance, a student can see the instructors with their profile pictures for better recognition and hit select to see the courses that these instructors teach.

Following SQL Statement to show all the instructors on the screen.

```
CREATE VIEW all_instructors AS
SELECT *
FROM INSTRUCTOR
WHERE INSTRUCTOR.department = @department;
```

Selecting instructor:

```
SELECT *
FROM INSTRUCTOR
WHERE INSTRUCTOR.ID = @selectedInstructor.ID
```

5.4 Course Search

5.4.1 Course Search by Department

Inputs: @SearchQuery, @selectedDepartment, @courseName

```
SELECT *
CASE WHEN @SearchQuery = department
FROM Course
WHERE Course.department = @selectedDepartment
```

5.4.2 Course Search by courseName

```
SELECT *
CASE WHEN @SearchQuery = courseName
FROM Course
WHERE Course.courseName = @courseName
```

6. Advanced Database Components

6.1 Views

6.1.1 Student GPA View

```
CREATE VIEW student_info(ID,name, gpa)
AS SELECT ID,name, avgGrade
FROM
Student NATURAL JOIN Student_Course
NATURAL JOIN (SELECT SUM(credits) as cSum, AVG(Grade*credits)/cSum AS avgGrade
FROM Course NATURAL JOIN Student_Course
GROUP BY Student_Course.ID)
```

6.1.2 View for the Schedule View using Student_Section relation

```
CREATE VIEW schedule(sectionOnDay, courseIDOnDay, courseNameOnDay)
```

```
AS SELECT Section.sectionID, Course.courseID, Course.courseName
FROM Student_Section
INNER JOIN Section
ON Section.sectionID = Student_Section.sectionID
INNER JOIN Sec_Time_Slot
ON Sec_Time_Slot.sectionID = Section.sectionID
INNER JOIN Time_Slot
ON Time_Slot.timeSlotID = Sec_Time_Slot.timeSlotID
INNER JOIN Course_Section
ON Section.sectionID = Course_Section.sectionID
INNER JOIN Course
ON Course.courseID = Course_Section.courseID
WHERE Student.ID = @IDofThisStudent
GROUP BY timeSlotID
/*Another query for viewing schedule using DISTINCT instead of GROUP BY*/
CREATE VIEW schedule(sectionOnDay, courseIDOnDay, courseNameOnDay)
AS SELECT DISTINCT TimeSlot.timeSlotID, Section.sectionID, Course.courseID,
Course.courseName
FROM Student_Section
INNER JOIN Section
ON Section.sectionID = Student_Section.sectionID
INNER JOIN Sec_Time_Slot
ON Sec_Time_Slot.sectionID = Section.sectionID
INNER JOIN Time_Slot
ON Time_Slot.timeSlotID = Sec_Time_Slot.timeSlotID
INNER JOIN Course_Section
ON Section.sectionID = Course_Section.sectionID
INNER JOIN Course
ON Course.courseID = Course_Section.courseID
WHERE Student.ID = @IDofThisStudent
```

6.2 TRIGGERS

- When a course is added to the schedule, currentlyTaking value of that course is set to true on the curriculum which is all the courses stored in the Student_Course relation for each distinct student ID(the ID derived from Person table).

```
CREATE TRIGGER updateCurriculumInsert
ON Student_Section
AFTER INSERT
AS
        DECLARE @added_section_id INT
        SELECT @added_section_id = section_id FROM inserted

        UPDATE Student_Course
        SET currentlyTaking = 1
        WHERE Student_Course.sectionID = @added_section_id
```

- Similarly when a course is removed from the schedule, currentlyTaking value of that course is set to false on the curriculum which is all the courses stored in the Student_Course relation for each distinct student ID(the ID derived from Person table).

```
CREATE TRIGGER updateCurriculumDelete
ON Student_Section
AFTER DELETE
AS
        DECLARE @added_section_id INT
        SELECT @added_section_id = section_id FROM deleted

        UPDATE Student_Course
        SET currentlyTaking = 0
        WHERE Student_Course.sectionID = @added_section_id
```

6.3 CONSTRAINTS

- The system can not be completely interacted without log-in.
- A student cannot register to courses with prerequisite courses unless they passed them before.
- A student cannot add a course with a full quota.

7. Implementation Plan

The Core of our system will be handled by the use of MySQL database management system and PHP as a communication language. Php, javascript, HTML and CSS can be used when needed in web design.

8. Website

Project website:

.