College Data Management System

Database Management Systems

MSCS_542L_256_23S

Super Six

Marist College

School of Computer Science and Mathematics

Submitted To: Dr. Reza Sadeghi

Spring 2023

Project Report of College data management system

Team Name

Super Six

Team members

Pradeep Reddy Baireddy.	pradeepreddy.baireddy1@marist.edu (Team Head)
Prajakta kshirsagar	prajaktakshirsagar2022@gmail.com (Team member)
Deepthi Niharika Gadepalli	DeepthiNiharika.Gadepalli1@marist.edu (Team member)
Bharadwaja Thota	bharadwaja.thota1@marist.edu (Team member)
Pradeep Reddy Macha	pradeepreddy.macha1@marist.edu (Team member)
Chakradhar Reddy Marepally	ChakradharReddy.Marepally1@marist.edu (Team member)

Table of Contents

Table of Figures

Project Objective:

Our project objective is to create a highly functional student information system that can efficiently manage and provide access to a wide range of student-related data. To achieve this, we have accomplished the following:

- Designed an Entity Relationship (ER) diagram and an Enhanced Entity Relationship (EER) diagram that clearly define the relationships and attributes of the ten tables in our database
- Created the ten tables in SQL Workbench and established relationships between them using foreign keys and other constraints
- Populated the tables with sample data to ensure their functionality and integrity
- Developed a Graphical User Interface (GUI) using Python's Tkinter library that provides secure and easy access to the system for administrators, faculty, and students
- Implemented user authentication and authorization protocols to ensure that each user can only access the data that they are authorized to see
- Designed a dashboard that provides an overview of important information such as student enrolment numbers, courses offered, and faculty information
- Created forms for adding new admissions, students, programs, courses, departments, and faculty members to the database
- Designed a search functionality that allows users to search for specific information based on various criteria such as student ID, course code, and department name
- Developed features for managing attendance records, grading, and course scheduling
- Tested the system thoroughly to ensure that it is functioning correctly and meeting our project requirements

Overall, our objective is to create a student information system that is user-friendly, secure, and efficient, and that can handle large amounts of data with ease. Our project aims to streamline the management of student data and provide valuable insights into the performance and progress of students and faculty alike.

Entity Relationship Model (ER Model):

Entity Relationship Diagram

Entity	Attributes
Student	Student_ID, FirstName, MiddleName, LastName, DOB, Gender, Age, Hostel_ID, Program_type
Faculty	Fac_ID, Fac_name, Designation, Office_hours, Dept_ID
Courses	Course_ID, Course_Name, Duration, Course_desc, Credits, Program_type
Department	Dept_ID, Dept_Name, Programs_offered, HOD, Office_Location
Admissions	Student_ID, Admission_Number, Application_date, Decision_Date, Test_written, Test_Score, Enrollment_Status
Programs	Program_ID, Program_Name, credits_required, Duration, Number_of_courses, Tution_Fee, Dept_ID
Grades_and_Attendance	Student_ID, Session_ID, Assignment_Score, Lab_score, Attendence, Quiz_Score, Student_grades, Class_ID
Student_Info	Student_ID, Mail, Phone, Father_Name, Mother_name, Address (Door, Street, City, State, Zip)
Sessions	Session_ID, Class_Location, Course_ID, Faculty_ID, Timings, Duration
Hostel	Hostel_ID, Hostel_Name, Amenities, Room_Number, Vacancy, Price_Range

Table 1 Entities and Attributes

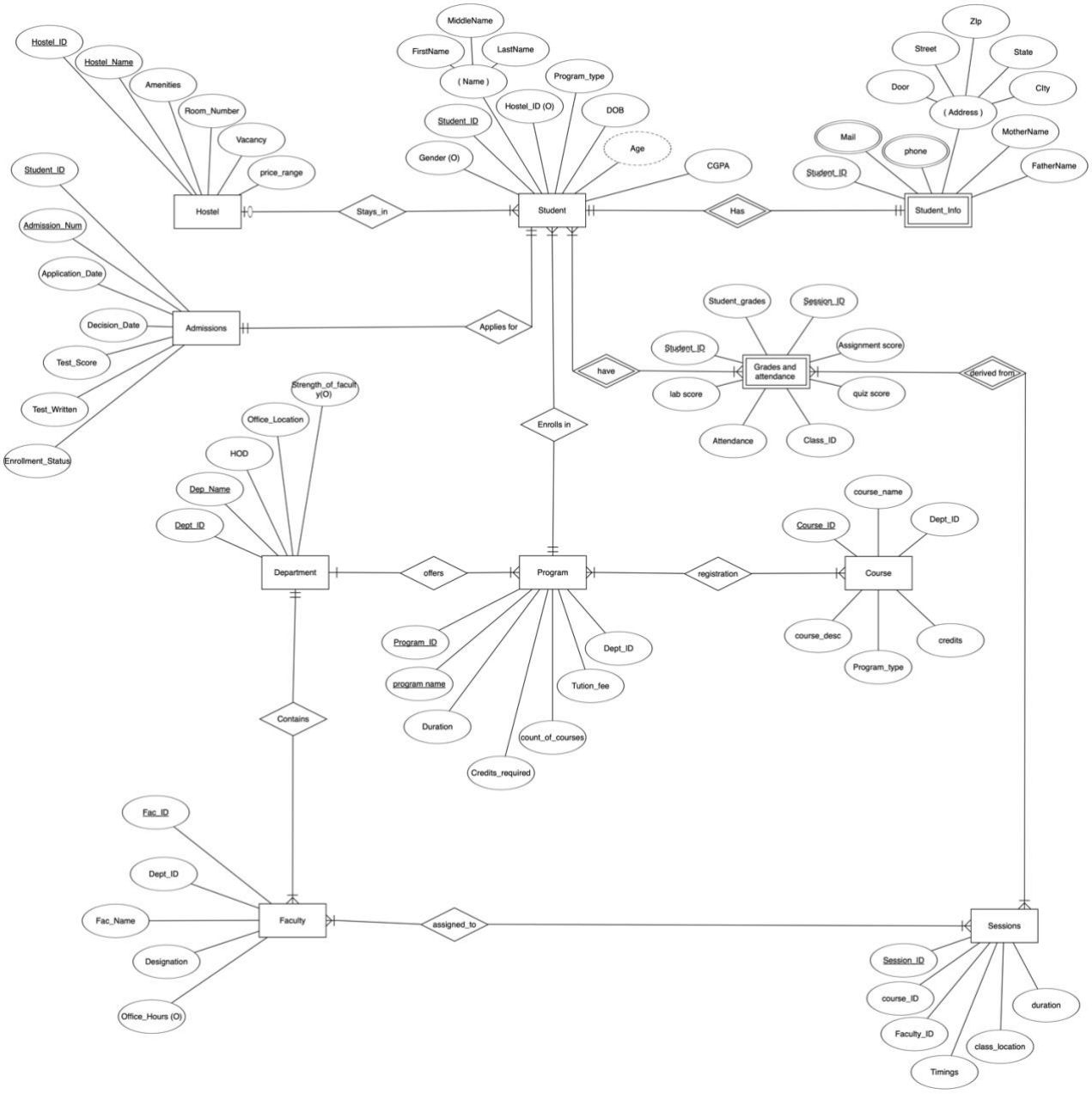


Figure 1 Conceptual design of Entity Relationships

Figure 1 shows the conceptual design of Entity Relationships using multiple entities, attributes and other important relations described below

Implementation of ER Diagram:

Entity: "Student"
Description: A person enrolled in an educational institution or program.
Attributes:
Student_ID: A unique identifier for each pupil.
FirstName: The student's given name or first name.
MiddleName: The student's middle name or initial.
LastName: The student's surname or last name.
Date of birth: The date of the student's birth.
The gender of the pupil, which is typically male or female.
Age: The student's age, calculated based on their date of birth.
Hostel_ID: The identification number for the student's hostel or dormitory, if applicable.
program_type: The type of program or course in which the student is currently enrolled.

These attributes are used to manage and monitor student data in the system of an educational institution.

Entity: “Faculty”
Description: A teacher or professor at an educational institution is described by this entity.
Attributes:
Fac_ID: The system-wide identifier for each faculty member.
FName: The given name or first name of the faculty member.
LName: The last name or surname of the faculty member.
Designation: The academic faculty member's employment title or position within the institution.
Office_hours: The time during which a faculty member is available to meet with students or other faculty members.
Dept_ID: The identifier for the academic department the faculty member belongs to.

These attributes are used to manage and monitor faculty information. The Fac_ID attribute is used to uniquely identify each faculty member. Name attributes (FName and LName) enable the system to store and retrieve faculty member names. The Designation attribute provides details regarding the faculty member's employment title or position. The Office_hours attribute enables the system to retain and retrieve information regarding faculty members' meeting availability. The Dept_ID attribute enables the system to store and retrieve data regarding the academic department to which the faculty member belongs.

Entity: "Course"
Description: A class or subject offered by the educational institution.
Attributes:
Course_ID: A unique identifier for each course in the system.
Course_Name: The name or title of the course.
Duration: The length of time the course is offered, such as "one semester" or "one year".
Course_desc: A brief description of the course content and objectives.
Credits: The number of credits awarded for successfully completing the course.
Program_type: The type of program or degree for which the course is required or elective.

In the educational institution's system, these attributes are utilized to manage and monitor course information. The Course_ID attribute is used to uniquely identify each course.

The Course_Name attribute specifies the course's name or title. The Duration attribute specifies the course's duration of availability. The Course_desc attribute provides a concise summary of the course's objectives and content. The Credits attribute denotes the quantity of credits awarded for completing the course successfully. The Program_type attribute describes the type of program or degree for which the course is required or optional.

Entity: "Department"
Description: An academic department at the educational institution.
Attributes:
Dept_ID: A unique identifier for each department in the system.
Dept_Name: The name of the department.
Courses_offered: The list of courses offered by the department.
HOD: The head of the department.
Office_Location: The location of the department's main office.

These attributes are used to administer and monitor departmental data within the system of an educational institution. The Dept_ID attribute functions as the department's unique identifier. The Dept_Name attribute specifies the department's name. The Courses_offered attribute lists the department's available courses. The HOD attribute designates the department chief. The Office_Location attribute specifies where the department's primary office is located.

Entity: "Admissions"
Description: The process of enrolling a student in the educational institution.
Attributes:
Student_ID: A unique identifier for the student who is applying for admission.
Admission_Number: A unique identifier assigned to the student upon admission to the institution.
Application_date: The date the student submitted the admission application.
Decision_Date: The date the admission decision was made.
Test_written: The type of test (if any) the student took as part of the admission process.
Test_Score: The score the student received on the admission test (if applicable).
Enrollment_Status: The status of the student's admission, such as "accepted", "rejected", or "waitlisted".

These attributes are used to oversee and monitor the admissions process for applicants to an educational institution. The Student_ID attribute functions as the student's unique identifier when applying for admission. Admission_Number is a unique identifier assigned to each pupil upon enrolment. The Application_date attribute specifies the date that the pupil submitted his or her application for admission. Decision_Date is the date that the admissions decision was made. The Test_written attribute indicates whether the applicant was required to take a test. The Test_Score attribute stores the student's grade on the admissions examination. (if applicable). The Enrollment_Status attribute stores the admission status of the pupil, such as "accepted," "rejected," or "waitlisted."

Entity: "Program"
Description: A course of study offered by the educational institution leading to a degree or diploma.
Attributes:
Program_ID: A unique identifier for each program in the system.
Program_Name: The name or title of the program.
Credits_required: The number of credits required to complete the program.
Duration: The length of time required to complete the program.
Number of Courses: The total number of courses required to complete the program.
Tuition_fee: The cost of tuition for the program.
Dept_ID: The identifier for the academic department that offers the program.

These attributes are utilized to administer and monitor program information within the system of an educational institution. The Program_ID attribute functions as the program's unique identifier. The Program_Name attribute specifies the program's name or title. The Credits_required attribute specifies the required quantity of credits to conclude the program. The Duration attribute specifies the amount of time necessary to finish the program. The Number of Courses attribute specifies the total number of required courses for program completion. The Tuition_fee attribute specifies the program's tuition fee. The Dept_ID attribute identifies the academic department offering the program.

Entity: "Grades and Attendance"
Description: A record of a student's academic performance and attendance in a specific class for a particular session.
Attributes:
Student_ID: A unique identifier for the student.
Session_ID: A unique identifier for the class session.
Assignment_score: The score the student received on their assignments.
Lab_score: The score the student received on their laboratory work.
Attendance: The percentage of classes the student attended.
Quiz_Score: The score the student received on their quizzes.
Student_grades: The overall grade the student received in the class.
Class_ID: The unique identifier for the class.

These attributes are used to manage and monitor a student's academic performance and attendance in a particular class for a specific session. The Student_ID attribute functions as a unique identifier for the student. The Session_ID attribute is the class session's unique identifier. The Assignment_score attribute stores the student's assignment grade. The Lab_score attribute stores the student's laboratory performance grade. The Attendance attribute indicates the percentage of classes the student attended. The Quiz_Score attribute stores the student's quiz score. The Student_grades attribute indicates the student's cumulative grade in the class. The Class_ID attribute serves as the class's unique identifier.

Entity: "Student Information"
Description: A record of personal information for a student.
Attributes:
Student_ID: A unique identifier for the student (primary key).
Mail: The email address of the student.
Phone: The phone number of the student.
Father_Name: The name of the student's father.

Mother_name: The name of the student's mother.
Address: The address of the student, including door number, street, city, state, and zip code.

These attributes are used to store a student's personal information. The Student_ID attribute functions as the student's unique identifier. (Primary key). The Mail attribute contains the student's email address. The Phone attribute contains the student's phone number. The Father_Name attribute holds the student's father's name. The Mother_name attribute holds the student's mother's name. The Address attribute stores the student's address, including the door number, street, city, state, and postal code.

Entity: "Session"
Description: A record of a class session for a course.
Attributes:
Session_ID: A unique identifier for the class session.
Class_Location: The location where the class is held.
Course_ID: A unique identifier for the course being taught in the session.
Faculty_ID: A unique identifier for the faculty member teaching the course.
Timings: The time of day when the class is held.
Duration: The length of the class session.

These attributes are used to manage and track a class session for a course. The class session is uniquely identified by the Session_ID property. The location of the class is kept in the Class_Location attribute. The course being taught in the session is uniquely identified by the Course_ID attribute. The faculty member who is instructing the course is uniquely identified by the Faculty_ID attribute. The class's time of day is indicated by the Timings attribute. Finally, the Duration attribute stores the length of the class session.

Entity: "Hostel"
Description: A record of a hostel or dormitory.
Attributes:
Hostel_ID: A unique identifier for the hostel (primary key).
Hostel_Name: The name of the hostel or dormitory.
Amenities: The facilities and services available to the students in the hostel.
Room_Number: The number of rooms in the hostel.
Vacancy: The number of available rooms in the hostel.
Price_Range: The price range for the rooms in the hostel.

These characteristics are employed to control and monitor data pertaining to a hotel or residence. The hostel's unique identifier is the Hostel_ID attribute. (Primary key). The name of the hostel or room is stored in the Hostel_Name property. The amenities attribute lists the amenities and services that the hostel's guests can use. The entire number of accommodations in the hotel is kept in the Room_Number attribute. The number of accommodations the hotel has accessible is indicated by the Vacancy attribute. The price range for the hostel's accommodations is lastly stored in the Price_Range property.

Multivalued Attributes:

For the Student_info which is a weak entity, there are 2 multivalued attributes that are mail and phone. A single student can have multiple phone numbers and mail addresses to be contacted so for our database design, these 2 are the multivalued attributes.

Composite Attributes:

For this design, we have Student name, Faculty name and Address as the composite attributes. Both student name and faculty name are further sub-divided as first name, middle name and last name. For address the sub attributes are door, street, zip, state and city.

Derived Attributes:

We have taken only one derived attribute that is age. Age is a derived attribute from DOB which is itself an attribute for the student entity

Weak Entity:

We have taken Student_info and Grades and Attendance as our weak entities for this database design. Both these weak entities are related to the student entity.

Strong entity:

The student, program, course entities are examples of strong entities. They do not rely on any other entities for representation and identification as they can exist uniquely.

Participations:

Total participation:

Every entity in the first entity set must have a relationship with at least one entity in the second entity set in order for there to be total involvement. In other words, there cannot be any entity in the first set without a corresponding entity in the second set. Total participation, also known as mandatory participation, is indicated by a double line joining the two entities. A student must have a matching record in the student information entity, and each record in the student information entity must be connected to a student, as in the case of a student and their personal information. Similarly, each student must have at least one admission record, at least one program enrolment record, and at least one course enrolment record. Double lines connecting the entities are used to indicate these relationships and total participation.

Partial participation:

A situation where a person or group participates in an activity or circumstance to some degree but not fully or completely is referred to as partial participation. Partial involvement in the context of a hostel and a student could imply that the student is residing in the hostel but is not actively participating in the events or socialising opportunities that the hostel offers.

For instance, a student may decide to reside in a hostel but spend the majority of their time studying by themselves in their rooms, rather than participating in the social events the hostel hosts. Alternatively, the student may select only those things that fit with their interests or preferences and take part in some but not all of them.

Cardinality Ratios:

One to one relationship:

One entity has a direct connection to another entity in such a way that one item in the first entity corresponds to one and only one item in the second entity, and vice versa. This is what is meant by the term "one-to-one relationship," which is a more straightforward explanation of the concept. For instance, in the context of students and student information, a one-to-one relationship can be established by having each student have their own unique collection of personal information (for example, their name, date of birth, and address), and each set of personal information corresponding to only one student. This indicates that there cannot be two or more individuals who share the same information regarding their personal details. The relationship between applicants and students provides yet another illustration of a one-to-one relationship. There can be only one student associated with an admittance record at any given time, and each individual student can have only one admission record. This indicates that a student cannot have numerous admission records, and that a single admission record cannot be linked to more than one student at a time.

One to many relationships:

A one-to-many relationship is one in which one entity is directly linked to another in that each item in the second entity corresponds to just one item in the first entity, but there are one or more items in the first entity. Hostels and students, for instance, can have a one-to-many relationship where one student can be allocated to only one hostel, but a hostel can house multiple students. This implies that while each hostel may be associated with multiple students, each student is only associated with one hostel. The connection between students and programs is another illustration of a one-to-many relationship. Although a student may enrol in numerous programs, only one group of students may be enrolled in any given program at any given time. This implies that each student can be connected to numerous programs, but that each program has a particular group of students that are connected to it.

Many to many relationships:

A many-to-many relationship is when several things in one entity can be linked to several items in another entity. In other words, every element in the first entity may be connected to a variety of elements in the second, and vice versa. A many-to-many connection can be created, for instance, in the context of programs and courses, where each program can offer a variety of courses, and each course can be offered by various programs. This implies that different programs may be linked to various studies. In the context of students, grades, and attendance, we can see another illustration of a many-to-many connection. Each student may take more than one class, receive more than one score, and have multiple students linked with each attendance record. As a result, a student may have numerous grades and attendance records, and multiple students may be linked to a single grade or attendance record. Last but not least, a many-to-many relationship between faculty and sessions can be created in which each faculty member can teach a variety of sessions, and each session can be taught by a

variety of faculty members. This implies that different staff members may be connected to various sessions.

Identifying and Non-Identifying Relationships:

The student and student_Info entities are in identifying relationship because the student_Info do not have a unique identifier to identify the entity uniquely. The admission entity and the student entity are in non-identifying relationship as both entities can be represented uniquely without dependency.

Enhanced Entity Relationship Model (EER Model):

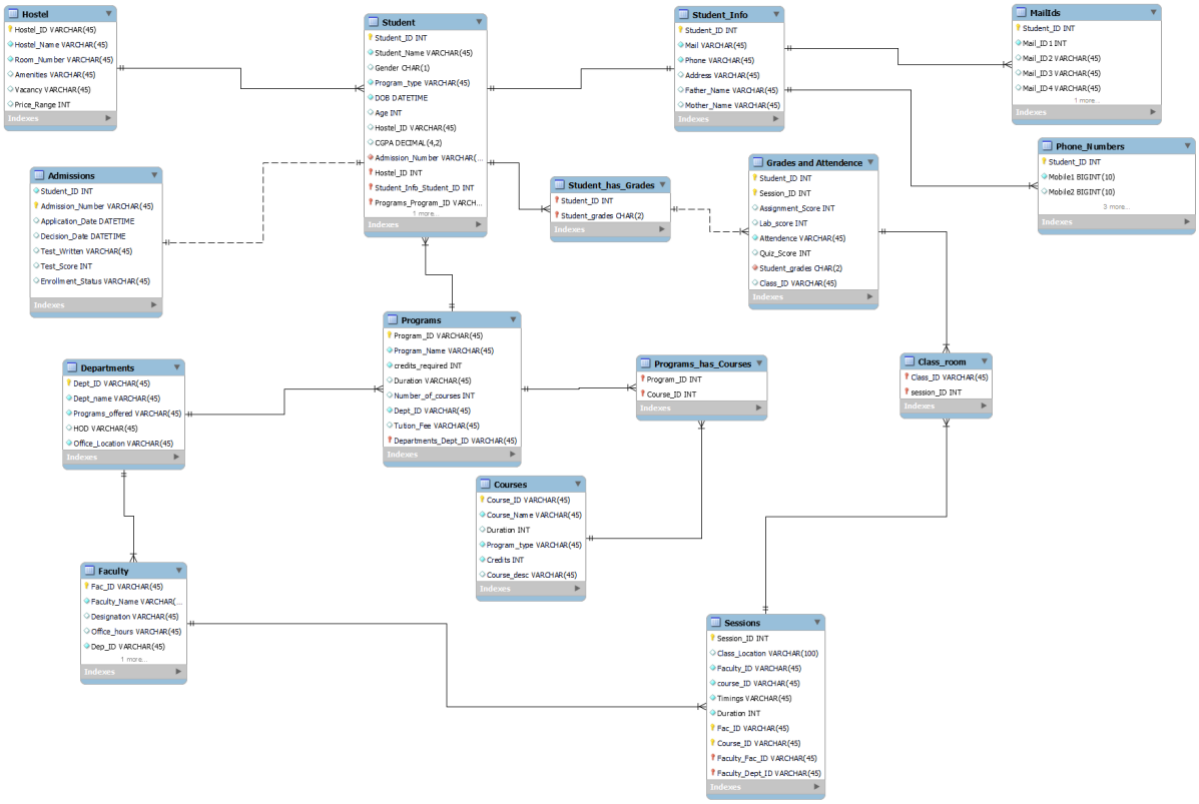


Figure 2 Enhanced Entity Relationship

Implementation of EER:

For creating an EER diagram, we used MySQL Workbench. Instead of going to the query tab we can go to the 2nd option in left menu of the Workbench through which we can design and view the EER diagram for our database. Using different tools, we can create the EER design as per our requirement [1].

The entity Student has an attribute hostel_id, which uniquely identifies each of the row in the entity Hostel, Program_type attribute is refferd as program_ID attribute in the "Program" entity, similarly each of the table interrelated to "Student" Entity can be uniquely related to other. Program. By the same pattern every entity can be related.

Every entity provided in the EER model is interrelated to each of the other entity, we are relating them using the primary and composite keys to manage the attributes in the entities. All the entities in the table except for "Grades and Attendance" has Primary Key. "Grades and Attendance" entity can be identified with the composite key by taking attributes "Student_ID" and "Session_ID". So, every entity in the model is uniquely identified, by means of which we can manage the data.

Database Development:

```
1
2   CREATE DATABASE IF NOT EXISTS SUPER_SIX;
3   USE SUPER_SIX;
4
```

Figure 3 Created a database

The snippet of code contains two SQL commands.

CREATE DATABASE IF NOT EXISTS SUPER_SIX is the first command; it creates a new database named "SUPER_SIX" if it does not already exist. The "IF NOT EXISTS" clause of this command guarantees the database is only created if it does not already exist.

USE SUPER_SIX is the second command; it selects the "SUPER_SIX" database for use. This indicates that subsequent SQL commands will be executed against this database.

Admission Table:

```
6   CREATE TABLE Admissions (
7   Student_ID INT NOT NULL,
8   Admission_Number VARCHAR(45) PRIMARY KEY,
9   Application_Date DATETIME,
10  Decision_Date DATETIME,
11  Test_Written VARCHAR(45),
12  Test_Score INT,
13  Enrollment_Status VARCHAR(45)
14  );
```

Figure 4 Admission table code

The code snippet creates a table called "Admissions" with columns representing various attributes related to student admissions.

The table has seven columns:

1. "Student_ID" column of type INT which is not nullable (i.e., must contain a value). This column represents the unique identifier of the student applying for admission.
2. "Admission_Number" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier assigned to the admission application.
3. "Application_Date" column of type DATETIME which represents the date and time when the application was submitted.
4. "Decision_Date" column of type DATETIME which represents the date and time when the admission decision was made.
5. "Test_Written" column of type VARCHAR(45) which represents the name of the test written by the student as part of the admission process.
6. "Test_Score" column of type INT which represents the score obtained by the student on the admission test.
7. "Enrollment_Status" column of type VARCHAR(45) which represents the status of the student's enrolment (e.g., accepted, rejected, waitlisted).

The primary key for the table is the "Admission_Number" column, which ensures that each admission application has a unique identifier. This column will be used to uniquely identify each row in the table.
There are no foreign keys or relationships defined in this table.

Hostel Table:

```
16  CREATE TABLE Hostel (
17      Hostel_ID VARCHAR(45) PRIMARY KEY,
18      Hostel_Name VARCHAR(45) NOT NULL,
19      Room_Number VARCHAR(45),
20      Amenities VARCHAR(45),
21      Vacancy VARCHAR(45),
22      Price_Range INT
23      );
```

Figure 5 Hostel table code

The code snippet creates a table called "Hostel" with columns representing various attributes related to hostels.
The table has six columns:

1. "Hostel_ID" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier of the hostel.
2. "Hostel_Name" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the name of the hostel.
3. "Room_Number" column of type VARCHAR(45) which represents the room number of the hostel.
4. "Amenities" column of type VARCHAR(45) which represents the amenities available in the hostel.
5. "Vacancy" column of type VARCHAR(45) which represents the vacancy status of the hostel.
6. "Price_Range" column of type INT which represents the price range of the hostel.

The primary key for the table is the "Hostel_ID" column, which ensures that each hostel has a unique identifier. This column will be used to uniquely identify each row in the table

Departments Table:

```
25  CREATE TABLE Departments (
26      Dept_ID VARCHAR(45) PRIMARY KEY,
27      Dept_name VARCHAR(45) NOT NULL,
28      programs_offered VARCHAR(45) NOT NULL,
29      HOD VARCHAR(45),
30      Office_Location VARCHAR(45)
31  );
```

Figure 6 Departments table code

The code snippet creates a table called "Departments" with columns representing various attributes related to academic departments.
The table has five columns:

1. "Dept_ID" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier of the department.
2. "Dept_name" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the name of the department.
3. "programs_offered" column of type VARCHAR(45) which is not nullable. This column represents the academic programs offered by the department.
4. "HOD" column of type VARCHAR(45) which represents the name of the head of department.
5. "Office_Location" column of type VARCHAR(45) which represents the location of the departmental office.

The primary key for the table is the "Dept_ID" column, which ensures that each department has a unique identifier. This column will be used to uniquely identify each row in the table.

Programs Table:

```
33  CREATE TABLE Programs (
34      Program_ID VARCHAR(45) PRIMARY KEY,
35      Program_Name VARCHAR(45) NOT NULL,
36      credits_required INT NOT NULL,
37      Duration VARCHAR(45),
38      Number_of_courses INT,
39      Tuition_Fee INT,
40      Department VARCHAR(45),
41      FOREIGN KEY (Department)
42      REFERENCES Departments (Dept_ID)
43  );
```

Figure 7 Programs table code

The code snippet creates a table called "Programs" with columns representing various attributes related to academic programs offered by a department.

The table has seven columns:

1. "Program_ID" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier of the program.
2. "Program_Name" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the name of the program.
3. "credits_required" column of type INT which is not nullable. This column represents the total number of credits required to complete the program.
4. "Duration" column of type VARCHAR(45) which represents the duration of the program.
5. "Number_of_courses" column of type INT which represents the number of courses in the program.
6. "Tuition_Fee" column of type INT which represents the tuition fee for the program.
7. "Department" column of type VARCHAR(45) which references the "Dept_ID" column in the "Departments" table using a foreign key constraint.

The primary key for the table is the "Program_ID" column, which ensures that each program has a unique identifier. This column will be used to uniquely identify each row in the table. The "Department" column in this table references the "Dept_ID" column in the "Departments" table using a foreign key constraint. This relationship indicates that each program belongs to a specific department. The foreign key constraint ensures that only valid department IDs are entered into the "Department" column of the "Programs" table.
This table could be used to store information about academic programs, such as their names, the total number of credits required, duration, number of courses, tuition fees, and the department to which they belong.

Student Table:

```
45  CREATE TABLE Student (
46  Student_ID INT PRIMARY KEY,
47  First_Name VARCHAR(45) NOT NULL,
48  Middle_Name VARCHAR(45),
49  Last_Name VARCHAR(45) NOT NULL,
50  Gender CHAR(1),
51  Program_type VARCHAR(45) NOT NULL,
52  DOB DATETIME NOT NULL,
53  Age INT,
54  Hostel_ID VARCHAR(45),
55  CGPA DECIMAL(4,2),
56  Admission_Number VARCHAR(45) NOT NULL,
57  FOREIGN KEY (Admission_Number)
58  REFERENCES Admissions (Admission_Number),
59  FOREIGN KEY (Hostel_ID)
60  REFERENCES Hostel (Hostel_ID),
61  FOREIGN KEY (Program_type)
62  REFERENCES Programs (Program_ID)
63  );
```

Figure 8 Student table code

The code snippet creates a table called "Student" with columns representing various attributes related to students enrolled in academic programs.
The table has eleven columns:

1. "Student_ID" column of type INT which is defined as the primary key of the table. This column represents the unique identifier of the student.
2. "First_Name" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the first name of the student.
3. "Middle_Name" column of type VARCHAR(45) which represents the middle name of the student.
4. "Last_Name" column of type VARCHAR(45) which is not nullable. This column represents the last name of the student.
5. "Gender" column of type CHAR(1) which represents the gender of the student.
6. "Program_type" column of type VARCHAR(45) which is not nullable. This column represents the academic program that the student is enrolled in.
7. "DOB" column of type DATETIME which is not nullable. This column represents the date of birth of the student.
8. "Age" column of type INT which represents the age of the student.
9. "Hostel_ID" column of type VARCHAR(45) which references the "Hostel_ID" column in the "Hostel" table using a foreign key constraint.
10. "CGPA" column of type DECIMAL(4,2) which represents the cumulative grade point average of the student.
11. "Admission_Number" column of type VARCHAR(45) which references the "Admission_Number" column in the "Admissions" table using a foreign key constraint.

The primary key for the table is the "Student_ID" column, which ensures that each student has a unique identifier. This column will be used to uniquely identify each row in the table.
The "Admission_Number" column in this table references the "Admission_Number" column in the "Admissions" table using a foreign key constraint. This relationship indicates that each student is associated with a specific admission record.
The "Hostel_ID" column in this table references the "Hostel_ID" column in the "Hostel" table using a foreign key constraint. This relationship indicates that each student may be associated with a specific hostel.
The "Program_type" column in this table references the "Program_ID" column in the "Programs" table using a foreign key constraint. This relationship indicates that each student is enrolled in a specific academic program.
This table could be used to store information about students, such as their names, gender, date of birth, age, academic program, admission number, CGPA, and hostel information.

Phone_Numbers Table:

```
76  CREATE TABLE Phone_Numbers(
77  Student_ID INT PRIMARY KEY,
78  mobile1 LONG NOT NULL,
79  mobile2 LONG,
80  mobile3 LONG,
81  mobile4 LONG,
82  FOREIGN KEY (Student_ID)
83  REFERENCES student(Student_ID)
84  );
```

Figure 9 Phone_numbers table code

The "Phone_Numbers" table is created to store multivalued attributes related to the "Student" table. It has five columns:

1. "Student_ID" column of type INT which is defined as the primary key of the table. This column references the "Student_ID" column in the "Student" table using a foreign key constraint. This relationship ensures that each row in the "Phone_Numbers" table corresponds to a unique student in the "Student" table.
2. "mobile1" column of type LONGwhich is not nullable (i.e., must contain a value). This column represents the primary mobile number of the student.
3. "mobile2" column of type LONG which represents a secondary mobile number of the student.
4. "mobile3" column of type LONGwhich represents a tertiary mobile number of the student.
5. "mobile4" column of type LONG which represents another mobile number of the student.

The primary key of the "Phone_Numbers" table is the "Student_ID" column, which ensures that each student has a unique entry in the table. The "Student_ID" column also serves as a foreign key that references the "Student" table. This relationship ensures that the mobile numbers stored in this table correspond to valid student records in the "Student" table.

Mail_ID Table:

```
86  CREATE TABLE Mail_ID(
87  Student_ID INT PRIMARY KEY,
88  Mail_ID1 VARCHAR(45) NOT NULL,
89  Mail_ID2 VARCHAR(45),
90  Mail_ID3 VARCHAR(45),
91  Mail_ID4 VARCHAR(45),
92  FOREIGN KEY (Student_ID)
93  REFERENCES student(Student_ID)
94  );
```

Figure 10 Mail_ID table code

Yes, the "Mail_ID" table is created to store multivalued attributes related to the "Student" table. It has five columns:

"Student_ID" column of type INT which is defined as the primary key of the table. This column references the "Student_ID" column in the "Student" table using a foreign key constraint. This relationship ensures that each row in the "Mail_ID" table corresponds to a unique student in the "Student" table.

1. "Mail_ID1" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the primary email address of the student.
2. "Mail_ID2" column of type VARCHAR(45) which represents a secondary email address of the student.
3. "Mail_ID3" column of type VARCHAR(45) which represents a tertiary email address of the student.
4. "Mail_ID4" column of type VARCHAR(45) which represents another email address of the student.

The primary key of the "Mail_ID" table is the "Student_ID" column, which ensures that each student has a unique entry in the table. The "Student_ID" column also serves as a foreign key that references the "Student" table. This relationship ensures that the email addresses stored in this table correspond to valid student records in the "Student" table.

Student_Info Table

```
CREATE TABLE Student_Info (
Student_ID INT PRIMARY KEY,
Address VARCHAR(45),
Father_Name VARCHAR(45),
Mother_Name VARCHAR(45),
FOREIGN KEY (Student_ID)
REFERENCES Student (Student_ID)
);
```

Figure 11 Student Info table code

The "Student_Info" table is created to store additional information about students that is not captured in other tables. It has six columns:

1. "Student_ID" column of type INT which is defined as the primary key of the table. This column references the "Student_ID" column in the "Student" table using a foreign key constraint. This relationship ensures that each row in the "Student_Info" table corresponds to a unique student in the "Student" table.
2. "Address" column of type VARCHAR(45) which represents the address of the student.
3. "Father_Name" column of type VARCHAR(45) which represents the name of the student's father.
4. "Mother_Name" column of type VARCHAR(45) which represents the name of the student's mother.

The primary key of the "Student_Info" table is the "Student_ID" column, which ensures that each student has a unique entry in the table. The "Student_ID" column also serves as a foreign key that references the "Student" table. This relationship ensures that the information stored in this table corresponds to valid student records in the "Student" table

Courses Table:

```
96  CREATE TABLE Courses (
97  Course_ID VARCHAR(45) PRIMARY KEY,
98  Course_Name VARCHAR(45) NOT NULL,
99  Duration INT,
100 Program_type VARCHAR(45) NOT NULL,
101 Credits INT NOT NULL,
102 Course_desc VARCHAR(150),
103 Department VARCHAR(45),
104 FOREIGN KEY (Department)
105 REFERENCES Departments (Dept_ID)
106 );
```

Figure 12 Courses table code

The code snippet creates a table called "Courses" with columns representing various attributes related to academic courses.

The table has seven columns:

1. "Course_ID" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier of the course.
2. "Course_Name" column of type VARCHAR(45) which is not nullable. This column represents the name of the course.
3. "Duration" column of type INT which represents the duration of the course.
4. "Program_type" column of type VARCHAR(45) which is not nullable. This column represents the academic program that the course is associated with.
5. "Credits" column of type INT which is not nullable. This column represents the number of credits associated with the course.
6. "Course_desc" column of type VARCHAR(150) which represents a brief description of the course.
7. "Department" column of type VARCHAR(45) which references the "Dept_ID" column in the "Departments" table using a foreign key constraint.

The primary key for the table is the "Course_ID" column, which ensures that each course has a unique identifier. This column will be used to uniquely identify each row in the table.

The "Department" column in this table references the "Dept_ID" column in the "Departments" table using a foreign key constraint. This relationship indicates that each course is associated with a specific department.

This table could be used to store information about academic courses, such as their names, duration, credit hours, program affiliation, department association, and a brief description.

Program_has_Courses Table:

```
109 • CREATE TABLE Program_has_Courses(
110   Course_ID VARCHAR(45),
111   Program_ID VARCHAR(45),
112   PRIMARY KEY(Course_ID,Program_ID),
113   FOREIGN KEY (Course_ID)
114   REFERENCES Courses(Course_ID),
115   FOREIGN KEY (Program_ID)
116   REFERENCES Programs(Program_ID)
117   );
```

Figure 13 Program_has_courses table code

The code snippet creates a junction table called "Program_has_Courses" with two foreign key constraints to the "Courses" and "Programs" tables.

The table has two columns:

1. "Course_ID" column of type VARCHAR(45) which references the "Course_ID" column in the "Courses" table using a foreign key constraint. This relationship indicates that each row in the "Program_has_Courses" table is associated with a specific course.
2. "Program_ID" column of type VARCHAR(45) which references the "Program_ID" column in the "Programs" table using a foreign key constraint. This relationship indicates that each row in the "Program_has_Courses" table is associated with a specific academic program.

The primary key for the table is a composite key consisting of both "Course_ID" and "Program_ID" columns. This ensures that each combination of course and program is unique in the table.

This table could be used to store information about which courses are included in which academic programs.

Faculty Table:

```
120 • CREATE TABLE Faculty (
121   Fac_ID VARCHAR(45) PRIMARY KEY,
122   Faculty_Name VARCHAR(45) NOT NULL,
123   Designation VARCHAR(45),
124   Office_hours VARCHAR(45),
125   Dept_ID VARCHAR(45) NOT NULL,
126   FOREIGN KEY (Dept_ID)
127   REFERENCES Departments (Dept_ID)
128   );
```

Figure 14 Faculty table code

This code snippet creates a table called "Faculty" with columns representing various attributes related to academic faculty.

The table has five columns:

1. "Fac_ID" column of type VARCHAR(45) which is defined as the primary key of the table. This column represents the unique identifier of the faculty.
2. "Faculty_Name" column of type VARCHAR(45) which is not nullable (i.e., must contain a value). This column represents the name of the faculty member.
3. "Designation" column of type VARCHAR(45) which represents the position held by the faculty member.
4. "Office_hours" column of type VARCHAR(45) which represents the office hours of the faculty member.
5. "Dept_ID" column of type VARCHAR(45) which is not nullable. This column references the "Dept_ID" column in the "Departments" table using a foreign key constraint. This column represents the department that the faculty member belongs to.

The primary key for the table is the "Fac_ID" column, which ensures that each faculty member has a unique identifier. This column will be used to uniquely identify each row in the table.

The "Dept_ID" column in this table references the "Dept_ID" column in the "Departments" table using a foreign key constraint. This relationship indicates that each faculty member belongs to a specific department.

This table could be used to store information about academic faculty, such as their names, positions, office hours, and department information.

Sessions Table:

```
141 • CREATE TABLE Sessions (
142     Session_ID INT PRIMARY KEY,
143     Class_Location VARCHAR(100),
144     Faculty_ID VARCHAR(45) NOT NULL,
145     Course_ID VARCHAR(45) NOT NULL,
146     Timings VARCHAR(45),
147     Duration INT,
148     FOREIGN KEY (Faculty_ID, Course_ID)
149     REFERENCES Faculty_teaches_Classes (Fac_ID, Course_ID)
150     );
```

Figure 15 Sessions table code

The code snippet creates a table called "Sessions" with columns representing various attributes related to academic sessions.

The table has six columns:

1. "Session_ID" column of type INT which is defined as the primary key of the table. This column represents the unique identifier of the session.
2. "Class_Location" column of type VARCHAR(100) which represents the location where the session will take place.
3. "Faculty_ID" column of type VARCHAR(45) which is not nullable. This column represents the unique identifier of the faculty member who will conduct the session.
4. "Course_ID" column of type VARCHAR(45) which is not nullable. This column represents the unique identifier of the course for which the session is being conducted.
5. "Timings" column of type TIME which represents the time when the session will begin.
6. "Duration" column of type INT which represents the duration of the session.

The "Faculty_ID" and "Course_ID" columns in this table reference the "Fac_ID" and "Course_ID" columns in the "Faculty_teaches_Classes" table using a foreign key constraint. This relationship indicates that each session is associated with a specific course that is being taught by a specific faculty member.

This table could be used to store information about academic sessions, such as the location, timings, and duration of the session, and the faculty member and course associated with it.

Class_room Table:

```
152 • CREATE TABLE Class_room(
153   Class_ID VARCHAR(45) NOT NULL,
154   Session_ID INT NOT NULL,
155   PRIMARY KEY (Class_ID,Session_ID),
156   FOREIGN KEY (Session_ID)
157   REFERENCES Sessions(Session_ID)
158   );
```

Figure 16 Class_room table code

This code creates a table called "Class_room" which contains information about the classrooms used for specific sessions.

The table has two columns:

1. "Class_ID" column of type VARCHAR(45) which represents the unique identifier of the classroom. This column cannot be null.
2. "Session_ID" column of type INT which represents the identifier of the session during which the classroom is used. This column cannot be null.

The primary key for the table is a combination of "Class_ID" and "Session_ID" columns, which ensures that each classroom is uniquely identified for each session.

The "Session_ID" column in this table references the "Session_ID" column in the "Sessions" table using a foreign key constraint. This relationship indicates that each classroom is associated with a specific session.

This table could be used to store information about the classrooms used for various sessions, such as the class location, faculty ID, course ID, timings, and duration.

Grades _and_Attendance Table:

```
160 • CREATE TABLE Grades_and_Attendance (
161     Student_ID INT NOT NULL,
162     Session_ID INT NOT NULL,
163     Assignment_Score INT,
164     Lab_score INT,
165     Attendance VARCHAR(45),
166     Quiz_Score INT,
167     Student_grades CHAR(2) NOT NULL,
168     Class_ID VARCHAR(45),
169     PRIMARY KEY (Student_ID, Session_ID),
170     FOREIGN KEY (Student_ID)
171     REFERENCES Student(Student_ID),
172     FOREIGN KEY (Session_ID)
173     REFERENCES Sessions (Session_ID),
174     FOREIGN KEY (Class_ID)
175     REFERENCES Class_room(Class_ID)
176     );
```

Figure 17 Grades_ and_Attendence table code

The code snippet creates a table called "Grades_and_Attendance" with columns representing various attributes related to the grades and attendance of students in a particular session.

The table has eight columns:

1. "Student_ID" column of type INT which represents the unique identifier of the student and is not nullable.
2. "Session_ID" column of type INT which represents the unique identifier of the session and is not nullable.
3. "Assignment_Score" column of type INT which represents the score obtained by the student in assignments.
4. "Lab_score" column of type INT which represents the score obtained by the student in labs.
5. "Attendance" column of type VARCHAR(45) which represents the attendance status of the student.
6. "Quiz_Score" column of type INT which represents the score obtained by the student in quizzes.
7. "Student_grades" column of type CHAR(2) which represents the final grade obtained by the student and is not nullable.
8. "Class_ID" column of type VARCHAR(45) which references the "Class_ID" column in the "Class_room" table using a foreign key constraint.

The primary key for the table is the combination of "Student_ID" and "Session_ID" columns, which ensures that each record represents a unique combination of student and session.

The "Student_ID" column in this table references the "Student_ID" column in the "Student" table using a foreign key constraint. This relationship indicates that each record in this table is associated with a specific student.

The "Session_ID" column in this table references the "Session_ID" column in the "Sessions" table using a foreign key constraint. This relationship indicates that each record in this table is associated with a specific session.

The "Class_ID" column in this table references the "Class_ID" column in the "Class_room" table using a foreign key constraint. This relationship indicates that each record in this table is associated with a specific class.

Students_has_grades Table:

```
178 • CREATE TABLE Student_has_grades(
179   Student_ID INT NOT NULL,
180   Student_grades CHAR(2) NOT NULL,
181   PRIMARY KEY(Student_ID,Student_grades),
182   FOREIGN KEY (Student_ID)
183   REFERENCES Student(Student_ID)
184   );
```

Figure 18 Students_has_grades table code

The code snippet creates a table called "Student_has_grades" with two columns that represent the relationship between the "Student" table and the "Grades_and_Attendance" table.

The table has two columns:

1. "Student_ID" column of type INT which is defined as the primary key of the table. This column represents the unique identifier of the student in the "Student" table.
2. "Student_grades" column of type CHAR(2) which is not nullable (i.e., must contain a value). This column represents the grades obtained by the student in the "Grades_and_Attendance" table.

The primary key for the table is a composite key that consists of both the "Student_ID" and "Student_grades" columns. This ensures that each combination of student and grades is unique.

This table could be used to store information about the grades obtained by each student in different courses.

Loading data and performance enhancements:

Handling foreign key constraints:

We addressed the foreign key inconsistency by setting foreign_key_checks to 0 and running the code, according to the instructions.

Initial run:

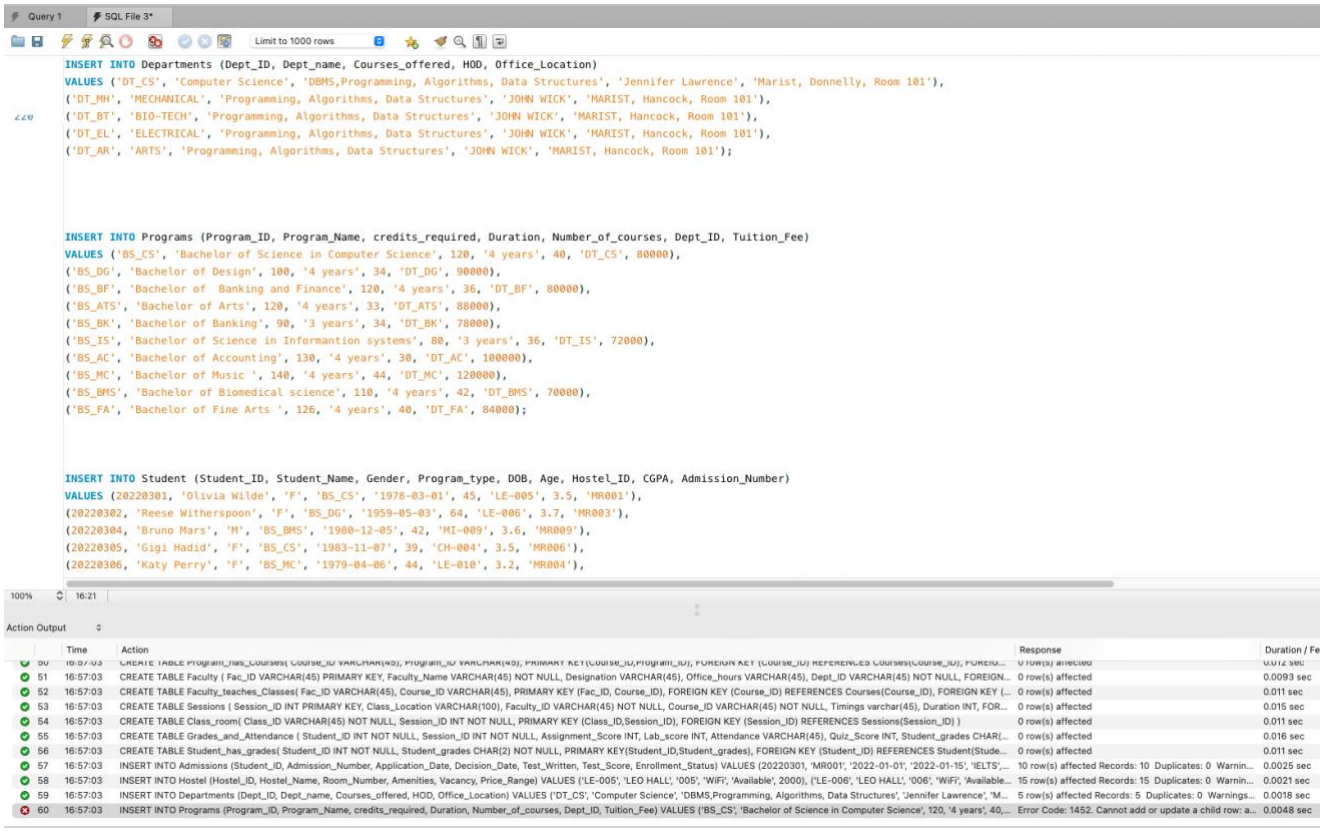
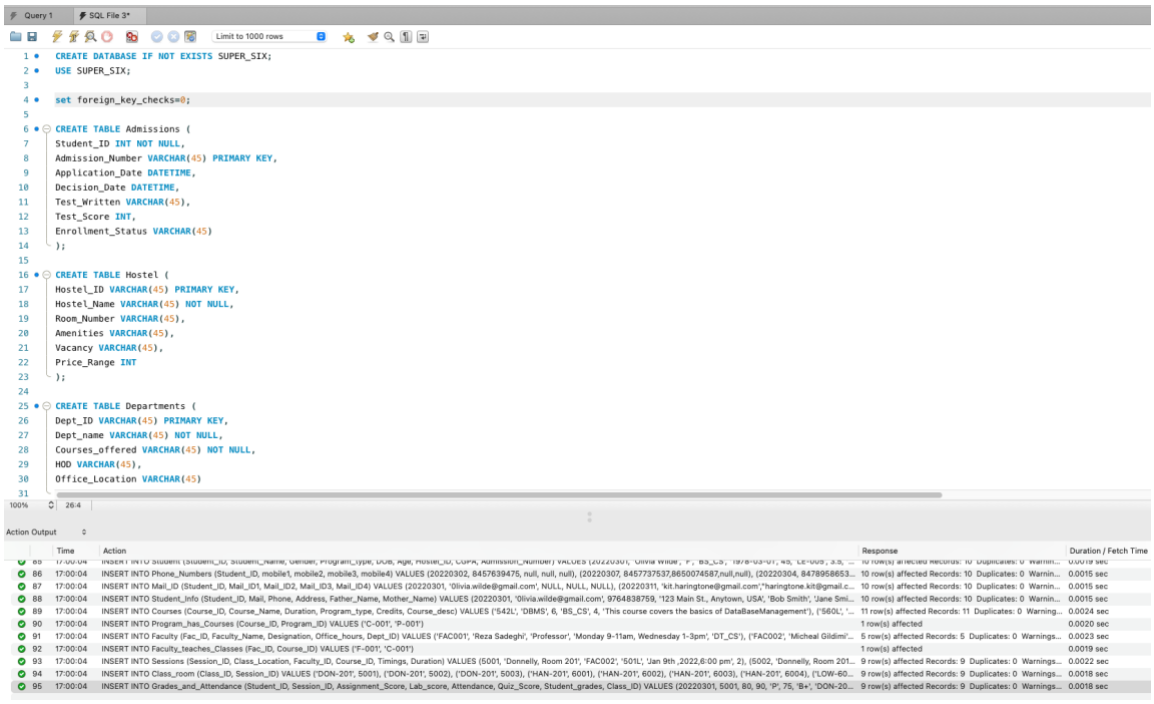


Figure 19 Handling foreign key constraints: Initial run

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`super_six`.`programs`, CONSTRAINT `programs_ibfk_1` FOREIGN KEY (`Dept_ID`) REFERENCES `departments` (`Dept_ID`)) 0.0048 sec

Setting foreign_key_checks = 0;



```
CREATE DATABASE IF NOT EXISTS SUPER_SIX;
USE SUPER_SIX;

set foreign_key_checks=0;

CREATE TABLE Admissions (
Student_ID INT NOT NULL,
Admission_Number VARCHAR(45) PRIMARY KEY,
Application_Date DATETIME,
Decision_Date DATETIME,
Test_Written VARCHAR(45),
Test_Score INT,
Enrollment_Status VARCHAR(45)
);

CREATE TABLE Hostel (
Hostel_ID VARCHAR(45) PRIMARY KEY,
Hostel_Name VARCHAR(45) NOT NULL,
Room_Number VARCHAR(45),
Amenities VARCHAR(45),
Vacancy VARCHAR(45),
Price_Range INT
);

CREATE TABLE Departments (
Dept_ID VARCHAR(45) PRIMARY KEY,
Dept_name VARCHAR(45) NOT NULL,
Courses_offered VARCHAR(45) NOT NULL,
HOD VARCHAR(45),
Office_Location VARCHAR(45)
```

Figure 20 Handling foreign key constraints

Resetting foreign_key_checks = 1;

Importing data:

Inserted data by creating ten instances for each table, which are in line with the schema defined in the phase-04. We inserted 10 instances in each table by following proper constraints, rules and limitations.

```
INSERT INTO Admissions (Student_ID, Admission_Number, Application_Date, Decision_Date, Test_Written, Test_Score, Enrollment_Status)
VALUES (20220301, 'MR001', '2022-01-01', '2022-01-15', 'IELTS', 7.5, 'Enrolled'),
(20220302, 'MR002', '2022-02-01', '2022-06-15', 'IELTS', 7, 'Enrolled'),
(20220304, 'MR003', '2022-01-03', '2022-06-15', 'IELTS', 6.5, 'Enrolled'),
(20220305, 'MR004', '2022-07-01', '2022-06-15', 'GRE', 295, 'Enrolled'),
(20220306, 'MR005', '2022-05-09', '2022-06-15', 'IELTS', 7, 'Enrolled'),
(20220303, 'MR006', '2022-04-01', '2022-06-16', 'TOEFL', 110, 'Enrolled'),
(20220307, 'MR007', '2022-02-03', '2022-06-16', 'PTE', 75, 'Enrolled'),
(20220311, 'MR008', '2022-05-06', '2022-06-16', 'GRE', 312, 'Enrolled'),
(20220312, 'MR009', '2022-05-05', '2022-06-16', 'IELTS', 6.5, 'Enrolled'),
(20220315, 'MR010', '2022-06-09', '2022-06-16', 'PTE', 78, 'Enrolled');
```

```
INSERT INTO Hostel (Hostel_ID, Hostel_Name, Room_Number, Amenities, Vacancy, Price_Range)
VALUES ('LE-005', 'LEO HALL', '005', 'WiFi', 'Available', 2000),
('LE-006', 'LEO HALL', '006', 'WiFi', 'Available', 2000),
('LE-009', 'LEO HALL', '009', 'WiFi', 'Available', 2000),
('LE-010', 'LEO HALL', '010', 'WiFi', 'Available', 2000),
('MI-007', 'MIDRISE HALL', '007', 'WiFi, AC, Gym', 'Available', 4000),
('MI-008', 'MIDRISE HALL', '008', 'WiFi, AC, Gym', 'Available', 4000),
('MI-009', 'MIDRISE HALL', '009', 'WiFi, AC, Gym', 'Available', 4000),
('MA-005', 'MARIUM HALL', '005', 'WiFi, AC', 'Available', 3000),
('MA-006', 'MARIUM HALL', '006', 'WiFi, AC', 'Available', 3000),
('MA-001', 'MARIUM HALL', '001', 'WiFi, AC', 'Available', 3000),
('MA-009', 'MARIUM HALL', '009', 'WiFi, AC', 'Available', 3000),
('CH-004', 'CHAMPAGNAT HALL', '004', 'WiFi, AC, Gym', 'Available', 3500),
('CH-005', 'CHAMPAGNAT HALL', '005', 'WiFi, AC, Gym', 'Available', 3500),
('SH-007', 'SHEAHAN HALL', '007', 'WiFi, Gym', 'Available', 2500),
('SH-003', 'SHEAHAN HALL', '003', 'WiFi, Gym', 'Available', 2500);
```

```
INSERT INTO Programs (Program_ID, Program_Name, credits_required, Duration, Number_of_courses, Dept_ID, Tuition_Fee)
VALUES ('BS_CS', 'Bachelor of Science in Computer Science', 120, '4 years', 40, 'DT_CS', 80000),
('BS_DG', 'Bachelor of Design', 100, '4 years', 34, 'DT_DG', 90000),
('BS_BF', 'Bachelor of  Banking and Finance', 120, '4 years', 36, 'DT_BF', 80000),
('BS_ATS', 'Bachelor of Arts', 120, '4 years', 33, 'DT_ATS', 88000),
('BS_BK', 'Bachelor of Banking', 90, '3 years', 34, 'DT_BK', 78000),
('BS_IS', 'Bachelor of Science in Informantion systems', 80, '3 years', 36, 'DT_IS', 72000),
('BS_AC', 'Bachelor of Accounting', 130, '4 years', 30, 'DT_AC', 100000),
('BS_MC', 'Bachelor of Music ', 140, '4 years', 44, 'DT_MC', 120000),
('BS_BMS', 'Bachelor of Biomedical science', 110, '4 years', 42, 'DT_BMS', 70000),
('BS_FA', 'Bachelor of Fine Arts ', 126, '4 years', 40, 'DT_FA', 84000);
```

```
INSERT INTO Departments (Dept_ID, Dept_name, Courses_offered, HOD, Office_Location)
VALUES ('DT_CS', 'Computer Science', 'DBMS,Programming, Algorithms, Data Structures', 'Jennifer Lawrence', 'Marist, Donnelly, Room 101')
('DT_MH', 'MECHANICAL', 'Programming, Algorithms, Data Structures', 'JOHN WICK', 'MARIST, Hancock, Room 101'),
('DT_BT', 'BIO-TECH', 'Programming, Algorithms, Data Structures', 'JOHN WICK', 'MARIST, Hancock, Room 101'),
('DT_EL', 'ELECTRICAL', 'Programming, Algorithms, Data Structures', 'JOHN WICK', 'MARIST, Hancock, Room 101'),
('DT_AR', 'ARTS', 'Programming, Algorithms, Data Structures', 'JOHN WICK', 'MARIST, Hancock, Room 101');
```

```
INSERT INTO Student (Student_ID, Student_Name, Gender, Program_type, DOB, Age, Hostel_ID, CGPA, Admission_Number)
VALUES (20220301, 'Olivia Wilde', 'F', 'BS_CS', '1978-03-01', 45, 'LE-005', 3.5, 'MR001'),
(20220302, 'Reese Witherspoon', 'F', 'BS_DG', '1959-05-03', 64, 'LE-006', 3.7, 'MR003'),
(20220304, 'Bruno Mars', 'M', 'BS_BMS', '1980-12-05', 42, 'MI-009', 3.6, 'MR009'),
(20220305, 'Gigi Hadid', 'F', 'BS_CS', '1983-11-07', 39, 'CH-004', 3.5, 'MR006'),
(20220306, 'Katy Perry', 'F', 'BS_MC', '1979-04-06', 44, 'LE-010', 3.2, 'MR004'),
(20220303, 'Natalie Portman', 'F', 'BS_CS', '1985-01-23', 37, 'SH-007', 3.1, 'MR002'),
(20220312, 'Demi Moore', 'F', 'BS_FA', '1979-03-02', 41, 'MA-006', 3.0, 'MR008'),
(20220307, 'Joaquin Pheonix', 'F', 'BS_CS', '1969-04-06', 54, 'MI-008', 3.9, 'MR007'),
(20220311, 'Kit Harington', 'M', 'BS_MC', '1973-06-15', 29, 'MA-009', 3.3, 'MR010'),
(20220315, 'Harry Potter', 'M', 'BS_IS', '1989-11-11', 34, 'SH-003', 3.7, 'MR005');
```

```
INSERT INTO Phone_Numbers (Student_ID, mobile1, mobile2, mobile3, mobile4)
VALUES (20220302, 8457639475, null, null, null),
(20220307, 8457737537,8650074587,null,null),
(20220304, 84789586530,null,7759789044,8987459905),
(20220311, 8540832865,null,null,7849666499),
(20220306, 8974605198,7805284968,8757783300,7800954349),
(20220305, 8867489509,8037600086,8975774883,8765483847),
(20220315, 9778477494,8786784565,null,9754678900),
(20220303, 8475874994,8476776408,null,null),
(20220301, 9764838759,8886374834,7980947748,null),
(20220312, 8476767485,7879999990,8875390384,null);
```

```
INSERT INTO Mail_ID (Student_ID, Mail_ID1, Mail_ID2, Mail_ID3, Mail_ID4)
VALUES (20220301, '0livia.wilde@gmail.com', NULL, NULL, NULL),
(20220311, 'kit.haringtone@gmail.com',"haringtone.kit@gmail.com","kit.haringtone1973@gmail.com",null),
(20220307, 'Joaquin.Pheonix@gmail.com',"Pheonix.Joaquin1969@gmail.com",null,null),
(20220315, 'Harry.Potter@marist.edu','Potter.Harry1989@gmail.com',null,null),
(20220312, 'Demi.Moore@marist.edu','Demi.Moor1979@yahoo.com','D.moore@gmial.com',null),
(20220306, 'Katy.Perry1979@marist.edu','Katy.Perry79@gmail.com',null,null),
(20220303, 'Nat.man@gmail.com','Natalie.Portman@gmail.com','N.Portman1985@yahoo.com',null),
(20220305, 'Gigi.Hadid@marist.edu','Hadid.gigi1983@yahoo.com',null,null),
(20220302, 'Reese.Witherspoon@marist.edu','Witherspoon.1959@gmail.com',"Reese.Wbddg@gmail.com",null),
(20220304, 'Bruno.Mars@marist.edu','Bruno.Mars1980@gmail.com',null,null);
```

```
INSERT INTO Student_Info (Student_ID, Mail, Phone, Address, Father_Name, Mother_Name)
VALUES (20220301, '0livia.wilde@gmail.com', 9764838759, '123 Main St., Anytown, USA', 'Bob Smith', 'Jane Smith'),
(20220311, 'kit.haringtone@gmail.com', 8540832865,'45 Noth clover USA','jack harington','jane harington'),
(20220307, 'Joaquin.Pheonix@gmail.com', 8457737537,'29 orchid ,USA, 12605','Dark Pheonix','Light Pheonix'),
(20220304, 'Bruno.Mars@marist.edu', 84789586530,null,null,null),
(20220305, 'Gigi.Hadid@marist.edu', 8867489509,null,'True Hadid','false Hadid'),
(20220302, 'Reese.Witherspoon@marist.edu', 8457639475,'75 Vernon street, NY ,USA',null,null),
(20220315, 'Harry.Potter@marist.edu', 9778477494,'Hogwarts, England','James Potter','Lily Potter'),
(20220306, 'Katy.Perry1979@marist.edu', 8974605198,null,'john perry','Don perry'),
(20220312, 'Demi.Moore@marist.edu', 8476767485,'24 Main street,USA','Yes Moore','No moore'),
(20220303, 'Nat.man@gmail.com', 8475874994,null,null,null);
```

```
INSERT INTO Courses (Course_ID, Course_Name, Duration, Credits, Course_desc)
VALUES ('542L', 'DBMS', 6, 4, 'This course covers the basics of DataBaseManagement'),
('560L', 'Networking', 6, 4, 'Netwotk designing'),
('501L', 'OOP', 4, 8, 'Ojective oriented programming'),
('521s', 'Data Mining', 10, 4, 'data extraction'),
('522s', 'Emerging Technologies', 6, 6, 'About Trending technologies'),
('570F', 'Accounting', 12, 4, 'Manage Accounts'),
('580F', 'Commication systems', 6, 4, null),
('520G', 'Designing object', 6, 4, 'plan for designing'),
('578A', 'Law Fundamentals', 18, 8, 'Fundamentals of law'),
('530C', 'Acoustics', 4, 4, 'Sound sytems'),
('540B', 'Anatomy', 12, 12, 'Human beings');
```

```
INSERT INTO Program_has_Courses (Course_ID, Program_ID)
VALUES ('542L', 'BS_CS'),
('542L', 'BS_IS'),
('560L','BS_CS'),
('560L','BS_IS'),
('501L','BS_CS'),
('521s', 'BS_IS'),
('522s', 'BS_IS'),
('522s', 'BS_MC'),
('570F', 'BS_BF'),
('570F', 'BS_IS'),
('580F', 'BS_BF'),
('520G', 'BS_DG'),
('578A', 'BS_ATS'),
```

```
INSERT INTO Faculty (Fac_ID, Faculty_Name, Designation, Office_hours, Dept_ID)
VALUES ('FAC001', 'Reza Sadeghi', 'Professor', 'Monday 9-11am, Wednesday 1-3pm', 'DT_CS'),
('FAC002', 'Micheal Gildimi', 'Professor', 'Monday 9-11am, Wednesday 1-3pm', 'DT_MH'),
('FAC003', 'Sandhya Aneja', 'Juniopr Lecteror', 'Monday 2-5pm', 'DT_BT'),
('FAC004', 'Jennifer Lawrence', 'HOD', 'Monday 2-5pm', 'DT_EL'),
('FAC005', 'Vishwanath Anand', 'proffesor', 'Monday 2-5pm', 'DT_AR');
```

```
INSERT INTO Sessions (Session_ID, Class_Location, Faculty_ID, Course_ID, Timings, Duration)
VALUES (5001, 'Donnelly, Room 201', 'FAC002', '501L', 'Jan 9th ,2022,6:00 pm', 2),
(5002, 'Donnelly, Room 201', 'FAC002', '501L', 'Jan 16th ,2022, 6:00 pm', 2),
(5003, 'Donnelly, Room 201', 'FAC002', '501L', 'Jan 23th ,2022, 6:00 pm', 2),
(6001, 'Hancock, Room 501', 'FAC001', '542L', 'Jan 20th ,2023, 6:30 pm', 3),
(6002, 'Hancock, Room 501', 'FAC001', '542L', 'Jan 27th ,2023, 6:30 pm', 3),
(6003, 'Hancock, Room 501', 'FAC001', '542L', 'Feb 4th ,2023, 6:30 pm', 3),
(6004, 'Hancock, Room 501', 'FAC001', '542L', 'Feb 11th ,2023 ,6:30 pm', 3),
(7001, 'Lowell thomas, Room 601', 'FAC003', '560L', 'Jan 19th ,2023, 3:30 pm', 3),
(7002, 'Lowell thomas, Room 601', 'FAC003', '560L', 'Jan 26th ,2023, 3:30 pm', 3);
```

```
INSERT INTO Class_room (Class_ID, Session_ID)
VALUES ('DON-201', 5001),
('DON-201', 5002),
('DON-201', 5003),
('HAN-201', 6001),
('HAN-201', 6002),
('HAN-201', 6003),
('HAN-201', 6004),
('LOW-601', 7001),
('LOW-601', 7002);
```

```
INSERT INTO Grades_and_Attendance (Student_ID, Session_ID, Assignment_Score, Lab_score, Attendance, Quiz_Score, Student_grades, Class_ID)
VALUES (20220301, 5001, 80, 90, 'P', 75, 'B+', 'DON-201'),
(20220301, 5002, 60, 70, 'P', 75, 'C+', 'DON-201'),
(20220301, 5003, 50, 60, 'P', 75, 'D+', 'DON-201'),
(20220302, 5001, 90, 95, 'P', 75, 'A+', 'DON-201'),
(20220302, 5002, 80, 90, 'P', 75, 'B+', 'DON-201'),
(20220302, 5003, 50, 65, 'P', 75, 'D', 'DON-201'),
(20220305, 5001, 90, 90, 'P', 75, 'A', 'DON-201'),
(20220305, 5002, 80, 90, 'P', 75, 'B+', 'DON-201'),
(20220305, 5003, 60, 75, 'P', 75, 'C', 'DON-201');
```

Insertion optimization:

There are several ways to insert rows into a SQL database table, each with its own advantages and disadvantages. Here are some of the most common types of insertions:

Single-row insertion: This method involves inserting one row at a time using the INSERT statement. This method is straightforward and easy to use but can be slow when inserting large amounts of data.

Bulk insertion: Bulk insertion involves inserting multiple rows at once using a single INSERT statement. This method can be much faster than single-row insertion, especially when inserting large amounts of data. However, it can be more complex to set up and requires careful attention to the data being inserted.

Subquery insertion: Subquery insertion involves using a SELECT statement to retrieve data from one or more tables and then inserting the results into another table. This method is useful when you need to combine data from multiple tables or perform complex transformations on the data before inserting it.

Temporary table insertion: This method involves creating a temporary table, inserting the data into the temporary table, and then using the temporary table to insert the data into the main table. This method is useful when dealing with large amounts of data that require complex transformations or validations.

Copy from file insertion: This method involves inserting data into a table from a file. This can be useful when you have data in an external file that you need to insert into a table.

Based on the database which we have created, we think Bulk insertion is an effective way to insert rows into the tables. We analysed the duration of different ways of insertions. when we tried single row insertion on Admissions table, it took 0.01829 Seconds for inserting 10 rows. When we use the bulk insertion approach, it took only 0.015 Seconds to insert 10 rows in the Admissions table. So, we used the Bulk insertion technique which we think is effective for our database.

Figure 21 Insertion Optimization time for Single-row Insertion

Figure 22 Insertion Optimization time for Bulk-rows Insertion

Normalization Check:

First Normal Form (1NF) is a property of a relational database that ensures that all tables in the database contain only atomic (indivisible) values. In other words, each table cell must contain a single, indivisible value and not a set of values or a list of values. Additionally, each column in a table must have a unique name, and the order in which data is stored in the table does not matter. By ensuring that all data in the database is atomic and organised into tables, 1NF helps to minimise data redundancy and improve data consistency and accuracy.

Based on the given database schema, the tables appear to satisfy the First Normal Form (1NF) requirements. Here are some reasons:

1. Each table has a primary key that uniquely identifies each row.
2. All columns in a table have atomic values, meaning each column contains only one value per row and does not contain multiple values or lists.
3. There are no repeating groups or arrays in any of the tables.
4. All columns in a table have the same data type.

Second Normal Form (2NF) is a database normalization technique used to eliminate data redundancy and improve data integrity.

A relation is in 2NF if it satisfies the following two conditions:

1. It is in 1NF.
2. All non-key attributes are fully functionally dependent on the primary key.
3. In other words, every non-key attribute in a 2NF relation must be dependent on the whole primary key, not just on a part of it. If a non-key attribute is dependent on only a part of the primary key, the relation violates the second normal form and must be decomposed into smaller, 2NF relations.
4. To check if a table is in 2NF, we need to do the following:
5. Check if the table is in 1NF

6. Identify the table's primary key and any other attributes that are functionally dependent on it
7. Check if any non-key attributes are dependent on only a part of the primary key
8. If any non-key attributes are dependent on only a part of the primary key, move them to a separate table with their dependent part of the primary key as the new primary key.

The database tables satisfy 2NF as each non-key attribute depends on the whole primary key, and there are no partial dependencies. The tables have been normalized to eliminate any redundant data and ensure data consistency.

2NF is satisfied because all non-key attributes are fully dependent on the primary key of each table. For example, in the "Admissions" table, all non-key attributes (Application_Date, Decision_Date, Test_Written, Test_Score, Enrollment_Status) are fully dependent on the primary key (Admission_Number).

Third normal form (3NF) is a database normalization technique that aims to minimize data redundancy in a relational database. It builds on the first and second normal forms (1NF and 2NF) by ensuring that each non-key attribute in a table is dependent only on the primary key, and not on any other non-key attribute.

In other words, a table is in 3NF if it meets the following criteria:

1. It is in second normal form (2NF).
2. All non-key attributes in the table are dependent on the primary key.
3. There are no transitive dependencies between non-key attributes.

   By eliminating the transitive dependencies between non-key attributes, 3NF helps to reduce the possibility of data inconsistencies and anomalies that can arise when updating or deleting data in the database.

3NF is also satisfied because there are no transitive dependencies between non-key attributes. For example, in the "Programs" table, the non-key attribute "Dept_ID" is directly related to the primary key, and there is no other non-key attribute that is dependent on "Dept_ID."

To provide some examples:

- A violation of 2NF could occur if a non-key attribute depends on only a part of the primary key, leading to partial dependencies. For instance, if in the "Student_Info" table, the "Address" attribute was dependent on only the "Phone" attribute, but not the "Student_ID" attribute, then there would be a violation of 2NF.
- A violation of 3NF could occur if a non-key attribute is dependent on another non-key attribute in the same table, leading to transitive dependencies. For example, if in the "Faculty" table, the "Office_hours" attribute was dependent on the "Designation" attribute, rather than the primary key "Fac_ID," then there would be a violation of 3NF.

Application development:

Graphical user experience design:

Main Menu:

The main menu page interface serves as a hub for the key options available in the application. These options encompass various functions such as login pages for both students and faculty, admission status updates for students, and descriptive pages providing detailed views of different departments and programs. In essence, the main menu page is the gateway that enables users to navigate through the various features and functionalities of the application.

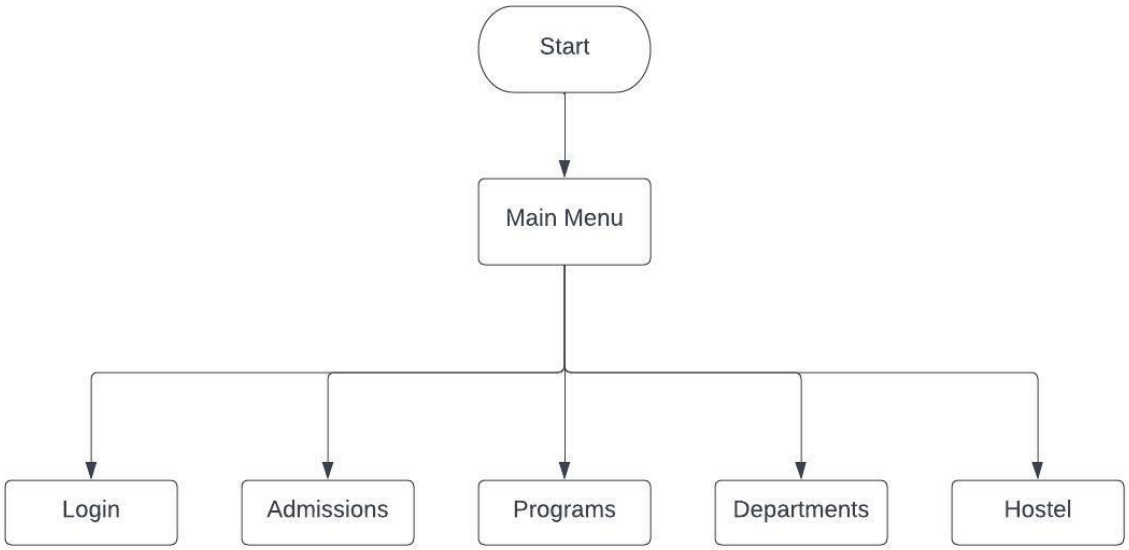


Figure 23 Main Menu Page Flow Chart

Login:

Upon accessing the login page, the user is presented with the option to select between student and faculty login. Depending on the user's selection, they will be redirected to either the student login page or the faculty login page. In other words, the login page serves as a gateway to the appropriate login portal based on the user's chosen category.

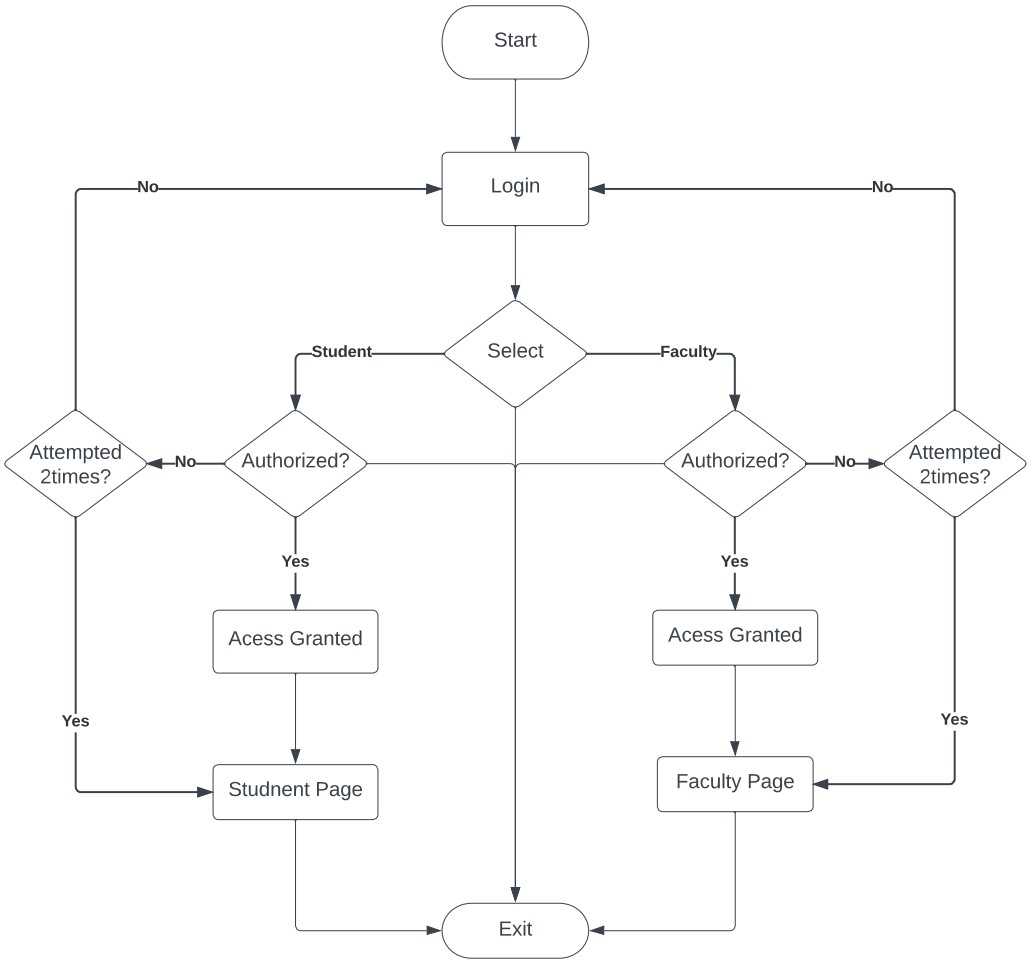


Figure 24 Main Login Page Flow Chart

Student Login

The student login page features several sections including the profile, courses, sessions, and logout. Within the profile section, the user can access their personal details. The courses section provides a list of all the courses the student is currently enrolled in and allows them to navigate to the course page for a more detailed view. Additionally, the sessions section displays a comprehensive list of all the sessions the student has attended. Overall, the student login page provides a user-friendly interface for students to access and manage their academic information.

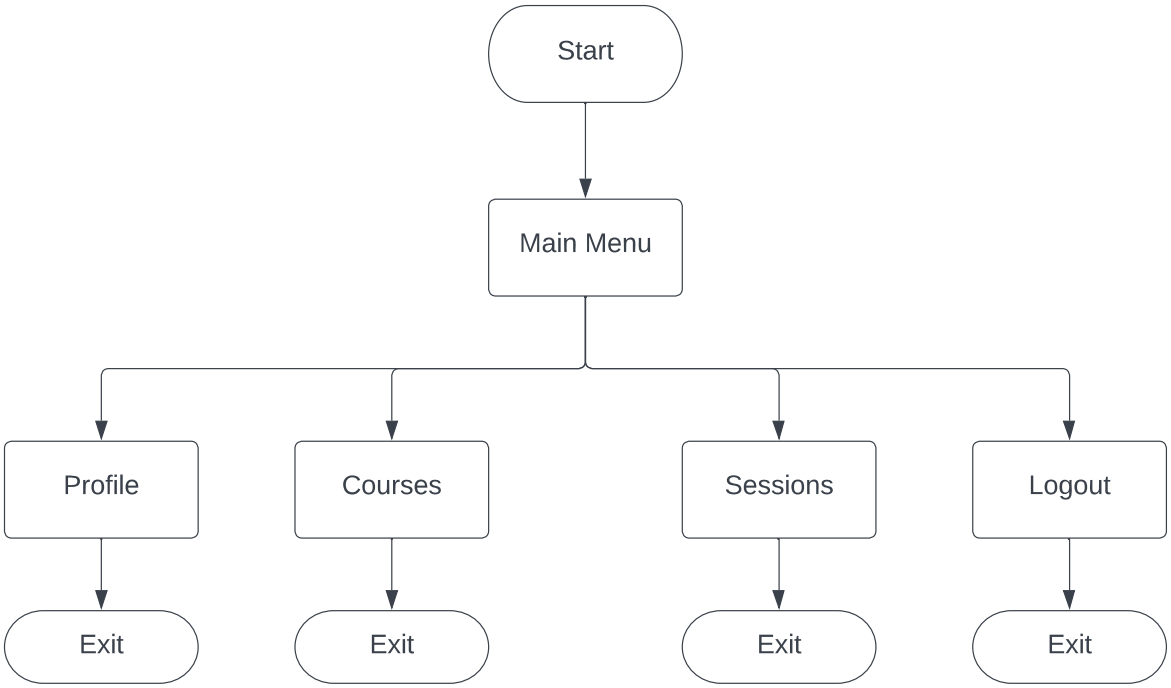


Figure 25 Student Login Page Flow Chart

Faculty login:

The faculty login page contains three main sections: profile, courses, and logout. In the profile section, faculty members can access and update their personal information. The courses section displays a list of all the courses the faculty teaches, along with sub-sections for each student enrolled in the course and sections for the course itself. This provides an easy-to-use interface for faculty to manage and access the academic information of their students. Lastly, the logout option allows the user to safely exit the application. Overall, the faculty login page is designed to provide a streamlined experience for faculty members to manage their courses and students.

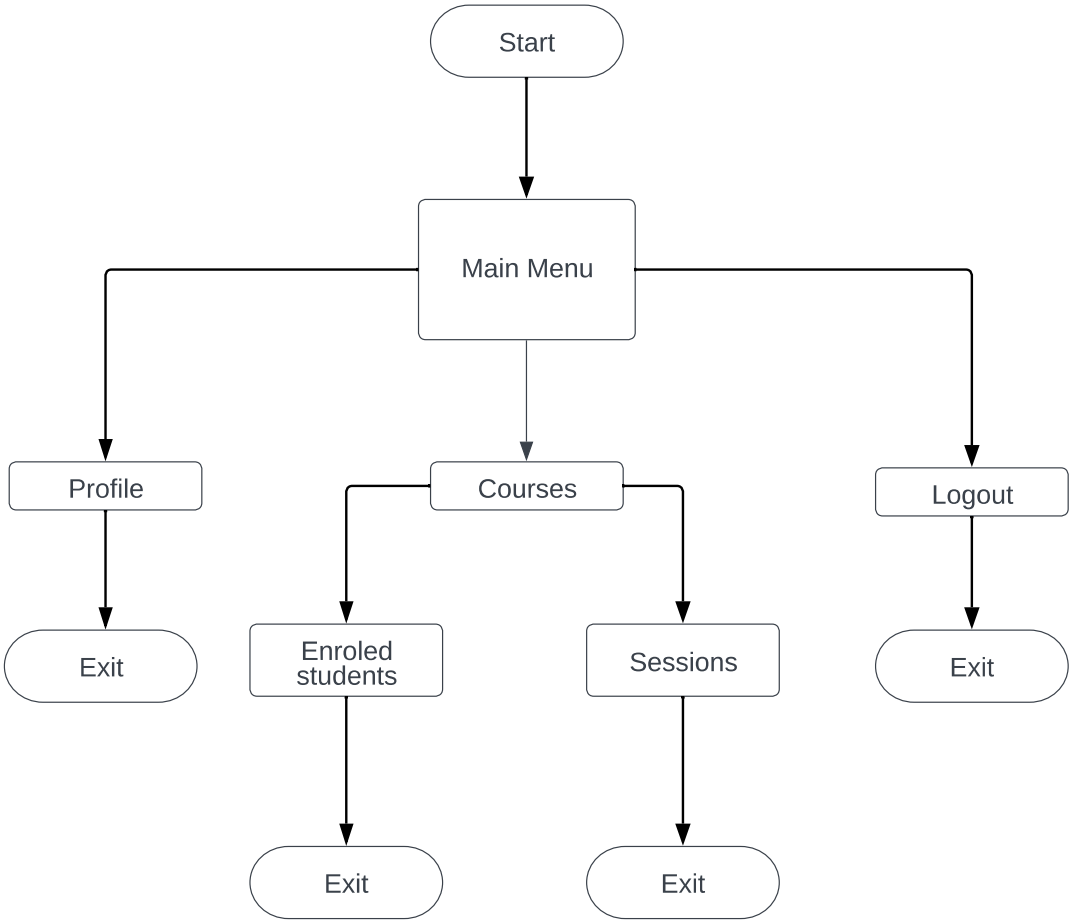


Figure 26 Faculty Login Page Flow Chart

Departments:

The Department page provides a detailed overview of the various departments. This page typically includes information such as the department's mission, faculty members, programs offered, and relevant news and events. It serves as a valuable resource for students, faculty, and staff who are interested in learning more about the academic departments and their programs. The Department page may also provide links to other related pages, such as individual faculty pages, program descriptions.

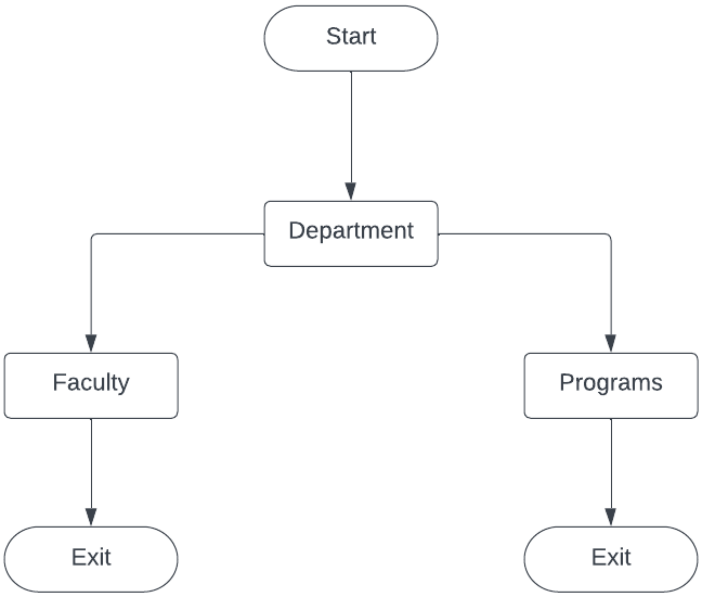


Figure 27 Departments Page Flow Chart

Admissions:

The admission page prompts the user to enter their admission number. If the admission number entered is valid, the page provides the user with their enrollment details. If an invalid admission number is entered, the page will prompt the user to re-enter a valid admission number. This process ensures that only authorized users can access their enrollment information and helps to maintain the security of the admission system.

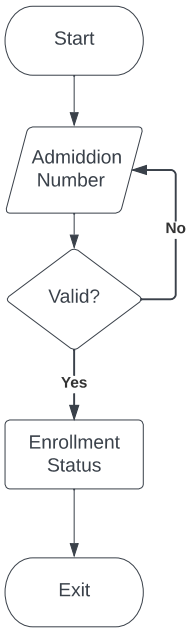


Figure 28 Admissions Page Flow Chart

Faculty:

The faculty page with profile, courses, and office hours provides essential information about the faculty members, including their background, courses taught, and availability to meet with students. This page serves as a valuable resource for students to connect with and learn from the institution's faculty.

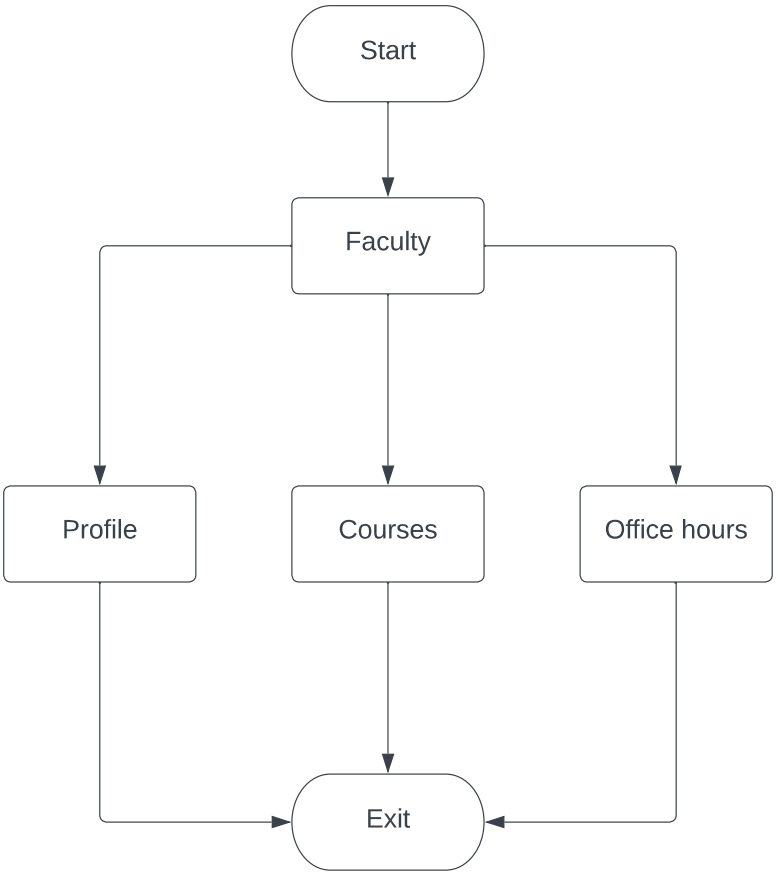


Figure 29 Faculty Page Flow Chart

Programs:

The programs page with details and courses provides important information about the academic programs offered, including curriculum, and course listings. This page serves as a valuable resource for students to explore and learn about the different programs available at this institution.

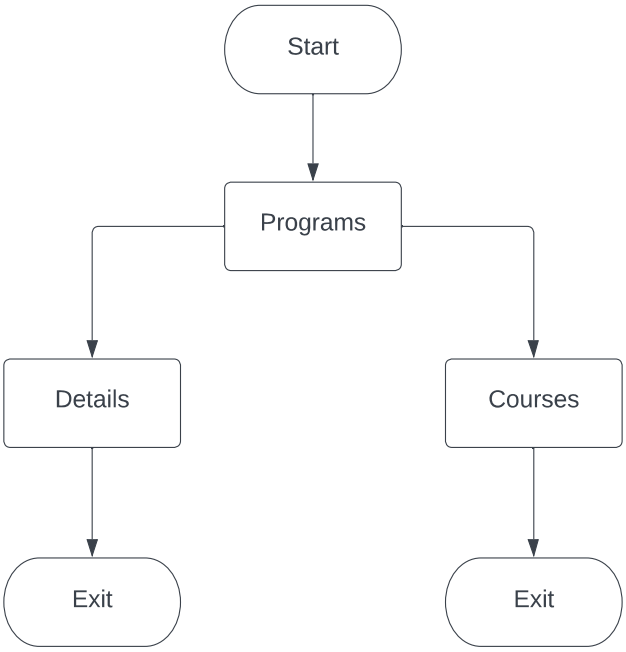


Figure 30 Programs Page Flow Chart

Courses:

The course page with course details is a section of an educational institution's website that provides detailed information about a specific course. This page typically includes an overview of the course, including the course title, description, prerequisites, and credit hours. The course page with course details serves as a valuable resource for students to learn about a particular course's content and expectations.

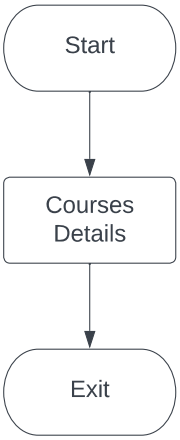


Figure 31 Courses Page Flow Chart

Views' implementation:

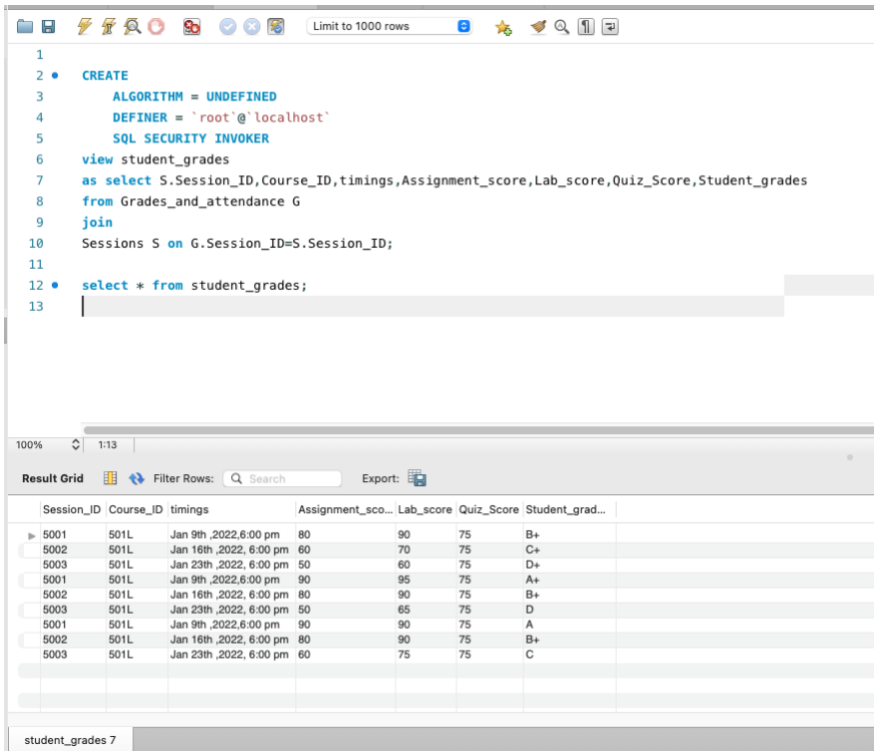


Figure 32 View Implementation for Student_grtades

The view student_grades is used for extracting student grades for student login page.

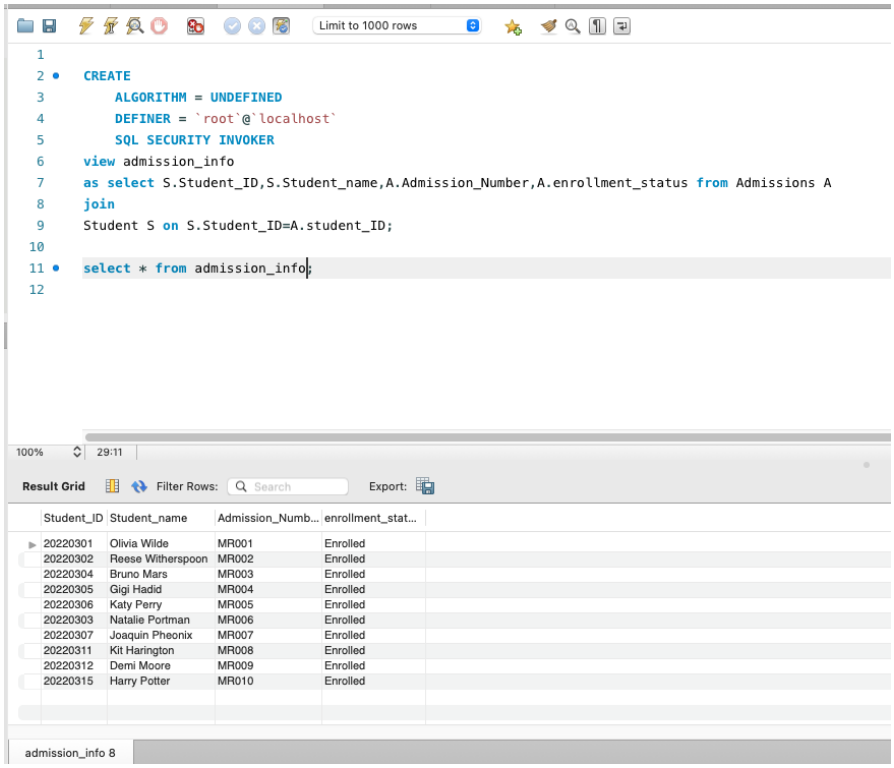


Figure 33 View Implementation for admission_info

The view admission_info is used for extracting enrollment details for admission page.

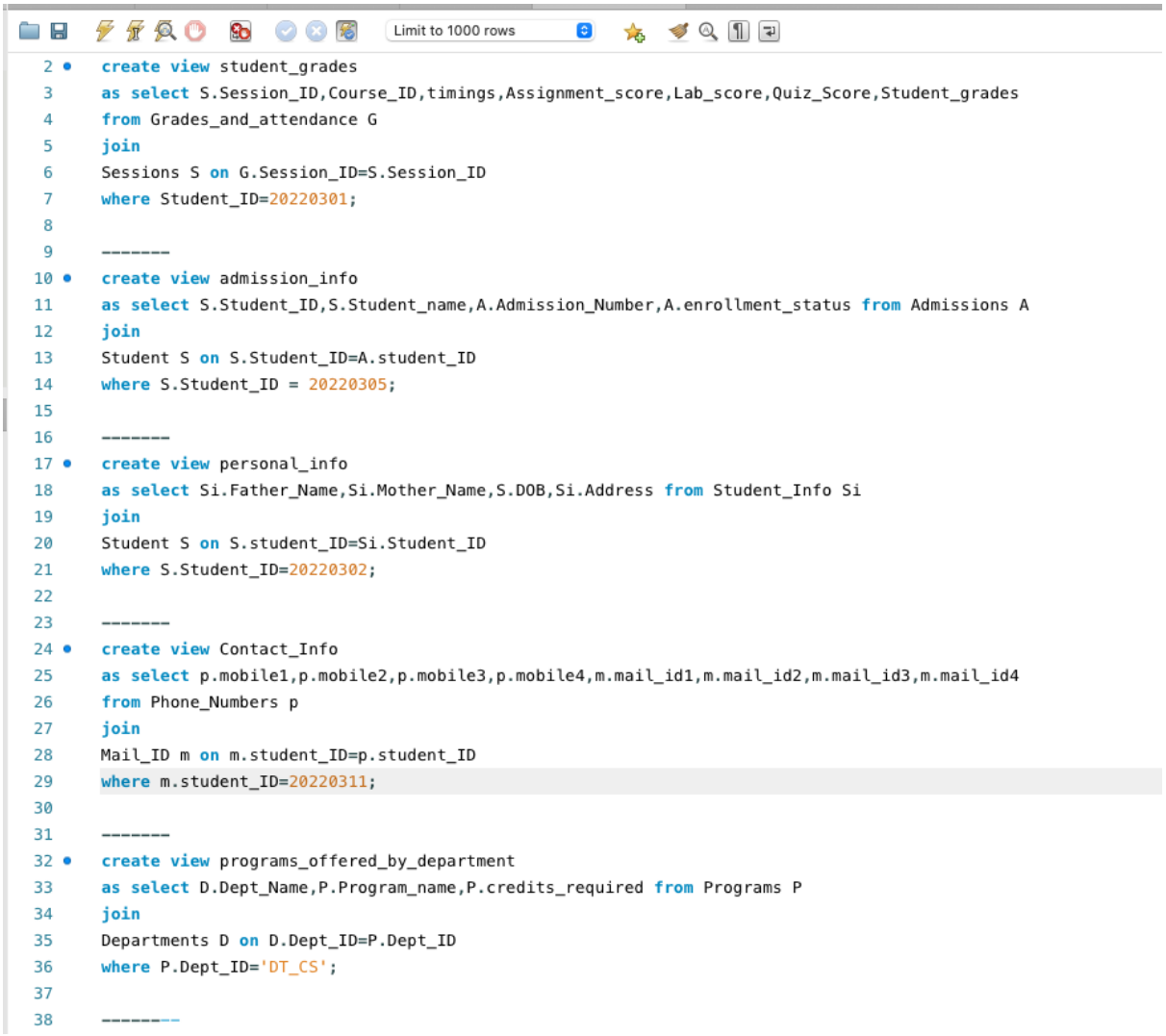


Figure 34 View Implementation for Personal_info, Contact_info and Programs_offered_by_department

The views presented above are examples that can be found in the "Phase05.2.sql" file which is included with the document.

Graphical user interface design:

Connection to database

The process for connecting to the database is performed using the **mysql.connector** module, and the code executes the **AdminLogin()** function when it is called at the end of the script.

The **AdminLogin()** function creates a new window using the **tk.Tk()** method from the **tkinter** library. It sets the window title and geometry using the **title()** and **geometry()** methods respectively. Then, it defines a nested function **login(win)** that attempts to connect to a MySQL database using the **mysql.connector.connect()** method. The **host**, **user**, **passwd**, and **database** parameters are passed to this method. If the connection is successful, the **OpenMain()** function is called, passing the window object and the database connection object as arguments, and a success message is displayed using the **messagebox.showinfo()** method. If the connection attempt fails, an error message is displayed using the **messagebox.showerror()** method.

```
db = mysql.connector.connect(
    host="localhost",
    user= username_entry.get(),
    passwd = password_entry.get(),
    database="super_six"
    )
```

Figure 35 Connection to database

Login page

This Python code defines a GUI-based user login system that enables the user to log into a MySQL database with a valid username and password. The process for connecting to the database is performed using the **mysql.connector** module, and the code executes the **AdminLogin()** function when it is called at the end of the script.

The **AdminLogin()** function creates a new window using the **tk.Tk()** method from the **tkinter** library. It sets the window title and geometry using the **title()** and **geometry()** methods respectively. Then, it defines a nested function **login(win)** that attempts to connect to a MySQL database using the **mysql.connector.connect()** method. The **host**, **user**, **passwd**, and **database** parameters are passed to this method. If the connection is successful, the **OpenMain()** function is called, passing the window object and the database connection object as arguments, and a success message is displayed using the **messagebox.showinfo()** method. If the connection attempt fails, an error message is displayed using the **messagebox.showerror()** method.

The function then creates a new frame object using the **tk.Frame()** method, which is used to hold the login form elements. The form elements include **tk.Label**, **tk.Entry**, and **tk.Button** widgets, which are arranged in a grid layout using the **grid()** method. The **username_entry**

and **password_entry** variables store the user's input for the username and password, respectively.

Finally, the **AdminLogin()** function packs the frame and enters the event loop using the **mainloop()** method, which waits for user input and responds to events. When the user clicks the 'Login' button, the **login()** function is called with the window object passed as an argument. This attempts to connect to the database and executes the appropriate depending on the outcome.

```
import tkinter as tk
from tkinter import messagebox
import mysql.connector
from MainPage import OpenMain


def AdminLogin():
    win = tk.Tk()
    win.title("User")
    win.geometry("400x400")
    def login(win):
        try:
            db = mysql.connector.connect(
                host="localhost",
                user= username_entry.get(),
                passwd = password_entry.get(),
                database="super_six"
                )

        except:
            messagebox.showerror(title="Error", message="Incorrect Username or Password")
        else:
            list = win.pack_slaves()
            for i in list:
                i.destroy()
            OpenMain(win,db)
            messagebox.showinfo(title="Successfully loggedIn", message="You have successfully logged in.")

    frame = tk.Frame()
    tk.Label(frame, text="Login").grid(row=0, column=0, columnspan=2, sticky="news",pady=30)

    tk.Label(frame, text="Username").grid(row=1, column=0)
    username_entry = tk.Entry(frame)
    username_entry.grid(row=1, column=1, pady=20)

    tk.Label(frame, text="Password").grid(row=2, column=0)
    password_entry = tk.Entry(frame)
    password_entry.grid(row=2, column=1, pady=20)

    tk.Button(frame, text="Login", command=lambda: login(win)).grid(row=3, column=0, columnspan=2, pady=30)

    frame.pack()
    win.mainloop()



AdminLogin()
```

Figure 36 Code snippet for Login page

Figure 37 Login page

Main Menu Page

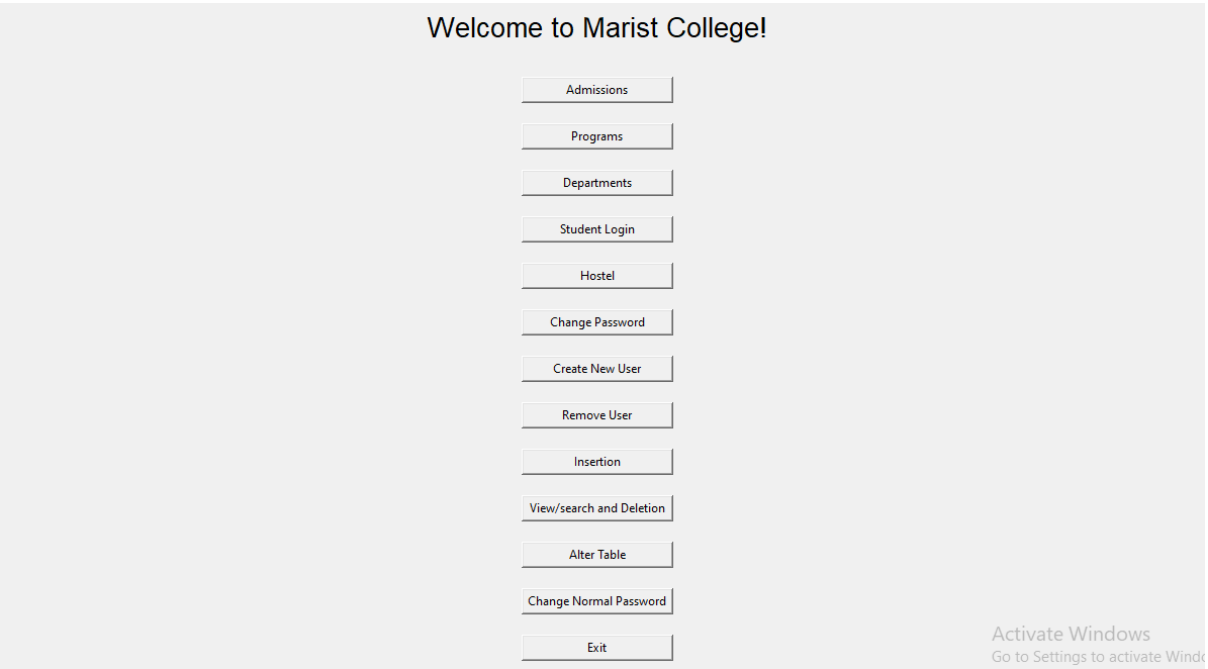


Figure 38 Admin User Main Menu

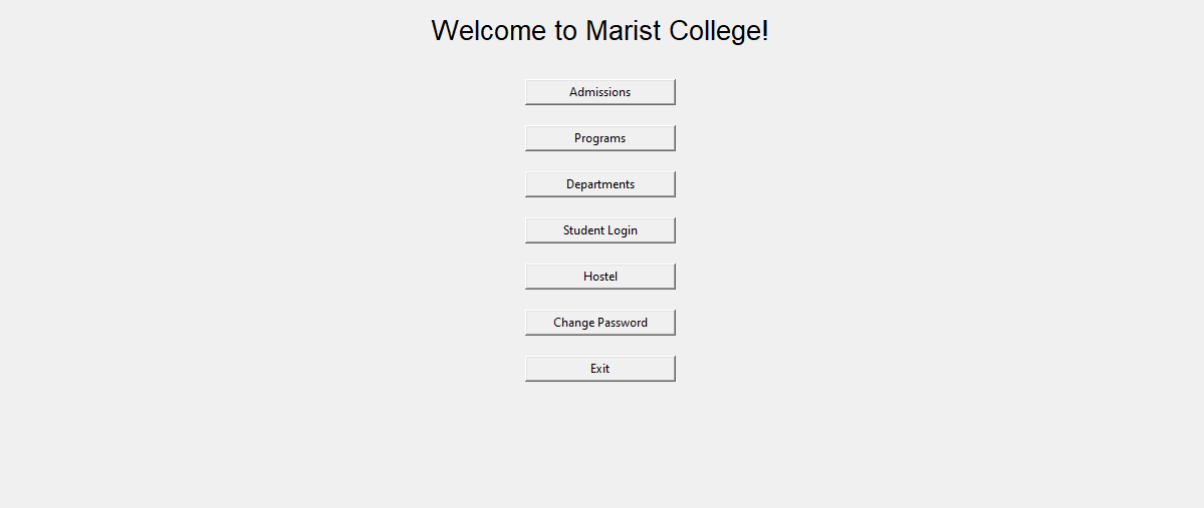


Figure 39 User Main Menu

```
def MainPage():

    welcome_label = tk.Label(win, text="Welcome to Marist College!", font=("Arial", 20))
    welcome_label.pack(pady=20)

    admissions_button = tk.Button(win, text="Admissions", command=AdmissionsPage, width=20)
    admissions_button.pack(pady=10)

    programs_button = tk.Button(win, text="Programs", command=lambda:sampletDeletionPage("Programs","Programs","main"), width=20)
    programs_button.pack(pady=10)

    departments_button = tk.Button(win, text="Departments", command=lambda:sampletDeletionPage("Department","Departments","main"), width=20)
    departments_button.pack(pady=10)

    login_button = tk.Button(win, text="Student Login", command=LoginPage, width=20)
    login_button.pack(pady=10)

    hostel_button = tk.Button(win, text="Hostel", command=show_hostel, width=20)
    hostel_button.pack(pady=10)

    Change_Password_button = tk.Button(win, text="Change Password", command=ChangeUserPassword, width=20)
    Change_Password_button.pack(pady=10)

    if(admin == True):
        User_button = tk.Button(win, text="Create New User", command=createuser, width=20)
        User_button.pack(pady=10)
        Remove_button = tk.Button(win, text="Remove User", command=RemoveUser, width=20)
        Remove_button.pack(pady=10)
        Insert_button = tk.Button(win, text="Insertion", command=InsertionPage, width=20)
        Insert_button.pack(pady=10)
        view_button = tk.Button(win, text="View/search and Deletion", command=DeletionPage, width=20)
        view_button.pack(pady=10)
        Alter_table = tk.Button(win, text="Alter Table", command=AlterTable, width=20)
        Alter_table.pack(pady=10)
        Change_normal_Password_button = tk.Button(win, text="Change Normal Password", command=ChangeNormalUserPassword, width=20)
        Change_normal_Password_button.pack(pady=10)

    Exit_button = tk.Button(win, text="Exit", command=lambda: exit("window"), width=20)
    Exit_button.pack(pady=10)
```

Figure 40 Code Snippet for Main Menu Page

The given code defines a function called MainPage() which is responsible for creating the main page of a college management system GUI application.

The function creates several GUI components using the tkinter library such as labels, buttons, and input fields. The components include a welcome message, buttons for admissions, programs, departments, student login, hostel, and change password.

The function also checks whether the user is an admin or not, and if so, it adds additional buttons for creating a new user, removing a user, inserting data, viewing and deleting data, altering a table, and changing a normal password.

Finally, an exit button is added to close the application. Overall, the function creates a basic GUI for a college management system with various features and options depending on the user's access level.

Action Pages

Search Page:

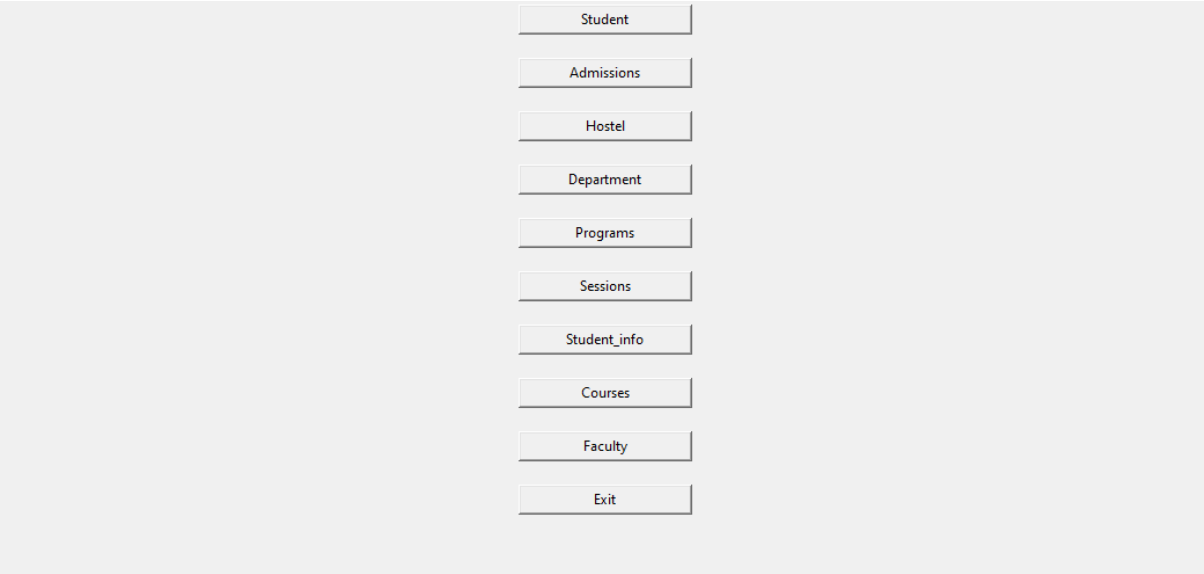


Figure 41 View/Search Page

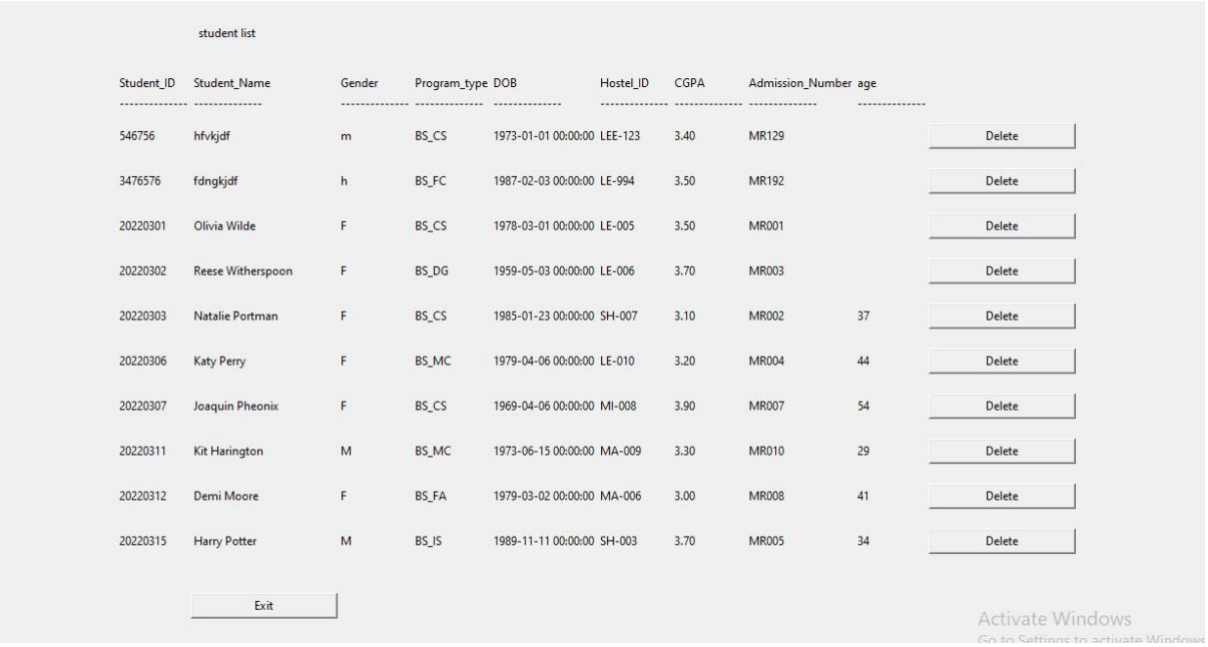


Figure 42 Student View Page

```
def ViewPage(title,table,*type):
    clearwindow()
    cur = database.cursor()
    win.title(title+" Page")
    cur.execute("SHOW COLUMNS from "+table)
    index=0
    tuple1 = []
    for i in cur.
        tuple1.ap
    def deletefn(id):
        try:
            cur.execute("set foreign_key_checks=0")
            cur.execute("delete from "+table+" where Student_ID = "+str(id))
            print("delete from "+table+" where Student_ID = "+str(id))
            database.commit()
        except Exception as e:
            print(e)
            exit()
        else:
            messagebox.showinfo(title="Successful", message="deleted Successfully")
            exit()
    frame = tk.Frame()
    cur.execute("select * from "+table)
    studentlist = cur.fetchall()
    tk.Label(frame, text=table+" list").grid(row=0, column=0, columnspan=2, sticky="news",pady=30)
    index1 = 0
    for i in tuple1:
        tk.Label(frame, text=i).grid(row=1,column=index1,sticky="w")
        tk.Label(frame, text="--------------").grid(row=2,column=index1,sticky="w")
        index1+=1
    r=3
    for i in studentlist:
        indexj = 0
        for j in i:
            tk.Label(frame, text = i[indexj]).grid(row=r, column=indexj, sticky="w")
            indexj+=1
        if(len(type) == 0):
            tk.Button(frame,text= "Delete",command=lambda: deletefn(i[0]),width=20).grid(row=r,column=indexj,pady=10)
        frame.pack()
        r=r+1
    tk.Button(frame,text= "Exit",command= exit,width=20).grid(row=r,column=1,pady=30)
    frame.pack()
```

(parameter) id: Any

Figure 43 Code Snippet for View Page

This code defines a function called ViewPage that takes in three parameters: title, table, and an optional parameter type.

The function clears the window, sets the window title to title, retrieves the column names of the specified table from the database, and stores them in a list called tuple1.

Next, the function queries the database for all rows in the specified table and stores the result in studentlist. It then creates a frame and displays the column names and a list of all students in the specified table.

If type is not provided, it also creates a "Delete" button for each row of data that calls the deletefn function when clicked. deletefn deletes the specified row from the table using the primary key Student_ID.

Finally, the function creates an "Exit" button that closes the window when clicked.

References:

[1] *https://www.edrawsoft.com/article/what-is-eer-diagram.html*

GitHub repository address:

https://github.com/PradeepReddy-Baireddy/MSCS_542L_256_23S_College-Data-Management-System_Super-Six