|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Faculty of Engineering & Technology** | | | | |
|  | **Ramaiah University of Applied Sciences** | | | | |
| **Department** | Computer Engineering | Science | and | **Programme** | B. Tech. CSE/AIML/ISE |
| **Semester/Batch** | 5th/2021 |  |  |  | |
| **Course Code** | 20CSC302A |  |  | **Course Title** | Database Systems |
| **Course Leader(s)** | Dr. Narendra Babu, Mrs. Sahana P Shankar, Mrs. Supriya M S | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Assignment** | | | | |  |  |
| Register No. | |  | 21ETCS002137 | Name of Student | | VRUSHALI SANTOSH KULKARNI |  |  |
| **Sections** |  | **Marking Scheme** | | | **Max**    **Marks** | | **First**  **Examiner**    **Marks** | **Second Examiner**    **Marks** |
| **Part A** | **A.1** | Discuss any two database models used in the modern day enterprise computing applications with suitable examples. | | | 05 | |  |  |
|  | **Part-A Max Marks** | | | **05** | |  |  |
| **P**  **art B** | **B2.1** | List of functional requirements | | | 02 | |  |  |
| **B2.2** | Implementation of relational database schema with appropriate attributes, and constraints using SQL commands | | | 10 | |  |  |
| **B2.3** | Design and implementation of GUI | | | 05 | |  |  |
| **B2.4** | Connection of front end with the database and discussion on the results | | | 03 | |  |  |
|  | **Part-B Max Marks** | | | **20** | |  |  |
|  |  | **Total Assignment Marks** | | | **25** | |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Course Marks Tabulation** | |  |
| **Component-**  **1(B)Assignment** | **First**  **Examiner** | **Remarks** | **Second Examiner** | **Remarks** |
| **A** |  |  |  |  |
| **B** |  |  |  |  |
| **Marks (out of 25)** |  |  |  |  |
| Signature of First Examiner |  |  | | Signature of Second Examiner |

**Please note:**

1. Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
2. The First Examiner is required to mark the comments in RED ink and the Second Examiner’s comments should be in GREEN ink.
3. The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
4. If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

**Assignment Instructions to students:**

1. The assignment consists of **2** questions: Part A –**1** Question, Part B- **1** Questions.
2. Maximum marks is **25**.
3. The assignment has to be neatly word processed as per the prescribed format.
4. **Submission Date:** 12/03/2024
5. **Submission after the due date is not permitted.**
6. **IMPORTANT**: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.
7. Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question

**PART A 05 Marks**

Enterprise computing applications are software applications designed to meet the complex and extensive requirements of large organizations or enterprises. These applications are critical for supporting various business processes, enhancing efficiency, and facilitating collaboration across different departments. Some the examples include Enterprise Resource Planning, Customer Relationship Management, Business Intelligence, Project Management Systems among others. You are required to generate a short report (no exceeding 300 Words) on the context which should address the following:

**A.1** Discuss any two database models used in the modern day enterprise computing applications with suitable examples.

1.**Relational databases**: These databases store data in organized tables with rows and columns. They use keys to link tables together, ensuring data consistency. It is mainly used for structured data. Imagine a customer relationship management (CRM) system - data on customers, orders, products, and payments can all be stored in relational tables with defined connections using keys

Ex: MySQL, Microsoft SQL Server, Oracle Database

MySQL is used by applications such as Uber and AirBNB to store customer data,etc.

Microsoft SQL Server is used by companies like Accenture.

2**. Document-oriented databases**: These store data in flexible documents, often using JSON format. This allows for variations in data structure within the same database. It follows NoSQL model for unstructured data. Content management systems (CMS) are a good example - articles with titles, authors, content, and tags can be stored as separate documents. This flexibility makes it easier to manage various types of information.

Ex: MongoDB, Couchbase DB

MongoDB is used by Forbes and Adobe for their database management needs.

**PART B 20 Marks**

Consider the **RUAS Student Management System** to manage the details of students in RUAS. The computerized system enables the users to access students’ data at any time and from any place. The system consists of the functionalities such as Student Details, Branch Details, Fee Payment, Exam Results and any other student related information needed by the university. It is required to undertake the following activities:

**B2.1** List of functional requirements

Functional requirements:

1. Student Management System:

* Enroll new students and maintain their information.
* Update existing student records.
* Find and view detailed information about students.
* Remove student records when necessary (e.g., graduation).

2. Branch Details:

* Add and update information about university departments.
* Search for and view details about specific branches.
* Delete branch information if needed (e.g., branch closure).

3. Fee Details:

* Record student fee payments.
* Check payment history

4. Exam Results:

* Record and manage exam results for students.
* Provide students with access to their own exam scores.

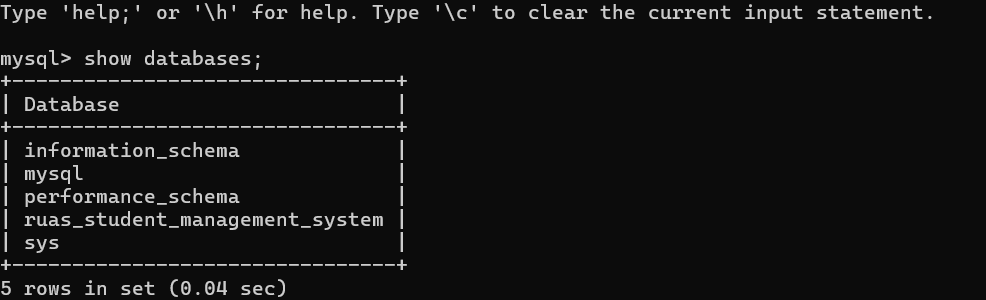
5. Data Security

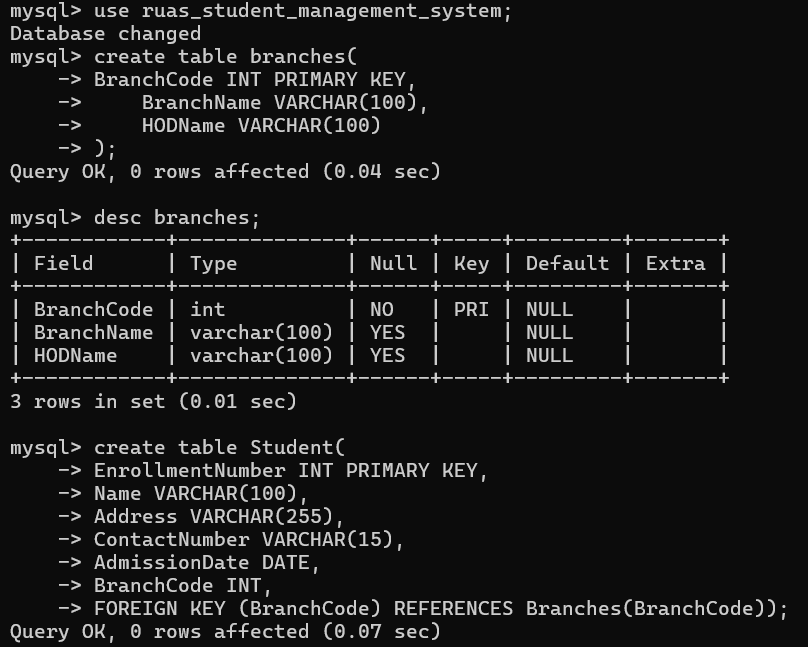
* Regularly back up the entire database to prevent data loss.
* Restore data in case of system failures or accidental deletion.

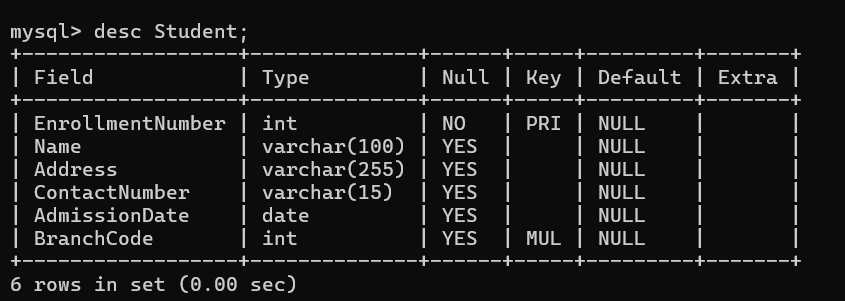
6. System Integration

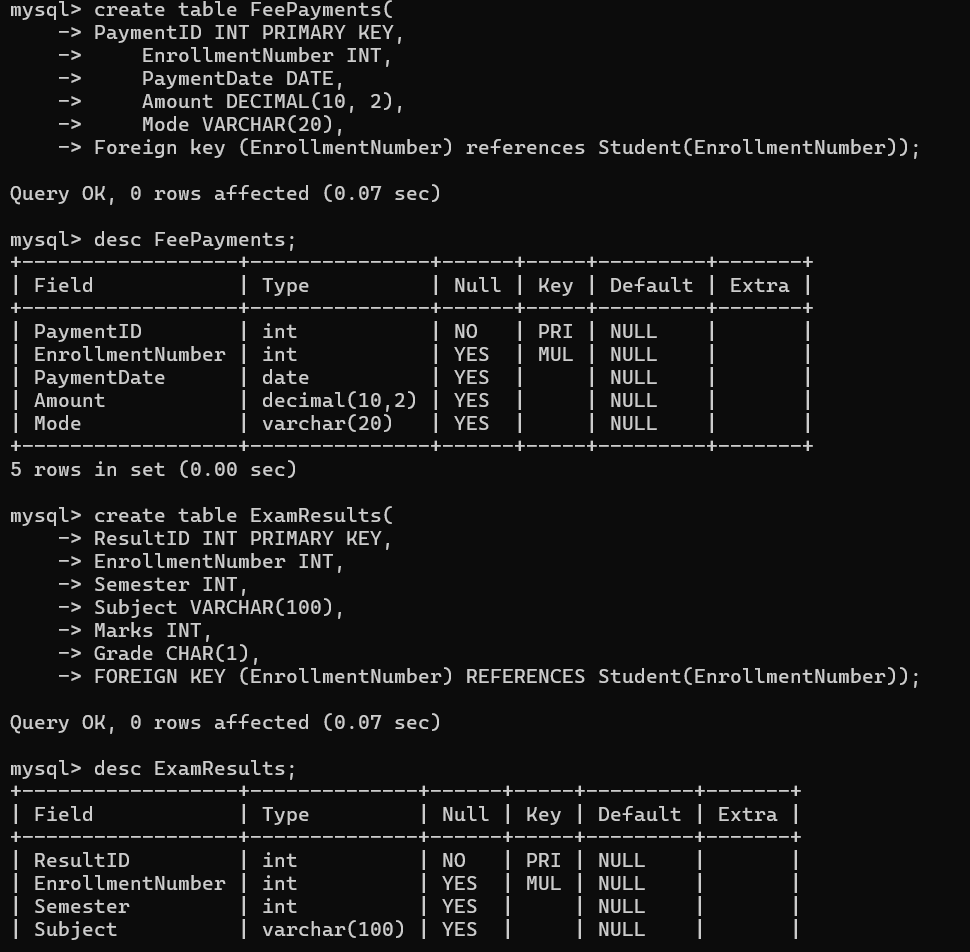
* Connect with other university systems like Library Records and Attendance of Students for seamless data exchange.

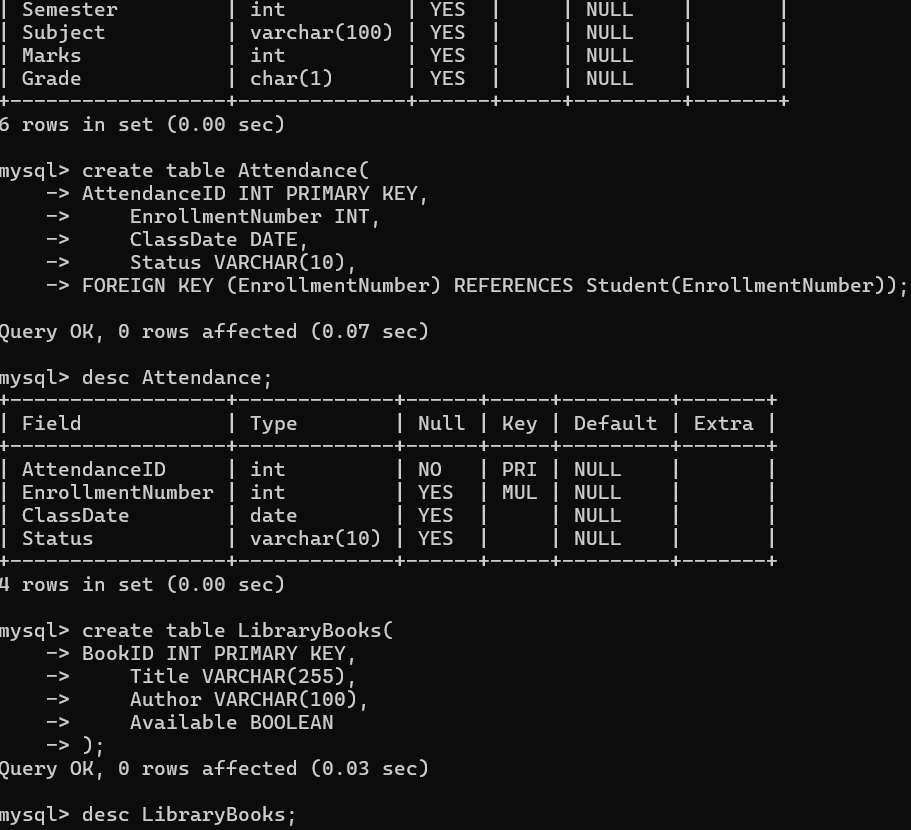
**B2.2** Implementation of relational database schema with appropriate attributes, and constraints using SQL commands

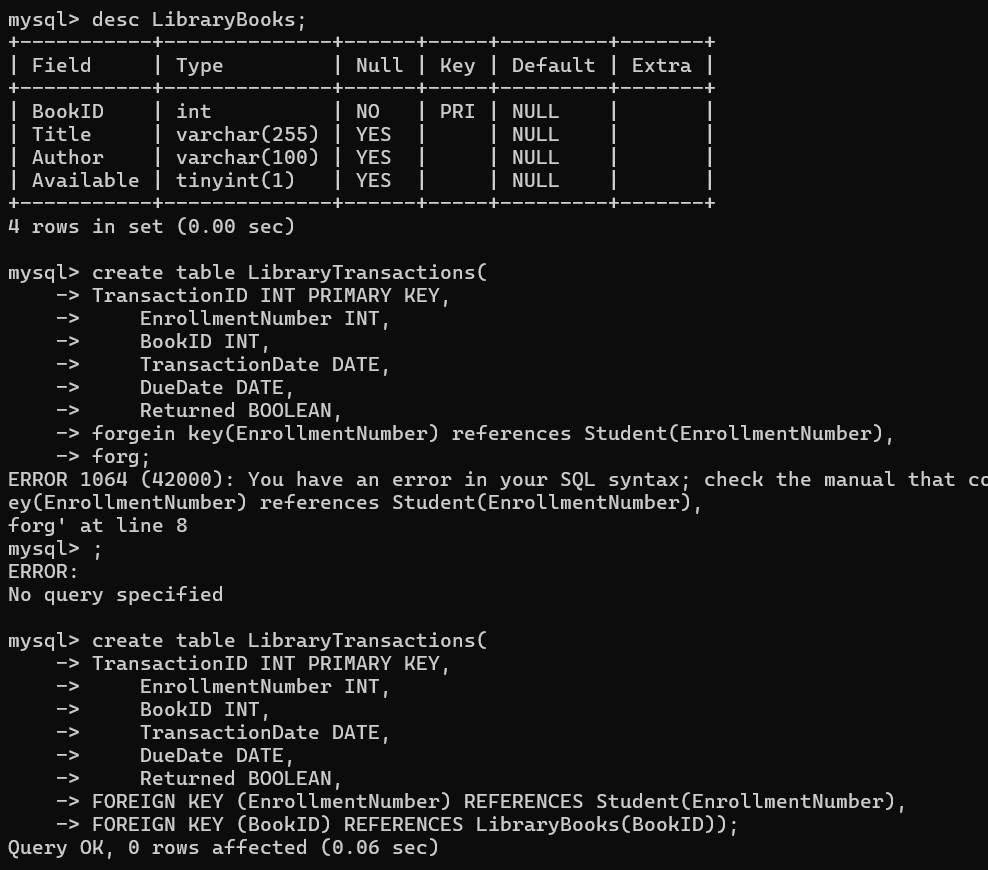


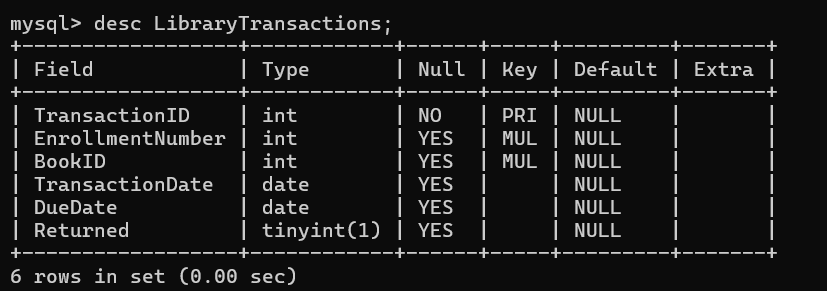




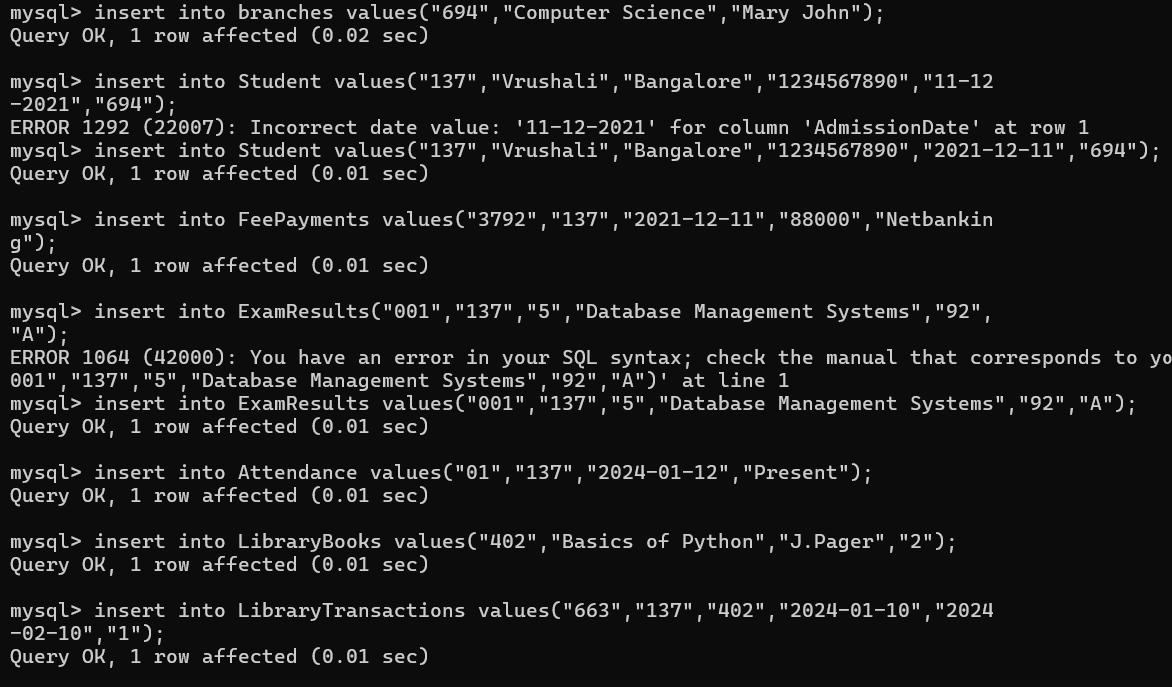








Populating the table with values:



**B2.3** Design and implementation of GUI

**B2.4** Connection of front end with the database and discussion on the results

import mysql.connector

class StudentManagementSystem:

    def \_\_init\_\_(self, host='localhost', user='root', password='Abcxyz@01', database='ruas\_student\_management\_system'):

        self.conn = mysql.connector.connect(

            host=host,

            user=user,

            password=password,

            database=database

        )

        self.cursor = self.conn.cursor()

        # Create tables if they do not exist

        self.create\_tables()

    def create\_tables(self):

        # Define table creation queries

        create\_student\_table = '''

            CREATE TABLE IF NOT EXISTS Student (

                EnrollmentNumber INT PRIMARY KEY,

                Name VARCHAR(255),

                Address VARCHAR(255),

                ContactNumber VARCHAR(20),

                AdmissionDate DATE,

                BranchCode INT

            )

        '''

        create\_branches\_table = '''

            CREATE TABLE IF NOT EXISTS Branches (

                BranchCode INT PRIMARY KEY,

                BranchName VARCHAR(100),

                HODName VARCHAR(100)

            )

        '''

        create\_fee\_payments\_table = '''

            CREATE TABLE IF NOT EXISTS FeePayments (

                PaymentID INT AUTO\_INCREMENT PRIMARY KEY,

                EnrollmentNumber INT,

                PaymentDate DATE,

                Amount DECIMAL(10, 2),

                Mode VARCHAR(20),

                FOREIGN KEY (EnrollmentNumber) REFERENCES Student(EnrollmentNumber)

            )

        '''

        create\_exam\_results\_table = '''

            CREATE TABLE IF NOT EXISTS ExamResults (

                ResultID INT AUTO\_INCREMENT PRIMARY KEY,

                EnrollmentNumber INT,

                Semester INT,

                Subject VARCHAR(100),

                Marks INT,

                Grade CHAR(1),

                FOREIGN KEY (EnrollmentNumber) REFERENCES Student(EnrollmentNumber)

            )

        '''

        create\_attendance\_table = '''

            CREATE TABLE IF NOT EXISTS Attendance (

                AttendanceID INT AUTO\_INCREMENT PRIMARY KEY,

                EnrollmentNumber INT,

                ClassDate DATE,

                Status VARCHAR(10),

                FOREIGN KEY (EnrollmentNumber) REFERENCES Student(EnrollmentNumber)

            )

        '''

        create\_library\_books\_table = '''

            CREATE TABLE IF NOT EXISTS LibraryBooks (

                BookID INT AUTO\_INCREMENT PRIMARY KEY,

                Title VARCHAR(255),

                Author VARCHAR(100),

                Available BOOLEAN

            )

        '''

        create\_library\_transactions\_table = '''

            CREATE TABLE IF NOT EXISTS LibraryTransactions (

                TransactionID INT AUTO\_INCREMENT PRIMARY KEY,

                EnrollmentNumber INT,

                BookID INT,

                TransactionDate DATE,

                DueDate DATE,

                Returned BOOLEAN,

                FOREIGN KEY (EnrollmentNumber) REFERENCES Student(EnrollmentNumber),

                FOREIGN KEY (BookID) REFERENCES LibraryBooks(BookID)

            )

        '''

        # Execute table creation queries

        self.cursor.execute(create\_student\_table)

        self.cursor.execute(create\_branches\_table)

        self.cursor.execute(create\_fee\_payments\_table)

        self.cursor.execute(create\_exam\_results\_table)

        self.cursor.execute(create\_attendance\_table)

        self.cursor.execute(create\_library\_books\_table)

        self.cursor.execute(create\_library\_transactions\_table)

        # Commit changes

        self.conn.commit()

    def add\_student(self, enrollment\_number, name, address, contact\_number, admission\_date, branch\_code):

        try:

            self.cursor.execute('''

                INSERT INTO Student

                (EnrollmentNumber, Name, Address, ContactNumber, AdmissionDate, BranchCode)

                VALUES (%s, %s, %s, %s, %s, %s)

            ''', (enrollment\_number, name, address, contact\_number, admission\_date, branch\_code))

            self.conn.commit()

            print("Student added successfully!")

        except Exception as e:

            print(f"Error adding student: {e}")

    def delete\_student(self, enrollment\_number):

        try:

            self.cursor.execute('''

                DELETE FROM Student

                WHERE EnrollmentNumber = %s

            ''', (enrollment\_number,))

            self.conn.commit()

            print("Student deleted successfully!")

        except Exception as e:

            print(f"Error deleting student: {e}")

    def view\_students(self):

        self.cursor.execute('SELECT \* FROM Student')

        students = self.cursor.fetchall()

        if students:

            for student in students:

                print(student)

        else:

            print("No students found.")

    def add\_branch(self, branch\_code, branch\_name, hod\_name):

        try:

            self.cursor.execute('''

                INSERT INTO branches

                (BranchCode, BranchName, HODName)

                VALUES (%s, %s, %s)

            ''', (branch\_code, branch\_name, hod\_name))

            self.conn.commit()

            print("Branch added successfully!")

        except Exception as e:

            print(f"Error adding branch: {e}")

    def delete\_branch(self, branch\_code):

        try:

            self.cursor.execute('''

                DELETE FROM branches and Student

                WHERE BranchCode = %s

            ''', (branch\_code,))

            self.conn.commit()

            print("Branch deleted successfully!")

        except Exception as e:

            print(f"Error deleting branch: {e}")

    def view\_branches(self):

        self.cursor.execute('SELECT \* FROM branches')

        branches = self.cursor.fetchall()

        if branches:

            for branch in branches:

                print(branch)

        else:

            print("No branches found.")

if \_\_name\_\_ == "\_\_main\_\_":

    sms = StudentManagementSystem()

    while True:

        print("\nStudent Management System")

        print("1. Add Student")

        print("2. View Students")

        print("3. Delete Student")

        print("4. Add Branch")

        print("5. View Branches")

        print("6.Delete Branch")

        print("7.Exit")

        choice = input("Enter your choice (1/2/3/4/5/6/7): ")

        if choice == '1':

            enrollment\_number = int(input("Enter Enrollment Number: "))

            name = input("Enter Name: ")

            address = input("Enter Address: ")

            contact\_number = input("Enter Contact Number: ")

            admission\_date = input("Enter Admission Date (YYYY-MM-DD): ")

            branch\_code = input("Enter Branch Code: ")

            sms.add\_student(enrollment\_number, name, address, contact\_number, admission\_date, branch\_code)

        elif choice == '2':

            sms.view\_students()

        elif choice == '3':

            enrollment\_number = input("Enter Enrollment Number to delete: ")

            sms.delete\_student(enrollment\_number)

        elif choice == '4':

            branch\_code = input("Enter Branch Code: ")

            branch\_name = input("Enter Branch Name: ")

            hod\_name = input("Enter HOD Name: ")

            sms.add\_branch(branch\_code, branch\_name, hod\_name)

            break

        elif choice == '5':

            sms.view\_branches()

        elif choice == '6':

            branch\_code = input("Enter Branch Code to delete: ")

            sms.delete\_branch(branch\_code)

        elif choice == '7':

            print("Exiting Student Management System.")

            break

        else:

            print("Invalid choice. Please enter 1, 2,3,4,5,6 or 7")

The part of python program the connects SQL database to front end

import mysql.connector

class StudentManagementSystem:

    def \_\_init\_\_(self, host='localhost', user='root', password='Abcxyz@01', database='ruas\_student\_management\_system'):

        self.conn = mysql.connector.connect(

            host=host,

            user=user,

            password=password,

            database=database

        )

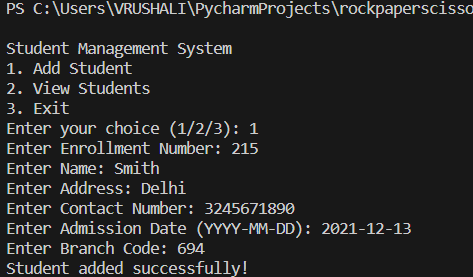
        self.cursor = self.conn.cursor()

We give the input for host, name of the user to connect to the server, the password to the MYSQL database, and the name of the MYSQL database.

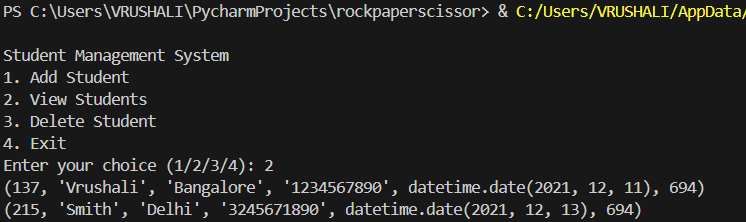
self.conn executes queries in sql linked to database.

OUTPUT:

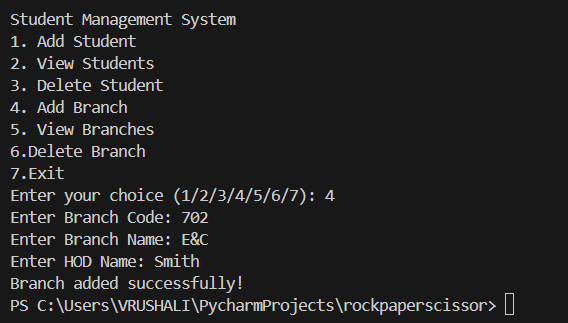
Add student using the python program:



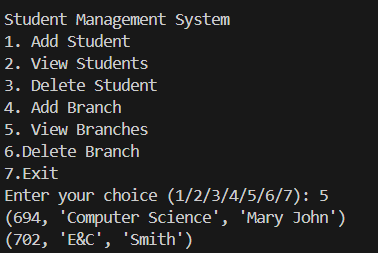
Viewing the students:



Adding a branch via python program:



Viewing branch



The Python program establishes connection with MySQL database. Changes made using methods in python will get updated in the database. MySQL is the official driver of Python by Oracle. It is a pure python implementation that communicates with MYSQL server by MYSQL server/client protocol.

**Note:** Make appropriate assumptions to make the specification complete.

🙚🙘