Table of Contents

CHAPTER 01: INTRODUCTION

1.1 Preface

The "Student Report Card System" project is designed to streamline the process of recording, updating, and generating reports of student marks using a combination of Python and MySQL. The system aims to provide a user-friendly interface for teachers to efficiently manage student data while ensuring accuracy and consistency in report generation.

In today's educational environment, managing student performance data is crucial. This project addresses the need for a simplified digital solution that eliminates the manual handling of records, reducing errors and saving time. By integrating Python and MySQL, we create a robust backend to handle data storage and retrieval, while Python's simplicity and versatility allow for easy interaction with the database.

This system includes essential features like adding new student records, updating existing marks, viewing individual performance, and generating comprehensive report cards. The choice of basic Python and SQL ensures that the project remains accessible and maintainable, without delving into complex programming paradigms or advanced database operations.

Throughout this project, the records are taken as sample of class XII students to demonstrate the functionalities. The development process emphasizes clear and efficient coding practices, adhering to the appropriate naming convention to maintain consistency and readability in code.

This preface outlines the purpose and scope of the "Student Report Card System," highlighting its relevance in modern educational contexts and its potential to improve administrative efficiency in schools.

1.2 Abstract

The "Student Report Card System" is a software application developed using Python and MySQL, designed to manage student records efficiently. This project addresses the administrative needs of educational institutions by providing a digital platform for teachers to add, update, view, and manage student marks and generate report cards.

The system comprises three primary components: the students table, which stores personal and academic information of students; the Marks table, which records subject-specific marks for each student; and the ReportCard table, which compiles the total marks and percentage for each student. Basic SQL queries facilitate the interaction between Python and the MySQL database, ensuring a smooth data flow.

Key features of this system include the ability to insert new student data, modify existing records, retrieve individual student performances, and generate cumulative report cards. The project adheres to fundamental Python principles, excluding advanced concepts like object-oriented programming, to maintain simplicity and clarity.

The use of the mysql.connector library ensures a seamless connection between Python and MySQL, allowing for real-time data operations. By following the camelCase naming convention, the code remains consistent and easy to understand.

Overall, the "Student Report Card System" provides a practical solution for educational institutions to manage student records effectively, highlighting the integration of basic programming concepts and database management.

1.3 Applications

The "Student Report Card System" has several practical applications within educational institutions, particularly in the areas of student data management and academic performance tracking. Below are the primary applications of this system:

A. **Student Records Management**: The system allows educational institutions to maintain an organized database of student information, including personal details, class, and section. This ensures that all data is easily accessible and up-to-date.

B. **Mark Entry and Updates**: Teachers can efficiently enter and update marks for individual students across various subjects. This eliminates the need for manual record-keeping and reduces the likelihood of errors in data entry.

C. **Performance Tracking**: The system provides a platform for tracking student performance over time. Teachers and administrators can view detailed records of students' marks, helping to identify trends and areas needing improvement.

D. **Report Card Generation**: One of the most critical applications of this system is generating report cards. The system calculates the total marks and percentage for each student and compiles this information into a report card, which can be easily printed or distributed electronically.

E. **Data Consistency and Integrity**: By using a centralized database, the system ensures data consistency and integrity. All information is stored securely, and updates are reflected in real-time across the system.

F. **Resource Efficiency**: The automated nature of the system reduces the time and resources required for manual data handling, allowing teachers to focus more on teaching and less on administrative tasks.

G. **Customization and Scalability**: The system can be customized to fit the specific needs of different educational institutions, and it can be scaled to handle larger volumes of data as the institution grows.

Overall, the "Student Report Card System" enhances the efficiency and accuracy of managing student academic records, making it an invaluable tool for schools and educational administrators.

1.4 Reliability

The "Student Report Card System" is designed to be a reliable solution for managing student records and academic performance data. Several key factors contribute to the reliability of this system:

A. **Data Accuracy**: The system minimizes human error by automating the process of entering, updating, and retrieving student data. SQL queries ensure accurate data manipulation, and Python's error-handling mechanisms further enhance reliability.

B. **Consistent Performance**: The use of Python and MySQL ensures consistent performance even as the volume of data grows. The system can handle multiple operations simultaneously without compromising speed or accuracy.

C. **Data Integrity**: Data integrity is maintained through the use of primary keys and relational database structures. Each student's records are uniquely identified, preventing duplication and ensuring that data remains consistent across all tables.

D. **Error Handling**: The system includes basic error-handling techniques to manage potential issues such as incorrect data entry, missing information, or database connection failures. These mechanisms ensure the system can handle errors gracefully without crashing.

E. **Real-Time Updates**: Changes made to student records are reflected in real-time across the database, ensuring that all users access the most current data. This real-time synchronization enhances the reliability of the information provided by the system.

F. **Database Backups**: Regular database backups can be scheduled to prevent data loss in case of unexpected failures. This feature ensures that the system can be quickly restored to its last known good state, preserving data reliability.

G. **Scalability**: The system is designed to handle an increasing number of records as the educational institution grows. Its scalable architecture ensures that performance and reliability are maintained even with larger datasets.

By integrating these features, the "Student Report Card System" provides a dependable platform for educators and administrators to manage student academic records effectively.

1.5 Advantages and Limitations

1.5.1 Advantages:

A. **Efficient Data Management**: The system centralizes student data, making it easy to store, retrieve, and manage information. This reduces manual paperwork and minimizes the risk of data loss.

B. **User-Friendly Interface**: Designed with simplicity in mind, the system allows teachers to add, update, and view student marks effortlessly, enhancing usability.

C. **Real-Time Data Updates**: Any changes made in the database are instantly updated and accessible, ensuring that all information is current and consistent across the platform.

D. **Automated Report Generation**: The system automates the calculation of total marks and percentages, quickly generating accurate report cards, which saves time and reduces errors in manual calculations.

E. **Data Integrity and Accuracy**: By using structured databases and primary keys, the system ensures data accuracy and prevents duplication, maintaining data integrity.

F. **Scalability**: The system can easily accommodate a growing number of students and data, making it suitable for both small and large educational institutions.

G. **Cost-Effective Solution**: As a digital platform, the system reduces the need for physical resources like paper, thereby saving costs associated with traditional record-keeping.

1.5.2 Limitations:

A. **Limited Advanced Features**: The system avoids advanced Python and SQL features, which might restrict more complex functionalities that could be beneficial in larger, more dynamic systems.

B. **Manual Data Entry**: While the system automates many processes, initial data entry is still manual, which can be time-consuming and prone to human error.

C. **Dependency on Internet and Power**: The system's reliance on digital infrastructure means it is dependent on stable internet and power supply, which can be a limitation in areas with unreliable services.

D. **Security Concerns**: Basic security measures might not be sufficient to protect sensitive data from potential cyber threats, requiring additional security layers for more robust protection.

E. **Limited Customization**: The system is designed with general use in mind and might need further customization to fit specific needs of different educational institutions, which could involve additional development effort.

F. **Data Recovery**: Although regular backups can be scheduled, the system itself doesn’t inherently handle data recovery in case of a major system failure, which could lead to data loss without proper backup protocols.

By considering these advantages and limitations, educational institutions can evaluate the suitability of the "Student Report Card System" for their specific needs and requirements.

1.6 Future Advancement and Upgradation

Future Advancement and Upgradation

The "Student Report Card System" is designed to be a foundational tool for managing student academic records. However, as the needs of educational institutions evolve, several future advancements and upgrades can be implemented to enhance the system’s capabilities and address emerging requirements.

A. **Integration of Advanced Reporting Features**: Future upgrades could include the integration of advanced reporting features, such as graphical representations of student performance over time, class averages, and subject-wise performance analysis. This would provide teachers and administrators with deeper insights into student progress and help in academic planning.

B. **Enhanced Security Measures**: As data security becomes increasingly important, future versions of the system could incorporate advanced security protocols, such as encryption, user authentication, and role-based access control, to protect sensitive student information from unauthorized access and cyber threats.

C. **Mobile Application Development**: To increase accessibility, a mobile application version of the system could be developed. This would allow teachers and administrators to manage student data and generate reports from their smartphones or tablets, providing flexibility in data handling.

D. **Integration with Learning Management Systems (LMS)**: The system could be integrated with popular Learning Management Systems (LMS) to streamline the sharing of student performance data between different educational tools, creating a more cohesive digital learning environment.

E. **Automated Notifications and Alerts**: Future upgrades could include automated notifications and alerts to inform students and parents about grades, upcoming assessments, and other important academic information. This feature would enhance communication and engagement.

F. **Cloud-Based Data Storage**: Moving the database to a cloud-based storage solution would improve data accessibility and scalability. Cloud integration would also facilitate automatic backups and disaster recovery, ensuring data is safe and always available.

G. **Support for Multiple Languages**: To cater to a diverse user base, the system could be upgraded to support multiple languages, making it more inclusive and adaptable for institutions with multilingual requirements.

H. **AI-Driven Performance Predictions**: Incorporating artificial intelligence to predict student performance based on historical data could provide early warnings about students who might need additional support, enabling proactive interventions.

I. **Customizable User Interfaces**: Allowing users to customize the user interface according to their preferences and needs can improve user experience and satisfaction, making the system more intuitive and user-friendly.

By implementing these advancements and upgrades, the "Student Report Card System" can continue to meet the evolving demands of educational institutions, enhancing its functionality and utility in the long term.

CHAPTER 02: TECHNICAL IMPLEMENTATION

2.1 System Requirements

To successfully implement and run the "Student Report Card Management System" the following hardware and software configurations are recommended:

2.1.1) Hardware Requirements

a) **Processor:** Dual-core processor or higher (Intel Core i3 or equivalent).

b) **RAM:** Minimum 4 GB (8 GB recommended for better performance).

c) **Storage:** At least 10 GB of free disk space to store the system files and database.

d) **Display:** Monitor with a resolution of 1366x768 or higher.

e) **Input Devices:** Keyboard and mouse for interaction.

f) **Other Peripherals:** A printer for printing bills and reports (optional).

2.1.2) Software Requirements

a) **Operating System:**

- Windows 8, 10 or later
- Linux (Ubuntu 20.04 or later)
- macOS (optional, with Python and MySQL support)

b) **Programming Environment:**

- Python 3.8 or later installed on the system.

c) **Database Management System:**

- XAMPP to install MySQL and APACHE server.

d) **Required Python Libraries:**

- `mysql-connector-python` (for connecting Python to MySQL)
- `math, statistics, csv` and other libraries to implementation of filters.

e) **Code Editor (Optional):**

- Visual Studio Code, PyCharm, or any text editor for writing and managing Python scripts.

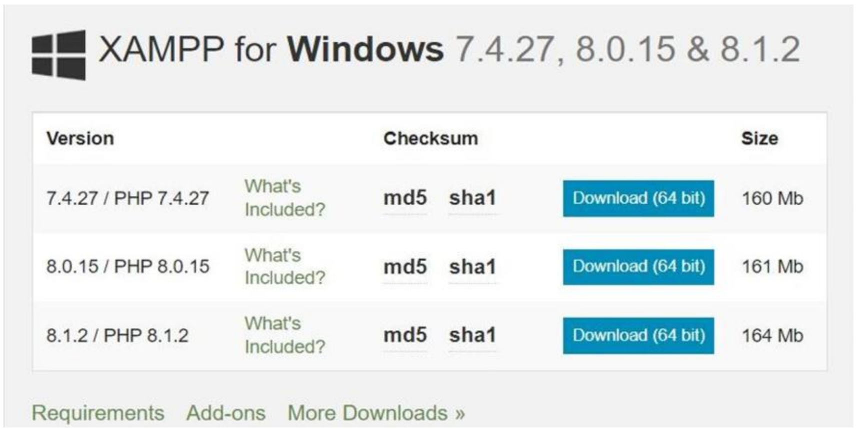
2.2 Installation of XAMPP Server

XAMPP is a free, open-source software package that provides a local server environment to run MySQL and other components required for web and database-based projects. Follow the steps below to install and configure XAMPP for the " Student Report Card Management System ":

Step 1: Download XAMPP

1. Visit the official XAMPP website: https://www.apachefriends.org.

2. Download the version compatible with the operating system (Windows, Linux, or macOS).



XAMPP for **Windows** 7.4.27, 8.0.15 & 8.1.2

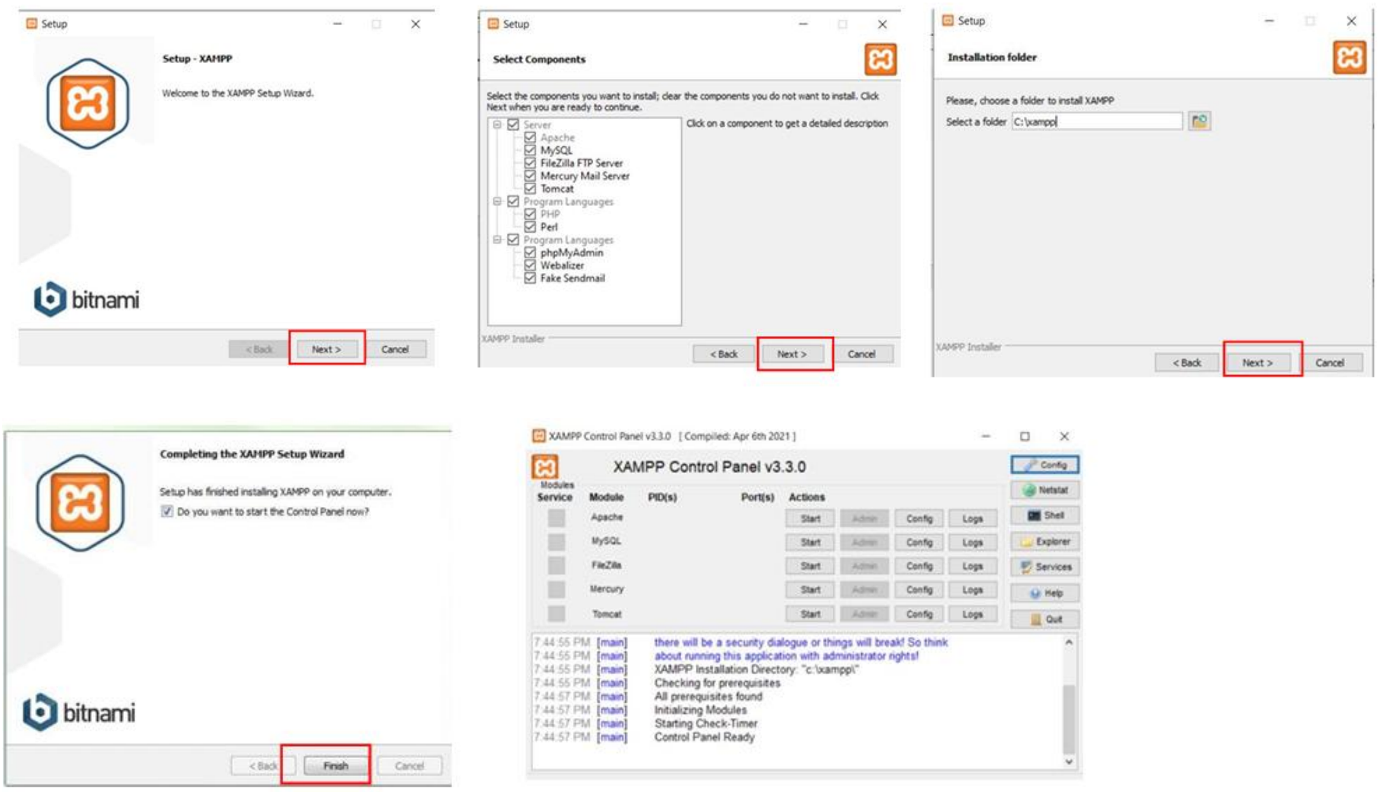
Version		Checksum			Size
7.4.27 / PHP 7.4.27	What's Included?	md5	sha1	Download (64 bit)	160 Mb
8.0.15 / PHP 8.0.15	What's Included?	md5	sha1	Download (64 bit)	161 Mb
8.1.2 / PHP 8.1.2	What's Included?	md5	sha1	Download (64 bit)	164 Mb

Requirements Add-ons More Downloads »

Fig. 2.1: XAMPP for Windows

Step 2: Install XAMPP

1. Run the downloaded installer file.
2. Follow the on-screen instructions in the installation wizard:
    - Select the components to install. For this project, ensure **MySQL** is selected.
    - Choose the installation directory (default is recommended).
3. Complete the installation by clicking "Finish."



Setup - XAMPP

Welcome to the XAMPP Setup Wizard.

Select Components

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

☑ Server
- ☑ Apache
- ☑ MySQL
- ☑ FileZilla FTP Server
- ☑ Mercury Mail Server
- ☑ Tomcat

☑ Program Languages
- ☑ PHP
- ☑ Perl

☑ Program Languages
- ☑ phpMyAdmin
- ☑ Webalizer
- ☑ Fake Sendmail

Click on a component to get a detailed description

Installation folder

Please, choose a folder to install XAMPP

Select a folder C:\xampp

Completing the XAMPP Setup Wizard

Setup has finished installing XAMPP on your computer.

☑ Do you want to start the Control Panel now?

XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

```
7:44:55 PM  [main]   there will be a security dialogue or things will break! So think
7:44:55 PM  [main]   about running this application with administrator rights!
7:44:55 PM  [main]   XAMPP Installation Directory: "c:\xampp\"
7:44:55 PM  [main]   Checking for prerequisites
7:44:57 PM  [main]   All prerequisites found
7:44:57 PM  [main]   Initializing Modules
7:44:57 PM  [main]   Starting Check-Timer
7:44:57 PM  [main]   Control Panel Ready
```

Fig. 2.2: XAMPP installation Process

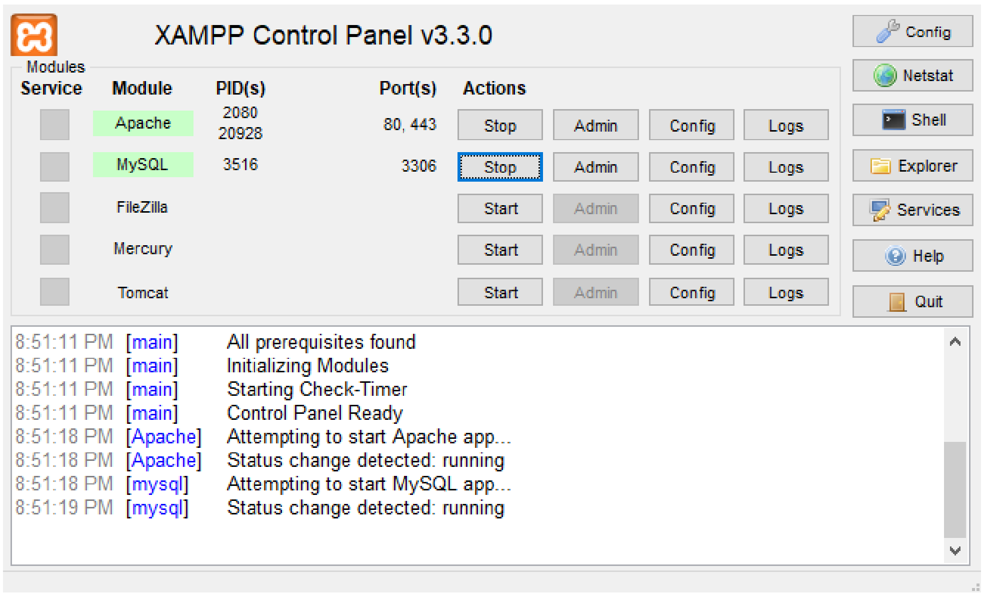
Step 3: Start XAMPP

1. Open the XAMPP Control Panel.

2. Start the **MySQL** service by clicking the "Start" button next to it.
3. Verify that MySQL is running (a green status indicator will appear).

Step 4: Access phpMyAdmin

1. Open a web browser and navigate to `http://localhost/phpmyadmin.`
2. Use phpMyAdmin to create and manage the database for the project.



Service	Module	PID(s)	Port(s)	Actions			
	Apache	2080 20928	80, 443	Stop	Admin	Config	Logs
	MySQL	3516	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
8:51:11 PM  [main]     All prerequisites found
8:51:11 PM  [main]     Initializing Modules
8:51:11 PM  [main]     Starting Check-Timer
8:51:11 PM  [main]     Control Panel Ready
8:51:18 PM  [Apache]   Attempting to start Apache app...
8:51:18 PM  [Apache]   Status change detected: running
8:51:18 PM  [mysql]    Attempting to start MySQL app...
8:51:19 PM  [mysql]    Status change detected: running
```

Fig. 2.3: Starting APACHE and MySQL sever

2.3 Testing the connection

To test the connection between Python and MySQL, we can use the **`mysql-connector-python`** library.

2.3.1. Install the MySQL Connector

```
pip install mysql-connector-python
```

2.3.2. Code to Test Connection

```
import mysql.connector
from mysql.connector import Error

try:
    # Establish a connection
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="ArohiNats123"
    )
    if connection.is_connected():
        print("Connection to MySQL was successful!")
        # Print MySQL server details
        db_info = connection.get_server_info()
```

```
            print(f"MySQL Server version: {db_info}")

    except Error as e:
        print(f"Error connecting to MySQL: {e}")
    finally:
        if 'connection' in locals() and connection.is_connected():
            connection.close()
            print("MySQL connection is closed.")
```

2.3.3. Output

If successful:

```
Connection to MySQL was successful!
MySQL Server version: 8.0.1
MySQL connection is closed.
```

If unsuccessful, it will print the error message.

2.4 List of Files

```
├── main.py → Entry point of the application
├── Database/ → Directory containing database-related files
│   └── setupDatabase.py → Script to create necessary MySQL tables
├── PythonFunctions/ → Directory containing Python function files
│   ├── databaseConnection.py → Manages database connections
│   ├── studentOperations.py → Functions for managing student details
│   ├── marksOperations.py → Functions for managing student marks
│   └── reportCardOperations.py → Functions for generating report cards
├── report_cards.csv → Sample CSV file for report cards
└── LICENSE → Project license file
```

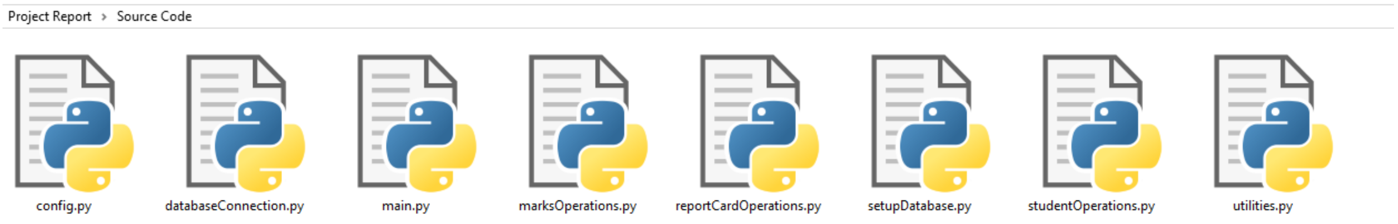


Fig. 2.4: List of Files

2.5 Key Functions of Each File

Following are the key functions of each file of this project:

2.5.1. main.py

- `displayMainMenu()`: Displays the main menu with options to add, update, and view student marks, generate report cards, etc.

  `run(): Calls the appropriate functions based on user input.`
- `exitProgram()`: Safely closes the program.

2.5.2. `databaseConnection.py`

- `createConnection()`: Establishes a connection to the MySQL database using credentials from config.py.
- `closeConnection(conn)`: Closes the open database connection after operations are completed.

2.5.3. `studentOperations.py`

- `addStudent(studentID, name, class, section)`: Adds a new student to the Students table.
- `updateStudent(studentID, name, class, section)`: Updates student information in the Students table.
- `viewStudent(studentID)`: Retrieves and displays details of a specific student.
- `viewAllStudents()`: Displays all students in the system.

2.5.4. `marksOperations.py`

- `addMarks(studentID, subject, marks)`: Adds or updates marks for a student in the Marks table.
- `updateMarks(studentID, subject, marks)`: Updates marks for a specific student and subject.
- `viewMarks(studentID)`: Retrieves and displays marks for a specific student.
- `viewAllMarks()`: Displays marks for all students.

2.5.5. `reportCardOperations.py`

- `generateReportCard(studentID)`: Calculates total marks and percentage for a student, then generates a report card.
- `viewReportCard(studentID)`: Displays the report card for a specific student.
- `viewAllReportCards()`: Displays report cards for all students.

2.5.6. `setupDatabase.py`

- `createTables()`: Creates the Students, Marks, and ReportCard tables in the MySQL database if they don't already exist.
- `initializeDatabase()`: Calls `createTables()` to set up the database schema when the system is first run.

2.5.7. `utilities.py`

- `validateInput(input)`: Validates user input to ensure it is in the correct format (e.g., numeric input for marks).

- `formatData(data)`: Formats data, such as names or grades, for consistent display.
- `calculatePercentage(totalMarks,maxMarks)`: Calculates the percentage based on total marks and maximum marks.
- `exportToCSV`: To export to data into report_cards.csv

2.5.8. `config.py`

This file stores configuration details such as:

```
dbHost: MySQL host (e.g., "localhost")
dbUser: MySQL username (e.g., "root")
dbPassword: MySQL password
dbName: Database name (e.g., "student_report_card")
```

2.6 Database Schema

Database Name: `ReportCards`

2.6.1. Students Table

Field	Data Type	Description
StudentID	`INT`	Unique identifier for each student (Primary Key)
Name	`VARCHAR(100)`	Name of the student
Class	`VARCHAR(10)`	Class of the student (e.g., 12)
Section	`VARCHAR(10)`	Section of the student (e.g., A)

2.6.2. Marks Table

Field	Data Type	Description
StudentID	`INT`	Links to StudentID in the Students table (Foreign Key)
Subject	`VARCHAR(100)`	Name of the subject (e.g., Mathematics)
Marks	`INT`	Marks obtained in the subject
Primary Key	`Composite of StudentID and Subject`	Ensures each student has unique subject entries
Foreign Key	`StudentID references Students(StudentID)`	Links marks to a specific student

2.6.3. ReportCard Table

Field	Data Type	Description
StudentID	`INT`	Links to StudentID in the Students table (Foreign Key)
TotalMarks	`INT`	Total marks obtained by the student across all subjects
Percentage	`DECIMAL(5, 2)`	Percentage calculated from the total marks
Primary Key	`StudentID`	Ensures each student has only one report card
Foreign Key	`StudentID references Students(StudentID)`	Links report card to a specific student

2.6.4. Relationship Between Students and Marks Table:

- **Type**: One-to-Many Relationship
  - Each student can have multiple marks entries for different subjects.
  - A student's StudentID in the Students table is linked to many entries in the Marks table (one entry for each subject).
  - This is achieved by using the StudentID as a foreign key in the Marks table.

2.6.5. Relationship Between Students and ReportCard Table:

- **Type**: One-to-One Relationship
  - Each student will have only one report card, containing the total marks and percentage for the student.
  - The StudentID from the Students table is used as a foreign key in the ReportCard table.
  - There is a **one-to-one** relationship because each student is associated with a single report card.

2.7 Entity Relationship Diagram

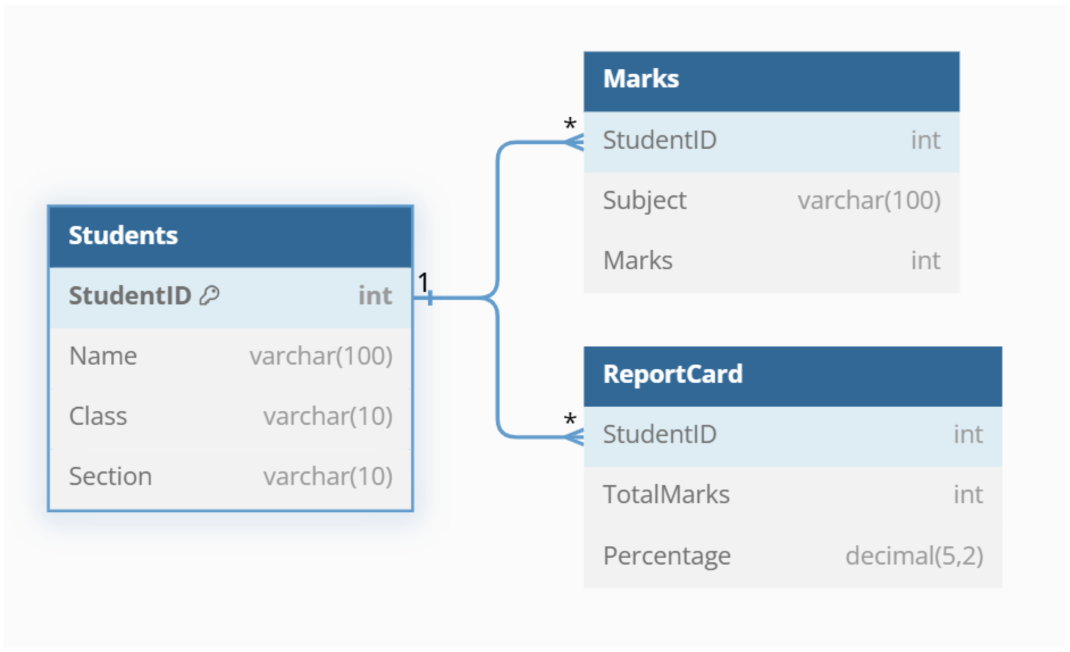


Fig 2.5: ER Diagram

Source URL to generate SQL queries into ER Diagram: https://dbdiagram.io/d

- **Students → Marks**: One-to-many relationship. A student can have multiple subjects with marks, but each subject entry in the Marks table corresponds to only one student.
- **Students → ReportCard**: One-to-one relationship. Each student has a single report card with total marks and percentage.
- **Marks → ReportCard**: Indirect relationship through StudentID. Marks are aggregated to calculate the total marks and percentage in the report card.

CHAPTER – 03: SOURCE CODE (DATABASE SCHEMA CREATION)

3.1 Key design considerations

a) **Normalization**
   - Tables are normalized to avoid data redundancy and maintain consistency.

b) **Indexes**
   - Primary keys ensure quick look up for each table.
   - Foreign keys maintain data integrity and relationships.

c) **Scalability**
   - The design can handle increasing data volumes by adding more rows without redesigning the schema.

d) **Data Integrity**
   - Foreign keys and data types ensure data consistency.

3.2 Source Code for `ReportCards.sql`

```sql
-- Active: 1731849624605@@127.0.0.1@3306@ReportCards

-- Create the database if it doesn't already exist
CREATE DATABASE IF NOT EXISTS ReportCards;

-- Use the created database
USE ReportCards;

-- Create Students table
CREATE TABLE IF NOT EXISTS Students (
    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100),
    Class VARCHAR(10),
    Section VARCHAR(10)
);

-- Create Marks table
CREATE TABLE IF NOT EXISTS Marks (
    StudentID INT,
    Subject VARCHAR(100),
    Marks INT,
    PRIMARY KEY (StudentID, Subject),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);

-- Create ReportCard table
CREATE TABLE IF NOT EXISTS ReportCard (
    StudentID INT PRIMARY KEY,
    TotalMarks INT,
    Percentage DECIMAL(5, 2),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

3.3 Testing the Code by Inserting Records in Each Table

3.3.1. Insert Records into Students Table

```
INSERT INTO Students (Name, Class, Section) VALUES
('Ravi Kumar', '12', 'A'),
('Priya Sharma', '12', 'B'),
('Aman Verma', '12', 'B'),
('Shivani Gupta', '12', 'A'),
('Vikas Singh', '12', 'C'),
('Anjali Yadav', '12', 'C'),
('Pooja Mehta', '12', 'A'),
('Nikhil Raj', '12', 'B'),
('Deepak Joshi', '12', 'C');

Query OK! 10 Rows Affected!
```

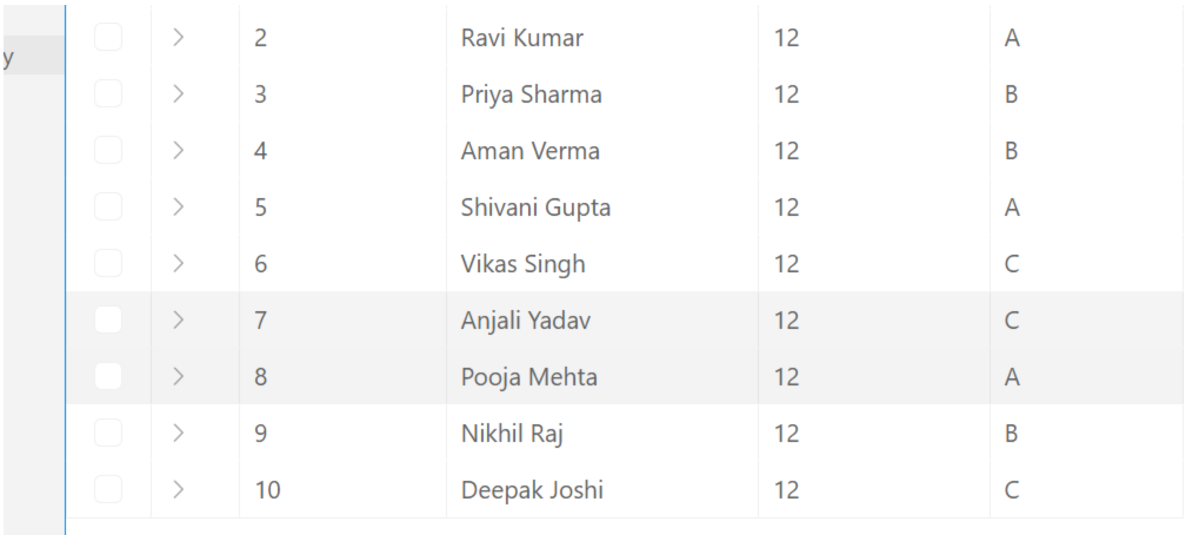
3.3.2. Insert Records into Marks Table

```
INSERT INTO Marks (StudentID, Subject, Marks) VALUES
(1, 'Physics', 85),(1, 'Chemistry', 88),(1, 'Biology', 78),(1,
'Mathematics', 92),(1, 'English', 89),
(2, 'Physics', 75),(2, 'Chemistry', 82),(2, 'Biology', 70),(2,
'Mathematics', 80),(2, 'English', 85),
(3, 'Physics', 88),(3, 'Chemistry', 79),(3, 'Biology', 81),(3,
'Mathematics', 90),(3, 'English', 92),
(4, 'Physics', 80),(4, 'Chemistry', 85),(4, 'Biology', 77),(4,
'Mathematics', 86),(4, 'English', 88),
(5, 'Physics', 91),(5, 'Chemistry', 84),(5, 'Biology', 79),(5,
'Mathematics', 89),(5, 'English', 90),
(6, 'Physics', 79),(6, 'Chemistry', 88),(6, 'Biology', 82),(6,
'Mathematics', 84),(6, 'English', 86),
(7, 'Physics', 82),(7, 'Chemistry', 79),(7, 'Biology', 75),(7,
'Mathematics', 87),(7, 'English', 80),
(8, 'Physics', 84),(8, 'Chemistry', 86),(8, 'Biology', 80),(8,
'Mathematics', 90),(8, 'English', 83),
(9, 'Physics', 90),(9, 'Chemistry', 82),(9, 'Biology', 78),(9,
'Mathematics', 85),(9, 'English', 87),
(10, 'Physics', 76),(10, 'Chemistry', 83),(10, 'Biology', 72),(10,
'Mathematics', 88),(10, 'English', 84);

 Query OK! 10 Rows Affected!
```

3.3.3. Insert Records into ReportCard Table

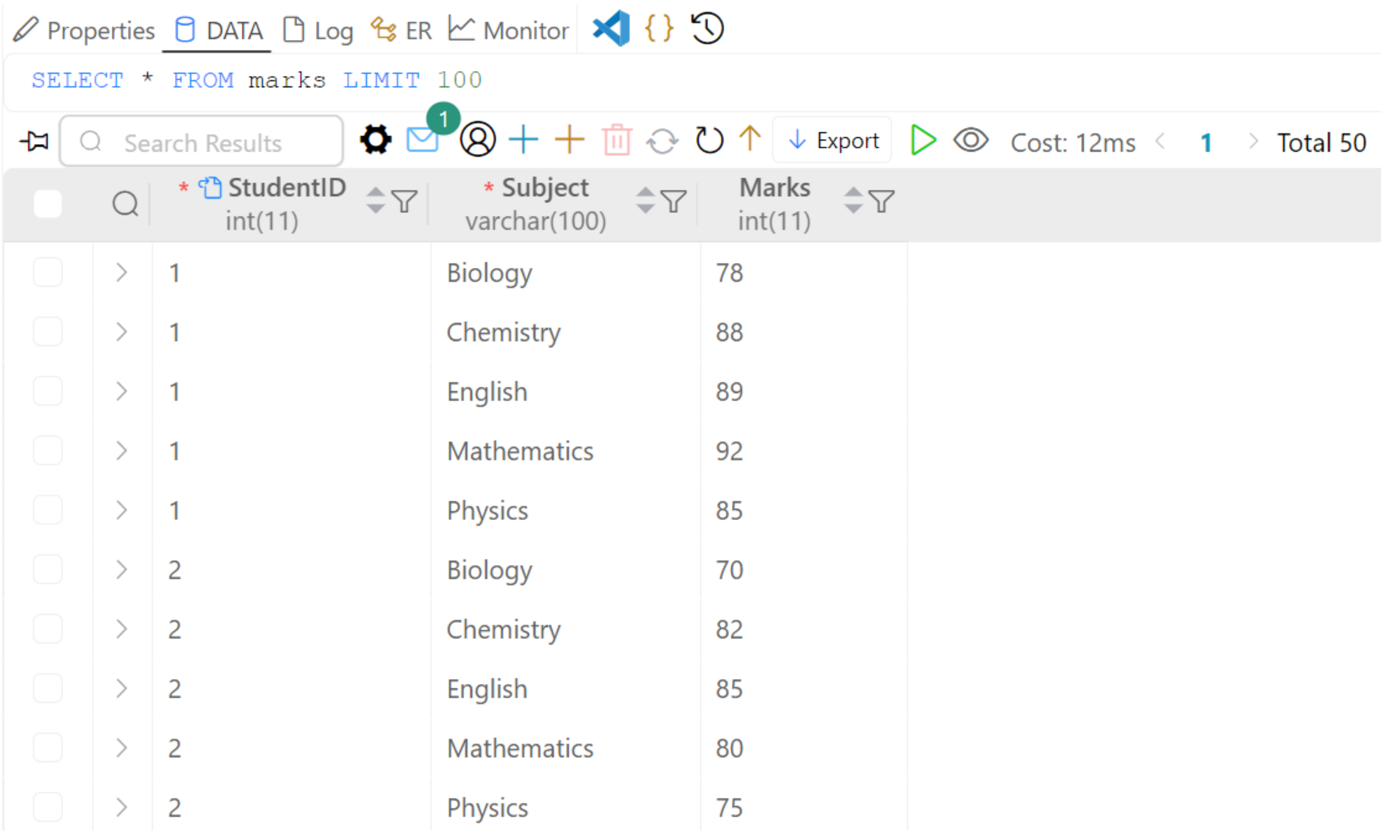
```
INSERT INTO ReportCard (StudentID, TotalMarks, Percentage) VALUES
(1, 434, 86.80), (2, 402, 80.40), (3, 430, 86.00), (4, 416, 83.20),
(5, 433, 86.60), (6, 418, 83.60), (7, 413, 82.60), (8, 434, 86.80),
(9, 423, 84.60), (10, 403, 80.60);

Query OK! 10 Rows Affected!
```

3.4 Instance of Relations



2	Ravi Kumar	12	A
3	Priya Sharma	12	B
4	Aman Verma	12	B
5	Shivani Gupta	12	A
6	Vikas Singh	12	C
7	Anjali Yadav	12	C
8	Pooja Mehta	12	A
9	Nikhil Raj	12	B
10	Deepak Joshi	12	C

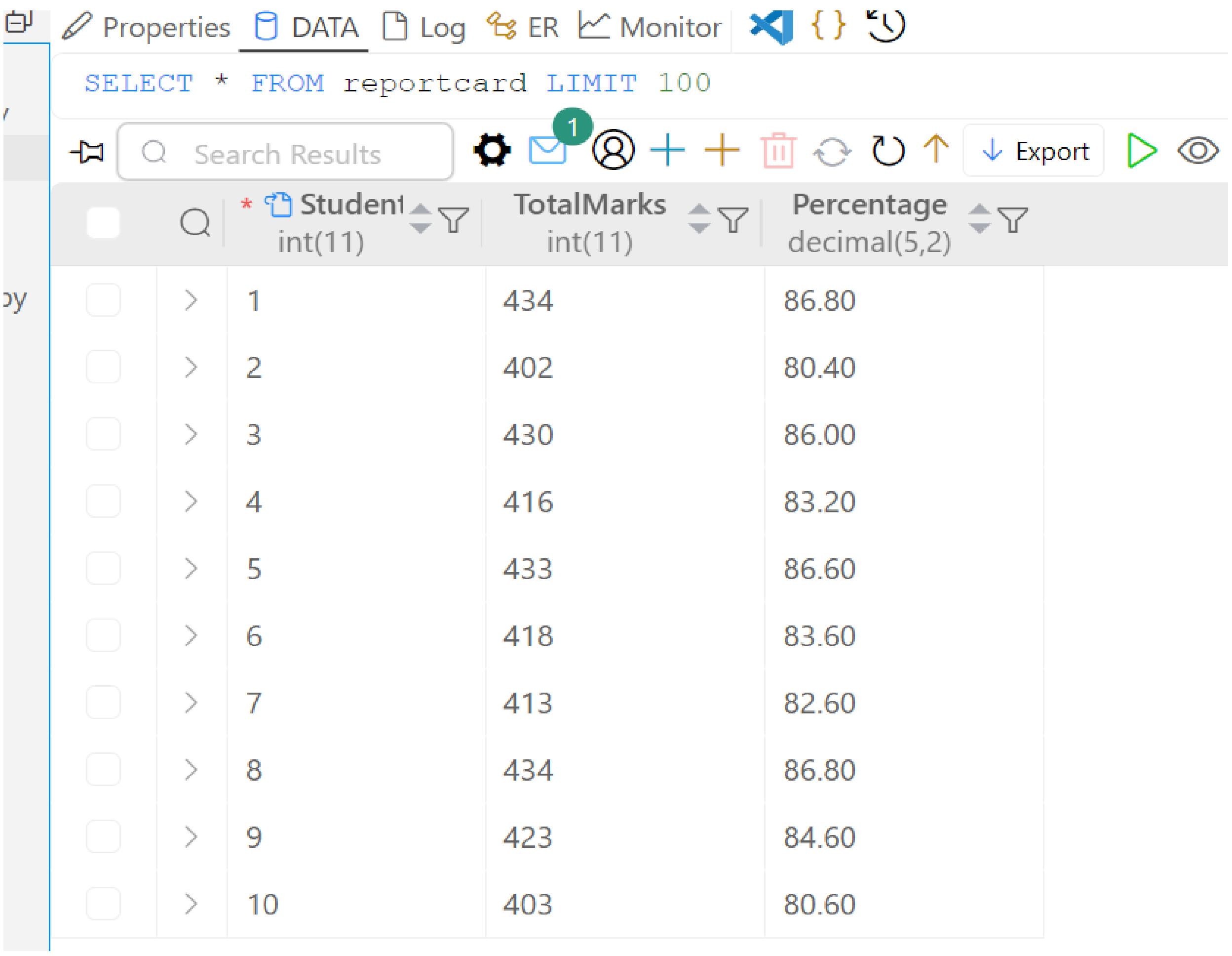
Fig. 3.3: Instance of `Students` table



```
SELECT * FROM marks LIMIT 100
```

* StudentID int(11)	* Subject varchar(100)	Marks int(11)
1	Biology	78
1	Chemistry	88
1	English	89
1	Mathematics	92
1	Physics	85
2	Biology	70
2	Chemistry	82
2	English	85
2	Mathematics	80
2	Physics	75

Fig. 3.4: Instance of `Marks` table



```
SELECT * FROM reportcard LIMIT 100
```

Student int(11)	TotalMarks int(11)	Percentage decimal(5,2)
1	434	86.80
2	402	80.40
3	430	86.00
4	416	83.20
5	433	86.60
6	418	83.60
7	413	82.60
8	434	86.80
9	423	84.60
10	403	80.60

Fig. 3.5: Instance of `ReportCard` table

CHAPTER 04: PYTHON PROGRAM SOURCE CODE

4.1 Database Configuration

Source File: **`config.py`**

```
dbConfig = {
    "host": "localhost", "user": "root", "password": "",
    "database": " ReportCards"
}
```

4.2 Database Connection

Source file: **`databaseConnection.py`**

```
import mysql.connector
from config import DB_CONFIG
def createConnection():
    try:
        connection = mysql.connector.connect(
            host=DB_CONFIG['host'],
            user=DB_CONFIG['user'],
            password=DB_CONFIG['password'],
            database=DB_CONFIG['database']
        )
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None
def closeConnection(conn):
    if conn:
        conn.close()
```

4.3 Functions for managing student details

Source file: **`studentOperations.py`**

```
import mysql.connector
from databaseConnection import createConnection, closeConnection

def addStudent(studentID, name, student_class, section):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()

        # SQL query to insert a new student into the Students table
        query = "INSERT INTO Students (StudentID, Name, Class,
                    Section) VALUES (%s, %s, %s, %s)"
        values = (studentID, name, student_class, section)

        cursor.execute(query, values)
        connection.commit()

        print(f"Student {name} added successfully.")
        closeConnection(connection)

def updateStudent(studentID, name, student_class, section):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()

        # SQL query to update student information in the Students table
        query = "UPDATE Students SET Name = %s, Class = %s, Section
                 = %s WHERE StudentID = %s"
        values = (name, student_class, section, studentID)

        cursor.execute(query, values)
        connection.commit()

        if cursor.rowcount > 0:
            print(f"Student {studentID} updated successfully.")
        else:
            print(f"Student {studentID} not found.")

        closeConnection(connection)

def viewStudent(studentID):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
       # Query to retrieve details of a specific student
        query = "SELECT * FROM Students WHERE StudentID = %s"
        cursor.execute(query, (studentID,))
        student = cursor.fetchone()

        if student:
            print("\nStudent Details:")
            print(f"Student ID: {student[0]}")
            print(f"Name: {student[1]}")
            print(f"Class: {student[2]}")
            print(f"Section: {student[3]}")
        else:
            print(f"Student with ID {studentID} not found.")
        closeConnection(connection)

def viewAllStudents():
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
        # SQL query to retrieve all students from the Students table
        query = "SELECT * FROM Students"
        cursor.execute(query)
        students = cursor.fetchall()

        if students:
            print("\nAll Students:")
            for student in students:
                print(f"Student ID: {student[0]}, Name: 
                   {student[1]}, Class: {student[2]}, Section: 
                   {student[3]}")
        else:
            print("No students found.")

        closeConnection(connection)

def deleteStudent(studentID):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
        try:
            # Delete student's report card
            cursor.execute("DELETE FROM ReportCard WHERE StudentID = 
                             %s", (studentID,))
            print(f"Report card for student {studentID} deleted.")
            # Delete student's marks
            cursor.execute("DELETE FROM Marks WHERE StudentID = %s", 
                            (studentID,))
            print(f"Marks for student {studentID} deleted.")
            # Delete student's details
            cursor.execute("DELETE FROM Students WHERE StudentID = 
                             %s", (studentID,))
            print(f"Student record for student {studentID}deleted.")
            connection.commit()

        except mysql.connector.Error as err:
            print(f"Error: {err}")
        finally:
            closeConnection(connection)
```

4.4 Student’s Marks Operations

Source File: **`marksOperations.py`**

```
import mysql.connector
from databaseConnection import createConnection, closeConnection

def addMarks(studentID, subject, marks):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
   # SQL query to insert marks for a specific student and subject
        query = "INSERT INTO Marks (StudentID, Subject, Marks)
                 VALUES (%s, %s, %s)"
        values = (studentID, subject, marks)

        cursor.execute(query, values)
        connection.commit()

        print(f"Marks for {subject} added successfully for student
              {studentID}.")
        closeConnection(connection)

def updateMarks(studentID, subject, marks):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
    # SQL query to update marks for a specific student and subject
        query = "UPDATE Marks SET Marks = %s WHERE StudentID = %s
                   AND Subject = %s"
        values = (marks, studentID, subject)
        cursor.execute(query, values)
        connection.commit()

        if cursor.rowcount > 0:
            print(f"Marks for {subject} updated successfully for
                  student {studentID}.")
        else:
            print(f"Marks for {subject} not found for student
                  {studentID}.")

        closeConnection(connection)

def viewMarks(studentID):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
        # SQL query to retrieve marks for a specific student
        query = "SELECT Subject,Marks FROM Marks WHERE StudentID=%s"
        cursor.execute(query, (studentID,))
        marks = cursor.fetchall()
        if marks:
            print("\nMarks for Student ID:", studentID)
            for subject, marks_obtained in marks:
                print(f"Subject: {subject}, Marks:{marks_obtained}")
        else:
            print(f"No marks found for student {studentID}.")
        closeConnection(connection)

def viewAllMarks():
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
        # SQL query to retrieve all marks from the Marks table
        query = "SELECT StudentID, Subject, Marks FROM Marks"
        cursor.execute(query)
        all_marks = cursor.fetchall()

        if all_marks:
            print("\nAll Marks:")
            for studentID, subject, marks in all_marks:
                print(f"Student ID: {studentID}, Subject: {subject},
                        Marks: {marks}")
        else:
            print("No marks found in the system.")
        closeConnection(connection)
```

4.5 Program to manage or generate report card

Source File: **`reportCardOperation.py`**

```
import mysql.connector
from databaseConnection import createConnection, closeConnection
from marksOperations import viewMarks
from utilities import calculatePercentage
def generateReportCard(studentID):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
        # Retrieve the marks for the student
        cursor.execute("SELECT Subject, Marks FROM Marks WHERE
                       StudentID = %s", (studentID,))
        marks = cursor.fetchall()
        if not marks:
            print(f"No marks found for student {studentID}.")
            closeConnection(connection)
            return
        # Calculate the total marks
        total_marks = sum(mark[1] for mark in marks)
        # Assuming the maximum marks for each subject is 100 and
the student has 5 subjects
        max_marks = len(marks) * 100  # Adjust this value if there
          are more/less subjects
        percentage = calculatePercentage(total_marks, max_marks)
        # Insert or update the report card
        cursor.execute("""
           INSERT INTO ReportCard (StudentID, TotalMarks,
           Percentage)
           VALUES (%s, %s, %s)
           ON DUPLICATE KEY UPDATE TotalMarks = %s, Percentage = %s
        """, (studentID, total_marks, percentage, total_marks,
             percentage))
        connection.commit()
        print(f"Report card generated for student {studentID} with
          Total Marks: {total_marks} and Percentage: {percentage}%")
        closeConnection(connection)

def viewReportCard(studentID):
    connection = createConnection()
    if connection:
        cursor = connection.cursor()
     # SQL query to retrieve the report card for a specific student
        query = "SELECT TotalMarks, Percentage FROM ReportCard
                WHERE StudentID = %s"
        cursor.execute(query, (studentID,))
        report_card = cursor.fetchone()

        if report_card:
            print("\nReport Card for Student ID:", studentID)
            print(f"Total Marks: {report_card[0]}")
            print(f"Percentage: {report_card[1]}%")
        else:
            print(f"Report card not found for student{studentID}.")
        closeConnection(connection)

def viewAllReportCards():
    connection = createConnection()
    all_report_cards = []
    if connection:
        cursor = connection.cursor()
        # SQL query to retrieve all data for report card
        query = """
        SELECT Students.Name, Marks.Subject, Marks.Marks,
        ReportCard.StudentID, ReportCard.TotalMarks,
        ReportCard.Percentage
        FROM ReportCard
        JOIN Students ON ReportCard.StudentID = Students.StudentID
        JOIN Marks ON ReportCard.StudentID = Marks.StudentID
        """
        cursor.execute(query)
        report_cards = cursor.fetchall()

        if report_cards:
            print("\nAll Report Cards:")
            for name, subject, marks, studentID, totalMarks,
             percentage in report_cards:
                print(f"Student Name: {name}, Subject: {subject},
                 Marks: {marks}, Student ID: {studentID}, Total
                 Marks: {totalMarks}, Percentage: {percentage}%")
                # Adding to the list to return for CSV export
                all_report_cards.append((name, subject, marks,
                 studentID, totalMarks, percentage))
        else:
            print("No report cards found.")
        closeConnection(connection)
    return all_report_cards
```

4.6 Setting up the database (for Backup)

Source File: **`setupDatabase.py`**

```
from databaseConnection import createConnection
from config import DB_CONFIG
def createTables():
    connection = createConnection()
    cursor = connection.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Students (
            StudentID INT AUTO_INCREMENT PRIMARY KEY,
            Name VARCHAR(100),
            Class VARCHAR(10),
            Section VARCHAR(10)
        )""")
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Marks (
            StudentID INT,
            Subject VARCHAR(100),
            Marks INT,
            PRIMARY KEY (StudentID, Subject),
            FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
        ) """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS ReportCard (
            StudentID INT,
            TotalMarks INT,
            Percentage DECIMAL(5,2),
            PRIMARY KEY (StudentID),
            FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
        ) """)
    connection.commit()
    connection.close()

def initializeDatabase():
    createTables()
    print("Database and tables created successfully.")
#"__main__":
initializeDatabase()
```

4.6 Utilities functions

Source File: **`utilities.py`**

```
import csv
def calculatePercentage(totalMarks, maxMarks):
    # Calculate the percentage based on total and maximum marks
    return (totalMarks / maxMarks) * 100

def exportToCSV(report_cards):
    if report_cards:
        try:
            with open("ReportCards.csv", mode="w", newline="") as file:
                writer = csv.writer(file)
                # Writing headers to the CSV file
                writer.writerow(["Student Name", "Subject", "Marks",
                        "Student ID", "Total Marks", "Percentage"])
                # Writing data rows to the CSV file
                for record in report_cards:
                    writer.writerow(record)
            print("Report cards exported successfully!")
        except Exception as e:
            print(f"Error exporting report cards to CSV: {e}")
    else:
        print("No report cards available to export.")
```

4.7 Driver Program

Source File: **`main.py`**

```
from studentOperations import addStudent, updateStudent,
viewAllStudents, deleteStudent
from marksOperations import addMarks, updateMarks, viewAllMarks
from reportCardOperations import generateReportCard, viewReportCard,
viewAllReportCards
from utilities import exportToCSV
def displayMainMenu():
    print("\n--- Student Report Card System ---")
    print("1. Add Student")
    print("2. Update Student")
    print("3. View All Students")
    print("4. Add Marks")
    print("5. Update Marks")
    print("6. View All Marks")
    print("7. Generate Report Card")
    print("8. View Report Card")
    print("9. Export All Report Cards to CSV")
    print("10. Delete Student Record")
    print("11. Exit")
def run():
    while True:
        displayMainMenu()
        try:
            choice = int(input("Please select an option (1-11): "))
            if choice == 1:  # Add Student
                studentID = int(input("Enter Student ID: "))
                name = input("Enter Student Name: ")
                class_ = input("Enter Class (e.g., 12): ")
                section = input("Enter Section (e.g., A): ")
                addStudent(studentID, name, class_, section)

            elif choice == 2:  # Update Student
                studentID = int(input("Enter Student ID: "))
                name = input("Enter Student Name: ")
                class_ = input("Enter Class (e.g., 12): ")
                section = input("Enter Section (e.g., A): ")
                updateStudent(studentID, name, class_, section)

            elif choice == 3:  # View All Students
                viewAllStudents()

            elif choice == 4:  # Add Marks
                studentID = int(input("Enter Student ID: "))
                subject = input("Enter Subject: ")
                marks = int(input("Enter Marks: "))
                addMarks(studentID, subject, marks)

            elif choice == 5:  # Update Marks
                studentID = int(input("Enter Student ID: "))
                subject = input("Enter Subject: ")
                marks = int(input("Enter Marks: "))
                updateMarks(studentID, subject, marks)

            elif choice == 6:  # View All Marks
                viewAllMarks()

            elif choice == 7:  # Generate Report Card
                studentID = int(input("Enter Student ID: "))
                generateReportCard(studentID)

            elif choice == 8:  # View Report Card
                studentID = int(input("Enter Student ID: "))
                viewReportCard(studentID)

            elif choice == 9:  # Export All Report Cards to CSV
                report_cards = viewAllReportCards()
                exportToCSV(report_cards)

            elif choice == 10:  # Delete Student Record
                studentID = int(input("Enter Student ID to delete:"))
                deleteStudent(studentID)

            elif choice == 11:  # Exit Program
                exitProgram()
            else:
                print("Invalid choice, please select between 1- 11.")
        except ValueError:
            print("Invalid input, please enter a number between 1-11.")

     def exitProgram():
         print("Exiting the program.")
         exit()
     #"__main__":
     run()
```

CHAPTER 05: EXECUTING THE PROGRAM

```
PS> python -u "c:\Users\Desktop\CSProject\Code\ main.py"

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 3

All Students:
Student ID: 1, Name: Sagar Maindola, Class: 12, Section: A
Student ID: 2, Name: Ravi Kumar, Class: 12, Section: A
Student ID: 3, Name: Priya Sharma, Class: 12, Section: B
Student ID: 4, Name: Aman Verma, Class: 12, Section: B
Student ID: 5, Name: Shivani Gupta, Class: 12, Section: A
Student ID: 6, Name: Vikas Singh, Class: 12, Section: C
Student ID: 7, Name: Anjali Yadav, Class: 12, Section: C
Student ID: 8, Name: Pooja Mehta, Class: 12, Section: A
Student ID: 9, Name: Nikhil Raj, Class: 12, Section: B
Student ID: 10, Name: Deepak Joshi, Class: 12, Section: C

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 1
Enter Student ID: 11
Enter Student Name: Saksham
Enter Class (e.g., 12): 12
Enter Section (e.g., A): A
Student Saksham added successfully.

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
```

```
11. Exit
Please select an option (1-11): 3

All Students:
Student ID: 1, Name: Sagar Maindola, Class: 12, Section: A
Student ID: 2, Name: Ravi Kumar, Class: 12, Section: A
Student ID: 3, Name: Priya Sharma, Class: 12, Section: B
Student ID: 4, Name: Aman Verma, Class: 12, Section: B
Student ID: 5, Name: Shivani Gupta, Class: 12, Section: A
Student ID: 6, Name: Vikas Singh, Class: 12, Section: C
Student ID: 7, Name: Anjali Yadav, Class: 12, Section: C
Student ID: 8, Name: Pooja Mehta, Class: 12, Section: A
Student ID: 9, Name: Nikhil Raj, Class: 12, Section: B
Student ID: 10, Name: Deepak Joshi, Class: 12, Section: C
Student ID: 11, Name: Saksham, Class: 12, Section: A

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 4
Enter Student ID: 11
Enter Subject: Biology
Enter Marks: 85
Marks for Biology added successfully for student 11.

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 4
Enter Student ID: 11
Enter Subject: Mathematics
Enter Marks: 85
Marks for Mathematics added successfully for student 11.

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
```

```
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 4
Enter Student ID: 11
Enter Subject: Physics
Enter Marks: 90
Marks for Physics added successfully for student 11.

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 7
Enter Student ID: 11
Report card generated for student 11 with Total Marks: 260 and Percentage:
86.66666666666667%

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 8
Enter Student ID: 1
Report Card for Student ID: 1
Total Marks: 434
Percentage: 86.80%

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 9
Report cards exported successfully to ReportCards.csv.
../ReportCards.csv
```

```
--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 10
Enter Student ID to delete: 5
Report card for student 5 deleted.
Marks for student 5 deleted.
Student record for student 5 deleted.

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 3

All Students:
Student ID: 1, Name: Sagar Maindola, Class: 12, Section: A
Student ID: 2, Name: Ravi Kumar, Class: 12, Section: A
Student ID: 3, Name: Priya Sharma, Class: 12, Section: B
Student ID: 4, Name: Aman Verma, Class: 12, Section: B
Student ID: 6, Name: Vikas Singh, Class: 12, Section: C
Student ID: 7, Name: Anjali Yadav, Class: 12, Section: C
Student ID: 8, Name: Pooja Mehta, Class: 12, Section: A
Student ID: 9, Name: Nikhil Raj, Class: 12, Section: B
Student ID: 10, Name: Deepak Joshi, Class: 12, Section: C
Student ID: 11, Name: Saksham, Class: 12, Section: A

--- Student Report Card System ---
1. Add Student
2. Update Student
3. View All Students
4. Add Marks
5. Update Marks
6. View All Marks
7. Generate Report Card
8. View Report Card
9. Export All Report Cards to CSV
10. Delete Student Record
11. Exit
Please select an option (1-11): 11
Exiting the program.
PS C:\Users\Sagar Maindola\Desktop\CSProject\Code>
```

CONCLUSION

The **Student Report Card System** project successfully demonstrates the ability to manage student data and generate reports using Python and MySQL. The system offers various functionalities to manage student details, marks, and generate report cards with a focus on simplicity and ease of use. Below are the key highlights of the system:

1. **Student Management**: The system allows for the addition, updating, and viewing of student details. Teachers can input basic student information such as StudentID, Name, Class, and Section.
2. **Marks Management**: The system facilitates the addition, updating, and viewing of student marks for different subjects. Teachers can input marks, which are linked to individual students, and can also update existing marks if required.
3. **Report Card Generation**: The core feature of the system involves generating report cards for students, calculating their total marks and percentage, and storing the results in the database. The system uses the total marks from all subjects to calculate a percentage, which is then displayed in the report card.
4. **Export to CSV**: A useful feature allows teachers to export the report cards of all students to a CSV file for offline analysis or record-keeping. This feature includes essential student information, marks for each subject, and the generated percentage.
5. **Data Integrity and Deletion**: The system ensures data integrity by allowing for the deletion of a student's records, including their details, marks, and report card. This helps maintain an accurate database as student information is updated or removed.
6. **MySQL Database**: The project uses MySQL for storing student details, marks, and report card information, ensuring the data is persistent and can be queried efficiently.

Overall, the **Student Report Card System** serves as an efficient tool for managing student data, marks, and generating automated report cards. The system simplifies administrative tasks for teachers and provides a reliable way to track and manage students' academic progress. It also offers the flexibility to handle various operations, such as adding, updating, viewing, deleting, and exporting student records. The integration of Python with MySQL ensures smooth data handling and scalability for larger sets of student data.

This project can be further expanded by adding features like subject-specific grade-based report cards, integrating a web-based interface, or adding user authentication to allow different levels of access to the system.

BIBLIOGRAPHY AND REFERENCES

❖ **NCERT Computer Science 2024**
Title: Computer Science (Class XI and XII)
Publisher: National Council of Educational Research and Training (NCERT)
Description: The official NCERT textbook for Class XII Computer Science, covering Python programming basics, MySQL database concepts, and project-based learning approaches.

❖ **Sumita Arora – Computer Science with Python 2024**
Title: Computer Science with Python (Class XI and XII)
Author: Sumita Arora
Description: A comprehensive book for Class XII students, providing detailed explanations of Python programming and database integration with MySQL. The book includes practical examples and project ideas.

❖ **CBSE Study Materials**
Source: Central Board of Secondary Education (CBSE)
Description: Official study resources and guidelines provided by CBSE for Class XII Computer Science and Informatics Practices, focusing on programming, database management, and project development.

❖ **MySQL Documentation**
Title: MySQL 8.0 Reference Manual
Source: Oracle Corporation
Website: https://dev.mysql.com/doc/
Description: Comprehensive documentation on MySQL, covering database creation, table management, and SQL query execution.

❖ **Python Official Documentation**
Title: Python 3 Documentation
Source: Python Software Foundation
Website: https://docs.python.org/3/
Description: Official Python documentation for understanding basic Python syntax, libraries, and database integration using the mysql.connector library.

❖ **GeeksforGeeks – Python and MySQL**
Website: https://www.geeksforgeeks.org/
Description: Online platform providing tutorials, examples, and solutions for integrating Python with MySQL and building simple database-driven applications.

❖ **Stack Overflow Discussions**
Source: https://stackoverflow.com/
Description: Developer community forums that provide solutions and best practices for Python programming and MySQL integration issues.