

# **BACSE101 Problem Solving using Python**

## **PROJECT REPORT**

on

# **UNIVERSITY/EXAMINATION PORTAL ADMIN/CLIENT ACCESS MANAGEMENT**

*Prepared by*

**Atharv Poundrik – 25BCE0948**

**Dhruv Mohan Parab – 25BCE0950**

**Vrushabh Vasudev Parab – 25BCE0906**

**LV Ram Charitesh -25BCE0949**

*Under the supervision of*

**Prof. Padma Priya R**



# **VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and Engineering  
Vellore Institute of Technology, Vellore.**

November 07, 2025

# **Table of Contents**

Abstract

1. Introduction
2. Problem Statement and Objectives
3. Implementation Code
4. Demo Screenshots
5. Important Links
6. Conclusion

## Abstract

*This project is a **command-line Examination Portal** built using **Python**. It uses **Role-Based Access Control (RBAC)** to manage university examination records securely. The system has two types of users **students** and **faculty** each with specific permissions.*

- ***Students** can log in to view their own academic details, such as General Management Scores, Domain-Specific Scores, total percentage, and remarks. However, they **cannot change or edit** any data.*
- ***Faculty members** have full administrative access. They can **add new student records, update marks, assign remarks, delete profiles, and sort the student database.***

*By separating user roles and permissions, the portal ensures **data security, accuracy, and integrity**. This makes it a reliable and scalable solution for managing sensitive examination data within an academic institution.*

# 1. Introduction

Managing academic records especially examination results requires both efficiency and strong data security. Traditional systems often fail to control access properly, leading to risks like unauthorized data changes or leaks. This project presents the **Examination Portal**, a Python-based system that uses **Role-Based Access Control (RBAC)** to securely manage academic information. It creates a clear two-level structure: **students** can only log in to view their own scores and remarks, while **faculty members** have full administrative access to update marks, manage student data, and organize records. By clearly defining what each user can and cannot do, the system prevents data misuse, simplifies administration, and ensures a safe, reliable, and modern solution for handling academic records.

## 1.1 Domain Information

The project belongs to the **Education Management System** domain, focusing on **university examination administration**. It addresses the management of student academic records and result processing through a secure and efficient **Role-Based Access Control (RBAC)** model. This domain emphasizes **data security, user authentication, and academic record management** within an institutional environment.

## 1.2 Software Libraries Used

The software libraries used are pandas, time and sqlite3.

## 1.3 Contributions by Team Members

The project's code contributions were divided as follows: **Atharv Poundrik** and **LV Ram Charitesh** focused on the **pandas** integration, while **Dhruv Mohan Parab** and **Vrushabh Vasudev Parab** handled the **SQL** aspects. Furthermore, every team member played a part in developing the **core pure Python** code.

## 1.4 Challenges Faced

Since performing CRUD operations directly in Pandas became complex and inefficient for managing large datasets, we integrated SQL-based CRUD commands within the Pandas workflow to simplify data manipulation and improve clarity.

## 2. Problem Statement and Objectives

### Problem Statement:

Managing sensitive university examination data through traditional or unsecured systems presents significant risks, primarily **unauthorized access, data tampering, and a lack of accountability**. Specifically, a unified access system grants students the potential to manipulate their own scores, and faculty members may struggle to efficiently update, track, and secure large volumes of records without a clear administrative interface. The core problem is the need for an **efficient, secure, and role-segregated system** to manage student academic performance data, ensuring that students can only view their information while providing faculty with robust, reliable tools for comprehensive data management (Create, Read, Update, Delete).

### Project Objectives:

The primary goal of this project is to develop a command-line Examination Portal using Python and Role-Based Access Control (RBAC) to securely manage university examination records. The specific objectives are:

1. **Develop a Secure Authentication System:** To implement a robust login mechanism that authenticates users and assigns roles (Student or Faculty) to enforce appropriate access controls.
2. **Segregate User Permissions (RBAC):** To ensure a strict separation of duties where:
  - **Students** are restricted to **Read-Only** access, allowing them to view their specific General Management Scores, Domain-Specific Scores, total percentage, and remarks.
  - **Faculty** are granted **Full Administrative (CRUD)** permissions to add new student records, update existing marks, assign remarks, delete student profiles, and sort the database.
3. **Implement CRUD Functionality for Faculty:** To provide faculty members with a comprehensive menu-driven interface to perform the following operations efficiently on the student data: **Create** (add new records), **Read** (view all records), **Update** (modify scores and remarks), and **Delete** (remove profiles).
4. **Ensure Data Integrity and Accuracy:** To utilize structured data management (using Pandas and integrated SQL logic) to maintain consistency and accuracy in all academic records and ensure all calculated metrics, like total percentage, are correctly derived.
5. **Deliver a User-Friendly Console Interface:** To design a clear, intuitive command-line interface that allows both faculty and students to navigate the system and access features with ease.

## 3. Implementation

### 3.1 Main Menu and Role Selection

The main function (main()) presents the initial menu to the user, allowing them to select their role (Student or Faculty) before logging in. This is the entry point for the RBAC system.

Option	Description	Access Role
1	Student Login	Student
2	Faculty Login	Faculty
3	Exit	General

```
Code:
def main():
    while True:
        print("\n--- Examination Portal ---")
        print("1. Student Login")
        print("2. Faculty Login")
        print("3. Exit")
        choice = input("Enter Choice: ")
        if choice == '1':
            student_login()
        elif choice == '2':
            faculty_login()
        elif choice == '3':
            break
        else:
            print("Invalid Choice!")
```

```
main()
```

## 3.2 Student Read-Only Access

The student\_login() function implements the **Read-Only** access restriction for students. After successful authentication against the student\_data.csv file, a student can only view their own academic details , ensuring data security.

### Student Features Menu:

**1.Show Marks:** Displays General Management Score, Domain-Specific Score, Total Score, and Percentage.

**2.Show Remarks:** Displays the assigned remark for the student.

**3 Logout**

Code:

```
def student_login():
    df = pd.read_csv("student_data.csv")
    print("----STUDENT LOGIN----")
    loginId = input("Enter your loginId: ")
    password = input("Enter password: ")
    student = df[(df["Login_IDs"] == loginId) & (df["PASSWORDS"] == password)]
    student = student.reset_index()
    if student.empty:
        print("Invalid Credentials. Try Again!")
        return
    print("Loading....")
    time.sleep(1)
    print(f'Welcome!! {student.loc[0]["NAME_OF_THE_STUDENT"]}')
    while True:
        print("1. Show Marks")
        print("2. Show Remarks")
        print("3. Logout")
        choice = int(input("Enter choice: "))
        time.sleep(0.5)
        if choice == 1:
            print("Loading....")
            print("1. General Management Score (OUT of 50)")
            print("2. Domain Specific Score (OUT of 50)")
            print("3. Total Score and Percentage")
            choice1 = int(input("Enter your choice: "))
            gms = student.loc[0]["GENERAL_MANAGEMENT_SCORE_OUT_of_50"]
            dss = student.loc[0]["DOMAIN_SPECIFIC_SCORE_OUT_50"]
            if choice1 == 1:
                print("Loading....")
                time.sleep(0.5)
                print("Your score is:", gms)
            elif choice1 == 2:
                print("Loading....")
                time.sleep(0.5)
                print("Your score is:", dss)
            elif choice1 == 3:
                print("Loading....")
                time.sleep(0.5)
                totalScore = gms + dss
                print("Your Total Score:", totalScore)
```

```
percentage = (totalScore/100) * 100
percentage = round(percentage)
print("Percentage obtained:",percentage,"%")
else:
    print("Invalid choice!")
elif choice == 2:
    print("Loading....")
    time.sleep(0.5)
    print(student.loc[0]["REMARKS"])
elif choice == 3:
    print("Logging out...")
    time.sleep(0.5)
    break
else:
    print("Invalid Choice")
```



### 3.3 Faculty Administrative Access (CRUD Operations)

The `faculty_login()` function provides **Full Administrative (CRUD)** permissions to faculty members. It uses both **pandas** for immediate data manipulation and a seamless **sqlite3** integration to manage the student database, simplifying complex operations. The faculty's CSV data is used for authentication against their login ID and password.

#### Faculty Features Menu:

**1.Upload/Update Student Marks:** Allows updating General Management Score, Domain-Specific Score, and Remarks for an existing student (an **Update** operation).

**2.Sort Students (SQL):** Allows sorting the student database by various columns like Name, Total Score, etc.

**3.Add Student (SQL):** Allows creating and adding a new student record to the database (a **Create** operation).

**4.Remove Student (SQL):** Allows deleting a student profile from the database (a **Delete** operation).

**5.Logout**

Code:

```
def faculty_login():
    import sqlite3
    df1 = pd.read_csv("student_data.csv")
    df2 = pd.read_csv("faculty_data.csv")

    print("----FACULTY LOGIN----")
    t_id = input("Enter login ID: ")
    password = input("Enter password: ")
    faculty = df2[(df2["LoginID"] == t_id) & (df2["password"] == password)]
    faculty = faculty.reset_index()

    if faculty.empty:
        print("Invalid credentials. Try again")
        return

    print("Loading...")
    time.sleep(1)
    print(f"Welcome!! {faculty.iloc[0]['Name']}")

    conn = sqlite3.connect("examination.db")
    cur = conn.cursor()

    df1.columns = df1.columns.str.strip()
    df1.columns = df1.columns.str.replace(' ', '_')
    df1.columns = df1.columns.str.replace('[()]', '', regex=True)
    df1.to_sql("students", conn, if_exists="replace", index=False)

    while True:
        print("1. Upload/Update Student Marks")
        print("2. Sort Students (SQL)")
        print("3. Add Student (SQL)")
        print("4. Remove Student (SQL)")
        print("5. Logout")
        choice = int(input("Enter your choice: "))
```

```
if choice == 1:
    student_id = input("Enter student ID: ")
    print("Loading....")
    time.sleep(0.5)
    if student_id not in list(df1["Login_IDs"]):
        print("Student not found")
    else:
        print(f"Update marks of {df1.loc[df1['Login_IDs'] == student_id,
'NAME_OF_THE_STUDENT'].values[0]}")
        gms = float(input("Enter General Management Score (OUT of 50): "))
        dss = float(input("Enter Domain Specific Score (OUT of 50): "))
        remark = input("Enter remark: ")
        total = gms + dss
        df1.loc[df1['Login_IDs'] == student_id,
'GENERAL_MANAGEMENT_SCORE_OUT_of_50'] = gms
        df1.loc[df1['Login_IDs'] == student_id, 'DOMAIN_SPECIFIC_SCORE_OUT_50'] = dss
        df1.loc[df1['Login_IDs'] == student_id, 'TOTAL_SCORE_OUT_of_100'] = total
        df1.loc[df1['Login_IDs'] == student_id, 'REMARKS'] = remark
        df1.to_csv("student_data.csv", index=False)

        df1.to_sql("students", conn, if_exists="replace", index=False)
        print("Marks/Remark updated successfully!")

elif choice == 2:
    print("\n--- Sort Students (SQL) ---")
    print("1. By Name")
    print("2. By Total Score")
    print("3. By General Management Score")
    print("4. By Domain Specific Score")
    sort_choice = int(input("Enter your choice: "))
    if sort_choice == 1:
        query = "SELECT * FROM students ORDER BY `NAME_OF_THE_STUDENT` ASC"
    elif sort_choice == 2:
        query = "SELECT * FROM students ORDER BY `TOTAL_SCORE_OUT_of_100`
DESC"
    elif sort_choice == 3:
        query = "SELECT * FROM students ORDER BY
`GENERAL_MANAGEMENT_SCORE_OUT_of_50` DESC"
    elif sort_choice == 4:
        query = "SELECT * FROM students ORDER BY
`DOMAIN_SPECIFIC_SCORE_OUT_50` DESC"
    else:
        print("Invalid choice!")
        continue

    sorted_df = pd.read_sql(query, conn)
    print(sorted_df[["Login_IDs", "NAME_OF_THE_STUDENT",
"TOTAL_SCORE_OUT_of_100"]])

elif choice == 3:
    print("\n--- Add Student (SQL) ---")
    login_id = input("Enter Login ID: ")

    cur.execute("SELECT * FROM students WHERE Login_IDs = ?", (login_id,))
    existing = cur.fetchone()
```

```
if existing:
    print("A student with this Login ID already exists. Please use a different ID.")
else:
    password = input("Enter Password: ")
    name = input("Enter Name: ")
    gms = float(input("Enter General Management Score (OUT of 50): "))
    dss = float(input("Enter Domain Specific Score (OUT 50): "))
    total = gms + dss
    remark = input("Enter Remark: ")

    cur.execute("""
        INSERT INTO students
        ("Login_IDs","PASSWORDS","NAME_OF_THE_STUDENT",
        "GENERAL_MANAGEMENT_SCORE_OUT_of_50","DOMAIN_SPECIFIC_SCORE_OUT_50",
        "TOTAL_SCORE_OUT_of_100","REMARKS")
        VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (login_id, password, name, gms, dss, total, remark))

    conn.commit()
    print("Student added successfully!")

    df1 = pd.read_sql("SELECT * FROM students", conn)
    df1.to_csv("student_data.csv", index=False)

elif choice == 4:
    print("\n--- Remove Student (SQL) ---")
    login_id = input("Enter Student Login ID to remove: ")

    cur.execute("DELETE FROM students WHERE Login_IDs = ?", (login_id,))
    conn.commit()

    if cur.rowcount == 0:
        print("No student found with that Login ID.")
    else:
        print("Student removed successfully!")
        df1 = pd.read_sql("SELECT * FROM students", conn)
        df1.to_csv("student_data.csv", index=False)

elif choice == 5:
    print("Logging out...\n")
    break

else:
    print("Invalid Choice")

conn.close()
```

## 4. Demo Screenshots

Project code screenshots:

```
1  import pandas as pd
2  import time
3
4  def load_data_students():
5      return pd.read_csv("student_data.csv")
6
7  def load_data_faculty():
8      return pd.read_csv("faculty_data.csv")
9
10 def student_login():
11     df = pd.read_csv("student_data.csv")
12     print("----STUDENT LOGIN----")
13     loginId = input("Enter your loginId: ")
14     password = input("Enter password: ")
15     student = df[(df["Login_IDs"] == loginId) & (df["PASSWORDS"] == password)]
16     student = student.reset_index()
17     if student.empty:
18         print("Invalid Credentials. Try Again!")
19         return
20     print("Loading...")
21     time.sleep(1)
22     print(f"Welcome!! {student.loc[0]["NAME_OF_THE_STUDENT"]}")
23     while True:
24         print("1. Show Marks")
25         print("2. Show Remarks")
26         print("3. Logout")
27         choice = int(input("Enter choice: "))
28         time.sleep(0.5)
29         if choice == 1:
30             print("Loading...")
31             print("1. General Management Score (OUT of 50)")
32             print("2. Domain Specific Score (OUT of 50)")
33             print("3. Total Score and Percentage")
```

```
34     choice1 = int(input("Enter your choice: "))
35     gms = student.loc[0]["GENERAL_MANAGEMENT_SCORE_OUT_of_50"]
36     dss = student.loc[0]["DOMAIN_SPECIFIC_SCORE_OUT_50"]
37     if choice1 == 1:
38         print("Loading....")
39         time.sleep(0.5)
40         print("Your score is:",gms)
41     elif choice1 == 2:
42         print("Loading....")
43         time.sleep(0.5)
44         print("Your score is:",dss)
45     elif choice1 == 3:
46         print("Loading....")
47         time.sleep(0.5)
48         totalScore = gms + dss
49         print("Your Total Score:",totalScore)
50         percentage = (totalScore/100) * 100
51         percentage = round(percentage)
52         print("Percentage obtained:",percentage,"%")
53     else:
54         print("Invalid choice!")
55 elif choice == 2:
56     print("Loading....")
57     time.sleep(0.5)
58     print(student.loc[0]["REMARKS"])
59 elif choice == 3:
60     print("Logging out...")
61     time.sleep(0.5)
62     break
63 else:
64     print("Invalid Choice")
65
```

```
66
67 def faculty_login():|
68     import sqlite3
69     df1 = pd.read_csv("student_data.csv")
70     df2 = pd.read_csv("faculty_data.csv")
71
72     print("----FACULTY LOGIN----")
73     t_id = input("Enter login ID: ")
74     password = input("Enter password: ")
75     faculty = df2[(df2["LoginID"] == t_id) & (df2["password"] == password)]
76     faculty = faculty.reset_index()
77
78     if faculty.empty:
79         print("Invalid credentials. Try again")
80         return
81
82     print("Loading...")
83     time.sleep(1)
84     print(f"Welcome!! {faculty.iloc[0]['Name']}")
85
86     conn = sqlite3.connect("examination.db")
87     cur = conn.cursor()
88
89     df1.columns = df1.columns.str.strip()
90     df1.columns = df1.columns.str.replace(' ', '_')
91     df1.columns = df1.columns.str.replace('[()]', '', regex=True)
92     df1.to_sql("students", conn, if_exists="replace", index=False)
93
94     while True:
95         print("1. Upload/Update Student Marks")
96         print("2. Sort Students (SQL)")
97         print("3. Add Student (SQL)")
98         print("4. Remove Student (SQL)")
```

```
99     print("5. Logout")
100     choice = int(input("Enter your choice: "))
101
102     if choice == 1:
103         student_id = input("Enter student ID: ")
104         print("Loading...")
105         time.sleep(0.5)
106         if student_id not in list(df1["Login_IDs"]):
107             print("Student not found")
108         else:
109             print(f"Update marks of {df1.loc[df1['Login_IDs'] == student_id, 'NAME_OF_THE_STUDENT'].values[0]}")
110             gms = float(input("Enter General Management Score (OUT of 50): "))
111             dss = float(input("Enter Domain Specific Score (OUT of 50): "))
112             remark = input("Enter remark: ")
113             total = gms + dss
114             df1.loc[df1['Login_IDs'] == student_id, 'GENERAL_MANAGEMENT_SCORE_OUT_of_50'] = gms
115             df1.loc[df1['Login_IDs'] == student_id, 'DOMAIN_SPECIFIC_SCORE_OUT_50'] = dss
116             df1.loc[df1['Login_IDs'] == student_id, 'TOTAL_SCORE_OUT_of_100'] = total
117             df1.loc[df1['Login_IDs'] == student_id, 'REMARKS'] = remark
118             df1.to_csv("student_data.csv", index=False)
119
120             df1.to_sql("students", conn, if_exists="replace", index=False)
121             print("Marks/Remark updated successfully!")
122
123     elif choice == 2:
124         print("\n--- Sort Students (SQL) ---")
125         print("1. By Name")
126         print("2. By Total Score")
127         print("3. By General Management Score")
128         print("4. By Domain Specific Score")
129         sort_choice = int(input("Enter your choice: "))
130         if sort_choice == 1:
```

```
131         query = "SELECT * FROM students ORDER BY `NAME_OF_THE_STUDENT` ASC"
132     elif sort_choice == 2:
133         query = "SELECT * FROM students ORDER BY `TOTAL_SCORE_OUT_of_100` DESC"
134     elif sort_choice == 3:
135         query = "SELECT * FROM students ORDER BY `GENERAL_MANAGEMENT_SCORE_OUT_of_50` DESC"
136     elif sort_choice == 4:
137         query = "SELECT * FROM students ORDER BY `DOMAIN_SPECIFIC_SCORE_OUT_50` DESC"
138     else:
139         print("Invalid choice!")
140         continue
141
142     sorted_df = pd.read_sql(query, conn)
143     print(sorted_df[["Login_IDs", "NAME_OF_THE_STUDENT", "TOTAL_SCORE_OUT_of_100"]])
144
145     elif choice == 3:
146         print("\n--- Add Student (SQL) ---")
147         login_id = input("Enter Login ID: ")
148         (variable) login_id: str
149         cur.execute("SELECT * FROM students WHERE Login_IDs = ?", (login_id,))
150         existing = cur.fetchone()
151
152         if existing:
153             print("A student with this Login ID already exists. Please use a different ID.")
154         else:
155             password = input("Enter Password: ")
156             name = input("Enter Name: ")
157             gms = float(input("Enter General Management Score (OUT of 50): "))
158             dss = float(input("Enter Domain Specific Score (OUT 50): "))
159             total = gms + dss
160             remark = input("Enter Remark: ")
161
162             cur.execute("""
```



```
163         INSERT INTO students ("Login_IDs","PASSWORDS","NAME_OF_THE_STUDENT",
164                                "GENERAL_MANAGEMENT_SCORE_OUT_of_50","DOMAIN_SPECIFIC_SCORE_OUT_50",
165                                "TOTAL_SCORE_OUT_of_100","REMARKS")
166         VALUES (?, ?, ?, ?, ?, ?, ?)
167         "", (login_id, password, name, gms, dss, total, remark))
168
169         conn.commit()
170         print("Student added successfully!")
171
172         df1 = pd.read_sql("SELECT * FROM students", conn)
173         df1.to_csv("student_data.csv", index=False)
174
175
176     elif choice == 4:
177         print("\n--- Remove Student (SQL) ---")
178         login_id = input("Enter Student Login ID to remove: ")
179
180         cur.execute("DELETE FROM students WHERE Login_IDs = ?", (login_id,))
181         conn.commit()
182
183         if cur.rowcount == 0:
184             print("No student found with that Login ID.")
185         else:
186             print("Student removed successfully!")
187             df1 = pd.read_sql("SELECT * FROM students", conn)
188             df1.to_csv("student_data.csv", index=False)
189
190
191     elif choice == 5:
192         print("Logging out...\n")
193         break
194
```



```
195         else:
196             print("Invalid Choice")
197
198     conn.close()
199
200
201     def main():
202
203         while True:
204             print("\n--- Examination Portal ---")
205             print("1. Student Login")
206             print("2. Faculty Login")
207             print("3. Exit")
208             choice = input("Enter Choice: ")
209             if choice == '1':
210                 student_login()
211             elif choice == '2':
212                 faculty_login()
213             elif choice == '3':
214                 break
215             else:
216                 print("Invalid Choice!")
217
218
219     main()
220
221
```

**Demo Output(Student Login):**

```
--- Examination Portal ---
1. Student Login
2. Faculty Login
3. Exit
Enter Choice: 1
----STUDENT LOGIN----
Enter your loginId: STFBCM001
Enter password: CamWoo53
Loading....
Welcome!! Camila Wood
1. Show Marks
2. Show Remarks
3. Logout
Enter choice: 1
Loading....
1. General Management Score (OUT of 50)
2. Domain Specific Score (OUT of 50)
3. Total Score and Percentage
Enter your choice: 1
Loading....
Your score is: 44
1. Show Marks
2. Show Remarks
3. Logout
Enter choice: 3
Logging out...

--- Examination Portal ---
1. Student Login
2. Faculty Login
3. Exit
Enter Choice: █
```

**Demo Output(Faculty Login):**

```
--- Examination Portal ---
1. Student Login
2. Faculty Login
3. Exit
Enter Choice: 2
----FACULTY LOGIN----
Enter login ID: Mwhite
Enter password: nrf23yma
Loading....
Welcome!! Mary White
1. Upload/Update Student Marks
2. Sort Students (SQL)
3. Add Student (SQL)
4. Remove Student (SQL)
5. Logout
Enter your choice: 4

--- Remove Student (SQL) ---
Enter Student Login ID to remove: HRVBCM001
Student removed successfully!
1. Upload/Update Student Marks
2. Sort Students (SQL)
3. Add Student (SQL)
4. Remove Student (SQL)
5. Logout
Enter your choice: 5
Logging out...

--- Examination Portal ---
1. Student Login
2. Faculty Login
3. Exit
Enter Choice: █
```

## 5.Important Links

KAGGLE LINK:

<https://www.kaggle.com/datasets/atharvbharaskar/students-test-data>

GITHUB LINK:

[https://github.com/VrushabhParab/Python\\_Project.git](https://github.com/VrushabhParab/Python_Project.git)

## 6. Conclusion

The Examination Portal successfully demonstrates a secure and efficient approach to managing university examination data using Python and Role-Based Access Control (RBAC). By clearly separating permissions between students and faculty, the system ensures data confidentiality, integrity, and accountability. Students can securely access only their own academic information, while faculty members have full administrative capabilities to manage records through intuitive CRUD operations integrated with Pandas and SQL.

The project not only highlights the importance of structured data management but also showcases how combining Python with database operations can simplify complex workflows. The system's modular design, data accuracy, and strong authentication make it a reliable solution that can be further expanded into a fully functional web or GUI-based application in the future.

Overall, the project achieves its objective of creating a scalable, user-friendly, and secure examination management system suitable for real-world educational institutions.