

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology in Information and Techonolgy

By

SARAVANAN S
37120064



DEPARTMENT OF INFORMATION AND TECHNOLOGY
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

AUGUST 2020



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF INFORMATION AND TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SARAVANAN (37120064)** who carried out the project entitled "**COLLEGE MANAGEMENT SYSTEM**" under my supervision from November 2019 to April 2020.

Internal Guide

DR.J.S.VIMALI,M.Tech.,Ph.D.,

Head of the Department

DR.R.SUBASHINI M.E.,Ph.D.

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I SARAVANAN S hereby declare that the Project Report entitled COLLEGE MANAGMENT SYSTEMdone by me under the guidance of Dr.J.S.Vimali,M.Tech.,Ph.d.,at SATHYABAMA INSTIUTE OF SCIENCE AND TECHNOLOGY is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information and Technology.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.R.Subhashini** Heads of the Department of Information and Techonology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.Vimali JS M.Tech.,Phd.,** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information and Technology** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE



Jan 9, 2021

saravanan s

has successfully completed

Introduction to Cybersecurity Tools & Cyber Attacks

an online non-credit course authorized by IBM and offered through Coursera

A handwritten signature in black ink that reads "Coreen Ryskamp".

Coreen Ryskamp
Program Manager
IBM Security Learning Services

**COURSE
CERTIFICATE**



Verify at coursera.org/verify/ZXJESQLGZ5X6

Coursera has confirmed the identity of this individual and their participation in the course.

ABSTRACT

This project deals with the various functioning in College management process. The main idea is to implement a proper process to system .In our existing system contains a many operations registration, student search, fees, attendance, exam records, performance of the student etc. All these activity takeout manually by administrator.

In our model, it deals with the Operations in system. For example, when students fill the registration form the record is stored in the database. And display the details of student is perform by retrieving information from database table.

TABLE OF CONTENTS

CHAPTER.NO	TITLE	PAGE.NO
	ABSTRACT	i
	TABLE OF CONTENTS	ii,iii
	LIST OF FIGURES	iv
	LIST OF TABLES	v
1	INTRODUCTION	2
	1.1 OBJECTIVE	2
	1.2 SCOPE	2
2	THEORITICAL BACKGROUND	3
	2.1PROBLEM IN EXSISTING SYSTEM	3
	2.2SOLUTION TO THESE PROBLEMS	3
	2.3 PROBLEM DEFINITION	4
3	DESIGNING COLLEGE MANAGMENT SYSTEM	5
	3.1 ANALYSIS MODEL	5
	3.2 STUDY OF SYSTEM	7
	3.2.1 GRAPICAL USER INTERFACE	7
	3.3 NUMBER OF MODULES	8
	3.4 SYSTEM ANALYSIS AND DESIGN	9
	3.5 TESTING	9
4	IMPLEMENTATIONOFCOLLEGEMANAGMENTSYS	11
	4.1 PACKAGE ANDMODULES REQUIRED	11
	4.1.1 TKINTER	11
	4.1.2 SQLITE 3 DATABASE FOR PYTHON	15

	SUMMARY AND CONCLUSION	22
	5.1 CONCLUSION	22
5.2	FUTURE ENHANCEMENT	23
	REFERENCE	23
	APPENDEIX	24
	A.CODE	24
	B.SCREENSHOTS	24

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO
3.1	SPIRAL MODEL	7
3.2	DFD DIAGRAM	10
4.1	TKINTER	11

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
4.1	TKINTER EXAMPLE	11
4.2	TKINTER FUNCTION	14
4.3	SQLITE3 FUNCTION	17

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

This is a python oriented application allows us to access the whole information about the college, staffs, students, facilities etc. This application provides a virtual tour of Campus. Here we will get the latest information about the students and staffs. This generic application designed for assisting the students of an institute regarding information on the courses, subjects, classes, assignments, grades and timetable. It also provides support that a faculty can also check about his daily schedule, can upload assignments, and notices to the students. Here administrator will manage the accounts of the student and faculties, makes the timetable, and upload the latest information about the campus.

1.2 SCOPE

- College information: Through this service one can access the complete information about the college campus such as courses available, admission procedure, placements, college events, achievements etc.
- Student tracking: Any company or any organization that want to check the summary about the student of the college, so that they will be able to choose the particular students for their campus placement And for that purpose they will be given a particular link through which they can access the information required.
- Student attendance status: It gives the attendance status of students. Faculty will update the attendance periodically and can be seen by students and parents.
- Student's performance in exams: This facility provides the performance of the student in each exam which is conducted by university or college such as midterm performance. Marks obtained by students in exams will be updated by faculties that can be access by students and parents.
- Exam Notification: This facility notifies students and parents about

examination schedule.

- Events: It will give information about different events that will be conducted by college time to time. Information about these events will be updated by administrator.
- Online assignments: This service provides the facility to faculty to upload assignments and to students to submit these assignments online.
- Information about staff: It will help in maintaining complete information about college faculty members such as their department, cadre, date of joining, salary, etc. Administrator will register new faculties and remove their account when they leave the college.

CHAPTER 2

THEORITICAL BACKGROUND

Today in colleges student details are entered manually. The student details in separate records are tedious task. Referring to all these records and updating is needed. There is a chance for more manual errors.

2.1 Problems in existing system:

- It was limited to a single system.
- It was less user-friendly.
- It have a lots of manual work (Manual system does not mean that we are working with pen and paper, it also include working on spread sheets and other simple software's)
- It requires more no of employees need to work.
- It was time consuming process.
- The present system was very less secure.
- It is unable to generate different kinds of report.

2.2 Solution to these problems:

The development of the new system contains the following activities, which try to automate the entire process keeping in view of the database integration approach.

- User friendliness is provided in the application with various controls.
- The system makes the overall project management much easier and flexible.
- It can be accessed over the Internet.
- Various classes have been used to provide file upload and mail features.

- There is no risk of data mismanagement at any level while the project development is under process.
- It provides high level of security using different protocols like https etc.

2.3 Problem Definition

The problem is to provide the complete information about the college campus. In which the college staff members, students and parents can access the information and will be familiar with college campus. It will provide interactive environment for the staff, students and parents by getting knowledge of student attendance, remarks, exams performances, grades, timetables, notices etc.

CHAPTER 3

DESIGNING COLLEGE MANGMENT SYSTEM

The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

3.1 ANAYALYSIS MODEL

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:
 1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
 2. Defining the requirements of the second prototype.
 3. Planning and designing the second prototype.
 4. Constructing and testing the second prototype.
- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

The following diagram shows how a spiral model acts like:

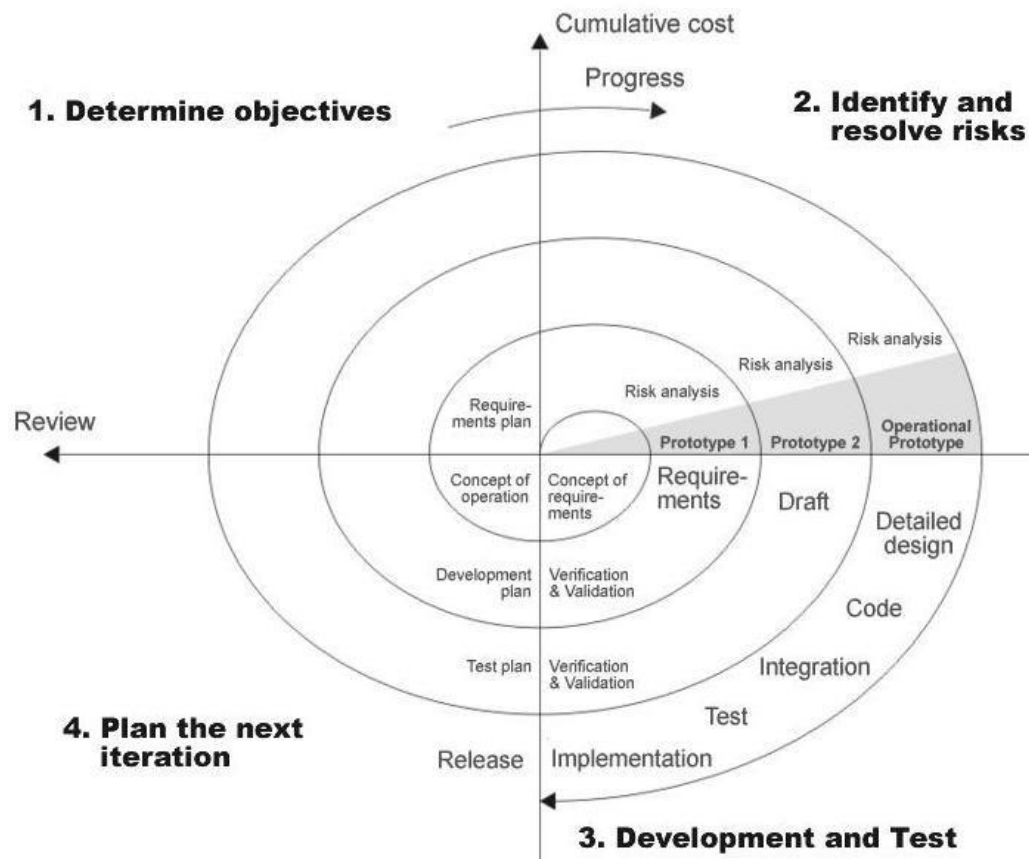


FIG 3.1

3.2 STUDY OF THE SYSTEM:

3.2.1 GRAPICAL USER INTERFACE

In the flexibility of the uses the interface has been developed a graphics concept in mind, associated through a browses interface. The GUI'S at the top level have been categorized as

1. Administrative user interface
2. The operational or generic user interface

The administrative user interface concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. The interfaces help the administrations with all the transactional states like Data insertion, Data deletion and Date updation along with the extensive data search capabilities.

The operational or generic user interface helps the users upon the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information helps the ordinary users in managing their own information in a customized manner as per the assisted flexibilities.

3.3 NUMBER OF MODULES

The system after careful analysis has been identified to be presented with the following modules:

The modules involved are:

1. College information: Through this service one can access the complete information about the college campus such as courses available, admission procedure, placements, college events, achievements etc.
2. Student tracking: Any company or any organization that want to check the summary about the student of the college, so that they will be able to choose the particular students for their campus placement And for that purpose they will be given a particular link through which they can access the information required.
3. Student attendance status: It gives the attendance status of students. Faculty will update the attendance periodically and can be seen by students and parents.
4. Student's performance in exams: This facility provides the performance of the student in each exam which is conducted by university or college such as midterm performance. Marks obtained by students in exams will be updated by faculties that can be access by students and parents.
5. Exam Notification: This facility notifies students and parents about examination schedule.
6. Events: it will give information about different events that will be conducted by college time to time. Information about these events will be updated by administrator.
7. Online assignments: This service provides the facility to faculty to upload assignments and to students to submit these assignments online.
8. Information about staff: It will help in maintaining complete information about

college faculty members such as their department, cadre, date of joining, salary, etc. Administrator will register new faculties and remove their account when they leave the college.

3.4 System Analysis and Design:

In this phase the software's overall structure and its nuances are defined. In terms of client server technology the no of tiers needed for the package architecture, database design, data structure design etc are defined in this phase. Analysis and Design are very crucial in entire development cycle. Any glitch in this phase could be expensive to solve in the later stage of software development. Hence following is the essential approach taken during website designing:

- ❖ DFD
- ❖ Database Designing
- ❖ Form Designing
- ❖ Pseudo code for methods

3.5 Testing:

Once the code is generated, the website testing begins. Different testing methodologies are done to unravel the bugs that were committed during the previous phases. Different testing methodologies are used:

- ❖ Acceptance testing
- ❖ White Box Testing
- ❖ Black Box Testing

Context Level Diagram

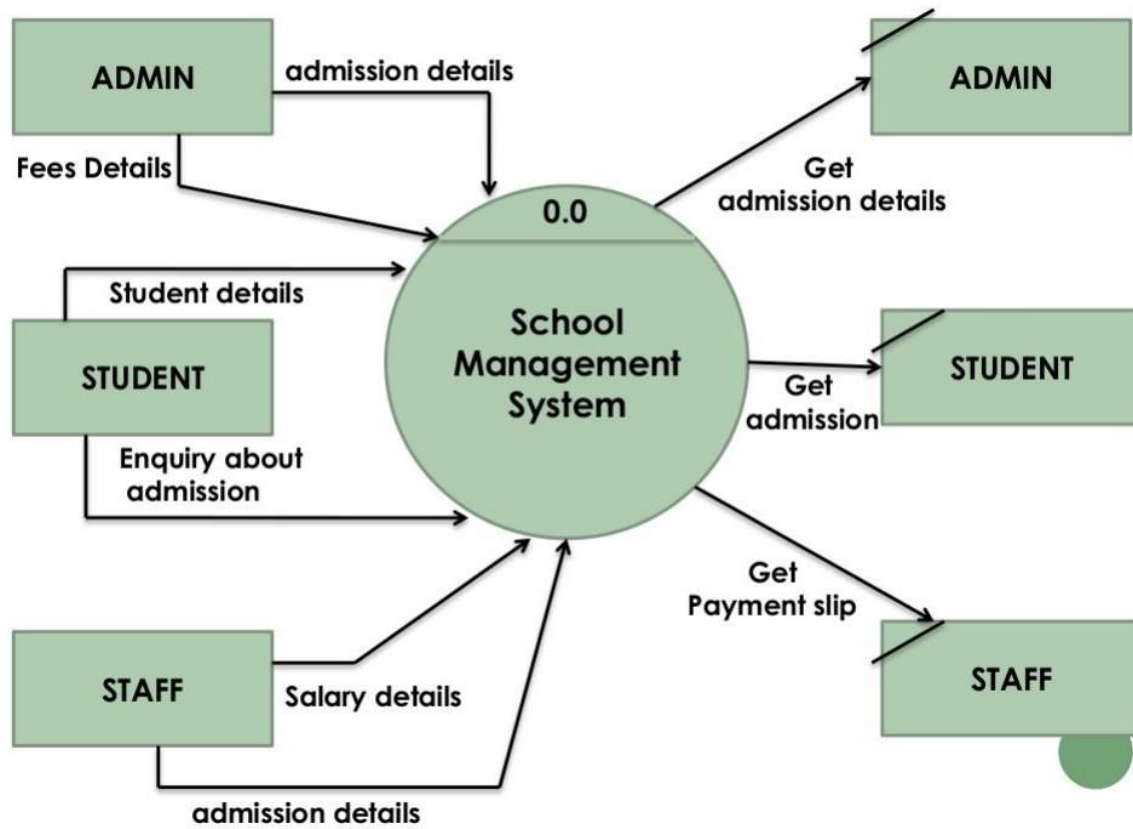


FIG 3.2

CHAPTER 4

IMPLEMENTATION OF THE COLLEGE MANAGMENT SYSTEM

4.1 PACKAGES AND MODULES REQUIRERD

4.1.1 TKINTER

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

1. the tkinter module.
2. Create the GUI application main window.
3. Add one or more of the above-mentioned widgets to the GUI application.
4. Enter the main event loop to take action against each event triggered by the user.

Example

```
#!/usr/bin/python  
  
import Tkinter  
top = Tkinter.Tk()  
# Code to add widgets will go here...  
top.mainloop()
```

TABLE 4.1

This would create a following window –



FIG 4.1

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table -

Sr.No.	Operator & Description
1	Button The Button widget is used to display buttons in your application.
2	Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3	Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame The Frame widget is used as a container widget to organize other widgets.
6	Label The Label widget is used to provide a single-line caption for other widgets. It can also contain images.

7	Listbox The Listbox widget is used to provide a list of options to a user.
8	Menubutton The Menubutton widget is used to display menus in your application.
9	Menu The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	Message The Message widget is used to display multiline text fields for accepting values from a user.
11	Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale The Scale widget is used to provide a slider widget.
13	Scrollbar The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text The Text widget is used to display text in multiple lines.

15	Toplevel The Toplevel widget is used to provide a separate window container.
16	Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17	PanedWindow A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox This module is used to display message boxes in your applications.

TABLE 4.2

4.1.2 SQLITE 3 DATABASE FOR PYTHON

Installation

SQLite3 can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249. You do not need to install this module separately because it is shipped by default along with Python version 2.5.x onwards.

To use sqlite3 module, you must first create a connection object that represents the database and then optionally you can create a cursor object, which will help you in executing all the SQL statements.

Python sqlite3 module APIs

Following are important sqlite3 module routines, which can suffice your requirement to work with SQLite database from your Python program. If you are looking for a more sophisticated application, then you can look into Python sqlite3 module's official documentation.

Sr.No.	API & Description
1	sqlite3.connect(database [,timeout ,other optional arguments]) This API opens a connection to the SQLite database file. You can use ":memory:" to open a database connection to a database that resides in RAM instead of on disk. If database is opened successfully, it returns a connection object. When a database is accessed by multiple connections, and one of the processes modifies the database, the SQLite database is locked until that transaction is committed. The timeout parameter specifies how long the connection should wait for the lock to go away until raising an exception. The default for the timeout parameter is 5.0 (five seconds). If the given database name does not exist then this call will create the database. You can specify filename with the required path as well if you want to create a database anywhere else except in the current directory.

2	connection.cursor([cursorClass]) This routine creates a cursor which will be used throughout of your database programming with Python. This method accepts a single optional parameter cursorClass. If supplied, this must be a custom cursor class that extends sqlite3.Cursor.
3	cursor.execute(sql [, optional parameters]) This routine executes an SQL statement. The SQL statement may be parameterized (i. e. placeholders instead of SQL literals). The sqlite3 module supports two kinds of placeholders: question marks and named placeholders (named style). For example – cursor.execute("insert into people values (?, ?)", (who, age))
4	connection.execute(sql [, optional parameters]) This routine is a shortcut of the above execute method provided by the cursor object and it creates an intermediate cursor object by calling the cursor method, then calls the cursor's execute method with the parameters given.
5	cursor.executemany(sql, seq_of_parameters) This routine executes an SQL command against all parameter sequences or mappings found in the sequence sql.
6	connection.executemany(sql[, parameters]) This routine is a shortcut that creates an intermediate cursor object by calling the cursor method, then calls the cursor.s executemany method with the parameters given.
7	cursor.executescript(sql_script) This routine executes multiple SQL statements at once provided in the form of script. It issues a COMMIT statement first, then executes the SQL script it gets as a parameter. All the SQL statements should be separated by a semi colon (;).
8	connection.executescript(sql_script) This routine is a shortcut that creates an intermediate cursor object by calling the cursor method, then calls the cursor's executescript method with the parameters given.

9	connection.total_changes() This routine returns the total number of database rows that have been modified, inserted, or deleted since the database connection was opened.
10	connection.commit() This method commits the current transaction. If you don't call this method, anything you did since the last call to commit() is not visible from other database connections.
11	connection.rollback() This method rolls back any changes to the database since the last call to commit().
12	connection.close() This method closes the database connection. Note that this does not automatically call commit(). If you just close your database connection without calling commit() first, your changes will be lost!
13	cursor.fetchone() This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.
14	cursor.fetchmany([size = cursor.arraysize]) This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.

TABLE 4.3

Connect To Database

Following Python code shows how to connect to an existing database. If the database does not exist, then it will be created and finally a database object will be returned.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";
```

Here, you can also supply database name as the special name **:memory:** to create a database in RAM. Now, let's run the above program to create our

database **test.db** in the current directory. You can change your path as per your requirement. Keep the above code in `sqlite.py` file and execute it as shown below. If the database is successfully created, then it will display the following message.

```
$chmod +x sqlite.py
$./sqlite.py
Open database successfully
```

Create a Table

Following Python program will be used to create a table in the previously created database.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')print "Opened database successfully";

conn.execute("""CREATE TABLE COMPANY
      (ID INT PRIMARY KEY    NOT NULL,
       NAME      TEXT  NOT NULL,
       AGE      INT   NOT NULL,
       ADDRESS   CHAR(50),
       SALARY     REAL);""")print "Table created successfully";

conn.close()
```

When the above program is executed, it will create the **COMPANY** table in your **test.db** and it will display the following messages –

```
Opened database successfully
Table created successfully
```

INSERT Operation

Following Python program shows how to create records in the **COMPANY** table created in the above example.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
```

```
VALUES (1, 'Paul', 32, 'California', 20000.00 );

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
VALUES (2, 'Allen', 25, 'Texas', 15000.00 );

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );

conn.commit()print "Records created successfully";
conn.close()
```

When the above program is executed, it will create the given records in the COMPANY table and it will display the following two lines –

```
Opened database successfully
Records created successfully
```

SELECT Operation

Following Python program shows how to fetch and display records from the COMPANY table created in the above example.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')print "Opened database successfully";

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")for row
in cursor:
    print "ID = ", row[0]
    print "NAME = ", row[1]
    print "ADDRESS = ", row[2]
    print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
```

```
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

UPDATE Operation

Following Python code shows how to use UPDATE statement to update any record and then fetch and display the updated records from the COMPANY table.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit()print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")for row
in cursor:
    print "ID = ", row[0]
    print "NAME = ", row[1]
    print "ADDRESS = ", row[2]
    print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 25000.0
```

```
ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

DELETE Operation

Following Python code shows how to use DELETE statement to delete any record and then fetch and display the remaining records from the COMPANY table.

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')print "Opened database successfully";

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")for row
in cursor:
    print "ID = ", row[0]
    print "NAME = ", row[1]
    print "ADDRESS = ", row[2]
    print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows deleted : 1
ID = 1
NAME = Paul
ADDRESS = California
```

SALARY = 20000.0

ID = 3

NAME = Teddy

ADDRESS = Norway

SALARY = 20000.0

ID = 4

NAME = Mark

ADDRESS = Rich-Mond

SALARY = 65000.0

CHAPTER 5 SUMMARY AND CONCLUSION

5.1 CONCLUSION

The project entitled as **College Management System** is the system that deals with the issues related to a particular institution.

- This project is successfully implemented with all the features mentioned in system requirements specification.
- The application provides appropriate information to users according to the chosen service.
- The project is designed keeping in view the day to day problems faced by a college.
- Deployment of our application will certainly help the college to reduce unnecessary wastage of time in personally going to each department for some information.

Awareness and right information about any college is essential for both the development of student as well as faculty. So this serves the right purpose in achieving the desired requirements of both the communities.

5.2 Future Enhancement:

- Online examination module would be introduced to conduct online examination.
- Scheduling of the staff. i.e. , time table setting of the staff
- Further, the faculty can upload the videos of their lectures on to this site and students who had missed those classes can view those videos.

REFERENCES

- I <https://www.python.org/>
- II <https://wiki.python.org/moin/TkInter>
- III https://www.tutorialspoint.com/sqlite/sqlite_python.html
- IV <https://docs.python.org/3/library/sqlite3.html>
- V <https://www.tutorialspoint.com/>

APPENDIX

A.CODE
B.SCREEN SHOTS

A.CODE

1 MENU.PY
2 STDINFO.PY
3 FEE.PY

MENU.PY

```
from tkinter import*  
import random  
import os
```

```
def __marksheet__():  
filename = 'Search_Page.py'
```

```

os.system(filename)
os.system('notepad'+filename)

def __Library__():
    filename = 'Library_Frontend.py'
    os.system(filename)
    os.system('notepad'+filename)

def __information__():
    filename = 'Std_info_FrontEnd.py'
    os.system(filename)
    os.system('notepad'+filename)

def __FeeReport__():
    filename = 'Fee_Frontend.py'
    os.system(filename)
    os.system('notepad'+filename)

def menu():
    root = Tk()
    root.title('Menu')
    root.geometry('1350x750')
    root.config(bg = 'navajo white')

    title_Frame = LabelFrame(root, font = ('arial',50,'bold'), width =
1000, height = 100, bg = 'navajo white', relief = 'raise', bd = 13)
    title_Frame.grid(row = 0, column = 0, pady = 50)

    title_Label = Label(title_Frame, text = 'MENU', font =
('arial',30,'bold'), bg = 'navajo white')
    title_Label.grid(row = 0, column = 0, padx = 150)

    #=====
    =====FRAMES=====
    =====

    Frame_1 = LabelFrame(root, font = ('arial',17,'bold'), width =
1000, height = 100, bg = 'navajo white', relief = 'ridge', bd = 10)
    Frame_1.grid(row = 1, column = 0, padx = 280)
    Frame_2 = LabelFrame(root, font = ('arial',17,'bold'), width =
1000, height = 100, bg = 'navajo white', relief = 'ridge', bd = 10)
    Frame_2.grid(row = 2, column = 0, padx = 130, pady = 7)
    Frame_3 = LabelFrame(root, font = ('arial',17,'bold'), width =
1000, height = 100, bg = 'navajo white', relief = 'ridge', bd = 10)
    Frame_3.grid(row = 3, column = 0, pady = 7)
    Frame_4 = LabelFrame(root, font = ('arial',17,'bold'), width =
1000, height = 100, bg = 'navajo white', relief = 'ridge', bd = 10)
    Frame_4.grid(row = 4, column = 0, pady = 7)

```

```

#=====
=====LABELS=====
=====
        Label_1 = Label(Frame_1, text = 'STUDENT PROFILE', font =
('arial',25,'bold'), bg = 'navajo white')
        Label_1.grid(row = 0, column = 0, padx = 50, pady = 5)
        Label_2 = Label(Frame_2, text = 'FEE REPORT', font =
('arial',25,'bold'), bg = 'navajo white')
        Label_2.grid(row = 0, column = 0, padx = 100, pady = 5)
        Label_3 = Label(Frame_3, text = 'LIBRARY SYSTEM', font =
('arial',25,'bold'), bg = 'navajo white')
        Label_3.grid(row = 0, column = 0, padx = 60, pady = 5)
        Label_4 = Label(Frame_4, text = 'MARKSHEET', font =
('arial',25,'bold'), bg = 'navajo white')
        Label_4.grid(row = 0, column = 0, padx = 101, pady = 5)

#=====
=====BUTTONS=====
=====
        Button_1 = Button(Frame_1, text = 'VIEW', font =
('arial',16,'bold'), width = 8, command = __information__)
        Button_1.grid(row = 0, column = 3, padx = 50)
        Button_2 = Button(Frame_2, text = 'VIEW', font =
('arial',16,'bold'), width = 8, command = __FeeReport__)
        Button_2.grid(row = 0, column = 3, padx = 50)
        Button_3 = Button(Frame_3, text = 'VIEW', font =
('arial',16,'bold'), width = 8, command = __Library__)
        Button_3.grid(row = 0, column = 3, padx = 50)
        Button_4 = Button(Frame_4, text = 'VIEW', font =
('arial',16,'bold'), width = 8, command = __marksheet__)
        Button_4.grid(row = 0, column = 3, padx = 50)

root.mainloop()

if __name__ == '__main__':
    menu()

```

STDINFO.PY

```
from tkinter import*
```

```

import tkinter.messagebox
import random
import Std_info_BackEnd
from tkinter import ttk

class Std_info():
    def __init__(self, master):
        self.master = master
        self.master.title('Student Information')
        self.master.geometry('1350x750')
        self.master.config(bg = 'navajowhite')

    def information():
        #=====
        =====Variables=====
        =====
        self.name = StringVar()
        self.fname = StringVar()
        self.mname = StringVar()
        self.address = StringVar()
        self.mobno = StringVar()
        self.email = StringVar()
        self.dob = StringVar()
        self.gender = StringVar()

        #=====
        =====Functions=====
        =====
        def StudentRec(event):
            try:
                global selected_tuple

                index = self.listbox.curselection()[0]
                selected_tuple = self.listbox.get(index)

                self.Entry_name.delete(0, END)
                self.Entry_name.insert(END, selected_tuple[1])
                self.Entry_fname.delete(0, END)
                self.Entry_fname.insert(END, selected_tuple[2])
                self.Entry_mname.delete(0, END)
                self.Entry_mname.insert(END, selected_tuple[3])
                self.Entry_address.delete(0, END)
                self.Entry_address.insert(END, selected_tuple[4])
                self.Entry_mobno.delete(0, END)
                self.Entry_mobno.insert(END, selected_tuple[5])
                self.Entry_emailID.delete(0, END)
                self.Entry_emailID.insert(END, selected_tuple[6])

```

```

        self.Entry_dob.delete(0, END)
        self.Entry_dob.insert(END, selected_tuple[7])
        self.Entry_gender.delete(0, END)
        self.Entry_gender.insert(END, selected_tuple[8])
    except IndexError:
        pass

    def Add():
        if(len(self.name.get()) != 0):
            Std_info_BackEnd.insert(self.name.get(),      self.fname.get(),
self.mname.get(),      self.address.get(),      self.mobno.get(),      self.email.get(),
self.dob.get(), \
            self.gender.get())
            self.listbox.delete(0, END)
            self.listbox.insert(END, (self.name.get(),      self.fname.get(),
self.mname.get(),      self.address.get(),      self.mobno.get(),      self.email.get(),
self.dob.get(), \
            self.gender.get()))

    def Display():
        self.listbox.delete(0, END)
        for row in Std_info_BackEnd.view():
            self.listbox.insert(END, row, str(' '))

    def Exit():
        Exit = tkinter.messagebox.askyesno("Login System", "Confirm
if you want to Exit")
        if Exit > 0:
            self.master.destroy()
            return

    def Reset():
        self.name.set("")
        self.fname.set("")
        self.mname.set("")
        self.address.set("")
        self.mobno.set("")
        self.email.set("")
        self.dob.set("")
        self.gender.set("")
        self.listbox.delete(0, END)

    def Delete():
        if(len(self.name.get()) != 0):

```

```

Std_info_BackEnd.delete(selected_tuple[0])
Reset()
Display()

def Search():
    self.listbox.delete(0, END)
    for row in Std_info_BackEnd.search(self.name.get(),
self.fname.get(), self.mname.get(), self.address.get(), self.mobno.get(),
self.email.get(), self.dob.get(),self.gender.get()):
        self.listbox.insert(END, row, str(' '))

def Update():
    if(len(self.name.get()) != 0):
        Std_info_BackEnd.delete(selected_tuple[0])
    if(len(self.name.get()) != 0):
        Std_info_BackEnd.insert(self.name.get(), self.fname.get(),
self.mname.get(), self.address.get(), self.mobno.get(), self.email.get(),
self.dob.get(), \
        self.gender.get())

        self.listbox.delete(0, END)
        self.listbox.insert(END, (self.name.get(), self.fname.get(),
self.mname.get(), self.address.get(), self.mobno.get(), self.email.get(),
self.dob.get(), \
        self.gender.get()))

#=====
=====Frames=====
=====

self.Main_Frame = LabelFrame(self.master, width = 1300,
height = 500, font = ('arial',20,'bold'), \
    bg = 'navajowhite',bd = 15, relief = 'ridge')
self.Main_Frame.grid(row = 0, column = 0, padx = 10, pady =
20)

self.Frame_1 = LabelFrame(self.Main_Frame, width = 600,
height = 400, font = ('arial',15,'bold'), \
    relief = 'ridge', bd = 10, bg = 'navajowhite', text = 'STUDENT
INFORMATION ')
self.Frame_1.grid(row = 1, column = 0, padx = 10)

self.Frame_2 = LabelFrame(self.Main_Frame, width = 750,
height = 400, font = ('arial',15,'bold'), \
    relief = 'ridge', bd = 10, bg = 'navajowhite', text = 'STUDENT

```

```

DATABASE')
        self.Frame_2.grid(row = 1, column = 1, padx = 5)

        self.Frame_3 = LabelFrame(self.master, width = 1200, height =
100, font = ('arial',10,'bold'), \
        bg = 'navajowhite', relief = 'ridge', bd = 13)
        self.Frame_3.grid(row = 2, column = 0, pady = 10)


        #=====
        =====Labels                                     of
        Frame_1=====
        ==
                self.Label_name = Label(self.Frame_1, text = 'Name', font =
('arial',20,'bold'), bg = 'navajowhite')
                self.Label_name.grid(row = 0, column = 0, sticky = W, padx =
20, pady = 10)
                self.Label_fname = Label(self.Frame_1, text = 'Father Name',
font = ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_fname.grid(row = 1, column = 0, sticky = W, padx =
20)
                self.Label_mname = Label(self.Frame_1, text = 'Mother Name',
font = ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_mname.grid(row = 2, column = 0, sticky = W, padx
= 20)
                self.Label_address = Label(self.Frame_1, text = 'Address', font
= ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_address.grid(row = 3, column = 0, sticky = W, padx
= 20)
                self.Label_mobno  = Label(self.Frame_1, text = 'Mobile
Number', font = ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_mobno.grid(row = 4, column = 0, sticky = W, padx =
20)
                self.Label_emailID = Label(self.Frame_1, text = 'Email ID', font
= ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_emailID.grid(row = 5, column = 0, sticky = W, padx
= 20)
                self.Label_dob = Label(self.Frame_1, text = 'Date of Birth', font
= ('arial',20,'bold'), bg = 'navajowhite')
                self.Label_dob.grid(row = 6, column = 0, sticky = W, padx = 20)
                self.Label_gender = Label(self.Frame_1, text = 'Gender', font =
('arial',20,'bold'), bg = 'navajowhite')
                self.Label_gender.grid(row = 7, column = 0, sticky = W, padx =
20, pady = 10)


        #=====
        =====Entries                                     of

```

```

Frame_1=====
==
        self.Entry_name = Entry(self.Frame_1, font = ('arial',17,'bold'),
textvariable = self.name)
        self.Entry_name.grid(row = 0, column = 1, padx = 10, pady = 5)
        self.Entry_fname = Entry(self.Frame_1, font = ('arial',17,'bold'),
textvariable = self.fname)
        self.Entry_fname.grid(row = 1, column = 1, padx = 10, pady =
5)
        self.Entry_mname = Entry(self.Frame_1, font = ('arial',17,'bold'),
textvariable = self.mname)
        self.Entry_mname.grid(row = 2, column = 1, padx = 10, pady =
5)
        self.Entry_address = Entry(self.Frame_1, font =
('arial',17,'bold'), textvariable = self.address)
        self.Entry_address.grid(row = 3, column = 1, padx = 10, pady =
5)
        self.Entry_mobno = Entry(self.Frame_1, font = ('arial',17,'bold'),
textvariable = self.mobno)
        self.Entry_mobno.grid(row = 4, column = 1, padx = 10, pady =
5)
        self.Entry_emailID = Entry(self.Frame_1, font =
('arial',17,'bold'), textvariable = self.email)
        self.Entry_emailID.grid(row = 5, column = 1, padx = 10, pady =
5)
        self.Entry_dob = Entry(self.Frame_1, font = ('arial',17,'bold'),
textvariable = self.dob)
        self.Entry_dob.grid(row = 6, column = 1, padx = 10, pady = 5)
        self.Entry_gender = ttk.Combobox(self.Frame_1, values = (
', 'Male', 'Female', 'Others'),\
font = ('arial',17,'bold'), textvariable = self.gender, width = 19)
        self.Entry_gender.grid(row = 7, column = 1, padx = 10, pady =
5)

#=====
=====Buttons of
self.Frame_3=====
=====
        self.btnSave = Button(self.Frame_3, text = 'SAVE', font =
('arial',17,'bold'), width = 8, command = Add)
        self.btnSave.grid(row = 0, column = 0, padx = 10, pady = 10)
        self.btnDisplay = Button(self.Frame_3, text = 'DISPLAY', font =
('arial',17,'bold'), width = 8, command = Display)
        self.btnDisplay.grid(row = 0, column = 1, padx = 10, pady = 10)
        self.btnReset = Button(self.Frame_3, text = 'RESET', font =
('arial',17,'bold'), width = 8, command = Reset)

```

```

        self.btnReset.grid(row = 0, column = 2, padx = 10, pady = 10)
        self.btnUpdate = Button(self.Frame_3, text = 'UPDATE', font =
('arial',17,'bold'), width = 8, command = Update)
        self.btnUpdate.grid(row = 0, column = 3, padx = 10, pady = 10)
        self.btnDelete = Button(self.Frame_3, text = 'DELETE', font =
('arial',17,'bold'), width = 8, command = Delete)
        self.btnDelete.grid(row = 0, column = 4, padx = 10, pady = 10)
        self.btnSearch = Button(self.Frame_3, text = 'SEARCH', font =
('arial',17,'bold'), width = 8, command = Search )
        self.btnSearch.grid(row = 0, column = 5, padx = 10, pady = 10)
        self.btnExit = Button(self.Frame_3, text = 'EXIT', font =
('arial',17,'bold'), width = 8, command = Exit)
        self.btnExit.grid(row = 0, column = 6, padx = 10, pady = 10)

```

```

#=====
=List                               Box                               and
self.scrollbar=====
=====

```

```

        self.scrollbar = Scrollbar(self.Frame_2)
        self.scrollbar.grid(row = 0, column = 1, sticky = 'ns')

        self.listbox = Listbox(self.Frame_2, width = 75, height = 20 ,
font = ('arial',12,'bold'))
        self.listbox.bind('<<ListboxSelect>>', StudentRec)
        self.listbox.grid(row = 0, column = 0)
        self.scrollbar.config(command = self.listbox.yview)

        information()

        root = Tk()
        obj = Std_info(root)
        root.mainloop()

```

FEE.PY

```

from tkinter import*
from tkinter import ttk
import tkinter.messagebox
import datetime
import Fee_Backend

```

```

class Fee():
    def __init__(self, master):

```

```

self.master = master
self.master.title('Fee Report')
self.master.geometry('1350x750')
self.master.config(bg='Navajo white')

```

```

#

```

```

=====Variables=====
=====

```

```

self.recpt = StringVar()
self.name = StringVar()
self.admsn = StringVar()
self.date = StringVar()
self.branch = StringVar()
self.sem = StringVar()
self.total = DoubleVar()
self.paid = DoubleVar()
self.due = DoubleVar()

```

```

#

```

```

=====Functions=====
=====

```

```

def Tuple(event):
    try:
        global st
        index = self.list.curselection()[0]
        st = self.list.get(index)

        self.recpt_entry.delete(0, END)
        self.recpt_entry.insert(END, st[1])
        self.name_entry.delete(0, END)
        self.name_entry.insert(END, st[2])
        self.admsn_entry.delete(0, END)
        self.admsn_entry.insert(END, st[3])
        self.Date_entry.delete(0, END)
        self.Date_entry.insert(END, st[4])
        self.branch_entry.delete(0, END)
        self.branch_entry.insert(END, st[5])
        self.sem_entry.delete(0, END)
        self.sem_entry.insert(END, st[6])
        self.total_entry.delete(0, END)
        self.total_entry.insert(END, st[7])
        self.paid_entry.delete(0, END)
        self.paid_entry.insert(END, st[8])
        self.due_entry.delete(0, END)
        self.due_entry.insert(END, st[9])
    except IndexError:
        pass

```

```

def Insert():

```

```

        if (len(self.admsn.get()) != 0):
            Fee_Backend.insert(self.recpt.get(), self.name.get(),
self.admsn.get(), self.date.get(),
                                self.branch.get(),          self.sem.get(),
self.total.get(), self.paid.get(),
                                self.due.get())
            self.list.delete(0, END)
            self.list.insert(END, (self.recpt.get(), self.name.get(),
self.admsn.get(), self.date.get(),
                                self.branch.get(),          self.sem.get(),
self.total.get(), self.paid.get(),
                                self.due.get()))

```

```

def View():
    self.list.delete(0, END)
    for row in Fee_Backend.view():
        self.list.insert(END, row, str(' '))

```

```

def Reset():
    self.recpt.set(' ')
    self.name.set(' ')
    self.admsn.set(' ')
    #self.date.set(' ')
    self.branch.set(' ')
    self.sem.set(' ')
    self.paid.set(' ')
    self.due.set(' ')
    self.Display.delete('1.0', END)
    self.list.delete(0, END)

```

```

def Delete():
    Fee_Backend.delete(st[0])
    Reset()
    View()

```

```

def Receipt():
    self.Display.delete('1.0', END)
    self.Display.insert(END, '\t\tRECEIPT' + '\n\n')
    self.Display.insert(
        END, '\tReceipt No.\t\t:' + self.recpt.get() + '\n')
    self.Display.insert(END, '\tStudent Name :'+
                        self.name.get() + '\n')
    self.Display.insert(END, '\tAdmission No.\t:'+
                        self.admsn.get() + '\n')
    self.Display.insert(
        END, '\tDate\t\t\t\t:' + self.date.get() + '\n')
    self.Display.insert(
        END, '\tBranch\t\t\t\t:' + self.branch.get() + '\n')
    self.Display.insert(

```

```

        END, '\tSemester \t      :' + self.sem.get() + '\n\n')

        x1 = (self.var_1.get())
        x2 = (self.paid.get())
        x3 = (x1 - x2)

        self.Display.insert(END, '\tTotal Amount :' + str(x1) +
'\n')
        self.Display.insert(END, '\tPaid Amount :' + str(x2) +
'\n')
        self.Display.insert(END, '\tBalance\t      :' + str(x3) +
'\n')

        self.due.set(x3)

    def Search():
        self.list.delete(0, END)
        for row in Fee_Backend.search(self.recpt.get(),
self.name.get(), self.admsn.get(), self.date.get(),
                                self.branch.get(),      self.sem.get(),
self.total.get(), self.paid.get(),
                                self.due.get()):
            self.list.insert(END, row, str(' '))

    def Update():
        Fee_Backend.delete(st[0])
        Insert()

    def Exit():
        Exit = tkinter.messagebox.askyesno(
            'Attention', 'Confirm, if you want to Exit')
        if Exit > 0:
            root.destroy()
            return

#
=====Frames=====
=====
        Main_Frame = Frame(self.master, bg='Navajo white')
        Main_Frame.grid()

        Title_Frame = LabelFrame(
            Main_Frame, width=1350, height=100, bg='Navajo
white', relief='ridge', bd=15)
        Title_Frame.pack(side=TOP)

        self.lblTitle = Label(Title_Frame, font=('arial', 40, 'bold'),
text='FEE REPORT',
                                bg='navajowhite', padx=13)

```

```

self.lblTitle.grid(padx=400)

Data_Frame = Frame(Main_Frame, width=1350,
height=350,
                    bg='Navajo white', relief='ridge', bd=15)
Data_Frame.pack(side=TOP, padx=15)

Frame_1 = LabelFrame(Data_Frame, width=850,
height=350, bg='Navajo white', relief='ridge', bd=8,
                    text='Informations', font=('arial', 15, 'bold'))
Frame_1.pack(side=LEFT, padx=10)

Frame_2 = LabelFrame(Data_Frame, width=495,
height=350, bg='Navajo white', relief='ridge', bd=8,
                    text='Fee Receipt', font=('arial', 15, 'bold'))
Frame_2.pack(side=RIGHT, padx=10)

List_Frame = Frame(Main_Frame, width=1350,
height=150,
                    bg='Navajo white', relief='ridge', bd=15)
List_Frame.pack(side=TOP, padx=15)

Button_Frame = Frame(Main_Frame, width=1350,
height=80,
                    bg='Navajo white', relief='ridge', bd=15)
Button_Frame.pack(side=TOP)

#
=====Labels=====
=====
self.recpt_label = Label(Frame_1, text='Receipt No. : ',
font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.recpt_label.grid(row=0, column=0, padx=15, sticky=W)

self.name_label = Label(Frame_1, text='Student Name : ',
font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.name_label.grid(row=1, column=0, padx=15,
sticky=W)

self.admsn_label = Label(Frame_1, text='Admission No. :
', font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.admsn_label.grid(row=2, column=0, padx=15,
sticky=W)

self.Date_label = Label(Frame_1, text='Date : ', font=(
    'arial', 14, 'bold'), bg='Navajo white')

```

```

self.Date_label.grid(row=3, column=0, padx=15, sticky=W)

self.branch_label = Label(Frame_1, text='Branch : ',
font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.branch_label.grid(row=4, column=0, padx=15,
sticky=W)

self.sem_label = Label(Frame_1, text='Semester : ', font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.sem_label.grid(row=5, column=0, padx=15, sticky=W)

self.total_label = Label(Frame_1, text='TOTAL AMOUNT :
', font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.total_label.grid(row=2, column=2, padx=5, sticky=W)

self.paid_label = Label(Frame_1, text='PAID AMOUNT : ',
font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.paid_label.grid(row=3, column=2, padx=5, sticky=W)

self.due_label = Label(Frame_1, text='BALANCE : ',
font=(
    'arial', 14, 'bold'), bg='Navajo white')
self.due_label.grid(row=4, column=2, padx=5, sticky=W)

#
=====Entries=====
=====
self.var_1 = DoubleVar(Frame_1, value='36265')
d1 = datetime.date.today()
self.date.set(d1)

self.recpt_entry = Entry(Frame_1, font=(
    'arial', 14), textvariable=self.recpt)
self.recpt_entry.grid(row=0, column=1, padx=15, pady=5)

self.name_entry = Entry(Frame_1, font=(
    'arial', 14), textvariable=self.name)
self.name_entry.grid(row=1, column=1, padx=15, pady=5)

self.admsn_entry = Entry(Frame_1, font=(
    'arial', 14), textvariable=self.admsn)
self.admsn_entry.grid(row=2, column=1, padx=15,
pady=5)

self.Date_entry = Entry(Frame_1, font=(
    'arial', 14), textvariable=self.date)

```

```

        self.Date_entry.grid(row=3, column=1, padx=15, pady=5)

        self.branch_entry = ttk.Combobox(Frame_1, values=(' ',
'CSE', 'IT', 'ETC/ET&T', 'Mechanical', 'Civil', 'EE', 'EEE'),
                                         font=('arial', 14), width=19,
textvariable=self.branch)
        self.branch_entry.grid(row=4, column=1, padx=15,
pady=5)

        self.sem_entry = ttk.Combobox(Frame_1, values=(' ',
'FIRST', 'SECOND', 'THIRD', 'FOURTH', 'FIFTH', 'SIXTH',
                                         'SEVENTH', 'EIGHTH'),
font=('arial', 14), width=19,
                                         textvariable=self.sem)
        self.sem_entry.grid(row=5, column=1, padx=15, pady=5)

        self.total_entry = Entry(Frame_1, font=(
'arial', 14), width=10, textvariable=self.var_1,
state='readonly')
        self.total_entry.grid(row=2, column=3, padx=8, pady=5)

        self.paid_entry = Entry(Frame_1, font=(
'arial', 14), width=10, textvariable=self.paid)
        self.paid_entry.grid(row=3, column=3, pady=5)

        self.due_entry = Entry(Frame_1, font=(
'arial', 14), width=10, textvariable=self.due)
        self.due_entry.grid(row=4, column=3, pady=7)

        #
        =====Frame_2=====
        =====
        self.Display = Text(Frame_2, width=42, height=12,
                             font=('arial', 14, 'bold'))
        self.Display.grid(row=0, column=0, padx=3)

        #
        =====List box and
scrollbar=====
        sb = Scrollbar(List_Frame)
        sb.grid(row=0, column=1, sticky='ns')

        self.list = Listbox(List_Frame, font=(
'arial', 13, 'bold'), width=140, height=8)
        self.list.bind('<<ListboxSelect>>', Tuple)
        self.list.grid(row=0, column=0)
        sb.config(command=self.list.yview)

        #

```

```

=====Buttons=====
=====

    btnSave = Button(Button_Frame, text='SAVE', font=(
        'arial', 14, 'bold'), width=10, command=Insert)
    btnSave.grid(row=0, column=0, padx=5, pady=5)

    btnDisplay = Button(Button_Frame, text='DISPLAY',
font=(
        'arial', 14, 'bold'), width=10, command=View)
    btnDisplay.grid(row=0, column=1, padx=5, pady=5)

    btnReset = Button(Button_Frame, text='RESET', font=(
        'arial', 14, 'bold'), width=10, command=Reset)
    btnReset.grid(row=0, column=2, padx=5, pady=5)

    btnReset = Button(Button_Frame, text='UPDATE', font=(
        'arial', 14, 'bold'), width=10, command=Update)
    btnReset.grid(row=0, column=3, padx=5, pady=5)

    btnSearch = Button(Button_Frame, text='SEARCH', font=(
        'arial', 14, 'bold'), width=10, command=Search)
    btnSearch.grid(row=0, column=4, padx=5, pady=5)

    btnDelete = Button(Button_Frame, text='DELETE', font=(
        'arial', 14, 'bold'), width=10, command=Delete)
    btnDelete.grid(row=0, column=5, padx=5, pady=5)

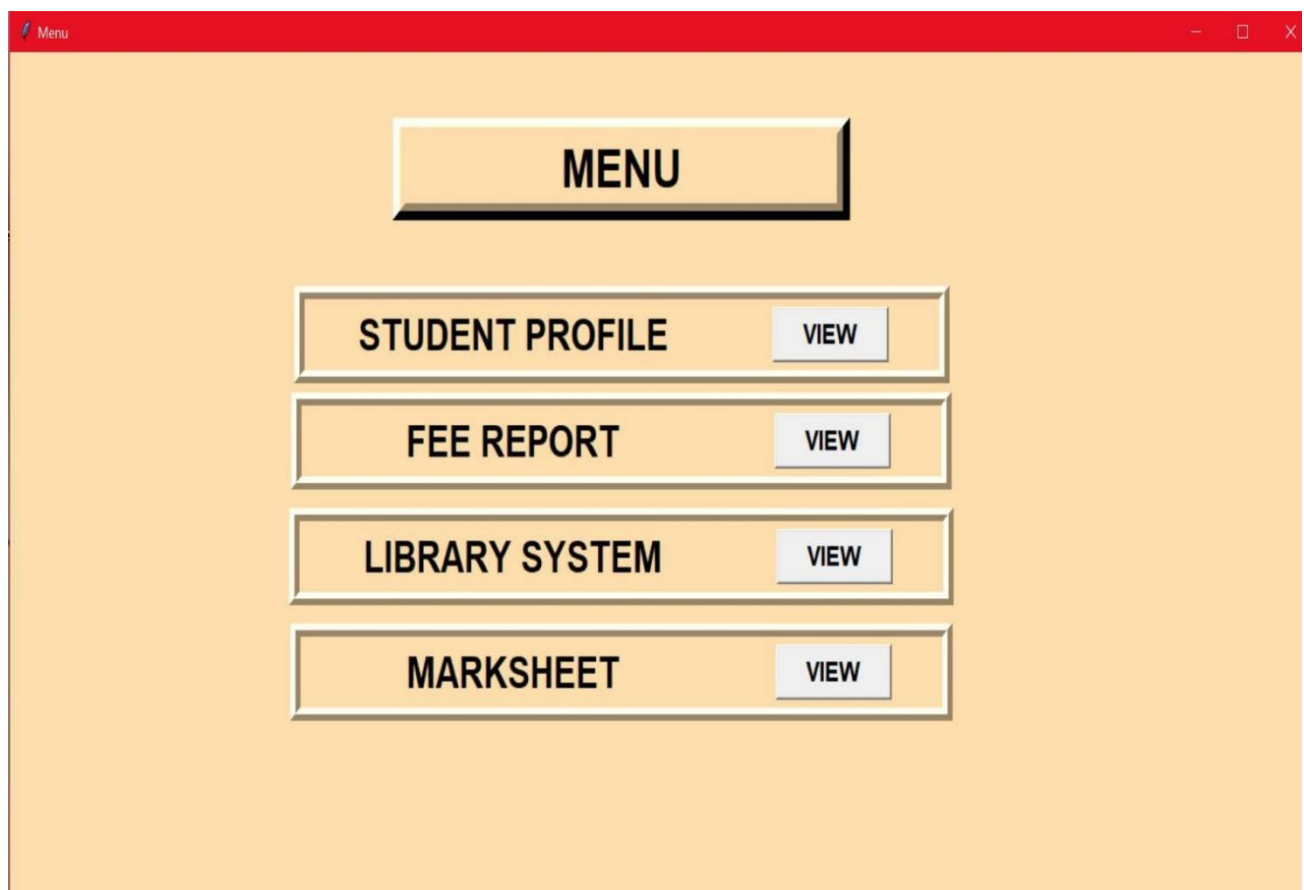
    btnReceipt = Button(Button_Frame, text='RECEIPT',
font=(
        'arial', 14, 'bold'), width=10, command=Receipt)
    btnReceipt.grid(row=0, column=6, padx=5, pady=5)

    btnExit = Button(Button_Frame, text='EXIT', font=(
        'arial', 14, 'bold'), width=10, command=Exit)
    btnExit.grid(row=0, column=7, padx=5, pady=5)

root = Tk()
obj = Fee(root)
root.mainloop()

```

SCREENSHOTS



Student Information

STUDENT INFORMATION

Name

Father Name

Mother Name

Address

Mobile Number

Email ID

Date of Birth

Gender

STUDENT DATABASE

SAVE

DISPLAY

RESET

UPDATE

DELETE

SEARCH

EXIT

43

Search Page

Enter Roll Number

SEARCH CREATE NEW

Library Management System

Library Membership Info:

Member type		Book ID	
Reference No.		Book Title	
First Name		Author	
Surname		Date Borrowed	
Address		Date Due	
Post Code		Days in Loan	
Mobile No.			

Book Details:

- C
- C++
- Java
- Python
- PHP
- Java Script
- My SQL
- Data Structure
- Linux
- Operating System