VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAVI-590014

A DMBS Mini-Project Report
On

"Student Information System"

Submitted in partial fulfillment of the requirements for the 5th semester of
Bachelor of Engineering in Computer Science and Engineering
of Visvesvaraya Technological University, Belgavi

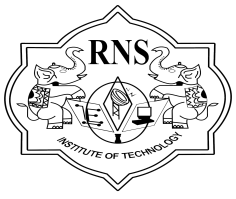
Submitted by:

Yash Vora	1RN16CS123
Vivek Kumar Singh	1RN16CS121

Under the guidance of:

Mr. Karanam Sunil Kumar
Assitant Professor
Dept. of CSE

Mrs.Manjula L
Assitant Professor
Dept. of CSE



Department of Computer Science and Engineering
RNS Institute of Technology
Channasandra, Dr. Vishnuvardhan Road, Bengaluru-560 098

2017-2018

RNS Institute of Technology
Channasandra, Dr. Vishnuvaradana Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the DBMS mini-project work entitled **"Student Information System"** has been successfully carried out by **Yash Vora** bearing USN **1RN16CS123** and **Vivek Kumar Singh** bearing USN **1RN16CS121**, bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **5th semester Bachelor of Engineering** in **Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi, during the academic year 2018-2019. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated. The project report has been approved as it satisfies the mini-project requirements of DBMS lab of 5th semester BE in CSE.

Mr. Karanam Sunil Kumar	**Mrs.Manjula L**	**Dr. G T Raju**
Assitant Professor	**Assitant Professor**	**Prof. & Head**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

External Viva:

Name of the Examiners **Signature with date**

1. **Name1**

2. **Name2**

ABSTRACT

The project "Student Information System" is to be used in a department to maintain the records of the students easily, Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting the relevant information may be very time-consuming. All these problems are solved using this project. The project has been thought about from a student point of view and revolves around the usual requirements for a student studying at an engineering level.

The objective of this project is making an interactive user-friendly website which makes the registration of students, teachers and their modification, deletion in an effective, easy way. Every entity can be easily searched according to their name. We have implemented the idea by making a web application that includes table visualization. Any staff using the website will be able to insert all the details, update and delete them when needed. The UI has been made very simple to provide ease of access for all types of users.

This project requires HTML, CSS in the frontend, Python for backend and database connectivity and SQLite for database management. We seek to expand the project by having a fully real-life model of the department in the college.

ACKNOWLEDGEMENTS

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all. We would like to thank **Dr. H N Shivashankar**, Director, RNSIT, Bangalore, for his moral support towards completing our project. We are grateful to **Dr. M K Venkatesha**, Principal, RNSIT, Bangalore, for his support towards completing this mini project. We would like to thank **Dr. G T Raju**, Dean of Engg., Prof. and Head, Department of Computer Science and Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice. We deeply express my sincere gratitude to my guide **Mr. Karanam Sunil Kumar** and **Mrs. Manjula L.**, Asst Prof, Department of CSE, RNSIT, Bangalore, for their able guidance, regular source of encouragement and assistance throughout this project. We would like to thank all the teaching and non-teaching staff of department of Computer Science and Engineering, RNSIT, Bengaluru for their constant support and encouragement.

Date:
Place:

Yash Vora 1RN16CS123
Vivek Kumar Singh 1RN16CS121

Contents

List of Figures

Chapter 1

Introduction To Database Management System

1.1 Introduction

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science. The word database is so commonly used that we must begin by defining what a database is. Our initial definition is quite general. A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database. The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term database is usually more restricted. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications. Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database

definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data. Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS. Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. Sharing a database allows multiple users and programs to access the database simultaneously.

1.2 History of DBMS

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.[4] Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Also in 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. During 1970s, the first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image. In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology. 3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modelling software.

1.3 Characteristics of DBMS

A number of characteristics distinguish the database approach from the much older approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. For example, one user, the grade reporting office, may keep files on students and their grades. Programs to print a student's transcript and to enter new grades are implemented as part of the application. A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files— and programs to manipulate these files—because each requires some data not avail able from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data. In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications. The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system.
- Insulation between programs and data, and data abstraction.
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing.

1.3.1 Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data, and it describes the structure of the primary database. The catalog is used by the DBMS software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application. Therefore, it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog. In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. For example, an application program written in C++ may have structure or class declarations, and a COBOL program has data division statements to define its files. Whereas file-processing software can access only specific databases,

DBMS software can access diverse databases by extracting the database definitions from the catalog and using these definitions.

1.3.2 Insulation between Programs and Data, and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence. In some types of database systems, such as object-oriented and object-relational systems users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence. The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

1.3.3 Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

1.3.4 Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called online transaction processing (OLTP) applications. A

fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently. The concept of a transaction has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. The DBMS must enforce several transaction properties. The isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The atomicity property ensures that either all the database operations in a transaction are executed or none are.

1.4 Applications of DBMS

Applications where we use Database Management Systems are:

- **Telecom:** There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.
- **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Education Sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online Shopping:** You must be aware of the online shopping websites such as Amazon, Flip kart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

Chapter 2

Requirement Analysis

2.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.
Processor : Intel Atom® processor or Intel® Core™ i3 processor
Processor Speed : 2.4 GHz
RAM : 1 GB
Storage Space : 40 GB
Monitor Resolution : 1024*768 or 1336*768 or 1280*1024

2.2 Software Requirements

Operating System - Windows 7 or later
Text Editor - Atom
SQL - sqlite3
Language - Python3.7.x
Browser - Any one which supports HTML and JavaScript
Additional tool - DB Browser

2.3 Functional Requirements

2.3.1 Major Entities

STUDENT: To store or modify the student details in student entity or table.
TEACHER: We can store or modify the teacher details in student entity or table.
ATTENDANCE: To store the attendance details of the student.
MARKSHEET: To store the marks scored by the students in internal examinations and calculation of the average marks.
COURSES: To store the details of the courses given by the university and the teacher who teach them.
SUBJECT: To store and update the subjects to be taken by the students in the institutuion.
CO CURRICULAR: To store the co curricular activities that the student participates in.

2.3.2 End User Requirements

1. Main Goals:
    - Our motto is to develop a software program for managing the entire college process related to student and teacher accounts and to keep each every track about their data like marksheet and classes and their various transaction processes efficiently.
    - Hereby, our main objective is the ease of accces of important data considering how fast the data exchange takes place.
2. Ease of access:
    - The details can be easily added, deleted or updated without any hassle.
    - Our software will perform and fulfill all the tasks that any administrator would desire.
3. Saving Lookup Time:
    - The person looking for the information doesn't need to go through the whole database to do small operation.

Chapter 3

Database Design

3.1 Entities, Attributes and Relationships

The database, called data, will have seven tables, teacher, student, class, subjects, courses, marksheet and branch. Each will hold information about either the student or teacher. The two tables will be linked through a foreign key. The student table has the following fields:

Figure 3.1: student table

Field	Description
student_id	unique id of the student
student_name	stores student name
date_of_birth	stores date of birth of the student
academic_year	stores the academic year the student is in
branch_code	stores the student's course branch code
semester	current semester of the student

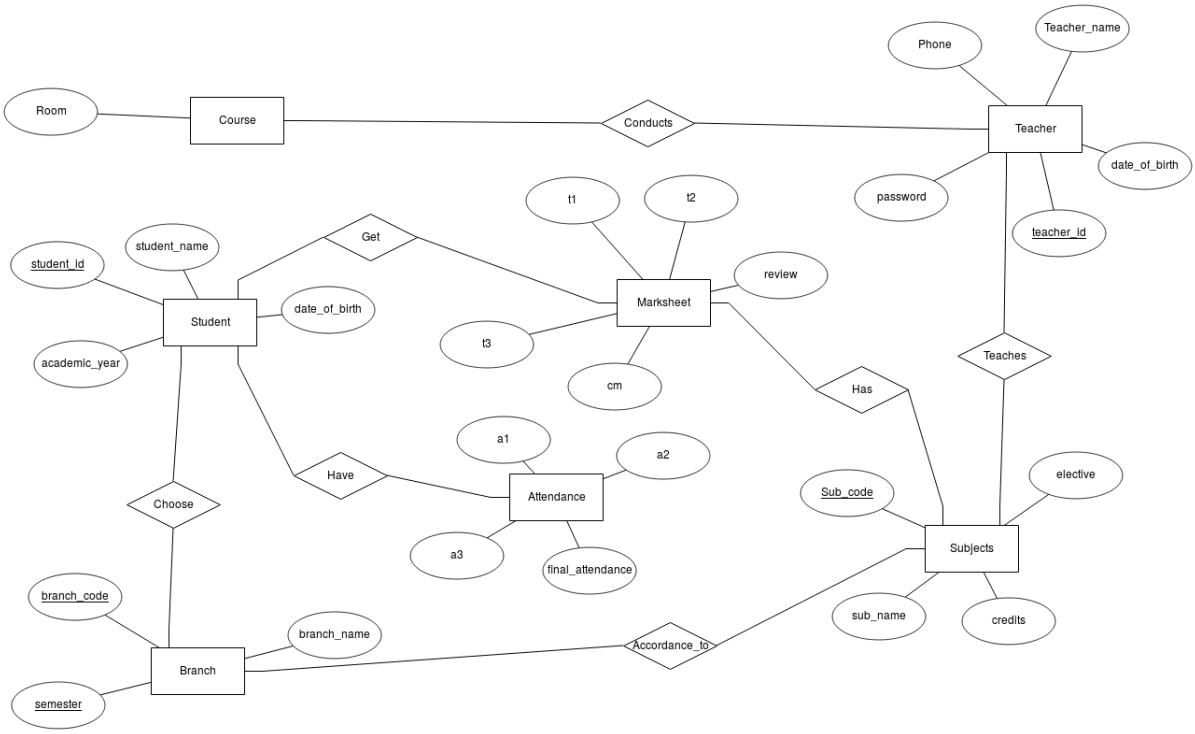
Since one teacher can teach many students, We thought it only right to insert a foreign key branch_code and semester into the student table. In addition, we have made the teacher account to have admin privilege. This will enable us to focus more on the programming than on particulars of the database.

3.2 Identify Major entities, attributes and relationships

- Login page to give access to priviledged teachers.
- Adding student and teacher details by the teacher.
- Changing the id and password of the teacher. which stores the details of transaction.
- Teachers can check all details butnot passwords and modify them.
- Teacher can delete the entities in the tables.
- Easy search facilities to get the reqiured information.

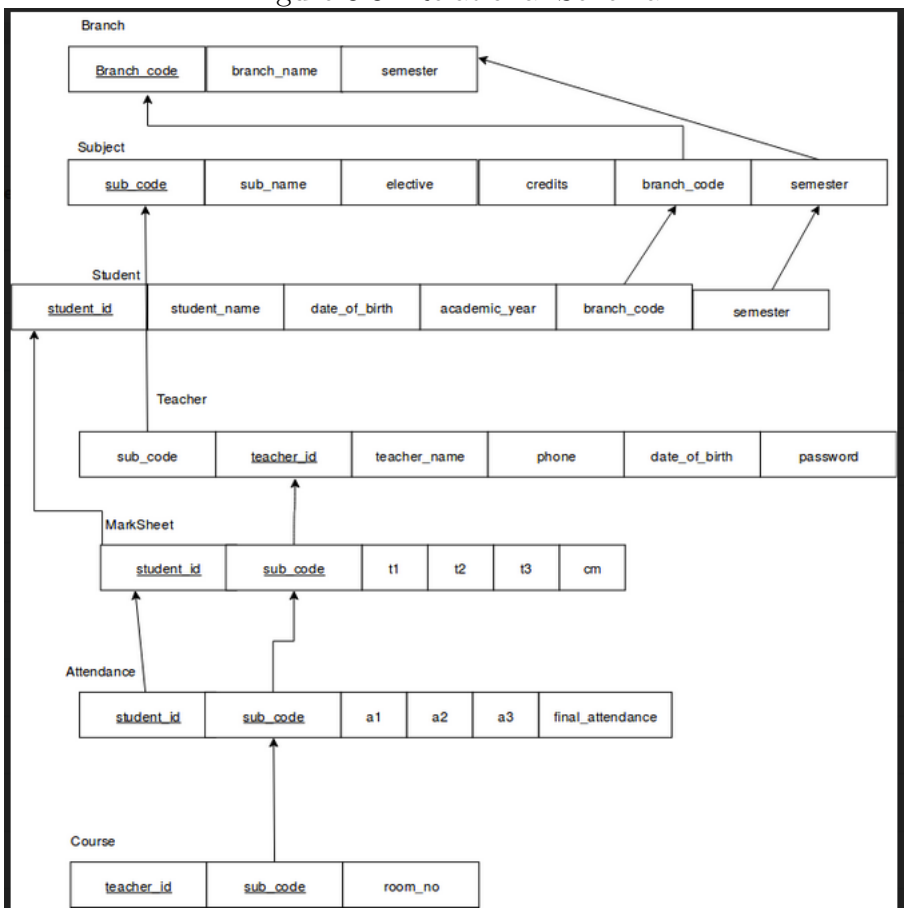
3.3 ER Schema

Figure 3.2: ERD



3.4 Schema Diagram

Figure 3.3: Relational Schema



Chapter 4

Description Of Tools And Technologies

4.1 HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript it forms a triad of cornerstone technologies for the World Wide Web. Web browsers receive HTML documents from a web server or from local storage and render them into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document. HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms, may be embedded into the rendered page. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as ¡img /¿ and ¡input /¿ introduce content into the page directly. Others such as ¡p¿...¡/p¿ surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page. HTML can embed programs written in a scripting language such as JavaScript which affect the behavior and content of web pages. Inclusion of CSS defines the look and layout of content.

4.2 Bootstrap - A CSS Framework

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

CSS is designed primarily to enable the separation of presentation and content, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant

CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Bootstrap is a free and open-source front-end web framework used for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.

Bootstrap is the second most-starred project on GitHub, with more than 111,600 stars and 51,500 forks.

4.3 Javascript

JavaScript, often abbreviated as JS, is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and this is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design; JavaScript was influenced by programming languages such as Self and Scheme.

4.4 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community after 30 years.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based develop-

ment model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

4.5 SQLite

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others.SQLite has bindings to many programming languages. Unlike client–server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. The library can also be called dynamically. The application program uses SQLite's functionality through simple function calls, which reduce latency in database access: function calls within a single process are more efficient than inter-process communication. SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing. SQLite read operations can be multitasked, though writes can only be performed sequentially. SQLite implements most of the SQL-92 standard for SQL but it lacks some features. For example, it partially provides triggers, and it can't write to views (however it provides INSTEAD OF triggers that provide this functionality). While it provides complex queries, it still has limited ALTER TABLE function, as it can't modify or delete columns.SQLite uses an unusual type system for an SQL-compatible DBMS; instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values; in language terms it is dynamically typed. Moreover, it is weakly typed in some of the same ways that Perl is: one can insert a string into an integercolumn (although SQLite will try to convert the string to an integer first, if the column's preferred type is integer). This adds flexibility to columns, especially when bound to a dynamically typed scripting language. However, the technique is not portable to other SQL products. The SQLite web site describes a "strict affinity" mode, but this feature has not yet been added.[11] However, it can be implemented with constraints like CHECK(typeof(x)='integer').

Chapter 5

Flask-SQLite Database Connectivity

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program. Flask is commonly used with MongoDB, which gives it more control over databases and history.

Applications that use the Flask framework include Pinterest, LinkedIn, and the community web page for Flask itself.

5.1 Database Schema

In Flask we need only a single table for the application and it will support SQLite. All we need to do is put the contents of the .sql file in the same folder where program.py file id there.

5.2 5 Steps to connect to the database in flask

There are 5 steps to connect any flask application with the database in sqlite. They are as follows:

- Import the dependencies
- Open a connection to an SQLite database file
- Load the data from the dataframe into the database
- Make functions to execute different SQL queries
- Closing connection

Chapter 6

Implementation

6.1 FlaskApp - Python

```
from flask import Flask, url_for, render_template, g, request, \
    redirect, flash, session, abort
from flask_login import LoginManager
from wtforms import Form, TextField, TextAreaField, validators,
\
    StringField, SubmitField
import os
import sqlite3
from urllib.parse import unquote
from jinja2 import Template

app = Flask(__name__)
DATABASE = "data.db"
# Config
app.config.from_object(__name__)

ID = None

def average(t1, t2, t3):
    return int((int(t1) + int(t2) + int(t3))//3)

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
        db.create_function('avg', 3, average)
        db.row_factory = sqlite3.Row
    return db


def query_db(query: object, args: object, one: object = False) -> object:
    cur = get_db().execute(query, args)
```

```
    rv = cur.fetchall()
    cur.close()
    return (rv[0] if rv else None) if one else rv


def change_db(query, args=()):
    cur = get_db().execute(query, args)
    get_db().commit()
    cur.close()

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()


@app.route("/")
def index():
    if not session.get('logged_in'):
        return login()
    else:
        return render_template("index.html", id=ID)


@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template("login.html", error=False)
    if request.method == 'POST':
        try:
            password = request.form['password']
            teacher_id = int(request.form['teacher_id'])
            print(password, teacher_id)
        except ValueError:
            return render_template("login.html", error=True)

        teacher = query_db("SELECT * FROM Teacher \
                                WHERE teacher_id = ?",
                           [teacher_id], one=True)
        if teacher is None:
            return render_template("login.html", error=True)
            # TODO
        if teacher["password"] == password:
            session['logged_in'] = True
            global ID
            ID = teacher_id
```

```
            return index()
        else:
            return render_template("login.html", error=True)


@app.route('/logout')
def logout():
    session.pop('logged_in', None)
    flash('You were logged out')
    return redirect(url_for('index'))


@app.route('/view', methods=['GET', 'POST'])
def view():
    global ID
    teacher = query_db("SELECT * FROM Teacher \
                        WHERE teacher_id=?", [ID], one=True)
    student_list = query_db("SELECT * FROM Student \
                            WHERE semester IN (SELECT semester FROM Subjec
                            Teacher WHERE Teacher.sub_code=Subject.sub_coo
    if request.method == 'GET':
        if not session.get('logged_in'):
            return login()
        else:
            return render_template("user.html", id=ID, teacher=teacher, err
    if request.method == 'POST':
        old = request.form['oldPassword']
        new = request.form['newPassword']

        if teacher['password'] == old:
            change_db(
                "UPDATE Teacher SET password = ? WHERE teacher_id = ?", (ne
            return render_template("user.html", id=ID,
                                   teacher=teacher, changed=True,
student_list=student_list)
        else:
            return render_template("user.html", id=ID,
                                   teacher=teacher, error=True,
student_list=student_list)


@app.route('/teachers')
def teachers():
    global ID
    print(ID)
    teacher_list = query_db("SELECT * FROM TEACHER", [])
    return render_template("/teachers_info.html", id=ID,
```

```
                            teacher_list=teacher_list)


@app.route('/students')
def students():
    global ID
    student_list = query_db("SELECT * FROM Student", [])
    return render_template("/students_info.html", id=ID,
                            student_list=student_list)


@app.route('/modify/<string:entity>/<string:uid>/<string:uid2>',
           methods=['GET', 'POST'])
def modify(uid, uid2, entity):
    global ID
    teacher_list = query_db("SELECT * FROM TEACHER", [])
    if entity == "teacher":
        values = query_db("SELECT * FROM Teacher \
                           WHERE teacher_id=?", [uid], one=True)
        if request.method == 'GET':
            return render_template("modify.html", id=ID,
                                   entity="Teacher",  identity=values)
        if request.method == 'POST':
            data = request.form.to_dict()
            dic = [data['teacher_id'], data['sub_code'],
                   data['teacher_name'], data['phone'], uid]
            change_db(
                "UPDATE Teacher SET teacher_id=?, sub_code=?,  "
                "teacher_name=?, phone=? WHERE teacher_id=?", dic)
            return logout()
    if entity == "student":
        values = query_db("SELECT * FROM Student \
                           WHERE student_id=?", [uid], one=True)
        print(values['student_name'])
        if request.method == 'GET':
            return render_template("modify.html", id=ID,
                                   entity="Student", identity=values)
        if request.method == 'POST':
            data = request.form.to_dict()
            print(data.keys())
            dic = [data['student_id'], data['student_name'],
                   data['academic_year'], data['branch_code'], uid]
            change_db(
                "UPDATE Student SET student_id=?, student_name=?,
"
                "academic_year=?, branch_code=? WHERE student_id=?", dic)
            return logout()
```

```
    if entity == "attendance":
        values = query_db("SELECT * FROM Attendance \
                           WHERE sub_code=? AND student_id=?", [uid2, uid
        if request.method == 'GET':
            return render_template("modify.html", id=ID,
                                   entity="Attendance", identity=values)
        if request.method == 'POST':
            data = request.form.to_dict()
            print(data.keys())
            #TODO
            # avg = int((int(data['a1']) + int(data['a2']) + int(data['a3'
            dic = [data['sub_code'], data['student_id'],
                   data['a1'], data['a2'], data['a3'], data['a1'], data['a
            #STORED_PROCEDURE_DEMO
            change_db(
                "UPDATE Attendance SET sub_code=?, student_id=?, "
                "a1=?, a2=?, a3=?, final_attendance=avg(?, ?, ?)
WHERE sub_code=? AND student_id=?", dic)
            return redirect(url_for("attendance"))
    if entity == "marksheet":
        values = query_db("SELECT * FROM Marksheet \
                           WHERE sub_code=? AND student_id=?",
                          [uid2, uid], one=True)
        if request.method == 'GET':
            return render_template("modify.html", id=ID,
                                   entity="Marksheet", identity=values)
        if request.method == 'POST':
            data = request.form.to_dict()
            print(data.keys())
            #TODO
            # avg = int((int(data['a1']) + int(data['a2']) + int(data['a3'
            dic = [data['sub_code'], data['student_id'],
                   data['t1'], data['t2'], data['t3'],
                  data['t1'], data['t2'], data['t3'], uid2, uid]
            #STORED_PROCEDURE_DEMO
            change_db(
                "UPDATE Marksheet SET sub_code=?, student_id=?,
"
                "t1=?, t2=?, t3=?, cm=avg(?, ?, ?) WHERE sub_code=? AND stu
            return redirect(url_for("marksheet"))
    if entity == "courses":
        values = query_db("SELECT * FROM Courses \
                           WHERE teacher_id=? AND sub_code=?",
                          [uid, uid2], one=True)
        print(values.keys())
        if request.method == 'GET':
            return render_template("modify.html", id=ID,
```

```
                                        entity="Courses", identity=values)
        if request.method == 'POST':
            data = request.form.to_dict()
            print(data.keys())
            dic = [data['teacher_id'], data['sub_code'],
                   data['Room'], uid, uid2]
            change_db(
                "UPDATE Courses SET teacher_id=?, sub_code=?, "
                "Room=? WHERE teacher_id=? AND sub_code=?", dic)
            return redirect(url_for("courses"))


@app.route("/marksheet")
def marksheet():
    global ID
    if not session.get('logged_in'):
        return login()
    else:
        entry_list = query_db("SELECT * FROM Marksheet", [])
        return render_template("marksheet.html", id=ID,
                               entry_list=entry_list)

@app.route("/attendance")
def attendance():
    global ID
    if not session.get('logged_in'):
        return login()
    else:
        entry_list = query_db("SELECT * FROM Attendance", [])
        return render_template("attendance.html", id=ID,
                               entry_list=entry_list)

@app.route("/courses")
def courses():
    global ID
    if not session.get('logged_in'):
        return login()
    else:
        entry_list = query_db("SELECT * FROM Courses", [])
        return render_template("courses.html", id=ID,
                               entry_list=entry_list)

@app.route("/branch")
def branch():
    global ID
    if not session.get('logged_in'):
        return login()
```

```
    else:
        entry_list = query_db("SELECT * FROM Branch", [])
        return render_template("branch.html", id=ID,
                               entry_list=entry_list)

@app.route("/semester")
def semester():
    global ID
    if not session.get('logged_in'):
        return login()
    else:
        semester_list = []
        for i in range(8):
            semester_list.append(query_db("SELECT * FROM Subject
"
                                          "WHERE semester=?", [i + 1]))
        return render_template("semester.html", id=ID,
                               semester_list=semester_list)

@app.route('/delete/<string:entity>/<string:uid>', methods=['GET', 'POST'])
def delete(uid, entity):
    global ID
    teacher = query_db("SELECT * FROM Teacher \
                        WHERE teacher_id=?", [ID], one=True)
    if entity == "teacher":
        values = query_db("SELECT * FROM Teacher \
                           WHERE teacher_id=?", [uid], one=True)
        if request.method == 'GET':
            return render_template("delete.html", id=ID,
                                   entity="Teacher", identity=values)
        if request.method == 'POST':
            change_db("DELETE FROM Teacher WHERE teacher_id=?",
                      [uid])
            return logout()
    if entity == "student":
        values = query_db("SELECT * FROM Student \
                           WHERE student_id=?", [uid], one=True)
        if request.method == 'GET':
            return render_template("delete.html", id=ID,
                                   entity="Student", identity=values)
        if request.method == 'POST':
            change_db("DELETE FROM Student WHERE student_id=?", [uid])
            return logout()

@app.route("/add/<string:entity>", methods=['GET', 'POST'])
def add(entity):
    global ID
```

```
    if not session.get('logged_in'):
        return login()
    else:
        if entity == "teacher":
            if request.method == 'GET':
                return render_template("add.html", id=ID, entity="Teacher")
            if request.method == 'POST':
                data = request.form.to_dict()
                dic = [data['teacher_id'], data['sub_code'],
                       data['teacher_name'], data['phone'],
                       data['date_of_birth'], data['password']]
                print(dic)
                change_db("INSERT INTO Teacher VALUES (?, ?, ?, ?, ?, ?)",
                return logout()
        if entity == "student":
            if request.method == 'GET':
                return render_template("add.html", id=ID, entity="Student")
            if request.method == 'POST':
                data = request.form.to_dict()
                dic = [data['student_id'], data['student_name'],
                       data['date_of_birth'], data[
                           'academic_year'], data['branch_code'],
                       data['semester']]
                change_db("INSERT INTO Student VALUES (?, ?, ?, ?, ?, ?)",
                return logout()
        if entity == "courses":
            if request.method == 'GET':
                return render_template("add.html", id=ID,
                                       entity="Courses")
            if request.method == 'POST':
                data = request.form.to_dict()
                dic = [data['teacher_id'], data['sub_code'],
                       data['Room']]
                change_db("INSERT INTO Courses VALUES (?, ?, ?", dic)
                return logout()



if __name__ == '__main__':
    app.secret_key = os.urandom(12)

    app.run(host="0.0.0.0", port=5000, debug=False)
```

Chapter 7

Snapshots

Chapter 8

Future Enhancements

ust like any other developer this project is the most basic website built using simple tools. We seek to increase the dynamic of the project by adding various other innovations to it. Such innovations would seem possible only with time, which we lack but regardless we strive to complete what we started.

We believe that apart from present functionalities we can add :

"HOBBIES" which will store the hobbies and extra cocurricular activities of the students.

"CONCURRENT PURCHASE" feature through which two teachers can log in and make changes at the same time.

"STUDENT LOGIN" so that each student may give his/her input on each teacher and subject. This can help in giving appreciation to teacher for their hard work and the areas in which they can improve.

Apart from these changes we are open to various suggestions and hope to implement them soon so that this website can be used by the students for their needs.

Chapter 9

Conclusion

The STUDENT INFORMATION SYSTEM has been designed to maintain the records of all the aspects of students with respect to college life.

It project was designed to model the working of an educational institution.It also contains the details about the teachers that work in the college, the courses and subject information are also saved in the database.The database system project includes triggers that allows the user to get the average of attendance and marks every time those values are modified. It also includes a stored procedure to calculate the total average in the attendance table.

The project provides simple retrieval techniques and easy updating and deletion operations thus helping in efficient maintenance of records.

References

[1] Fundamentals of database systems by (Elmasri Navathe, 2000)
Article: Student Information System: https://en.wikipedia.org/wiki/Student_information_system

[2] Learning SQLite : https://www.tutorialspoint.com/sqlite/
Python and Flask tutorials : http://flask.pocoo.org/docs/0.12/, http://www.youtube.com

[3] Python : https://docs.python.org/3/

[4] ER Diagram and Schema : https://erdplus.com

[5] Stack Overflow: https://stackoverflow.com/