**GRADUATE THESIS MANAGEMENT SYSTEM (GTS)**

A web-based application built with Python Flask and MS SQL Server to manage graduate theses efficiently.

YAREN ÇETİNKAYA

TABLE OF CONTENTS

[Table of Contents 1](#_Toc155562315)

[1. Introduction 3](#_Toc155562316)

[2. ER 3](#_Toc155562317)

[3. Database Structure and Associations 4](#_Toc155562318)

[3.1. Thesis Table 4](#_Toc155562319)

[3.2. Universities Table 5](#_Toc155562320)

[3.3. Institutes Table 5](#_Toc155562321)

[3.4. Persons Table 5](#_Toc155562322)

[3.5. Supervisors Tables 6](#_Toc155562323)

[3.6. Subject Topics Table 6](#_Toc155562324)

[3.7. Keywords Table 6](#_Toc155562326)

[3.8. Thesis Keywords Table 7](#_Toc155562327)

[4. Rules and Integrity Constraints 7](#_Toc155562328)

[4.1. Thesis Submission Date Trigger 7](#_Toc155562327)

[4.2. Thesis Language Trigger 8](#_Toc155562327)

[5. Filling Tables 8](#_Toc155562331)

[6. Query 9](#_Toc155562333)

[7. Graduate Thesis Management System (GTS) Web Application: Design and Development 10](#_Toc155562337)

[7.1. Overview of the Application 10](#_Toc155562338)

[7.2. Programs Used 10](#_Toc155562339)

[7.3. Structure of App 11](#_Toc155562343)

[7.3.1. Routes 11](#_Toc155562344)

[7.3.2. Templates 12](#_Toc155562345)

[7.3.3. Database Connection to Flask 12](#_Toc155562346)

[7.4. Application Operation’s 13](#_Toc155562347)

[7.4.1. Adding Thesis 13](#_Toc155562348)

[7.4.2. Adding Persons 15](#_Toc155562349)

[7.4.3. Adding Universities&Institutes 15](#_Toc155562350)

[7.4.4. Viewing Thesis 16](#_Toc155562351)

[7.4.5. Searching Thesis 16](#_Toc155562351)

[7.4.6. Updating and Deleting Data 17](#_Toc155562351)

[7.5. Database Interaction and Management 19](#_Toc155562352)

[7.6. ApplicationsChallenges and Solutions 22](#_Toc155562353)

[7.7. Application Photos 23](#_Toc155562354)

[8. All Used Codes 25](#_Toc155562355)

**1.INTRODUCTION**

This Graduate Thesis System (GTS) project focuses on designing a Microsoft SQL Server database to efficiently manage data related to completed graduate theses. The objective is to create a robust and normalized relational database structure, ensuring data integrity, efficient querying capabilities, and seamless scalability. By utilizing Microsoft SQL Server, the project aims to deliver a reliable and user-friendly platform for effective interaction with the database. This system enhances the management and retrieval of graduate thesis information, enabling academic institutions and researchers to organize and access critical data with ease and precision.

**2. ER**

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3. DATABASE STRUCTURE AND ASSOCIATIONS**

These SQL commands were utilized to create each table and define the relationships between them. Logical assumptions were made during the table creation process, ensuring the integrity of the overall database design.

The following tables have been created based on this code:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.1. THESIS TABLE**

Serves as the primary table for storing thesis details, including a unique thesis number (thesis\_no), title, abstract, author information, year, type, university, institute, page count, language, and submission date.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.2. UNIVERSITIES TABLE**

Stores information about universities, including a unique identifier (university\_id) and the name of the university.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.3. INSTITUTES TABLE**

Holds data about academic institutes, such as a unique identifier (institute\_id), name, and a link to the associated university.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.4. PERSONS TABLE**

Contains details about individuals participating in the thesis process, including a unique identifier (person\_id), title, and name.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.5. SUPERVISOR TABLE**

Records details about thesis supervisors, including the thesis they oversee and their unique supervisor ID.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.6. SUBJECT TOPIC TABLE**

Stores various subject topics with a unique identifier (topic\_id) and a descriptive name.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.7. THESIS KEYWORDS TABLE**

Represent different keywords associated with the thesis.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.8. KEYWORDS TABLE**

Stores different keywords associated with the thesis.

metin, diyagram, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**4. RULES AND INTEGRITY CONSTRAINTS**

**4.1. THESIS SUBMISSION DATE TRIGGER**

Check if an author’s thesis submission dates are in logical order.

**metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**4.2. THESIS LANGUAGE TRIGGER**

Verify whether the thesis language flows logically.

metin, ekran görüntüsü, bilgisayar, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**5. FILLING TABLES**

The illustration explains how a thesis is added to the Graduate Thesis System (GTS). At the center, a web form allows users to input thesis details such as title, author, university, keywords, and topics. These inputs are then mapped to the corresponding database tables, including Theses, Persons, Universities, Keywords, and Topics. Arrows indicate the flow of data and relationships between the form and the database, visually demonstrating the system’s functionality.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

The INSERT command is used to add a new record to the SQL database. This command enters the data that you specified in the target table. Adds a new row to the given table. Allows the appropriate data to be entered into the columns of the table. It must be compatible with restrictions such as Primary Key and Foreign Key.

metin, sayı, numara, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**6. QUERY**

The data must be available in the associated tables, for the query to work smoothly. Questioning of all thesis with their thesis no, title, abstract, author name (person’s name), supervisor name, year, submission date, topic and keywords.

**metin, ekran görüntüsü, yazılım, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**7. GRADUATE THESIS MANAGEMENT SYSTEM (GTS) WEB APPLICATION: DESIGN AND DEVELOPMENT**

**7.1. OVERVIEW OF THE APPLICATION**

The GTS web application is designed to help graduate thesis records be managed and retrieved. Its primary purpose is to provide an easy-to-use interface to interact with the Microsoft SQL Server database. Various data are dealt with in the system, like information about thesis, authors, universities, and institutes where these authors belong, and specific characteristics of theses like titles, abstracts, and keywords.

The application was developed in Python, using the Flask framework, which allows for flexible and scalable development. To add the functionality of a web application, it was necessary to install the pip module dependencies, among them, Flask. Integrating Flask with Microsoft SQL Server was carried out using the pyodbc library together with ODBC Driver 17 for SQL Server, allowing fluent integration with the database.

These will involve adding new theses records, updating entries, and querying information within the database. It has been designed in a manner that the system is friendly and efficient enough to enable both staff and students to undertake basic operations without involving themselves with certain complications.

In a fusion that realizes the powerful functionalities of Microsoft SQL Server through its modular design, Flask manages secure data handling. Being extensible and maintainable as it grows, addition and enhancement are provided as necessary to further the evolution of the Graduate Thesis System. Such a combination of a reliable database system with a dynamic Web framework makes this application very efficient, modern, user-centered support for handling graduate thesis.

**7.2. PROGRAMS USED**

In the development of the Graduate Thesis System application, various technologies and frameworks have been used to ensure robust functionality, ease of use and system scalability. Below are the main programs used:

* Microsoft SQL Server: SQL Server is a program developed by Microsoft that is used for relational database management. The meaning of SQL is an acronym that stands for structured query language because the program uses queries, or requests, to locate and analyze the data that is stored in relational databases.
* Python: Python has become a staple in data science, allowing data analysts and other professionals to use the language to conduct complex statistical calculations, create data visualizations, build machine learning algorithms, manipulate and analyze data, and complete other data-related tasks.
* Flask: Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file.
* Pyodbc: pyODBC is a python library that is used for connecting ODBC (Open Database Connectivity) databases through an interface. This simple to use package provides generic methods for accessing and interacting with various database platforms as well Microsoft SQL Server.
* ODBC: Open Database Connectivity (ODBC) is an open standard application programming interface (API) that allows application programmers to easily access data stored in a database.

**7.3. STRUCTURE OF APP**

The Graduate Thesis System Flask application is designed to ensure seamless database interaction and deliver an intuitive user interface. The key elements of the application's structure are outlined below:

**7.3.1. Routes**

Routes, or endpoints, are the URLs that the application responds to. Each route is linked to a specific Python function that runs when the route is accessed. In a Flask application, various routes are defined to manage different functionalities.

@app.route('/'): Home page where the summary of all thesis records is displayed and redirection buttons are available.

@app.route('/add\_keyword\_and\_thesis'): The page where new thesis records and keywords are uploaded.

@app.route('/add\_person\_and\_supervisor'): The page where contact records are added.

@app.route('/add\_university): The page where add university list.

@app.route('/add\_institute): The page that has university departmants.

@app.route('/search', methods=['GET', 'POST']): A page where you can search for each data in the thesis table one by one.

and more routes for adding, editing, deleting, and retrieving. Each route is intended to handle specific tasks, such as fetching data, rendering forms, or modifying the database, depending on the user's interactions.

**7.3.2. Templates**

Templates in Flask are rendered using the Jinja2 engine, which allows HTML files to include placeholders for dynamic content provided by the application. In the GTS application, templates form the backbone of the user interface, presenting data and forms to users. Jinja, Flask's template library, handles rendering these templates into HTML that is displayed in the browser. To ensure security, Jinja is configured to automatically escape all data included in HTML templates.

index.htlm: Index. html is an HTML file that serves as the home page for a website. It's the first file that visitors to flask website see.

add person, university, institute, topic. html: Includes forms for adding new thesis, person, institute, university, etc.

search.html: A file prepared for detailed thesis search.

Templates enable a clear distinction between the application's business logic and its presentation layer, making the code easier to maintain and the user interface more adaptable.

**7.3.3. Database Connection to Flask**

The application establishes a connection to a Microsoft SQL Server database using the pyodbc library, which serves as a Python interface for SQL Server. This connection is configured with parameters such as the server address, database name, username, and password.  
The connection is generally set up at the start of the application and is reused for executing queries throughout its runtime.  
Transactions may be handled automatically or manually, depending on the type of operation. For instance, adding or modifying records might require transaction control to ensure that changes are either saved or reverted as necessary.  
Cursors, obtained from the connection object, are used to run SQL queries, retrieve data, and carry out other database-related tasks.  
Proper management of database connections allows the application to run queries efficiently and handle multiple requests seamlessly, ensuring a reliable backend for the Flask application.

metin, yazılım, bilgisayar simgesi, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4. APPLICATION OPERATIONS’**

The Flask application for the Graduate Thesis System is designed to provide a comprehensive suite of functionalities to manage the thesis database efficiently. Below are the key operations that users can perform:

**7.4.1. Adding Thesis**

Thesis Records: Users can create new thesis entries through the 'add\_data' page by supplying required information, including the title, abstract, author, year, type, and the affiliated university and institute. Before inserting the data, the application ensures it aligns with the database schema.  
The /add\_keyword\_and\_thesis route in the Flask application handles the process of adding new thesis records to the database.

metin, ekran görüntüsü, sayı, numara, makbuz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4.2. Adding Persons**

Persons can only add themselves as supervisors.metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4.3. Adding Universities & Institutes**

Users can add new universities and their associated institutes.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4.4. Viewing Thesis**

metin, sayı, numara, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4.5. Searching Thesis**

Searching Screen: The index page provides an overview of all thesis records, enabling users to explore the entire collection effortlessly.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Searching Method: Users can conduct advanced searches using various criteria like author, university, keywords, and more. This feature includes a form where users can define their search parameters, allowing the application to fetch relevant records from the database.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7.4.6. Updating and Deleting Data**

Users can update existing records, such as modifying a thesis's details, editing an author's name, or changing university information. Each entity type has a dedicated interface for making updates, ensuring both data integrity and ease of use. Additionally, users can delete records from the database, such as a thesis, a person, or an institute. The application provides confirmation prompts to avoid accidental deletions and manages related data appropriately, ensuring, for example, that deleting a thesis also removes its associated supervisor relationships.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Integrity:

The application enforces referential integrity, ensuring that the deletion of a record does not result in orphaned entries or disrupt the consistency of the database. For instance; there have already taken number 1 so basically means that one person can take unique value in database table.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

These features collectively offer a comprehensive interface for managing all components of the Graduate Thesis System, from the in-depth management of thesis records to handling related entities like authors and institutions, enabling users to interact with the database efficiently and effectively.

**7.5. DATABASE INTERACTION AND MANAGEMENT**

The Flask application interacts with the Microsoft SQL Server database through the pyodbc library, which provides functions and methods to execute SQL Server commands from Python. Here's how the application manages database interaction:

Database Connection: At the start of the application, a connection is established to the Microsoft SQL Server database using credentials (host, database, user, password). This connection is used to create a cursor object, which is then used to execute SQL commands and queries.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

SQL Queries: The application uses SQL commands to communicate with the database, such as:  
INSERT statements for adding new entries to tables like Theses, Persons, Universities, and others.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

SELECT statements are used to retrieve data, such as listing all theses or performing searches based on specific criteria.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

UPDATE is used to modify existing records, such as changing the details of a thesis or updating a person's name.



DELETE for removing records from the database.



**Transaction management** to maintain data integrity, the application utilizes transactions. For operations like adding a thesis, which require inserting data into multiple tables, a transaction is initiated. If an error arises during any step, the transaction is rolled back, preventing any changes to the database. If all operations succeed, the transaction is committed, and the changes are permanently saved.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

In **Managing relationships** the application ensures the integrity of relationships between various entities. For instance, when adding a thesis, it verifies that the specified author exists in the Persons table and that the chosen university and institute are valid. Additionally, it handles many-to-many relationships, such as a thesis being associated with multiple topics or keywords.

Error Handling: The application is built to manage database errors effectively. For example, if an insert operation violates a constraint (such as a duplicate entry or a foreign key constraint), the application captures the exception, rolls back the transaction if needed, and displays a relevant error message to the user.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Database Schema Conformance: The application adheres to the defined database schema, including data types, constraints, and relationships. For instance, it ensures that the thesis number falls within the permitted range or that the language matches one of the predefined options.

**7.6. APPLICATION CHALLENGES AND SOLUTIONS**

Throughout the development of the Graduate Thesis System, various obstacles were faced in creating a functional and user-friendly application. Below are some of the major challenges encountered and the strategies used to resolve them:

**Handling Data Across Multiple Tables**  
Inserting related data into multiple tables simultaneously, such as a thesis, its author, supervisors, and topics, was complex and prone to errors.

**Solution:** Transactional control was implemented in Python using pyodbc, ensuring that all insert operations were processed as a single atomic transaction. This guaranteed that either all changes were committed or none, preserving the integrity of the database.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Designing an Intuitive User Interface**  
Creating a user-friendly and aesthetically appealing interface that would simplify navigation and data entry.

**Solution:** HTML and CSS were used to build dynamic, responsive web pages. Bootstrap ensured the application was styled for responsiveness, making it accessible across different devices and screen sizes.

**7.7. APPLICATION PHOTOS**

Index page:

metin, sayı, numara, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Updating topics page:

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Adding institutes page:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

How trigger works:



metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**8. ALL USED CODES**

from flask import Flask, request, render\_template, redirect, url\_for, flash

import pyodbc

app = Flask(\_\_name\_\_)

app.secret\_key = 'yarensecretkey'

def get\_db\_connection():

    conn = pyodbc.connect(

        r'DRIVER={ODBC Driver 17 for SQL Server};'

        r'SERVER=DESKTOP-V9R50E7\SQLEXPRESS;'

        r'DATABASE=Graduate\_Thesis\_System;'

        r'Trusted\_Connection=yes;'

    )

    return conn

# Home page and listing all thesis

@app.route('/')

def index():

    conn = get\_db\_connection()

    cursor = conn.cursor()

    query = "SELECT \* FROM Thesis"

    cursor.execute(query)

    results = cursor.fetchall()

    conn.close()

    return render\_template('index.html', theses=results)

#Add new thesis page

@app.route('/add\_keyword\_and\_thesis', methods=['GET', 'POST'])

def add\_keyword\_and\_thesis():

    if request.method == 'POST':

        conn = None

        try:

            keyword\_id = request.form.get('keyword\_id')

            keyword = request.form.get('keyword')

            topic\_id = request.form.get('topic\_id')

            topic\_name = request.form.get('topic\_name')

            thesis\_no = request.form.get('thesis\_no')

            title = request.form.get('title')

            abstract = request.form.get('abstract')

            year = request.form.get('year')

            thesis\_type = request.form.get('type')

            language = request.form.get('language')

            university\_id = request.form.get('university\_id')

            institute\_id = request.form.get('institute\_id')

            pages\_no = request.form.get('pages\_no')

            author\_id = request.form.get('author\_id')

            submission\_date = request.form.get('submission\_date')

            print("Formdan alınan veriler:")

            print(f"KeywordID: {keyword\_id}, Keyword: {keyword}")

            print(f"ThesisNo: {thesis\_no}, Title: {title}, Abstract: {abstract}, Year: {year}, Type: {thesis\_type}")

            print(f"Language: {language}, UniversityID: {university\_id}, InstituteID: {institute\_id}, PagesNo: {pages\_no}")

            print(f"AuthorID: {author\_id}, SubmissionDate: {submission\_date}")

            supported\_languages = ['English', 'Turkish', 'French']

            if language not in supported\_languages:

                flash(f"Invalid language: {language}. Accepted values are: {', '.join(supported\_languages)}", 'danger')

                return redirect(url\_for('add\_keyword\_and\_thesis'))

            conn = get\_db\_connection()

            cursor = conn.cursor()

            conn.autocommit = False

            query\_keywords = """

            INSERT INTO Keywords (KeywordID, Keyword)

            VALUES (?, ?)

            """

            cursor.execute(query\_keywords, (keyword\_id, keyword))

            print(f"Keywords tablosuna eklendi: KeywordID={keyword\_id}, Keyword={keyword}")

            query\_subject\_topics = """

            INSERT INTO SubjectTopics (TopicID, TopicName)

            VALUES (?, ?)

            """

            print("SubjectTopics tablosuna ekleme yapılıyor...")

            cursor.execute(query\_subject\_topics, (topic\_id, topic\_name))

            print(f"SubjectTopics tablosuna eklendi: TopicID={topic\_id}, TopicName={topic\_name}")

            query\_thesis\_topic = """

            INSERT INTO ThesisTopic (TopicID, ThesisNo)

            VALUES (?, ?)

            """

            print("ThesisTopic tablosuna ekleme yapılıyor...")

            cursor.execute(query\_thesis\_topic, (topic\_id, thesis\_no))

            print(f"ThesisTopic tablosuna eklendi: TopicID={topic\_id}, ThesisNo={thesis\_no}")

            query\_thesis\_keyword = """

            INSERT INTO ThesisKeyword (ThesisNo, KeywordID)

            VALUES (?, ?)

            """

            print("ThesisKeyword tablosuna ekleme yapılıyor...")

            cursor.execute(query\_thesis\_keyword, (thesis\_no, keyword\_id))

            # Add data to the Thesis table

            query\_thesis = """

            INSERT INTO Thesis (ThesisNo, Title, Abstract, Year, Type, Language, UniversityID, InstituteID, PagesNo, AuthorID, SubmissionDate)

            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

            """

            print("Thesis tablosuna ekleme yapılıyor...")

            cursor.execute(query\_thesis, (thesis\_no, title, abstract, year, thesis\_type, language, university\_id, institute\_id, pages\_no, author\_id, submission\_date))

            print("Veritabanı commit ediliyor...")

            conn.commit()

            flash('Keyword and Thesis added successfully!', 'success')

        except Exception as e:

            print(f"Hata oluştu: {str(e)}")

            if conn:

                conn.rollback()

            flash(f'Error: {str(e)}', 'danger')

        finally:

            print("Veritabanı bağlantısı kapatılıyor...")

            if conn:

                conn.close()

        return redirect(url\_for('index'))

    return render\_template('add\_keyword\_and\_thesis.html')

#New person adding page

@app.route('/add\_person\_and\_supervisor', methods=['GET', 'POST'])

def add\_person\_and\_supervisor():

    if request.method == 'POST':

        person\_id = request.form.get('person\_id')

        name = request.form.get('name')

        title = request.form.get('title')

        thesis\_no = request.form.get('thesis\_no')

        try:

            conn = get\_db\_connection()

            cursor = conn.cursor()

            conn.autocommit = False

            query\_persons = "INSERT INTO Persons (PersonID, Name, Title) VALUES (?, ?, ?)"

            cursor.execute(query\_persons, (person\_id, name, title))

            query\_supervisor = "INSERT INTO Supervisor (PersonID, ThesisNo) VALUES (?, ?)"

            cursor.execute(query\_supervisor, (person\_id, thesis\_no))

            conn.commit()

            conn.close()

            flash('Person and Supervisor added successfully!', 'success')

        except Exception as e:

            conn.rollback()

            conn.close()

            flash(f'Error: {str(e)}', 'danger')

            return redirect(url\_for('add\_person\_and\_supervisor'))

    return render\_template('add\_person\_and\_supervisor.html')

@app.route('/add\_university', methods=['GET', 'POST'])

def add\_university():

    if request.method == 'POST':

        university\_id = request.form.get('university\_id')

        name = request.form.get('name')

        try:

            conn = get\_db\_connection()

            cursor = conn.cursor()

            query\_university = "INSERT INTO Universities (UniversityID, Name) VALUES (?, ?)"

            cursor.execute(query\_university, (university\_id, name))

            #

            conn.commit()

            flash('University added successfully!', 'success')

        except Exception as e:

            conn.rollback()

            flash(f'Error: {str(e)}', 'danger')

        finally:

            conn.close()

    return render\_template('add\_university.html')

#New institute adding page

@app.route('/add\_institute', methods=['GET', 'POST'])

def add\_institute():

    if request.method == 'POST':

        institute\_id = request.form.get('institute\_id')

        university\_id = request.form.get('university\_id')

        name = request.form.get('name')

        try:

            conn = get\_db\_connection()

            cursor = conn.cursor()

            query\_institute = """

            INSERT INTO Institutes (InstituteID, UniversityID, Name)

            VALUES (?, ?, ?)

            """

            cursor.execute(query\_institute, (institute\_id, university\_id, name))

            conn.commit()

            conn.close()

            flash('Institute added successfully!', 'success')

        except Exception as e:

            conn.rollback()

            conn.close()

            flash(f'Error: {str(e)}', 'danger')

            return redirect(url\_for('add\_institute'))

    return render\_template('add\_institute.html')

@app.route('/view/<table\_name>')

def view\_table(table\_name):

    try:

        conn = get\_db\_connection()

        cursor = conn.cursor()

        query = f"SELECT \* FROM {table\_name}"

        cursor.execute(query)

        columns = [column[0] for column in cursor.description]

        rows = cursor.fetchall()

        return render\_template('view\_table.html', table\_name=table\_name, columns=columns, rows=rows)

    except Exception as e:

        flash(f'Error fetching data: {str(e)}', 'danger')

        return redirect(url\_for('index'))

    finally:

        conn.close()

@app.context\_processor

def utility\_processor():

    return dict(zip=zip)

#Editing tables page

@app.route('/edit/<table\_name>/<int:id>', methods=['GET', 'POST'])

def edit\_table(table\_name, id):

    try:

        conn = get\_db\_connection()

        cursor = conn.cursor()

        id\_column = (

            'UniversityID' if table\_name.lower() == 'universities' else

            'TopicID' if table\_name.lower() == 'subjecttopics' else

            'InstituteID' if table\_name.lower() == 'institutes' else

            'ID'

        )

        if request.method == 'POST':

            form\_data = request.form.to\_dict()

            updates = ", ".join([f"{key} = ?" for key in form\_data.keys()])

            values = list(form\_data.values()) + [id]

            query = f"UPDATE {table\_name} SET {updates} WHERE {id\_column} = ?"

            cursor.execute(query, values)

            conn.commit()

            flash(f'Record updated in {table\_name}!', 'success')

            return redirect(url\_for('view\_table', table\_name=table\_name))

        query = f"SELECT \* FROM {table\_name} WHERE {id\_column} = ?"

        cursor.execute(query, (id,))

        record = cursor.fetchone()

        columns = [column[0] for column in cursor.description]

        return render\_template('edit\_table.html', table\_name=table\_name, columns=columns, record=record)

    except Exception as e:

        flash(f'Error: {str(e)}', 'danger')

        return redirect(url\_for('view\_table', table\_name=table\_name))

    finally:

        conn.close()

@app.route('/delete/<table\_name>/<int:id>', methods=['POST'])

def delete\_record(table\_name, id):

    try:

        conn = get\_db\_connection()

        cursor = conn.cursor()

        primary\_key\_columns = {

            'universities': 'UniversityID',

            'subjecttopics': 'TopicID',

            'institutes': 'InstituteID',

            'persons': 'PersonID',

        }

        id\_column = primary\_key\_columns.get(table\_name.lower(), 'ID')

        query = f"DELETE FROM {table\_name} WHERE {id\_column} = ?"

        cursor.execute(query, (id,))

        conn.commit()

        flash(f'Record deleted from {table\_name}!', 'success')

    except Exception as e:

        flash(f'Error deleting record: {str(e)}', 'danger')

    finally:

        if 'conn' in locals():

            conn.close()

    return redirect(url\_for('view\_table', table\_name=table\_name))

#Thesis search page with each data

@app.route('/search', methods=['GET', 'POST'])

def search():

    results = []

    if request.method == 'POST':

        column\_name = request.form.get('column\_name')

        search\_value = request.form.get('search\_value')

        try:

            conn = get\_db\_connection()

            cursor = conn.cursor()

            query = f"SELECT \* FROM Thesis WHERE {column\_name} LIKE ?"

            cursor.execute(query, ('%' + search\_value + '%',))

            results = cursor.fetchall()

            conn.close()

        except Exception as e:

            flash(f"Error: {str(e)}", 'danger')

    return render\_template('search.html', results=results)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)