Database interaction with Flask app using the ORM approach:

Relational Database Management System (RDBMS)

1. **Definition**:
    - A **Relational Database Management System (RDBMS)** is a type of database management system that organizes data into **tables** (rows and columns).
    - Data in RDBMS is stored in a structured format, following relationships defined between tables.
2. **Features of RDBMS**:
    - **Structured Storage**: Data is stored in rows (records) and columns (fields).
    - **Relationships**: Establishes relationships between tables using primary and foreign keys.
    - **Scalability**: Suitable for large-scale applications.
    - **ACID Compliance**: Ensures **Atomicity, Consistency, Isolation, and Durability** of database transactions.
3. **Examples of RDBMS**:
    - SQLite
    - MySQL
    - PostgreSQL
    - Microsoft SQL Server
    - Oracle Database

SQL (Structured Query Language)

1. **Definition**:
    - SQL is the standard programming language used to interact with relational databases.
    - It allows users to **create**, **read**, **update**, and **delete** (CRUD) data in a database.
2. **Key SQL Operations**:
    - **DDL (Data Definition Language)**:
        - Commands like `CREATE`, `ALTER`, `DROP`, `TRUNCATE` which define the structure of the database.
    - **DML (Data Manipulation Language)**:
        - Commands like `INSERT`, `UPDATE`, `DELETE`, which manipulate the data in the database.

    - **DCL (Data Control Language)**:
        - Commands like `GRANT`, `REVOKE`, which control access to the database.
    - **TCL (Transaction Control Language)**:
        - Commands like `COMMIT`, `ROLLBACK`, which manage transactions.
    - **DRL (Data Retrieval Language)**: `SELECT` command, to get/read the records from the table.
3. **Advantages of SQL**:
    - Easy to learn and use.
    - Widely supported across all RDBMS platforms.
    - Enables powerful querying and manipulation of data.

Object-Relational Mapping (ORM) Approach

1. **What is ORM?**
    - Object-Relational Mapping (ORM) is a programming technique used to interact with a relational database using **object-oriented principles**.
    - It eliminates the need to write raw SQL queries by mapping database tables to **classes** and rows to **objects**.
2. **Why Use ORM?**
    - **Simplifies Code**: Developers can work with objects instead of writing SQL queries.
    - **Database Independence**: Makes applications portable across different RDBMS platforms.
    - **Security**: Reduces the risk of SQL injection by abstracting database interactions.
    - **Readability**: ORM-based code is easier to read and maintain than raw SQL.
3. **How ORM Works**:
    - **Classes as Tables**: ORM maps Python classes to database tables.
    - **Objects as Rows**: Each object instance represents a row in the corresponding table.
    - **Attributes as Columns**: Class attributes map to table columns.
    - **Methods for Operations**: ORM provides methods for creating, reading, updating, and deleting data.
4. **Examples of ORM Tools**:
    - **SQLAlchemy**: A popular Python ORM that integrates seamlessly with Flask.
    - **Django ORM**: Used in Django web applications.

- **Hibernate**: A Java-based ORM.

Benefits of Using ORM in Flask Applications

1. **Abstraction**:
   Developers can focus on the application logic without worrying about the underlying database queries.
2. **Faster Development**:
   Eliminates repetitive SQL code, making the development process faster and more efficient.
3. **Cross-Database Compatibility**:
   ORM tools can work with multiple databases without requiring code changes.
4. **Ease of Maintenance**:
   Changes to the database schema or logic can be reflected easily in the corresponding classes.

SQLite:

- Python and Flask can connect to a variety of RDBMS software, including PostgreSQL, MYSQL, SQLite and more.
- SQLite is a simple RDBMS software that comes with Flask and can handle all our needs.
- SQLite(despite its name) can actually scale quite well for basic applications(100,000 hits per day)

Using SQLAlchemy with Flask:

- SQLAlchemy is a popular database toolkit and Object-Relational Mapper (ORM) for Python applications. It simplifies database interactions by allowing developers to work with Python objects instead of raw SQL queries for create, update, select and delete from our database. Flask integrates seamlessly with SQLAlchemy via the `Flask-SQLAlchemy` extension.

  pip install Flask-SQLAlchemy

- To begin working with the Databases, we'll do the following:
    1. Set up the SQLite database in a Flask App
    2. Create a Model in Flask App
    3. Perform the basic CRUD on out Model

Folder Structure:

```
Flask_SQLAlchemy_App
                |
                |--app.py
                |
                |--templates/
                |        |--index.html
                |
                |--app.db
```

Configuring Flask-SQLAlchemy:

1. **Database Configuration in Flask:**

Example 1 Basic:

app.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.db'
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

########################################
```

```python
# Define the Student model


class Student(db.Model):
    __tablename__ = "student"



   id = db.Column(db.Integer, primary_key=True)
   name = db.Column(db.String(100), nullable=False)
   age = db.Column(db.Integer, nullable=False)
   course = db.Column(db.String(50), nullable=False)



# parameterized constructor, (Optional)
    def __init__(self, name, age, course):

        self.name = name
        self.age = age
        self.course = course

# String representation of the object
    def __repr__(self):
        return f"Id is:{self.id} Name is: {self.name} Age is: {self.age}
course is: {self.course} "


# Database operations within the application context
with app.app_context():
    db.create_all()

    # Creating the student object

    student1 = Student("Raj", 25, "Computer Science")
    student2 = Student("Simran",22, "IT")

    # Without having a parameterized constructor
   #student1 = Student(name="Raj", age=25, course="Computer Science")
```

```
    # student2 = Student(name="Simran", age=22, course="IT")


    # Here roll will become None
    print(student1)
    print(student2)

    # db.session.add(student1)
    # db.session.add(student2)

    db.session.add_all([student1, student2])
    db.session.commit()

    # Here The id will be 1 and 2
#Primary key column will be auto-generated if we don't give its value
    print(student1)
    print(student2)
```

Code Breakdown and Explanation

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
```

1. Importing Required Modules

- **`Flask`**: This is the Flask framework, used to create the web application.
- **`SQLAlchemy`**: This is the ORM (Object Relational Mapper) that allows us to interact with a relational database like SQLite using Python objects instead of raw SQL queries.

```
app = Flask(__name__)
```

2. Initializing Flask App

- `Flask(__name__)` creates an instance of the Flask application.
- `__name__` refers to the current module name and helps Flask determine where to look for resources like templates and static files.

```
# Configure SQLite database
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.db'
```

3. Setting Up the Database URI

- `app.config` is used to configure the Flask application.
- `'SQLALCHEMY_DATABASE_URI'` is a special key that tells SQLAlchemy which database to use.
- `'sqlite:///students.db'` means:
  - `sqlite://` → Specifies that SQLite is the database system.
  - `students.db` → The database file will be created in the same directory as this script.
  - If `students.db` does not exist, it will be created automatically when the database is initialized.

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

4. Disabling Modification Tracking

- `SQLALCHEMY_TRACK_MODIFICATIONS` is a setting that enables/disables tracking of object modifications before committing them.
- Why `False`?

- Saves memory (less overhead).
- Avoids warnings from Flask-SQLAlchemy.
- Improves performance (since change tracking is not needed).

```
# Initialize database
db = SQLAlchemy(app)
```

5. Initializing SQLAlchemy

- `SQLAlchemy(app)`: Creates a database object (`db`) that we can use to define and interact with the database.
- This line connects SQLAlchemy with the Flask app.

```
class Student(db.Model):
```

6. Defining the Student Model (Database Table)

- `db.Model`: This makes `Student` a model (table) in the database.
- Each `Student` instance represents one row in the database.

```
__tablename__ = "student"
```

7. Adding table name

- This explicitly sets the name of the database table associated with the `Student` model.
- By default, SQLAlchemy automatically assigns a table name based on the class name (`Student` → `student` in lowercase).

- But when we define `__tablename__ = "student"`, we are manually specifying that the table should be named `"student"` in the database.

8. Defining the `id` Column (Primary Key)

- `db.Column(...)` defines a column in the database.
- `db.Integer` → The data type of this column is Integer.
- `primary_key=True` → This means:
  - Each student will have a unique `id`.
  - Automatically increments for each new student.

```
name = db.Column(db.String(100), nullable=False)
```

9. Defining the `name` Column

- `db.String(100)` → This column stores text with a maximum of 100 characters.
- `nullable=False` → This means the name cannot be empty (NULL).

```
age = db.Column(db.Integer, nullable=False)
```

10. Defining the `age` Column

- `db.Integer` → Stores the student's age as an integer.
- `nullable=False` → Age cannot be empty.

```
course = db.Column(db.String(50), nullable=False)
```

11. Defining the `course` Column

- `db.String(50)` → Stores the course name (max 50 characters).
- `nullable=False` → Course cannot be empty.

```
def __repr__(self):
     return f"<Student {self.name}>"
```

12. Defining `__repr__` Method

- Purpose: This defines how the `Student` object is represented when printed.

Example Usage:
python
CopyEdit

```
student = Student(name="Alice", age=20, course="Computer Science")
print(student)  # Output: <Student Alice>
```

-
- This makes debugging easier.

```
# Create database
with app.app_context():
    db.create_all()
```

13. Creating the Database

- `app.app_context()`: Creates an application context, allowing SQLAlchemy to interact with the database.
- `db.create_all()`: Creates the database and all tables based on the models (`Student` table in this case).
- If `students.db` already exists, it does nothing.
- If `students.db` does not exist, it creates it automatically.

Complete Workflow

1. Flask App is Created → `app = Flask(__name__)`
2. SQLite Database is Configured → `app.config['SQLALCHEMY_DATABASE_URI']`
3. SQLAlchemy is Initialized → `db = SQLAlchemy(app)`
4. Database Model (`Student`) is Defined → `class Student(db.Model)`
5. Database is Created → `db.create_all()`
6. Now, You Can Add and Retrieve Students from the Database!

Example 2 With os:

app.py:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
import os


basedir = os.path.abspath(os.path.dirname(__file__))
print(__file__)  # e:\PythonWS\PythonApp\app.py

print(basedir)  # e:\PythonWS\PythonApp

app = Flask(__name__)


app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + \
    os.path.join(basedir, "app.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db = SQLAlchemy(app)

########################################
```

```python
# Define the Student model


class Student(db.Model):
    __tablename__ = "student"

    roll = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    address = db.Column(db.String(100))
    marks = db.Column(db.Integer)


# parameterized constructor, (Optional)
    def __init__(self, name, address, marks):

        self.name = name
        self.address = address
        self.marks = marks

# String representation of the object
    def __repr__(self):
        return f"Roll is:{self.roll} Name is: {self.name} Address is:
{self.address} Marks is: {self.marks} "


# Database operations within the application context
with app.app_context():
    db.create_all()

    # Creating the student object

    student1 = Student("Raj", "Delhi", 850)
    student2 = Student("Simran", "Mumbai", 750)

    # Without having a parameterized constructor
   #student1 = Student(roll=101, name="Ram", address="Delhi", marks=850)
    # student2 = Student(name="Shyam", address="Mumbai", marks=650)


    # Here roll will become None
```

```
    print(student1)
    print(student2)

    # db.session.add(student1)
    # db.session.add(student2)

    db.session.add_all([student1, student2])
    db.session.commit()

    # Here The roll will be 1 and 2
#Primary key column will be auto-generated if we don't give its value
    print(student1)
    print(student2)
```

Important Point:

- For performing any insert, update and delete operation we must make use of the commit() function, otherwise the record will not be affected.
- But for performing the select, or read operation there is no need to use the commit() function.
- If we don't use the **basedir** and use the Database URI as: **`sqlite:///app.db`** then it will create this **app.db** file inside the instance folder in our application relative location.

Note: once we run the above application the app.db file will be created inside the current working directory

DB browser for SQLite database:

- If you want to inspect your `app.db` SQLite file and its tables, you can use a graphical user interface (GUI) tool.

Steps to Open the `app.db` File:

Using DB Browser for SQLite

1. **Download and Install**:
    - Download from https://sqlitebrowser.org/.
    - Install the application based on your operating system.
2. **Open `app.db`**:
    - Launch DB Browser for SQLite.
    - Click on **File > Open Database**.
    - Navigate to the location of your `app.db` file.
    - Select and open the file.
3. **View Tables**:
    - Go to the **Database Structure** tab.
    - You'll see a list of all the tables (e.g., `student`).
    - Click on a table to view its columns and structure.
4. **View Data**:
    - Switch to the **Browse Data** tab.
    - Select the table you want to inspect (e.g., `student`) to view its rows.

- **Inserting Data (Create)**
    - You can add new records (rows) to your database using SQLAlchemy's `session.add()` and `session.commit()` methods

      ```
      # Create a new student
      new_student = Student(name="Alice", address="New York", marks=900)
      # Add the student to the session and commit the transaction
      db.session.add(new_student)
      db.session.commit()
      ```

    - Alternatively, you can add multiple records at once using `add_all()`:

      ```
      # Add multiple students
      students = [
      ```

```
    Student(name="Bob", address="California", marks=800),
    Student(name="Charlie", address="Texas", marks=850)
]

# Add all students and commit
db.session.add_all(students)
db.session.commit()
```

- **Selecting all the student records from the table:**

```
all_students = Student.query.all()
print(all_students)
```

- **Selecting by Primary key (roll) column:**

```
student = Student.query.get(1)
print(student)
```

```
#Note the above method is deprecated in the latest version of Flask, instead we should use
# the following method:
```

```
    student = db.session.get(Student, 1000)
    if(student):
        print(student)
    else:
        print("No Student found")
```

.

- **Selecting by non-primary key column:**
    - It will return the list of objects.

    - filter_by() function: for performing simple comparison like ==

```
students_with_marks = Student.query.filter_by(marks= 850).all()
print(students_with_marks)
```

- To get the first record:

```
student = Student.query.filter_by(name="Raj").first()
print(student)
```

- filter() function: for performing some complex comparison like (>, ==, and, or, etc.,)

```
# Marks greater than 800
students = Student.query.filter(Student.marks > 800).all()

# Using multiple conditions with logical operators
students = Student.query.filter( Student.marks > 800, Student.address == 'Delhi' ).all() # Marks greater than 800 and address is Delhi
```

- **Updating a specific record in the table**
  - Here first of all we need to find the object which we need to update

```
student  = db.session.get(Student, 1000)
student.address=”Chennai”
student.marks = 700
db.session.add(student)
db.session.commit()
```

  - # Update multiple records with filter condition

```
Student.query.filter(Student.address == "Mumbai").update( {Student.marks: 800}, synchronize_session=False )
db.session.commit()
```

- **Deleting a specific record from the table:**
  - Here also first we need to find the object which we need to delete

```
student  = db.session.get(Student, 1000)
db.session.delete(student)
db.session.commit()
```

- Deleting multiple records

```
# Delete all students with marks less than 700
Student.query.filter(Student.marks < 700).delete()
db.session.commit()
```

- **Count the Number of total Students/Records**:

```
total_students = Student.query.count()
print(f"Total students: {total_students}")
```

- # Count students with marks greater than 800

```
high_marks_count = Student.query.filter(Student.marks > 800).count()
print(high_marks_count)
```

- **Limiting the Results:**

- # Get the first 3 students

```
top_students = Student.query.limit(3).all()
print(top_students)
```

Example Application1:

- Create a flask application to get the students' details from the form page and after submitting the details, store them inside the database and display all the students' records inside a table from the database.

Folder Structure:

FlaskDatabaseApp1

```
|
|-- app.py
|
|-- templates/|
        |
        |-- base.html
```

|-- index.html

|-- student.html

|

|-- app.db (This will be created automatically when the database is initialized)

app.py:

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
import os
# Initialize Flask application and configure database
app = Flask(__name__)
basedir = os.path.abspath(os.path.dirname(__file__))

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + \
    os.path.join(basedir, "app.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False


db = SQLAlchemy(app)

# Database model


class Student(db.Model):
    roll = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(200), nullable=False)
    marks = db.Column(db.Integer, nullable=False)

    def __repr__(self):
        return f"Student('{self.roll}', '{self.name}', '{self.address}',
'{self.marks}')"


# Create the database tables
with app.app_context():
```

```
    db.create_all()

# Route to display home page


@app.route("/")
def index():
    return render_template("index.html")

# Route to register a student


@app.route("/student", methods=["GET", "POST"])
def student():
    if request.method == "POST":
        # Fetch the form data
        roll = request.form.get("roll")
        name = request.form.get("name")
        address = request.form.get("address")
        marks = request.form.get("marks")

        # Create a new student record in the database
        new_student = Student(
            roll=int(roll),
            name=name,
            address=address,
            marks=int(marks)
        )

        # Add the new student to the database session and commit
        db.session.add(new_student)
        db.session.commit()

    # Query all students from the database
    students = Student.query.all()
    return render_template("student.html", students=students)


if __name__ == '__main__':
    app.run(debug=True)
```

base.html:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title_block %} {% endblock %}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"

integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">
</head>

<body bgcolor="Cyan">
    <header>
        <h1 class="text-center">Welcome to Chitkara</h1>
    </header>
    <main>
        {% block main_block %}
        {% endblock %}
    </main>
    <footer>
        <p class="text-center">&copy; 2025 Student Information System</p>
    </footer>
</body>

</html>
```

index.html:

```
{% extends "base.html" %}
```

```
{% block title_block %} Home Page {% endblock %}

{% block main_block %}
<a href="{{url_for('student')}}">Register A Student</a>
{% endblock %}
```

student.html:

```
{% extends "base.html" %}

{% block title_block %} Student Page {% endblock %}

{% block main_block %}
<h2 class="text-center">Student Registration Screen</h2>
<a href="{{url_for('index')}}">Back</a>

<form action="{{url_for('student')}}" method="POST">
    <div class="form-group">
        <label for="roll">Roll Number</label>
        <input type="number" class="form-control" id="roll"
placeholder="Enter Roll Number" name="roll" required>
    </div>
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name"
placeholder="Enter Name" name="name" required>
    </div>
    <div class="form-group">
        <label for="address">Address</label>
        <input type="text" class="form-control" id="address"
placeholder="Enter Address" name="address" required>
    </div>
    <div class="form-group">
        <label for="marks">Marks</label>
        <input type="number" class="form-control" id="marks"
placeholder="Enter Marks" name="marks" required>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
</form>
```

```
{% if students %}
<hr>
<table class="table table-primary">
    <thead>
        <tr>
            <th>Roll Number</th>
            <th>Student Name</th>
            <th>Student Address</th>
            <th>Student Marks</th>
        </tr>
    </thead>
    <tbody>
        {% for student in students %}
        <tr>
            <td>{{ student.roll }}</td>
            <td>{{ student.name }}</td>
            <td>{{ student.address }}</td>
            <td>{{ student.marks }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
{% endif %}
{% endblock %}
```

Extending the above application: Using CRUD operation:

FlaskDatabaseApp1

```
        |
        |-- app.py
        |
        |-- templates/|
              |
```

|-- base.html
|-- index.html
|-- student.html
|--update_student.html
|
|-- app.db (This will be created automatically when the database is initialized)

app.py:

```
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
import os

# Initialize Flask application and configure database
app = Flask(__name__)
basedir = os.path.abspath(os.path.dirname(__file__))


app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + \
    os.path.join(basedir, "app.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False


db = SQLAlchemy(app)

# Database model


class Student(db.Model):
    roll = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(200), nullable=False)
    marks = db.Column(db.Integer, nullable=False)
   # email= db.Column(db.String(100), unique=True)


    def __repr__(self):
```

```
        return f"Student('{self.roll}', '{self.name}', '{self.address}',
'{self.marks}')"


# Create the database tables
with app.app_context():
    db.create_all()

# Route to display home page


@app.route("/")
def index():
    return render_template("index.html")

# Route to register a student


@app.route("/student", methods=["GET", "POST"])
def student():
    if request.method == "POST":
        # Fetch the form data
        roll = request.form.get("roll")
        name = request.form.get("name")
        address = request.form.get("address")
        marks = request.form.get("marks")

        # Create a new student record in the database
        new_student = Student(
            roll=int(roll),
            name=name,
            address=address,
            marks=int(marks)
        )

        # Add the new student to the database session and commit
        db.session.add(new_student)
        db.session.commit()

    # Query all students from the database
```

```
    students = Student.query.all()
    return render_template("student.html", students=students)

# Route to update a student record


@app.route("/update/<int:roll>", methods=["GET", "POST"])
def update_student(roll):
  #student = Student.query.get_or_404(roll)
  student = db.session.get(Student, roll)

    if request.method == "POST":
        student.name = request.form.get("name")
        student.address = request.form.get("address")
        student.marks = request.form.get("marks")

        db.session.commit()
        return redirect(url_for('student'))

    return render_template("update_student.html", student=student)

# Route to delete a student record


@app.route("/delete/<int:roll>")
def delete_student(roll):
    student = db.session.get(Student, roll)
    db.session.delete(student)
    db.session.commit()
    return redirect(url_for('student'))


if __name__ == '__main__':
    app.run(debug=True)
```

base.html:

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title_block %} {% endblock %}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"

integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">
</head>

<body bgcolor="Cyan">
    <header>
        <h1 class="text-center">Welcome to Chitkara</h1>
    </header>
    <main>
        {% block main_block %}
        {% endblock %}
    </main>
    <footer>
        <p class="text-center">&copy; 2025 Student Information System</p>
    </footer>
</body>

</html>
```

index.html:

```
{% extends "base.html" %}

{% block title_block %} Home Page {% endblock %}

{% block main_block %}
<a href="{{url_for('student')}}">Register A Student</a>
{% endblock %}
```

student.html:

```
{% extends "base.html" %}

{% block title_block %} Student Page {% endblock %}

{% block main_block %}
<h2 class="text-center">Student Registration Screen</h2>
<a href="{{url_for('index')}}">Back</a>

<form action="{{url_for('student')}}" method="POST">
    <div class="form-group">
        <label for="roll">Roll Number</label>
        <input type="number" class="form-control" id="roll"
placeholder="Enter Roll Number" name="roll" required>
    </div>
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name"
placeholder="Enter Name" name="name" required>
    </div>
    <div class="form-group">
        <label for="address">Address</label>
        <input type="text" class="form-control" id="address"
placeholder="Enter Address" name="address" required>
    </div>
    <div class="form-group">
        <label for="marks">Marks</label>
        <input type="number" class="form-control" id="marks"
placeholder="Enter Marks" name="marks" required>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
</form>

{% if students %}
<hr>
<table class="table table-primary">
    <thead>
```

```
        <tr>
            <th>Roll Number</th>
            <th>Student Name</th>
            <th>Student Address</th>
            <th>Student Marks</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for student in students %}
        <tr>
            <td>{{ student.roll }}</td>
            <td>{{ student.name }}</td>
            <td>{{ student.address }}</td>
            <td>{{ student.marks }}</td>
            <td>
                <a href="{{ url_for('update_student', roll=student.roll)
}}" class="btn btn-warning btn-sm">Update</a>
                <a href="{{ url_for('delete_student', roll=student.roll)
}}" class="btn btn-danger btn-sm"
                    onclick="return confirm('Are you sure you want to
delete this record?')">Delete</a>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
{% endif %}
{% endblock %}
```

update_student.html:

```
{% extends "base.html" %}

{% block title_block %} Update Student {% endblock %}

{% block main_block %}
<h2 class="text-center">Update Student Details</h2>
<a href="{{url_for('student')}}">Back</a>
```

```
<form action="{{url_for('update_student', roll=student.roll)}}"
method="POST">
    <div class="form-group">
        <label for="roll">Roll Number</label>
        <input type="number" class="form-control" id="roll" name="roll"
value="{{ student.roll }}" disabled>
    </div>
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name" name="name"
value="{{ student.name }}" required>
    </div>
    <div class="form-group">
        <label for="address">Address</label>
        <input type="text" class="form-control" id="address"
name="address" value="{{ student.address }}" required>
    </div>
    <div class="form-group">
        <label for="marks">Marks</label>
        <input type="number" class="form-control" id="marks" name="marks"
value="{{ student.marks }}" required>
    </div>
    <button type="submit" class="btn btn-primary">Update</button>
</form>
{% endblock %}
```