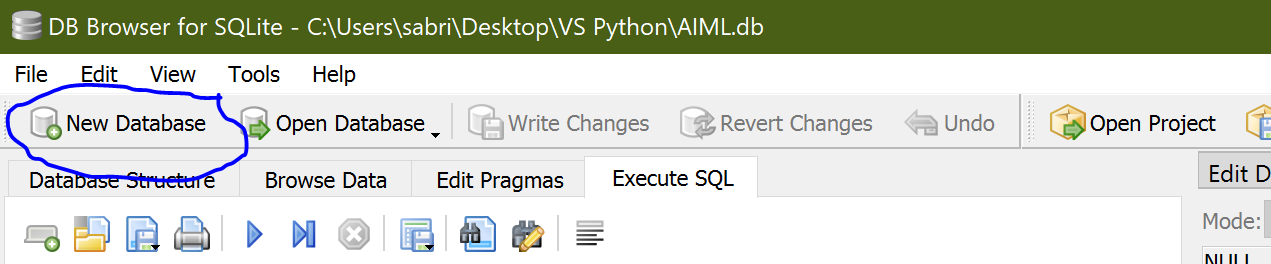
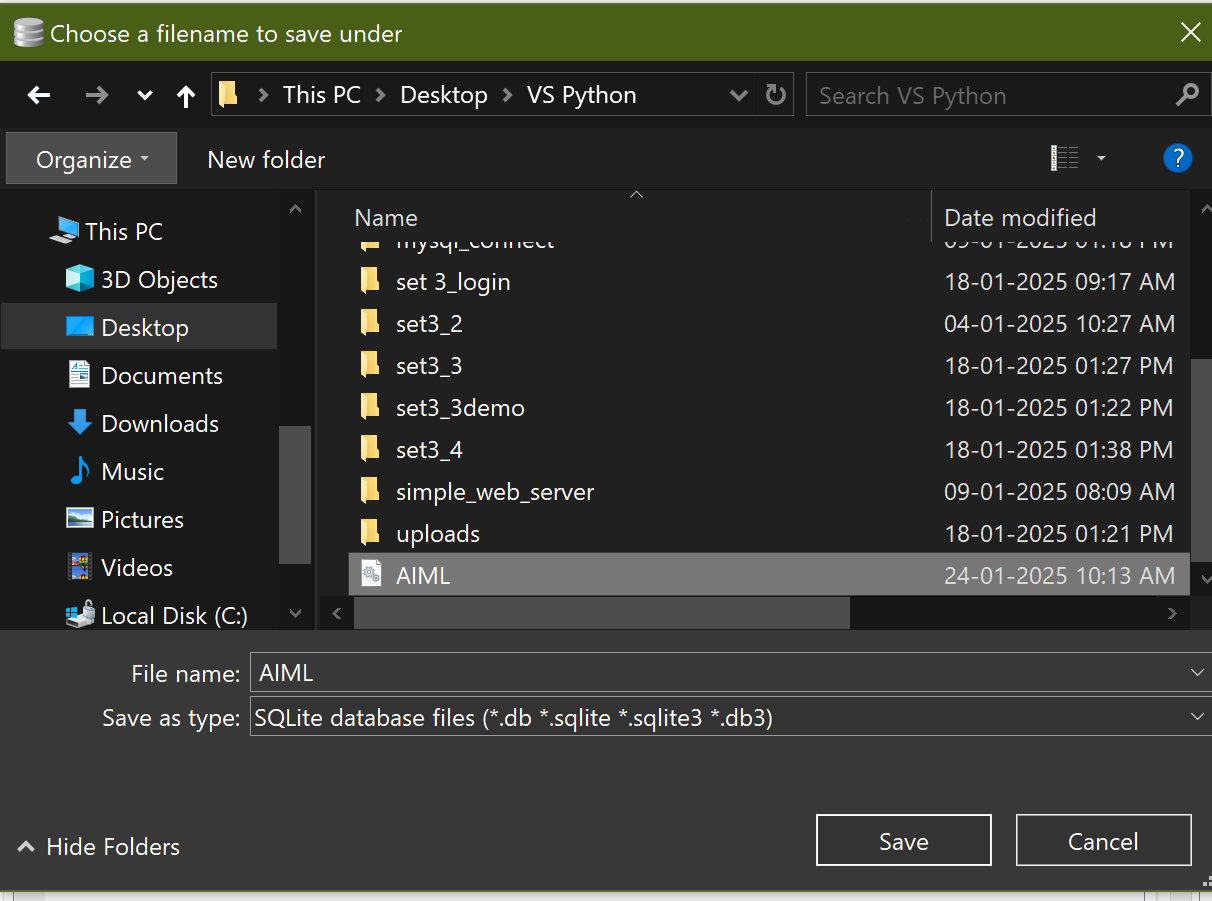
**Flask-** **Database connectivity Sqlite3, MySQL**

**Database Download Link**

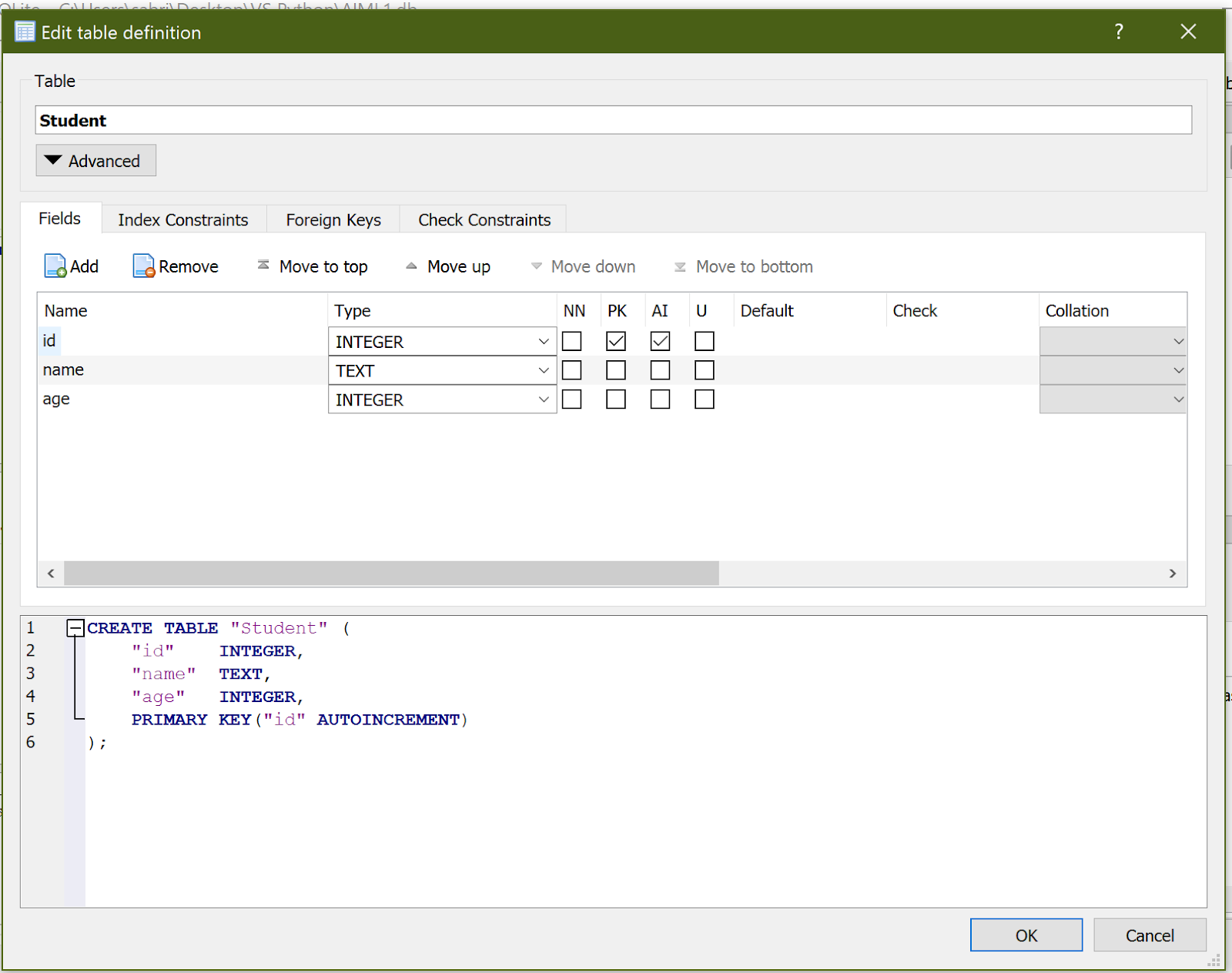
<https://sqlitebrowser.org/dl/>

Step 1:Download software and install it.

Step 2:Craete a DATABASE 1st on name AIML

Step 3: Create a Table save the DB In the local system 

Step 4: Create columns required for your DataBase



Step 5:Sample code checking connectivity

import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)

conn = sqlite3.connect('AIML.db')

# Create a cursor object

cur = conn.cursor()

cur.execute('SELECT \* FROM Student')

# Fetch all results

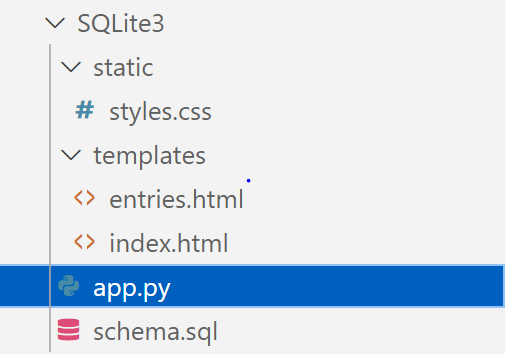
rows = cur.fetchall()

# Print the results

for row in rows:

    print(row)

Example2:



app.py

from flask import Flask, render\_template, request, g

import sqlite3

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

DATABASE = 'database.db'

def get\_db():

    db = getattr(g, '\_database', None)

    if db is None:

        db = g.\_database = sqlite3.connect(DATABASE)

    return db

@app.teardown\_appcontext

def close\_connection(exception):

    db = getattr(g, '\_database', None)

    if db is not None:

        db.close()

def init\_db():

    with app.app\_context():

        db = get\_db()

        cursor = db.cursor()

        cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='entries';")

        if cursor.fetchone() is None:

            with app.open\_resource('schema.sql', mode='r') as f:

                db.cursor().executescript(f.read())

            db.commit()

@app.route('/')

def home():

    return render\_template('index.html')

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

def add\_entry():

    db = get\_db()

    db.execute('INSERT INTO entries (content) VALUES (?)', [request.form['content']])

    db.commit()

    return 'Entry added successfully'

@app.route('/entries')

def show\_entries():

    db = get\_db()

    cursor = db.execute('SELECT id, content FROM entries')

    entries = cursor.fetchall()

    return render\_template('entries.html', entries=entries)

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

    init\_db()  # Initialize the database when the app starts

    app.run(debug=True)

index.html

<!-- templates/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Simple Web Server</title>

    <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

    <div class="container">

        <h1>Welcome to the Simple Web Server!</h1>

        <p>This is a static HTML page served by Flask.</p>

        <form method="POST" action="/add">

            <input type="text" name="content" placeholder="Enter some data">

            <input type="submit" value="Add Entry">

        </form>

        <br>

        <a href="/entries">View All Entries</a>

    </div>

</body>

</html>

entries.html

<!-- templates/entries.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Entries</title>

    <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

    <div class="container">

        <h1>Entries</h1>

        <table>

            <tr>

                <th>ID</th>

                <th>Content</th>

            </tr>

            {% for entry in entries %}

            <tr>

                <td>{{ entry[0] }}</td>

                <td>{{ entry[1] }}</td>

            </tr>

            {% endfor %}

        </table>

    </div>

</body>

</html>

Styles.css

/\* static/styles.css \*/

body {

    font-family: Arial, sans-serif;

    background-color: #f0f0f0;

    margin: 0;

    padding: 0;

}

.container {

    width: 50%;

    margin: 100px auto;

    padding: 20px;

    background-color: #fff;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    border-radius: 8px;

}

h1 {

    color: #333;

    text-align: center;

}

form {

    display: flex;

    flex-direction: column;

    align-items: center;

}

input[type="text"] {

    padding: 10px;

    margin-bottom: 10px;

    width: 80%;

    border: 1px solid #ccc;

    border-radius: 4px;

}

input[type="submit"] {

    padding: 10px 20px;

    background-color: #007BFF;

    color: #fff;

    border: none;

    border-radius: 4px;

    cursor: pointer;

}

input[type="submit"]:hover {

    background-color: #0056b3;

}

Schema.sql

CREATE TABLE entries (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    content TEXT NOT NULL

);

**Flask-Mail:**

Flask-Mail is an extension for Flask that integrates email support into your Flask application. It allows you to configure email servers, send emails, and manage email templates easily. Below, we'll cover the key concepts and components needed to use Flask-Mail effectively.

**Key Concepts**

1. **Configuration** Flask-Mail requires a set of configurations to connect to your email server. These configurations include the server address, port, security settings, and login credentials. Here are the main configuration options:
   * MAIL\_SERVER: The address of your mail server (e.g., smtp.gmail.com).
   * MAIL\_PORT: The port to connect to the mail server (e.g., 587 for TLS, 465 for SSL).
   * MAIL\_USE\_TLS: Boolean value indicating whether to use TLS (Transport Layer Security).
   * MAIL\_USE\_SSL: Boolean value indicating whether to use SSL (Secure Sockets Layer).
   * MAIL\_USERNAME: The email address or username used to log in to the mail server.
   * MAIL\_PASSWORD: The password associated with the email address or username.
2. **Initializing Flask-Mail** You need to create an instance of the Mail class and initialize it with your Flask app. This instance will be used to send emails.
3. **Creating and Sending Emails** The Message class is used to create email messages. You can set the subject, sender, recipients, body, and other properties of the email. The Mail instance's send method is used to send the email.
4. **Templates** Email templates can be used to create the content of your emails. You can render HTML templates using Flask's render\_template function and pass them to the email body.

**Example Usage**

Here's an example that demonstrates the setup and use of Flask-Mail:

**app.py**:

python

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

from flask\_mail import Mail, Message

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

app.secret\_key = 'supersecretkey'

# Configuration for Flask-Mail

app.config['MAIL\_SERVER'] = 'smtp.gmail.com' # Replace with your mail server

app.config['MAIL\_PORT'] = 587

app.config['MAIL\_USE\_TLS'] = True

app.config['MAIL\_USE\_SSL'] = False

app.config['MAIL\_USERNAME'] = “prathiba.r96@gmail.com

app.config['MAIL\_PASSWORD'] = '123456789'

mail = Mail(app)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/send\_mail', methods=['POST'])

def send\_mail():

subject = request.form['subject']

recipient = request.form['recipient']

body = request.form['body']

msg = Message(subject, sender=app.config['MAIL\_USERNAME'], recipients=[recipient])

msg.body = body

mail.send(msg)

flash('Email sent successfully!', 'success')

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

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

app.run(debug=True)

**templates/index.html**:

html

<!DOCTYPE html>

<html>

<head>

<title>Send Email</title>

</head>

<body>

<h1>Send an Email</h1>

{% with messages = get\_flashed\_messages(with\_categories=true) %}

{% if messages %}

<ul>

{% for category, message in messages %}

<li class="{{ category }}">{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

<form method="POST" action="{{ url\_for('send\_mail') }}">

<label for="subject">Subject:</label>

<input type="text" id="subject" name="subject" required><br>

<label for="recipient">Recipient:</label>

<input type="email" id="recipient" name="recipient" required><br>

<label for="body">Body:</label>

<textarea id="body" name="body" required></textarea><br>

<input type="submit" value="Send Email">

</form>

</body>

</html>

To Run

**1.python3 -m venv venv**

**2.venv\Scripts\activate**

**3.cd your\_folder**

**4.pip install Flask**

**5.python my\_flask\_app.py**