Name: Jayshree Patil

Project Title: AI Agent for SQL Query Execution

Introduction

Organizations store vast amounts of data in relational databases. However, not all users are proficient in **SQL**. This project aims to build an **AI-powered SQL agent** that allows users to input queries in natural language, which are then converted into **SQL queries**, executed on a **PostgreSQL** database, and displayed in a structured format.

This project leverages **Google Gemini AI** for natural language processing (NLP), **Flask** for API development, **PostgreSQL** as the database, and **Streamlit** for the user interface.

Problem Statement

Many organizations require seamless interaction with databases for **data retrieval and insights**. However, non-technical users struggle with writing SQL queries. This AI agent will bridge the gap by enabling:

- Conversion of **natural language queries** to SQL
- Execution of queries on a **PostgreSQL** database
- Returning results in **JSON** or tabular format

Technologies & Libraries Used

1.Programming Language:

- **Python**

2.Database:

- **PostgreSQL (Primary)**
- MySQL (Alternative)

3.AI Model for SQL Conversion:

- **Google Gemini API** (via google-generativeai package)

4.Backend Framework:

- **Flask**

5.Frontend UI (Optional):

- **Streamlit**

Name: Jayshree Patil

Dependencies & Libraries:

- Flask → For backend API
- psycopg2 → For PostgreSQL database connection
- mysql-connector-python → For MySQL database connection
- google-generativeai → For AI-based SQL conversion
- pandas → For data handling in Streamlit
- re → For cleaning SQL queries
- json → For formatting query results
- decimal → For handling numeric values
- datetime → For date handling

Installation Commands:

```
pip install Flask psycopg2 mysql-connector-python google-generativeai pandas
```

Project Architecture

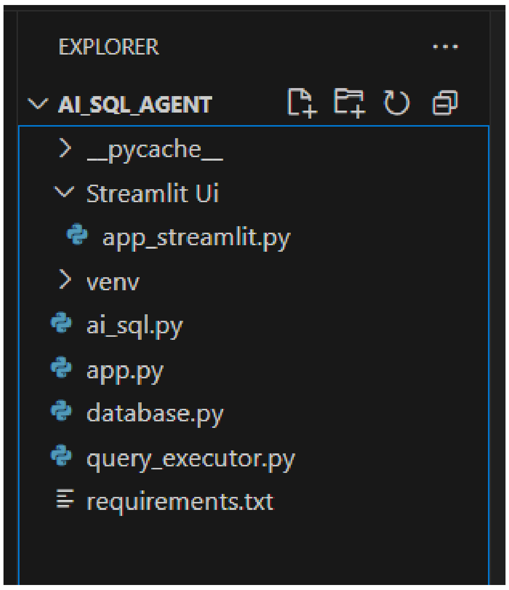
Backend (Flask API):

- database.py: Handles database connection and query execution.
- ai_sql.py: Converts natural language queries to SQL using Google Gemini API.
- query_executor.py: Processes AI-generated SQL and executes it.
- app.py: Main Flask application that serves the API.

Frontend (Streamlit UI):

- streamlit_app.py: Provides a simple UI for user input and displays query results.

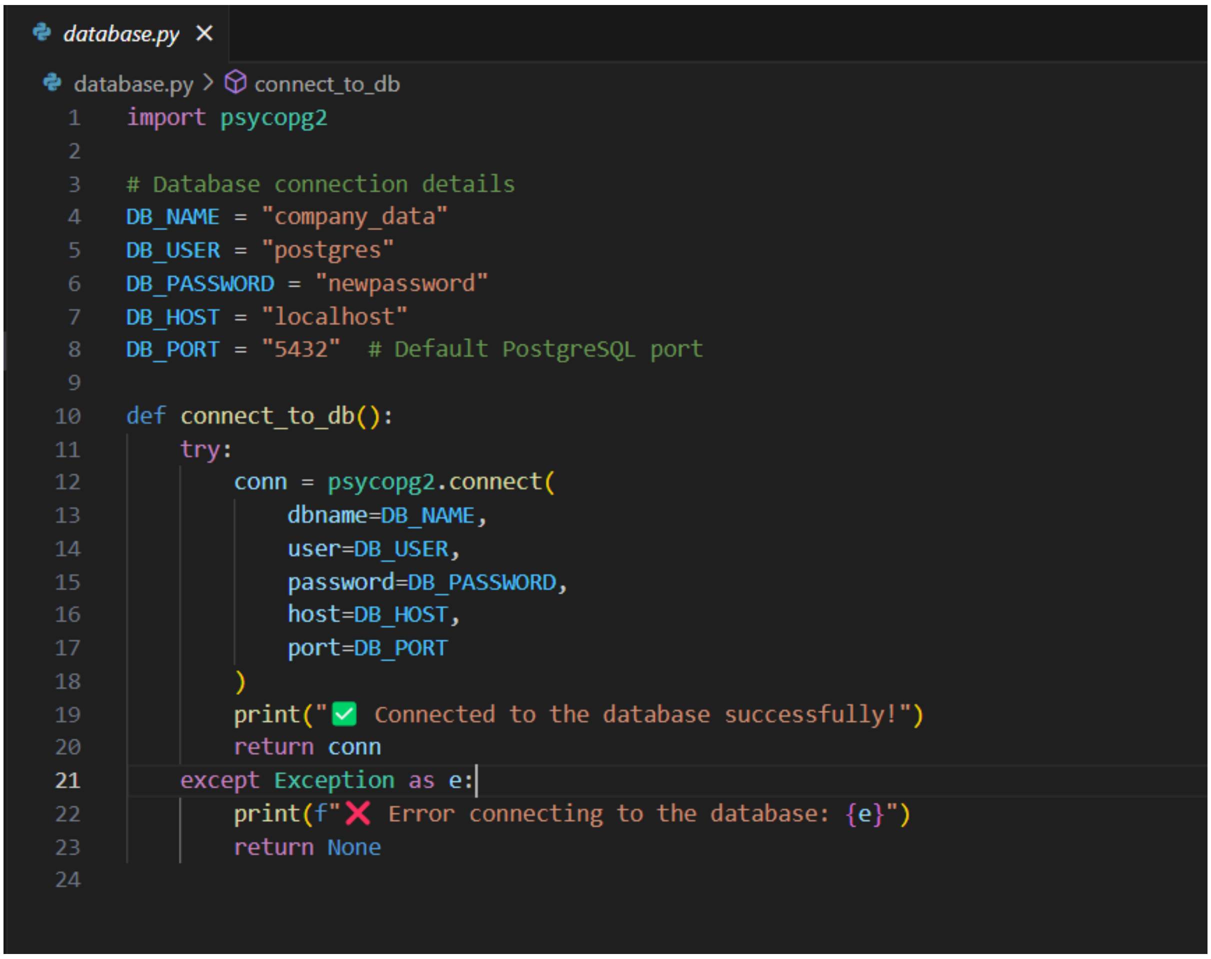
Name: Jayshree Patil



Code Explanation

1.database.py

This code connects to a PostgreSQL database using provided credentials. It returns a connection object on success, or None if the connection fails.

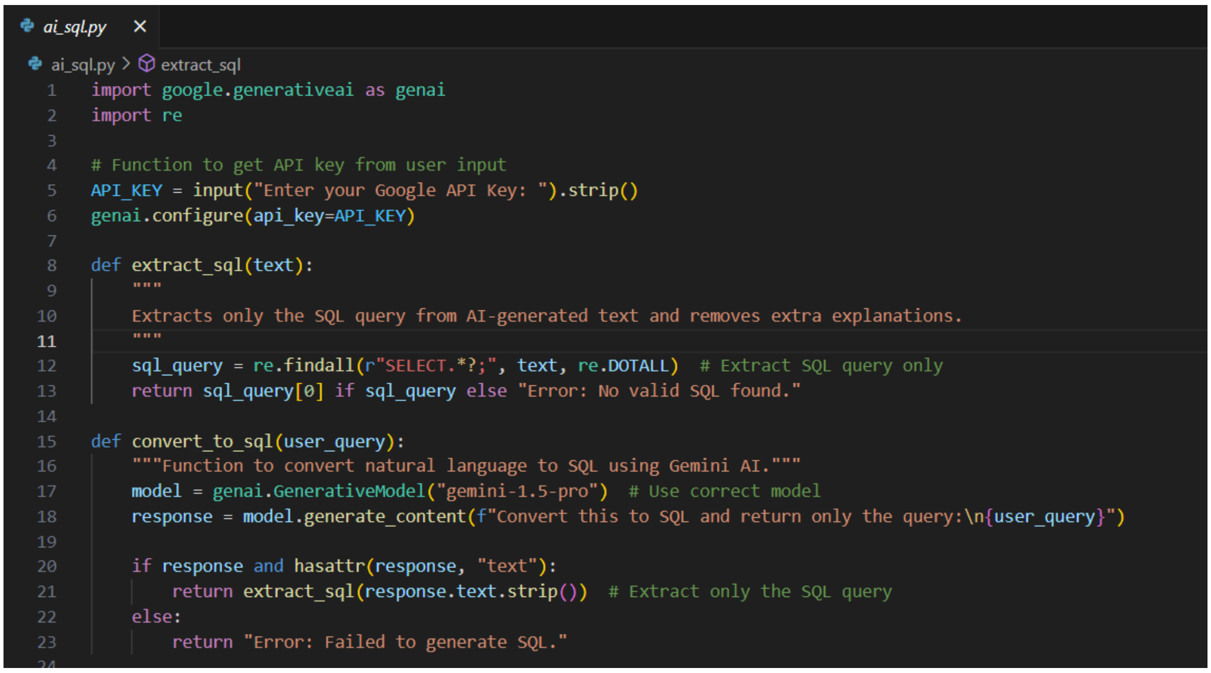


```python
import psycopg2

# Database connection details
DB_NAME = "company_data"
DB_USER = "postgres"
DB_PASSWORD = "newpassword"
DB_HOST = "localhost"
DB_PORT = "5432"  # Default PostgreSQL port

def connect_to_db():
    try:
        conn = psycopg2.connect(
            dbname=DB_NAME,
            user=DB_USER,
            password=DB_PASSWORD,
            host=DB_HOST,
            port=DB_PORT
        )
        print("✅ Connected to the database successfully!")
        return conn
    except Exception as e:
        print(f"❌ Error connecting to the database: {e}")
        return None
```

Name: Jayshree Patil

2.AI Model (ai_sql.py)

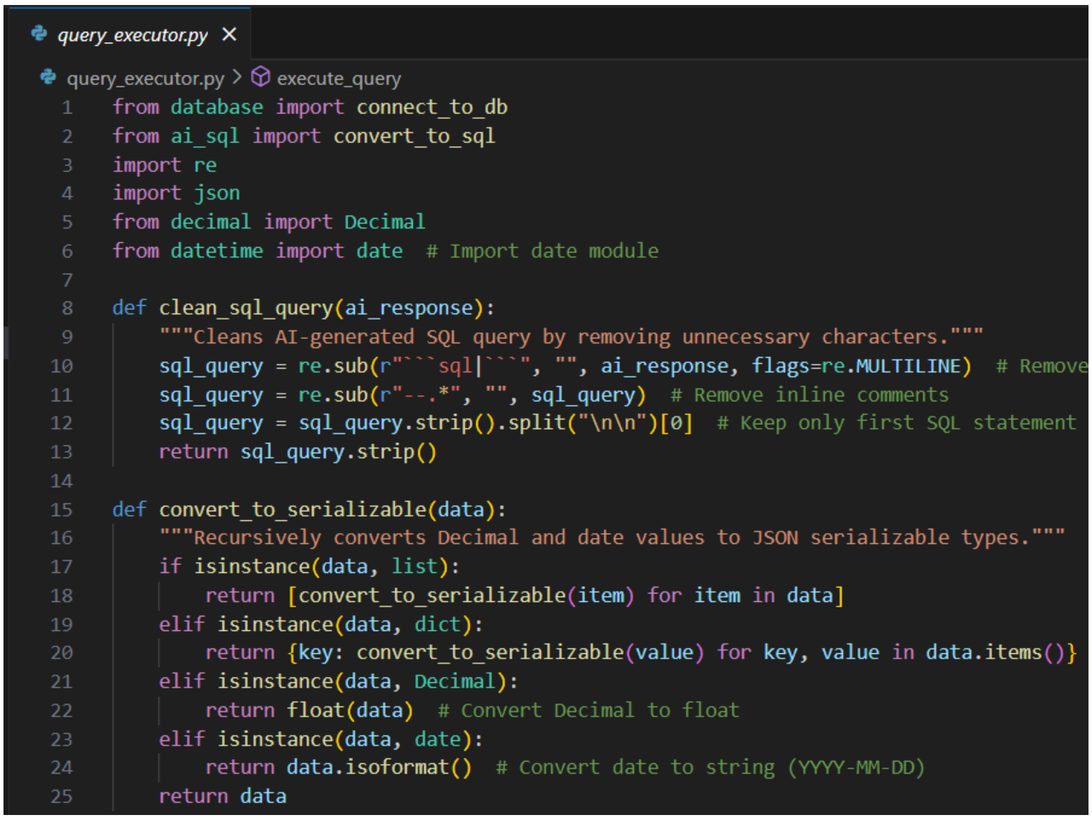
- **Converts user input to SQL** using Google Gemini AI.
- **Extracts** the SQL query from the response.



ai_sql.py > extract_sql

```python
import google.generativeai as genai
import re

# Function to get API key from user input
API_KEY = input("Enter your Google API Key: ").strip()
genai.configure(api_key=API_KEY)

def extract_sql(text):
    """
    Extracts only the SQL query from AI-generated text and removes extra explanations.
    """
    sql_query = re.findall(r"SELECT.*?;", text, re.DOTALL)  # Extract SQL query only
    return sql_query[0] if sql_query else "Error: No valid SQL found."

def convert_to_sql(user_query):
    """Function to convert natural language to SQL using Gemini AI."""
    model = genai.GenerativeModel("gemini-1.5-pro")  # Use correct model
    response = model.generate_content(f"Convert this to SQL and return only the query:\n{user_query}")

    if response and hasattr(response, "text"):
        return extract_sql(response.text.strip())  # Extract only the SQL query
    else:
        return "Error: Failed to generate SQL."
```

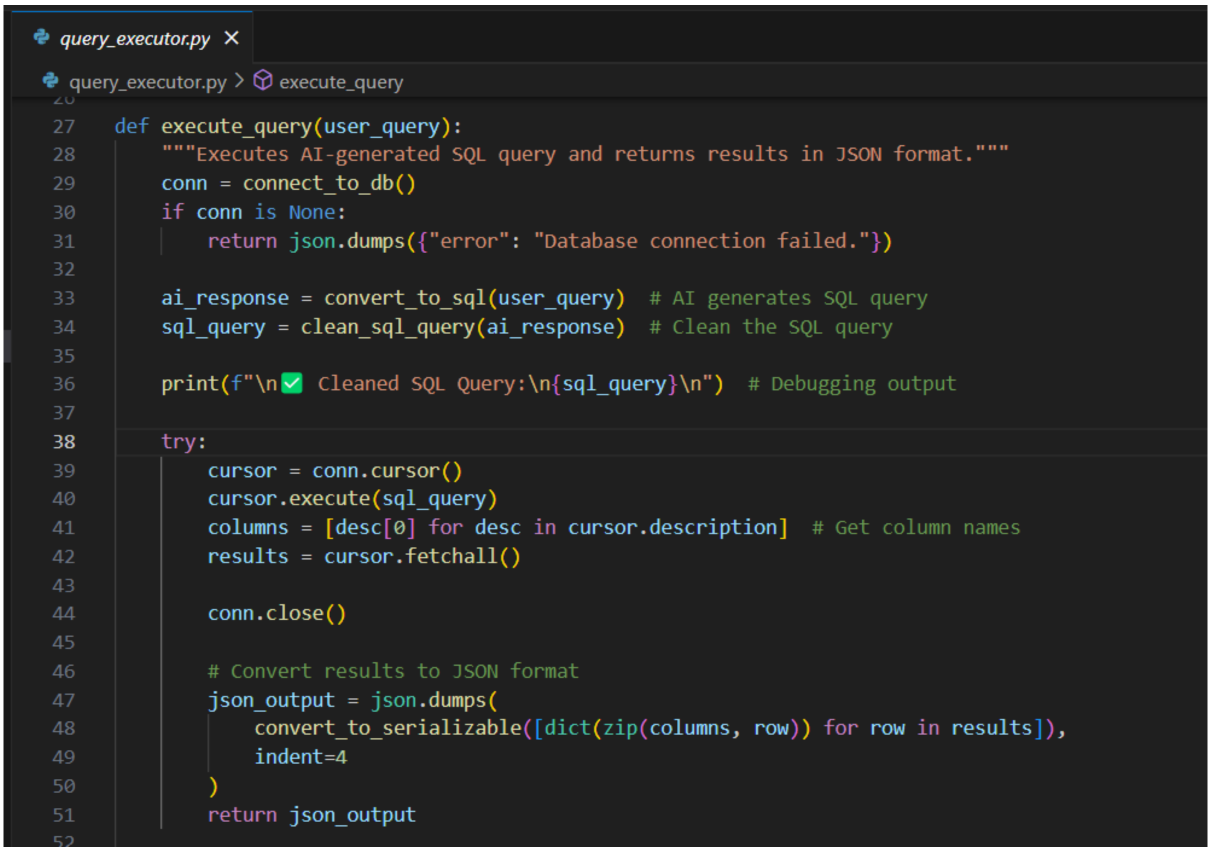
3. Query Execution (query_executor.py)



query_executor.py > execute_query

```python
from database import connect_to_db
from ai_sql import convert_to_sql
import re
import json
from decimal import Decimal
from datetime import date  # Import date module

def clean_sql_query(ai_response):
    """Cleans AI-generated SQL query by removing unnecessary characters."""
    sql_query = re.sub(r"```sql|```", "", ai_response, flags=re.MULTILINE)  # Remove
    sql_query = re.sub(r"--.*", "", sql_query)  # Remove inline comments
    sql_query = sql_query.strip().split("\n\n")[0]  # Keep only first SQL statement
    return sql_query.strip()

def convert_to_serializable(data):
    """Recursively converts Decimal and date values to JSON serializable types."""
    if isinstance(data, list):
        return [convert_to_serializable(item) for item in data]
    elif isinstance(data, dict):
        return {key: convert_to_serializable(value) for key, value in data.items()}
    elif isinstance(data, Decimal):
        return float(data)  # Convert Decimal to float
    elif isinstance(data, date):
        return data.isoformat()  # Convert date to string (YYYY-MM-DD)
    return data
```

Name: Jayshree Patil



query_executor.py

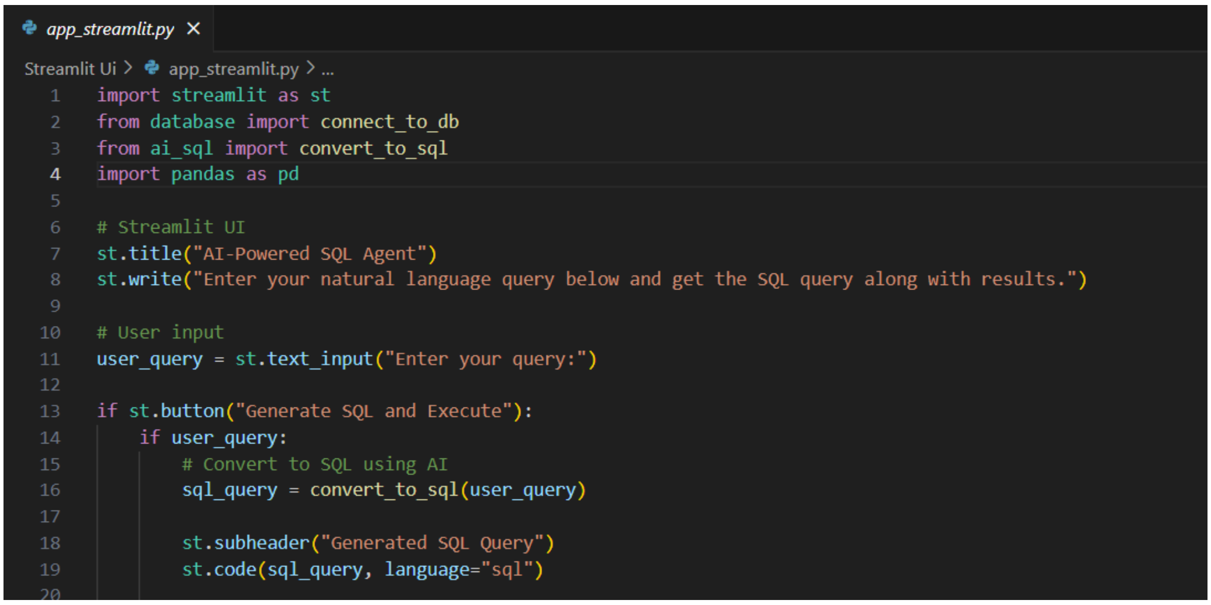
query_executor.py > execute_query

```python
def execute_query(user_query):
    """Executes AI-generated SQL query and returns results in JSON format."""
    conn = connect_to_db()
    if conn is None:
        return json.dumps({"error": "Database connection failed."})

    ai_response = convert_to_sql(user_query)  # AI generates SQL query
    sql_query = clean_sql_query(ai_response)  # Clean the SQL query

    print(f"\n✅ Cleaned SQL Query:\n{sql_query}\n")  # Debugging output

    try:
        cursor = conn.cursor()
        cursor.execute(sql_query)
        columns = [desc[0] for desc in cursor.description]  # Get column names
        results = cursor.fetchall()

        conn.close()

        # Convert results to JSON format
        json_output = json.dumps(
            convert_to_serializable([dict(zip(columns, row)) for row in results]),
            indent=4
        )
        return json_output
```

- **Executes AI-generated SQL queries** on the database.
- **Handles errors** and returns structured JSON output.

4.User Interface (Streamlit UI)

Provides a simple UI for query input and execution.



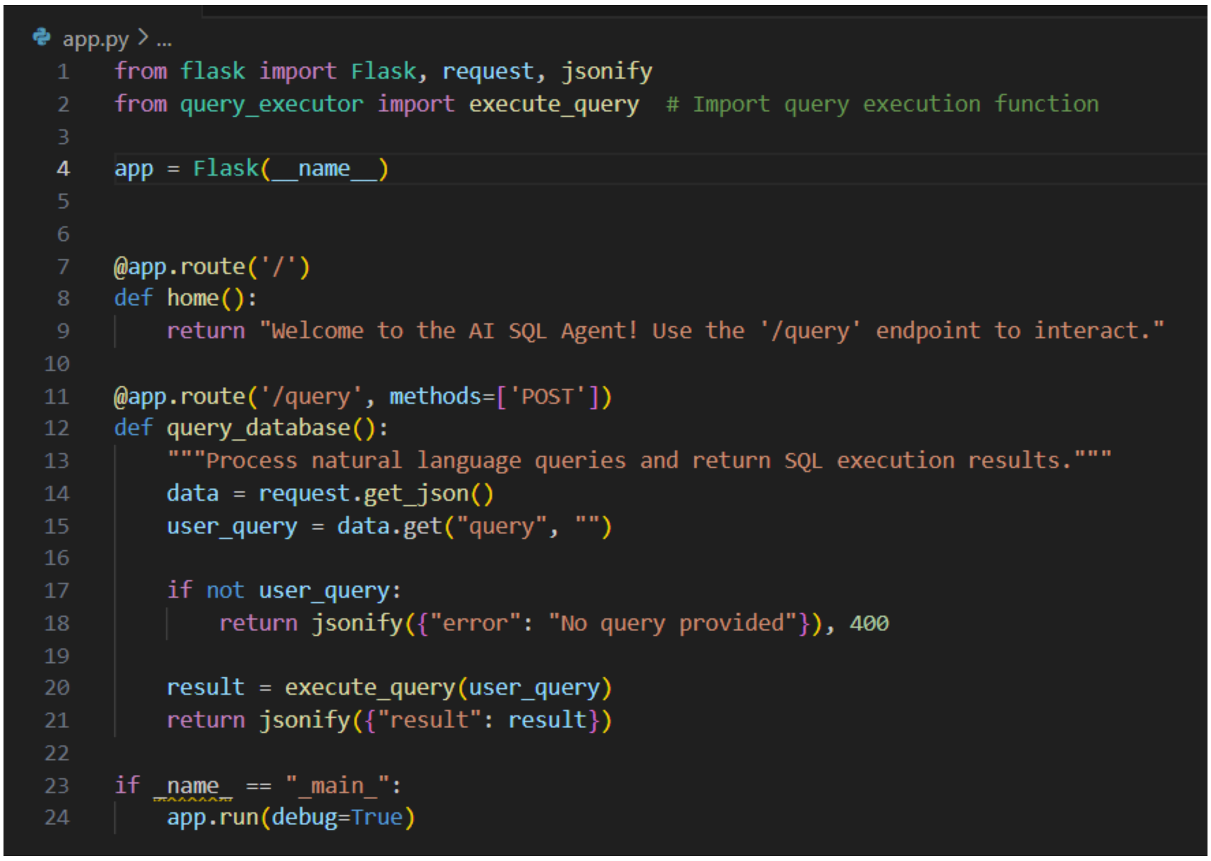
app_streamlit.py

Streamlit Ui > app_streamlit.py > ...

```python
import streamlit as st
from database import connect_to_db
from ai_sql import convert_to_sql
import pandas as pd

# Streamlit UI
st.title("AI-Powered SQL Agent")
st.write("Enter your natural language query below and get the SQL query along with results.")

# User input
user_query = st.text_input("Enter your query:")

if st.button("Generate SQL and Execute"):
    if user_query:
        # Convert to SQL using AI
        sql_query = convert_to_sql(user_query)

        st.subheader("Generated SQL Query")
        st.code(sql_query, language="sql")
```

Name: Jayshree Patil

5. API Endpoint (app.py - Flask API)



app.py > ...

```python
from flask import Flask, request, jsonify
from query_executor import execute_query  # Import query execution function

app = Flask(__name__)


@app.route('/')
def home():
    return "Welcome to the AI SQL Agent! Use the '/query' endpoint to interact."

@app.route('/query', methods=['POST'])
def query_database():
    """Process natural language queries and return SQL execution results."""
    data = request.get_json()
    user_query = data.get("query", "")

    if not user_query:
        return jsonify({"error": "No query provided"}), 400

    result = execute_query(user_query)
    return jsonify({"result": result})

if _name_ == "_main_":
    app.run(debug=True)
```

Flask API to handle user queries and return results.

6. Testing

a) Run the API:

Using CURL:

curl -X POST http://127.0.0.1:5000/query -H "Content-Type: application/json" -d '{"query": "Show all employees in Mumbai."}'

Using Postman:

1. Open Postman
2. Set request type to POST
3. URL: http://127.0.0.1:5000/query
4. Go to Body → raw → JSON
5. Input:

{

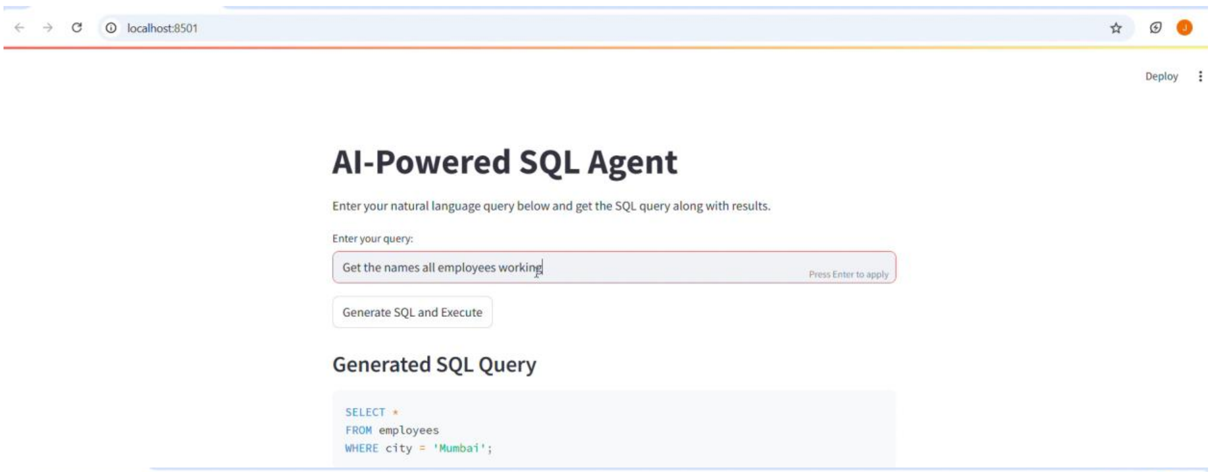
"query": "List all employees in the IT department.}

Name: Jayshree Patil

b) Run Streamlit UI:

streamlit run app_streamlit.py

Output:-



AI-Powered SQL Agent

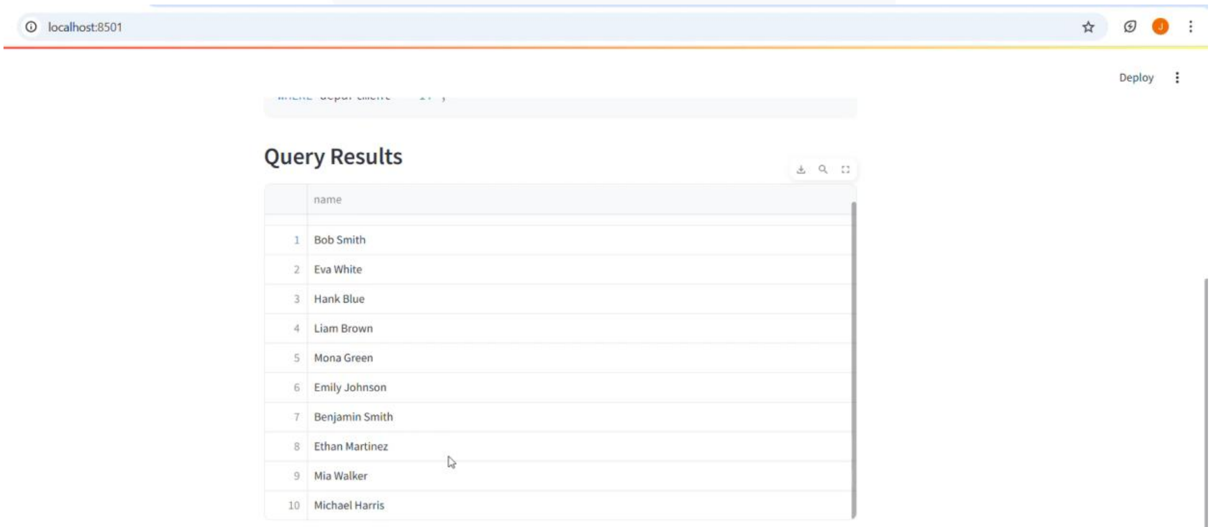
Enter your natural language query below and get the SQL query along with results.

Enter your query:

Get the names all employees working

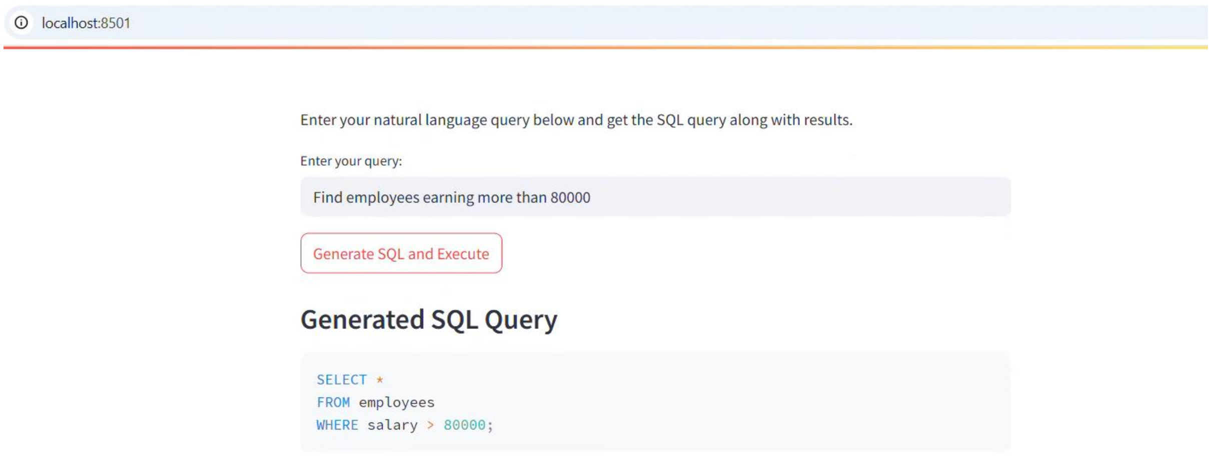
Generated SQL Query

```sql
SELECT *
FROM employees
WHERE city = 'Mumbai';
```



Query Results

	name
1	Bob Smith
2	Eva White
3	Hank Blue
4	Liam Brown
5	Mona Green
6	Emily Johnson
7	Benjamin Smith
8	Ethan Martinez
9	Mia Walker
10	Michael Harris



Enter your natural language query below and get the SQL query along with results.

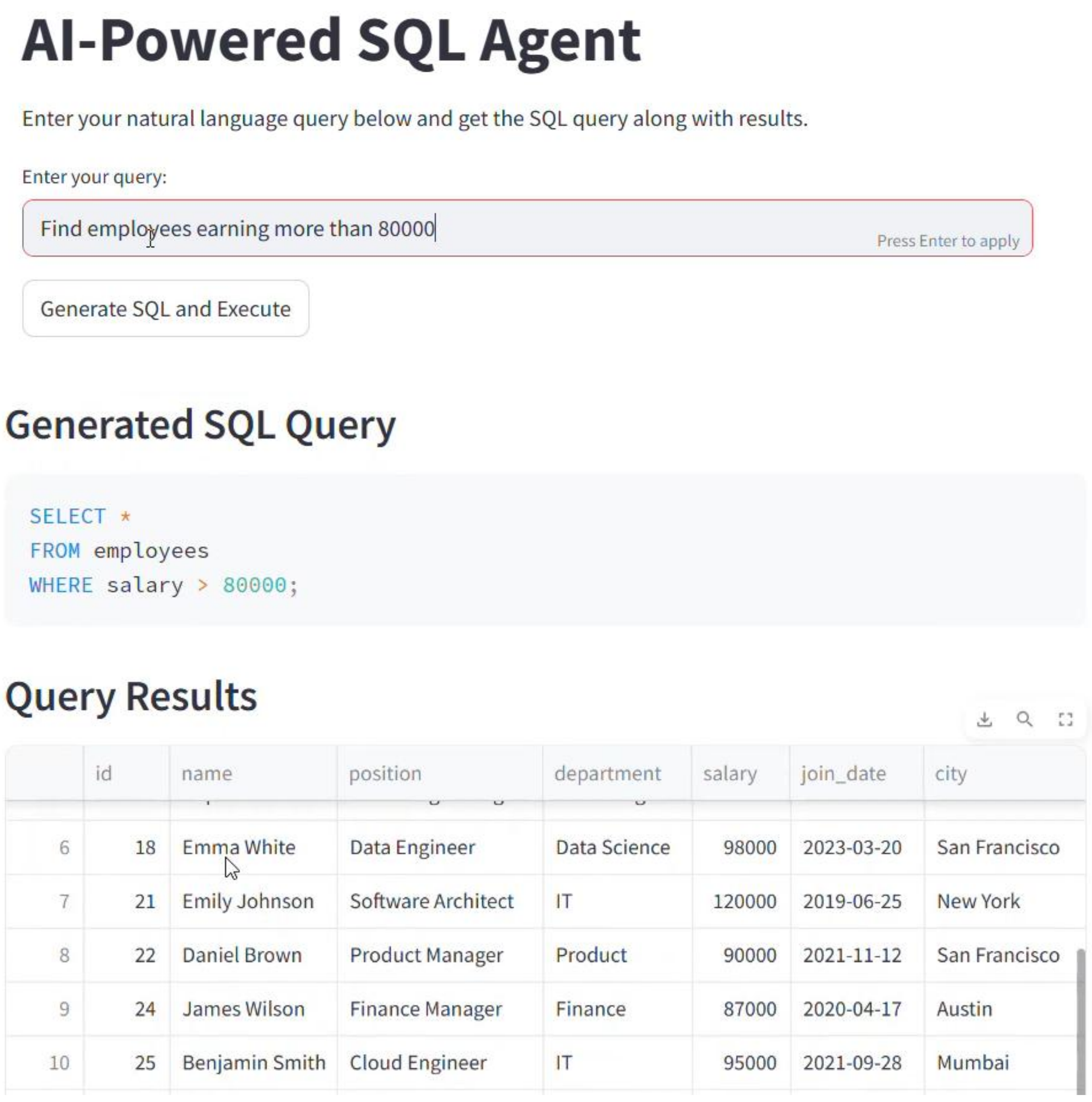
Enter your query:

Find employees earning more than 80000

Generated SQL Query

```sql
SELECT *
FROM employees
WHERE salary > 80000;
```

Name: Jayshree Patil



AI-Powered SQL Agent

Enter your natural language query below and get the SQL query along with results.

Enter your query:

Find employees earning more than 80000

Generated SQL Query

```
SELECT *
FROM employees
WHERE salary > 80000;
```

Query Results

	id	name	position	department	salary	join_date	city
6	18	Emma White	Data Engineer	Data Science	98000	2023-03-20	San Francisco
7	21	Emily Johnson	Software Architect	IT	120000	2019-06-25	New York
8	22	Daniel Brown	Product Manager	Product	90000	2021-11-12	San Francisco
9	24	James Wilson	Finance Manager	Finance	87000	2020-04-17	Austin
10	25	Benjamin Smith	Cloud Engineer	IT	95000	2021-09-28	Mumbai

7. Conclusion

- Successfully developed an AI-powered SQL Agent.
- Allows users to interact with databases without SQL knowledge.
- Implements Google Gemini AI for natural language to SQL conversion.
- Provides Flask API for backend and Streamlit UI for user interaction.
- Ensures security with query validation and SQL injection protection.