IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro

MENTOR: **K. RATNA KUMARI MADAM**

TEAM ID: **LTVIP2026TMIDS38632**

TEAM LEADER: ***ADAPA DURGA BHAVANI***

Roll number: SBAP0032712 durgabhavani2469.adapa@gmail.com

TEAM MEMBERS:

1) REKHA DEVI

Roll number: SBAP0032720 devi5530750@gmail.com

2) NIDADAVOLU CHILOCHANA LAKSHMI BHAVANI

Roll number: SBAP003272 bhavaninidadhavolunidadhavolu@gmail.com

3) MADIMI VANDANA

Roll number: SBAP0032732 vandanamadimi@gmail.com

INTRODUCTION

IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro

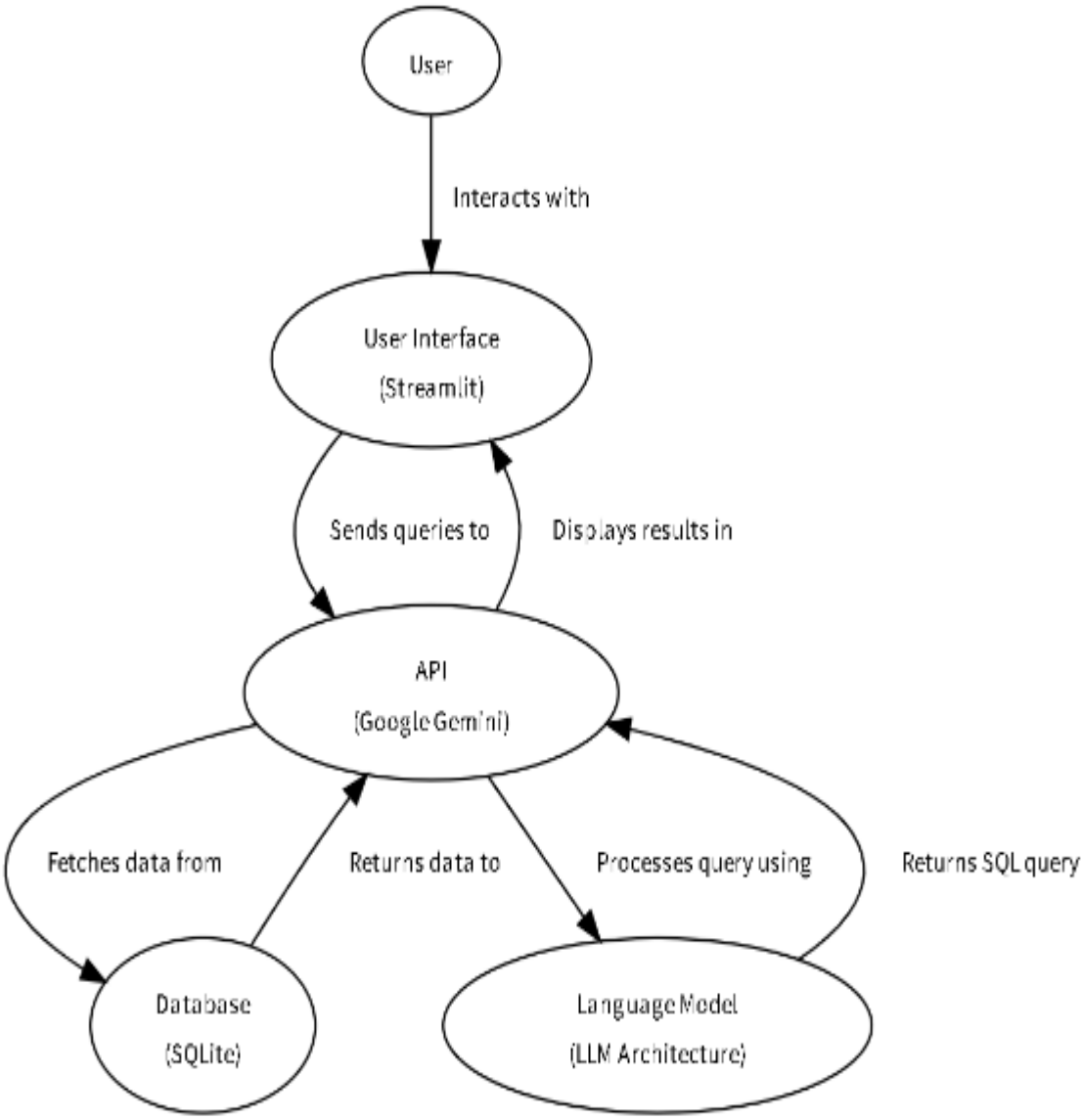
Scenario 1: Intelligent Query Assistance

IntelliSQL enhances the querying process by providing intelligent assistance to users. Whether they are novice or experienced SQL users, IntelliSQL guides them through crafting complex queries with ease. By understanding natural language queries, IntelliSQL suggests relevant SQL statements, offers syntax suggestions, and assists in optimizing query performance, thereby streamlining the database interaction experience.

Scenario 2: Data Exploration and Insights

For data analysts and researchers, IntelliSQL serves as a powerful tool for exploring and gaining insights from databases. Users can pose natural language questions about the data, and IntelliSQL intelligently translates these queries into SQL commands to retrieve the desired information. With its advanced language understanding capabilities, IntelliSQL facilitates efficient data exploration, enabling users to uncover hidden patterns, trends, and insights within the database.

Architecture:



Prior Knowledge

You must have the prior knowledge of the following topics to complete this project.

Generative AI Concepts

NLP: https://www.tutorialspoint.com/natural_language_processing/index.htm

Generative AI: https://en.wikipedia.org/wiki/Generative_artificial_intelligence

About Gemini: https://deepmind.google/technologies/gemini/#introduction

Gemini API: https://ai.google.dev/gemini-api/docs/get-started/python

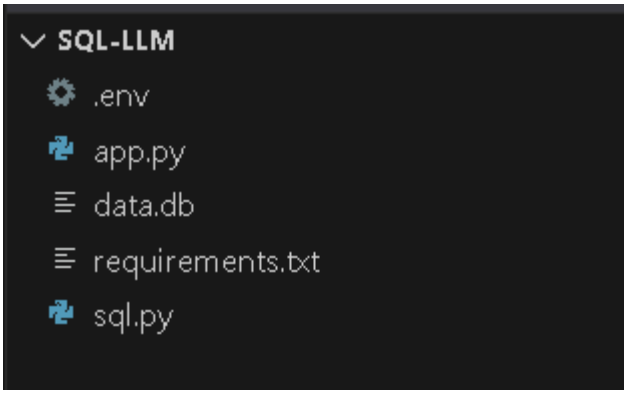
Gemini Demo: https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb

Streamlit: https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Sqlite3: https://www.sqlite.org/docs.html

Project Structure

Create the Project folder which contains files as shown below:

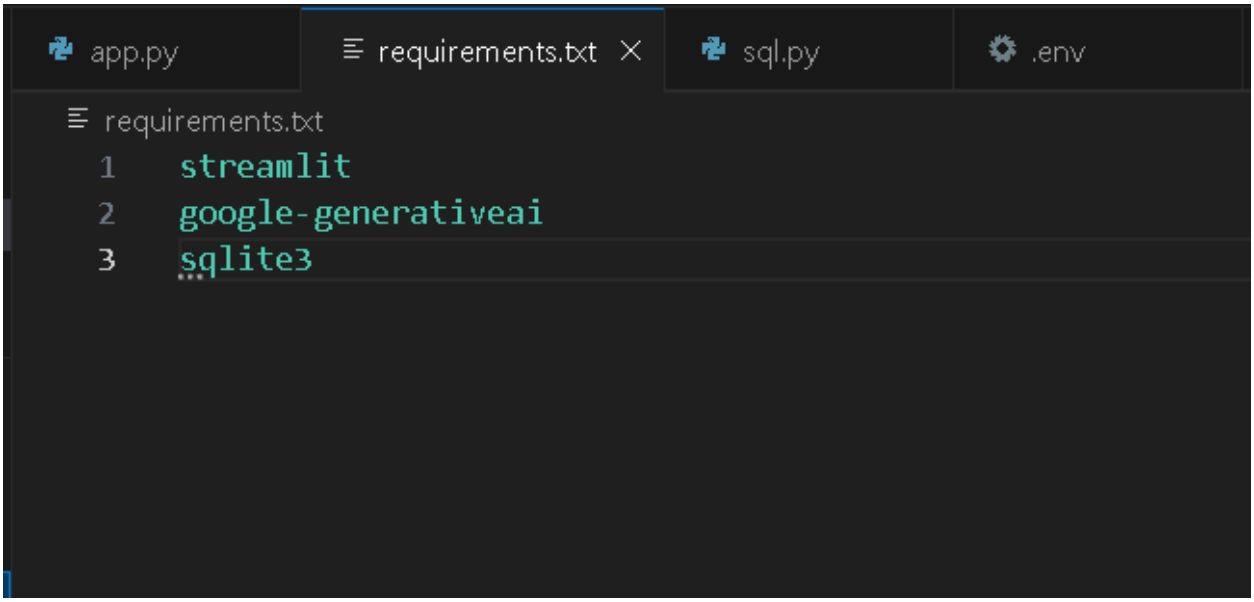


- sql.py: It is used to create a database for our demonstration. This file can be created by following milestone 3.
- .env file: It securely stores the Google API key.
- app.py: It serves as the primary application file housing both the model and Streamlit UI code.
- requirements.txt: It enumerates the libraries necessary for installation to ensure proper functioning.
- data.db: It is the database created using sql.py
- Additionally, ensure proper file organization and adhere to best practices for version contr

Milestone 1 : Requirements Specification

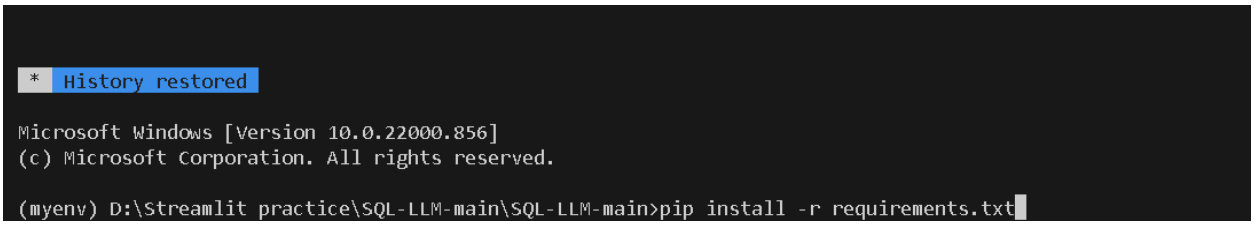
Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

Activity 1 : Create a requirements.txt file to list the required libraries



- streamlit: An open-source app framework used to create and share custom web applications for machine learning and data science projects with minimal code.
- google-generativeai: A Python client library for interacting with Google's Generative AI models, enabling the generation of text, images, and other media based on trained AI models.
- sqlite3: A lightweight, disk-based database engine that doesn't require a separate server process, used for managing databases with SQL in Python applications

Activity 2 : Install the required libraries.



- Open the terminal.
- Activate the environment
- ‘myenv’ is the name of the environment
- Run the command: pip install -r requirements.txt
- This command installs all the libraries listed in the requirements.txt file.

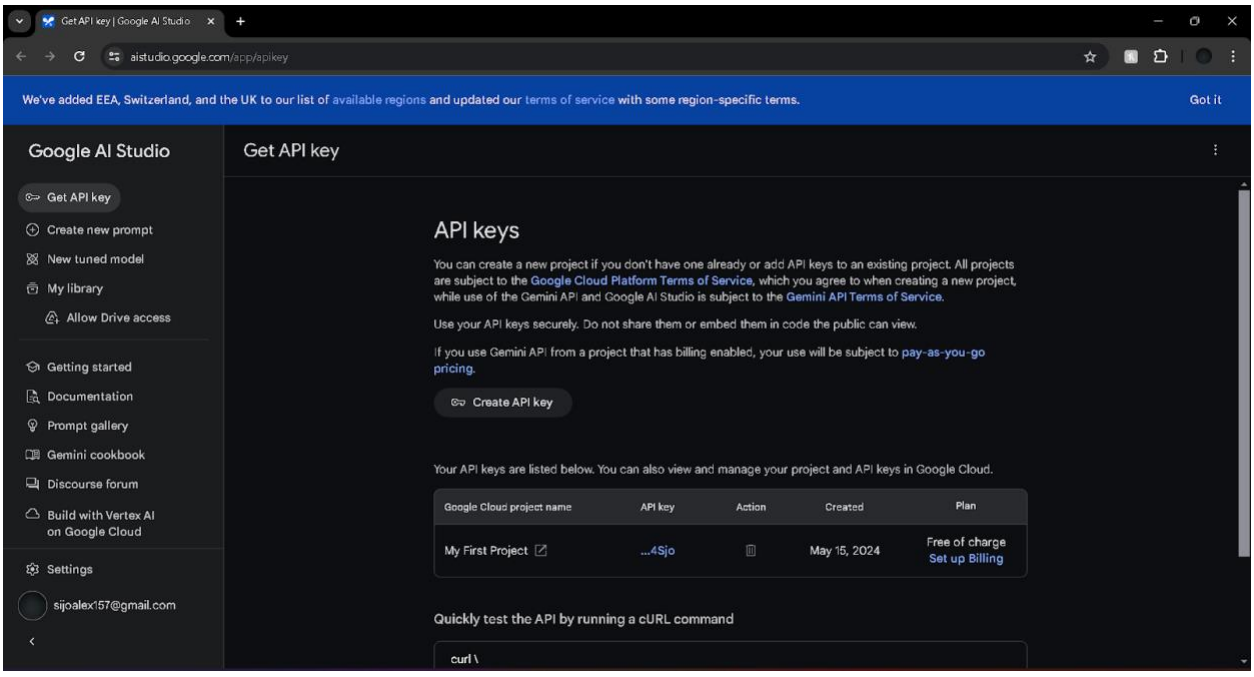
Milestone 2 : Initialization of Google API Key

The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs. It acts as a form of identification, allowing users to access specific Google services and resources. This key plays a crucial role in authorizing and securing API requests, ensuring that only authorized users can access and utilize Google’s services.

Activity 1 : Generate Google API Key

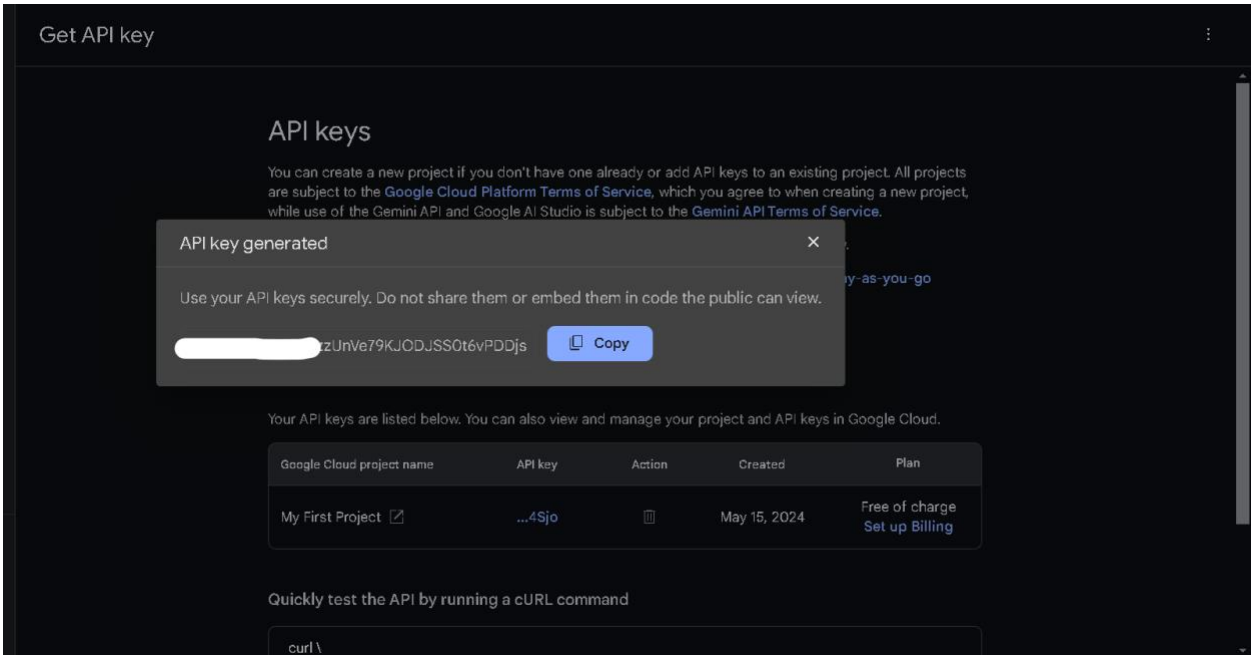
Click the provided link to access the following webpage.

Link: https://aistudio.google.com/app/apikey



After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another webpage as shown below.

Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.



Copy the newly generated API key as it is required for loading the Gemini Pro pre-trained model.

Activity 2 : Initialize Google API Key

```
GOOGLE_API_KEY = "<Enter the copied Google API Key>"
```

- Create a .env file and define a variable named GOOGLE_API_KEY.
- Assign the copied Google API key to this variable.
- Paste the API key obtained from the previous steps here.

Milestone 3 : Database creation using sqlite3

In this milestone, we establish a connection to a SQLite3 database, create a table to store student information, and insert multiple records into this table to demonstrate the database creation and data insertion process.

Activity 1 : Establishing Connection and Creating the Table

In this activity, we establish a connection to a SQLite3 database, create a cursor object to execute SQL commands, and define the schema for the Students table.

```
import sqlite3

connection = sqlite3.connect("data.db")

cursor = connection.cursor()

table = '''
CREATE TABLE Students(name VARCHAR(30), class VARCHAR(10), marks INT, company VARCHAR(30))
'''

cursor.execute(table)
```

- Importing SQLite3: We import the sqlite3 library to work with the SQLite database.
- Establishing a Connection: The sqlite3.connect("data.db") command creates a connection object to a SQLite database file named data.db. If the file does not exist, it will be created.
- Creating a Cursor: The connection.cursor() command creates a cursor object. Cursors allow us to execute SQL commands and retrieve data from the database.

- Defining the Table Schema: The table variable holds a SQL command that defines the schema for a table named Students, with columns for name, class, marks, and company.
- Executing the Table Creation Command: The cursor.execute(table) command executes the SQL command stored in the table variable, creating the Students table in the database.

Activity 2: Inserting Records and Querying Data

This activity involves inserting multiple records into the Students table, querying the table to retrieve all records, and displaying the data.

```
13  cursor.execute('''insert into Students values('Sijo', 'BTech', 75, 'JSW')''')
14  cursor.execute('''insert into Students values('Lijo', 'MTech', 69, 'TCS')''')
15  cursor.execute('''insert into Students values('Rijo', 'BSc', 79, 'WIPRO')''')
16  cursor.execute('''insert into Students values('Sibin', 'MSc', 89, 'INFOSYS')''')
17  cursor.execute('''insert into Students values('Dilsha', 'MCom', 99, 'Cyient')''')
18
19  print("the inserted records-")
20  df = cursor.execute('''select * from Students''')
21
22  for row in df:
23      print(row)
```

- Inserting Records:
- Each cursor.execute command inserts a new record into the Students table with specified values for name, class, marks, and company.
- Example: cursor.execute('''insert into Students values('Sijo', 'BTech', 75, 'JSW')''') inserts a record with 'Sijo' as the name, 'BTech' as the class, 75 as the marks, and 'JSW' as the company.
- Querying the Table:
- The df = cursor.execute('''select * from Students''') command retrieves all records from the Students table.
- The for loop iterates over the result set df, and each row is printed to the console using print(row).

Activity 3 : Committing Changes and Closing the Connection

In this final activity, we commit the changes to the database and close the connection to ensure all resources are properly released.

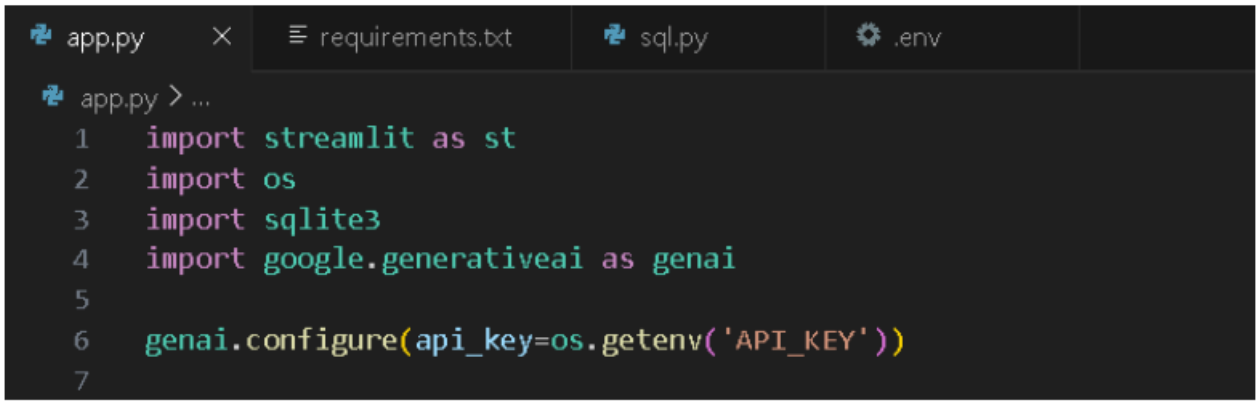
```
24
25  connection.commit()
26  connection.close()
27
```

- Committing the Transaction: The connection.commit() command saves all the changes made to the database during the session.
- Closing the Connection: The connection.close() command closes the connection to the database, ensuring that all resources are released.

Milestone 4 : Interfacing with Pre-trained Model

To interface with the pre-trained model, we'll start by creating an app.py file, which will contain both the model and Streamlit UI code.

Activity 1 : Load the Gemini Pro pre-trained model



- Imports the Streamlit library, which is used to create interactive web applications for machine learning and data science.
- Imports the os module, which provides a way of using operating system dependent functionality like reading or writing to the file system, and accessing environment variables.
- Imports the sqlite3 module, which provides a lightweight disk-based database engine. It allows for the creation and management of SQLite databases in Python applications.
- Imports the google.generativeai module with an alias genai. This module is used for interacting with Google's Generative AI models, which can generate content based on AI algorithms.
- Configures the Google Generative AI client with an API key.
- os.getenv('API_KEY') retrieves the API key from the environment variable named API_KEY.
- This setup is necessary to authenticate and authorize the requests made to Google's Generative AI services.

Activity 2 : Prompt Configuration for Converting English Questions to SQL Queries

```
prompt = [
    """
    You are an expert in converting English questions to SQL query!
    The SQL database has the name STUDENTS and has the following columns - NAME, CLASS,
    Marks, Company \n\nFor example,\nExample 1 - How many entries of records are present?,
    the SQL command will be something like this: SELECT COUNT(*) FROM STUDENTS;
    \nExample 2 - Tell me all the students studying in MCom class?,
    the SQL command will be something like this: SELECT * FROM STUDENTS
    WHERE CLASS="MCom";

    """
]
```

prompt = [...]

- Defines a list named prompt containing a single multi-line string. This string provides instructions to the AI model on how to convert English questions into SQL queries.

Instruction Details:

- Role Description: Instructs the AI that it is an expert in converting English questions to SQL queries.
- Database Information: Specifies that the SQL database is named STUDENTS and includes columns NAME, CLASS, Marks, and Company.

Examples:

- Example 1: Converts the question "How many entries of records are present?" to the SQL query SELECT COUNT(*) FROM STUDENTS;.
- Example 2: Converts the question "Tell me all the students studying in MCom class?" to the SQL query SELECT * FROM STUDENTS WHERE CLASS="MCom";.

Activity 3: Function Definition for Generating SQL Queries from Natural Language Questions

```
20
21  def get_response(que, prompt):
22      model = genai.GenerativeModel("gemini-pro")
23      response = model.generate_content([prompt[0], que])
24      return response.text
25
```

def get_response(que, prompt):

- Defines a function named get_response that takes two parameters: que (the query in natural language) and prompt (the list containing the instructional prompt).

model = genai.GenerativeModel("gemini-pro")

- Creates an instance of the generative model from the google.generativeai package, specifying the model name "gemini-pro".

response = model.generate_content([prompt[0], que])

- Calls the generate_content method on the model instance, passing a list containing the instructional prompt (prompt[0]) and the user's query (que).
- Stores the generated response in the variable response.

return response.text

- Returns the textual content of the generated response.

Activity 4 : Function to Read SQL Query Results

```
26  def read_query(sql, db):
27      conn = sqlite3.connect(db)
28      cursor = conn.cursor()
29      cursor.execute(sql)
30      rows = cursor.fetchall()
31      conn.commit()
32      conn.close()
33      return rows
34
```

- Purpose: Reads results from an SQL query.
- Parameters:
    - sql: SQL query to execute.
    - db: Database file path.
- Functionality:
    - Connects to the specified SQLite database.
    - Creates a cursor object to interact with the database.
    - Executes the provided SQL query.
    - Fetches all the rows returned by the query.
    - Commits any pending transaction to the database.
    - Closes the connection to the database.
    - Returns the fetched rows.

Milestone 5 : Model Deployment

We deploy our model using the Streamlit framework, a powerful tool for building and sharing data applications quickly and easily. With Streamlit, we can create interactive web applications that allow users to interact with our models in real-time, providing an intuitive and seamless experience.

Activity 1: Integrate with Web Framework

The webpage is organized into three main sections to provide users with a comprehensive experience:

- Function to Render Home Page

```
def page_home():
    st.markdown("""
        <style>
            body {
                background-color: #2E2E2E;
            }
            .main-title {
                text-align: center;
                color: #4CAF50; /* Green color for headings */
                font-size: 2.5em;
            }
            .sub-title {
                text-align: center;
                color: #4CAF50; /* Green color for headings */
                font-size: 1.5em;
            }
            .offerings {
                padding: 20px;
                color: white; /* White color for body text */
            }
            .offerings h2 {
                color: #4CAF50; /* Green color for headings */
            }
            .offerings ul {
                list-style-type: none;
                padding: 0;
            }
            .offerings li {
                margin: 10px 0;
                font-size: 18px;
            }
            .custom-sidebar {
                background-color: #2E2E2E;
                color: white;
            }
        </style>
    """, unsafe_allow_html=True)

    st.markdown("<h1 class='main-title'>Welcome to IntelliSQL!</h1>", unsafe_allow_html=True)
    st.markdown("<h2 class='sub-title'>Revolutionizing Database Querying with Advanced LLM Capabilities</h2>", unsafe_allow_html=Tru

    col1, col2 = st.columns([1, 1])

    with col1:
        st.image("https://cdn1.iconfinder.com/data/icons/business-dual-color-glyph-set-3/128/Data_warehouse-1024.png", use_column_wi

    with col2:
        st.markdown("""
        <div class='offerings'>
            <h2>Wide Range of Offerings</h2>
            <ul>
                <li>💡 Intelligent Query Assistance</li>
                <li>🔍 Data Exploration and Insights</li>
                <li>📊 Efficient Data Retrieval</li>
                <li>🚀 Performance Optimization</li>
                <li>🔧 Syntax Suggestions</li>
                <li>📈 Trend Analysis</li>
            </ul>
        </div>
        """, unsafe_allow_html=True)
```

- Purpose: Renders the home page of IntelliSQL.

- Function Name: page_home()
- Functionality:
    - Applies custom CSS styles to the page for visual enhancement.
    - Sets background color to dark gray (#2E2E2E).
    - Defines CSS classes for different elements like titles, offerings, and sidebar.
    - Renders a main title "Welcome to IntelliSQL!" in green color and centered.
    - Renders a sub-title "Revolutionizing Database Querying with Advanced LLM Capabilities" in green color and centered.
    - Divides the layout into two columns using Streamlit's columns function.
    - In the first column (col1):
    - Displays an image of a data warehouse icon.
    - In the second column (col2):
    - Renders a list of offerings using HTML markup, including intelligent query assistance, data exploration, efficient data retrieval, performance optimization, syntax suggestions, and trend analysis.
    - Utilizes unsafe_allow_html=True parameter in st.markdown to allow rendering HTML content within Streamlit.

- Function to Render About Page:

```
95
96  def page_about():
97      st.markdown("""
98          <style>
99              .content {
100                 color: white; /* White color for body text */
101             }
102         </style>
103     """, unsafe_allow_html=True)
104     st.markdown("<h1 style='color: #4CAF50;'>About IntelliSQL</h1>", unsafe_allow_html=True)
105     st.markdown("<div class='content'>", unsafe_allow_html=True)
106     st.markdown("""
107     <h2>IntelliSQL is an innovative project aimed at revolutionizing database querying using advanced Language Model capabilities.
108     """, unsafe_allow_html=True)
109     st.markdown("</div>", unsafe_allow_html=True)
110
111     st.image("https://download.logo.wine/logo/Oracle_SQL_Developer/Oracle_SQL_Developer-Logo.wine.png", use_column_width=True)
112
```

- Purpose: Renders the about page of IntelliSQL.
- Function Name: page_about()
- Functionality:
    - Applies custom CSS styles to the page.
    - Sets text color to white for elements with the content class.
    - Renders the main title "About IntelliSQL" in green color.
    - Wraps the main content in a div with the content class.
    - Describes the IntelliSQL project:
    - Emphasizes its goal to revolutionize database querying.
    - Highlights the use of advanced Large Language Model (LLM) architecture.
    - Mentions the platform's intelligent and intuitive interaction with SQL databases.
    - Displays an image of the Oracle SQL Developer logo using Streamlit's st.image function.

- Function to Render Intelligent Query Assistance Page:

```
def page_intelligent_query_assistance():
    st.markdown("""
        <style>
            .tool-input {
                margin-bottom: 20px;
                color: white; /* White color for body text */
            }
            .response {
                margin-top: 20px;
                color: white; /* White color for body text */
            }
        </style>
    """, unsafe_allow_html=True)

    st.markdown("<h1 style='color: #4CAF50;'>Intelligent Query Assistance</h1>", unsafe_allow_html=True)
    st.write("""
    IntelliSQL enhances the querying process by providing intelligent assistance to users. Whether they are novice or experienced S
    """)

    col1, col2 = st.columns([2, 1])
```

```
    with col1:
        st.markdown("<div class='tool-input'>", unsafe_allow_html=True)
        que = st.text_input("Enter Your Query:", key="sql_query")
        submit = st.button("Get Answer", key="submit_button", help="Click to retrieve the SQL data")
        st.markdown("</div>", unsafe_allow_html=True)

        if submit or que:
            try:
                response = get_response(que, prompt)
                st.write(f"**Generated SQL Query:** `{response}`")
                response = read_query(response, "data.db")
                st.markdown("<div class='response'>", unsafe_allow_html=True)
                st.subheader("The Response is:")
                st.table(response)
                st.markdown("</div>", unsafe_allow_html=True)
            except Exception as e:
                st.subheader("Error:")
                st.error(f"An error occurred: {e}")

    with col2:
        st.image("https://cdn-icons-png.flaticon.com/512/9850/9850877.png", use_column_width=True)
```

- Purpose: Renders the Intelligent Query Assistance page of IntelliSQL.
- Function Name: page_intelligent_query_assistance()
- Functionality:
- Applies custom CSS styles:

    - Adds margins and sets text color to white for elements with the tool-input and response classes.
    - Renders the main title "Intelligent Query Assistance" in green color.
- Describes the IntelliSQL querying process:
    - Explains how IntelliSQL helps users craft complex queries.
    - Mentions support for natural language queries, SQL statement suggestions, syntax help, and performance optimization.
    - Creates a two-column layout using Streamlit's columns function.
- In the first column (col1):
    - Adds a div with the tool-input class for styling.
    - Provides a text input field for the user to enter their SQL query.
    - Adds a button to submit the query.
- If the button is clicked or a query is entered:
- Calls get_response to generate an SQL query based on the user's input.
    - Displays the generated SQL query.
    - Calls read_query to execute the SQL query on a database (data.db).
    - Displays the query results in a table.
    - Handles and displays any errors that occur during the process.
- In the second column (col2):
    - Displays an image related to intelligent querying using Streamlit's st.image function.

- Function to Initialize and Control the IntelliSQL Application:

```
155
156 def main():
157     st.set_page_config(
158         page_title="IntelliSQL",
159         page_icon="🌟",
160         layout="wide"
161     )
162     st.sidebar.title("Navigation")
163     st.sidebar.markdown("<style>.sidebar .sidebar-content {background-color: #2E2E2E; color: white;}</style>", unsafe_allow_html=Tru
164     pages = {
165         "Home": page_home,
166         "About": page_about,
167         "Intelligent Query Assistance": page_intelligent_query_assistance,
168     }
169     selection = st.sidebar.radio("Go to", list(pages.keys()))
170     page = pages[selection]
171     page()
172
173 if __name__ == "__main__":
174     main()
175
```

- Purpose: Sets up the main structure and navigation of the IntelliSQL application.
- Function Name: main()
- Functionality:
    - Page Configuration:
    - Uses st.set_page_config to set up the page title, icon, and layout.
    - Sets the page title to "IntelliSQL".
    - Sets the page icon to a star emoji ("????").
    - Sets the layout to wide.
- Sidebar Setup:
    - Adds a title "Navigation" to the sidebar.

    - Applies custom CSS to style the sidebar with a dark background (#2E2E2E) and white text.

- Page Navigation:
    - Defines a dictionary pages mapping page names to their respective functions:page_home,page_about,page_intelligent_query_assistance
    - Adds a radio button widget to the sidebar for navigation, allowing the user to select a page from the keys of the pages dictionary.
- Page Rendering:
    - Retrieves the selected page from the radio button.
    - Calls the corresponding function to render the selected page.
- Entry Point:
    - Ensures the main() function is called when the script is executed directly by checking if __name__ == "__main__": main().

Activity 2: Host the Application

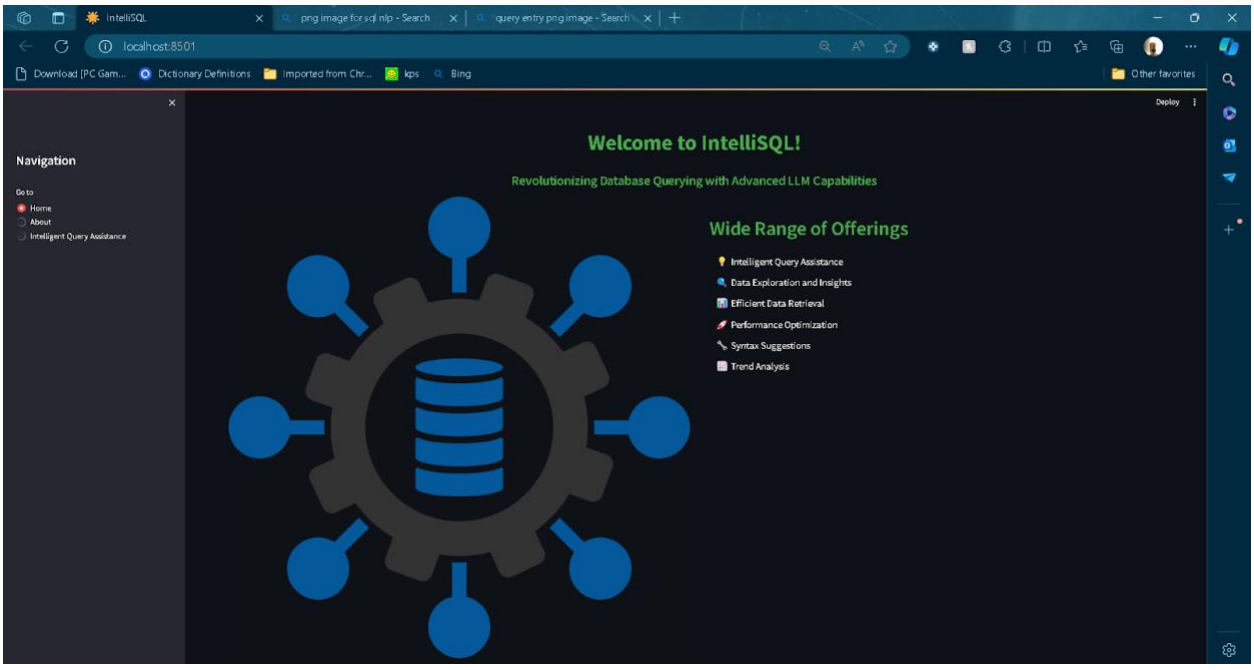
Launching the Application:

- To host the application, go to the terminal, type - streamlit run app.py
- Here app.py refers to a python script.

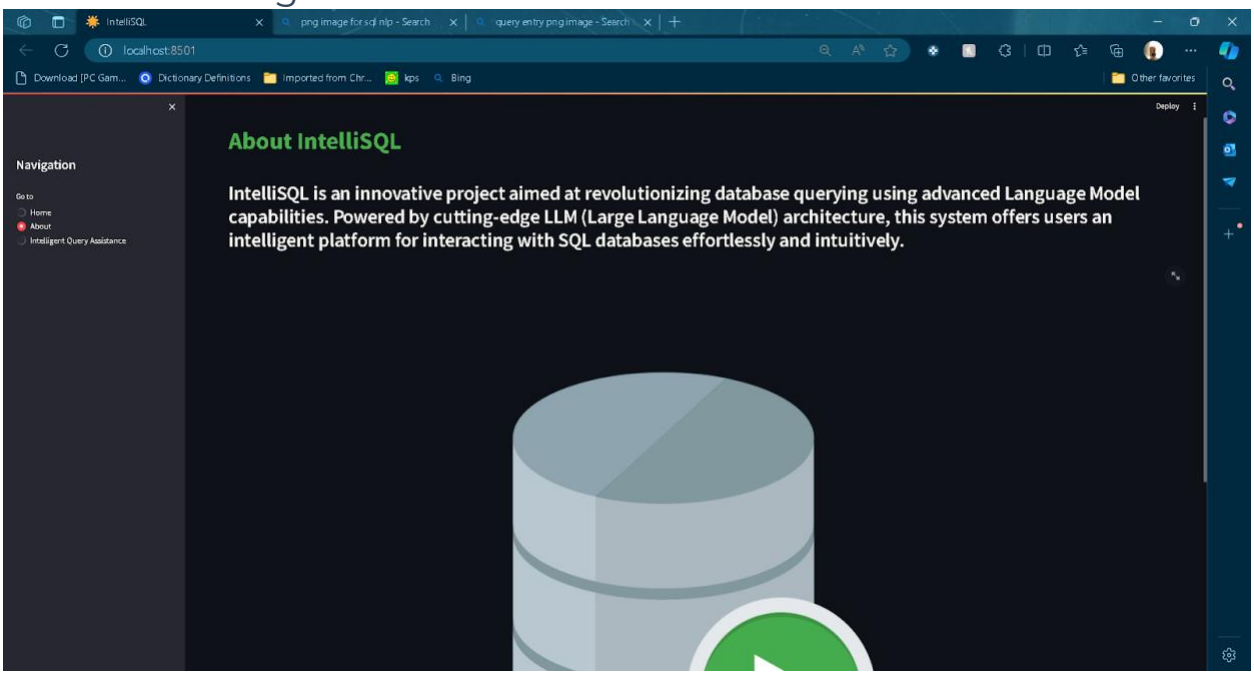
```
(myenv) D:\Streamlit practice\SQL-LLM-main\SQL-LLM-main>streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.0.200:8501
```

Run the command to get the below results

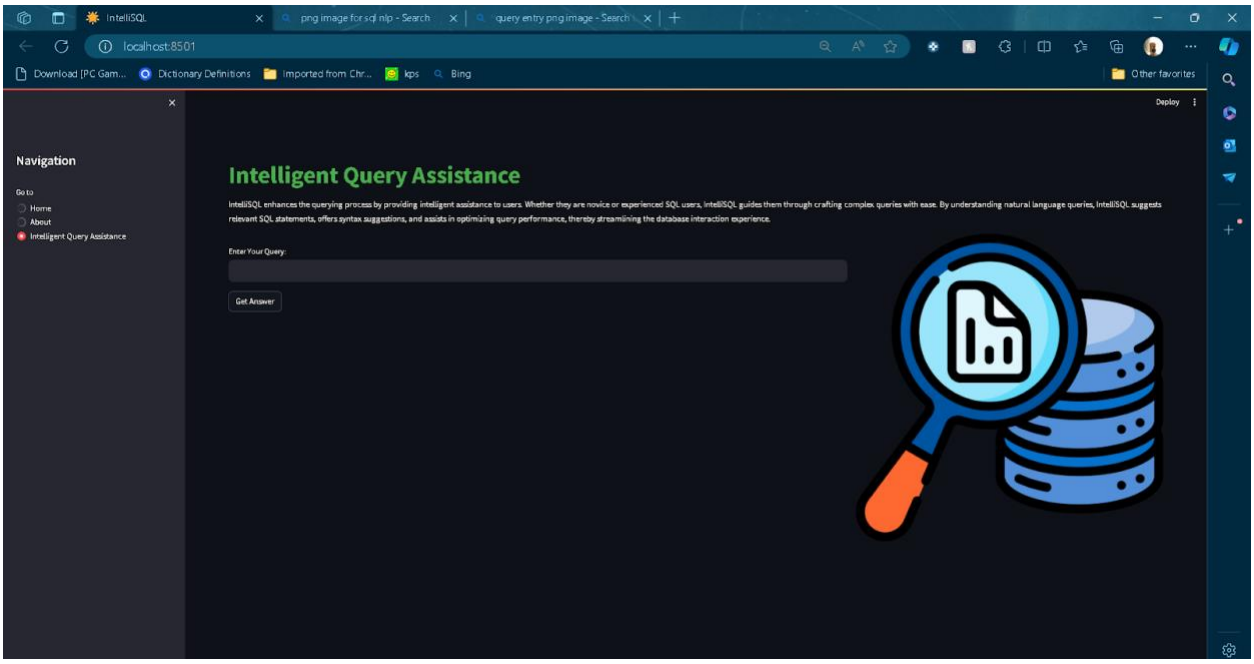
1. Home Page



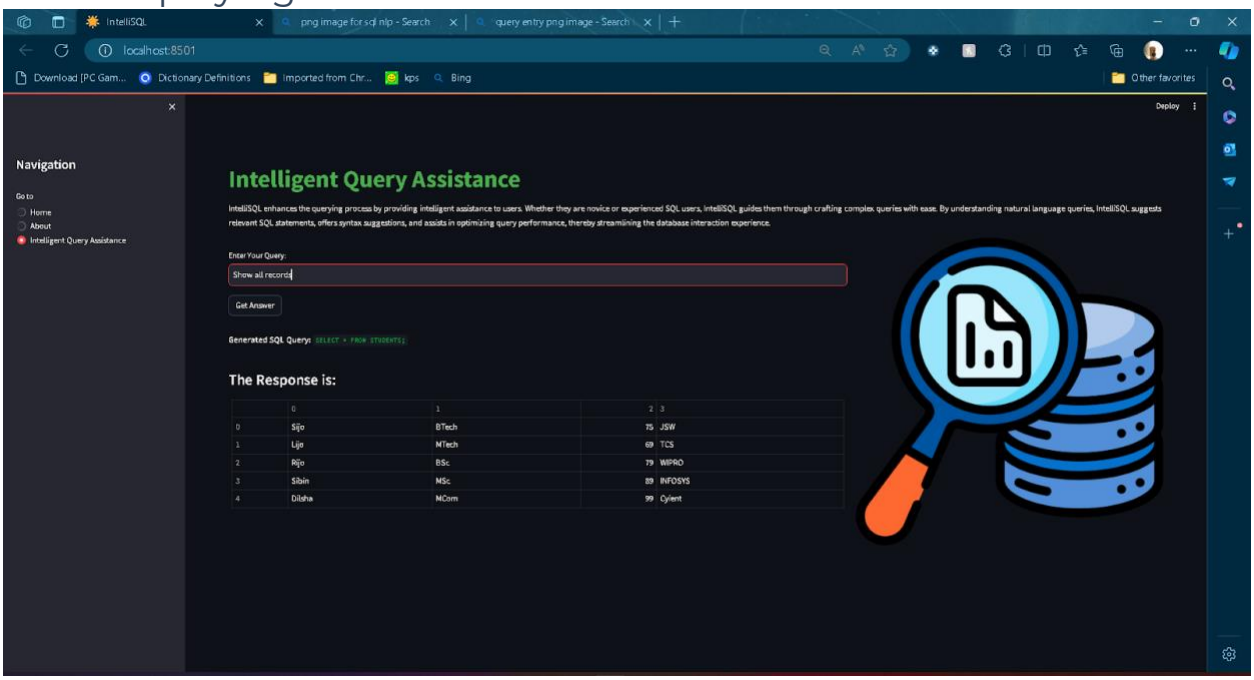
2. About Page



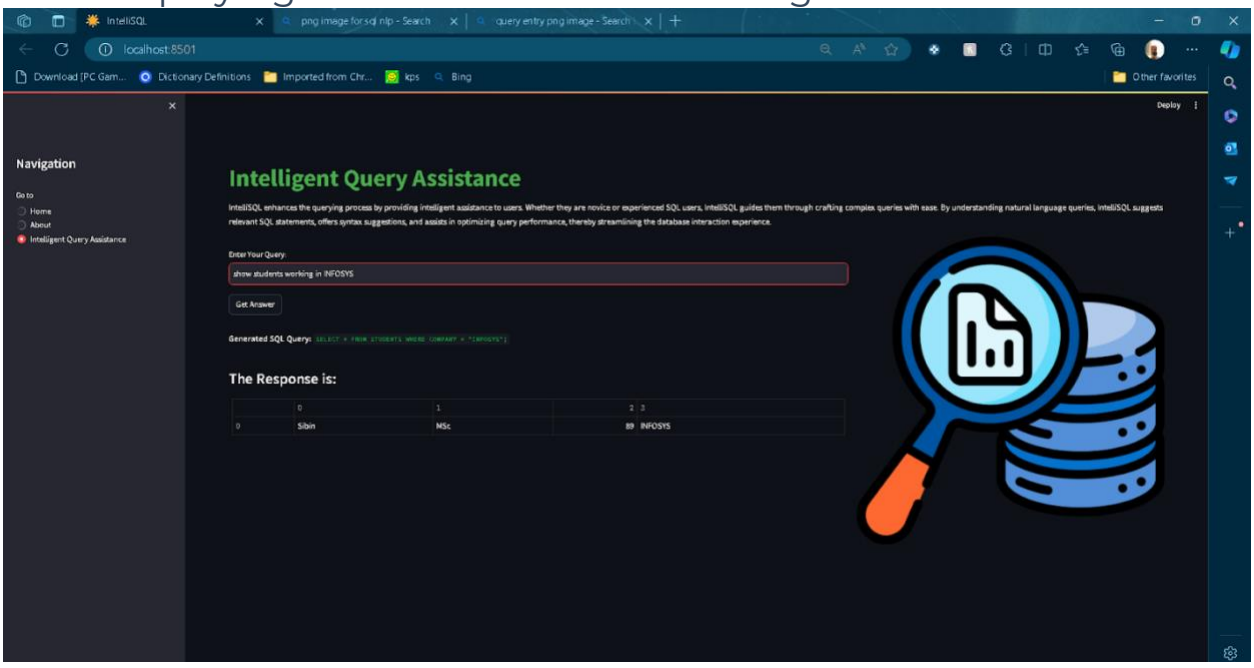
3. Query Conversion page NLP to SQL



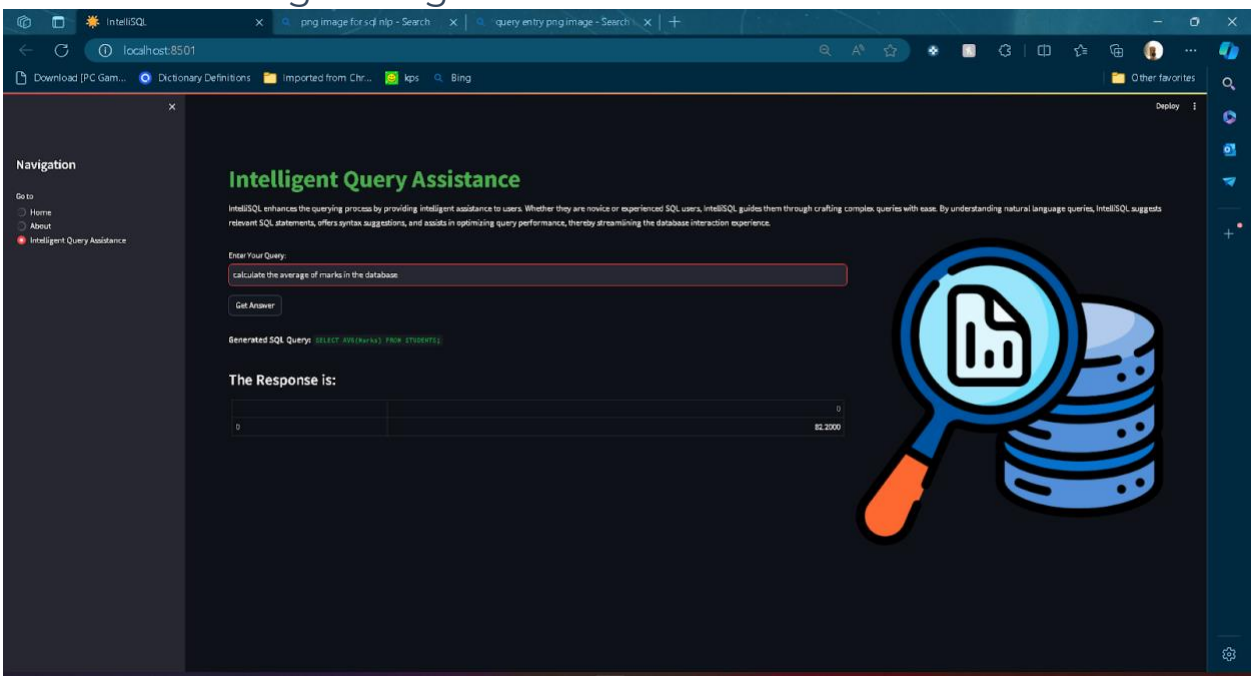
4. Displaying all records from data.db



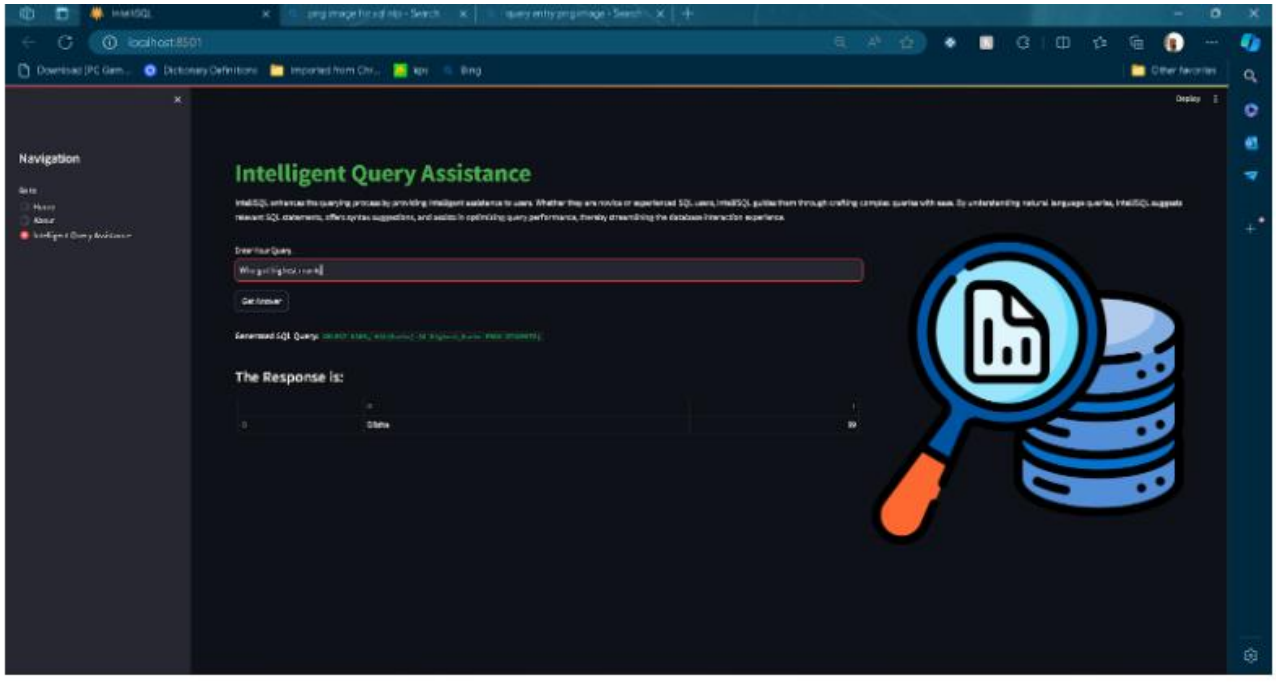
5. Displaying records of student working at INFOSYS



6. Calculating average of marks scored



7. Finding out the highest marks



COMMENTS

FIRST AND FOREMOST, I SINCERELY GRATITUDE TO OUR ESTEEMED INSTITUTE SRI VASAVI DEGREE COLLEGE, FOR GIVING ME THIS OPPORTUNITY TO FULFILL OUR WARM DREAM TO BECOME A GRADUATE. OUR SINCERE GRADITUDE TO OUR LONG-TERM INTERNSHIP GUIDE **SRI L LAKSHMI NARAYANA**, LECTURER DEPARTMENT OF COMPUTER SCIENCE FOR TIMELY COOPERATION AND VALUABLE SUGGESTIONS WHILE CARRYING OUT THIS INTERNSHIP.

I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO **SRI L LAKSHMI NARAYANA**, HOD IN COMPUTER SCIENCE FOR PERMITTING ME TO DO MY PROJECT INTERNSHIP. I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO **SRI M RAMA KRISHNA**, PRINCIPAL FOR PROVIDING A FAVOURABLE ENVIRONMENT AND SUPPORTING ME DURING THE DEVELOPMENT OF THIS INTERNSHIP.

THANK YOU, SMART BRIDGE

---- ADAPA DURGA BHAVANI

TEAM LEADER

SIGNATURE OF THE HOD

SIGNATURE OF THE.PRINCIPAL