

# QueryCraft: Text-to-SQL Converter using Gemini Model

## Project Description

QueryCraft is an advanced project powered by a Large Language Model (LLM) designed to seamlessly convert natural language questions into SQL queries. This innovative system is aimed at simplifying database interactions by allowing users to query databases using everyday language, without needing to know SQL. QueryCraft can be utilized across various scenarios, providing robust solutions to different user needs.

### **Scenario 1: Business Analytics**

QueryCraft allows business analysts to ask questions in natural language, such as "What were the total sales last quarter?" The LLM interprets this question and converts it into an accurate SQL query. This feature enables analysts to quickly access and analyze data without requiring SQL expertise, thereby improving productivity and decision-making.

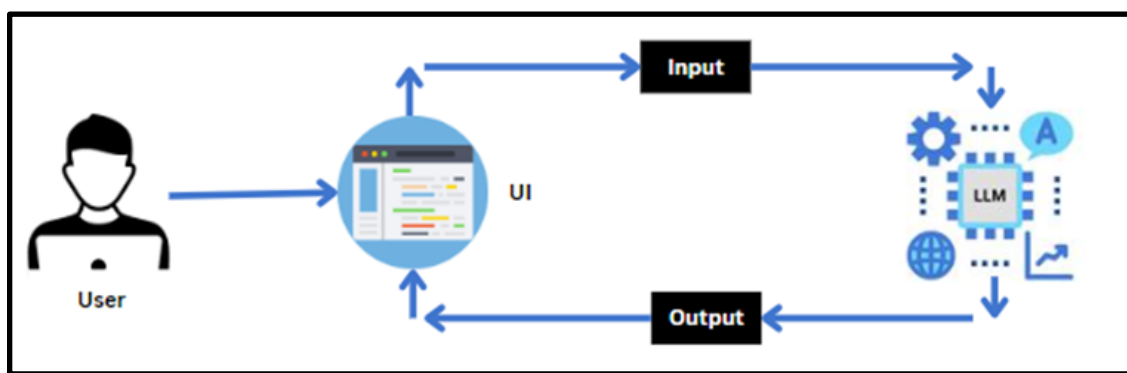
### **Scenario 2: Customer Support**

With QueryCraft, customer support representatives can simply input questions like "Show the recent orders placed by user ID 12345" into the system. The LLM converts these questions into SQL queries, enabling quick and efficient data retrieval. This reduces response times and enhances customer satisfaction.

### **Scenario 3: Educational Tools**

QueryCraft can be integrated into educational platforms to assist students by allowing them to input queries in plain English, such as "List all students who scored above 90." The LLM converts these inputs into SQL queries, helping students learn how natural language maps to SQL commands. This facilitates a more intuitive and engaging learning experience.

## Technical Architecture



## **Project Flow:**

- User interacts with the UI to enter the input.
- User input is collected from the UI and transmitted to the backend using the Google API key.
- The input is then forwarded to the Gemini Pro pre-trained model via an API call.
- The Gemini Pro pre-trained model processes the input and generates the output.
- The results are returned to the frontend for formatting and display.

## **To accomplish this, we have to complete all the activities listed below:**

- **Requirements Specification**
  - Create a requirements.txt file to list the required libraries.
  - Install the required libraries
- **Initialization of Google API Key**
  - Generate Google API Key
  - Initialize Google API Key
- **Interfacing with Pre-trained Model**
  - Load the Gemini Pro pre-trained model
  - Implement a function to get gemini response
  - Write a prompt for gemini model
- **Model Deployment**
  - Integrate with Web Framework
  - Host the Application

## **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](https://www.tutorialspoint.com/natural_language_processing/index.htm)
- Generative AI: [https://en.wikipedia.org/wiki/Generative\\_artificial\\_intelligence](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/>

## **Project Structure**

Create the Project folder which contains files as shown below:

- images folder: It is established to store the images utilized in the user interface.
- .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.
- Additionally, ensure proper file organization and adhere to best practices for version control.

## **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.**

```
# libraries need to be installed
streamlit
streamlit_extras
google-generativeai
python-dotenv
Pillow
```

- streamlit: Streamlit is a powerful framework for building interactive web applications with Python.
- streamlit\_extras: Additional utilities and enhancements for Streamlit applications.
- google-generativeai: Python client library for accessing the GenerativeAI API, facilitating interactions with pre-trained language models like Gemini Pro.
- python-dotenv: Python-dotenv allows you to manage environment variables stored in a .env file for your Python projects.
- Pillow: Pillow is a Python Imaging Library (PIL) fork that adds support for opening, manipulating, and saving many different image file formats.

**Activity 2: Install the required libraries**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\dhurvi\patel\Desktop\SCIT_academic\Smart_Bridge_Internship\SB_Company_work\May_Month\Text to SQL LLM Gemini> pip install -r requirements.txt
```

- Open the terminal.
- Run the command: `pip install -r requirements.txt`
- This command installs all the libraries listed in the requirements.txt file.

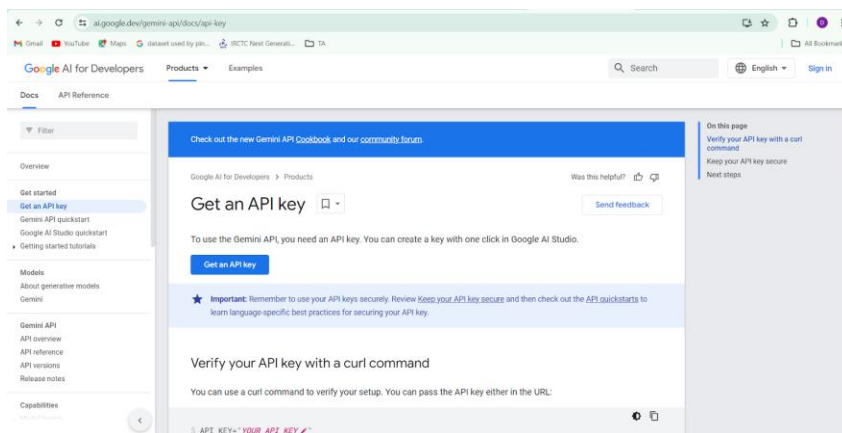
## **Milestone 2: Initialization of Google API Key**

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.

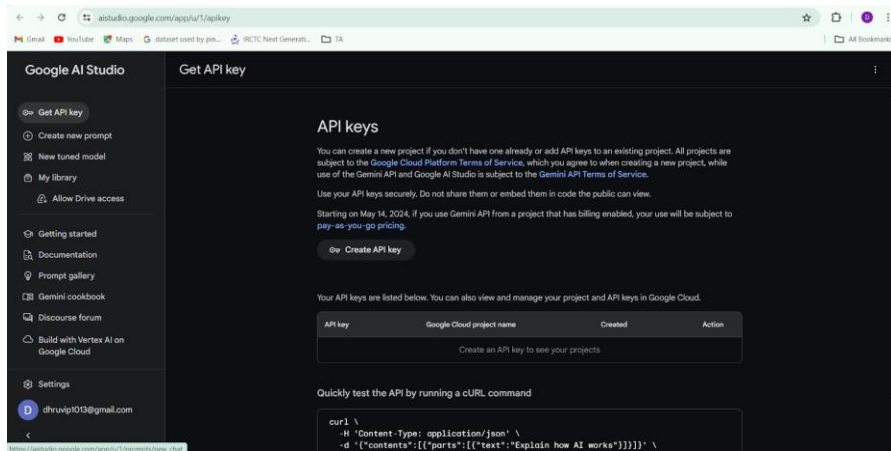
### **Activity 1: Generate Google API Key**

Click the provided link to access the following webpage.

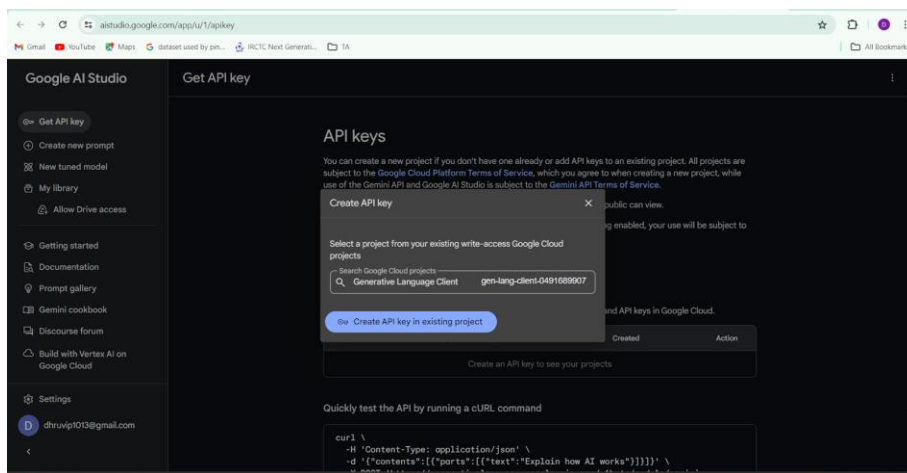
Link: <https://ai.google.dev/gemini-api/docs/api-key>



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: 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

```
from dotenv import load_dotenv
import streamlit as st
from streamlit_extras import add_vertical_space as avs
import google.generativeai as genai
import os
from PIL import Image

load_dotenv()

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

model = genai.GenerativeModel('gemini-pro')
```

- The code begins by importing necessary libraries and modules, including dotenv, Streamlit, os, GenerativeAI from Google, PIL (Python Imaging Library), and a custom module for adding vertical space in Streamlit.
- It loads environment variables from the .env file using the load\_dotenv() function.
- The GenerativeAI module is configured with the Google API key stored in the environment variable GOOGLE\_API\_KEY.
- A GenerativeModel object named "model" is created using the Gemini Pro pre-trained model from Google.
- The code is essentially setting up the environment, configuring the GenerativeAI module with the API key, and loading the Gemini Pro model for generating responses to user inputs in the Streamlit app.

### Activity 2: Implement a function to get gemini response

```
def get_gemini_response(input):
    response = model.generate_content(input)
    return response.text
```

- The function get\_gemini\_response takes an input text as a parameter.
- It calls the generate\_content method of the model object to generate a response.
- The generated response is returned as text.

### Activity 3: Write a prompt for gemini model

```

input_prompt_template = """
You are an AI language model that can convert plain English descriptions of database queries into valid SQL statements.
Your task is to read an English description of a query and output the corresponding SQL query.

Follow these guidelines:

Identify the table(s) involved.
Determine the specific columns needed.
Recognize any conditions or filters that should be applied.
Include any sorting, grouping, or aggregation if specified.
Format the SQL query properly.
Here are some examples:

Example 1:
Input: "Retrieve the names and ages of all employees who are older than 30 years."
Output: SELECT name, age FROM employees
        WHERE age > 30;

Example 2:
Input: "Get the total sales amount from the orders table."
Output: SELECT SUM(amount) FROM orders;

Example 3:
Input: "Find the list of customers who made purchases in the last month, sorted by the date of purchase in descending order."
Output: SELECT customer_name FROM purchases
        WHERE purchase_date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
        ORDER BY purchase_date DESC;

Example 4:
Input: "Show the average salary of employees grouped by department."
Output: SELECT department, AVG(salary) FROM employees
        GROUP BY department;

Your task: Convert the following English descriptions into SQL queries.

Input: {sql_text}

Things you need to remember everytime-
Print -"SQL Query: " and from the next line write the sql query snippet
Present the results following the same format as the examples provided above.
Break the query snippet as shown in examples
"""

```

- The provided code snippet is a template for generating input prompts for a text-to-SQL converter AI model.
- It defines a template string with placeholders for SQL text inputs.
- The purpose is to instruct the AI model on how to convert English descriptions of database queries into valid SQL statements.
- The template includes guidelines for the AI model, such as identifying tables, determining columns, applying conditions or filters, and formatting SQL queries properly.
- It provides examples of English descriptions and their corresponding SQL queries to illustrate the expected input-output format.
- The template also specifies the format for printing the SQL query snippet as output.
- The {sql\_text} placeholder indicates where the English description of the query will be inserted.
- Overall, the template streamlines the process of generating input prompts for training and testing the text-to-SQL converter model.

## **Milestone 4: 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 four main sections to provide users with a comprehensive experience:

- Introduction:

```
# Streamlit UI configuration
st.set_page_config(page_title="Text to SQL Query", layout="wide")

avs.add_vertical_space(4)

col1, col2 = st.columns([3, 2])
with col1:
    st.title("QueryCraft")
    st.header("Making Database Queries as Easy as Conversation!")
    st.markdown(
        """<p style='text-align: justify;'>
        Introducing QueryCraft, your revolutionary solution for simplifying database interactions
        through natural language queries. Powered by a cutting-edge Large Language Model (LLM),
        QueryCraft seamlessly converts everyday language into accurate SQL queries, eliminating the
        need for SQL expertise. Whether you're a business analyst seeking quick insights, a customer
        support representative streamlining data retrieval, or an educator enhancing student learning
        experiences, QueryCraft offers versatile solutions tailored to your needs. Say goodbye to
        complex database interactions and hello to effortless data access with QueryCraft. Unlock the
        power of natural language querying today!
        </p>""", unsafe_allow_html=True
    )

with col2:
    img = Image.open("images/icon.png")
    st.image(img, use_column_width=True)
```

- This Streamlit code sets up a web page titled "Text to SQL Query" with a wide layout.
  - It creates two columns, with the first column being wider than the second.
  - The first column displays the title "QueryCraft" and a header that introduces the application as a revolutionary tool for converting natural language into SQL queries.
  - HTML formatting is used for better text alignment.
  - The second column shows an image representing the QueryCraft icon.
  - Overall, the code provides a clear and visually appealing introduction to QueryCraft, combining informative text and an image.
- Offering:



```
col1, col2 = st.columns([3, 2])
with col2:
    st.header("Wide Range of Offerings")
    st.write("- Seamless Natural Language to SQL Conversion: Effortlessly convert natural language questions into SQL queries.")
    st.write("- Business Analytics: Enable business analysts to quickly access and analyze data without needing SQL expertise.")
    st.write("- Customer Support: Streamline data retrieval for customer support representatives, reducing response times and enhancing customer satisfaction.")
    st.write("- Educational Tools: Assist students in learning SQL by allowing them to input queries in plain English.")
    st.write("- Increased Productivity: Improve decision-making and productivity by simplifying database interactions.")
    st.write("- Intuitive User Experience: Provide an intuitive and engaging experience for users across various scenarios.")
    st.write("- Versatile Solutions: Cater to the needs of business analysts, customer support representatives, and educators.")

with col1:
    img1 = Image.open("images/icon1.png")
    st.image(img1, use_column_width=True)
```

- This Streamlit code creates a web page layout with two columns.
  - In the second column, it displays a header titled "Wide Range of Offerings" and a list of key features of QueryCraft.
  - The first column displays an image representing the QueryCraft icon using the Image.open() function and st.image() method.
  - Overall, the code succinctly presents the key offerings of QueryCraft alongside a visual representation.
- Text to SQL Query Conversion Application:

```
col1, col2 = st.columns([3, 2])
with col1:
    st.markdown("<h1 style='text-align: center;'>Start Your Query Conversion</h1>", unsafe_allow_html=True)
    sql_text = st.text_area("Provide the text")

    submit = st.button("Submit")

    if submit:
        # Insert the user input into the input prompt template
        input_prompt = input_prompt_template.format(sql_text = sql_text)
        response = get_gemini_response(input_prompt)
        st.subheader(response)

with col2:
    img2 = Image.open("images/icon2.png")
    st.image(img2, use_column_width=True)
```

- This Streamlit code sets up a webpage with two columns.
- In the first column, it displays a centered header "Start Your Query Conversion" using HTML for styling.
- It provides a text area for users to input their natural language description of a query.
- When the "Submit" button is clicked, the user's input is formatted into an input prompt template, and the get\_gemini\_response function is called to convert the natural language description into an SQL query.

- The result is displayed as a subheader. In the second column, an image is displayed using the `Image.open()` function and the `st.image()` method, providing a visual element to the interface.
- This code allows users to interactively convert natural language queries into SQL queries through a user-friendly interface.

- FAQ:

```
col1, col2 = st.columns([2, 3])
with col2:
    st.write("Question: What is QueryCraft?")
    st.write("""Answer: QueryCraft is an advanced project powered by a Large Language Model (LLM) that
converts natural language questions into SQL queries, simplifying database interactions for
users without SQL expertise.""")
    avs.add_vertical_space(3)
    st.write("Question: How can QueryCraft help business analysts?")
    st.write("""Answer: QueryCraft allows business analysts to ask questions in natural language and converts
these questions into accurate SQL queries. This enables analysts to quickly access and analyze data,
improving productivity and decision-making.""")
    avs.add_vertical_space(3)
    st.write("Question: Can QueryCraft be used in customer support?")
    st.write("""Answer: Yes, customer support representatives can use QueryCraft to input questions like 'Show the
recent orders placed by user ID 12345,' which the LLM converts into SQL queries. This helps in quick and
efficient data retrieval, reducing response times and enhancing customer satisfaction.""")

with col1:
    avs.add_vertical_space(8)
    img3 = Image.open("images/icon3.png")
    st.image(img3, use_column_width=True)
```

- This Streamlit code sets up a webpage with two columns.
- In the second column, it displays three FAQ questions about QueryCraft, each displayed using the `st.write()` method.
- Vertical space is added between each question-answer pair for better readability using the `avs.add_vertical_space()` function.
- In the first column, an image representing QueryCraft is displayed using the `Image.open()` function and the `st.image()` method.
- This code creates an informative FAQ section about QueryCraft, allowing users to quickly understand its features and benefits.

## Activity 2: Host the Application

Launching the Application:

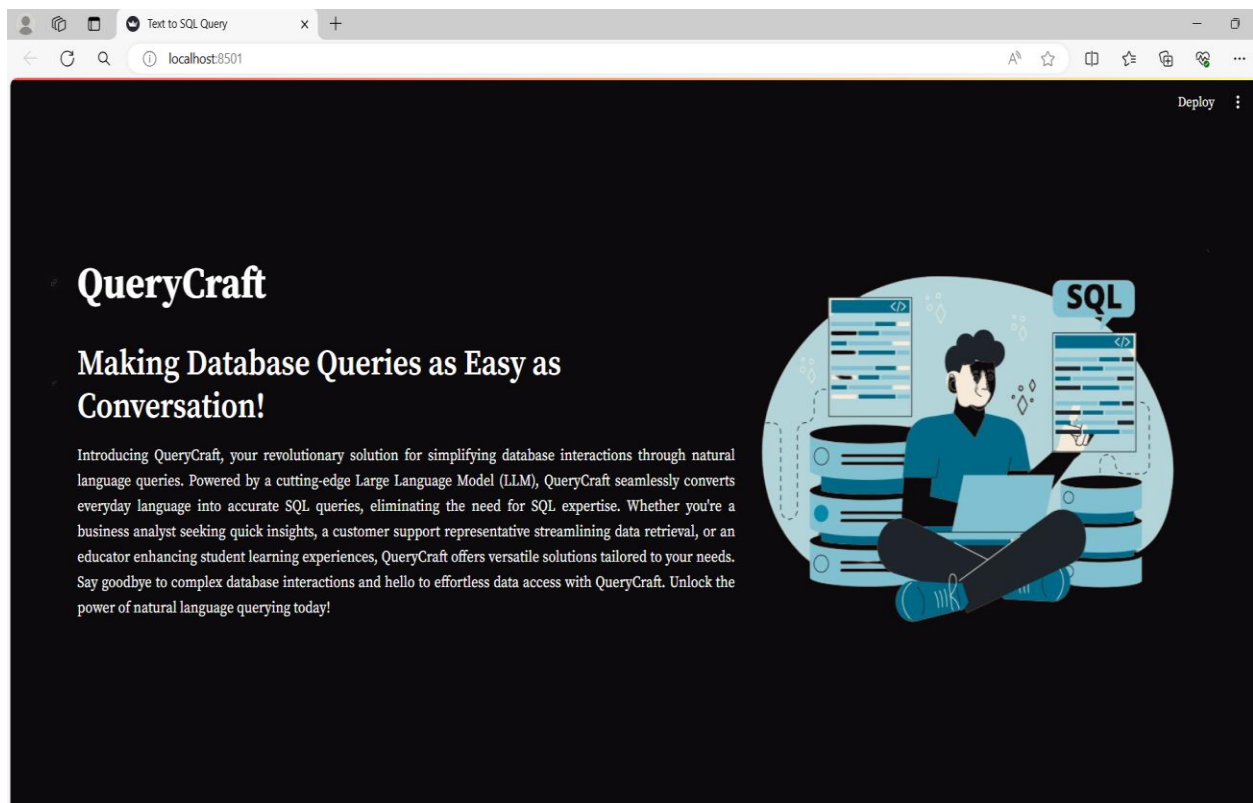
- To host the application, go to the terminal, type - `streamlit run app.py`
- Here `app.py` refers to a python script.

```
PS C:\Users\dhurvi patel\Desktop\SCIT academic\Smart_Bridge_Internship\SB_Company work\May_Month\Text to SQL LLM Gemini> streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.29.80:8501
```


Run the command to get the below results



Text to SQL Query

localhost:8501

Deploy



## Wide Range of Offerings

- **Seamless Natural Language to SQL Conversion:** Effortlessly convert natural language questions into SQL queries.
- **Business Analytics:** Enable business analysts to quickly access and analyze data without needing SQL expertise.
- **Customer Support:** Streamline data retrieval for customer support representatives, reducing response times and enhancing customer satisfaction.
- **Educational Tools:** Assist students in learning SQL by allowing them to input queries in plain English.
- **Increased Productivity:** Improve decision-making and productivity by simplifying database interactions.
- **Intuitive User Experience:** Provide an intuitive and engaging experience for users across various scenarios.
- **Versatile Solutions:** Cater to the needs of business analysts, customer support representatives, and educators.

Text to SQL Query


localhost:8501

Deploy

## Start Your Query Conversion

Provide the text

Submit



## INPUT 1

Text to SQL Query

localhost:8501

Deploy


## Start Your Query Conversion

Provide the text

Show me the first names, last names, and email addresses of employees who work in the Sales department and were hired after January 1, 2022. Sort the results by last name in ascending order.

Press Ctrl+Enter to apply

Submit



## OUTPUT 1

Text to SQL Query

localhost:8501

Deploy

## Start Your Query Conversion


Provide the text

Show me the first names, last names, and email addresses of employees who work in the Sales department and were hired after January 1, 2022. Sort the results by last name in ascending order.

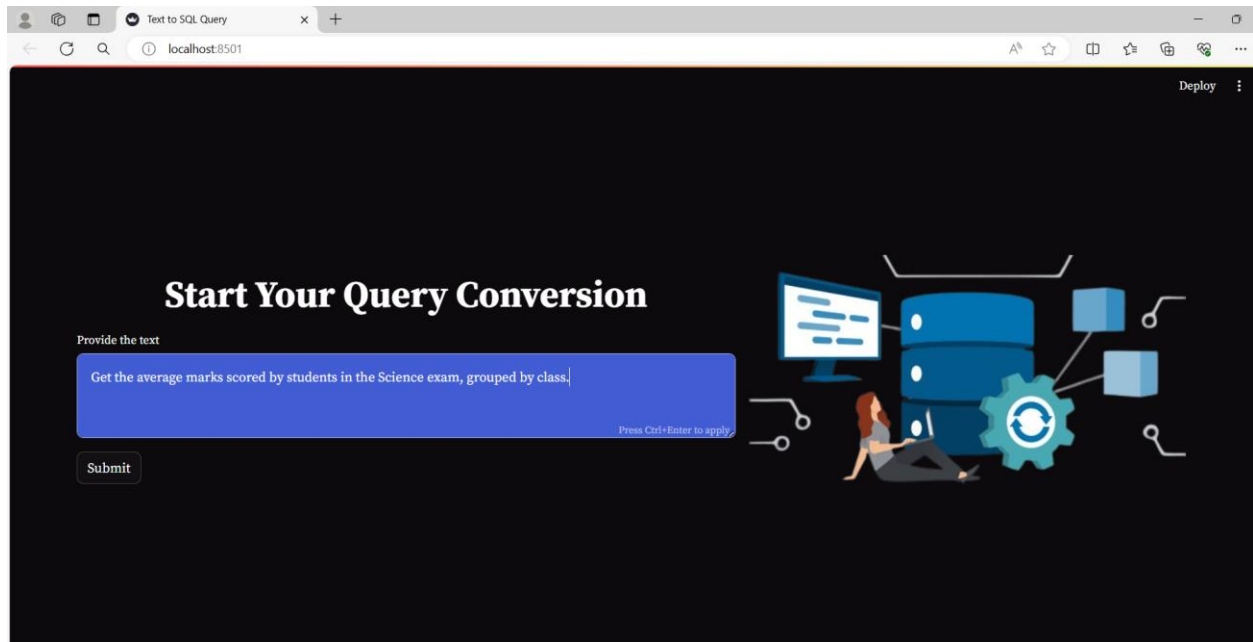
Submit

**SQL Query:**

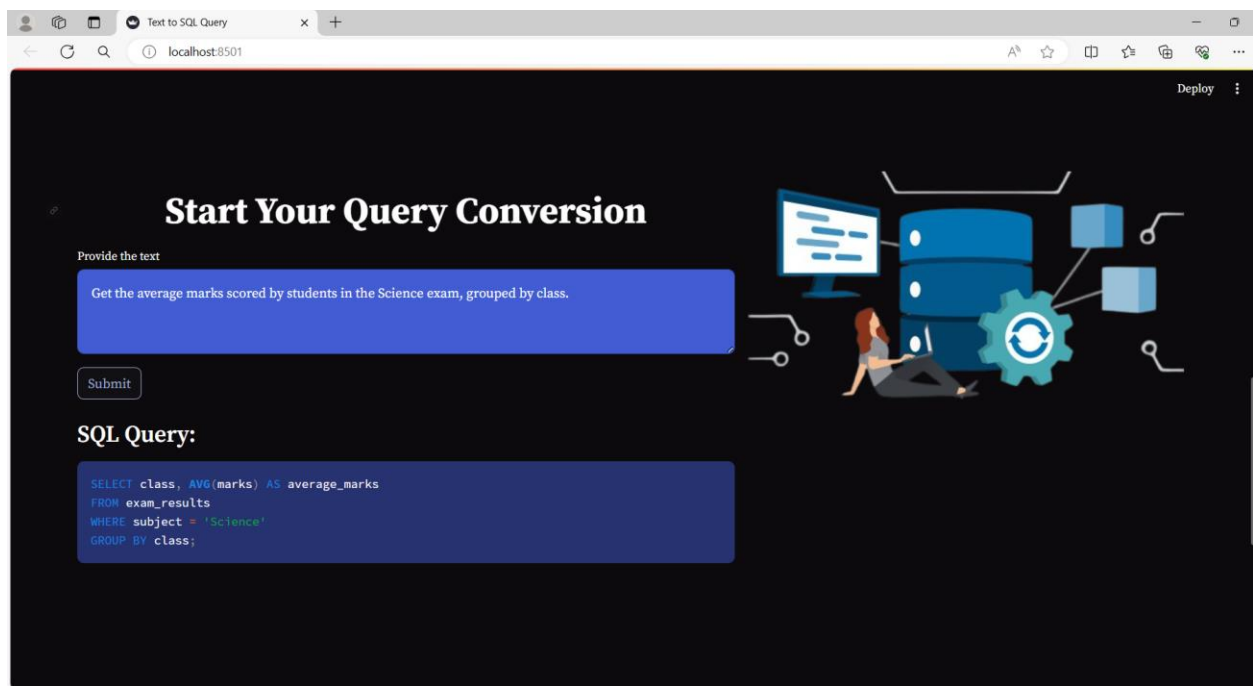
```
SELECT first_name, last_name, email
FROM employees
WHERE department = 'Sales'
AND hire_date > '2022-01-01'
ORDER BY last_name ASC;
```



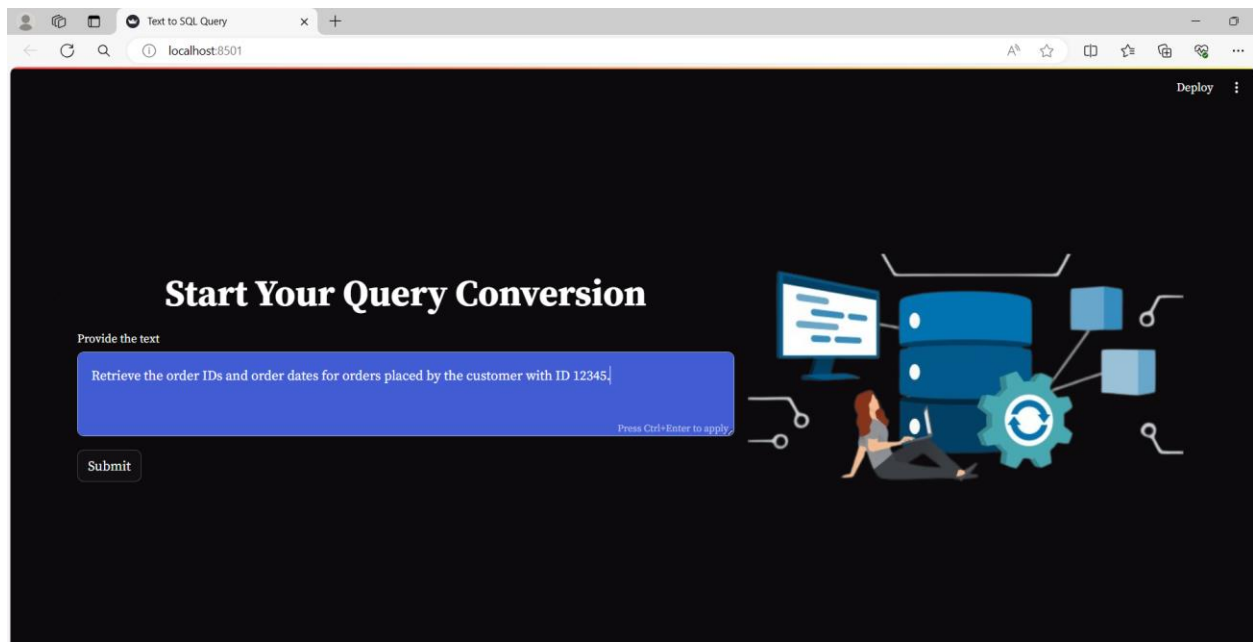
## INPUT 2



## OUTPUT 2



## INPUT 3



## OUTPUT 3

