An AI application that can chat with any SQL database.

Apurv Agarwal · Follow

10 min read · Nov 2, 2023

Is it possible to chat with your SQL database? In this tutorial we will create this application using python. We will use streamlit for frontend and langchain for AI integration.

First step is to create a virtual environment and install the dependencies. We also will create the project directory and root file named app.py and .env to store environment variables. For this project we will need only OPENAI_API_KEY. Replace it with your own.

```
virtualenv chatdb
source chatdb/bin/activate
pip install langchain openai sqlalchemy streamlit python-dotenv


mkdir chatdb
cd chatdb
touch app.py
touch .env
```

```
## .env
OPENAI_API_KEY=sk-NcGHMSIv3POeMXAEf.............
```

Now we'll start writing our app. Each of the below code snippets below can be run independently.

Let's first create basic chat interface. This interface will consist of an input which will take a database URI. Through this we will connect to our choice of database and start the chat.

```
import streamlit as st
import requests
import os

# Function to establish connection and read metadata for the database
def connect_with_db(uri):
    st.session_state.db_uri = uri
    return {"message": "Connection established to Database!"}

# Function to call the API with the provided URI
def send_message(message):
    return {"message": message}



# ## Instructions
st.subheader("Instructions")
st.markdown(
    """
    1. Enter the URI of your RDS Database in the text box below.
    2. Click the **Start Chat** button to start the chat.
    3. Enter your message in the text box below and press **Enter** to send the
    """
)

# Initialize the chat history list
chat_history = []

# Input for the database URI
uri = st.text_input("Enter the RDS Database URI")

if st.button("Start Chat"):
    if not uri:
        st.warning("Please enter a valid database URI.")
    else:
```

```
        st.info("Connecting to the API and starting the chat...")
        chat_response = connect_with_db(uri)
        if "error" in chat_response:
            st.error("Error: Failed to start the chat. Please check the URI and
        else:
            st.success("Chat started successfully!")

# Chat with the API (a mock example)
st.subheader("Chat with the API")

# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    st.chat_message("user").markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})

    # response = f"Echo: {prompt}"
    response = send_message(prompt)["message"]
    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        st.markdown(response)
    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content": response})

# Run the Streamlit app
if __name__ == "__main__":
    st.write("This is a simple Streamlit app for starting a chat with an RDS Dat
```

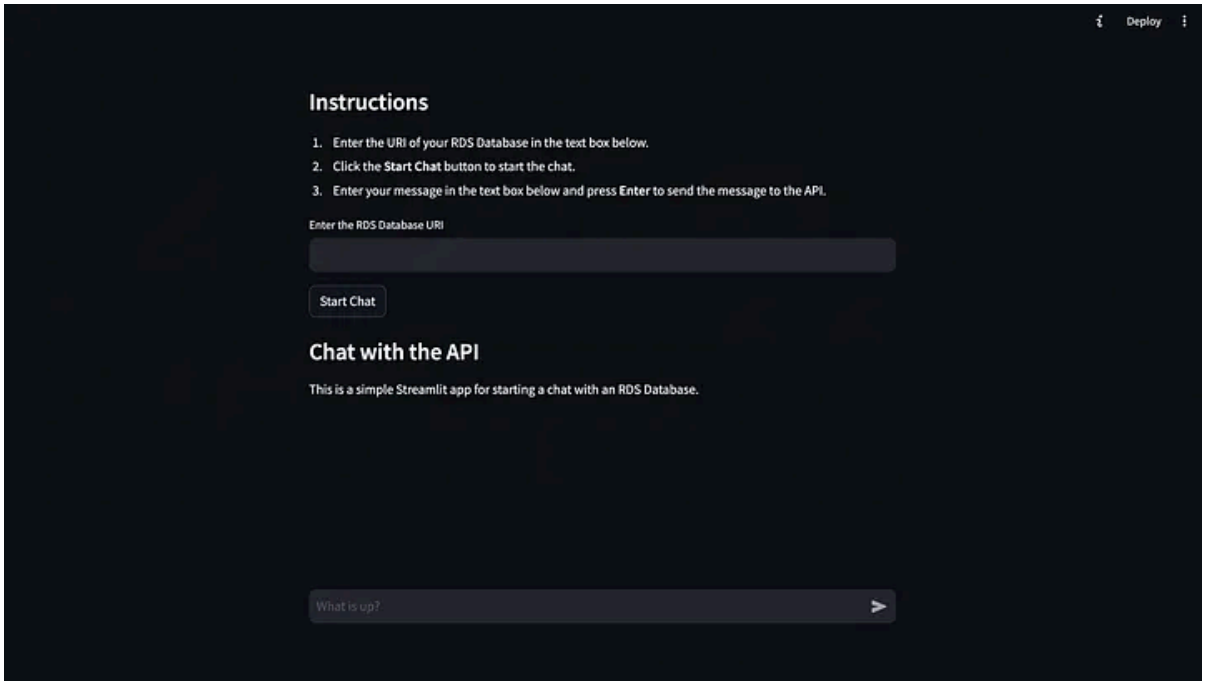
Now let's write a function that connects to the database and gets the basic information about the table.

This system will basically work by creating a relevant SQL query and executing it.

If you suppose want to create an SQL query through ChatGPT, you will need to provide relevant context which might include relevant table names, relevant column names and perhaps relevant values that go into the where

clause. We will try to automate the same thing here. As soon as we connect to the database we will get all the tables and columns to pass in the user's prompt.

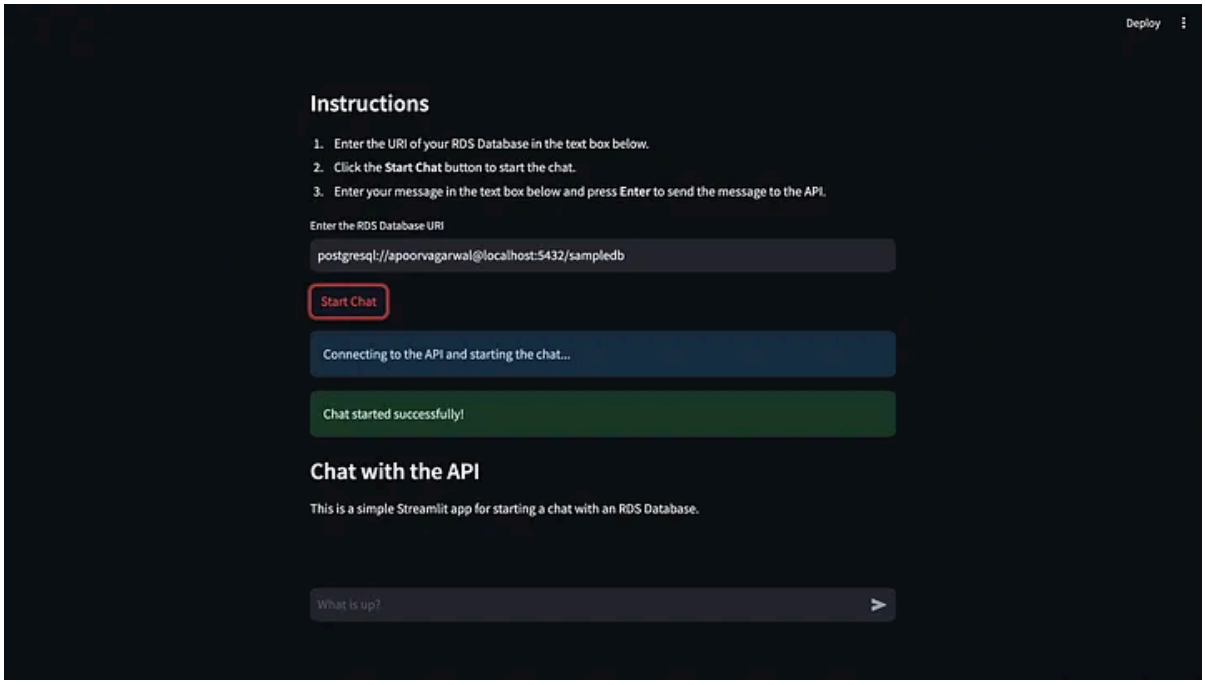
In below code when we click on Start Chat, the system will read the information of the database and save it in a CSV.
We will create a folder csvs to save the CSV with a unique id in. And we will save this unique id in session of streamlit, to check the right csv later on when user asks the query.




```
import streamlit as st
import requests
import os
import pandas as pd
from uuid import uuid4
import psycopg2


folders_to_create = ['csvs']
# Check and create folders if they don't exist
for folder_name in folders_to_create:
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)
        print(f"Folder '{folder_name}' created.")
    else:
```

```
        print(f"Folder '{folder_name}' already exists.")



def get_basic_table_details(cursor):
    cursor.execute("""SELECT
            c.table_name,
            c.column_name,
            c.data_type
        FROM
            information_schema.columns c
        WHERE
            c.table_name IN (
                SELECT tablename
                FROM pg_tables
                WHERE schemaname = 'public'
    );""")
    tables_and_columns = cursor.fetchall()
    return tables_and_columns





def save_db_details(db_uri):

    unique_id = str(uuid4()).replace("-", "_")
    connection = psycopg2.connect(db_uri)
    cursor = connection.cursor()

    tables_and_columns = get_basic_table_details(cursor)

    ## Get all the tables and columns and enter them in a pandas dataframe
    df = pd.DataFrame(tables_and_columns, columns=['table_name', 'column_name',
    filename_t = 'csvs/tables_' + unique_id + '.csv'
    df.to_csv(filename_t, index=False)

    cursor.close()
    connection.close()

    return unique_id



# Function to establish connection and read metadata for the database
def connect_with_db(uri):
    st.session_state.unique_id = save_db_details(uri)
    return {"message": "Connection established to Database!"}

# Function to call the API with the provided URI
def send_message(message):
    return {"message": message}



# ## Instructions
```

```
st.subheader("Instructions")
st.markdown(
    """
    1. Enter the URI of your RDS Database in the text box below.
    2. Click the **Start Chat** button to start the chat.
    3. Enter your message in the text box below and press **Enter** to send the
    """
)

# Initialize the chat history list
chat_history = []

# Input for the database URI
uri = st.text_input("Enter the RDS Database URI")

if st.button("Start Chat"):
    if not uri:
        st.warning("Please enter a valid database URI.")
    else:
        st.info("Connecting to the API and starting the chat...")
        chat_response = connect_with_db(uri)
        if "error" in chat_response:
            st.error("Error: Failed to start the chat. Please check the URI and
        else:
            st.success("Chat started successfully!")

# Chat with the API (a mock example)
st.subheader("Chat with the API")

# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    st.chat_message("user").markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})

    # response = f"Echo: {prompt}"
    response = send_message(prompt)["message"]
    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        st.markdown(response)
    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content": response})

# Run the Streamlit app
if __name__ == "__main__":
    st.write("This is a simple Streamlit app for starting a chat with an RDS Dat
```



Now we'll define the function that takes in the user's query, combines it with database metadata and pass it to LLM for a resulting SQL query.

```
import streamlit as st
import requests
import os
import pandas as pd
from uuid import uuid4
import psycopg2

from langchain.prompts import ChatPromptTemplate
from langchain.prompts.chat import SystemMessage, HumanMessagePromptTemplate

from langchain.llms import OpenAI, AzureOpenAI
from langchain.chat_models import ChatOpenAI, AzureChatOpenAI
from langchain.embeddings import OpenAIEmbeddings
from dotenv import load_dotenv



folders_to_create = ['csvs']
# Check and create folders if they don't exist
for folder_name in folders_to_create:
    if not os.path.exists(folder_name):
```

```
        os.makedirs(folder_name)
        print(f"Folder '{folder_name}' created.")
    else:
        print(f"Folder '{folder_name}' already exists.")




## load the API key from the environment variable
load_dotenv()
openai_api_key = os.getenv("OPENAI_API_KEY")


llm = OpenAI(openai_api_key=openai_api_key)
chat_llm = ChatOpenAI(openai_api_key=openai_api_key, temperature=0.4)
embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)





def get_basic_table_details(cursor):
    cursor.execute("""SELECT
            c.table_name,
            c.column_name,
            c.data_type
        FROM
            information_schema.columns c
        WHERE
            c.table_name IN (
                SELECT tablename
                FROM pg_tables
                WHERE schemaname = 'public'
    );""")
    tables_and_columns = cursor.fetchall()
    return tables_and_columns




def save_db_details(db_uri):

    unique_id = str(uuid4()).replace("-", "_")
    connection = psycopg2.connect(db_uri)
    cursor = connection.cursor()

    tables_and_columns = get_basic_table_details(cursor)

    ## Get all the tables and columns and enter them in a pandas dataframe
    df = pd.DataFrame(tables_and_columns, columns=['table_name', 'column_name',
    filename_t = 'csvs/tables_' + unique_id + '.csv'
    df.to_csv(filename_t, index=False)

    cursor.close()
    connection.close()
```

```
    return unique_id
```

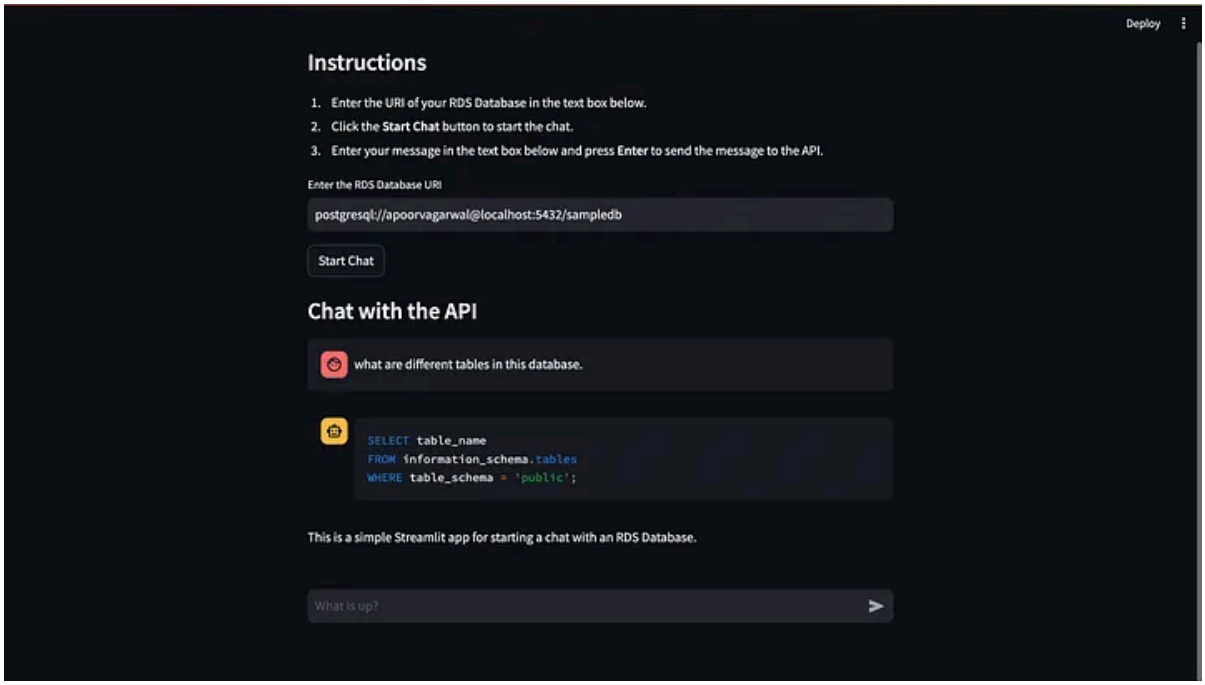
```
def generate_template_for_sql(query, table_info, db_uri):
    template = ChatPromptTemplate.from_messages(
            [
                SystemMessage(
                    content=(
                        f"You are an assistant that can write SQL Queries."
                        f"Given the text below, write a SQL query that answers t
                        f"DB connection string is {db_uri}"
                        f"Here is a detailed description of the table(s): "
                        f"{table_info}"
                        "Prepend and append the SQL query with three backticks '


                    )
                ),
                HumanMessagePromptTemplate.from_template("{text}"),

            ]
        )

    answer = chat_llm(template.format_messages(text=query))
    return answer.content
```

```
def get_the_output_from_llm(query, unique_id, db_uri):
    ## Load the tables csv
    filename_t = 'csvs/tables_' + unique_id + '.csv'
    df = pd.read_csv(filename_t)

    ## For each relevant table create a string that list down all the columns an
    table_info = ''
    for table in df['table_name']:
        table_info += 'Information about table' + table + ':\n'
        table_info += df[df['table_name'] == table].to_string(index=False) + '\n

    return generate_template_for_sql(query, table_info, db_uri)
```

```
# Function to establish connection and read metadata for the database
def connect_with_db(uri):
    st.session_state.db_uri = uri
    st.session_state.unique_id = save_db_details(uri)
```

```
    return {"message": "Connection established to Database!"}

# Function to call the API with the provided URI
def send_message(message):
    return {"message": get_the_output_from_llm(message, st.session_state.unique_



# ## Instructions
st.subheader("Instructions")
st.markdown(
    """
    1. Enter the URI of your RDS Database in the text box below.
    2. Click the **Start Chat** button to start the chat.
    3. Enter your message in the text box below and press **Enter** to send the
    """
)

# Initialize the chat history list
chat_history = []

# Input for the database URI
uri = st.text_input("Enter the RDS Database URI")

if st.button("Start Chat"):
    if not uri:
        st.warning("Please enter a valid database URI.")
    else:
        st.info("Connecting to the API and starting the chat...")
        chat_response = connect_with_db(uri)
        if "error" in chat_response:
            st.error("Error: Failed to start the chat. Please check the URI and
        else:
            st.success("Chat started successfully!")

# Chat with the API (a mock example)
st.subheader("Chat with the API")

# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    st.chat_message("user").markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})

    # response = f"Echo: {prompt}"
    response = send_message(prompt)["message"]
```

```
    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        st.markdown(response)
    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content": response})

# Run the Streamlit app
if __name__ == "__main__":
    st.write("This is a simple Streamlit app for starting a chat with an RDS Dat
```



Finally, we run our generated SQL query and display those in the result.

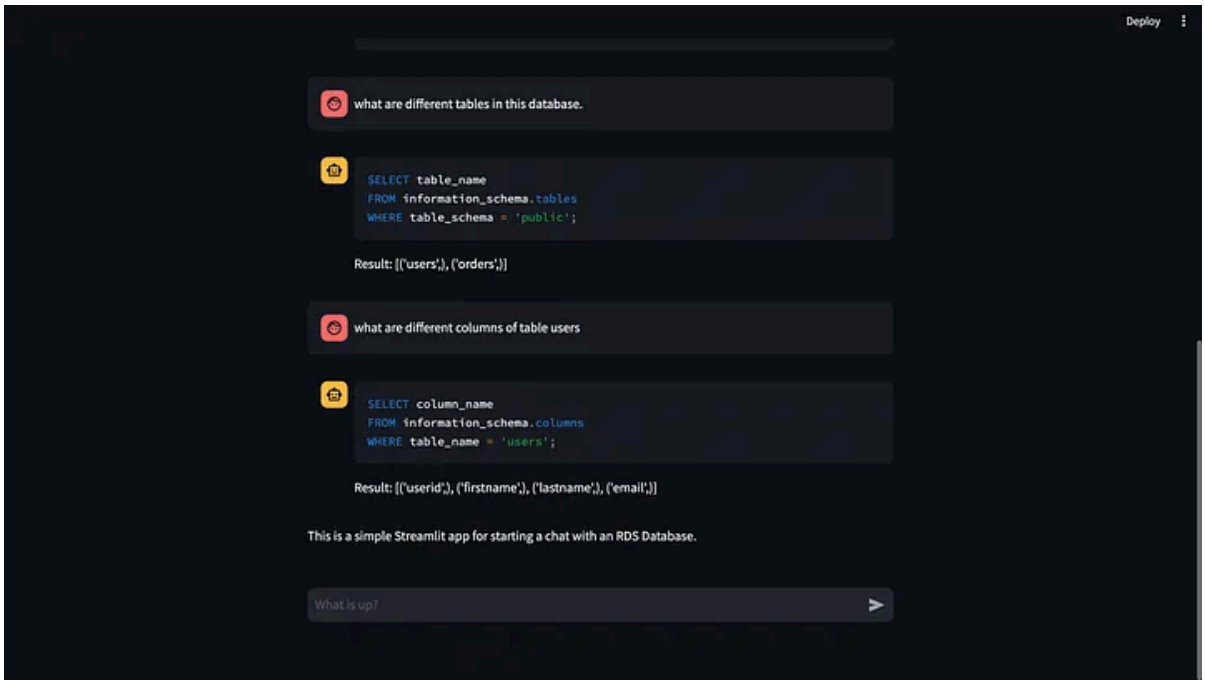
```
import streamlit as st
import requests
import os
import pandas as pd
from uuid import uuid4
import psycopg2

from langchain.prompts import ChatPromptTemplate
from langchain.prompts.chat import SystemMessage, HumanMessagePromptTemplate

from langchain.llms import OpenAI, AzureOpenAI
from langchain.chat_models import ChatOpenAI, AzureChatOpenAI
from langchain.embeddings import OpenAIEmbeddings
from dotenv import load_dotenv
```

```
folders_to_create = ['csvs']
# Check and create folders if they don't exist
for folder_name in folders_to_create:
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)
        print(f"Folder '{folder_name}' created.")
    else:
        print(f"Folder '{folder_name}' already exists.")




## load the API key from the environment variable
load_dotenv()
openai_api_key = os.getenv("OPENAI_API_KEY")


llm = OpenAI(openai_api_key=openai_api_key)
chat_llm = ChatOpenAI(openai_api_key=openai_api_key, temperature=0.4)
embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)





def get_basic_table_details(cursor):
    cursor.execute("""SELECT
            c.table_name,
            c.column_name,
            c.data_type
        FROM
            information_schema.columns c
        WHERE
            c.table_name IN (
                SELECT tablename
                FROM pg_tables
                WHERE schemaname = 'public'
    );""")
    tables_and_columns = cursor.fetchall()
    return tables_and_columns




def save_db_details(db_uri):

    unique_id = str(uuid4()).replace("-", "_")
    connection = psycopg2.connect(db_uri)
    cursor = connection.cursor()

    tables_and_columns = get_basic_table_details(cursor)
```

```
    ## Get all the tables and columns and enter them in a pandas dataframe
    df = pd.DataFrame(tables_and_columns, columns=['table_name', 'column_name',
    filename_t = 'csvs/tables_' + unique_id + '.csv'
    df.to_csv(filename_t, index=False)

    cursor.close()
    connection.close()

    return unique_id




def generate_template_for_sql(query, table_info, db_uri):
    template = ChatPromptTemplate.from_messages(
            [
                SystemMessage(
                    content=(
                        f"You are an assistant that can write SQL Queries."
                        f"Given the text below, write a SQL query that answers t
                        f"DB connection string is {db_uri}"
                        f"Here is a detailed description of the table(s): "
                        f"{table_info}"
                        "Prepend and append the SQL query with three backticks '


                    )
                ),
                HumanMessagePromptTemplate.from_template("{text}"),

            ]
        )

    answer = chat_llm(template.format_messages(text=query))
    return answer.content




def get_the_output_from_llm(query, unique_id, db_uri):
    ## Load the tables csv
    filename_t = 'csvs/tables_' + unique_id + '.csv'
    df = pd.read_csv(filename_t)

    ## For each relevant table create a string that list down all the columns an
    table_info = ''
    for table in df['table_name']:
        table_info += 'Information about table' + table + ':\n'
        table_info += df[df['table_name'] == table].to_string(index=False) + '\n

    return generate_template_for_sql(query, table_info, db_uri)
```

```
def execute_the_solution(solution, db_uri):
    connection = psycopg2.connect(db_uri)
    cursor = connection.cursor()
    _,final_query,_ = solution.split("```")
    final_query = final_query.strip('sql')
    cursor.execute(final_query)
    result = cursor.fetchall()
    return str(result)





# Function to establish connection and read metadata for the database
def connect_with_db(uri):
    st.session_state.db_uri = uri
    st.session_state.unique_id = save_db_details(uri)

    return {"message": "Connection established to Database!"}

# Function to call the API with the provided URI
def send_message(message):
    solution = get_the_output_from_llm(message, st.session_state.unique_id, st.s
    result = execute_the_solution(solution, st.session_state.db_uri)
    return {"message": solution + "\n\n" + "Result:\n" + result}



# ## Instructions
st.subheader("Instructions")
st.markdown(
    """
    1. Enter the URI of your RDS Database in the text box below.
    2. Click the **Start Chat** button to start the chat.
    3. Enter your message in the text box below and press **Enter** to send the
    """
)

# Initialize the chat history list
chat_history = []

# Input for the database URI
uri = st.text_input("Enter the RDS Database URI")

if st.button("Start Chat"):
    if not uri:
        st.warning("Please enter a valid database URI.")
    else:
        st.info("Connecting to the API and starting the chat...")
        chat_response = connect_with_db(uri)
        if "error" in chat_response:
            st.error("Error: Failed to start the chat. Please check the URI and
        else:
            st.success("Chat started successfully!")
```

```
# Chat with the API (a mock example)
st.subheader("Chat with the API")

# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    st.chat_message("user").markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})

    # response = f"Echo: {prompt}"
    response = send_message(prompt)["message"]
    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        st.markdown(response)
    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content": response})

# Run the Streamlit app
if __name__ == "__main__":
    st.write("This is a simple Streamlit app for starting a chat with an RDS Dat
```



As you can see we can see the SQL and the results of the SQL in above screenshot. This works well when we have small database, now what if we have a really large database. If we try to pass list of all the tables and columns in the prompt, it might exceed the token limit placed by openAI's API.

To solve this issue we can use power of Retrieval Augmented Generation. We can basically save all the tables and columns in a vector database and retrieve the names of only the most relevant tables and columns.

We can also implement chat memory through which can use entire chat history to create a prompt.

We'll discuss these use cases in subsequent parts.

A second part of this tutorial is out which dives in how we can chat with very large database
https://medium.com/@systemdesigner/an-ai-application-that-can-chat-with-with-very-large-sql-databases-acd730fcfa26

AI Sql Databases Langchain

Written by Apurv Agarwal

232 Followers

Follow

Fullstack Developer

More from Apurv Agarwal

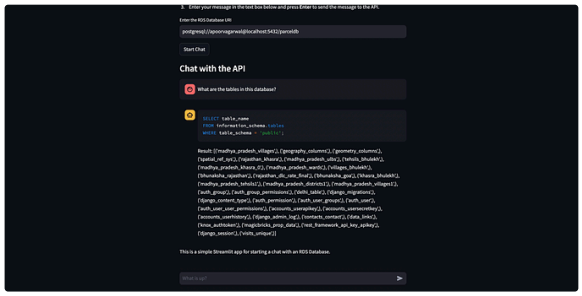
Apurv Agarwal

How to build AI agents to automate web browsing with human level...

One step towards building AGI is creating agents that can browse the internet, talk to...

3 min read · Feb 12, 2024

140 1



Apurv Agarwal

An AI application that can chat with very large SQL databases.

In the last article we created a simple application that can chat with an SQL...

18 min read · Nov 4, 2023

331 3

Apurv Agarwal

How to get started with AI, ML and Data Science, particularly LLMs...

The purpose of this article is to help someone enter the emerging fields of AI/ML,...

4 min read · May 22, 2024

15

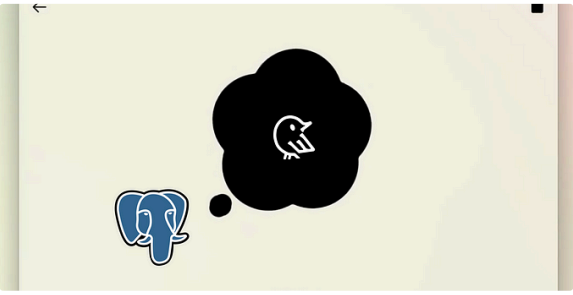
Apurv Agarwal

Next Gold Rush is happening: Making AI inference cheaper and...

3 min read · May 24, 2024

See all from Apurv Agarwal

Recommended from Medium



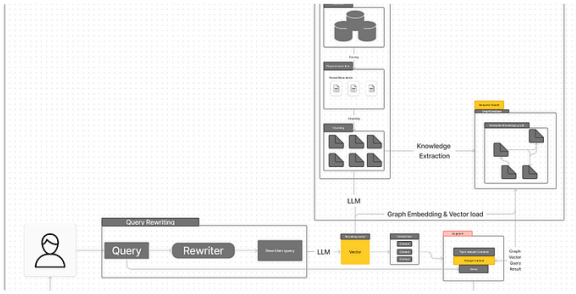
Howard Chi in WrenAI

How to use OpenAI GPT-4o to query your database?

Today, OpenAI released its latest LLM model, GPT-4o. People are sharing crazy applicatio...

5 min read · May 14, 2024

1.1K 13



Jeong ii tae

From RAG to GraphRAG , What is the GraphRAG and why i use it?

Before discussing RAG and GraphRAG,

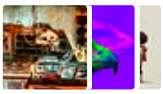
13 min read · Mar 12, 2024

332 2

Lists

Generative AI Recommended Reading

52 stories · 1119 saves

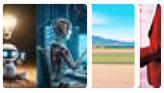


What is ChatGPT?

9 stories · 371 saves

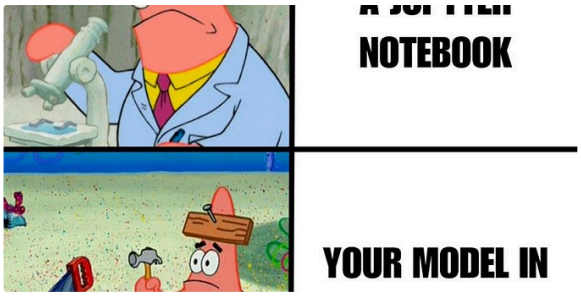
The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 398 saves



Natural Language Processing

1499 stories · 1026 saves



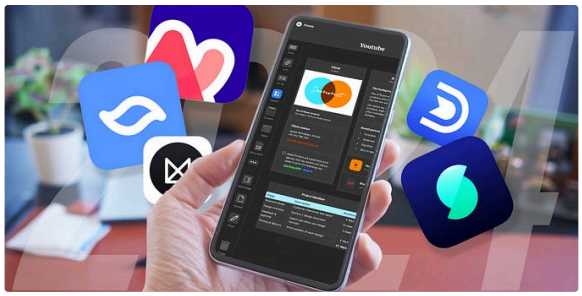
Mandar Karhade, MD. PhD. in Towards AI

Why RAG Applications Fail in Production

It worked as a prototype; then all went down!

· 8 min read · Mar 19, 2024

1.6K 18



Shu Omi in Shu Omi's Blog

My AI-Powered Productivity Setup for 2024

If you're looking for a new productivity setup for this year, I've got lots of great options for...

· 13 min read · May 30, 2024

708 12

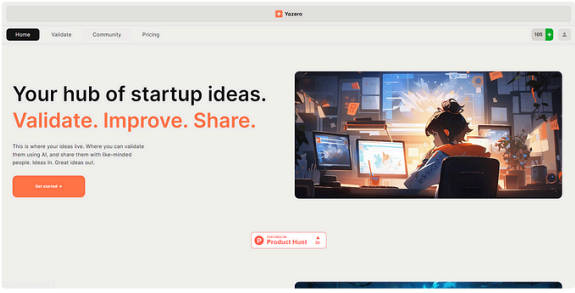
Uzman Ali

This Guy Makes $1M+ per Year With 0 Employees

Ivan Kutskir—the world's most underrated solopreneur

· 6 min read · May 11, 2024

2.4K 39



Artem Shelamanov in Python in Plain English

How I Built My First AI Startup (With No Experience)

My detailed journey with advices on building startups.

9 min read · Apr 30, 2024

2K 47

See more recommendations