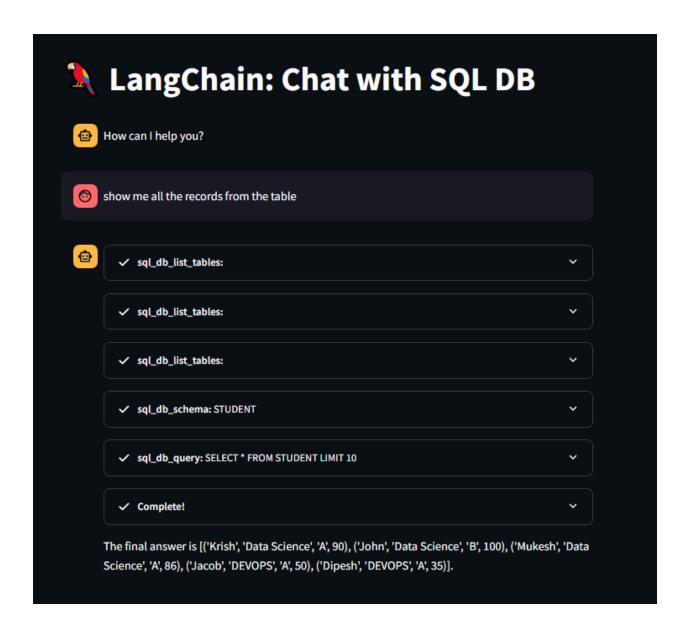
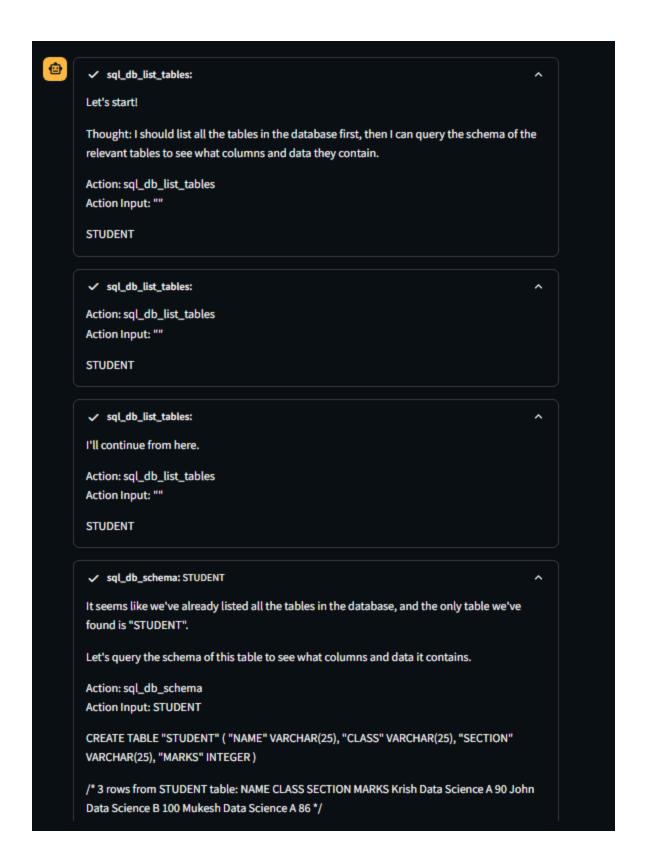
# **SQL Streamlit APP**





# **Import libraries:**

#### pip install mysql-connector-python SQLAlchemy

import streamlit as st

from pathlib import Path

from langchain\_community.agent\_toolkits.sql.base import create\_sql\_agent from langchain\_community.utilities import SQLDatabase

from langchain.agents.agent\_types import AgentType

from langchain\_community.callbacks.streamlit import StreamlitCallbackHandler

from langchain\_community.agent\_toolkits.sql.toolkit import SQLDatabaseToolk it

from sqlalchemy import create\_engine import sqlite3

from langchain\_groq import ChatGroq

#### Path Handling (pathlib.Path)

- Purpose: Works with file/folder paths across different operating systems.
- Key Use: Locate SQL database files on disk.

#### SQL Agent ( langchain.agents. create\_sql\_agent )

- Purpose: Creates an AI that can talk to SQL databases.
- **Key Use**: Lets users ask questions in plain English (e.g., "Show me top customers") and converts them to SQL queries.

#### **SQL Database Connection**

### langchain. sql\_database . SQLDatabase

- Purpose: Connects LangChain to your SQL database.
- Key Use: Bridges between natural language and database queries.

#### Agent Types ( langchain.agents.agent\_types.AgentType )

• Purpose: Defines how the Al thinks (e.g., should it explain its steps?).

• **Key Use**: Usually set to **AgentType.OPENAI\_FUNCTIONS** for SQL tasks.

#### Streamlit Callbacks ( langchain.callbacks.StreamlitCallbackHandler )

- Purpose: Shows the AI's thought process in the Streamlit app.
- **Key Use**: Displays live SQL queries and intermediate steps.

#### SQL Tools ( langchain.agents.agent\_toolkits.SQLDatabaseToolkit )

What it does: A LangChain toolkit that provides a set of pre-configured tools specifically for interacting with an SQLDatabase object. These tools typically include capabilities like listing tables, describing table schemas, and executing SQL queries. This toolkit is often used by create\_sql\_agent.

#### Database Engine ( sqlalchemy.create\_engine )

- **Purpose**: Creates a connection to your SQL database.
- **Key Use**: Required for LangChain to interact with the database.

What it does: Part of the SQLAlchemy library (a popular SQL toolkit and Object Relational Mapper for Python). <a href="mailto:create\_engine">create\_engine</a> is used to establish a connection pool to a database, providing a consistent way to connect to various SQL databases (e.g., SQLite, PostgreSQL, MySQL) using a URL-like connection string. LangChain's SQLDatabase often uses this under the hood.

# **Page Config**

```
st.set_page_config(page_title="LangChain: Chat with SQL DB", page_icon =" \ ")
st.title(" \ LangChain: Chat with SQL DB")
```

## **SQL Settings**

LOCALDB="USE\_LOCALDB" MYSQL="USE\_MYSQL"

#### LOCALDB="USE\_LOCALDB"

- Think of this as creating a little sticky note that says "USE\_LOCALDB".
- It's just a label or a name that you'll use later in your computer code to refer to the choice of using the simple, small filing cabinet (SQLite).

#### MYSQL="USE\_MYSQL"

Similar to the above, this is another sticky note that says "USE\_MYSQL".



We created these 2 because we want to work with 2 databases.

## **Create Radio Buttons**

radio\_opt=["Use SQLLite 3 Database- Student.db","Connect to you MySQL Da tabase"]

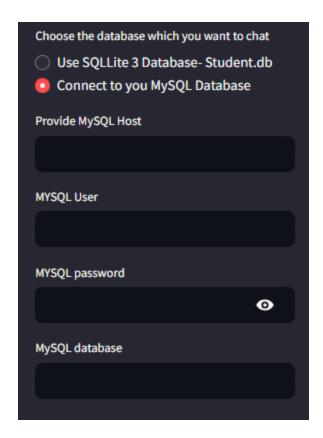
selected\_opt=st.sidebar.radio(label="Choose the database which you want to chat", options=radio\_opt)

Choose the database which you want to chat



Connect to you MySQL Database

```
if radio_opt.index(selected_opt)==1:
    db_uri=MYSQL
    mysql_host=st.sidebar.text_input("Provide MySQL Host")
    mysql_user=st.sidebar.text_input("MYSQL User")
    mysql_password=st.sidebar.text_input("MYSQL password",type="password")
    mysql_db=st.sidebar.text_input("MySQL database")
else:
    db_uri=LOCALDB
```



.index(): This is a special "action" that you can perform on a list. You tell the list: "Hey list, where is this item located?" It will then tell you the position number (the "index") of that item.

#### Ex. If the user chose "Connect to you MySQL Database"

• selected\_opt is now "Connect to you MySQL Database" .

- The code becomes: radio\_opt.index("Connect to you MySQL Database")
  - This will return 1
- Python looks at radio\_opt and says: "Okay, the text 'Connect to you MySQL Database' is at position 1 in this list."
- So, radio\_opt.index(selected\_opt) evaluates to 1

## **API Key**

```
api_key=st.sidebar.text_input(label="GRoq API Key",type="password")

if not db_uri:
    st.info("Please enter the database information and uri")

if not api_key:
    st.info("Please add the groq api key")
```

## **LLM Model**

```
## LLM model
Ilm= ChatGroq(groq_api_key=api_key,model_name="Llama3-8b-8192",streami
ng=True)
```

# configure\_db

• This code defines a special function called **configure\_db**. Its job is to set up a connection to a database so your chatbot can talk to it.

```
@st.cache_resource(ttl="2h")
def configure_db(db_uri,mysql_host=None,mysql_user=None,mysql_passwor
d=None,mysql_db=None):
  if db_uri==LOCALDB:
```

```
dbfilepath=(Path(__file__).parent/"student.db").absolute()
    print(dbfilepath)
    creator = lambda: sqlite3.connect(f"file:{dbfilepath}?mode=ro", uri=True)
    return SQLDatabase(create_engine("sqlite:///", creator=creator))

elif db_uri==MYSQL:
    if not (mysql_host and mysql_user and mysql_password and mysql_db):
        st.error("Please provide all MySQL connection details.")
        st.stop()
    return SQLDatabase(create_engine(f"mysql+mysqlconnector://{mysql_user}))

er}:{mysql_password}@{mysql_host}/{mysql_db}"))
```

#### @st.cache\_resource(ttl="2h")

- This is like a "memory helper" for the program
- It remembers the database connection for 2 hours ("ttl" means "time to live")
- Makes the program faster by not reconnecting every time

def configure\_db( db\_uri , mysql\_host=None, mysql\_user=None,
mysql\_password=None, mysql\_db=None):

• These are extra ingredients that are *only* needed if you pick **MySQL**. None just means "no value yet" or "optional unless needed."

#### **LOCALDB**

#### if db\_uri==LOCALDB

• If the user selects LOCALDB, we set up the file path to .db file.

#### (Path(\_\_file\_\_).parent/"student.db").absolute()

• Return the **absolute path** like:

program is in a folder called my\_app/chatbot.py, this takes you to the my\_app folder.

.absolute(): This gets you the full, exact address of that student.db file, from the very top of your computer's file system (like C:/Users/YourName/my\_app/student.db).



!! (Path(\_file\_).parent/"student.db").absolute() this code will not work inside Jupyter NB.

In Jupyter, Use →

db\_path = (Path.cwd() / "student.db").absolute()

d:\Python Env\LangChain\student.db

creator = lambda : sqlite3.connect (f"file:{dbfilepath}?mode=ro",
uri=True) :

- sqlite3.connect(...): This is the actual command to connect to an SQLite database.
- file: Says it's a file path.
- {dbfilepath}: This plugs in the exact file address you just calculated.
- ?mode=ro: This is a very important part that means "read-only mode." It's like saying, "When connecting to this database, only allow reading data, not changing or adding data." This is often used for security or to ensure the agent doesn't accidentally modify your database.
- uri=True: This tells sqlite3.connect that the first argument is a URI string.

## WHY lambda?

create\_engine doesn't want an already open connection. It wants a recipe to open connections.

- When you call sqlite3.connect(...), it immediately opens a database connection. It's like immediately picking up the phone and calling someone.
- create\_engine from SQLAlchemy is designed to manage a pool of connections. It
  doesn't want you to give it an already open connection. Instead, it wants a
  way to create new connections whenever it needs one from its pool. It's like

giving create\_engine a set of instructions for "how to call someone" rather than a phone call already in progress.

```
return SQLDatabase( create_engine ("sqlite:///", creator=creator)):
```

This line takes the blueprint you just made and actually builds the "database-talking tool" that the AI agent will use.

- "sqlite:///": This is a simple connection string that tells create\_engine that you want to set up plumbing for an SQLite database. The triple slash /// indicates a relative path to a local database file (or an in-memory database if no path is given).
- create\_engine is part of SQLAlchemy, a tool to work with databases.
- SQLDatabase wraps it to make it easier to use in your app.
- So this line returns a working connection to your local SQLite database.

### **MYSQL**

if not (mysql\_host and mysql\_user and mysql\_password and mysql\_db): st.error("Please provide all MySQL connection details.") st.stop()

- Halt the execution if all parameters are not provided.
- In Streamlit, st.stop() is a function that immediately halts the execution of the app at the point where it's called. It's useful when you want to prevent the rest of your Streamlit script from running under certain conditions—like waiting for user input or validating data.

# return SQLDatabase ( create\_engine (f"mysql+mysqlconnector://{mysql\_user}: {mysql\_password}@{mysql\_host}/{mysql\_db}"))

- This is very similar to the SQLite return, but for MySQL.
- mysql+mysqlconnector://: Tells the engine it's MySQL and to use the mysqlconnector library to connect.
- {mysql\_user}:{mysql\_password}: Your username and password for the MySQL database.
- @{mysqLhost}: The address of the computer where the MySQL database lives (e.g., localhost if it's on your own computer, or an IP address).
- /{mysql\_db}: The specific name of the database within MySQL that you want to connect to.

#### mysql+mysqlconnector://

: This part specifies two things:

- mysql: This is the **dialect name**. It tells SQLAlchemy that you want to connect to a **MySQL database**.
- +mysqlconnector: This is the **driver name**. It tells SQLAlchemy which specific Python library it should use to establish the actual low-level connection to the MySQL server. In this case, it's telling SQLAlchemy to use the mysql-connector-python library (which you'd typically install via pip install mysql-connector-python). Other common MySQL drivers include pymysql (then it would be mysql+pymysql://), or mysqldb (then mysql+mysqldb://).



So, mysql+mysqlconnector:// means: "I want to talk to a MySQL database, and please use the mysql-connector-python library to do the talking."

# What is f"mysql+mysqlconnector://user:password@host/db" and what is this URL?

This is a complete **database connection URL (or URI)** formatted according to a standard that SQLAlchemy understands. It's essentially a compact way to provide all the necessary information for a program to log into and access a specific database.

## Configure:

```
if db_uri==MYSQL:
    db=configure_db(db_uri,mysql_host,mysql_user,mysql_password,mysql_db)
else:
    db=configure_db(db_uri)
```

#### It checks:

- If you selected MySQL → Connects using all the login details (username, password, etc.)
- 2. **If you selected SQLite** → Connects to the simple file database ( student.db ) with no extra details needed
- 1. if db\_uri == MYSQL:
  - Meaning: "If the user picked MySQL..."
  - What Happens?
    - The code calls configure\_db() with all the MySQL login details:
      - mysql\_host (Where the database is, like an address)
      - mysql\_user (Your username, like a library card number)
      - mysql\_password (Your secret password)
      - mysql\_db (Which database to use, like choosing a bookshelf)

#### 2. else:

- Meaning: "If the user DID NOT pick MySQL (so they picked SQLite)..."
- What Happens?
  - The code calls configure\_db() with just db\_uri (no extra details needed).
  - Since SQLite is just a file (student.db), it doesn't need passwords or usernames!

## **Toolkit & Agent**

```
db = configure_db (...)
```

## **Set Message Bubble**

```
if "messages" not in st.session_state or st.sidebar.button("Clear message hist
ory"):
    st.session_state["messages"] = [{"role": "assistant", "content": "How can I
help you?"}]

for msg in st.session_state.messages:
    st.chat_message(msg["role"]).write(msg["content"])
```

- Start with assistant bubble saying: How can I help you?
- Then write each message in respective bubble

#### **Use query:**

user\_query=st.chat\_input(placeholder="Ask anything from the database")

```
if user_query:
    st.session_state.messages.append({"role": "user", "content": user_query})
    st.chat_message("user").write(user_query)

with st.chat_message("assistant"):
    streamlit_callback=StreamlitCallbackHandler(st.container())
    response=agent.run(user_query,callbacks=[streamlit_callback])
    st.session_state.messages.append({"role":"assistant","content":respons
e})
    st.write(response)
```

- If user enters a message, append that message to st.session\_state.messages
- & Print it in the user's chat bubble

#### with st.chat\_message("assistant"):

- Creates an assistant chat bubble
- StreamlitCallbackHandler → Shows the Al's thinking process (like when you see ChatGPT "typing")
- agent.run() → The AI actually processes your question and generates an answer

## **Full Code:**

```
import streamlit as st
from pathlib import Path
from langchain_community.agent_toolkits.sql.base import create_sql_agent
from langchain_community.utilities import SQLDatabase
from langchain.agents.agent_types import AgentType
from langchain_community.callbacks.streamlit import StreamlitCallbackHandl
er
from langchain_community.agent_toolkits.sql.toolkit import SQLDatabaseToolk
it
from sqlalchemy import create_engine
import sqlite3
from langchain_groq import ChatGroq
st.set_page_config(page_title="LangChain: Chat with SQL DB", page_icon
="(")"
st.title(" \ LangChain: Chat with SQL DB")
LOCALDB="USE_LOCALDB"
MYSQL="USE_MYSQL"
radio_opt=["Use SQLLite 3 Database- Student.db","Connect to you MySQL Da
tabase"]
selected_opt=st.sidebar.radio(label="Choose the database which you want to
chat", options=radio_opt)
if radio_opt.index(selected_opt) = =1:
  db_uri=MYSQL
  mysql_host=st.sidebar.text_input("Provide MySQL Host")
  mysql_user=st.sidebar.text_input("MYSQL User")
  mysql_password=st.sidebar.text_input("MYSQL password",type="passwor
d")
  mysql_db=st.sidebar.text_input("MySQL database")
else:
  db_uri=LOCALDB
```

```
api_key=st.sidebar.text_input(label="GRoq API Key",type="password")
if not db_uri:
  st.info("Please enter the database information and uri")
if not api_key:
  st.info("Please add the groq api key")
## LLM model
Ilm=ChatGroq(groq_api_key=api_key,model_name="Llama3-8b-8192",streami
ng=True)
@st.cache_resource(ttl="2h")
def configure_db(db_uri,mysql_host=None,mysql_user=None,mysql_passwor
d=None,mysql_db=None):
  if db_uri==LOCALDB:
    dbfilepath=(Path(__file__).parent/"student.db").absolute()
    print(dbfilepath)
    creator = lambda: sqlite3.connect(f"file:{dbfilepath}?mode=ro", uri=True)
    return SQLDatabase(create_engine("sqlite:///", creator=creator))
  elif db_uri==MYSQL:
    if not (mysql_host and mysql_user and mysql_password and mysql_db):
       st.error("Please provide all MySQL connection details.")
       st.stop()
    return SQLDatabase(create_engine(f"mysql+mysqlconnector://{mysql_us
er}:{mysql_password}@{mysql_host}/{mysql_db}"))
if db_uri==MYSQL:
  db=configure_db(db_uri,mysql_host,mysql_user,mysql_password,mysql_db)
else:
  db=configure_db(db_uri)
## toolkit
toolkit=SQLDatabaseToolkit(db=db,llm=llm)
```

```
agent=create_sql_agent(
  Ilm=Ilm,
  toolkit=toolkit,
  verbose=True,
  agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION
)
if "messages" not in st.session_state or st.sidebar.button("Clear message hist
ory"):
  st.session_state["messages"] = [{"role": "assistant", "content": "How can I
help you?"}]
for msg in st.session_state.messages:
  st.chat_message(msg["role"]).write(msg["content"])
user_query=st.chat_input(placeholder="Ask anything from the database")
if user_query:
  st.session_state.messages.append({"role": "user", "content": user_query})
  st.chat_message("user").write(user_query)
  with st.chat_message("assistant"):
    streamlit_callback=StreamlitCallbackHandler(st.container())
    response=agent.run(user_guery,callbacks=[streamlit_callback])
    st.session_state.messages.append({"role":"assistant","content":respons
e})
    st.write(response)
```

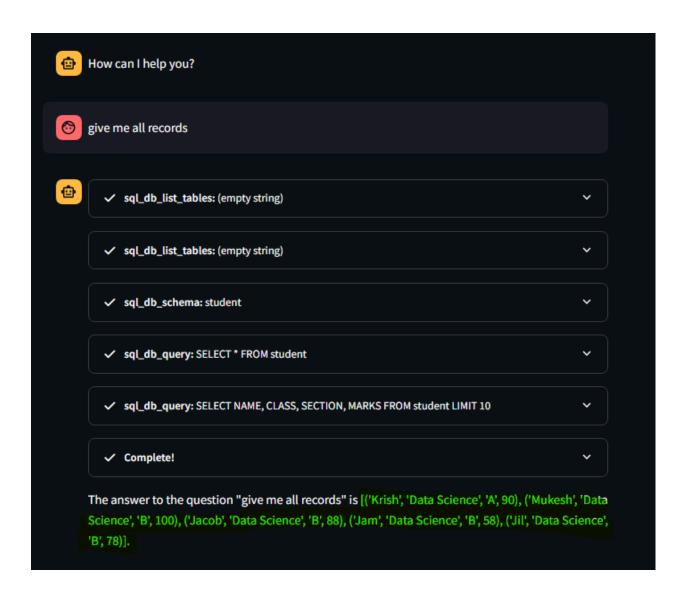
# **Connect to MySQL Server**

localhost:3306

root

**Password** = 12345

STUDENT



#### **Original Table:**

