





# SQL Streamlit APP



## LangChain: Chat with SQL DB

 How can I help you?

 show me all the records from the table



✓ sql\_db\_list\_tables: ▾

✓ sql\_db\_list\_tables: ▾


✓ sql\_db\_list\_tables: ▾

✓ sql\_db\_schema: STUDENT ▾

✓ sql\_db\_query: SELECT \* FROM STUDENT LIMIT 10 ▾

✓ Complete! ▾

The final answer is [('Krish', 'Data Science', 'A', 90), ('John', 'Data Science', 'B', 100), ('Mukesh', 'Data Science', 'A', 86), ('Jacob', 'DEVOPS', 'A', 50), ('Dipesh', 'DEVOPS', 'A', 35)].



✓ sql\_db\_list\_tables: ^

Let's start!

Thought: I should list all the tables in the database first, then I can query the schema of the relevant tables to see what columns and data they contain.

Action: sql\_db\_list\_tables

Action Input: ""

STUDENT

✓ sql\_db\_list\_tables: ^

Action: sql\_db\_list\_tables

Action Input: ""

STUDENT

✓ sql\_db\_list\_tables: ^

I'll continue from here.

Action: sql\_db\_list\_tables

Action Input: ""

STUDENT

✓ sql\_db\_schema: STUDENT ^

It seems like we've already listed all the tables in the database, and the only table we've found is "STUDENT".

Let's query the schema of this table to see what columns and data it contains.

Action: sql\_db\_schema

Action Input: STUDENT

CREATE TABLE "STUDENT" ( "NAME" VARCHAR(25), "CLASS" VARCHAR(25), "SECTION" VARCHAR(25), "MARKS" INTEGER )

/\* 3 rows from STUDENT table: NAME CLASS SECTION MARKS Krish Data Science A 90 John Data Science B 100 Mukesh Data Science A 86 \*/

## 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 SQLDatabaseToolkit
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 AI 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). `create_engine` 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 SQLite 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



Use SQLite 3 Database- Student.db



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

```

The screenshot shows a dark-themed Streamlit sidebar. At the top, it says "Choose the database which you want to chat". There are two radio buttons: "Use SQLite 3 Database- Student.db" (unselected) and "Connect to you MySQL Database" (selected, indicated by a red dot). Below the radio buttons, there are four text input fields: "Provide MySQL Host", "MYSQL User", "MYSQL password" (with a toggle eye icon), and "MySQL database".

`.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
llm= ChatGroq(groq_api_key=api_key,model_name="Llama3-8b-8192",streaming=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_password=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 SQLiteDatabase(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 SQLiteDatabase(create_engine(f"mysql+mysqlconnector://{mysql_us
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:

`.parent` : This means "**go up one level**" from that paper's location. So, if your 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 SQLiteDatabase( 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.
- `SQLiteDatabase` 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.
- `@{mysql_host}` : 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:

1. **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

```
## toolkit
toolkit=SQLDatabaseToolkit(db=db,llm=llm)

agent=create_sql_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION
)
```



`db = configure_db (...)`

## Set Message Bubble

```
if "messages" not in st.session_state or st.sidebar.button("Clear message history"):
    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":response})
        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 AI'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 SQLite 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
llm=ChatGroq(groq_api_key=api_key,model_name="Llama3-8b-8192",streaming=True)

@st.cache_resource(ttl="2h")
def configure_db(db_uri,mysql_host=None,mysql_user=None,mysql_password=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 SQLiteDatabase(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 SQLiteDatabase(create_engine(f"mysql+mysqlconnector://{mysql_user}:{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(
    llm=llm,
    toolkit=toolkit,
    verbose=True,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION
)

if "messages" not in st.session_state or st.sidebar.button("Clear message history"):
    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_query,callbacks=[streamlit_callback])
        st.session_state.messages.append({"role":"assistant","content":response})
    st.write(response)

```


## Connect to MySQL Server


localhost:3306


root

**Password** = 12345

STUDENT

 How can I help you?

 give me all records



✓ sql\_db\_list\_tables: (empty string) ▾

✓ sql\_db\_list\_tables: (empty string) ▾

✓ sql\_db\_schema: student ▾

✓ sql\_db\_query: SELECT \* FROM student ▾

✓ sql\_db\_query: SELECT NAME, CLASS, SECTION, MARKS FROM student LIMIT 10 ▾

✓ Complete! ▾

The answer to the question "give me all records" is `[('Krish', 'Data Science', 'A', 90), ('Mukesh', 'Data Science', 'B', 100), ('Jacob', 'Data Science', 'B', 88), ('Jam', 'Data Science', 'B', 58), ('Jil', 'Data Science', 'B', 78)].`

## Original Table:

	NAME	CLASS	SECTION	MARKS
▶	Krish	Data Science	A	90
	Mukesh	Data Science	B	100
	Jacob	Data Science	B	88
	Jam	Data Science	B	58
	Jil	Data Science	B	78