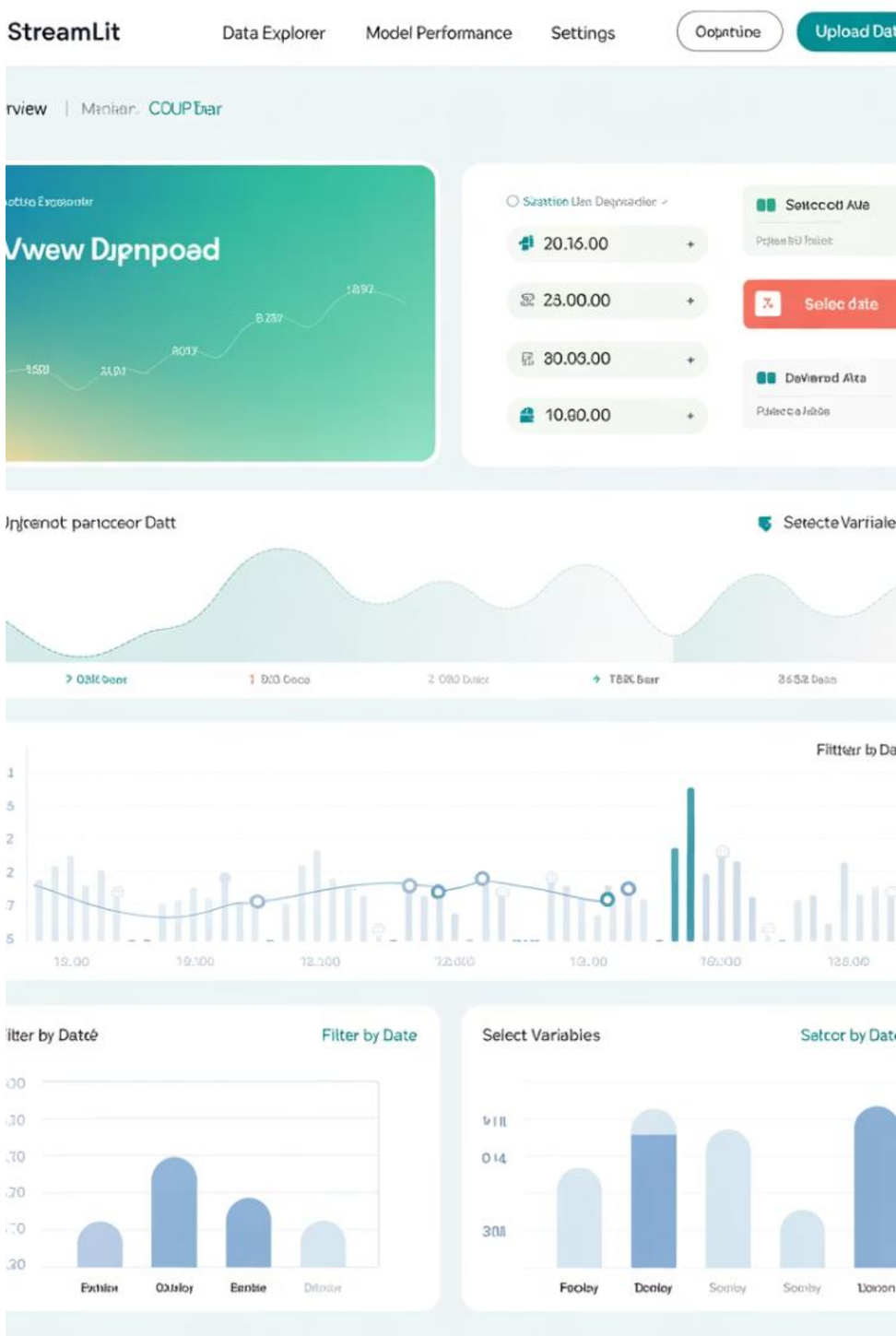
Streamlit and SQLite3: A Beginner-Friendly Tutorial

JZ by Jonathan Zouari



Introduction to Streamlit and SQLite3

2

Streamlit

Open-source Python library for creating interactive web apps without web development knowledge.

SQLite3

Lightweight database included with Python. Perfect for small to medium applications.

Easy Integration

Build data-driven apps in minutes with minimal coding coding experience required. required.

Quick Deployment

Turn Python scripts into shareable web applications applications with simple commands.

In this tutorial, we'll build a Streamlit app that collects user information and stores it in a SQLite database. Perfect for beginners wanting to beginners wanting to create their first data application.

Step 1: Installing Streamlit

Installing Streamlit is straightforward using pip, Python's package manager. Before you begin, ensure you have Python installed on your system. installed on your system.

1 Verify Python Installation

Open your terminal or command prompt and check your Python version:

```
python --version
```

2 Install Streamlit Package

Use pip to install the Streamlit library: library:

```
pip install streamlit
```

3 Verify Installation

Confirm Streamlit was installed correctly (In Terminal):

```
streamlit hello
```

This command launches Streamlit's Streamlit's demo app in your default default web browser.

Once installation is complete, you're ready to begin building your first Streamlit application!

Troubleshooting

If you encounter issues running Streamlit after installation, try these common solutions:

Command Not Found

If the streamlit command isn't found, found, ensure your Python scripts scripts directory is in your PATH environment variable.

Alternative Command

Try running **python -m streamlit hello** as an alternative to the direct direct streamlit command.

Virtual Environment

Consider using a virtual environment for your projects to prevent package conflicts between different installations.

For more detailed troubleshooting information, visit the official documentation at **docs.streamlit.io**.

Step 2: Creating a Simple Streamlit App

Now that Streamlit is installed, let's build our first app. We'll start with the basics before adding database functionality.

Create Project Directory

Make a new folder named "streamlit_sqlite_tutorial" to organize your project files.

Create Python Script

Create a new file named "app.py" using your your favorite code editor.

Write Basic App Code

Add the core imports and structure needed for a Streamlit application.

```
import streamlit as st

st.title("User Information App")       # App title
st.write("Welcome! This is a simple Streamlit app.")  # Display some text
```

In the code above, we imported Streamlit and gave our app a title and a welcome message. The st.title() function displays a large header, and st.write() can display text (or other data) on the app.

Running Your Streamlit Application

Launch Command

In your terminal, navigate to your project project folder and type **streamlit run app.py**

Browser Opens

Streamlit starts a local server and automatically opens your app in a new new browser tab.

Hot Reload

Edit your code and save. Streamlit detects detects changes and instantly updates updates your app.

Your browser should display "User Information App" and your welcome message. This hot-reload feature makes development fast and interactive - perfect for iterative design.

Step 3: Connecting to a SQLite Database

To use SQLite with your Streamlit app, you first need to import the necessary library

```
import sqlite3
```

Next, Create (or connect to) a database file:

```
# Connect to local SQLite database (creates the file if it doesn't exist)
conn = sqlite3.connect('example.db')
c = conn.cursor()
```

Create a table for our data (if it doesn't already exist). We want a table to store user information (name and age). Add the following code to `app.py` after establishing the connection: connection:

```
# Create a table for users if it doesn't exist
c.execute("""
    CREATE TABLE IF NOT EXISTS users (
        name TEXT,
        age INTEGER
    )
""")
conn.commit()
```

Step 4: Creating Forms in Streamlit

Now that we've connected to our SQLite database, let's create a form to collect user information. Streamlit provides a convenient way to group input widgets into forms.

1. **Use `st.form` to define a form section**. Inside your `app.py`, after the database setup code, add:

```
st.header("Add a New User")  # Section header in the app

# Define a form for input
with st.form(key="user_form"):
    name = st.text_input("Name", placeholder="Enter your name")
    age = st.number_input("Age", min_value=0, max_value=120, step=1)
    submit_btn = st.form_submit_button(label="Submit")
```

Here's what this does:

- `st.header("Add a New User")` adds a subsection title in the app interface.
- `with st.form(key="user_form"):` creates a form container. All input elements inside this `with` block will be part of the form.
- `st.text_input("Name", ...)` creates a text input field labeled "Name". We've provided a placeholder text to guide the user.
- `st.number_input("Age", ...)` creates a numeric input for age, with a minimum of 0 and maximum of 120 (just an example range).
- `st.form_submit_button("Submit")` adds a submit button to the form. This function returns `True` when the form is submitted (when the user clicks "Submit").

How forms work: The app will not process the inputs inside the form until the user clicks the **Submit** button. When that happens, all the data from the form fields is sent in one batch to the Streamlit app backend. We can detect this by checking the value of `submit_btn`.

Creating Forms in Streamlit (part B)

Now that we've connected to our SQLite database, let's create a form to collect user information. Streamlit provides a convenient way to group input widgets into forms.

2. **Handle the form submission.** Right **after** the `with st.form(...):` block (i.e., not indented, so it's outside the form), we add code to handle what happens when the form is submitted:

```
if submit_btn:
    # When the form is submitted, insert the new data into the database
    c.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age))
    conn.commit()
    st.success(f"Stored new entry: **{name} (Age {age})**")
```

In this code:

- We check `if submit_btn:` which will be `True` only when the user has pressed the Submit button.
- We execute an SQL INSERT command to add the provided `name` and `age` into the **users** table. Notice we use placeholders `(? , ?)` in the SQL and pass a tuple `(name, age)` as the parameters. This is a parameterized query, which is a safe way to insert variables into SQL statements (it helps prevent SQL injection and handles quoting for us).
- We call `conn.commit()` again to save the insertion to the database.
- We use `st.success()` to show a success message in the app, confirming to the user that their entry was stored. We format the message to bold the name and show the age for clarity.

Step 5: Displaying Entries from the Database

After inserting data, it's useful to display the current contents of the database so the user can see the list of entries they have added. We will query the database for all stored records and display them in our app, either as a table or a list.

Steps to display the data:

1. **Query the database for all entries.** We can use a SELECT query to get all rows from the **users** table. Add the following code *after* the form handling section (so it runs on every app rerun):

```
# Fetch and display all entries from the database
c.execute("SELECT name, age FROM users")
rows = c.fetchall()
```

Here, `c.execute("SELECT name, age FROM users")` runs a query to retrieve all records. `c.fetchall()` retrieves all results of the query as a list of tuples. Each tuple corresponds to a row (with name and age).

Step 5: Displaying Entries from the Database (Part B)

1. **Display the results in the app.** We can use Streamlit elements to show this data. For example:

```
st.subheader("Current Users in Database")
if rows:
    # Display the entries in a table format
    import pandas as pd
    df = pd.DataFrame(rows, columns=["Name", "Age"])
    st.table(df)
else:
    st.write("No entries yet.")
```

1. In this snippet:
   - We add a subheader "Current Users in Database" for clarity.
   - We check `if rows:` – if the list is not empty, we proceed to display the data. If it's empty (no entries yet), we display a friendly message.
   - We import pandas as pd (pandas is a popular library for data manipulation; if you don't have it installed, you can install it via `pip install pandas`). We create a DataFrame from the `rows` list, specifying column names "Name" and "Age". This tabular structure is convenient for display.
   - We use `st.table(df)` to render a static table in the app showing all the current entries. Streamlit will nicely format the pandas DataFrame as a table. (Alternatively, `st.dataframe(df)` could be used for an interactive table where users can scroll and sort, but `st.table` is sufficient for a simple static display.)
   - If there are no rows, we simply print "No entries yet." to indicate the database is empty.

Running Your Streamlit App

After creating your Streamlit application, you need to run it using the Streamlit command-line interface:

```
streamlit run app.py
```

When you execute this command in your terminal:

- Streamlit will start a local web server
- Your default web browser will automatically open with your app
- By default, the app runs on http://localhost:8501/
- Any changes to your code will trigger an automatic reload of the app

To stop the application, simply press Ctrl+C in the terminal where the app is running.