LESSON 57

Introduction to SQLite Databases

80 minutes (2 periods) | Python Web Backend | Outcome 3.2 | Outcome 3.3 | Outcome 2.16 | Outcome 2.18 | Outcome 1.22 | Outcome 2.20 | Outcome 2.15

WHAT THIS LESSON DOES

In this lesson, you'll explore the essentials of databases by setting up SQLite in your Flask project. Learn to create a database, define tables, insert data via forms, and test data persistence, ensuring information remains even after app restarts.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
3.2	create a basic relational database to store and retrieve a variety of forms of data types
3.3	use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs
2.16	use data types that are common to procedural high-level languages
2.18	collect, store and sort both continuous and discrete data
1.22	read, write, test, and modify computer programs
2.20	identify and fix/debug warnings and errors in computer code and modify as required
2.15	explain what is meant by the World Wide Web (WWW) and the Internet, including the client server model, hardware components and communication protocols

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts of databases and their role in data persistence.
- Learn to set up and configure a SQLite database within a Flask project.
- Develop skills in creating and managing database tables using SQL commands.
- Acquire the ability to integrate user input with a database through Flask routes.
- Gain proficiency in retrieving and displaying stored data in a web application.

BEFORE THE LESSON

- Run through the whole build yourself first: create the venv, pip install flask, run create_db.py, submit the form and confirm users.db populates. Flask installing behind the school filter is the usual snag.
- Confirm SQLite needs no install (it ships with Python) so students do not waste time hunting for it.
- Have the four files clear in your head: create_db.py, app.py, templates/form.html and templates/users.html. Students routinely forget the templates folder.

Introduction to SQLite Databases

HOW THE LESSON RUNS

- Introduction
- Create a Project Folder
- Create the Virtual Environment
- Install Flask
- What is a Database?
- What is SQL?
- Create the Database and Table
- Set Up Flask App and Form
- Handle POST Request and Insert Data
- Test Data Persistence
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Create a Project Folder. Insist everyone names the folder identically and opens it as the VS Code workspace root. Relative paths like 'users.db' break later if the folder is not the open root.
- 3. Create the Virtual Environment. Walk the room while the venv creates. Check the interpreter shown bottom-left of VS Code points at .venv before anyone installs Flask, or the install lands in the wrong place.
- 4. Install Flask. Say out loud that SQLite is built into Python so no second install is needed. Students expect to install a database and go looking for one.
- 5. What is a Database? Use the spreadsheet analogy but name the terms deliberately: table, row (record), column (field), primary key. These map straight onto the CREATE TABLE they write next.
- 6. What is SQL? Frame SQL as the language you use to talk to the database, separate from Python. The Python sqlite3 module just carries SQL strings across.
- 7. Create the Database and Table. Read the CREATE TABLE line aloud, mapping id, name and age to the concepts just taught. Stress that running this file once creates users.db as a real file they can see in the tree.
- 8. Set Up Flask App and Form. Check everyone has a templates folder with form.html before running. A missing template gives a confusing error, not a clear one.
- 9. Handle POST Request and Insert Data. Point at the ? placeholders and (name, age) tuple. Explain briefly that this is why we never glue user input straight into the SQL string: it blocks injection.
- 10. Test Data Persistence. Make the persistence test physical: add a user, fully stop the app, restart it, visit /users. Data still there is the whole point of the lesson.
- 11. Challenge. Steer students to reuse the SELECT pattern with a WHERE clause and a placeholder. The trap is building the query by string concatenation; require placeholders.

WHAT TO WATCH FOR

- Students expect data typed into a form to persist like a saved file automatically, and do not see why a database is needed. Contrast a Python variable, gone when the program ends, with a row in users.db. Restart the app to show the variable is lost but the row survives.
- Confusing what runs where: they think SQL is Python, or that the form validates the data. Separate the layers on the board: HTML form in the browser, Flask route in Python, SQL sent to the database. Note age is cast with int() because form data always arrives as text.
- Believing the app works because the form submits and shows 'User added successfully!', without checking the data actually stored. Require every student to visit /users after submitting. A success message is not evidence; the visible row is.

DIFFERENTIATION

- Emerging: If the venv or Flask install is fighting them, pair them with a working setup so they still reach the database concepts. Give them create_db.py pre-written so the SQL is the focus. Have them describe, in plain words, what a table, row and column are before they write any SQL.
- Developing: Ask them to trace one submitted form value from the browser input, through request.form, into the INSERT, and back out via SELECT.
- Proficient: Point them at the search challenge, then push toward Higher Level: ask why placeholders matter and have them describe an SQL injection attempt the ? syntax prevents. Let them add UPDATE or DELETE routes, evidence they could reuse in a coursework project needing data persistence.