LESSON 58

Retrieving and Displaying Data

80 minutes (2 periods) | Python Web Backend | Outcome 3.2 | Outcome 3.3 | Outcome 2.16 | Outcome 2.18 | Outcome 2.7 | Outcome 1.22 | Outcome 2.15

WHAT THIS LESSON DOES

In this lesson, you'll learn to query and display data in your Flask app using your SQLite database. You'll create a hobbies table, extend the users table, install a database viewer extension, and build a search feature to find users.

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
2.7	implement algorithms using a programming language to solve a range of problems
1.22	read, write, test, and modify computer programs
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 how to query and retrieve data from a SQLite database using SQL statements.
- Develop skills to display database data dynamically in a Flask web application.
- Learn to modify database structures by adding columns and creating new tables.
- Master the use of SQL JOINs to connect and present related data across multiple tables.
- Implement search functionality to filter and display data based on user input.

BEFORE THE LESSON

- Confirm every machine has the previous lesson's FlaskSQLiteIntro folder with a working users.db and app.py; students who lost it cannot start.
- Test the alexcvzz SQLite extension installs through the school filter on one machine before class, since the Extensions marketplace is sometimes blocked.
- Run through the ALTER TABLE and CREATE TABLE scripts yourself first: a second run of the add-column script throws a duplicate column error, so know the message students will see.

Retrieving and Displaying Data

HOW THE LESSON RUNS

- Introduction
- Install SQLite Extension
- Add Columns to Users Table
- Create Hobbies Table
- Add Form and Route for Hobbies
- View Hobbies List
- Associate Hobby with User
- SQL JOINs
- Display Email and Hobbies with the Users
- What is Querying Data?
- Create Search Page
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Install SQLite Extension. Show the right-click Open Database step on the projector once. The SQLITE EXPLORER panel is where students lose it. Point at where it appears so they can verify their own data all lesson.
- 3. Add Columns to Users Table. Stress this script runs once only. Re-running it errors because the column already exists. If a student's script fails, check whether they ran it twice rather than debugging the SQL.
- 4. Create Hobbies Table. Point out IF NOT EXISTS makes this one safe to re-run, unlike the previous step. Have them refresh the SQLite explorer to see the empty hobbies table appear before moving on.
- 5. Add Form and Route for Hobbies. Highlight the ? placeholder in the INSERT and the tuple of values. This is parameterised query safety, not string joining. Name it as protection against SQL injection at Higher Level.
- 6. View Hobbies List. Walk through the Jinja for-loop and hobby[0], hobby[1] index access. fetchall returns tuples, so the numbers are column positions. Confirm they added at least one hobby first, or the table shows empty.
- 7. Associate Hobby with User. Show why the home route now has to query hobbies before rendering the form: the dropdown is populated from the database. The option value is hobby_id, the displayed text is the name.
- 8. SQL JOINs. Draw two small tables on the board and physically trace one INNER JOIN and one LEFT JOIN. The difference is what happens to unmatched rows. Note SQLite does not support RIGHT JOIN, so LEFT is what they will use.
- 9. Display Email and Hobbies with the Users. Explain why LEFT JOIN not INNER: a user with no hobby should still appear. Test with one user who has no hobby to prove the row still shows with a blank hobby column.
- 10. What is Querying Data? Anchor SELECT, FROM, WHERE and LIKE against the queries they just wrote rather than as abstract syntax. Highlight the % wildcards in LIKE as the mechanism behind partial-match search.
- 11. Create Search Page. Point out the if/elif on search_type builds a different query per option. Ask why the code uses a parameter placeholder for the search term rather than pasting it into the query string.
- 12. Challenge. Set this as the finish-early task, not a whole-class step. A nav bar repeated on every page is the natural cue to introduce a shared base template later. Leave that door open.

WHAT TO WATCH FOR

- Students re-run the ALTER TABLE script when something goes wrong, then get confused by a duplicate-column error and assume their whole database is broken. Explain that ALTER TABLE add-column runs once. Show them the SQLite explorer to confirm the column is already there, then have them move on rather than re-running.
- Students expect INNER JOIN and LEFT JOIN to give the same result, so a user with no hobby vanishing from the list looks like a bug. Have them run the same query with INNER then LEFT and compare row counts. The disappearing user is the whole point of the difference.
- Students think a page that displays correctly means the query is correct, missing that they only tested users who happen to have hobbies. Insist they add one user with no hobby and one with an unusual name, then search for both. Correct-looking output on easy data proves nothing.

DIFFERENTIATION

- Emerging: Pair students whose users.db is missing or broken with a working machine so they can still follow the JOIN and query concepts even if their own app lags behind. Give them the exact run order as a checklist: update columns, create hobbies, add a hobby, then add a user. Order matters here more than syntax.
- Developing: Ask them to predict what a query returns before running it, then check. Reasoning about SELECT and WHERE output is the step between copying and understanding.
- Proficient: Have them rewrite the search route to allow a blank search term returning all users, and explain the edge case. This is the kind of input handling worth documenting in a Coursework Project. Challenge them to explain why parameterised queries with ? are safer than building the query string themselves, at Higher Level depth.