Files vs Databases

Python Course · Module 6.5

ElnurBDa

Module 6.5 — When to Use Files, When to Use Databases

Goals:

- Compare text/CSV files with relational databases
- Understand when simple files are enough
- See when you should move to **SQLite / SQL**

Think of this as a decision guide before you learn SQL syntax.

Recap — What Files Do Well

From Module 6, files are great for:

- Logs (`activity.log`)
- Configuration (`config.ini`, `.env`)
- Small exports (`report.txt`, `users.csv`)
- One-off data exchange between tools

Strengths:

- Very simple to inspect with any text editor
- Easy to copy, email, archive
- No extra server process required

Weaknesses:

- Harder to **search/filter** efficiently
- No built-in concept of relationships between rows
- Concurrency (many writers/readers) can be tricky

Recap — What Databases Do Well

From Module 7, relational databases (like SQLite) are great for:

- Structured data in **tables** with columns
- Fast querying with **indexes**
- Enforcing rules via constraints (`NOT NULL`, `UNIQUE`, `FOREIGN KEY`)
- Multiple readers (and some writers) at the same time

Strengths:

- Complex filters and aggregations with `WHERE`, `ORDER BY`, `JOIN`, etc.
- Central source of truth for your data
- Easier to evolve schema over time

Weaknesses:

- More concepts to learn (schema, types, constraints)
- Needs careful backups and migrations
- Not ideal for raw logs or one-off binary blobs

Scenario 1 – Log Tracker

Use **files** when:

- You’re writing simple append-only logs
- You mostly read them in full (or by lines)
- You don’t need complex queries

Possible upgrade:

- Move from plain text to **CSV** for slightly more structure
- Only move to a database if you need to filter heavily (e.g., by user/date)

Rule of thumb:

> If you only ever read all entries and scroll, a file is fine.

Scenario 2 – Score Tracker / Student Manager

For students, courses, scores:

- Many records (potentially hundreds or thousands)
- Need to search by email, course, status
- Might need to join with other tables later (attendance, tasks)

Files:

- Quick CSV exports for sharing or backup
- OK for tiny datasets with simple needs

Database:

- Better when you need to update scores often
- Safer with unique constraints on email
- Enables joins with other entities (e.g., students ↔ courses)

In this course, Module 7 moves these use-cases into SQLite.

Comparison Table – Files vs SQLite

Aspect	Files (txt/CSV)	SQLite (DB file)
Structure	Free-form / simple rows	Tables with columns & types
Validation	Manual in code	Constraints in schema
Queries	Manual loops/filtering	SQL (SELECT, WHERE, JOIN)
Concurrency	Basic, not coordinated	Better but still limited
Tools	Editors, Excel, scripts	sqlite3 CLI, DB browsers
Best for	Logs, configs, exports	Core application data

Takeaway:

- Start with files when prototyping
- Migrate to SQLite once queries or relations become complex

Migration Example — Users CSV → Users Table

Suppose you have users.csv:

```
id,username,email,score
1,admin,admin@example.com,99
2,guest,guest@example.com,42
```

You want to move to SQLite:

1. Design schema:

```sql
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    score INTEGER DEFAULT 0
);
```

2. Write a one-time Python script that:

- Reads users.csv with csv.reader
- Inserts each row into users using **parameterized** queries

After that, your main app can ignore the CSV and work only with SQLite.
