From a Table to an API

A database, the Python that talks to it, and the server in front

E190BO · AI-Amplified Prototyping

PAUSE THE VIDEO

At the end of the last video, the server restarted and every idea was gone.

Where would you put them so they stay?

Outline

1. **A database is a file you ask questions**
2. **A second table:** `JOIN` and `GROUP BY`
3. **The same table, from Python**
4. **Put it behind the API**
5. **Run it, use it, restart it**

One file, `app.db`, the whole way through.

1 · A database is a file you ask questions

A table: columns, rows, and a key

```sql
CREATE TABLE idea (
  id    INTEGER PRIMARY KEY,          -- the database picks it
  title TEXT NOT NULL,
  body  TEXT NOT NULL DEFAULT ''
);

INSERT INTO idea (title) VALUES ('Automatic plant waterer');
INSERT INTO idea (title, body)
  VALUES ('Sticker vending machine', 'One sticker, one quarter');
```

- Every **row** is one idea. The **primary key**, `id`, is unique: we never sent one, so the database picked 1 and 2.
- **SQL** (Structured Query Language) is how you ask. **SQLite** keeps the whole database in one file.

demos/databases-1-one-table/one-table.sql

▶ LIVE Ask it questions

1. Terminal: `sqlite3 app.db`, then `.headers on` and `.mode box` (just display settings).
2. Paste the `CREATE TABLE` and the two `INSERT`s.
3. `SELECT * FROM idea;`
4. `SELECT title FROM idea WHERE id = 2;`
5. `SELECT * FROM idea WHERE id = 99;`
6. `.quit`, then `ls`: there's `app.db`.

id	title	body
1	Automatic plant waterer	
2	Sticker vending machine	One sticker, one quarter

PAUSE THE VIDEO

`SELECT * FROM idea WHERE id = 99;` came back with nothing.

Is that an error?

And who decides what an API says about it?

2 · A second table

Votes get their own table

```sql
CREATE TABLE vote (
  id       INTEGER PRIMARY KEY,
  idea_id INTEGER NOT NULL REFERENCES idea(id),   -- which idea: a foreign key
  voter   TEXT NOT NULL
);

INSERT INTO vote (idea_id, voter) VALUES (1, 'ada'), (1, 'grace'), (2, 'ada');
```

id	idea_id	voter
1	1	ada
2	1	grace
3	2	ada

- One row **per vote**. `idea_id` is a **foreign key**: it holds another table's primary key.

demos/databases-2-two-tables/two-tables.sql

`JOIN` follows a key; `GROUP BY` groups rows

```
SELECT idea.title, vote.voter
FROM vote
  JOIN idea ON idea.id = vote.idea_id;
```

title	voter
Automatic plant waterer	ada
Automatic plant waterer	grace
Sticker vending machine	ada

```
SELECT idea_id, COUNT(*) AS votes
FROM vote GROUP BY idea_id;
```

idea_id	votes
1	2
2	1

- `JOIN` lines up each vote with the idea its `idea_id` points at, so a vote can show a title.
- `GROUP BY` collapses rows that share a value, and `COUNT` counts each group.

PAUSE THE VIDEO

You want each idea’s title, with how many votes it has.

Which tools from this chapter do you need?

3 · The same table, from Python

A class is a table

```python
class Idea(SQLModel, table=True):
    id: int | None = Field(
        default=None, primary_key=True)
    title: str
    body: str = ""
```

```sql
CREATE TABLE idea (
  id    INTEGER PRIMARY KEY,
  title TEXT NOT NULL,
  body  TEXT NOT NULL DEFAULT ''
);
```

- **SQLModel** is an **ORM** (object-relational mapper): you write classes, and it writes the SQL.
- `table=True` makes the class a table, named `idea`. Each attribute is a column.
- `id` is `None` until the row is saved; then the database fills it in.

An engine, a session, and the SQL it sends

```python
from sqlmodel import SQLModel, Field, Session, create_engine, select

# ... class Idea, as on the last slide ...

engine = create_engine("sqlite:///app.db", echo=True)   # echo: print the SQL
SQLModel.metadata.create_all(engine)  # make any table that isn't there yet

with Session(engine) as s:
    s.add(Idea(title="Bike lock that texts you"))
    s.commit()
    for idea in s.exec(select(Idea)):
        print(idea.id, idea.title)
```

```
INSERT INTO idea (title, body) VALUES (?, ?)
('Bike lock that texts you', '')
SELECT idea.id, idea.title, idea.body FROM idea
```

- The **engine** points at the file. A **session** is one conversation: `add`, then `commit` to save.

demos/databases-3-orm/db.py; log trimmed to the SQL

▶ LIVE Run it on the same file

1. Terminal: `python db.py`, in the folder with `app.db`.
2. Point at the log: `INSERT INTO idea ...`, then `SELECT ...`.
3. Point at the output:

```
1 Automatic plant waterer
2 Sticker vending machine
3 Bike lock that texts you
```

4. “Two of those we typed by hand in `sqlite3`. Python found them.”

PAUSE THE VIDEO

You add `author: str = ""` to the `Idea` class and run `db.py` again.

Does the table get an `author` column?

4 · Put it behind the API

The list becomes a table

Last video: a list in memory

```
ideas = []

@app.get("/ideas")
def list_ideas():
    return ideas

@app.post("/ideas", status_code=201)
def create_idea(payload: IdeaCreate):
    idea = {"id": len(ideas) + 1,
            **payload.model_dump()}
    ideas.append(idea)
    return idea
```

Now: the table in `app.db`

```
engine = create_engine("sqlite:///app.db", echo=True)
SQLModel.metadata.create_all(engine)

@app.get("/ideas")
def list_ideas():
    with Session(engine) as s:
        return s.exec(select(Idea)).all()

@app.post("/ideas", status_code=201)
def create_idea(payload: IdeaCreate):
    idea = Idea(**payload.model_dump())
    with Session(engine) as s:
        s.add(idea)
        s.commit()
        s.refresh(idea)   # read back the id
        return idea
```

demos/databases-4-api/main.py; above the routes, the Idea table class and the IdeaCreate request class

What each piece is for

- `IdeaCreate` is still what a client may send, with no `id`. `Idea` is what's stored. Two classes, as in the last video.
- `Idea(**payload.model_dump())` turns the request into a row.
- **One session per request**, opened and closed inside the function: every request stands alone.
- `return` a row, or a list of rows, and FastAPI turns it into JSON.

5 · Run it, use it, restart it

▶ LIVE Ideas that survive a restart

1. Terminal 1: `uvicorn main:app --reload`, next to the same `app.db`.
2. Browser: `/docs` → `GET /ideas` → **Execute**: the three ideas from before.
3. `POST /ideas` → `{"title": "Desk fan that follows you"}` → **201**, `"id": 4`.
4. Terminal 1: the `INSERT INTO idea ...` it sent.
5. Terminal 1: **Ctrl-C**, then start it again. `GET /ideas`: **all four**.
6. Terminal 2: `sqlite3 app.db "SELECT * FROM idea;"`: the row the API wrote.

The whole path

```
browser —HTTP→ uvicorn → your function → Session —SQL→ app.db
browser ←JSON— uvicorn ← your function ← Session ←rows— app.db
```

- **uvicorn** holds the port and loops (the last video).
- **Your function** decides what to do: 201, 404, which rows.
- **The ORM** writes the SQL; **SQLite** keeps the data in a file.

PAUSE THE VIDEO

Where does the desk fan idea live now?

What would delete it, and what wouldn’t?

What Lab 5 adds

Lab 5	From
A `Vote` class, with `idea_id: int = Field(foreign_key="idea.id")`	chapters 2 and 3
`vote_count` on every idea in `GET /ideas`	`JOIN` + `GROUP BY`, together: the class notebook
`GET /ideas/{id}`: `s.get(Idea, idea_id)`, and a 404 when it's `None`	the pause in chapter 1
`POST /ideas/{id}/vote`	all of it