https://www.digitalocean.com/community/tutorials/how-to-use-an-sqlite-database-in-a-flask-application erstelle mir auf basis dieser Quelle einen Beispielcode um cur.execute, fetchall() und render_template zu veranschaulichen

Unten ist ein **minimales Beispielprojekt**, das genau die drei Dinge aus der DigitalOcean-Anleitung zeigt:

- **cur.execute(...)** (Cursor-API)
- **fetchall()** (alle Zeilen holen)
- **render_template(...)** (Daten ins Jinja-Template geben)

(Orientiert an dem Muster aus der Quelle: conn.execute(...).fetchall() + render_template(...) und dem Cursor-Insert mit cur.execute(...).)

Projektstruktur

```
flask_sqlite_demo/
  app.py
  init_db.py
  schema.sql
  templates/
    index.html
```

---

1) schema.sql

```
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  title TEXT NOT NULL,
  content TEXT NOT NULL
);
```

---

2) init_db.py (zeigt: cur.execute)

```
import sqlite3

connection = sqlite3.connect("database.db")

with open("schema.sql", "r", encoding="utf-8") as f:
    connection.executescript(f.read())

cur = connection.cursor()

# cur.execute(...) veranschaulichen (wie in der DO-Anleitung)
cur.execute(
    "INSERT INTO posts (title, content) VALUES (?, ?)",
    ("First Post", "Content for the first post"),
)
cur.execute(
    "INSERT INTO posts (title, content) VALUES (?, ?)",
    ("Second Post", "Content for the second post"),
)

connection.commit()
connection.close()

print("database.db wurde erstellt und befüllt.")
```

Ausführen:

```
python init_db.py
```

Das Cursor-Insert mit cur.execute(...) entspricht genau dem Ansatz aus der Quelle.

---

3) app.py (zeigt: execute + fetchall + render_template)

```
import sqlite3
```

```
from flask import Flask, render_template

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect("database.db")
    conn.row_factory = sqlite3.Row  # Zugriff wie dict: row["title"]
    return conn

@app.route("/")
def index():
    conn = get_db_connection()

    # Variante wie in der Quelle: conn.execute(...).fetchall()
    posts = conn.execute("SELECT id, created, title, content FROM posts ORDER BY created DESC").fetchall()

    conn.close()

    # render_template(...) veranschaulichen: posts ins Template geben
    return render_template("index.html", posts=posts)

if __name__ == "__main__":
    app.run(debug=True)
```

Hier siehst du das zentrale Pattern aus der Quelle: SQL ausführen → fetchall() → an render_template() übergeben.

---

4) templates/index.html (zeigt Nutzung der übergebenen Daten)

```
<!doctype html>
```

```
<html lang="de">
 <head>
  <meta charset="utf-8" />
  <title>Posts</title>
 </head>
 <body>
  <h1>Posts aus SQLite</h1>

  {% if posts %}
   <ul>
    {% for post in posts %}
     <li>
      <strong>{{ post["title"] }}</strong>
      <small>(ID: {{ post["id"] }}, {{ post["created"] }})</small>
      <div>{{ post["content"] }}</div>
     </li>
    {% endfor %}
   </ul>
  {% else %}
   <p>Keine Posts gefunden.</p>
  {% endif %}
 </body>
</html>
```

Kurz erklärt (was wo passiert)

- **init_db.py**: Cursor holen → **cur.execute(...)** Inserts → commit().
- **app.py**: Connection → **conn.execute(...).fetchall()** → render_template("index.html", posts=posts).

Wenn du willst, kann ich dir das gleiche Beispiel noch einmal so umbauen, dass **auch das SELECT über cur = conn.cursor(); cur.execute(…); cur.fetchall()** läuft (damit du beide Varianten nebeneinander siehst).