LEHRERINFO ZUM VIDEO

Eine Datenbank-App mit SQL und Python

FACH	STUFE	DAUER	VIDEO
Informatik	Gymnasium · Sek II	14 min	youtube.com/@Erklaerbaer

01 Lernziele

Nach dem Video können die Schülerinnen und Schüler …

- aus Anforderungen ein normalisiertes Schema entwickeln (ER-Modell → Tabellen in 3. Normalform),
- Tabellen mit PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE und CHECK anlegen und die SQLite-Besonderheit `PRAGMA foreign_keys` erklären,
- komplexe Abfragen schreiben (mehrere JOINs, GROUP BY mit HAVING, LEFT JOIN, Unterabfrage),
- eine Python-App mit `sqlite3` und parametrisierten Abfragen bauen,
- SQL-Injection erklären und verhindern sowie Passwörter mit Salt und langsamem Hash speichern,
- Abfragen mit Index und Ausführungsplan beschleunigen und Transaktionen einsetzen.

02 Voraussetzungen

ER-Modell, JOIN, GROUP BY, Normalformen (Folge 12 · Advanced); Index, binäre Suche, Baum, Hash (Folgen 6 und 9 · Advanced); Klassen (Folge 10 · Advanced); Hash, Salt, SQL-Injection (Folge 13 · Advanced); SQLite und DB Browser (Folge 16 · Grundlagen). Python 3 mit dem mitgelieferten Modul `sqlite3`.

03 Ablauf des Videos

ZEIT	KAPITEL	INHALT UND BEISPIEL
0:00	Einstieg	schnell geschrieben gegen schnell, korrekt, sicher · Ident · Fahrplan
0:31	Was du brauchst	Rückblick 12A, 06A/09A, 10A, 13A
0:55	ER-Modell	sechs Entitäten · Beziehungen 1 : n · Exemplare als eigene Tabelle · 3. NF
1:45	Tabellen mit Regeln	Constraints · CHECK-Test · PRAGMA foreign_keys · Merksatz
2:48	Abfragen für Profis	überfällig · Top 5 · nie ausgeliehen · Unterabfrage PAUSE-FRAGE > 3 offene Ausleihen
4:10	Die Python-App	`sqlite3`, Klasse LibraryRepository · Menü · Suche «Potter»
5:08	Sicherheit	Injection mit f-String gegen Platzhalter PAUSE-FRAGE · Salt + scrypt · Rollen · Merksatz
6:35	Schnell bleiben	10 000 Ausleihen · SCAN gegen SEARCH USING INDEX · Messung · Schreibkosten · Merksatz
7:41	Transaktionen	UPDATE + INSERT atomar · Rollback-Test · Überweisung
8:30	Empfehlungen	Self-Join auf loans · Momo → Der Hobbit, Anne Frank, Harry Potter
9:05	Rückblick	Bausteine aus 6, 9–14 · Mediathek-App komplett
9:37	Prüfungsfallen	fünf Fallen (siehe 04)
10:36	Zusammenfassung	Kernaussagen · zurück zum Einstieg
11:16	Quiz	4 Fragen · Endkarte Folge 17

04 Prüfungsfallen

- SQLite prüft Fremdschlüssel automatisch – erst mit `PRAGMA foreign_keys = ON`, und zwar pro Verbindung.
- Eingaben in Anführungszeichen setzen reicht gegen Injection – nur Platzhalter bzw. Prepared Statements sind sicher.
- Mehr Indizes = immer schneller – jeder Index bremst INSERT/UPDATE und braucht Platz.
- SHA-256 ohne Salt reicht für Passwörter – nötig sind langsame, gesalzene Verfahren (Argon2, bcrypt, scrypt).

- `COUNT(*)` und `COUNT(spalte)` sind gleich – `COUNT(spalte)` zählt keine NULL-Werte (Beispieldaten: 13 Ausleihen, 9 Rückgaben).

05 Einsatz im Unterricht

Mitbauen: `python beispieldaten.py` legt `mediathek.db` an; danach die Abfragen im DB Browser oder in Python nachvollziehen.

Messen: `python beispieldaten.py 10000` und die Suche mit und ohne Index selbst messen (`time.perf_counter`).

Sicherheit: Unsichere und sichere Suche gegenüberstellen – ausschliesslich in der eigenen Übungsdatenbank.

Erweitern: Vormerkungen einbauen: Wer etwas vormerkt, wird bei der Rückgabe benachrichtigt (Aufgabe 6).

Hinweis: Alle Ergebnisse im Video stammen aus dem Code im Anhang (Python 3.12, SQLite 3.49) mit den erfundenen Beispieldaten; «überfällig» bezieht sich auf den 30.10.2026. Die Messwerte (Suche 0,21 ms ohne und 0,016 ms mit Index bei 10 013 Ausleihen; 20 000 INSERT in 13 ms ohne und 35 ms mit drei Indizes) wurden auf unserem Rechner mit einer Datenbank im Arbeitsspeicher gemessen und hängen vom Gerät ab. SQL-Injection nur am eigenen Übungsprojekt zeigen; rechtlich siehe Folge 13 (CH Art. 143bis StGB, DE § 202a StGB, AT § 118a StGB).

06 Quellen

- SQLite-Dokumentation: Foreign Key Support, CREATE INDEX, EXPLAIN QUERY PLAN – sqlite.org/docs.html
- Python-Dokumentation: `sqlite3`, `hashlib.scrypt`, `hmac.compare_digest` – docs.python.org/3/library
- OWASP Cheat Sheets: SQL Injection Prevention, Password Storage – cheatsheetseries.owasp.org
- E. F. Codd (1970): A Relational Model of Data for Large Shared Data Banks, CACM 13(6)

NAME _____ KLASSE _____ DATUM _____

Eine Datenbank-App mit SQL und Python

1 Constraints ●○○

Welche Regel verhindert jeweils den Fehler? a) zwei Mitglieder mit demselben Anmeldenamen · b) eine Ausleihe für ein Exemplar, das es nicht gibt · c) eine Rückgabe vor der Ausleihe · d) ein Medium ohne Titel

2 Abfragen ●●○

Schreibe a) alle Medien vom Typ «game» · b) die Anzahl Ausleihen pro Klasse · c) alle Mitglieder mit mehr als drei offenen Ausleihen · d) alle Medien, die gerade kein verfügbares Exemplar haben.

3 Injection erkennen ●●○

`"SELECT * FROM members WHERE login = '" + login + "'"` – **Was** passiert bei der Eingabe `' OR '1'='1`? **Schreibe** die sichere Fassung mit `sqlite3`.

4 Index ●●○

Die Abfrage «alle Ausleihen, die bis morgen fällig sind» ist langsam. **Welchen** Index legst du an? **Was** zeigt `EXPLAIN QUERY PLAN` vorher und nachher? **Nenne** einen Nachteil.

5 Transaktion ●●○

Die Rückgabe besteht aus zwei Änderungen. **Welche**? **Was** geht schief, wenn das Programm zwischen den beiden abstürzt und keine Transaktion verwendet?

6 Vormerkungen ●●●

Ergänze `LibraryRepository` um `reserve(media_id, member_id)` und `next_reservation(media_id)` (wer hat am frühesten vorgemerkt?). **Welche** Regel im Schema verhindert doppelte Vormerkungen?

Lösungen

1 Constraints

a) `UNIQUE` auf `login` · b) Fremdschlüssel `REFERENCES copies(id)` mit `PRAGMA foreign_keys = ON` · c) `CHECK (return_date IS NULL OR return_date >= loan_date)` · d) `NOT NULL` auf `title`

2 Abfragen

a) `SELECT * FROM media WHERE kind = 'game';` · b) `SELECT c.name, COUNT(*) FROM loans l JOIN members mb ON mb.id = l.member_id JOIN classes c ON c.id = mb.class_id GROUP BY c.id;` · c) `SELECT member_id, COUNT(*) FROM loans WHERE return_date IS NULL GROUP BY member_id HAVING COUNT(*) > 3;` · d) `SELECT md.title FROM media md JOIN copies cp ON cp.media_id = md.id GROUP BY md.id HAVING SUM(cp.available) = 0;`

3 Injection

Es entsteht `… WHERE login = '' OR '1'='1'`: Die Bedingung ist immer wahr, die Abfrage liefert alle Mitglieder. Sicher: `db.execute("SELECT * FROM members WHERE login = ?", (login,))` – die Eingabe bleibt ein Wert.

4 Index

`CREATE INDEX idx_loans_due ON loans(due_date);` · vorher `SCAN loans`, nachher `SEARCH loans USING INDEX idx_loans_due (due_date<?)` · Nachteil: jedes INSERT und UPDATE muss den Index mitpflegen, er braucht Speicherplatz.

5 Transaktion

1. `UPDATE loans SET return_date = …` · 2. `UPDATE copies SET available = 1`. Ohne Transaktion kann die Ausleihe als zurückgegeben gelten, das Exemplar aber weiter als ausgeliehen markiert bleiben (oder umgekehrt). Mit `with self.db:` werden beide gemeinsam festgeschrieben oder beide zurückgerollt.

6 Vormerkungen

`reserve`: `INSERT INTO reservations (media_id, member_id, created) VALUES (?, ?, ?)` in einem `with self.db:`-Block · `next_reservation`: `SELECT member_id FROM reservations WHERE media_id = ? ORDER BY created LIMIT 1` · Doppelte Vormerkungen verhindert `UNIQUE (media_id, member_id)` (Fehler `IntegrityError` abfangen).

Q Quiz im Video

1 B (Platzhalter) · 2 A (wie die Datenbank vorgeht) · 3 B (referentielle Integrität) · 4 C (alle Änderungen rückgängig)

Anhang A: schema.sql

Schema in 3. Normalform. Wird von `LibraryRepository.create_schema()` eingelesen.

```
-- Mediathek-App – Serie Informatik, Folge 16 (Advanced) · Schema in 3. Normalform
-- In SQLite prüft die Datenbank Fremdschlüssel erst nach: PRAGMA foreign_keys = ON (pro Verbindung!)

CREATE TABLE classes (
    id       INTEGER PRIMARY KEY,
    name     TEXT NOT NULL UNIQUE,
    teacher  TEXT NOT NULL
);

CREATE TABLE members (
    id        INTEGER PRIMARY KEY,
    name      TEXT NOT NULL,
    class_id  INTEGER REFERENCES classes(id),
    role      TEXT NOT NULL CHECK (role IN ('student', 'teacher')),
    login     TEXT NOT NULL UNIQUE,
    pw_salt   BLOB NOT NULL,
    pw_hash   BLOB NOT NULL
);

CREATE TABLE media (
    id      INTEGER PRIMARY KEY,
    title   TEXT NOT NULL,
    author  TEXT,
    year    INTEGER CHECK (year BETWEEN 1400 AND 2100),
    kind    TEXT NOT NULL CHECK (kind IN ('book', 'game'))
);

CREATE TABLE copies (
    id         INTEGER PRIMARY KEY,
    media_id   INTEGER NOT NULL REFERENCES media(id),
    available  INTEGER NOT NULL DEFAULT 1 CHECK (available IN (0, 1))
);

CREATE TABLE loans (
    id           INTEGER PRIMARY KEY,
    copy_id      INTEGER NOT NULL REFERENCES copies(id),
    member_id    INTEGER NOT NULL REFERENCES members(id),
    loan_date    TEXT NOT NULL,
    due_date     TEXT NOT NULL,
    return_date  TEXT,
    CHECK (due_date >= loan_date),
    CHECK (return_date IS NULL OR return_date >= loan_date)
);

CREATE TABLE reservations (
    id         INTEGER PRIMARY KEY,
    media_id   INTEGER NOT NULL REFERENCES media(id),
    member_id  INTEGER NOT NULL REFERENCES members(id),
    created    TEXT NOT NULL,
    UNIQUE (media_id, member_id)
);
```

Anhang B: mediathek_app.py

Python 3, nur Standardbibliothek. Getestet mit Python 3.12 und SQLite 3.49.

```
# Mediathek-App – Serie Informatik, Folge 16 (Advanced)
# Python 3 + sqlite3 · parametrisierte Abfragen · Passwörter mit scrypt und Salt · Transaktionen
import datetime
import hashlib
import hmac
import os
import pathlib
import sqlite3

SCHEMA = pathlib.Path(__file__).with_name("schema.sql")


def hash_password(password, salt=None):
    """Langsamer, gesalzener Hash (scrypt). Gleiche Passwörter ergeben verschiedene Hashes."""
    salt = salt or os.urandom(16)
    digest = hashlib.scrypt(password.encode("utf-8"), salt=salt, n=2**14, r=8, p=1)
    return salt, digest


class LibraryRepository:
    def __init__(self, path="mediathek.db"):
        self.db = sqlite3.connect(path)
        self.db.row_factory = sqlite3.Row
        self.db.execute("PRAGMA foreign_keys = ON")        # sonst prüft SQLite keine Fremdschlüssel

    def create_schema(self):
        self.db.executescript(SCHEMA.read_text(encoding="utf-8"))

    # ── Mitglieder und Anmeldung ──────────────────────────────────────
    def add_member(self, name, class_id, role, login, password):
        salt, digest = hash_password(password)
        with self.db:                                      # Transaktion: alles oder nichts
            cur = self.db.execute(
                "INSERT INTO members (name, class_id, role, login, pw_salt, pw_hash) VALUES (?, ?, ?, ?, ?, ?)",
                (name, class_id, role, login, salt, digest))
        return cur.lastrowid

    def check_login(self, login, password):
        row = self.db.execute("SELECT id, role, pw_salt, pw_hash FROM members WHERE login = ?", (login,)).fetchone()
        if row is None:
            return None
        _, digest = hash_password(password, row["pw_salt"])
        return (row["id"], row["role"]) if hmac.compare_digest(digest, row["pw_hash"]) else None

    # ── Suchen ────────────────────────────────────────────────────────
    def search(self, text):
        return self.db.execute(
            "SELECT id, title, author, year FROM media WHERE title LIKE ? ORDER BY title",
            ("%" + text + "%",)).fetchall()

    def find_member_unsafe(self, name):
        # NUR ZUR DEMONSTRATION: Eingabe wird Teil des SQL-Textes → SQL-Injection
        return self.db.execute(f"SELECT id, name FROM members WHERE name = '{name}'").fetchall()

    def find_member(self, name):
        return self.db.execute("SELECT id, name FROM members WHERE name = ?", (name,)).fetchall()

    # ── Ausleihen ─────────────────────────────────────────────────────
    def lend(self, copy_id, member_id, today=None, days=21):
        today = today or datetime.date.today()
        due = today + datetime.timedelta(days=days)
        with self.db:                                      # UPDATE + INSERT gemeinsam oder gar nicht
            cur = self.db.execute("UPDATE copies SET available = 0 WHERE id = ? AND available = 1", (copy_id,))
            if cur.rowcount == 0:
                raise ValueError("Exemplar ist nicht verfügbar")
            self.db.execute(
                "INSERT INTO loans (copy_id, member_id, loan_date, due_date) VALUES (?, ?, ?, ?)",
                (copy_id, member_id, today.isoformat(), due.isoformat()))

    def give_back(self, loan_id, today=None):
        today = (today or datetime.date.today()).isoformat()
        with self.db:
            row = self.db.execute("SELECT copy_id FROM loans WHERE id = ? AND return_date IS NULL", (loan_id,)).fetchone()
            if row is None:
                raise ValueError("Keine offene Ausleihe mit dieser Nummer")
            self.db.execute("UPDATE loans SET return_date = ? WHERE id = ?", (today, loan_id))
            self.db.execute("UPDATE copies SET available = 1 WHERE id = ?", (row["copy_id"],))

    # ── Abfragen für Profis ───────────────────────────────────────────
    def overdue(self, today=None):
        today = (today or datetime.date.today()).isoformat()
        return self.db.execute("""
            SELECT mb.name, c.name AS class, md.title, l.due_date
            FROM loans l
            JOIN members mb ON mb.id = l.member_id
            JOIN classes c  ON c.id = mb.class_id
            JOIN copies cp  ON cp.id = l.copy_id
            JOIN media md   ON md.id = cp.media_id
            WHERE l.return_date IS NULL AND l.due_date < ?
            ORDER BY l.due_date""", (today,)).fetchall()

    def top5(self, class_name):
        return self.db.execute("""
```

```
            SELECT md.title, COUNT(*) AS n
            FROM loans l
            JOIN members mb ON mb.id = l.member_id
            JOIN classes c  ON c.id = mb.class_id
            JOIN copies cp  ON cp.id = l.copy_id
            JOIN media md   ON md.id = cp.media_id
            WHERE c.name = ?
            GROUP BY md.id ORDER BY n DESC, md.title LIMIT 5""", (class_name,)).fetchall()

    def never_lent(self):
        return self.db.execute("""
            SELECT md.title FROM media md
            LEFT JOIN copies cp ON cp.media_id = md.id
            LEFT JOIN loans l   ON l.copy_id = cp.id
            GROUP BY md.id
            HAVING COUNT(l.id) = 0                          -- COUNT(spalte) zählt keine NULL
            ORDER BY md.title""").fetchall()

    def members_without_loans(self):
        return self.db.execute(
            "SELECT name FROM members WHERE id NOT IN (SELECT member_id FROM loans) ORDER BY name").fetchall()

    def recommend(self, media_id):
        """Wer dieses Medium lieh, lieh auch … (Self-Join auf loans, Baustein aus Folge 14 Advanced)."""
        return self.db.execute("""
            SELECT m2.title, COUNT(DISTINCT a.member_id) AS n
            FROM loans a
            JOIN copies ca ON ca.id = a.copy_id
            JOIN loans b   ON b.member_id = a.member_id
            JOIN copies cb ON cb.id = b.copy_id
            JOIN media m2  ON m2.id = cb.media_id
            WHERE ca.media_id = ? AND cb.media_id <> ?
            GROUP BY m2.id ORDER BY n DESC, m2.title LIMIT 3""", (media_id, media_id)).fetchall()


def menu(repo):
    """Einfaches Konsolenmenü wie beim Taschenrechner aus Folge 15."""
    while True:
        choice = input("1 suchen · 2 überfällig · 3 Empfehlung · 0 Ende: ")
        if choice == "1":
            for r in repo.search(input("Titel enthält: ")):
                print(r["id"], r["title"], r["year"])
        elif choice == "2":
            for r in repo.overdue():
                print(r["name"], r["class"], r["title"], "fällig seit", r["due_date"])
        elif choice == "3":
            for r in repo.recommend(int(input("Medium-Nummer: "))):
                print(r["title"], "·", r["n"], "Personen")
        elif choice == "0":
            break


if __name__ == "__main__":
    menu(LibraryRepository())
```

Anhang C: beispieldaten.py

Erfundene Beispieldaten und Testdaten-Generator. `python beispieldaten.py` oder `python beispieldaten.py 10000`.

```
# Beispieldaten und Testdaten-Generator - Serie Informatik, Folge 16 (Advanced)
# python beispieldaten.py          → mediathek.db mit kleinen, erfundenen Beispieldaten
# python beispieldaten.py 10000    → zusätzlich 10 000 erfundene Ausleihen für die Index-Messung
import datetime
import random
import sys

from mediathek_app import LibraryRepository


def fill(repo):
    db = repo.db
    with db:
        db.executemany("INSERT INTO classes (id, name, teacher) VALUES (?, ?, ?)", [(1, "8a", "Keller"), (2, "8b", "Brunner")])
        db.executemany("INSERT INTO media (id, title, author, year, kind) VALUES (?, ?, ?, ?, ?)", [
            (1, "Der Hobbit", "J. R. R. Tolkien", 1937, "book"), (2, "Momo", "Michael Ende", 1973, "book"),
            (3, "Tschick", "Wolfgang Herrndorf", 2010, "book"), (4, "Die Tribute von Panem", "Suzanne Collins", 2008, "book"),
            (5, "Harry Potter und der Stein der Weisen", "J. K. Rowling", 1997, "book"),
            (6, "Das Tagebuch der Anne Frank", "Anne Frank", 1947, "book"), (7, "Krabat", "Otfried Preussler", 1971, "book"),
            (8, "Schach", None, None, "game")])
        db.executemany("INSERT INTO copies (id, media_id) VALUES (?, ?)", [(i, m) for i, m in enumerate([1, 2, 2, 3, 4, 5, 5, 6, 7, 8], start=1)])
    for name, cls, role, login in [("Noah", 2, "student", "noah"), ("Lea", 1, "student", "lea"), ("Mia", 1, "student", "mia"),
                                   ("Elias", 2, "student", "elias"), ("Ben", 2, "student", "ben"), ("Frau Keller", 1, "teacher", "keller")]:
        repo.add_member(name, cls, role, login, "Demo-" + login + "-2026")
    d = datetime.date
    # (Exemplar, Mitglied, ausgeliehen, zurück oder None) - chronologisch, kein Exemplar doppelt verliehen
    for copy, member, lent, back in [(10, 6, d(2026, 9, 1), d(2026, 9, 2)), (2, 3, d(2026, 9, 1), d(2026, 9, 15)), (3, 2, d(2026, 9, 2), d(2026, 9, 20)),
                                     (8, 2, d(2026, 9, 3), d(2026, 9, 25)), (1, 1, d(2026, 9, 5), d(2026, 9, 14)), (9, 4, d(2026, 9, 8), d(2026, 9, 29)),
                                     (1, 3, d(2026, 9, 16), d(2026, 9, 30)), (2, 1, d(2026, 9, 16), d(2026, 9, 30)), (6, 1, d(2026, 9, 20), None),
                                     (8, 3, d(2026, 10, 1), None), (1, 2, d(2026, 10, 2), d(2026, 10, 20)), (4, 4, d(2026, 10, 5), None),
                                     (9, 2, d(2026, 10, 21), None)]:
        repo.lend(copy, member, today=lent)
        if back:
            loan_id = repo.db.execute("SELECT max(id) FROM loans").fetchone()[0]
            repo.give_back(loan_id, today=back)


def fill_many(repo, n, seed=16):
    rnd = random.Random(seed)
    with repo.db:
        repo.db.executemany("INSERT INTO members (name, class_id, role, login, pw_salt, pw_hash) VALUES (?, ?, 'student', ?, x'00', x'00')",
                            [(f"Test {i}", rnd.choice([1, 2]), f"test{i}") for i in range(1, 501)])
        start = datetime.date(2025, 8, 18)
        rows = []
        for _ in range(n):
            lent = start + datetime.timedelta(days=rnd.randrange(400))
            rows.append((rnd.randrange(1, 11), rnd.randrange(7, 507), lent.isoformat(), (lent + datetime.timedelta(days=21)).isoformat(),
                         (lent + datetime.timedelta(days=rnd.randrange(1, 21))).isoformat()))
        repo.db.executemany("INSERT INTO loans (copy_id, member_id, loan_date, due_date, return_date) VALUES (?, ?, ?, ?, ?)", rows)


if __name__ == "__main__":
    repo = LibraryRepository("mediathek.db")
    repo.create_schema()
    fill(repo)
    if len(sys.argv) > 1:
        fill_many(repo, int(sys.argv[1]))
    print("mediathek.db angelegt")
```