Python Programming - SQLite Database

> **Mục tiêu học tập**: Làm chủ SQLite - database nhẹ, mạnh mẽ để lưu trữ và truy vấn dữ liệu có cấu trúc trong Python.

1. Giới Thiệu Database

1.1. Database là gì?

🤔 Đặt vấn đề

File CSV/TXT có giới hạn khi dữ liệu phức tạp:

```
# ❌ CSV - khó query phức tạp
# scores.csv
S001,Math,85
S001,Physics,92
S002,Math,88
S002,Physics,84

# Tìm học sinh có điểm Math > 85 VÀ Physics > 85?
# Phải đọc hết file, parse, filter → chậm!
```

💡 Giải quyết: Database

Database lưu trữ dữ liệu có cấu trúc, hỗ trợ truy vấn mạnh mẽ với SQL.

```
# ☑ SQLite - Query nhanh
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

# Tìm học sinh thỏa điều kiện - 1 câu SQL!
cursor.execute('''
    SELECT student_id FROM scores
    WHERE subject = 'Math' AND score > 85
    INTERSECT
    SELECT student_id FROM scores
    WHERE subject = 'Physics' AND score > 85
''')

results = cursor.fetchall()
print(results)  # [('S001',)]
```

1.2. SQLite là gì?

SQLite là database engine:

- ☑ **Serverless** - không cần cài server riêng
- ☑ **File-based** - data lưu trong 1 file .db
- ☑ **Built-in Python** - module `sqlite3` có sẵn
- ☑ **Lightweight** - nhẹ, nhanh, đủ cho app nhỏ/vừa
- ☑ **ACID compliant** - đảm bảo tính toàn vẹn dữ liệu

Khi nào dùng SQLite?

- ☑ App desktop, mobile, embedded
- ☑ Prototyping, development
- ☑ Data < vài GB
- ☑ Ít concurrent writes

Khi nào KHÔNG dùng?

- ❌ High concurrency (nhiều writes đồng thời)
- ❌ Large-scale web apps → dùng PostgreSQL, MySQL
- ❌ Network database → dùng client-server DB

1.3. SQL Cơ Bản

SQL (Structured Query Language) là ngôn ngữ truy vấn database.

4 nhóm lệnh chính:

1. **DDL** (Data Definition) - Định nghĩa cấu trúc
    - `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`
2. **DML** (Data Manipulation) - Thao tác dữ liệu
    - `INSERT`, `UPDATE`, `DELETE`
3. **DQL** (Data Query) - Truy vấn
    - `SELECT`
4. **DCL** (Data Control) - Phân quyền
    - `GRANT`, `REVOKE` (ít dùng trong SQLite)

---

2. Module `sqlite3`

2.1. Import và Connect

```
import sqlite3
```

```
# Kết nối database (tạo file nếu chưa có)
conn = sqlite3.connect('mydata.db')

# Kết nối in-memory (test, không lưu file)
conn = sqlite3.connect(':memory:')

# Đóng kết nối
conn.close()
```

2.2. Cursor Object

Cursor thực thi SQL commands và lấy kết quả.

```
import sqlite3

# Kết nối
conn = sqlite3.connect('school.db')

# Tạo cursor
cursor = conn.cursor()

# Thực thi SQL
cursor.execute('SELECT * FROM students')

# Lấy kết quả
results = cursor.fetchall()

# Đóng
cursor.close()
conn.close()
```

2.3. Context Manager (KHUYẾN NGHỊ)

```
import sqlite3

# ☑ Dùng with - tự động commit và close
with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM students')
    results = cursor.fetchall()
# Tự động commit và close
```

⚠ **Lưu ý:** `with` chỉ tự động commit/rollback, KHÔNG close connection. Nhưng thường OK vì connection sẽ close khi script kết thúc.

```
# Nếu muốn chắc chắn close
conn = sqlite3.connect('school.db')
```

```
try:
    cursor = conn.cursor()
    # ... operations ...
    conn.commit()
finally:
    conn.close()
```

---

3. CREATE TABLE - Tạo Bảng

3.1. Cú Pháp

```
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
)
```

3.2. Data Types Trong SQLite

SQLite có 5 storage classes:

Storage Class	Mô tả	Python Type
`NULL`	Null value	`None`
`INTEGER`	Số nguyên	`int`
`REAL`	Số thực	`float`
`TEXT`	Chuỗi	`str`
`BLOB`	Binary data	`bytes`

⚠ **SQLite linh hoạt:** Column type là gợi ý, không ép buộc nghiêm ngặt.

3.3. Constraints (Ràng Buộc)

Constraint	Ý nghĩa
`PRIMARY KEY`	Khóa chính (unique, not null)
`NOT NULL`	Không được null
`UNIQUE`	Giá trị unique
`DEFAULT`	Giá trị mặc định
`CHECK`	Điều kiện kiểm tra
`FOREIGN KEY`	Khóa ngoại (liên kết bảng)

3.4. Ví Dụ CREATE TABLE

```
import sqlite3

with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # Tạo bảng students
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS students (
            student_id TEXT PRIMARY KEY,
            name TEXT NOT NULL,
            age INTEGER,
            city TEXT
        )
    ''')

    # Tạo bảng scores
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS scores (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id TEXT NOT NULL,
            subject TEXT NOT NULL,
            score INTEGER CHECK(score >= 0 AND score <= 100),
            FOREIGN KEY (student_id) REFERENCES students(student_id)
        )
    ''')

    print("Tables created successfully!")
```

Giải thích:

- `IF NOT EXISTS` - chỉ tạo nếu chưa có (tránh lỗi)
- `AUTOINCREMENT` - tự động tăng ID
- `CHECK` - ràng buộc điểm 0-100
- `FOREIGN KEY` - liên kết student_id với bảng students

---

4. INSERT - Thêm Dữ Liệu

4.1. INSERT Một Row

```
import sqlite3

with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # Cách 1: Hard-coded values
    cursor.execute('''
        INSERT INTO students (student_id, name, age, city)
```

```
        VALUES ('S001', 'Nguyễn Văn An', 20, 'Hà Nội')
    ''')

    print("Inserted successfully!")
```

4.2. INSERT Với Parameters (⭐ QUAN TRỌNG)

```
# ☑ KHUYẾN NGHỊ - Parameterized query (tránh SQL injection)
student_id = 'S002'
name = 'Trần Thị Bình'
age = 21
city = 'TP.HCM'

with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # ? là placeholder
    cursor.execute('''
        INSERT INTO students (student_id, name, age, city)
        VALUES (?, ?, ?, ?)
    ''', (student_id, name, age, city))

    print(f"Inserted {cursor.rowcount} row(s)")
```

⚠ **KHÔNG BAO GIỜ** dùng f-string/format để build SQL:

```
# ✗ NGUY HIỂM - SQL Injection!
name = "An'; DROP TABLE students; --"
cursor.execute(f"INSERT INTO students (name) VALUES ('{name}')")

# ☑ AN TOÀN - Parameterized
cursor.execute("INSERT INTO students (name) VALUES (?)", (name,))
```

4.3. INSERT Nhiều Rows

```
students = [
    ('S003', 'Lê Văn Chi', 19, 'Đà Nẵng'),
    ('S004', 'Phạm Thị Dũng', 22, 'Cần Thơ'),
    ('S005', 'Hoàng Văn Em', 20, 'Hà Nội')
]

with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # executemany - insert nhiều rows
    cursor.executemany('''
        INSERT INTO students (student_id, name, age, city)
```

```
        VALUES (?, ?, ?, ?)
    ''', students)

    print(f"Inserted {cursor.rowcount} rows")
```

4.4. INSERT Từ CSV (Migration)

```
import sqlite3
import csv

def migrate_csv_to_db(csv_file, db_file):
    """
    Migrate dữ liệu từ CSV sang SQLite
    """
    with sqlite3.connect(db_file) as conn:
        cursor = conn.cursor()

        # Tạo bảng
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS scores (
                student_id TEXT,
                subject TEXT,
                score INTEGER
            )
        ''')

        # Đọc CSV và insert
        with open(csv_file, 'r', encoding='utf-8') as f:
            reader = csv.reader(f)
            next(reader)  # Skip header

            data = [(row[0], row[1], int(row[2])) for row in reader]

            cursor.executemany('''
                INSERT INTO scores (student_id, subject, score)
                VALUES (?, ?, ?)
            ''', data)

        print(f"Migrated {cursor.rowcount} rows from CSV to database")

# Test
migrate_csv_to_db('scores.csv', 'school.db')
```

---

5. SELECT - Truy Vấn Dữ Liệu

5.1. SELECT Cơ Bản

```
import sqlite3

with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # SELECT tất cả
    cursor.execute('SELECT * FROM students')
    results = cursor.fetchall()

    for row in results:
        print(row)
    # ('S001', 'Nguyễn Văn An', 20, 'Hà Nội')
    # ('S002', 'Trần Thị Bình', 21, 'TP.HCM')
    # ...
```

5.2. Fetch Methods

Method	Mô tả
`fetchone()`	Lấy 1 row (tuple)
`fetchmany(n)`	Lấy n rows (list of tuples)
`fetchall()`	Lấy tất cả rows

```
cursor.execute('SELECT * FROM students')

# Lấy 1 row
row = cursor.fetchone()
print(row)  # ('S001', 'Nguyễn Văn An', 20, 'Hà Nội')

# Lấy 3 rows
rows = cursor.fetchmany(3)
print(rows)  # [(...), (...), (...)]

# Lấy tất cả
all_rows = cursor.fetchall()
print(all_rows)
```

5.3. SELECT Với WHERE

```
# Tìm học sinh ở Hà Nội
cursor.execute('''
    SELECT * FROM students
    WHERE city = ?
''', ('Hà Nội',))

results = cursor.fetchall()
```

```
for student in results:
    print(student)

# Tìm học sinh tuổi > 20
cursor.execute('''
    SELECT name, age FROM students
    WHERE age > ?
''', (20,))

results = cursor.fetchall()
```

5.4. SELECT Với ORDER BY

```
# Sắp xếp theo tuổi tăng dần
cursor.execute('''
    SELECT * FROM students
    ORDER BY age ASC
''')

# Sắp xếp theo tên giảm dần
cursor.execute('''
    SELECT * FROM students
    ORDER BY name DESC
''')
```

5.5. SELECT Với LIMIT

```
# Lấy 5 học sinh đầu tiên
cursor.execute('''
    SELECT * FROM students
    LIMIT 5
''')

# Lấy 5 học sinh, bỏ qua 10 đầu (pagination)
cursor.execute('''
    SELECT * FROM students
    LIMIT 5 OFFSET 10
''')
```

5.6. Aggregate Functions

```
# Đếm số học sinh
cursor.execute('SELECT COUNT(*) FROM students')
count = cursor.fetchone()[0]
print(f"Total students: {count}")
```

```
# Điểm trung bình
cursor.execute('SELECT AVG(score) FROM scores')
avg = cursor.fetchone()[0]
print(f"Average score: {avg:.2f}")

# Điểm cao nhất
cursor.execute('SELECT MAX(score) FROM scores')
max_score = cursor.fetchone()[0]

# Điểm thấp nhất
cursor.execute('SELECT MIN(score) FROM scores')
min_score = cursor.fetchone()[0]

# Tổng điểm
cursor.execute('SELECT SUM(score) FROM scores')
total = cursor.fetchone()[0]
```

5.7. GROUP BY

```
# Điểm trung bình từng môn
cursor.execute('''
    SELECT subject, AVG(score) as avg_score
    FROM scores
    GROUP BY subject
''')

results = cursor.fetchall()
for subject, avg in results:
    print(f"{subject}: {avg:.2f}")

# Số học sinh mỗi thành phố
cursor.execute('''
    SELECT city, COUNT(*) as count
    FROM students
    GROUP BY city
''')
```

5.8. JOIN - Kết Hợp Bảng

```
# JOIN students và scores
cursor.execute('''
    SELECT students.name, scores.subject, scores.score
    FROM students
    INNER JOIN scores ON students.student_id = scores.student_id
''')

results = cursor.fetchall()
for name, subject, score in results:
    print(f"{name} - {subject}: {score}")
```

```
# Tìm học sinh có điểm Math > 85
cursor.execute('''
    SELECT DISTINCT students.name
    FROM students
    INNER JOIN scores ON students.student_id = scores.student_id
    WHERE scores.subject = 'Math' AND scores.score > 85
''')
```

---

6. UPDATE - Cập Nhật Dữ Liệu

6.1. UPDATE Cơ Bản

```
# Cập nhật tuổi của 1 học sinh
with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    cursor.execute('''
        UPDATE students
        SET age = ?
        WHERE student_id = ?
    ''', (22, 'S001'))

    print(f"Updated {cursor.rowcount} row(s)")
```

6.2. UPDATE Nhiều Columns

```
# Cập nhật nhiều thông tin
cursor.execute('''
    UPDATE students
    SET name = ?, age = ?, city = ?
    WHERE student_id = ?
''', ('Nguyễn Văn An (Updated)', 23, 'Hà Nội', 'S001'))
```

6.3. UPDATE Với Điều Kiện

```
# Tăng điểm tất cả môn Math lên 5 điểm
cursor.execute('''
    UPDATE scores
    SET score = score + 5
    WHERE subject = 'Math' AND score <= 95
''')

print(f"Updated {cursor.rowcount} rows")
```

7. DELETE - Xóa Dữ Liệu

7.1. DELETE Với WHERE

```
# Xóa 1 học sinh
with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    cursor.execute('''
        DELETE FROM students
        WHERE student_id = ?
    ''', ('S005',))

    print(f"Deleted {cursor.rowcount} row(s)")
```

7.2. DELETE Nhiều Rows

```
# Xóa tất cả học sinh tuổi < 18
cursor.execute('''
    DELETE FROM students
    WHERE age < 18
''')

# Xóa tất cả điểm Math
cursor.execute('''
    DELETE FROM scores
    WHERE subject = 'Math'
''')
```

7.3. DELETE ALL (Cẩn Thận!)

```
# ⚠ Xóa TẤT CẢ dữ liệu trong bảng
cursor.execute('DELETE FROM scores')  # Nguy hiểm!

# Hoặc dùng TRUNCATE (nhanh hơn, reset auto-increment)
cursor.execute('DELETE FROM scores')
cursor.execute('DELETE FROM sqlite_sequence WHERE name="scores"')
```

8. Ứng Dụng Cho Assessment

8.1. Bài 3 - SQLite Version

Database Schema

```
import sqlite3

def create_database():
    """Tạo database và tables"""
    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS scores (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                student_id TEXT NOT NULL,
                subject TEXT NOT NULL,
                score INTEGER CHECK(score >= 0 AND score <= 100)
            )
        ''')

        print("Database created successfully!")

create_database()
```

Load CSV Data

```
def load_scores_from_csv(csv_file):
    """Load điểm từ CSV vào database"""
    import csv

    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        with open(csv_file, 'r', encoding='utf-8') as f:
            reader = csv.reader(f)
            next(reader)  # Skip header

            for row in reader:
                student_id, subject, score = row
                cursor.execute('''
                    INSERT INTO scores (student_id, subject, score)
                    VALUES (?, ?, ?)
                ''', (student_id, subject, int(score)))

        print(f"Loaded data from {csv_file}")

# Load data
load_scores_from_csv('scores.csv')
```

Highest Score Subject (Main Function)

```
def highest_score_subject(student_id):
    """
    Tìm môn có điểm cao nhất của học sinh

    Args:
        student_id (str): Mã sinh viên

    Returns:
        str: Tên môn học, hoặc None nếu không tìm thấy
    """
    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        # Query môn có điểm cao nhất
        cursor.execute('''
            SELECT subject, score
            FROM scores
            WHERE student_id = ?
            ORDER BY score DESC
            LIMIT 1
        ''', (student_id,))

        result = cursor.fetchone()

        if result:
            return result[0]  # Trả về subject
        else:
            return None

# Test
print(highest_score_subject('S001'))  # Physics
print(highest_score_subject('S002'))  # Chemistry
print(highest_score_subject('S999'))  # None
```

Additional Queries

```
def get_student_scores(student_id):
    """Lấy tất cả điểm của học sinh"""
    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        cursor.execute('''
            SELECT subject, score
            FROM scores
            WHERE student_id = ?
            ORDER BY subject
        ''', (student_id,))

        return cursor.fetchall()
```

```
def get_average_score(student_id):
    """Tính điểm trung bình"""
    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        cursor.execute('''
            SELECT AVG(score)
            FROM scores
            WHERE student_id = ?
        ''', (student_id,))

        result = cursor.fetchone()
        return result[0] if result[0] else 0

def get_top_students(subject, limit=5):
    """Top học sinh theo môn"""
    with sqlite3.connect('school.db') as conn:
        cursor = conn.cursor()

        cursor.execute('''
            SELECT student_id, score
            FROM scores
            WHERE subject = ?
            ORDER BY score DESC
            LIMIT ?
        ''', (subject, limit))

        return cursor.fetchall()

# Test
scores = get_student_scores('S001')
print(f"S001 scores: {scores}")

avg = get_average_score('S001')
print(f"S001 average: {avg:.2f}")

top = get_top_students('Math', 3)
print(f"Top 3 Math: {top}")
```

---

9. Advanced Topics

9.1. Transactions

```
import sqlite3

def transfer_score():
    """Demo transaction - transfer điểm giữa 2 học sinh"""
    conn = sqlite3.connect('school.db')

    try:
```

```
        cursor = conn.cursor()

        # Giảm điểm S001
        cursor.execute('''
            UPDATE scores
            SET score = score - 10
            WHERE student_id = 'S001' AND subject = 'Math'
        ''')

        # Tăng điểm S002
        cursor.execute('''
            UPDATE scores
            SET score = score + 10
            WHERE student_id = 'S002' AND subject = 'Math'
        ''')

        # Commit nếu cả 2 thành công
        conn.commit()
        print("Transaction successful!")

    except Exception as e:
        # Rollback nếu có lỗi
        conn.rollback()
        print(f"Transaction failed: {e}")

    finally:
        conn.close()
```

9.2. Row Factory - Dict-like Results

```
import sqlite3

# Mặc định: results là tuples
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute('SELECT * FROM students')
print(cursor.fetchone())  # ('S001', 'An', 20, 'Hà Nội')

# ☑ Row factory: results như dictionaries
conn.row_factory = sqlite3.Row
cursor = conn.cursor()
cursor.execute('SELECT * FROM students')
row = cursor.fetchone()

print(row['name'])        # An
print(row['age'])         # 20
print(dict(row))          # {'student_id': 'S001', 'name': 'An', ...}
```

9.3. Indexing - Tăng Tốc Query

```
# Tạo index trên cột thường query
with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    # Index trên student_id
    cursor.execute('''
        CREATE INDEX IF NOT EXISTS idx_student_id
        ON scores(student_id)
    ''')

    # Index trên subject
    cursor.execute('''
        CREATE INDEX IF NOT EXISTS idx_subject
        ON scores(subject)
    ''')

    print("Indexes created!")

# Query sẽ nhanh hơn với index
```

9.4. Views - Virtual Tables

```
# Tạo view cho query phức tạp
with sqlite3.connect('school.db') as conn:
    cursor = conn.cursor()

    cursor.execute('''
        CREATE VIEW IF NOT EXISTS student_avg_scores AS
        SELECT
            student_id,
            AVG(score) as avg_score,
            COUNT(*) as subject_count
        FROM scores
        GROUP BY student_id
    ''')

    # Query view như table bình thường
    cursor.execute('SELECT * FROM student_avg_scores')
    results = cursor.fetchall()

    for student_id, avg, count in results:
        print(f"{student_id}: Avg={avg:.2f}, Subjects={count}")
```

---

10. Bài Tập Thực Hành

Bài 1: Student Management System

```python
class StudentDB:
    """Class quản lý database học sinh"""

    def __init__(self, db_file='students.db'):
        self.db_file = db_file
        self._create_tables()

    def _create_tables(self):
        """Tạo tables"""
        with sqlite3.connect(self.db_file) as conn:
            cursor = conn.cursor()

            cursor.execute('''
                CREATE TABLE IF NOT EXISTS students (
                    student_id TEXT PRIMARY KEY,
                    name TEXT NOT NULL,
                    age INTEGER,
                    email TEXT UNIQUE
                )
            ''')

    def add_student(self, student_id, name, age, email):
        """Thêm học sinh mới"""
        with sqlite3.connect(self.db_file) as conn:
            cursor = conn.cursor()

            try:
                cursor.execute('''
                    INSERT INTO students (student_id, name, age, email)
                    VALUES (?, ?, ?, ?)
                ''', (student_id, name, age, email))

                print(f"Added student: {name}")
                return True

            except sqlite3.IntegrityError as e:
                print(f"Error: {e}")
                return False

    def get_student(self, student_id):
        """Lấy thông tin học sinh"""
        with sqlite3.connect(self.db_file) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()

            cursor.execute('''
                SELECT * FROM students
                WHERE student_id = ?
            ''', (student_id,))

            return cursor.fetchone()

    def update_student(self, student_id, **kwargs):
```

```
        """Cập nhật thông tin"""
        fields = ', '.join([f"{k} = ?" for k in kwargs.keys()])
        values = list(kwargs.values()) + [student_id]

        with sqlite3.connect(self.db_file) as conn:
            cursor = conn.cursor()

            cursor.execute(f'''
                UPDATE students
                SET {fields}
                WHERE student_id = ?
            ''', values)

            print(f"Updated {cursor.rowcount} student(s)")

    def delete_student(self, student_id):
        """Xóa học sinh"""
        with sqlite3.connect(self.db_file) as conn:
            cursor = conn.cursor()

            cursor.execute('''
                DELETE FROM students
                WHERE student_id = ?
            ''', (student_id,))

            print(f"Deleted {cursor.rowcount} student(s)")

    def list_all_students(self):
        """Liệt kê tất cả học sinh"""
        with sqlite3.connect(self.db_file) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()

            cursor.execute('SELECT * FROM students ORDER BY name')
            return cursor.fetchall()

# Test
db = StudentDB()
db.add_student('S001', 'Nguyễn Văn An', 20, 'an@email.com')
db.add_student('S002', 'Trần Thị Bình', 21, 'binh@email.com')

student = db.get_student('S001')
print(dict(student))

db.update_student('S001', age=22, email='an.new@email.com')

students = db.list_all_students()
for s in students:
    print(f"{s['student_id']}: {s['name']} - {s['age']} tuổi")
```

Bài 2: Library Management

```
def create_library_db():
    """Tạo database thư viện"""
    with sqlite3.connect('library.db') as conn:
        cursor = conn.cursor()

        # Bảng books
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS books (
                book_id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                author TEXT NOT NULL,
                isbn TEXT UNIQUE,
                available INTEGER DEFAULT 1
            )
        ''')

        # Bảng members
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS members (
                member_id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                email TEXT UNIQUE NOT NULL,
                join_date TEXT DEFAULT CURRENT_DATE
            )
        ''')

        # Bảng loans
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS loans (
                loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
                book_id INTEGER NOT NULL,
                member_id INTEGER NOT NULL,
                loan_date TEXT DEFAULT CURRENT_DATE,
                return_date TEXT,
                FOREIGN KEY (book_id) REFERENCES books(book_id),
                FOREIGN KEY (member_id) REFERENCES members(member_id)
            )
        ''')

def borrow_book(member_id, book_id):
    """Mượn sách"""
    with sqlite3.connect('library.db') as conn:
        cursor = conn.cursor()

        # Kiểm tra sách có available không
        cursor.execute('''
            SELECT available FROM books WHERE book_id = ?
        ''', (book_id,))

        result = cursor.fetchone()
        if not result or result[0] == 0:
            print("Sách không available!")
```

```
            return False

        # Tạo loan record
        cursor.execute('''
            INSERT INTO loans (book_id, member_id)
            VALUES (?, ?)
        ''', (book_id, member_id))

        # Cập nhật availability
        cursor.execute('''
            UPDATE books
            SET available = 0
            WHERE book_id = ?
        ''', (book_id,))

        print("Mượn sách thành công!")
        return True

# Setup và test
create_library_db()
# ... thêm books, members, test borrow ...
```

Bài 3: Analytics Report

```
def generate_score_report(db_file='school.db'):
    """
    Tạo báo cáo phân tích điểm

    Returns:
        dict: Các thống kê
    """
    with sqlite3.connect(db_file) as conn:
        cursor = conn.cursor()

        report = {}

        # Tổng số học sinh
        cursor.execute('SELECT COUNT(DISTINCT student_id) FROM scores')
        report['total_students'] = cursor.fetchone()[0]

        # Điểm trung bình chung
        cursor.execute('SELECT AVG(score) FROM scores')
        report['overall_avg'] = cursor.fetchone()[0]

        # Điểm TB từng môn
        cursor.execute('''
            SELECT subject, AVG(score) as avg, COUNT(*) as count
            FROM scores
            GROUP BY subject
            ORDER BY avg DESC
```

```
        ''')
        report['subject_stats'] = cursor.fetchall()

        # Top 5 học sinh
        cursor.execute('''
            SELECT student_id, AVG(score) as avg_score
            FROM scores
            GROUP BY student_id
            ORDER BY avg_score DESC
            LIMIT 5
        ''')
        report['top_students'] = cursor.fetchall()

        # Phân bố điểm
        cursor.execute('''
            SELECT
                CASE
                    WHEN score >= 90 THEN 'A'
                    WHEN score >= 80 THEN 'B'
                    WHEN score >= 70 THEN 'C'
                    WHEN score >= 60 THEN 'D'
                    ELSE 'F'
                END as grade,
                COUNT(*) as count
            FROM scores
            GROUP BY grade
            ORDER BY grade
        ''')
        report['grade_distribution'] = cursor.fetchall()

        return report

# Generate report
report = generate_score_report()

print(f"=== SCORE ANALYTICS REPORT ===")
print(f"Total Students: {report['total_students']}")
print(f"Overall Average: {report['overall_avg']:.2f}")

print("\nSubject Statistics:")
for subject, avg, count in report['subject_stats']:
    print(f"  {subject}: {avg:.2f} (n={count})")

print("\nTop 5 Students:")
for student_id, avg in report['top_students']:
    print(f"  {student_id}: {avg:.2f}")

print("\nGrade Distribution:")
for grade, count in report['grade_distribution']:
    print(f"  {grade}: {count}")
```

11. Best Practices

11.1. ☑ DO

```
# ☑ Dùng parameterized queries
cursor.execute('SELECT * FROM students WHERE id = ?', (student_id,))

# ☑ Dùng with statement
with sqlite3.connect('db.db') as conn:
    cursor = conn.cursor()
    # ...

# ☑ Dùng try-except cho integrity errors
try:
    cursor.execute('INSERT INTO students ...')
except sqlite3.IntegrityError:
    print("Duplicate key!")

# ☑ Tạo indexes cho columns thường query
cursor.execute('CREATE INDEX idx_name ON students(name)')

# ☑ Dùng LIMIT cho large result sets
cursor.execute('SELECT * FROM students LIMIT 100')

# ☑ Close connections khi xong
conn.close()
```

11.2. ❌ DON'T

```
# ❌ SQL injection risk
name = user_input
cursor.execute(f"SELECT * FROM students WHERE name = '{name}'")

# ❌ Không commit changes
cursor.execute('INSERT INTO students ...')
# Quên conn.commit()

# ❌ Fetch all cho large tables
cursor.execute('SELECT * FROM huge_table')
all_rows = cursor.fetchall()  # Out of memory!

# ❌ Không xử lý exceptions
cursor.execute('INSERT ...')  # Có thể lỗi IntegrityError

# ❌ Hard-code database filename
conn = sqlite3.connect('mydata.db')  # Nên parameterize
```

---

12. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Connection & Cursor:

- ☐ `sqlite3.connect()` - kết nối database
- ☐ `cursor.execute()` - thực thi SQL
- ☐ `conn.commit()` - lưu changes
- ☐ `conn.close()` - đóng connection
- ☐ `with` statement - auto commit

CRUD Operations:

- ☐ `CREATE TABLE` - tạo bảng
- ☐ `INSERT` - thêm dữ liệu
- ☐ `SELECT` - truy vấn (WHERE, ORDER BY, LIMIT)
- ☐ `UPDATE` - cập nhật
- ☐ `DELETE` - xóa

Advanced:

- ☐ Aggregate functions (COUNT, AVG, MAX, MIN, SUM)
- ☐ `GROUP BY` - nhóm dữ liệu
- ☐ `JOIN` - kết hợp bảng
- ☐ Parameterized queries (? placeholders)
- ☐ Transactions (commit/rollback)

Assessment:

- ☐ Migrate CSV → SQLite
- ☐ Query với WHERE và ORDER BY
- ☐ Aggregate queries

🎯 Lưu ý quan trọng

1. **LUÔN dùng parameterized queries** - tránh SQL injection
2. **LUÔN dùng** `with` hoặc `try-finally` - đảm bảo close
3. **Commit changes** - `conn.commit()` sau INSERT/UPDATE/DELETE
4. **Index thường-query columns** - tăng performance
5. **Xử lý IntegrityError** - duplicate keys, foreign key violations
6. **SQLite thread-safe**: Cẩn thận với concurrent access

---

13. Câu Hỏi Thường Gặp (FAQ)

Q1: SQLite vs CSV - Khi nào dùng cái nào?

A:

Use Case	CSV	SQLite
Dữ liệu đơn giản, flat	☑	

Use Case	CSV	SQLite
Cần query phức tạp		☑
Cần relationships (foreign keys)		☑
Concurrent access		☑
Data integrity constraints		☑
Portable, human-readable	☑	

Q2: Tại sao dùng `?` thay vì f-string?

A: SQL Injection - nguy hiểm!

```
# ❌ NGUY HIỂM
name = "'; DROP TABLE students; --"
cursor.execute(f"DELETE FROM students WHERE name = '{name}'")
# SQL: DELETE FROM students WHERE name = ''; DROP TABLE students; --'
# → XÓA TOÀN BỘ TABLE!

# ☑ AN TOÀN - parameterized query
cursor.execute("DELETE FROM students WHERE name = ?", (name,))
# SQLite tự động escape, không thực thi DROP
```

Q3: `commit()` vs `with` statement?

A:

```
# Cách 1: Manual commit
conn = sqlite3.connect('db.db')
cursor = conn.cursor()
cursor.execute('INSERT ...')
conn.commit()  # Phải commit
conn.close()

# Cách 2: with - auto commit
with sqlite3.connect('db.db') as conn:
    cursor = conn.cursor()
    cursor.execute('INSERT ...')
# Auto commit ở đây

# ⚠ Lưu ý: with KHÔNG auto close connection
# Nhưng thường OK vì connection close khi script end
```

Q4: `fetchone()` vs `fetchall()`?

A:

```
# fetchone() - 1 row, hoặc None
cursor.execute('SELECT * FROM students WHERE id = ?', (1,))
row = cursor.fetchone()
if row:
    print(row)

# fetchall() - list of rows (có thể rỗng)
cursor.execute('SELECT * FROM students')
rows = cursor.fetchall()  # [] nếu không có data

for row in rows:
    print(row)
```

Q5: Làm sao để export SQLite → CSV?

A:

```
import sqlite3
import csv

def export_to_csv(db_file, table_name, csv_file):
    """Export table sang CSV"""
    with sqlite3.connect(db_file) as conn:
        cursor = conn.cursor()

        # Lấy data
        cursor.execute(f'SELECT * FROM {table_name}')
        rows = cursor.fetchall()

        # Lấy column names
        column_names = [desc[0] for desc in cursor.description]

        # Ghi CSV
        with open(csv_file, 'w', newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(column_names)  # Header
            writer.writerows(rows)          # Data

        print(f"Exported {len(rows)} rows to {csv_file}")

# Test
export_to_csv('school.db', 'students', 'students_export.csv')
```
