SQLite3: Повна теорія та практика

1. Що таке SQLite3?

SQLite — це невеликий, швидкий та автономний рушій реляційної бази даних, який не потребує окремого сервера. Він вбудовується безпосередньо в твою програму. Користуючись бібліотекою sqlite3 у Python, ти фактично взаємодієш з локальним файлом, який є повноцінною базою даних.

Переваги:

- **Легкість та автономність:** Не потребує встановлення, налаштування чи адміністрування. Просто імпортуєш бібліотеку, і готово.
- **Файлова система:** Вся база даних зберігається в одному файлі на диску, що робить її дуже портативною.
- **Простота:** Ідеально підходить для невеликих і середніх проєктів, де не потрібен потужний сервер.
- **Відсутність ліцензій:** SQLite є public domain, тому ти можеш використовувати його безкоштовно в будь-яких цілях.

Недоліки:

- **Не для високих навантажень:** Не призначений для великої кількості одночасних запитів на запис (запис блокує весь файл БД).
- **Відсутність багатокористувацьких можливостей:** Не підходить для програм, де багато користувачів одночасно працюють з одними й тими ж даними.

Причини використання:

SQLite — це чудовий вибір для десктопних програм, мобільних застосунків, тестування, кешування даних та невеликих вебсайтів.

2. Процес роботи та підключення

Процес роботи з SQLite у Python завжди починається з підключення до бази даних і отримання об'єктів **з'єднання** та **курсора**.

Синтаксис:

```
import sqlite3

# 1. Створення або підключення до файлу бази даних 'my_database.db'
```

```
connection = sqlite3.connect('my_database.db')

# 2. Створення об'єкта курсора
cursor = connection.cursor()

# Тепер можна виконувати запити через об'єкт 'cursor'
```

Пояснення процесу:

- **sqlite3.connect('my_database.db')**: Ця функція встановлює з'єднання (connection) з базою даних. Якщо файл my_database.db не існує, він буде автоматично створений.
- **connection.cursor()**: Об'єкт connection створює **курсор** (cursor). Це твій основний робочий інструмент. Всі SQL-запити виконуються саме через нього.

3. Створення бази та таблиць

База даних створюється автоматично при першому підключенні. Наступний крок — створення таблиць, які будуть зберігати наші дані.

Синтаксис:

```
# Запит для створення таблиці `students`
sql_create_table = """
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    age INTEGER,
    grade TEXT
);
"""
cursor.execute(sql_create_table)

# Після виконання будь-яких змін в структурі БД, їх потрібно зафіксувати
connection.commit()
```

- **CREATE TABLE IF NOT EXISTS ...**: Це SQL-команда, яка створює таблицю students, якщо вона ще не існує.
- **id INTEGER PRIMARY KEY**: Створює унікальний ідентифікатор для кожного запису. PRIMARY KEY гарантує унікальність.

- **TEXT NOT NULL**: Колонка name буде зберігати текст і не може бути порожньою.

4. Наповнення даними

Дані можна додавати по одному або пакетом. Завжди використовуй параметризовані запити, щоб уникнути **SQL-ін'єкцій**.

Синтаксис:

```
# 1. Вставка одного запису
name = "Alice"
age = 18
grade = "A"
cursor.execute("INSERT INTO students (name, age, grade) VALUES (?, ?, ?)", (name, age, grade))

# 2. Вставка декількох записів за один раз
students_data = [
    ("Bob", 20, "B"),
    ("Charlie", 19, "A"),
    ("Diana", 21, "C")
]
cursor.executemany("INSERT INTO students (name, age, grade) VALUES (?, ?, ?)", students_data)

connection.commit() # Фіксуємо зміни
```

- **?**: Це заповнювач. SQLite автоматично замінить його на значення з кортежу або списку.

5. Запити на отримання даних

Щоб отримати дані з таблиці, використовується запит SELECT, а потім методи курсора fetchone(), fetchmany(), fetchall().

Синтаксис:

```
# Отримати всі записи з таблиці students
cursor.execute("SELECT * FROM students")
all_students = cursor.fetchall()
```

```
print(f"Всі студенти: {all_students}")

# Отримати тільки студентів, чий вік більше 19
cursor.execute("SELECT name, age FROM students WHERE age > ?", (19,))
older_students = cursor.fetchall()
print(f"Студенти старше 19: {older_students}")

# Отримати перший рядок, який відповідає умові
cursor.execute("SELECT * FROM students WHERE name = ?", ("Bob",))
bob_data = cursor.fetchone()
print(f"Дані Боба: {bob_data}")
```

6. Оновлення даних

Для зміни вже існуючих даних використовується команда UPDATE.

Синтаксис:

```
# Оновити оцінку студента Bob'a
new_grade = "A"
student_name = "Bob"
cursor.execute("UPDATE students SET grade = ? WHERE name = ?",
(new_grade, student_name))
connection.commit()
print(f"Оцінка студента {student_name} оновлена на {new_grade}.")
```

- **WHERE name = ?**: Ця частина запиту є критично важливою, оскільки вона вказує, які саме рядки потрібно оновити. Без WHERE оновляться всі рядки в таблиці.

7. Поєднання запиту з оновленням даних

Часто потрібно спочатку отримати дані, а потім, на основі цих даних, оновити інші. Це типова операція в транзакціях.

Приклад: Припустимо, потрібно збільшити вік усіх студентів, чий вік менше 20, на один рік.

Синтаксис:

```
# 1. Вибираємо студентів, чий вік менше 20
cursor.execute("SELECT name, age FROM students WHERE age < 20")
young_students = cursor.fetchall()

# 2. Перебираємо отримані дані та оновлюємо вік
for name, age in young_students:
    new_age = age + 1
    cursor.execute("UPDATE students SET age = ? WHERE name = ?",
(new_age, name))

# 3. Фіксуємо всі оновлення
connection.commit()
print("Вік молодих студентів успішно оновлено.")

# Перевірка
cursor.execute("SELECT name, age FROM students")
print("Оновлені дані:", cursor.fetchall())

# Завершення роботи
connection.close()
```

- **Пояснення:** Ми виконуємо SELECT для ідентифікації потрібних записів, а потім, використовуючи цикл, проходимося по кожному з них і виконуємо окремий UPDATE для оновлення. Весь цей процес відбувається в рамках однієї транзакції до виклику connection.commit().

Практичний проєкт: Система керування замовленнями

Цей міні-проєкт — це консольна програма, яка імітує просту систему керування замовленнями. Вона демонструє, як об'єднати різні SQL-операції для створення функціональної програми.

Мета проєкту

Створити систему, де користувачі можуть:

1. **Зареєструватися** та **авторизуватися**.
2. **Переглядати** доступні товари.
3. **Додавати** нові товари (якщо це робить "адміністратор").
4. **Робити замовлення** на товари, що є на складі.

Структура бази даних

Ми створимо три таблиці, пов'язані між собою:

Таблиця users	Таблиця products	Таблиця orders
id (PRIMARY KEY)	id (PRIMARY KEY)	id (PRIMARY KEY)
username	name	user_id (FOREIGN KEY)
password	price	product_id (FOREIGN KEY)
	stock	quantity
		order_date

Пояснення кожного блоку коду

1. Функція setup_database()

- **Роль:** Це "ініціалізатор" нашої бази даних.

- **Дія:** Створює всі необхідні таблиці (`users`, `products`, `orders`), якщо вони ще не існують. Це запобігає помилкам при повторному запуску програми.
- **Важливо:** Використовує `conn.commit()` для збереження структури таблиць.

```
import sqlite3
import datetime

# Глобальна змінна для ідентифікатора поточного користувача
current_user = None

def get_db_connection():
    """Створює і повертає об'єкт з'єднання з базою даних
'store.db'."""
    return sqlite3.connect('store.db')

def setup_database():
    """Налаштовує базу даних, створюючи необхідні таблиці, якщо вони
не існують."""
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        # Таблиця для користувачів
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT NOT NULL UNIQUE,
                password TEXT NOT NULL
            );
        """)

        # Таблиця для продуктів
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS products (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL UNIQUE,
                price REAL NOT NULL,
                stock INTEGER NOT NULL
```

```
            );
        """)

        # Таблиця для замовлень
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS orders (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                product_id INTEGER NOT NULL,
                quantity INTEGER NOT NULL,
                order_date TEXT NOT NULL,
                FOREIGN KEY (user_id) REFERENCES users(id),
                FOREIGN KEY (product_id) REFERENCES products(id)
            );
        """)
        conn.commit()
        print("База даних успішно налаштована.")
    except sqlite3.OperationalError as e:
        print(f"Помилка налаштування бази даних: {e}")
    finally:
        conn.close()
```

2. Функція `register_user(username, password)`

- **Роль:** Додає нового користувача.
- **Дія:** Використовує `INSERT` з параметрами (`?`) для безпечної вставки даних.
- **Особливість:** Містить блок `try...except sqlite3.IntegrityError`, який обробляє ситуацію, коли користувач із таким `username` вже існує (через `UNIQUE` обмеження в таблиці).

```
def register_user(username, password):
    """Реєструє нового користувача в системі, використовуючи INSERT."""
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        cursor.execute("INSERT INTO users (username, password) VALUES
(?, ?)", (username, password))
```

```
        conn.commit()
        print(f"Користувача '{username}' успішно зареєстровано!")
    except sqlite3.IntegrityError:
        print(f"Помилка: Користувач з іменем '{username}' вже існує.")
    finally:
        conn.close()
```

3. Функція `login_user(username, password)`

- **Роль:** Перевіряє існуючого користувача.
- **Дія:** Використовує `SELECT` з `WHERE` для пошуку користувача за логіном і паролем.
- **Важливо:** Якщо користувача знайдено, функція повертає його `id`, який потім зберігається в глобальній змінній `current_user` для подальшої роботи в системі.

```
def login_user(username, password):
    """Авторизує користувача, перевіряючи логін та пароль за допомогою
SELECT."""
    global current_user
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT id FROM users WHERE username = ? AND
password = ?", (username, password))
    user = cursor.fetchone()
    conn.close()

    if user:
        current_user = user[0]  # Зберігаємо id користувача для
подальшої роботи
        print(f"Вітаємо, {username}!")
        return True
    else:
        print("Неправильний логін або пароль.")
        return False
```

4. Функція `add_product(name, price, stock)`

- **Роль:** Додає новий товар до каталогу.
- **Дія:** Використовує `INSERT` для додавання запису в таблицю `products`.
- **Особливість:** Так само, як і у `register_user`, використовує обробку `IntegrityError`, щоб запобігти дублюванню товарів.

```
def add_product(name, price, stock):
    """Додає новий продукт до таблиці `products`, використовуючи
INSERT."""
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        cursor.execute("INSERT INTO products (name, price, stock)
VALUES (?, ?, ?)", (name, price, stock))
        conn.commit()
        print(f"Продукт '{name}' успішно додано.")
    except sqlite3.IntegrityError:
        print(f"Помилка: Продукт з назвою '{name}' вже існує.")
    finally:
        conn.close()
```

```
# Дані для 10 різних позицій
products_data = [
("Клавіатура Razer BlackWidow", "Клавіатури", 129.99, 50),
("Мишка Logitech MX Master 3", "Мишки", 99.50, 75),
("Монітор Dell U2723QE", "Монітори", 650.00, 20),
("Ноутбук MacBook Air M2", "Ноутбуки", 1199.00, 15),
("Смартфон Samsung Galaxy S23", "Телефони", 899.99, 30),
("Навушники Sony WH-1000XM5", "Аксесуари", 349.00, 40),
("Вебкамера Logitech C920", "Аксесуари", 75.00, 60),
("Планшет Apple iPad Air", "Планшети", 599.00, 25),
("Зовнішній диск Samsung T7", "Аксесуари", 89.99, 80),
("Принтер Epson EcoTank L3250", "Принтери", 250.00, 10)
]
cursor.executemany("INSERT INTO products (name, category, price,
stock) VALUES (?, ?, ?, ?)", products_data)
```

```
connection.commit()
```

5. Функція `view_products()`

- **Роль:** Відображає список усіх доступних товарів.
- **Дія:** Виконує простий `SELECT *` до таблиці `products` і перебирає результат за допомогою циклу, щоб вивести кожен товар на екран.

```
def view_products():
    """Відображає список усіх продуктів, використовуючи SELECT."""
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT id, name, price, stock FROM products")
    products = cursor.fetchall()
    conn.close()

    if products:
        print("\n--- Доступні продукти ---")
        for product in products:
            print(f"ID: {product[0]}, Назва: {product[1]}, Ціна:
${product[2]:.2f}, В наявності: {product[3]} шт.")
        print("--------------------------\n")
    else:
        print("Наразі немає доступних продуктів.")
```

6. Функція `make_order(product_id, quantity)`

- **Роль:** Основна бізнес-логіка замовлення.
- **Дія:**
    1. Виконує `SELECT` для перевірки, чи існує продукт і чи є його достатньо на складі.
    2. Якщо все добре, використовує `UPDATE` для зменшення кількості товару на складі.
    3. Потім виконує `INSERT` для створення нового запису в таблиці `orders`.
    4. Якщо виникає помилка, використовується `conn.rollback()`, щоб скасувати всі зміни, внесені в рамках цієї транзакції (наприклад, щоб повернути кількість товару, якщо запис про замовлення не було

створено).
5. Зберігає всі зміни за допомогою `conn.commit()`.

```
def make_order(product_id, quantity):
    """Створює нове замовлення, оновлюючи наявність товару та
записуючи замовлення."""
    if current_user is None:
        print("Будь ласка, увійдіть у систему, щоб зробити
замовлення.")
        return

    conn = get_db_connection()
    cursor = conn.cursor()

    # 1. Перевірка наявності товару та достатньої кількості за допомогою
SELECT
    cursor.execute("SELECT stock FROM products WHERE id = ?",
(product_id,))
    product_stock = cursor.fetchone()

    if product_stock and product_stock[0] >= quantity:
        try:
            # 2. Оновлення кількості товару на складі за допомогою
UPDATE
            new_stock = product_stock[0] - quantity
            cursor.execute("UPDATE products SET stock = ? WHERE id =
?", (new_stock, product_id))

            # 3. Додавання запису про замовлення за допомогою INSERT
            order_date = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
            cursor.execute("INSERT INTO orders (user_id, product_id,
quantity, order_date) VALUES (?, ?, ?, ?)",
                           (current_user, product_id, quantity,
order_date))

            # 4. Фіксація транзакції
            conn.commit()
```

```
            print("Замовлення успішно створено!")
        except Exception as e:
            conn.rollback() # Відкочуємо зміни у разі помилки
            print(f"Помилка при створенні замовлення: {e}")
        finally:
            conn.close()
    else:
        print("Помилка: Недостатньо товару на складі або невірний ID
продукту.")
        conn.close()
```

7. Функція `main_menu()` **та** `if __name__ == '__main__':`

- **Роль:** Забезпечує користувацький інтерфейс та керує логікою програми.
- **Дія:** Безкінечний цикл `while` показує різні меню залежно від того, чи авторизований користувач (`current_user` не `None`).
- **Важливо:** Блок `if __name__ == '__main__':` запускає програму, викликаючи `setup_database()` один раз на початку, а потім `main_menu()`.

Спробувати реалізувати самостійно