SQLite

SQLite - це вбудована реляційна база даних, яка зберігає дані в локальному файлі без необхідності встановлення окремого серверу баз даних. Вона надає простий, легкий у використанні механізм управління даними, що робить її дуже популярною для невеликих та середніх проектів, а також для навчання та розробки прототипів.

Основні особливості SQLite:

1. **Безпека та надійність**: SQLite має високий рівень надійності та цілісності даних.
2. **Вбудована**: Немає необхідності у встановленні окремого серверу баз даних, оскільки SQLite працює непрямо в програмі.
3. **Крос-платформенність**: Підтримується на більшості операційних систем.
4. **Простота використання**: Легка у використанні та навчанні, особливо для початківців.
5. **Ефективність**: Працює досить швидко, особливо для невеликих та середніх обсягів даних.
6. **Один файл бази даних**: Всі дані зберігаються у одному файлі бази даних, що полегшує резервне копіювання та переміщення даних.
7. **Невеликі вимоги до ресурсів**: Вимагає мало оперативної пам'яті та процесорних ресурсів.

SQLite є відмінним вибором для невеликих проектів, мобільних додатків, прототипування та для випробування різних концепцій баз даних.

Також, важливо відзначити, що SQLite має свої особливості та обмеження, такі як відсутність підтримки деяких операцій, які можуть бути доступні в інших базах даних. Отже, перед використанням SQLite, важливо добре ознайомитися з його можливостями та особливостями.

Процес роботи

Python за замовчуванням має вбудовану підтримку роботи з базами даних SQLite. Для цього використовується встроєна бібліотека sqlite3, яка в Python доступна як

модуль з такою самою назвою.

Щоб підключитися до бази даних, в цій бібліотеці визначена функція connect():

```
sqlite3.connect(database, timeout=5.0, detect_types=0, isolation_level='DEFERRED', check_same_thread=True, factory=sqlite3.Connection, cached_statements=128, uri=False)
```

Ця функція приймає наступні параметри:

- `database`: шлях до файлу бази даних. Якщо база даних розташована в пам'яті, а не на диску, то для відкриття з'єднання використовується ":memory:".
- `timeout`: період часу в секундах, після якого генерується виняток, якщо файл бази даних зайнятий іншим процесом.
- `detect_types`: керує відповідністю типів SQLite типам Python. Значення 0 вимикає відповідність.
- `isolation_level`: встановлює рівень ізоляції з'єднання і визначає процес відкриття неявних транзакцій. Можливі значення: "DEFERRED" (за замовчуванням), "EXCLUSIVE", "IMMEDIATE" або None (неявні транзакції вимкнені).
- `check_same_thread`: якщо True (за замовчуванням), то тільки потік, який створив з'єднання, може його використовувати. Якщо False, з'єднання може використовуватися кількома потоками.
- `factory`: клас фабрики, який застосовується для створення з'єднання. Повинен представляти клас, похідний від Connection. За замовчуванням використовується клас sqlite3.Connection.
- `cached_statements`: кількість SQL-інструкцій, які мають бути кешовані. За замовчуванням - 128.
- `uri`: булеве значення, якщо True, то шлях до бази даних розглядається як URI.

Але зазвичай для локального використання вистачає і такого:

```
sqlite3.connect(databasename,check_same_thread=False)
```

Обов'язковим параметром є шлях до бази даних. Результатом функції є об'єкт з'єднання (об'єкт класу Connection), через який можна взаємодіяти з базою даних.

Наприклад, підключення до бази даних "metanit.db", яка розташована в тій самій папці, що і поточний скрипт (якщо такої бази даних немає, то вона автоматично створюється):

```
import sqlite3

con = sqlite3.connect("db.db")
```

Також вам надано приклад створення таблиці та виконання запитів до бази даних. Якщо у вас є конкретні запити чи питання, будь ласка, повідомте мене.

Додавання даних

Для додавання даних використовується SQL-інструкція INSERT. Для додавання одного рядка використовується метод execute() об'єкта Cursor:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

# Додаємо рядок в таблицю people
cursor.execute("INSERT INTO people (name, age) VALUES ('Tom', 38)")
# Виконуємо транзакцію
con.commit()
```

Тут додається один рядок, де name = "Tom", а age = 38.

Вираз INSERT неявно відкриває транзакцію, для завершення якої необхідно викликати метод commit() поточного об'єкта Connection.

Встановлення параметрів

За допомогою другого параметра в методі execute() можна передати значення для параметрів SQL-запиту:

```
import sqlite3
```

```
con = sqlite3.connect("db.db")
cursor = con.cursor()

# Дані для додавання
bob = ("Bob", 42)
cursor.execute("INSERT INTO people (name, age) VALUES (?, ?)", bob)

con.commit()
```

У цьому випадку додаються значення, представлені кортежем bob. У SQL-запиті використовуються підстановочні знаки ?. Замість цих символів при виконанні запиту будуть вставлені дані з кортежу data. Отже, перший елемент кортежу - рядок "Bob", передається на місце першого знаку ?, другий елемент - число 42 передається на місце другого знака ?.

Якщо подивитися на вміст бази даних, там можна знайти всі додані об'єкти.

Багатозначний вставка

Метод executemany() дозволяє вставити набір рядків:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

# Дані для додавання
people = [("Sam", 28), ("Alice", 33), ("Kate", 25)]
cursor.executemany("INSERT INTO people (name, age) VALUES (?, ?)", people)

con.commit()
```

У метод cursor.executemany() по суті передається те ж саме вираз SQL, але тепер дані визначені у вигляді списку кортежів people. Фактично кожний кортеж в цьому списку представляє окремий рядок - дані окремого користувача, і при виконанні методу для кожного кортежу буде створюватися свій вираз INSERT INTO.

Отримання даних

Для отримання даних використовується SQL-команда SELECT. Після виконання цієї команди курсор отримує дані, які можна отримати за допомогою одного з методів: fetchall() (повертає список з усіма рядками), fetchmany() (повертає вказану кількість рядків) і fetchone() (повертає одну в наборі рядок).

Отримання всіх рядків

Наприклад, отримаємо всі раніше додані дані з таблиці people:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

# Отримуємо всі дані з таблиці people
cursor.execute("SELECT * FROM

 people")
print(cursor.fetchall())
```

При виконанні цієї програми на консоль буде виведений список рядків, де кожен рядок представляє кортеж:

```
[(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28), (4, 'Alice', 33), (5, 'Kate', 25)]
```

Якщо потрібно, можна перебрати список, використовуючи стандартні циклічні конструкції:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

cursor.execute("SELECT * FROM people")
for person in cursor.fetchall():
    print(f"{person[1]} - {person[2]}")
```

Результат роботи програми:

```
Tom - 38
Bob - 42
Sam - 28
Alice - 33
Kate - 25
```

Важливо відзначити, що в прикладі вище непотрібно викликати метод fetchall, можна перебрати курсор в циклі, як звичайний набір:

```
for person in cursor:
    print(f"{person[1]} - {person[2]}")
```

Отримання частини рядків

Отримання частини рядків за допомогою методу fetchmany(), в який передається кількість рядків:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

cursor.execute("SELECT * FROM people")
# Витягуємо перші 3 рядки в отриманому наборі
print(cursor.fetchmany(3))
```

Результат роботи програми:

```
[(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28)]
```

Виконання цього методу витягує наступні раніше не витягнені рядки:

```
print(cursor.fetchmany(3))  # [(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28)]
print(cursor.fetchmany(3))  # [(4, 'Alice', 33), (5, 'Kate', 25)]
```

Отримання одного рядка

Метод fetchone() витягає наступний рядок у вигляді кортежу значень і повертає його. Якщо рядків більше немає, то повертає None:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()
```

```
cursor.execute("SELECT * FROM people")
# Витягуємо один рядок
print(cursor.fetchone())    # (1, 'Tom', 38)
```

Цей метод зручно використовувати, коли нам потрібно витягнути з бази даних лише один об'єкт:

```
import sqlite3

con = sqlite3.connect("db.db")
cursor = con.cursor()

cursor.execute("SELECT name, age FROM people WHERE id=2")
# Розкладаємо кортеж на дві змінні
name, age = cursor.fetchone()
print(f"Ім'я: {name}    Вік: {age}")    # Ім'я: Bob   Вік: 42
```

Тут ми отримуємо з бази даних рядок з id=2, і отриманий результат розкладаємо на дві змінні name і age (оскільки кортеж в Python можна розкласти на окремі значення).