SQL for DS

1. **Встановлення бібліотек:**
   Переконайтеся, що у вас встановлені необхідні бібліотеки. Дві основні бібліотеки -
   `sqlite3` (для баз даних SQLite) зазвичай вже є у стандартному пакеті

Робота з `sqlite3` включає в себе декілька основних етапів, включаючи підключення до бази даних, створення таблиць, вставку, вибірку та оновлення даних. Ось кілька основних методологій для роботи з `sqlite3` та даними:

1. **Підключення до бази даних:**
   Для роботи з базою даних SQLite використовується `sqlite3.connect()`. Цей метод створює об'єкт з'єднання, який потрібно використовувати для взаємодії з базою даних.

```
import sqlite3

conn = sqlite3.connect('mydatabase.db')
```

2. **Створення таблиць:**
   Визначте схему бази даних, а потім використовуйте `CREATE TABLE` для створення таблиць.

```
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS users
                  (id INTEGER PRIMARY KEY, name TEXT, ag
e INTEGER)''')
conn.commit()
```

3. **Вставка даних:**
   Використовуйте `INSERT INTO` для додавання нових записів до таблиці.

```
cursor.execute("INSERT INTO users (name, age) VALUES (?,
?)", ('John', 25))
```

```
conn.commit()
```

4. **Вибірка даних:**
   Використовуйте
   `SELECT` для отримання даних з таблиці.

```
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

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

5. **Оновлення даних:**
   Використовуйте
   `UPDATE` для зміни існуючих записів.

```
cursor.execute("UPDATE users SET age = ? WHERE name =
?", (30, 'John'))
conn.commit()
```

6. **Видалення даних:**
   Використовуйте
   `DELETE` для видалення записів з таблиці.

```
cursor.execute("DELETE FROM users WHERE name = 'John'")
conn.commit()
```

7. **Закриття з'єднання:**
   Після виконання операцій з базою даних необхідно закрити з'єднання.

```
conn.close()
```

Це базові методології для роботи з `sqlite3` в Python. Зверніть увагу, що для безпечного використання слід використовувати параметризовані запити (за допомогою знака питання та кортежа значень), щоб уникнути SQL-ін'єкцій.

Використання pandas з SQLite може полегшити аналіз та маніпулювання даними в Python. Ось кілька прикладів того, як взаємодіяти з базою даних SQLite за допомогою бібліотеки pandas:

1. **Створення DataFrame з результатами SQL-запиту:**
   Можна використовувати функцію `read_sql_query()` для виконання SQL-запитів та автоматичного створення DataFrame на основі результатів.

   ```
   import sqlite3
   import pandas as pd

   conn = sqlite3.connect('mydatabase.db')

   # Виконання SQL-запиту та створення DataFrame
   query = "SELECT * FROM mytable"
   df = pd.read_sql_query(query, conn)

   conn.close()
   ```

2. **Збереження DataFrame в базі даних:**
   Ви можете зберегти існуючий DataFrame в базі даних, використовуючи метод `to_sql()`.

   ```
   import sqlite3
   import pandas as pd

   conn = sqlite3.connect('mydatabase.db')

   # Припустимо, що df - ваш DataFrame
   df.to_sql('new_table', conn, index=False, if_exists='rep
   lace')

   conn.close()
   ```

   Опції:

   - `index=False`: Запобігає збереженню індексів DataFrame в базі даних.

- `if_exists='replace'`: Заміняє існуючу таблицю, якщо вона вже існує. Інші варіанти: 'fail', 'append', 'replace'.

3. **Використання параметрів у SQL-запитах:**
   Параметризовані SQL-запити допомагають уникнути SQL-ін'єкцій та забезпечити безпеку. Можна використовувати параметри в SQL-запитах та передавати їх через кортеж.

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('mydatabase.db')

# Параметризований SQL-запит
param = ('John',)
query = "SELECT * FROM mytable WHERE name = ?"
df = pd.read_sql_query(query, conn, params=param)

conn.close()
```

4. **Маніпулювання та аналіз даних за допомогою pandas:**
   Після завантаження даних у DataFrame можна використовувати всі можливості pandas для аналізу та маніпулювання даними.

```
# Приклад: вивести перші 5 рядків DataFrame
print(df.head())
```

Це лише декілька основних прикладів. pandas надає багато інших функцій для роботи з даними, які можна використовувати разом з SQLite.