Python Datenbank API (DB-API)

Einführung

Die Python DB-API bietet einen standardisierten Weg für Python-Anwendungen, um mit relationalen Datenbanken zu kommunizieren. Fast jede Python-Datenbank-Modulimplementierung folgt dieser API, einschließlich populärer Datenbanken wie SQLite (`sqlite3`), MySQL, PostgreSQL und mehr.

Kernkonzepte der DB-API

Verbindung herstellen

Um mit einer Datenbank zu arbeiten, müssen Sie zunächst eine Verbindung herstellen. Dies geschieht in der Regel durch Aufrufen der `connect()`-Funktion des Datenbankmoduls, die ein Verbindungsobjekt zurückgibt.

```
import sqlite3
conn = sqlite3.connect('meineDatenbank.db')
```

Cursor-Objekte

Ein Cursor-Objekt wird verwendet, um Befehle an die Datenbank zu senden und Daten abzurufen. Sie erstellen ein Cursor-Objekt, indem Sie die `cursor()`-Methode eines Verbindungsobjekts aufrufen.

```
cursor = conn.cursor()
```

SQL-Befehle ausführen

Mit dem Cursor-Objekt können Sie SQL-Befehle ausführen, wie z.B. `SELECT`, `INSERT`, `UPDATE`, und `DELETE`.

```
cursor.execute("SELECT * FROM meineTabelle")
```

Daten abrufen

Nach dem Ausführen eines `SELECT`-Befehls können Sie die Methode `fetchall()`, `fetchone()`, oder `fetchmany(size)` des Cursor-Objekts verwenden, um die abgerufenen Daten zu erhalten.

```
rows = cursor.fetchall()
for row in rows:
    print(row)
```

Transaktionen

Änderungen an der Datenbank werden in Transaktionen gruppiert. Eine Transaktion beginnt mit dem ersten SQL-Befehl nach dem letzten `commit()` oder `rollback()` und endet mit einem `commit()` (speichert die Änderungen) oder `rollback()` (verwirft die Änderungen).

```
conn.commit()
```

Ressourcen freigeben

Es ist wichtig, Cursor und Verbindungen ordnungsgemäß zu schließen, wenn sie nicht mehr benötigt werden.

```
cursor.close()
conn.close()
```

Wichtige DB-API-Elemente

- **Exception-Hierarchie**: Die DB-API definiert eine Reihe von Standard-Exceptions, die es ermöglichen, Fehler zu behandeln, die während der Datenbankinteraktionen auftreten können.
- **Parametrisierte Abfragen**: Um SQL-Injection zu vermeiden, sollten Sie immer parametrisierte Abfragen verwenden, indem Sie Platzhalter in SQL-Befehlen nutzen und die Werte als zusätzliche Argumente übergeben.

Zusammenfassung

Die Python DB-API bietet eine konsistente Schnittstelle für die Interaktion mit relationalen Datenbanken. Durch die Einhaltung dieser API können Python-Anwendungen unabhängig vom verwendeten Datenbanksystem entwickelt werden, was die Entwicklung vereinfacht und die Portabilität des Codes verbessert.

Datenbankanbindungen mit sqlite3

Einführung

Das `sqlite3`-Modul ermöglicht Python-Anwendungen, SQLite-Datenbanken zu erstellen, darauf zuzugreifen und sie zu manipulieren. SQLite unterstützt die grundlegenden SQL-Operationen und ist für eine Vielzahl von Anwendungen geeignet, von einfachen bis zu mittelgroßen Projekten.

Grundlegende Verwendung

Eine Datenbankverbindung herstellen

Um mit einer SQLite-Datenbank zu arbeiten, müssen Sie zunächst eine Verbindung herstellen. Falls die angegebene Datenbank nicht existiert, wird sie automatisch erstellt.

```
import sqlite3

# Verbindung zu einer Datenbank herstellen (oder erstellen, falls nicht
vorhanden)
conn = sqlite3.connect('meineDatenbank.db')
```

Eine Tabelle erstellen

Mit SQLite können Sie Tabellen erstellen, um Ihre Daten zu strukturieren. Verwenden Sie dafür SQL-Befehle zusammen mit `conn.execute()`.

```
# Eine neue Tabelle erstellen
conn.execute('''CREATE TABLE IF NOT EXISTS tiere
             (id INTEGER PRIMARY KEY,
             name TEXT NOT NULL,
             art TEXT NOT NULL);''')
conn.commit()
```

Daten einfügen

Fügen Sie Daten in Ihre Tabelle ein, indem Sie SQL-Insert-Befehle verwenden.

```
# Daten in die Tabelle einfügen
conn.execute("INSERT INTO tiere (name, art) VALUES ('Rex', 'Hund')")
conn.commit()
```

Daten abfragen

Um Daten aus der Tabelle zu lesen, verwenden Sie SQL-Select-Befehle.

```
# Daten aus der Tabelle abfragen
cursor = conn.execute("SELECT id, name, art FROM tiere")
for row in cursor:
    print(f"ID: {row[0]}, Name: {row[1]}, Art: {row[2]}")
```

Ressourcen freigeben

Es ist wichtig, die Datenbankverbindung ordnungsgemäß zu schließen, wenn Sie fertig sind.

```
conn.close()
```

Transaktionen und Sicherheit

- **Transaktionen**: `sqlite3` unterstützt SQL-Transaktionen, die es ermöglichen, mehrere Operationen als eine einzige, atomare Einheit auszuführen.
- **Sicherheit**: Achten Sie auf SQL-Injection, besonders bei der Verwendung von Benutzereingaben in Ihren SQL-Befehlen. Verwenden Sie Parameterisierte Queries, um diesem Risiko vorzubeugen.

Zusammenfassung

Das `sqlite3`-Modul bietet eine einfache und effiziente Möglichkeit, Datenbankoperationen in Python durchzuführen. Es eignet sich hervorragend für Prototyping, kleinere Anwendungen und Situationen, in denen eine leichte Datenbanklösung bevorzugt wird.