Python + SQLite

Übersicht

- Einleitung
- SQLite
  - Konzept
  - Syntax
  - DB Browser
  - Anwendungen
- Python sqlite3
  - Dokumentation
  - Connection & Cursor Objekt
  - Demo (Jupyter)

Einleitung

Ziele:

1. Übersicht SQLite
2. Übersicht sqlite3 API
3. Beispiel Implementation

SQLite – Konzept

«SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.» (SQLite, 2000)

https://sqlite.org/

SQLite – Konzept

- Zero-Configuration, muss nicht installiert oder aufgesetzt werden
- Compact, gesamte Funktionalität in 1 Datei (sqlite3), <1MB Festplattenspeicherbedarf
- Serverless, braucht keinen Server, Zugriff über API
- Userless, braucht keine User oder Priviledges
- Single file, gesamte Datenbank in einer einzelnen Datei

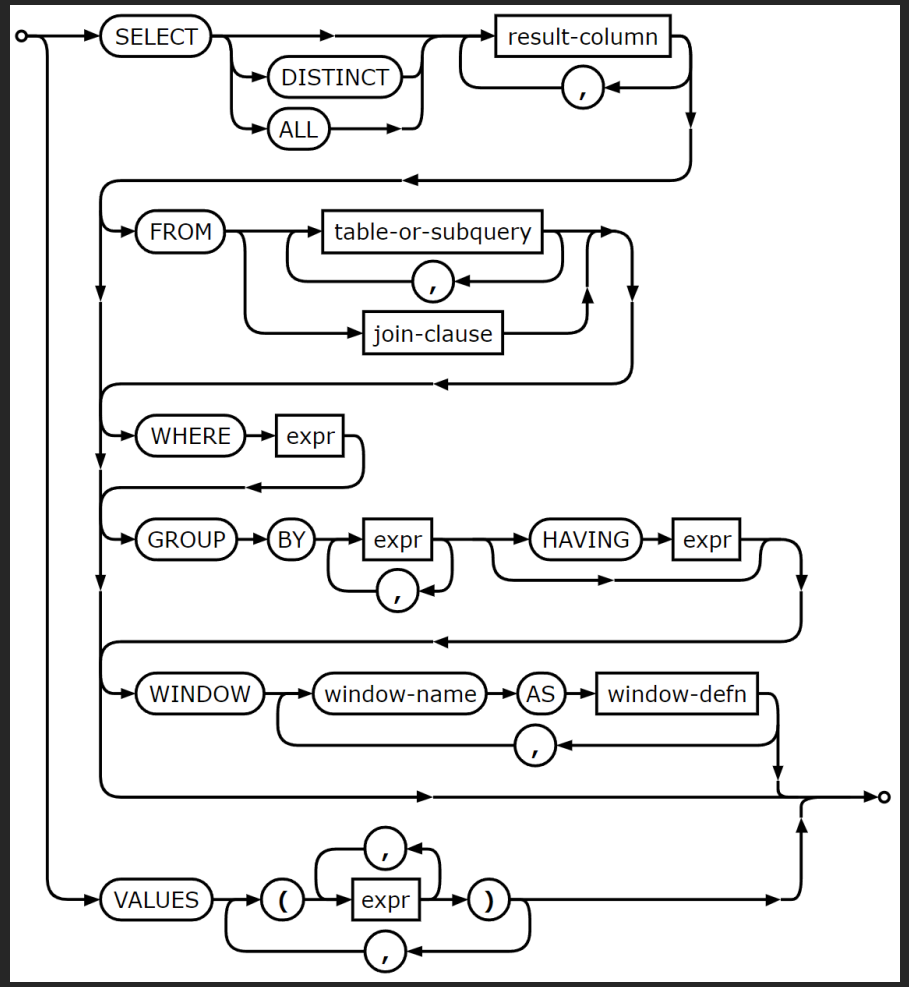
SQLite – Syntax

- Bietet die meisten SQL Befehle
- Kein RIGHT und FULL OUTER JOIN
- Nur RENAME TABLE, ADD/RENAME/DROP Column, kein ADD CONSTRAINT, ALTER COLUMN oder ähnliche Befehle
- Trigger nur für FOR EACH ROW, kein FOR EACH STATEMENT
- Views sind read-only (kein DELETE, INSERT oder UPDATE)
- Keine Permissions (GRANT oder REVOKE)

SQLite – Syntax

- Verfügbare Datentypen
  - (NULL, Fallback wenn keine Daten übermittelt)
  - INTEGER
  - REAL
  - TEXT
  - BLOB
- SQLite kann als «weak type» angesehen werden, bei CREATE TABLE muss kein Datentyp pro Spalte angegeben werden
- Kein BOOLEAN (stattdessen INT 1 oder 0)

SQLite – Syntax



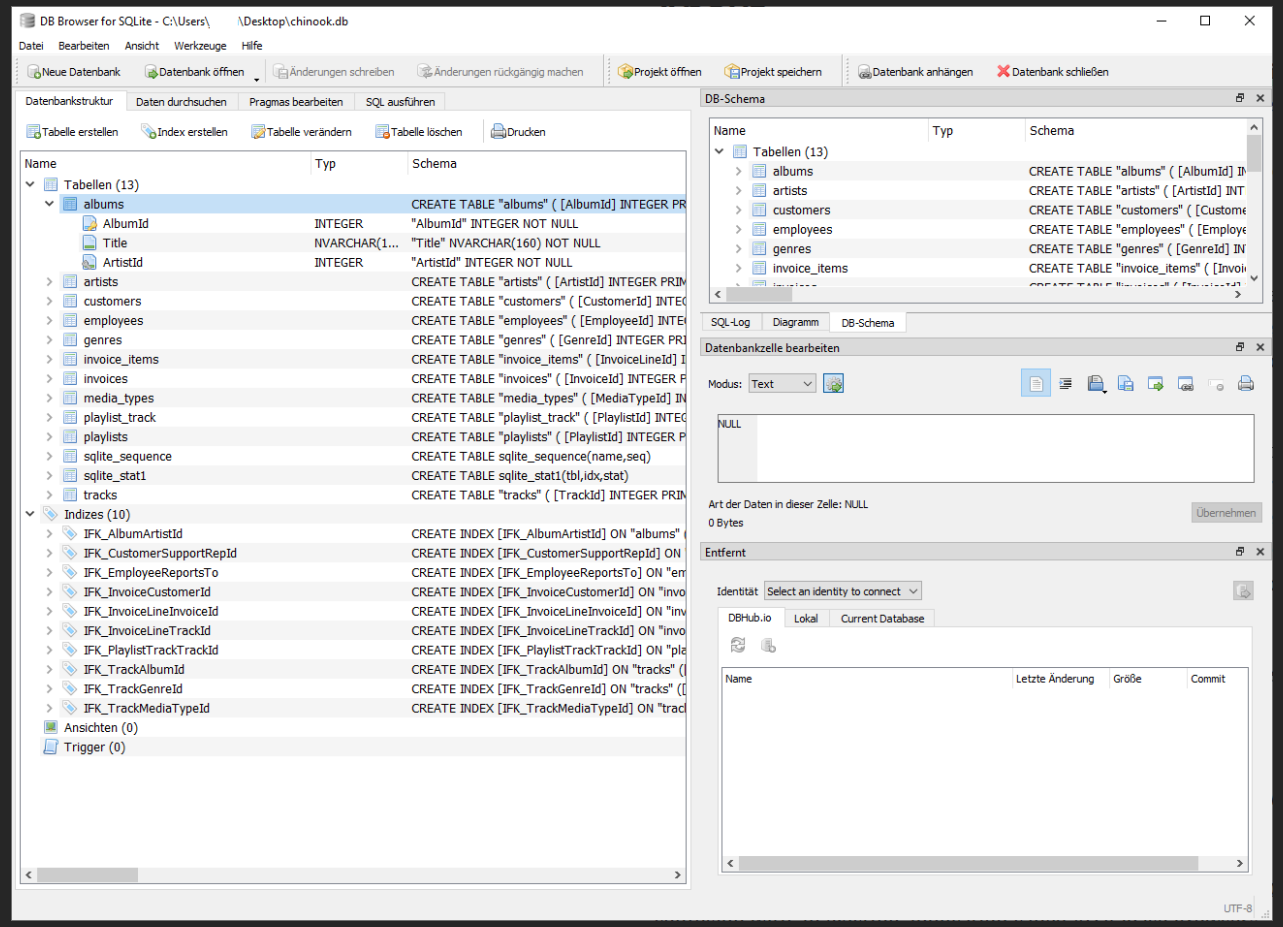
- Syntax Dokumentation als Syntax Diagramme
- GROSS sind Keywords
- klein sind Parameter

SQLite – Syntax

147 gültige Keywords (SQL Spezifikation definiert 916 Keywords)

ABORT	BETWEEN	CURRENT_TIME	ESCAPE	GLOB	INTO	NOTNULL	PRIMARY	ROW	UNIQUE
ACTION	BY	CURRENT_TIMESTAMP	EXCEPT	GROUP	IS	NULL	QUERY	ROWS	UPDATE
ADD	CASCADE	DATABASE	EXCLUDE	GROUPS	ISNULL	NULLS	RAISE	SAVEPOINT	USING
AFTER	CASE	DEFAULT	EXCLUSIVE	HAVING	JOIN	OF	RANGE	SELECT	VACUUM
ALL	CAST	DEFERRABLE	EXISTS	IF	KEY	OFFSET	RECURSIVE	SET	VALUES
ALTER	CHECK	DEFERRED	EXPLAIN	IGNORE	LAST	ON	REFERENCES	TABLE	VIEW
ALWAYS	COLLATE	DELETE	FAIL	IMMEDIATE	LEFT	OR	REGEXP	TEMP	VIRTUAL
ANALYZE	COLUMN	DESC	FILTER	IN	LIKE	ORDER	REINDEX	TEMPORARY	WHEN
AND	COMMIT	DETACH	FIRST	INDEX	LIMIT	OTHERS	RELEASE	THEN	WHERE
AS	CONFLICT	DISTINCT	FOLLOWING	INDEXED	MATCH	OUTER	RENAME	TIES	WINDOW
ASC	CONSTRAINT	DO	FOR	INITIALLY	MATERIALIZED	OVER	REPLACE	TO	WITH
ATTACH	CREATE	DROP	FOREIGN	INNER	NATURAL	PARTITION	RESTRICT	TRANSACTION	WITHOUT
AUTOINCREMENT	CROSS	EACH	FROM	INSERT	NO	PLAN	RETURNING	TRIGGER	
BEFORE	CURRENT	ELSE	FULL	INSTEAD	NOT	PRAGMA	RIGHT	UNBOUNDED	
BEGIN	CURRENT_DATE	END	GENERATED	INTERSECT	NOTHING	PRECEDING	ROLLBACK	UNION	

SQLite – DB Browser



- GUI für einfachen Zugriff auf SQLite Datenbank-Dateien
- Analyse der Datenbankstruktur
- Ausführen von Queries

https://sqlitebrowser.org/

SQLite – Anwendungen

- Google nutzt SQLite in jedem Android-Gerät (vom OS für Apps zur Speicherung grosser Datenmengen angeboten) und in Google Chrome
- Apple nutzt SQLite für iTunes, nahezu alle Mac OS-X Anwendungen und in iOS Geräten zur Datenspeicherung
- Microsoft nutzt SQLite in Windows 10
- **Python bietet seit v2.5 eine Implementation für SQLite**

Python sqlite3

SQLite API in Python

Python sqlite3 - Dokumentation

```
import sqlite3
con = sqlite3.connect('example.db')
cur = con.cursor()
```

- Importieren der gesamten API mittels «`import sqlite3`»
- Aufbauen einer Verbindung mittels «`sqlite3.connect()`», Rückgabe eines «`Connection`» Objekts
- Datenbank wird erstellt oder Verbindung aufgebaut, «`con`» beinhaltet die Verbindung zur DB
- Erhalt eines «`Cursor`» Objekts mittels «`con.cursor()`»
- Ausführen von Queries gegen die Datenbank über «`cur`»

Python sqlite3 - Dokumentation

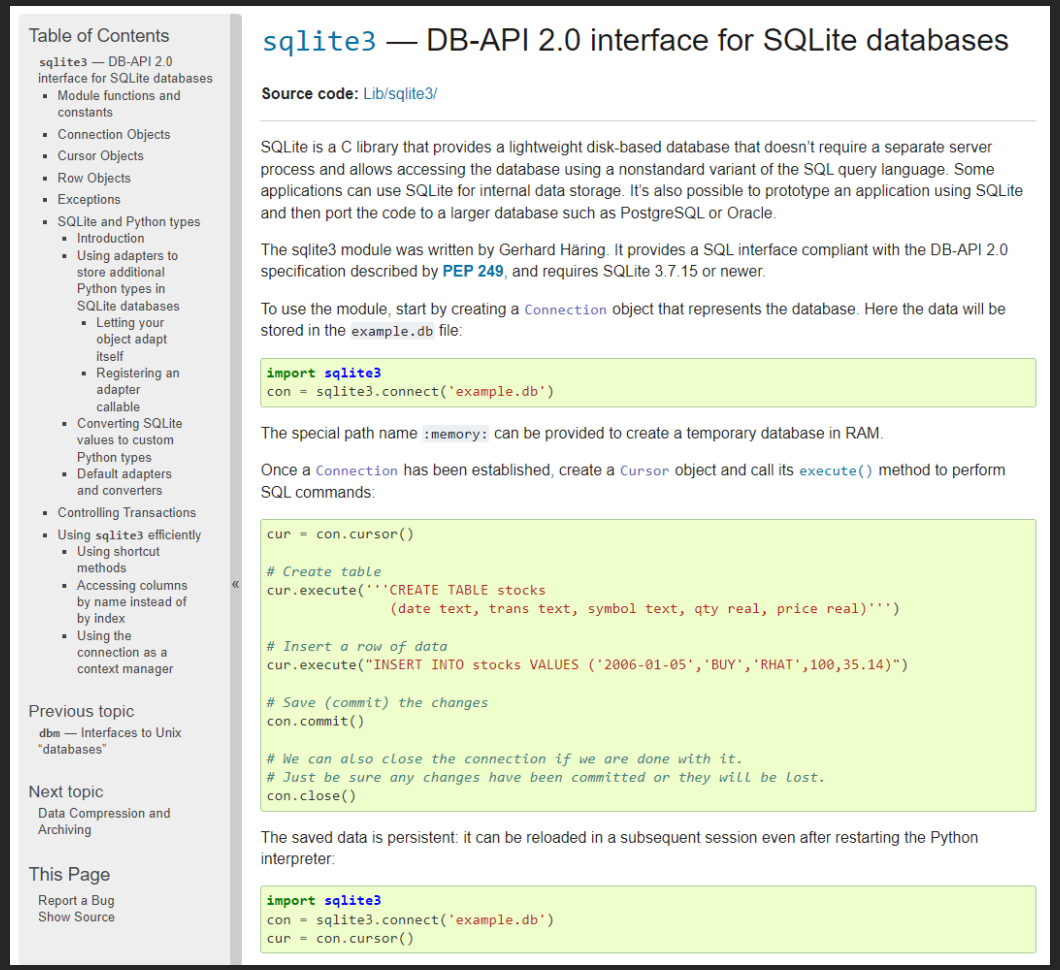


Table of Contents

sqlite3 — DB-API 2.0 interface for SQLite databases

Source code: Lib/sqlite3/

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by **PEP 249**, and requires SQLite 3.7.15 or newer.

To use the module, start by creating a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
con = sqlite3.connect('example.db')
```

The special path name `:memory:` can be provided to create a temporary database in RAM.

Once a `Connection` has been established, create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
cur = con.cursor()

# Create table
cur.execute('''CREATE TABLE stocks
               (date text, trans text, symbol text, qty real, price real)''')

# Insert a row of data
cur.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")

# Save (commit) the changes
con.commit()

# We can also close the connection if we are done with it.
# Just be sure any changes have been committed or they will be lost.
con.close()
```

The saved data is persistent: it can be reloaded in a subsequent session even after restarting the Python interpreter:

```
import sqlite3
con = sqlite3.connect('example.db')
cur = con.cursor()
```

- Gesamte Dokumentation der API unter «https://docs.python.org/3/library/sqlite3.html»

Python sqlite3 – Connection & Cursor

- «`Connection`» Objekt ist Dreh- und Angelpunkt der Datenbankverbindung.
- «`sqlite3.connection(path)`» gibt «`Connection`» Objekt zurück. Werden über eine Verbindung Änderungen an der Datenbank angestossen, so ist die Datenbank für alle anderen Verbindungen gesperrt (read only)
- «`con.cursor()`» gibt «`Cursor`» Objekt zurück für Ausführung von Queries
- «`con.commit()`» wendet Änderungen auf Datenbank an und gibt die Datenbank wieder frei
- «`con.rollback()`» macht alle Änderungen seit dem letzten Commit rückgängig
- «`con.close()`» schliesst die Verbindung
- «`con.backup(target)`» erstellt ein Backup der aktuellen Datenbank

Python sqlite3 – Connection & Cursor

- «Cursor» ist das zentrale Objekt zum Ausführen von Queries gegen die Datenbank
- Erhalt eines «`Cursor`» Objekts über «`con.cursor()`»
- «`cur.execute()`» führt einen einzelnen SQL Befehl aus
- «`cur.executemany()`» führt mehrere, parametrisierte SQL Anweisungen aus
- «`cur.fetchone()`» fragt ein einzelnes Resultat der letzten Query ab
- «`cur.fetchmany(size)`» fragt die angegebene Anzahl an Resultaten der letzten Query ab
- «`cur.fetchall()`» fragt alle Resultate der letzen Query ab
- «`cur.close()`» schliesst das «`Cursor`» Objekt