Python: Relational Database Integration

Using a Local Database with SQLite



Douglas Starnes
Author

linktr.ee/douglasstarnes

Popular Relational Databases

MySQL (PHP)

PostgreSQL (Python)

Microsoft SQL Server (.NET)

SQLite



Runs locally and not on a server



Connect by referencing a path to the database file



Commonly used as a development database for convenience



Mobile applications use it for local storage



Desktop applications use it for settings and configuration

The `sqlite3` Python Package

Connect to and access SQLite databases

Included as part of the Python standard library

Available in most Python distributions

Specialized distributions might not be supported

Using `sqlite3` in 5 Steps

1. Connect to the database
2. Get a cursor from the connection
3. Execute a SQL query with the cursor
4. Retrieve the results
5. Close the cursor and connection

`sqlite3` in Action

```
import sqlite3
```

Import the sqlite3 module



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")
```

Connect to the database



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")

cursor = cn.cursor()
```

Get a cursor



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")

cursor = cn.cursor()

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

Execute a SQL query



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")

cursor = cn.cursor()

cursor.execute("SELECT * FROM events")

results = cursor.fetchall()
```

Retrieve the results.
`fetchall()` **will get all rows**



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")

cursor = cn.cursor()

cursor.execute("SELECT * FROM events")

results = cursor.fetchall()

# use the results
```

Parse the results.
By default, the results are a list of tuples.



sqlite3 in Action

```
import sqlite3

cn = sqlite3.connect("countdown.db")

cursor = cn.cursor()

cursor.execute("SELECT * FROM events")

results = cursor.fetchall()

# use the results

cursor.close()
connection.close()
```

Close the cursor and connection



SQL Operations that Modify Data

The `SELECT` statement (read operation in CRUD) does not modify data

For operations that modify data, call `commit()` after executing the query

Commit the changes before closing the cursor

```
f = open("my_file.txt", "r")
print(f.readlines())
f.close()
```

Resources in a `with` block

Using the `open()` function to open a file, you must explicity call `close()` to close it.



```
with open(“my_file.text”, “r”) as f:
    print(f.readlines())
```

Resources in a `with` block

Opening a file in a `with` block will create a context manager that will automatically close the file when the block completes.



```
import sqlite3

with sqlite3.connect(“countdown.db”) as cn:
    cursor = cn.cursor()
    cursor.execute(“SELECT * FROM events”)
    results = cursor.fetchall()
    # use the results
    cursor.close()
```

Resources in a `with` block

You can do the same thing with a sqlite3 database connection.



```
import sqlite3

with sqlite3.connect(“countdown.db”) as cn:
    cursor = cn.cursor()
    cursor.execute(“SELECT * FROM events”)
    results = cursor.fetchall()
    # use the results
    cursor.close()
```

Resources in a `with` block

You can do the same thing with a sqlite3 database connection.

The connection will automatically be closed with the block completes.



```
import sqlite3

with sqlite3.connect("countdown.db") as cn:
    cursor = cn.cursor()
    cursor.execute("SELECT * FROM events")
    results = cursor.fetchall()
    # use the results
    cursor.close()
```

Resources in a `with` block

You can do the same thing with a sqlite3 database connection.

The connection will automatically be closed with the block completes.

You must still explicitly call the `close()` method on the cursor.



Let's Build a Countdown Application!

Manage a collection of events

Events have a name, date, priority and are private or public

Supports the CRUD operations of a relational DB

Typer handles the CLI features

Rich provides formatting

There's a Pattern Developing

Create a connection

Get a cursor

Execute a query

Fetch the results

Close the cursor (and maybe the connection)

Row Factories



By default, `sqlite3` returns results as a list of tuples



No attempt is made to comply with any kind of structure



The row factory associates results with a custom structure



Set the `row_factory` attribute of the database connection



The `sqlite3` module provides the generic `Row` class

```
import sqlite3

with sqlite3.connect("countdown.db") as cn:
    cn.row_factory = sqlite3.Row
    cursor = cn.cursor()
    cursor.execute("SELECT * FROM events")
    results = cursor.fetchall()
    # use the results
    cursor.close()
```

Using a Row Factory

The `fetchall()` method now returns a list of `Row` objects



```
import sqlite3

with sqlite3.connect("countdown.db") as cn:
    cn.row_factory = sqlite3.Row
    cursor = cn.cursor()
    cursor.execute("SELECT * FROM events")
    results = cursor.fetchall()
    # use the results
    cursor.close()

    print(results[0]["event_name"])
```

Using a Row Factory

The `fetchall()` method now returns a list of `Row` objects

The `Row` is a dictionary-like structure



```python
from dataclasses import dataclass

@dataclass
class Event:
    id: int
    event_name: str
    event_date: str
    priority: int
    is_private: bool
```

From Rows to Python Objects



```python
from dataclasses import dataclass

@dataclass
class Event:
    id: int
    event_name: str
    event_date: str
    priority: int
    is_private: bool

event = Event(**results[0])
print(event.event_name)
```

From Rows to Python Objects

The Row can be used to initialize a dataclass



```python
from dataclasses import dataclass

@dataclass
class Event:
    id: int
    event_name: str
    event_date: str
    priority: int
    is_private: bool

event = Event(**results[0])
print(event.event_name)
events = [Event(**result) for result in results]
```

From Rows to Python Objects

The Row can be used to initialize a dataclass

A basic list comprehension can transform all of the results



```python
def event_factory(cursor, row):
    return Event(
        id=int(row[0]),
        event_name=row[1],
        event_date=row[2],
        priority=int(row[3]),
        is_private=bool(row[4]),
    )
```

Custom Row Factory

Define a function that accepts a cursor and a row, as a tuple.



```python
def event_factory(cursor, row):
    return Event(
        id=int(row[0]),
        event_name=row[1],
        event_date=row[2],
        priority=int(row[3]),
        is_private=bool(row[4]),
    )
```

Custom Row Factory

Define a function that accepts a cursor and a row, as a tuple.

In the function, use the row to initialize a new dataclass



```
def event_factory(cursor, row):
    return Event(
        id=int(row[0]),
        event_name=row[1],
        event_date=row[2],
        priority=int(row[3]),
        is_private=bool(row[4]),
    )

with sqlite3.connect("countdown.db") as cn:
    cn.row_factory = event_factory
```

Custom Row Factory

Define a function that accepts a cursor and a row, as a tuple.

In the function, use the row to initialize a new dataclass

Set the `row_factory` to the function

