Руководство по SQLite в Python

Обновлено: 13.05.2021

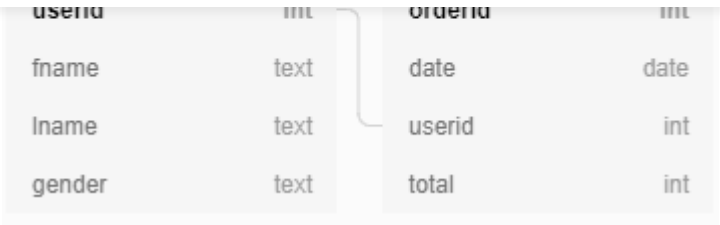
SQL и Python — обязательные инструменты для любого специалиста в сфере анализа данных. Это руководство — все, что вам нужно для первоначальной настройки и освоения основ работы с SQLite в Python. Оно включает следующие пункты:

- Загрузка библиотеки
- Создание и соединение с базой данных
- Создание таблиц базы данных
- Добавление данных
- Запросы на получение данных
- Удаление данных
- И многое другое!

`SQLite3` (часто говорят просто SQLite) — это часть стандартного пакета Python 3, поэтому ничего дополнительно устанавливать не придется.

Что будем создавать

В процессе этого руководства создадим базу данных в SQLite с помощью Python, несколько таблиц и настроим отношения:



Типы данных SQLite в Python

SQLite для **Python предлагает меньше типов данных**, чем есть в других реализациях SQL. С одной стороны, это накладывает ограничения, но, с другой стороны, в SQLite многое сделано проще. Вот основные типы:

- `NULL` — значение `NULL`
- `INTEGER` — целое число
- `REAL` — число с плавающей точкой
- `TEXT` — текст
- `BLOB` — бинарное представление крупных объектов, хранящееся в точности с тем, как его ввели

К сожалению, других привычных для SQL типов данных в SQLite нет.

Первые шаги с SQLite в Python

Начнем руководство с загрузки библиотеки. Для этого нужно использовать следующую команду:

```
import sqlite3
```

Следующий шаг — создание базы данных.

Создание базы данных SQLite в Python

Есть несколько способов создания базы данных в Python с помощью SQLite. Для этого используется объект `Connection`, который и представляет собой базу. Он создается с помощью функции `connect()`.

Создадим файл `.db`, поскольку это стандартный способ управления базой SQLite. Файл будет называться `orders.db`. За соединение будет отвечать переменная `conn`.

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

Эта строка создает объект `connection`, а также новый файл `orders.db` в рабочей директории. Если нужно использовать другую, ее нужно обозначить явно:

```
conn = sqlite3.connect(r'ПУТЬ-К-ПАПКИ/orders.db')
```

Если файл уже существует, то функция `connect` осуществит подключение к нему.

> перед строкой с путем стоит символ «r». Это дает понять Python, что речь идет о «сырой» строке, где символы «/» не отвечают за экранирование.

Функция `connect` создает соединение с базой данных SQLite и возвращает объект, представляющий ее.

Еще один способ создания баз данных с помощью SQLite в Python — создание их в памяти. Это отличный вариант для тестирования, ведь такие базы существуют только в оперативной памяти.

```
conn = sqlite3.connect(:memory:)
```

Однако в большинстве случаев (и в этом руководстве) будет использоваться описанный до этого способ.

Создание объекта cursor

После создания объекта соединения с базой данных нужно создать объект `cursor` . Он позволяет делать SQL-запросы к базе. Используем переменную `cur` для хранения объекта:

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

Теперь выполнять запросы можно следующим образом:

```
cur.execute("ВАШ-SQL-ЗАПРОС-ЗДЕСЬ;")
```

Обратите внимание на то, что сами запросы должны быть помещены в кавычки — это важно. Это могут быть одинарные, двойные или тройные кавычки. Последние используются в случае особенно длинных запросов, которые часто пишутся на нескольких строках.

Создание таблиц в SQLite в Python

Пришло время создать первую таблицу в базе данных. С объектами соединения (`conn`) и `cursor` (`cur`) это можно сделать. Будем следовать этой схеме.

Начнем с таблицы `users` .

```
cur.execute("""CREATE TABLE IF NOT EXISTS users(
   userid INT PRIMARY KEY,
   fname TEXT,
   lname TEXT,
   gender TEXT);
""")
conn.commit()
```

В коде выше выполняются следующие операции:

1. Функция `execute` отвечает за SQL-запрос
2. SQL генерирует таблицу `users`

ничего ли не поменялось.
4. Создаем первые четыре колонки: `userid`, `fname`, `lname` и `gender`. `Userid` — это основной ключ.
5. Сохраняем изменения с помощью функции `commit` для объекта соединения.

Для создания второй таблицы просто повторим последовательность действий, используя следующие команды:

```
cur.execute("""CREATE TABLE IF NOT EXISTS orders(
   orderid INT PRIMARY KEY,
   date TEXT,
   userid TEXT,
   total TEXT);
""")
conn.commit()
```

После исполнения этих двух скриптов база данных будет включать две таблицы. Теперь можно добавлять данные.

Добавление данных с SQLite в Python

По аналогии с запросом для создания таблиц для добавления данных также нужно использовать объект `cursor`.

```
cur.execute("""INSERT INTO users(userid, fname, lname, gender)
   VALUES('00001', 'Alex', 'Smith', 'male');""")
conn.commit()
```

В Python часто приходится иметь дело с переменными, в которых хранятся значения. Например, это может быть кортеж с информацией о пользователе.

```
user = ('00002', 'Lois', 'Lane', 'Female')
```

Если его нужно **загрузить в базу данных**, тогда подойдет следующий формат:

В данном случае все значения заменены на знаки вопроса и добавлен параметр, содержащий значения, которые нужно добавить.

Важно заметить, что SQLite ожидает получить значения в формате кортежа. Однако в переменной может быть и список с набором кортежей. Таким образом можно добавить несколько пользователей:

```
more_users = [('00003', 'Peter', 'Parker', 'Male'), ('00004', 'Bruc
```

Но нужно использовать функцию `executemany` вместо обычной `execute`:

```
cur.executemany("INSERT INTO users VALUES(?, ?, ?, ?);", more_users
conn.commit()
```

Если применить `execute`, то функция подумает, то пользователь хочет передать в таблицу два объекта (два кортежа), а не два кортежа, каждый из которых содержит по 4 значения для каждого пользователя. Хотя в первую очередь вообще должна была возникнуть ошибка.

SQLite и предотвращение SQL-инъекций

Использование способа с вопросительными знаками (?, ?, …) также помогает противостоять SQL-инъекциям. Поэтому рекомендуется использовать его, а не упомянутый до этого.

Скрипты для загрузки данных

Следующие скрипты можно скопировать и вставить для добавления данных в обе таблицы:

```
customers = [
  ('00005', 'Stephanie', 'Stewart', 'female'), ('00006', 'Sincere',
  ('00008', 'Litzy', 'Yates', 'female'), ('00009', 'Jaxon', 'Mills'
  ('00011', 'Kamari', 'Holden', 'female'), ('00012', 'Gaige', 'Summ
  ('00014', 'Angelica', 'Barnes', 'female'), ('00015', 'Leah', 'Pit
  ('00017', 'Joe', 'Walsh', 'male'), ('00018', 'Reagan', 'Cooper',
  ('00020', 'Avery', 'Floyd', 'male'), ('00021', 'Elianna', 'Simmon
  ('00023', 'Elaine', 'Mcintosh', 'female'), ('00024', 'Myla', 'Mck
  ('00026', 'Rohan', 'Peterson', 'male'), ('00027', 'Irene', 'Walte
  ('00029', 'Perla', 'Jefferson', 'female'), ('00030', 'Ashley', 'K
]
orders = [
  ('00001', '2020-01-01', '00025', '178'), ('00002', '2020-01-03',
  ('00004', '2020-01-10', '00015', '110'), ('00005', '2020-01-11',
  ('00007', '2020-01-14', '00028', '227'), ('00008', '2020-01-18',
  ('00010', '2020-01-26', '00017', '116'), ('00011', '2020-01-28',
  ('00013', '2020-02-02', '00015', '177'), ('00014', '2020-02-05',
  ('00016', '2020-02-12', '00008', '180'), ('00017', '2020-02-14',
  ('00019', '2020-02-22', '00002', '168'), ('00020', '2020-02-26',
  ('00022', '2020-03-02', '00019', '211'), ('00023', '2020-03-05',
  ('00025', '2020-03-10', '00006', '45'), ('00026', '2020-03-11', '
  ('00028', '2020-03-17', '00030', '189'), ('00029', '2020-03-20',
  ('00031', '2020-03-23', '00012', '135'), ('00032', '2020-03-24',
  ('00034', '2020-03-28', '00007', '173'), ('00035', '2020-03-30',
  ('00037', '2020-04-03', '00009', '95'), ('00038', '2020-04-06', '
  ('00040', '2020-04-12', '00019', '118'), ('00041', '2020-04-15',
  ('00043', '2020-04-21', '00003', '50'), ('00044', '2020-04-25', '
```

```
    ('00052', '2020-05-10', '00022', '65'), ('00053', '2020-05-14', '
    ('00055', '2020-05-21', '00008', '163'), ('00056', '2020-05-24',
    ('00058', '2020-05-30', '00022', '130'), ('00059', '2020-05-31',
    ('00061', '2020-06-03', '00017', '206'), ('00062', '2020-06-04',
    ('00064', '2020-06-09', '00025', '170'), ('00065', '2020-06-11',
    ('00067', '2020-06-14', '00015', '30'), ('00068', '2020-06-16', '
    ('00070', '2020-06-22', '00005', '184'), ('00071', '2020-06-23',
    ('00073', '2020-06-30', '00022', '149'), ('00074', '2020-07-04',
    ('00076', '2020-07-09', '00007', '156'), ('00077', '2020-07-13',
    ('00079', '2020-07-20', '00019', '81'), ('00080', '2020-07-22', '
    ('00082', '2020-07-27', '00014', '65'), ('00083', '2020-07-30', '
    ('00085', '2020-08-05', '00006', '236'), ('00086', '2020-08-06',
    ('00088', '2020-08-08', '00004', '157'), ('00089', '2020-08-11',
    ('00091', '2020-08-16', '00014', '249'), ('00092', '2020-08-18',
    ('00094', '2020-08-20', '00025', '179'), ('00095', '2020-08-22',
    ('00097', '2020-08-28', '00004', '206'), ('00098', '2020-08-30',
    ('00100', '2020-09-02', '00022', '226')
]
```

Используйте следующие запросы:

```
cur.executemany("INSERT INTO users VALUES(?, ?, ?, ?);", customers)
cur.executemany("INSERT INTO orders VALUES(?, ?, ?, ?);", orders)
conn.commit()
```

Получение данных с SQLite в Python

Следующий момент касательно SQLite в Python — выбор данных. Структура формирования запроса та же, но к ней будет добавлен еще один элемент.

Использование fetchone() в SQLite в Python

```
cur.execute("SELECT * FROM users;")
one_result = cur.fetchone()
print(one_result)
```

Она вернет следующее:

```
[(1, 'Alex', 'Smith', 'male')]
```

Использование fetchmany() в SQLite в Python

Если же нужно получить много данных, то используется функция `fetchmany()` . Выполним другой скрипт для генерации 3 результатов:

```
cur.execute("SELECT * FROM users;")
three_results = cur.fetchmany(3)
print(three_results)
```

Он вернет следующее:

```
[(1, 'Alex', 'Smith', 'male'), (2, 'Lois', 'Lane', 'Female'), (3, '
```

Использование fetchall() в SQLite в Python

Функцию `fetchall()` можно использовать для получения всех результатов. Вот что будет, если запустить скрипт:

```
cur.execute("SELECT * FROM users;")
all_results = cur.fetchall()
print(all_results)
```

Теперь рассмотрим **процесс удаления данных** с SQLite в Python. Здесь та же структура. Предположим, нужно удалить любого пользователя с фамилией «Parker». Напишем следующее:

```
cur.execute("DELETE FROM users WHERE lname='Parker';")
conn.commit()
```

Если затем сделать следующей запрос:

```
cur.execute("select * from users where lname='Parker'")
print(cur.fetchall())
```

Будет выведен пустой список, подтверждающий, что запись удалена.

Объединение таблиц в SQLite в Python

Наконец, посмотрим, как использовать объединение данных для более сложных запросов. Предположим, нужно сгенерировать запрос, включающий имя и фамилию каждого покупателя заказа.

Для этого напишем следующее:

```
cur.execute("""SELECT *, users.fname, users.lname FROM orders
    LEFT JOIN users ON users.userid=orders.userid;""")
print(cur.fetchall())
```

Тот же подход работает с другими SQL-операциями.

Выводы

В этом материале вы узнали все, что требуется для работы с SQLite в Python: загрузка библиотеки, создание баз и таблиц, добавление, запрос и удаление данных.

Появились вопросы? Задайте на Яндекс.Кью

У сайта есть сообщество на Кью >> **Python Q** <<. Там я, эксперты и участники отвечаем на вопросы по python и программированию.

материалы, документации и уроки на русском. На сайте опубликовано множество статей по основам python и библиотекам, уроков для начинающих и примеров написания программ.
Мои контакты: Почта Телеграм Кью

Статьи по теме

Как запускать внешние процессы, используя **Python** и модуль **subprocess**

Ошибка **NameError** в **Python**

Руководство по использованию **list comprehension**

Как вызвать функцию в **Python?**

Удаление элемента из списка в **Python (clear, pop, remove, del)**

Функция **abs()** для получения модуля числа