Python по умолчанию поддерживает работу с базой данных SQLite. Для этого применяется встроенная библиотека **sqlite3**, которая в python доступна в виде одноименного модуля.

Для подключения к бд в этой библиотеке определена функция **connect()**:

```
sqlite3.connect(database, timeout=5.0, detect_types=0, isolation_level='DEFERRED',
check_same_thread=True, factory=sqlite3.Connection, cached_statements=128, uri=False)
```

Она принимает следующие параметры:

- database: путь к файлу базы данных. Если база данных расположена в памяти, а не на диске, то для открытия подключения используется ":memory:"
- timeout: период времени в секундах, через который генерируется исключение, если файл бд занят другим процессом
- detect_types: управляет сопоставлением типов SQLite с типами Python. Значение 0 отключает сопоставление
- isolation_level : устанавливает уровень изоляции подключения и определяет процесс отрытия неявных транзакций. Возможные значения: "DEFERRED" (значение по умолчанию), "EXCLUSIVE", "IMMEDIATE" или None (неявные транзакции отключены)
- check_same_thread: если равно True (значение по умолчанию), то только поток, который создал подключение, может его использовать. Если равно False, подключение может использоваться несколькими потоками.
- factory: класс фабрики, который применяется для создания подключения. Должен представлять класс, производный от Connection. По умолчанию используется класс sqlite3.Connection
- cached_statements: количество SQL-инструкций, которые должны кэшироваться. По умолчанию равно 128.
- uri: булевое значение, если равно True, то путь к базе данных рассматривается как адрес URI

Обязательным параметром функции является путь к базе данных. Результатом функции является объект подключения (объект класса Connection), через затем можно взаимодействовать с базой данных.

Например, подключение к базе данных "metanit.db", которая располагается в той же папке, что и текущий скрипт (если такая база данных отсутствует, то она автоматически создается):

```
import sqlite3;

con = sqlite3.connect("metanit.db")
```

Сопоставление типов SQLite и Python

Прежде чем начать работать с базой данных, следует понимать, как сопоставляются типы SQLite и типы Python. По умолчанию применяются следующие сопоставления:

Python	SQLite
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Следует отметить, что при необходимости мы можем переопределять сопоставление, применяя кастомные конвертеры типов.

Получение курсора

Для выполнения выражений SQL и получения данных из БД, необходимо создать курсор. Для этого у объекта Connection вызывается метод cursor(). Этот метод возвращает объект **Cursor**:

```
import sqlite3;
# создаем подключение
con = sqlite3.connect("metanit.db")
# получаем курсор
cursor = con.cursor()
```

Выполнение запросов к базе данных

Для выполнения запросов и получения данных класс Cursor предоставляет ряд методов:

- execute(sql, parameters=(), /): выполняет одну SQL-инструкцию. Через второй параметр в код SQL можно передать набор параметров в виде списка или словаря
- executemany(sql, parameters, /): выполняет параметризованное SQL-инструкцию. Через второй параметр принимает наборы значений, которые передаются в выполняемый код SQL.

- executescript(sql_script, /): выполняет SQL-скрипт, который может включать множество SQL-инструкций
- fetchone(): возвращает одну строку в виде кортежа из полученного из БД набора строк
- fetchmany(size=cursor.arraysize): возвращает набор строк в виде списка. количество возвращаемых строк передается через параметр. Если больше строк нет в наборе, то возвращается пустой список.
- fetchall(): возвращает все (оставшиеся) строки в виде списка. При отсутствии строк возвращается пустой список.

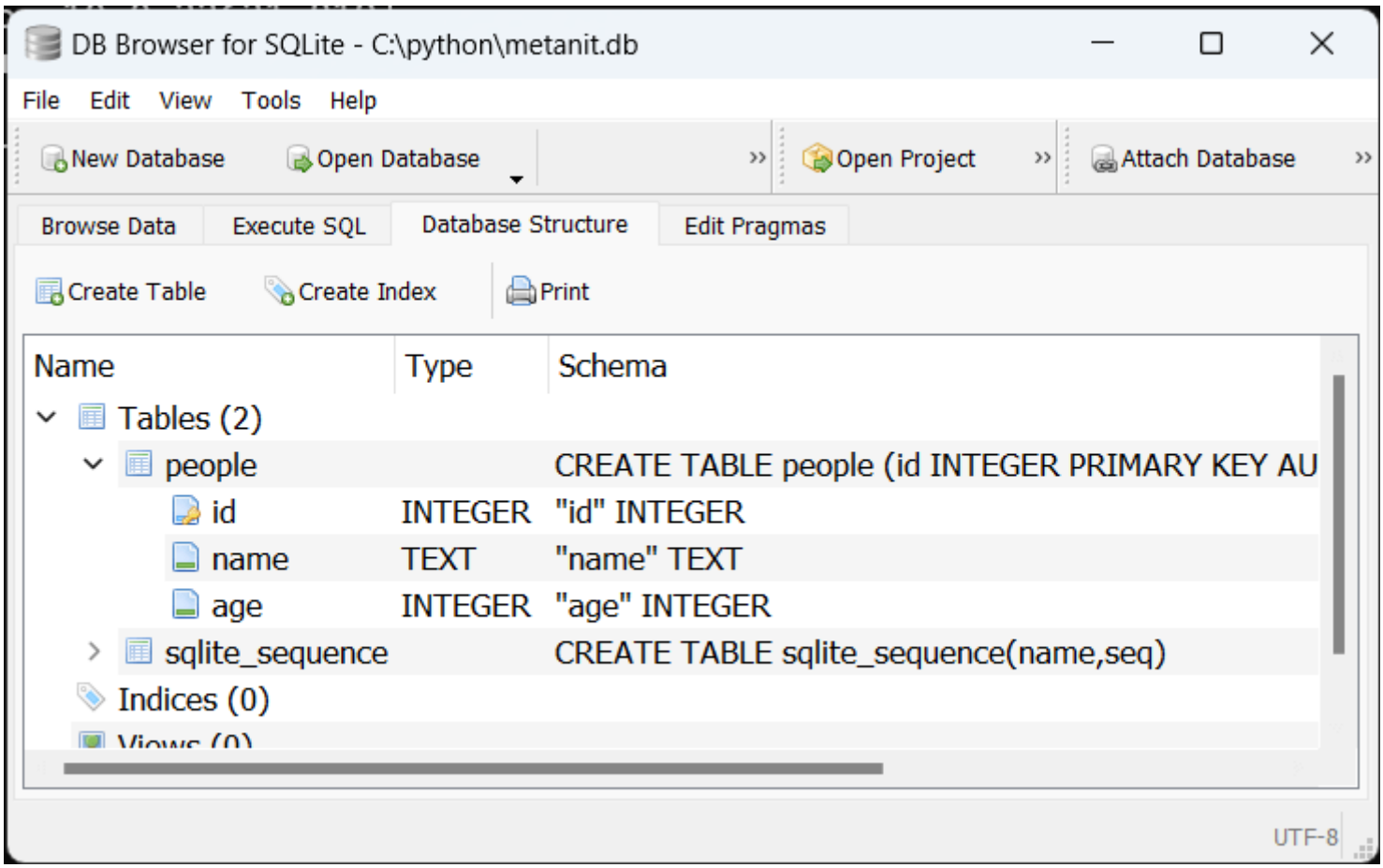
Создание таблицы

Для создания таблицы в SQLite применяется инструкция CREATE TABLE. Например, создадим в базе данных "metanit.db" таблицу people:

```
import sqlite3;

con = sqlite3.connect("metanit.db")
cursor = con.cursor()

# создаем таблицу people
cursor.execute("""CREATE TABLE people
                  (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  name TEXT,
                  age INTEGER)
               """)
```

В метод cursor.execute() передается инструкция CREATE TABLE, которая создает таблицу people с тремя столбцами. Столбец id представляет идентификатор пользователя, хранит данные типа Integer, то есть число, и также представляет первичный ключ, значение которого будет автоматически генерироваться и инкрементироваться с каждой новой строкой. Второй столбец - name представляет строку - имя пользователя. И третий столбец - age представляет возраст пользователя.

После выполнения скрипта мы можем открыть базу данных в каком-нибудь браузере баз данных SQLite, например, в DB Browser for SQLite и увидеть созданную таблицу



Основные операции с данными в SQLite

Рассмотрим основные операции с базой данных SQLite с помощью библиотеки sqlite3 на примере таблицы:

```
CREATE TABLE people (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    age INTEGER
)
```

Добавление данных

Для добавления данных применяется SQL-инструкция **INSERT**. Для добавления одной строки используем метод execute() объекта Cursor:

```
import sqlite3;


con = sqlite3.connect("metanit.db")
cursor = con.cursor()
```

```
# добавляем строку в таблицу people
cursor.execute("INSERT INTO people (name, age) VALUES ('Tom', 38)")
# выполняем транзакцию
con.commit()
```

Здесь добавляется одна строка, где name = "Tom", a age = 38.

Выражение INSERT неявно открывает транзакцию, для завершения которой необходимо вызвать метод **commit()** текущего объекта Connection.

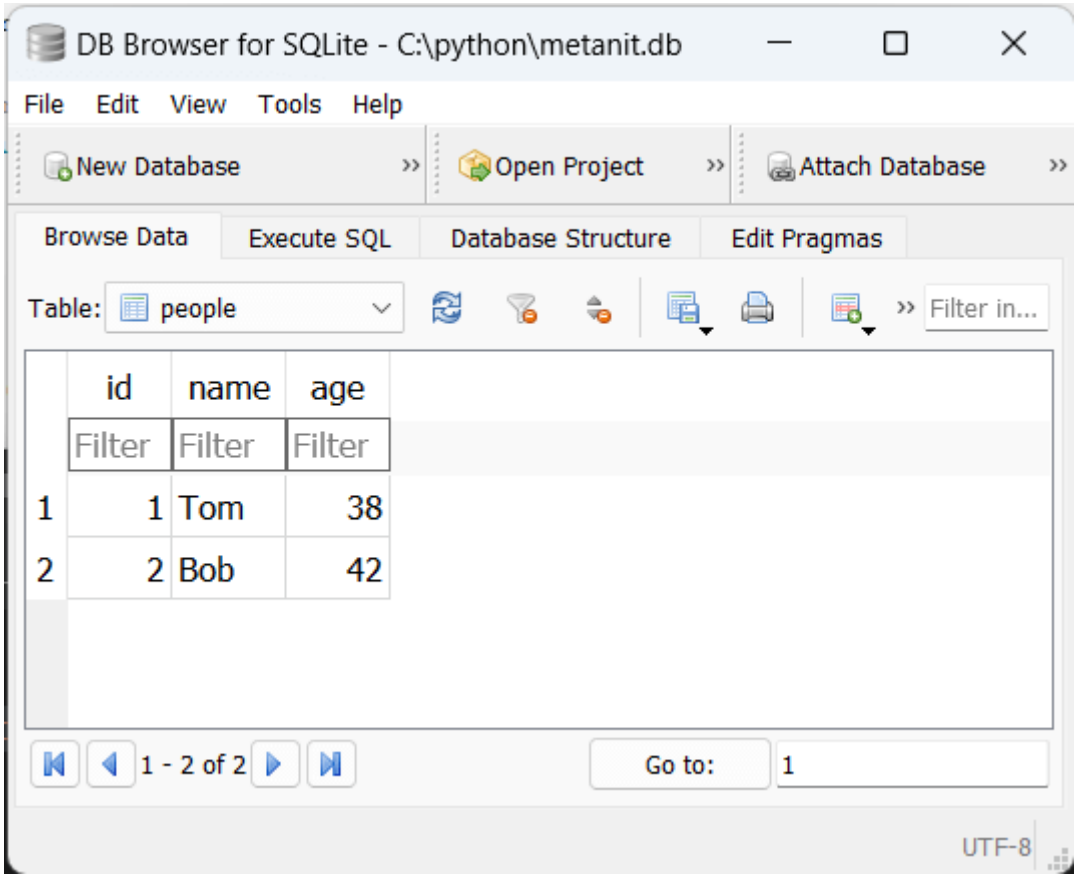
Установка параметров

С помощью второго параметра в метод execute() можно передать значения для параметров SQL-запроса:

```
import sqlite3;

con = sqlite3.connect("metanit.db")
cursor = con.cursor()

# данные для добавления
bob = ("Bob", 42)
cursor.execute("INSERT INTO people (name, age) VALUES (?, ?)", bob)

con.commit()
```

В данном случае добавляемые в БД значения представляют кортеж bob. В SQL-запросе вместо конкретных значений используются знаки подстановки ?. Вместо этих символов при выполнении запроса будут вставляться данные из кортежа data. Так, первый элемент кортежа - строка "Bob" передается на место первого знакак ?, второй элемент - число 42 передается на место второго знака ?.

И если мы посмотрим на содержимое базы данных, то найдем там все добавленные объекты:

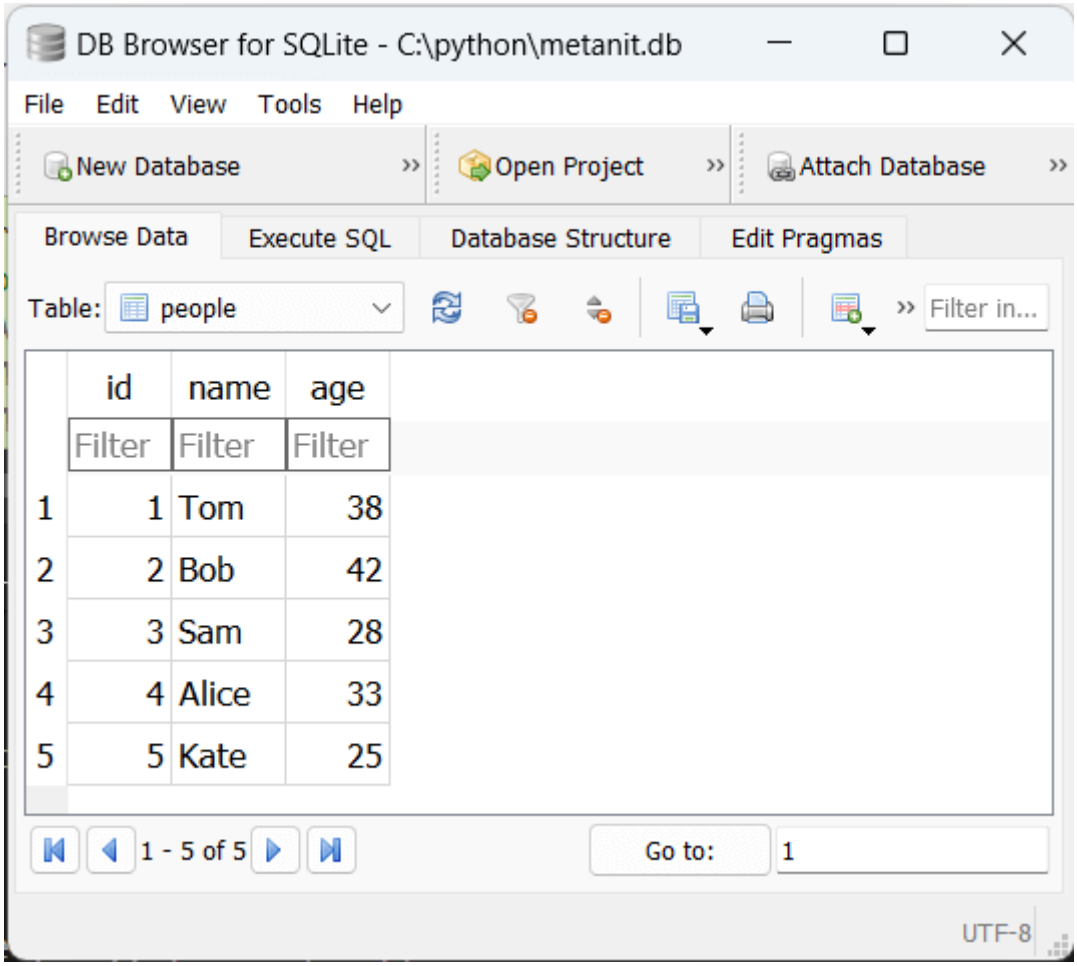


Множественная вставка

Метод **executemany()** позволяет вставить набор строк:

```
import sqlite3;

con = sqlite3.connect("metanit.db")
cursor = con.cursor()

# данные для добавления
people = [("Sam", 28), ("Alice", 33), ("Kate", 25)]
cursor.executemany("INSERT INTO people (name, age) VALUES (?, ?)", people)

con.commit()
```

В метод cursor.executemany() по сути передается то же самое выражение SQL, только теперь данные определены в виде списка кортежей people. Фактически каждый кортеж в этом списке представляет отдельную строку - данные отдельного пользователя, и при выполнении метода для каждого кортежа будет создаваться свое выражение INSERT INTO



Получение данных

Для получения данных применяется SQL-команда SELECT. После выполнения этой команды курсор получает данные, которые можно получить с помощью одного из методов: fetchall() (возвращает список со всеми строками), fetchmany() (возвращает указанное количество строк) и fetchone() (возвращает одну в наборе строку).

Получение всех строк

Например, получим все ранее добавленные данные из таблицы people:

```
import sqlite3;


con = sqlite3.connect("metanit.db")
cursor = con.cursor()


# получаем все данные из таблицы people
cursor.execute("SELECT * FROM people")
```

```
print(cursor.fetchall())
```

При выполнении этой программы на консоль будет выведен список строк, где каждая строка представляет кортеж:

[(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28), (4, 'Alice', 33), (5, 'Kate', 25)]

При необходимости мы можем перебрать список, используя стандартные циклические конструкции:

```
import sqlite3;


con = sqlite3.connect("metanit.db")
cursor = con.cursor()


cursor.execute("SELECT * FROM people")
for person in cursor.fetchall():
    print(f"{person[1]} - {person[2]}")
```

Результат работы программы:

Tom - 38

Bob - 42

Sam - 28

Alice - 33

Kate - 25

Стоит отметить, что в примере выше необязательно вызывать метод fetchall, мы можем перебрать курсор в цикле как обычный набор:

```
for person in cursor:
    print(f"{person[1]} - {person[2]}")
```

Получение части строк

Получение части строк с помощью метода fetchmany(), в который передается количество строк:

```
import sqlite3;
```

```
con = sqlite3.connect("metanit.db")
cursor = con.cursor()

cursor.execute("SELECT * FROM people")
# извлекаем первые 3 строки в полученном наборе
print(cursor.fetchmany(3))
```

Результат работы программы:

```
[(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28)]
```

Выполнение этого метода извлекает следующие ранее неизвлеченные строки:

```
# извлекаем первые 3 строки в полученном наборе
print(cursor.fetchmany(3))  # [(1, 'Tom', 38), (2, 'Bob', 42), (3, 'Sam', 28)]
# извлекаем следующие 3 строки в полученном наборе
print(cursor.fetchmany(3))  # [(4, 'Alice', 33), (5, 'Kate', 25)]
```

Получение одной строки

Метод **fetchone()** извлекает следующую строку в виде кортежа значений и возвращает его. Если строк больше нет, то возвращает None:

```
import sqlite3;

con = sqlite3.connect("metanit.db")
cursor = con.cursor()

cursor.execute("SELECT * FROM people")
# извлекаем одну строку
print(cursor.fetchone())    # (1, 'Tom', 38)
```

Данный метод удобно применять, когда нам надо извлечь из базы данных только один объект:

```
import sqlite3;

con = sqlite3.connect("metanit.db")
```

```
cursor = con.cursor()


cursor.execute("SELECT name, age FROM people WHERE id=2")
# раскладываем кортеж на две переменных
name, age = cursor.fetchone()
print(f"Name: {name}  Age: {age}")  # Name: Bob  Age: 42
```

Здесь получаем из бд строку с id=2, и полученный результат раскладываем на две переменных name и age (так как кортеж в Python можно разложить на отдельные значения)

Обновление

Для обновления в SQL выполняется команда UPDATE. Например, заменим у всех пользователей имя с Tom на Tomas:

```
import sqlite3;
con = sqlite3.connect("metanit.db")
cursor = con.cursor()
# обновляем строки, где name = Tom
cursor.execute("UPDATE people SET name ='Tomas' WHERE name='Tom'")
# вариант с подстановками
# cursor.execute("UPDATE people SET name =? WHERE name=?", ("Tomas", "Tom"))
con.commit()
# проверяем
cursor.execute("SELECT * FROM people")
print(cursor.fetchall())  # [(1, 'Tomas', 38), (2, 'Bob', 42), (3, 'Sam', 28), (4, 'Alice', 33), (5, 'Kate', 25)]
```

Для выполнения обновления также надо выполнять метод con.commit()

Удаление данных

Для удаления в SQL выполняется команда DELETE. Например, удалим всех пользователей с именем Bob:

```
import sqlite3;
```

```
con = sqlite3.connect("metanit.db")
cursor = con.cursor()

# обновляем строки, где name = Tom
cursor.execute("DELETE FROM people WHERE name=?", ("Bob",))
con.commit()

# проверяем
cursor.execute("SELECT * FROM people")
print(cursor.fetchall())    # [(1, 'Tomas', 38), (3, 'Sam', 28), (4, 'Alice', 33), (5, 'Kate', 25)]
```

Для выполнения удаления также надо выполнять метод con.commit()