Модуль 11. Получение данных из разных источников

[Введение 1](#_Toc143242309)

[Получение и обработка данных из разных источников. Модули csv, json 2](#_Toc143242310)

[Данные CSV 2](#_Toc143242311)

[Работа со словарями 4](#_Toc143242312)

[Бинарные файлы 6](#_Toc143242313)

[Язык описания данных json 8](#_Toc143242314)

[Конвертация из JSON в Python 8](#_Toc143242315)

[Конвертировать из Python в JSON 9](#_Toc143242316)

[Как конвертировать кириллицу 12](#_Toc143242317)

[Форматирование результата 13](#_Toc143242318)

[Упорядочивание результата 13](#_Toc143242319)

[Десериализация JSON 14](#_Toc143242320)

[Работа с БД на примере модуля sqlite3 16](#_Toc143242321)

[Что такое СУБД и реляционные БД 16](#_Toc143242322)

[Работа с файлами Excel. Модуль openpyxl 25](#_Toc143242323)

[Запись данных в файл Excel 25](#_Toc143242324)

[Диаграммы и графики 26](#_Toc143242325)

# Введение

Превосходная гибкость и простота использования Python делают его одним из самых популярных языков программирования, особенно для Data Scientists. Большая часть этого заключается в том, насколько просто работать с большими наборами данных.

Сегодня каждая технологическая компания строит стратегию передачи данных. Все они поняли, что наличие правильных данных: проницательных, чистых и как можно большего их количества дает им ключевое конкурентное преимущество. Данные, если они используются эффективно, могут дать глубокое понимание, которое невозможно обнаружить где-либо еще.

За прошедшие годы список возможных форматов, в которых вы можете хранить свои данные, значительно расширился. Но есть 3, которые доминируют в их повседневном использовании: CSV, JSON и XML. В этой статье я расскажу вам о самых простых способах работы с этими 3 популярными форматами данных в Python!

# Получение и обработка данных из разных источников. Модули csv, json

## Данные CSV

Файл CSV является наиболее распространенным способом хранения ваших данных. Вы обнаружите, что большая часть данных, полученных на соревнованиях Kaggle, хранится таким образом. Мы можем выполнять чтение и запись CSV-файла с помощью встроенной библиотеки Python CSV. Обычно мы читаем данные в список списков.

Одним из распространенных файловых форматов, которые хранят в удобном виде информацию, является формат csv. Каждая строка в файле csv представляет отдельную запись или строку, которая состоит из отдельных столбцов, разделенных запятыми. Собственно поэтому формат и называется Comma Separated Values. Но хотя формат csv - это формат текстовых файлов, Python для упрощения работы с ним предоставляет специальный встроенный модуль csv.

Рассмотрим работу модуля на примере:

import csv

FILENAME = "users.csv"

users = [

["Tom", 28],

["Alice", 23],

["Bob", 34]

]

with open(FILENAME, "w", newline="") as file:

writer = csv.writer(file)

writer.writerows(users)

with open(FILENAME, "a", newline="") as file:

user = ["Sam", 31]

writer = csv.writer(file)

writer.writerow(user)

В файл записывается двухмерный список - фактически таблица, где каждая строка представляет одного пользователя. А каждый пользователь содержит два поля - имя и возраст. То есть фактически таблица из трех строк и двух столбцов.

При открытии файла на запись в качестве третьего параметра указывается значение newline="" - пустая строка позволяет корректно считывать строки из файла вне зависимости от операционной системы.

Для записи нам надо получить объект writer, который возвращается функцией csv.writer(file). В эту функцию передается открытый файл. А собственно запись производится с помощью метода writer.writerows(users) Этот метод принимает набор строк. В нашем случае это двухмерный список.

Если необходимо добавить одну запись, которая представляет собой одномерный список, например, ["Sam", 31], то в этом случае можно вызвать метод writer.writerow(user)

В итоге после выполнения скрипта в той же папке окажется файл users.csv, который будет иметь следующее содержимое:

Tom,28

Alice,23

Bob,34

Sam,31

Для чтения из файла нам наоборот нужно создать объект reader:

import csv

FILENAME = "users.csv"

with open(FILENAME, "r", newline="") as file:

reader = csv.reader(file)

for row in reader:

print(row[0], " - ", row[1])

При получении объекта reader мы можем в цикле перебрать все его строки:

Tom - 28

Alice - 23

Bob - 34

Sam – 31

## Работа со словарями

В примере выше каждая запись или строка представляла собой отдельный список, например, ["Sam", 31]. Но кроме того, модуль csv имеет специальные дополнительные возможности для работы со словарями. В частности, функция csv.DictWriter() возвращает объект writer, который позволяет записывать в файл. А функция csv.DictReader() возвращает объект reader для чтения из файла. Например:

import csv

FILENAME = "users.csv"

users = [

{"name": "Tom", "age": 28},

{"name": "Alice", "age": 23},

{"name": "Bob", "age": 34}

]

with open(FILENAME, "w", newline="") as file:

columns = ["name", "age"]

writer = csv.DictWriter(file, fieldnames=columns)

writer.writeheader()

# запись нескольких строк

writer.writerows(users)

user = {"name" : "Sam", "age": 41}

# запись одной строки

writer.writerow(user)

with open(FILENAME, "r", newline="") as file:

reader = csv.DictReader(file)

for row in reader:

print(row["name"], "-", row["age"])

Запись строк также производится с помощью методов writerow() и writerows(). Но теперь каждая строка представляет собой отдельный словарь, и кроме того, производится запись и заголовков столбцов с помощью метода writeheader(), а в метод csv.DictWriter в качестве второго параметра передается набор столбцов.

При чтении строк, используя названия столбцов, мы можем обратиться к отдельным значениям внутри строки: row["name"].

Иногда лучше, чтобы все строки были в кавычках. Конечно, в данном случае достаточно простой пример, но когда в строках больше значений, то кавычки позволяют указать, где начинается и заканчивается значение.

Модуль csv позволяет управлять этим. Для того, чтобы все строки записывались в CSV-файл с кавычками, надо добавить параметр quoting=csv.QUOTE\_NONNUMERIC

writer = csv.writer(f, quoting=csv.QUOTE\_NONNUMERIC)

Указание разделителя Иногда в качестве разделителя используются другие значения. В таком случае должна быть возможность подсказать модулю, какой именно разделитель использовать. Например, если в файле используется разделитель ;

reader = csv.reader(f, delimiter=';')

**Задание 1.** Имеется лог-файл crm\_log.txt с информацией о времени входа пользователей в CRM-систему и о времени выхода. Напишите программу, которая определяет пользователей, работавших в системе более 4 часов, и записывает их имена и фамилии в файл best\_employees.csv.

Пример лога crm\_log.txt:

Егор Тимофеев, 09:10, 16:50

Марина Абрамова, 12:00, 15:59

Никита Круглов, 09:10, 12:45

Анна Семенова, 08:10, 12:30

Юлия Сафонова, 10:10, 10:50

Михаил Колесников, 11:10, 12:10

Выходные данные в файле best\_employees.csv

Егор Тимофеев, 460 мин

Анна Семенова, 260 мин

users = []  
f = open('text.txt', encoding='utf-8')  
for line in f:  
 line = line.split(',')  
 start, end = line[-2], line[-1]  
 start = start.split(':')  
 end = end.split(':')  
 start = int(start[0])\*60 + int(start[1])  
 end = int(end[0])\*60 + int(end[1])  
 delta = end - start  
 if (delta) >=240:  
 users.append((line[0], f'{end-start} мин'))  
  
print(users)  
import csv  
  
with open('best\_employees.csv', "w", newline="", encoding='utf-8') as file:  
 writer = csv.writer(file)  
 writer.writerows(users)

# Язык описания данных json

Сегодня говорим о таком языке описания данных как json и как с ним управляться в программах на языке Python. Вопрос актуальный просто потому, что json широко используется для передачи и хранения данных. Например, очень удобно хранить настройки программы именно в этом формате.

JSON является синтаксисом для хранения и обмена данными. JSON — это текст, написанный в стиле объекта JavaScript.

Python имеет встроенный модуль json, который может использоваться для работы с данными JSON.

import json

## Конвертация из JSON в Python

Если у вас есть строка JSON, вы можете провести над ней парсинг с помощью метода json.loads ().  
Как результат, будет [словарь python](https://pythonru.com/uroki/slovar-dict-uroki-po-python-dlja-nachinajushhih).  
Конвертируем из JSON в Python:

import json

# немного JSON:

x = '{"name":"Viktor", "age":30, "city":"Minsk"}'

# парсинг x:

y = json.loads(x)

# результатом будет словарь Python:

print(y["age"])

Вывод:

30

## Конвертация из Python в JSON

Если у вас есть объект Python, вы можете преобразовать его в строку JSON с помощью метода json.dumps().

import json

# создаем словарь x:

x = {

"name": "Viktor",

"age": 30,

"city": "Minsk"

}

# конвертируем в JSON:

y = json.dumps(x)

# в результате получаем строк JSON:

print(y)

Вывод:

{"name": "Viktor", "age": 30, "city": "Minsk"}

Вы можете преобразовать следующие типов объекты Python в строки JSON:

* dict
* list
* tuple
* string
* int
* float
* True
* False
* None

Конвертируем объекты Python в строки JSON и выведите значения:

Когда вы конвертируете из Python в JSON, объекты Python преобразуются в эквивалент JSON:

| Python | JSON |
| --- | --- |
| dict | Object |
| list | Array |
| tuple | Array |
| str | String |
| int | Number |
| float | Number |
| True | true |
| False | false |
| None | null |

Конвертируйте объект Python, содержащий все типы конвертируемых данных:

import json

x = {

"name": "Viktor",

"age": 30,

"married": True,

"divorced": False,

"children": ("Anna","Bogdan"),

"pets": None,

"cars": [

{"model": "BMW 230", "mpg": 27.5},

{"model": "Ford Edge", "mpg": 24.1}

]

}

print(json.dumps(x))

Вывод:

{"name": "Viktor", "age": 30, "married": true, "divorced": false, "children": ["Anna", "Bogdan"], "pets": null, "cars": [{"model": "BMW 230", "mpg": 27.5}, {"model": "Ford Edge", "mpg": 24.1}]}

## Как конвертировать кириллицу

Если в данных Python есть символы кириллицы, метод json.dumps() преобразует их с кодировкой по умолчанию. Что бы сохранить кириллицу используйте параметр ensure\_ascii=False

import json

x = {

"name": "Виктор"

}

y = {

"name": "Виктор"

}

print(json.dumps(x))

print(json.dumps(y, ensure\_ascii=False))

Вывод:

{"name": "\u0412\u0438\u043a\u0442\u043e\u0440"}

{"name": "Виктор"}

## Форматирование результата

В приведенном выше примере выводится строка JSON, но читать ее не так просто, без отступов и переносов строк.

У метода json.dumps() есть параметры, облегчающие чтение результата.  
Используем параметр indent для определения количества отступов:

json.dumps(x, indent=4)

Вы также можете определить разделители, значение которых по умолчанию — ,, :, где запятая и пробел используются для разделения каждого объекта, а двоеточие и пробел — для разделения ключей и значений.  
Используем параметр separators чтобы изменить разделитель по умолчанию:

json.dumps(x, indent=4, separators=(". ", " = "))

## Упорядочивание результата

Метод json.dumps() имеет параметры для упорядочивания ключей в результате.  
Используем параметр sort\_keys чтобы указать, должен ли сортироваться результат.

json.dumps(x, indent=4, sort\_keys=True)

Сохранить эту информацию на диск — критично, так что ваша задача — [записать на файл](https://python-scripts.com/work-with-files-python).

Используя [контекстный менеджер Python](https://python-scripts.com/contextlib), вы можете создать файл под названием **data\_file.json** и открыть его в режиме write (файлы JSON имеют расширение **.json**).

Python

|  |  |
| --- | --- |
| 1  2 | with open("data\_file.json", "w") as write\_file:  json.dump(data, write\_file) |

## Десериализация JSON

Отлично, похоже вам удалось поймать экземпляр дикого JSON! Теперь нам нужно предать ему форму. В **модуле json** вы найдете **load**() и **loads**() для превращения кодированных данных JSON в объекты Python.

Технически, эта конверсия не является идеальной инверсией **таблицы сериализации**. По сути, это значит что если вы кодируете объект сейчас, а затем декодируете его в будущем, вы можете не получить тот же объект назад. Я представляю это как своего рода телепортацию: мои молекулы распадаются в точке А и собираются в точке Б. Буду ли я тем же самым человеком?

В реальности, это как попросить одного друга перевести что-нибудь на японский, а потом попросить другого друга перевести это обратно на русский. В любом случае, самым простым примером будет **кодировать кортеж** и получить назад список после декодирования, вот так:

a = (8, "Q")  
b = json.dumps(a)  
c = json.loads(b)  
  
print(a == c) # False  
  
print(type(a)) # <class 'tuple'>  
print(type(c)) # <class 'list'>  
  
print(a == tuple(c)) # True

Для тестового API, мы воспользуемся FAKESTOREAPI отличный источник [фейковых данных](https://python-scripts.com/elizabeth) JSON для практических целей.

Вам нужно выполнить запрос API в сервис FAKESTOREAPI, так что просто используйте пакет [requests](https://python-scripts.com/requests" \t "_blank), чтобы он сделал за вас всю грязную работу. Добавьте следующие импорты вверху файла:

import json

import requests

from pprint import pprint  
  
import requests, csv  
BASE\_URL = 'https://fakestoreapi.com'  
response = requests.get(f"{BASE\_URL}/products")  
json = response.json()[:2]  
*# json['price'] = 200  
# json['description'] = 'мой товар'  
# pprint(json)*columns = list(json[1].keys())  
*# print(columns)  
А теперь запишем данный в csv*

with open('text.csv', "a", newline="",encoding='utf-8') as file:  
 writer = csv.DictWriter(file, fieldnames=columns)  
 *# writer.writeheader()* writer.writerows(json)

# Бинарные файлы

Бинарные файлы в отличие от текстовых хранят информацию в виде набора байт. Для работы с ними в Python необходим встроенный модуль pickle. Этот модуль предоставляет два метода:

dump(obj, file): записывает объект obj в бинарный файл file

load(file): считывает данные из бинарного файла в объект

При открытии бинарного файла на чтение или запись также надо учитывать, что нам нужно применять режим "b" в дополнение к режиму записи ("w") или чтения ("r"). Допустим, надо надо сохранить два объекта:

import pickle

FILENAME = "user.dat"

name = "Tom"

age = 19

with open(FILENAME, "wb") as file:

pickle.dump(name, file)

pickle.dump(age, file)

with open(FILENAME, "rb") as file:

name = pickle.load(file)

age = pickle.load(file)

print("Имя:", name, "\tВозраст:", age)

С помощью функции dump последовательно записываются два объекта. Поэтому при чтении файла также последовательно посредством функции load мы можем считать эти объекты. Консольный вывод программы:

Имя: Tom Возраст: 28

Подобным образом мы можем сохранять и извлекать из файла наборы объектов:

import pickle

FILENAME = "users.dat"

users = [

["Tom", 28, True],

["Alice", 23, False],

["Bob", 34, False]

]

with open(FILENAME, "wb") as file:

pickle.dump(users, file)

with open(FILENAME, "rb") as file:

users\_from\_file = pickle.load(file)

for user in users\_from\_file:

print("Имя:", user[0], "\tВозраст:", user[1], "\tЖенат(замужем):", user[2])

В зависимости от того, какой объект мы записывали функцией dump, тот же объект будет возвращен функцией load при считывании файла.

Консольный вывод:

Имя: Tom Возраст: 28 Женат(замужем): True

Имя: Alice Возраст: 23 Женат(замужем): False

Имя: Bob Возраст: 34 Женат(замужем): False

Выводы:

Итак, сериализация (англ. serialization, marshalling) – это способ преобразования структуры данных в линейную форму, которую можно сохранить или передать по сети. Обратный процесс преобразования сериализованного объекта в исходную структуру данных называется десериализацией (англ. deserialization, unmarshalling).

В стандартной библиотеке Python три модуля позволяют сериализовать и десериализовать объекты:

marshal

json

pickle

Кроме того, Python поддерживает XML, который также можно применять для сериализации объектов.

Модуль pickle специфичен для Python — результаты сериализации могут быть прочитаны только другой программой на Python.

# Работа с БД на примере модуля sqlite3

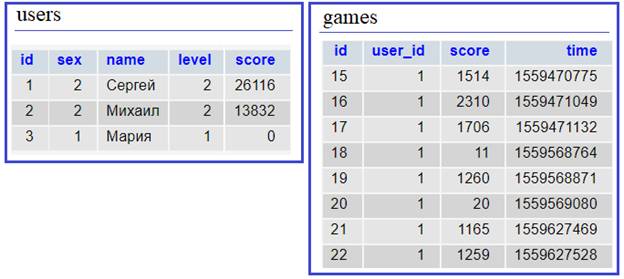
## Что такое СУБД и реляционные БД

Этим занятием мы открываем небольшой цикл по базам данных в Python. И, как всегда, вначале отметим на вопрос: для чего вообще нужен этот инструмент и что он собой представляет?

Предположим, что мы делаем игру, например, «Сапер» и в ней предполагается сохранять результаты игр и профили игроков. Обычно, для этих целей создают базу данных (БД) с набором таблиц. Пусть это будут таблицы «Игроки» (users) и «Игры» (games).



Каждая таблица имеет заданную структуру (набор определенных столбцов) и определенное количество записей – строк. Например, так:



Структура первой таблицы users определяется набором полей (столбцов):

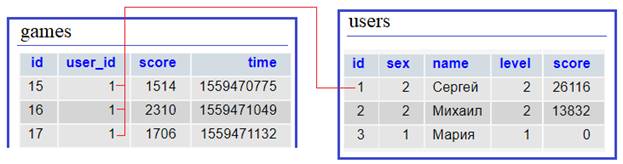
* id (тип integer) – уникальный идентификатор записи (строки);
* sex (integer) – пол игрока (1 – женский; 2 – мужской);
* name (text) – имя игрока;
* level (integer) – уровень игрока;
* score (integer) – максимальное число очков, заработанных игроком.

Структура второй таблицы games, следующая:

* id (тип integer) – уникальный идентификатор записи (строки);
* user\_id (integer) – внешний ключ для связи с таблицей users;
* score (integer) – число очков, набранных в игре;
* time (integer) – время начала игры.

Разумеется, это лишь примеры таблиц для хранения данных, чтобы вам было проще понимать как все это организовано. И, кроме того, структура каждой таблицы определяется разработчиком и может быть самой разной.

В приведенном примере, таблица games имеет внешний ключ user\_id, содержащий id игрока, участвующего в соответствующих играх. Затем, по этому ключу можно будет выполнять связывание таблиц и получать сводные данных по играм и данным игрока. Такая связь по-английски звучит как relation, откуда и пошло название реляционные базы данных, то есть, базы, содержащие таблицы с возможностью связываться между собой. В дальнейшем мы будем говорить исключительно о таких типах БД.



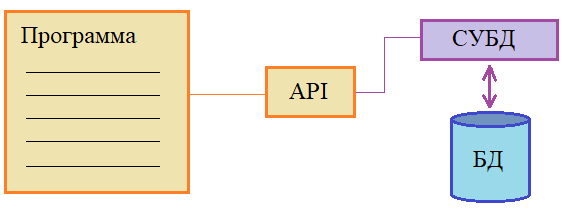
Ну, хорошо, с понятием базы данных разобрались. Но зачем так «городить огород», когда можно просто взять и записать все во внешний файл и там хранить информацию об игроках и играх?

Дело в том, что при работе с обычным файлом возникает ряд сопутствующих задач, которые необходимо будет решить программисту.

Например, как обеспечить безопасность данных, если в момент записи произойдет программный сбой или попросту не хватит места на носителе? Продумать формат хранимой информации непосредственно в файле.

Как обратиться только к отдельной записи и как соединить записи из нескольких таблиц. И так далее. Это только первичные вопросы. В ходе реализации наверняка подтянутся и другие. Чтобы не иметь таких типовых проблем программисты пользуются различными СУБД для организации хранения различных данных.

Более детально это выглядит так. После выбора определенной СУБД программист получает доступ к ее API (Application Programming Interface) – программному интерфейсу для взаимодействия с СУБД. Фактически, к набору функций, через которые производится работа с базами данных.



Для дальнейшего изложения материала нам нужно определиться с СУБД. При программировании на Python популярными являются следующие:

PyMySQL, Python SQLite и Python PostgreSQL

В рамках наших занятий мы будем рассматривать пакет SQLite как один из самых распространенных и простых в использовании. Кроме того, данный модуль идет в стандартной поставке языка Python, поэтому вам его, скорее всего, не придется отдельно устанавливать.

SQLite используется для создания легковесной дисковой БД, то есть, эта СУБД не поддерживает сетевое взаимодействие (разве что удаленный доступ к файлу БД, но не более того). Также она имеет ограничение при многопользовательском доступе: запись данных может осуществлять только один поток или процесс в один момент времени. А вот на чтение таких ограничений нет. SQLite работает по принципу «один пишет – многие читают» и старается развиваться и функционировать по правилу «минимальный, но полный набор». Она идеально подходит для хранения данных различных приложений.

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

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

## Практическая работа с sqlite3

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

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

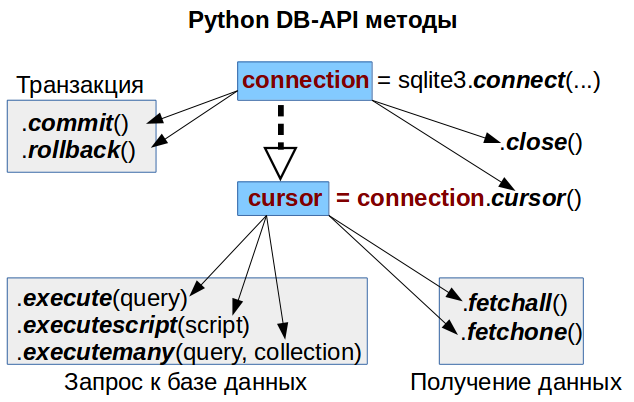
**import** sqlite3,random  
  
con = sqlite3.connect(**"people.db"**)

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

Объектная модель



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:

cursor = con.cursor()

Создание таблицы

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

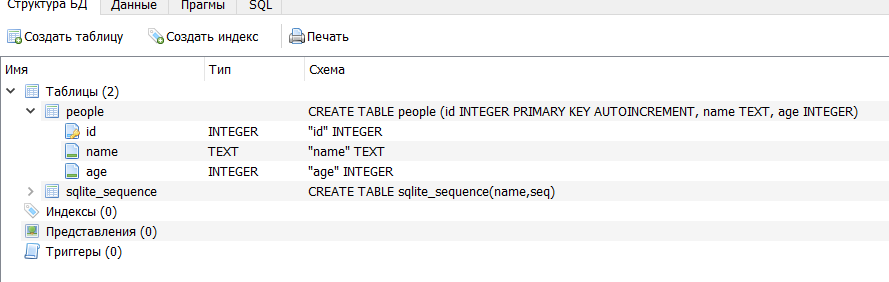
cursor.execute(**"""CREATE TABLE people  
 (id INTEGER PRIMARY KEY AUTOINCREMENT,   
 name TEXT,   
 age INTEGER)  
 """**)

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



con = sqlite3.connect(**"bd.db"**)  
cursor = con.cursor()  
  
cursor.execute(**"INSERT INTO people (name, age) VALUES ('Tom', 38)"**)

Здесь добавляется одна строка, где name = "Tom", а 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:

USERS = [**'АЛЕКСАНДР'**,**'ПАВЕЛ'**, **ОЛЬГА**, **'ОЛЕГ**, **'ИРИНА**, **'АНДРЕЙ**, **'ЕВГЕНИЙ'**, **'ЛИЛИЯ', 'АЛЕКСЕЙ'**]  
rand\_user, age = random.choice(USERS), random.randint(20,60)  
  
print(rand\_user)

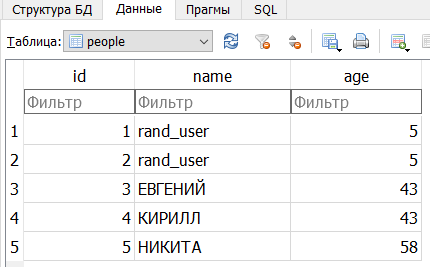
user = (rand\_user,age)  
cursor.execute(**"INSERT INTO people (name, age) VALUES (?, ?)"**, user)  
  
con.commit()

В данном случае добавляемые в БД значения представляют кортеж user.

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 "user" передается на место первого знака ?, 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:

list\_people = []  
**for** \_ **in** range(3):  
 rand\_user, age = random.choice(USERS), random.randint(20, 60)  
 user = (rand\_user,age)  
 list\_people.append(user)  
  
print(list\_people)

cursor.executemany(**"INSERT INTO people (name, age) VALUES (?, ?)"**, list\_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

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

ЕВГЕНИЙ - 43

КИРИЛЛ - 43

НИКИТА – 58

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:

cursor.execute(**"SELECT \* FROM people"**)  
print(cursor.fetchall())

[(15, 'ПАВЕЛ', 38), (16, 'СЕРГЕЙ', 48), (17, 'АЛЕКСАНДР', 49)]

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]**}"**)

ПАВЕЛ - 38

СЕРГЕЙ - 48

АЛЕКСАНДР – 49

Получение части строк

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

print(cursor.fetchmany(2))

[(15, 'ПАВЕЛ', 38), (16, 'СЕРГЕЙ', 48)]

Выполнение этого метода извлекает следующие ранее неизвлеченные строки:

print(cursor.fetchmany(2))  
print(cursor.fetchmany(3))

Получение одной строки

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

print(cursor.fetchone()) # (1, 'Tom', 38)

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

cursor.execute("SELECT name, age FROM people WHERE id=2")

или

cursor.execute(  
 "SELECT name, "  
 "age FROM people WHERE name='ОЛЕГ'")  
*# print(cursor.fetchall())*print(cursor.fetchone())

# 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)

Установим <https://sqlitebrowser.org/> и попробуем извне поработать администратором базы данных.

Обновление

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

cursor.execute(**"UPDATE people SET name ='Саша' WHERE name='АЛЕКСАНДР'"**)

# вариант с подстановками

# cursor.execute("UPDATE people SET name =? WHERE name=?", ("Tomas", "Tom"))

con.commit()

# проверяем

cursor.execute("SELECT \* FROM people")

print(cursor.fetchall())

Удаление данных

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

cursor.execute("DELETE FROM people WHERE name=?", ("ПАВЕЛ",))

con.commit()

# проверяем

cursor.execute("SELECT \* FROM people")

print(cursor.fetchall())

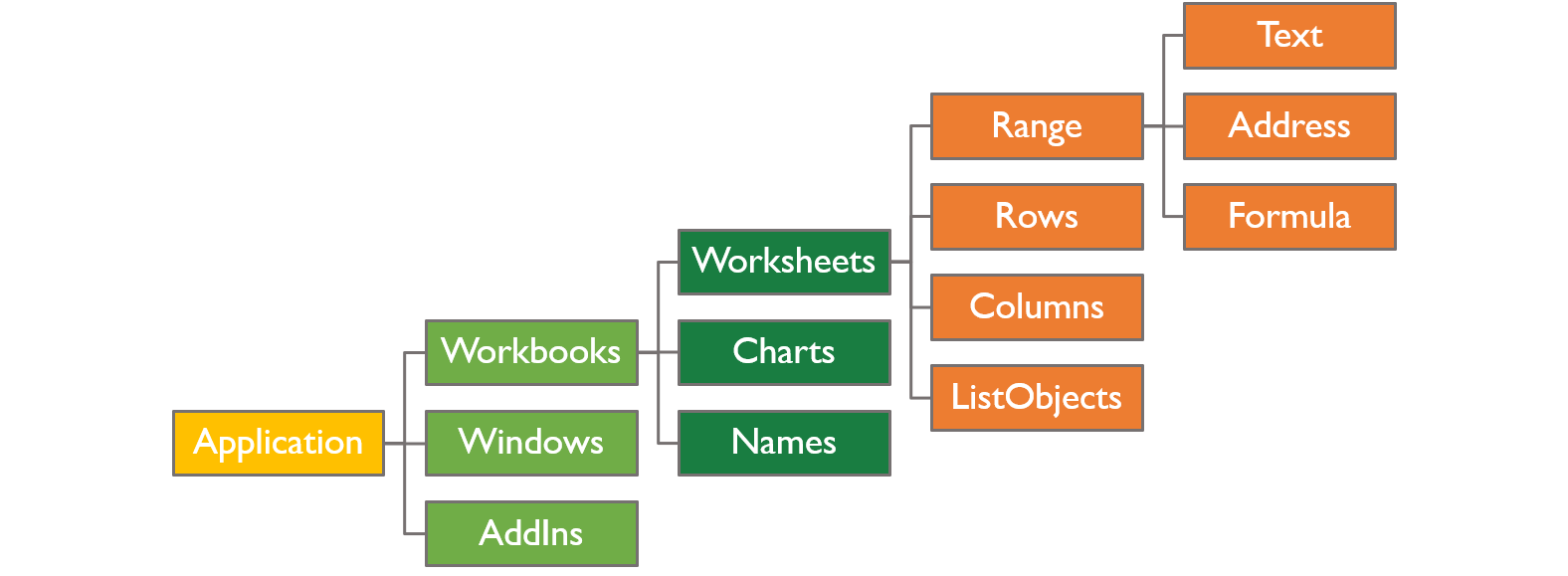
Конечно, на этом занятии мы лишь рассмотрели примеры, принцип создания вложенных запросов. На практике они могут разрастаться и становиться довольно объемными, включать в себя различные дополнительные операции для выполнения нетривиальных действий с таблицами БД.

На этом мы завершим обзор SQL-языка. Этого материала вам будет вполне достаточно для начальной работы с БД. По мере развития сможете дальше углубляться в эту тему и узнавать множество новых подходов к реализации различных задач с помощью SQL-запросов.

Хороший тренажер по SQL <https://stepik.org/course/63054/syllabus>

# Работа с файлами Excel. Модуль openpyxl

Объектная модель



Перед созданием своей первой программы в Excel необходимо разобраться в объектную модели приложения. Имея четкое понимания об объектной модели Excel вы сможете управлять практически всеми объектами, создавать/изменять листы, графики, отдельные файлы  и многое другое.

## Запись данных в файл Excel

Используя метод excel\_doc = op.Workbook() создаем рабочую книгу. Методом create\_sheet(title = 'Лист1', index = 0) создаем рабочие листы книги.

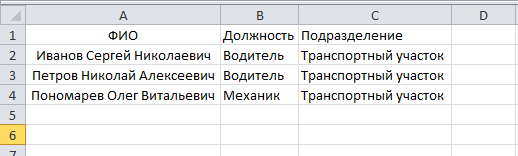
У метода есть два параметра title название листи и index номер листа если их несколько. Метод sheet[f"A{i}"] = "текст" используется для записи значений в таблицу. Сохранение файла осуществляется методом save().

Если указать название файла и расширение, то файл сохранится в той же директории, где находится проект.

Если необходимо сохранить файл в другом месте необходимо указать полный путь имя и расширение файла.

|  |
| --- |
|  |
|  | import openpyxl as op |
|  |  |
|  | list\_tab = [ |
|  | ["ФИО", "Должность","Подразделение"], |
|  | ["Иванов Сергей Николаевич", "Водитель","Транспортный участок"], |
|  | ["Петров Николай Алексеевич", "Водитель","Транспортный участок"], |
|  | ["Пономарев Олег Витальевич", "Механик","Транспортный участок"] |
|  | ] |
|  |  |
|  | excel\_doc = op.Workbook()#Создаем рабочую книгу |
|  | excel\_doc.create\_sheet(title = 'Лист1', index = 0)#Создаем лист |
|  | sheetnames = excel\_doc.sheetnames #Получение списка листов книги |
|  | sheet = excel\_doc[sheetnames[0]] |
|  | i = 1 |
|  | for row in list\_tab: |
|  | a,b,c = row #Распаковываем значения списка в переменные |
|  | sheet[f"A{i}"] = a #Записываем значения в таблицу |
|  | sheet[f"B{i}"] = b |
|  | sheet[f"C{i}"] = c |
|  | i += 1 |
|  |  |
|  | excel\_doc.save('Тест.xlsx') |

Результат работы:



## Диаграммы и графики

Библиотека openpyxl поддерживает создание графиков, диаграмм и других объектов excel с использование данных, хранящихся в электронной таблице.

Для создания диаграммы, необходимо выполнить следующие действия:

Импортировать необходимые модули библиотеки openpyxl для работы с графиками и диаграммами.

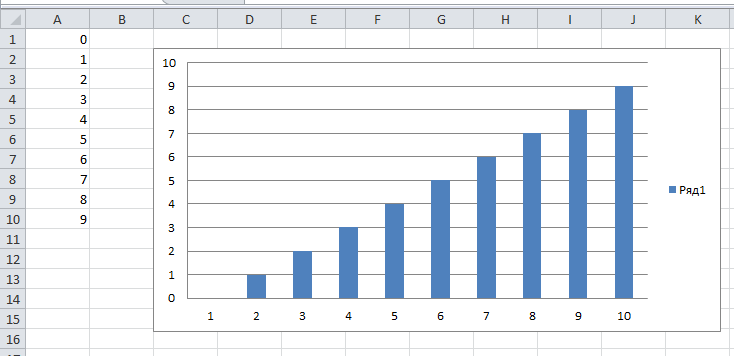
Создать объект Reference() указав диапазон ячеек с данными для построения графика.  
Создаем объект chart() указывая тип создаваемого объекта: график, диаграмма и др.

С методом add\_chart() передаем создаваемый объект и указываем начальное расположение графика. Верхний левый угол диаграммы привязан к ячейке С2, а размер по умолчанию 15 х 7,5 см.

import openpyxl as op

|  |
| --- |
|  |
|  | from openpyxl.chart import BarChart, Reference, Series |
|  |  |
|  | excel\_doc = op.Workbook() #Создаем рабочую книгу |
|  | excel\_doc.create\_sheet(title = 'Лист1', index = 0)#Создаем лист |
|  | sheetnames = excel\_doc.sheetnames #Получение списка листов книги |
|  | sheet = excel\_doc[sheetnames[0]] |
|  | for i in range(10): |
|  | sheet[f"A{i+1}"]= i #Заполняем файл excel для построенияя графика |
|  |  |
|  |  |
|  | values = Reference(sheet, min\_col=1, min\_row=1, max\_col=1, max\_row=10)# указываем область с данными для построения графика |
|  | chart = BarChart() #Указываем тип графика |
|  | chart.add\_data(values)#Передаем значения для построения графика |
|  | sheet.add\_chart(chart, "C2")#Добавляем график на страницу, указывая положение на листе |
|  | excel\_doc.save("Тест.xlsx") #Сохраняем файл |

Результат работы:



Аналогично можно создавать диаграммы и графики, точечные, круговые, вызывая методы: chart.LineChart(), chart.ScatterChart(), chart.PieChart()

В данной статье мы рассмотрели основы работы с библиотекой openpyxl, научились читать и записывать файлы Excel, создавать новые файлы, а также строить диаграммы и графики.