Лекция 9: Работа с базами данных

Автор: Сергей Вячеславович Макрушин, 2022 г.

e-mail: s-makrushin@yandex.ru (mailto:s-makrushin@yandex.ru)

V 0.3 10.11.2022

In [10]:
```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[10]:

Разделы:

-

- к оглавлению

SQLite

-

- к оглавлению

SQLite — компактная встраиваемая СУБД. SQLite реализована на языке C. SQLite представляет собой SQL-движок со следующими фичами:

- компактный
- быстрый
- автономный
- надежный
- полнофункциональный
- "Встраиваемая" (embedded) означает, что SQLite не использует парадигму клиент-сервер. Т.е. движок SQLite
  - **НЕ является отдельно работающим процессом**, с которым взаимодействует программа;
  - **представляет собой библиотеку**, с которой программа компонуется, и движок становится составной частью программы.
  - обменивается данными с клиентским кодом с использованием вызовов функций (API) библиотеки SQLite, что уменьшает накладные расходы, время отклика и упрощает программу.
- SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа.
- Перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется, что обеспечивает простоту их реализации.
- ACID-функции достигаются в том числе за счёт создания файла журнала.

SQLite - самая распространенная СУБД в мире. SQLite встроена во все современные мобильные телефоны и в большинство компьютеров и в бесчисленное множество других приложений. По умолчанию SQLite включена в:

- Google's Android
- LG's webOS
- Windows 10
- Apple adopted it as an option in macOS's Core Data API from the original implementation in Mac OS X 10.4 onwards, and also for administration of videos and songs, and in iOS for storage of text messages on the iPhone.[43]

- Blackberry's BlackBerry 10 OS
- Symbian OS
- Nokia's Maemo
- Linux Foundation's MeeGo
- NetBSD
- FreeBSD where starting with 10-RELEASE version in January 2014, it is used by the core package management system. illumos
- Oracle Solaris 10 where the Service Management Facility database is serialized for booting.
- MorphOS since version 3.10
- Tizen

Доступ к базе данных SQLite из Python

In [74]:
```
import  sqlite3 # Подключаем модуль
sqlite3.apilevel #  Получаем номер  спецификации
```

Out[74]: '2.0'

In [75]:
```
sqlite3.sqlite_version
```

Out[75]: '3.30.0'

Согласно спецификации DB-API 2.0 последовательность работы с базой данных выглядит следующим образом:

1. Производится **подключение** к базе данных с помощью функции `connect()` . Функция возвращает объект соединения, с помощью которого осуществляется дальнейшая работа с базой данных.
2. Создается **объект-курсор**.
3. **Выполняются SQL-запросы и обрабатываются результаты**. Перед выполнением первого запроса. который изменяет записи (INSERT, REPIACE, UPDATE и DELETE), автоматически запускается транзакция.
4. **Завершается транзакция** или отменяются все изменения в рамках транзакции.
5. **Закрывается объект-курсор**.
6. **Закрывается соединение** с базой данных.

Создание и открытие базы данных

Для создания и открытия базы данных используется функция `connect()` . Функция имеет следующий формат:

`connect(database[, timeout] [, lsolatlon_level] [, detect types] [, factory] [, check_same_thread] [, cached_statements])`

- В параметре database указывается абсолютный или относительный путь к базе данных.
  - Если база данных не существует, то она будет создана и открыта для работы.
  - Если база данных уже существует, то она просто открывается без удаления имеющихся данных.
  - Вместо пути к базе данных можно указать значение `:memory:`, которое означает, что база данных будет создана в оперативной памяти. После закрытия такой базы все данные будут удалены.
- Все **остальные параметры являются необязательными** и могут быть указаны в произвольнам порядке путем присвоения значения названию параметра.
- Необязательный параметр `timeout` задает время ожидания снятия блокировки с открываемой базы данных. По умолчанию значение параметра timeout равно пяти секундам. Предназначение остальных параметров мы рассмотрим немного позже.

Функция `connect()` возвращает объект соединения, с помощью которого осуществляется вся дальнейшая работа с базой данных. Если открыть базу данных не удалось, то возбуждается исключение. Соединение закрывается, когда вызывается метод `close()` объекта соединения.

In [76]:
```
# import  sqlite3 #  Подключаем модуль  sqlite3
con = sqlite3.connect("testdb.db") # Открьrпаем базу данных

#  Работаем с  базой данных

con.close() #  Закрьrпаем базу данных
```

После создания объекта соединения необходимо создать **объект-курсор**. Все дальнейшие запросы должны производиться через этот объект. Создание объекта-курсора производится с помощью метода `cursor()`. Для выполнения запроса к базе данных предназначены следующие методы объекта-курсора:

- `close()` - закрывает объект-курсор.
- `executescript(< SQL-запросы через точку с запятой>)` - выполняет несколько SQL-запросов за один раз. Если в процессе выполнения запросов возникает ошибка, то метод возбуждает исключение.
- `execute(< SQL-запрос> [, <Значения>])` - выполняет один SQL-запрос. Если в процессе выполнения запроса возникает ошибка, то метод возбуждает исключение.

In [77]:
```
# import  sqlite3

con = sqlite3.connect("catalog.db")
cur = con.cursor() #  Создаем объект-курсор
```

In [63]:
```
# несколько SQL выражений создающих простую БД:
sql  = """
CREATE TABLE ALBUM
(
    ALBUM_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    TITLE VARCHAR(200)  NOT NULL,
    ARTIST_ID INTEGER  NOT NULL,
    FOREIGN KEY (ARTIST_ID) REFERENCES ARTIST (ARTIST_ID)
);

CREATE TABLE ARTIST
(
    ARTIST_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME VARCHAR(200),
    COMMENT VARCHAR(400)
);

CREATE TABLE GENRE
(
    GENRE_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME VARCHAR(200)
);

CREATE TABLE PLAY_LIST
(
    PLAY_LIST_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME VARCHAR(200)
);

CREATE TABLE PLAY_LIST_TRACK
(
    PLAY_LIST_ID INTEGER  NOT NULL,
    TRACK_ID INTEGER  NOT NULL,
    CONSTRAINT PLAY_LIST_TRACK PRIMARY KEY  (PLAY_LIST_ID, TRACK_ID),
    FOREIGN KEY (PLAY_LIST_ID) REFERENCES PLAY_LIST (PLAY_LIST_ID),
    FOREIGN KEY (TRACK_ID) REFERENCES TRACK (TRACK_ID)
);

CREATE TABLE TRACK
(
    TRACK_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME VARCHAR(200)  NOT NULL,
    ALBUM_ID INTEGER,
    MEDIA_TYPE_ID INTEGER  NOT NULL,
    GENRE_ID INTEGER,
    COMPOSER VARCHAR(200),
    MILLISECONDS INTEGER  NOT NULL,
    FOREIGN KEY (ALBUM_ID) REFERENCES ALBUM (ALBUM_ID),
    FOREIGN KEY (GENRE_ID) REFERENCES GENRE (GENRE_ID)
);"""
```

In [78]:
```
try:  # Обрабатываем исктючения
    cur.executescript(sql)  # Выполняем SQL-запросы
except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
#     input()
```

```
Запрос успешно выполнен
```

In [7]:
```
ls
```

```
 ’®¬ ў гбва®©бвўЃ С Ѓ Ё¬ЃЃв ¬ЃвЄЁ.
 ‘ЃаЁ©л© ®¬Ґа в®¬ : D65B-CBD9

‘ЁбвЃ¬Ѓ Ѓ г¤ Ґвбп  ©вЁ гЄ § л© Їгвм.


 ‘®¤Ґа¦Ё¬®Ѓ Ї ЇЄЁ C:\Users\‘ЃаJЃ©\YandexDisk\Python\Ipynb\TOBD_2021\06_database\lec

11.10.2021  09:07    <DIR>          .
11.10.2021  09:07    <DIR>          ..
10.10.2021  22:34    <DIR>          .ipynb_checkpoints
07.10.2021  10:58         2я214я400 648989.ppt
07.10.2021  10:58            36я864 catalog.db
07.10.2021  10:58    <DIR>          img
07.10.2021  10:58             2я745 lec_v1.css
07.10.2021  10:58    <DIR>          materials
07.10.2021  10:58                 0 testdb.db
07.10.2021  10:58            75я815 TOBD_lec_06_database_v2..ipynb
11.10.2021  09:07            75я808 TOBD_lec_06_database_v2.ipynb
10.10.2021  22:34         1я796я477 TOBD_lec_06_database_v2.pdf
07.10.2021  10:58             3я070 Untitled.ipynb
07.10.2021  10:58         2я226я176 ,ўЃ¤ЃЁЃ ў п§лЄ SQL.ppt
               9 д ©«®ў      6я431я355 Ў ©в
               5 Ї Ї®Є  12я616я486я912 Ў ©в бў®Ў®¤®
```

Добавление записи в таблицу

В этом примере мы использовали метод `commit()` объекта соединения.

- Метод `commit()` позволяет **завершить транзакцию**, которая запускается автоматически.
- Если метод **не вызвать** и при этом закрыть соединение с базой данных, то все произведенные изменения **будут автоматически отменены**.
- Запросы, изменяющие записи (INSERT, REPLACE, UPDATE, DELETE), **нужно завершать** вызовом метода `commit()` .

In [79]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
cur = con.cursor() #  Создаем объект-курсор

# SQL выражение
sql  = """
INSERT INTO GENRE (NAME)
VALUES ('Jazz')"""
```

In [80]:
```
try:  # Обрабатываем исктючения
    cur.execute(sql)  # Выполняем SQL-запросы
except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
#     input()
```

```
Запрос успешно выполнен
```

Часто в SQL-запрос необходимо подставлять данные, полученные от пользователя.

- **!!!** Если данные от пользователя не обработать и подставить в SQL-запрос, то пользователь получает возможность видоизменить запрос и, например, зайти в закрытый раздел без ввода пароля. Такой вид атаки называется **SQL injection**.
- Чтобы значения были правильно подставлены, нужно их передавать в виде кортежа или словаря во втором параметре метода `execute()`. В этом случае в SQL-запросе указываются следующие специальные заполнители:
  - `?` - при указании значения в виде кортежа;
  - `:<Ключ>` - при указании значения в виде словаря.

In [81]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
cur = con.cursor() #  Создаем объект-курсор
# SQL выражение
```

In [82]:
```
# подстановка позиционного параметра:

sql_1 = '''
INSERT INTO Artist (Name)
VALUES (?)'''
v1 = ('AC/DC',)
v2 = ('Aerosmith',)
```

In [83]:
```
# подстановка позиционного параметра (альтернативный синтаксис):

sql_2 = '''
INSERT INTO Artist (Name)
VALUES ({})'''
v3 = "'Alanis Morissette'"
```

In [84]:
```
# подстановка нескольких позиционных параметров:

sql_3 = '''
INSERT INTO Artist (Name, Comment)
VALUES (?, ?)'''
v4 = ('Alanis Morissette', 'My test comment')
v5 = ('Apocalyptica', 'It is my favorite artist')
```

In [85]:
```
# подстановка по ключу:

sql_4 = '''
INSERT INTO Artist (Name, Comment)
VALUES (:n, :com)'''
v6 = {'n': "Guns N' Roses", 'com': 'Funny guys'}
v7 = {'n': 'Iron Maiden', 'com': """Test symbols v " v ' v """}
```

In [86]:
```
try:
    cur.execute(sql_1, v1)
    cur.execute(sql_1, v2)

    # Опасный способ, если пользователь может передать значение:
    cur.execute(sql_2.format(v3))

    cur.execute(sql_3, v4)
    cur.execute(sql_3, v5)

    cur.execute(sql_4, v6)
    cur.execute(sql_4, v7) # тестирование экранирования символов

except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
Запрос успешно выполнен
```

`executemany(< SQL-запрос>, < Последовательность>)` - выполняет SQL-запрос несколько раз, при этом подставляя значения из последовательности.

- Каждый элемент последовательности должен быть кортежем (при использовании заполнителя `?`) или словарем (при использовании заполнителя `:<Ключ>`).
- Вместо последовательности можно указать объект-итератор или объект-генератор.
- Если в процессе выполнения запроса возникает ошибка, то метод возбуждает исключение;

In [87]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
cur = con.cursor() #  Создаем объект-курсор
# SQL выражение

sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val_list = [('The Clash',), ('The Cult',), ('The Doors',)]

try:
    cur.executemany(sql, val_list)
except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
Запрос успешно выполнен
```

Модуль `sqlite3` содержит также методы `execute()` , `executernany()` и `executescript()` объекта соединения, которые позволяют выполнить запрос без создания объекта-курсора.

In [47]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
# cur = con.cursor() #  НЕ создаем объект-курсор
# SQL выражение

sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val_list = [('The Police',), ('The Rolling Stones',)]

try:
    con.executemany(sql, val_list)
except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    # cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
Запрос успешно выполнен
```

Объект-курсор поддерживает несколько атрибутов:

- `lastrowid` - индекс последней добавленной записи с помощью инструкции `INSERT` и метода `execute()`. Если индекс не определен, то атрибут будет содержать значение `None`.

В качестве примера добавим новую рубрику и выведем ее индекс:

- `rowcount` - количество измененных или удаленных записей. Если количество не определено, то атрибут имеет значение `-1`;
- `description` - содержит кортеж кортежей с именами полей в результате выполнения инструкции `SELECT`. Каждый внутренний кортеж состоит из семи элементов. Первый элемент содержит название поля, а остальные элементы всегда имеют значение `None`.

In [89]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
# cur = con.cursor() #  НЕ создаем объект-курсор
# SQL выражение

sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val_list = [('The Police',), ('The Rolling Stones',)]
val = ('The Tea Party',)
```

In [90]:
```
try:
    res = con.executemany(sql, val_list)
    print('rowcount:', res.rowcount)
    print('descriptiont:', res.description)

    res = con.execute(sql, val)
    print('lastrowid:', res.lastrowid)

except sqlite3.DatabaseError as  err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    # cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
rowcount: 2
descriptiont: None
lastrowid: 67
Запрос успешно выполнен
```

Обработка результата запроса

`fetchone()` - при каждом вызове возвращает одну запись из результата запроса в виде кортежа, а затем перемещает указатель текущей позиции.

- Если записей больше нет, метод возвращает значение `None`.

In [91]:
```
# import  sqlite3
con  =  sqlite3.connect("catalog.db")
cur  =  con.cursor()
cur.execute('SELECT * FROM Artist')

fo = cur.fetchone()

print(fo)
```

```
(1, 'AC/DC', None)
```

`__next__()` - при каждом вызове возвращает одну запись из результата запроса в виде кортежа, а затем перемещает указатель текущей позиции. Если записей больше нет, метод возбуждает исключение `StopIteration`. Выведем все записи из таблицы user с помощью метода `next()`.

In [70]:
```
# import  sqlite3
con  =  sqlite3.connect("catalog.db")
cur  =  con.cursor()
cur.execute('SELECT * FROM Artist')

print(next(cur))
print(next(cur))
print(next(cur))
```

```
(1, 'AC/DC', None)
(2, 'Aerosmith', None)
(3, 'Alanis Morissette', None)
```

Цикл for на каждой итерации вызывает метод __next__() автоматически. Поэтому для перебора записей достаточно указать объект-курсор в качестве параметра цикла.

In [92]:
```
# import  sqlite3
con  =  sqlite3.connect("catalog.db")
cur  =  con.cursor()
cur.execute('SELECT * FROM Artist')

for v in cur:
#     fh = cur.fetchone()
    print(v)
```

```
(1, 'AC/DC', None)
(2, 'Aerosmith', None)
(3, 'Alanis Morissette', None)
(4, 'Alanis Morissette', 'My test comment')
(5, 'Apocalyptica', 'It is my favorite artist')
(6, "Guns N' Roses", 'Funny guys')
(7, 'Iron Maiden', 'Test symbols v " v \' v ')
(8, 'The Clash', None)
(9, 'The Cult', None)
(10, 'The Doors', None)
(11, 'The Police', None)
(12, 'The Rolling Stones', None)
(13, 'The Police', None)
(14, 'The Rolling Stones', None)
(15, 'The Tea Party', None)
(16, 'New value two', None)
(17, 'AC/DC', None)
(18, 'Aerosmith', None)
(19, 'Alanis Morissette', None)
(20, 'Alanis Morissette', 'My test comment')
(21, 'Apocalyptica', 'It is my favorite artist')
(22, "Guns N' Roses", 'Funny guys')
(23, 'Iron Maiden', 'Test symbols v " v \' v ')
(24, 'The Clash', None)
(25, 'The Cult', None)
(26, 'The Doors', None)
(27, 'The Police', None)
(28, 'The Rolling Stones', None)
(29, 'The Tea Party', None)
(30, 'New value two', None)
(31, None, None)
(32, 'AC/DC', None)
(33, 'Aerosmith', None)
(34, 'Alanis Morissette', None)
(35, 'Alanis Morissette', 'My test comment')
(36, 'Apocalyptica', 'It is my favorite artist')
(37, "Guns N' Roses", 'Funny guys')
(38, 'Iron Maiden', 'Test symbols v " v \' v ')
(39, 'The Clash', None)
(40, 'The Cult', None)
(41, 'The Doors', None)
(42, 'The Police', None)
(43, 'The Rolling Stones', None)
(44, 'The Police', None)
(45, 'The Rolling Stones', None)
(46, 'The Tea Party', None)
(47, 'The Police', None)
(48, 'The Rolling Stones', None)
(49, 'The Tea Party', None)
(50, 'New value two', None)
(51, 'The Police', None)
(52, 'The Rolling Stones', None)
(53, 'The Tea Party', None)
(54, None, None)
(55, 'AC/DC', None)
```

```
(56, 'Aerosmith', None)
(57, 'Alanis Morissette', None)
(58, 'Alanis Morissette', 'My test comment')
(59, 'Apocalyptica', 'It is my favorite artist')
(60, "Guns N' Roses", 'Funny guys')
(61, 'Iron Maiden', 'Test symbols v " v \' v ')
(62, 'The Clash', None)
(63, 'The Cult', None)
(64, 'The Doors', None)
(65, 'The Police', None)
(66, 'The Rolling Stones', None)
(67, 'The Tea Party', None)
```

In [93]:

```
# import  sqlite3
con  =  sqlite3.connect("catalog.db")
cur  =  con.cursor()
cur.execute('SELECT * FROM Artist')

for rid, name, comment in cur:
#     fh = cur.fetchone()
    print('rid: {:2}, name: {:25}, comment: {}'.format(rid, name, comment))
```

```
rid:  1, name: AC/DC                     , comment: None
rid:  2, name: Aerosmith                 , comment: None
rid:  3, name: Alanis Morissette         , comment: None
rid:  4, name: Alanis Morissette         , comment: My test comment
rid:  5, name: Apocalyptica              , comment: It is my favorite artist
rid:  6, name: Guns N' Roses             , comment: Funny guys
rid:  7, name: Iron Maiden               , comment: Test symbols v " v ' v
rid:  8, name: The Clash                 , comment: None
rid:  9, name: The Cult                  , comment: None
rid: 10, name: The Doors                 , comment: None
rid: 11, name: The Police                , comment: None
rid: 12, name: The Rolling Stones        , comment: None
rid: 13, name: The Police                , comment: None
rid: 14, name: The Rolling Stones        , comment: None
rid: 15, name: The Tea Party             , comment: None
rid: 16, name: New value two             , comment: None
rid: 17, name: AC/DC                     , comment: None
rid: 18, name: Aerosmith                 , comment: None
rid: 19, name: Alanis Morissette         , comment: None
rid: 20, name: Alanis Morissette         , comment: My test comment
rid: 21, name: Apocalyptica              , comment: It is my favorite artist
rid: 22, name: Guns N' Roses             , comment: Funny guys
rid: 23, name: Iron Maiden               , comment: Test symbols v " v ' v
rid: 24, name: The Clash                 , comment: None
rid: 25, name: The Cult                  , comment: None
rid: 26, name: The Doors                 , comment: None
rid: 27, name: The Police                , comment: None
rid: 28, name: The Rolling Stones        , comment: None
rid: 29, name: The Tea Party             , comment: None
rid: 30, name: New value two             , comment: None

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-93-be6f94427f91> in <module>
      6 for rid, name, comment in cur:
      7 #     fh = cur.fetchone()
----> 8     print('rid: {:2}, name: {:25}, comment: {}'.format(rid, name, comment))

TypeError: unsupported format string passed to NoneType.__format__
```

`fetchmany([size=cursor.arraysize])` - при каждом вызове возвращает список записей из

результата запроса, а затем перемещает указатель текущей позиции.

- Каждый элемент списка является кортежем.
- Количество элементов, выбираемых за один раз, задается с помощью необязательного параметра или значения атрибута arraysize объекта-курсора.
- Если количество записей в результате запроса меньше указанного количества элементов списка, то количество элементов списка будет соответствовать оставшемуся количеству записей.
- Если записей больше нет, метод возвращает пустой список.

`fetchall()` - возвращает список всех (или всех оставшихся) записей из результата запроса.

- Каждый элемент списка является кортежем.
- Если записей больше нет, то метод возвращает пустой список.

In [56]:
```
# import  sqlite3
con  =  sqlite3.connect("catalog.db")
cur  =  con.cursor()
cur.execute('SELECT * FROM Artist')

print('cur.arraysize: ',cur.arraysize)
fm = cur.fetchmany(10)
print(fm)

print()
fa = cur.fetchall()
print(fa)
```

```
cur.arraysize:  1
[(1, 'AC/DC', None), (2, 'Aerosmith', None), (3, 'Alanis Morissette', None), (4, 'Alan
is Morissette', 'My test comment'), (5, 'Apocalyptica', 'It is my favorite artist'),
(6, "Guns N' Roses", 'Funny guys'), (7, 'Iron Maiden', 'Test symbols v " v \' v '),
(8, 'The Clash', None), (9, 'The Cult', None), (10, 'The Doors', None)]

[(11, 'The Police', None), (12, 'The Rolling Stones', None), (13, 'The Police', None),
(14, 'The Rolling Stones', None), (15, 'The Tea Party', None), (16, 'New value two', N
one), (17, 'AC/DC', None), (18, 'Aerosmith', None), (19, 'Alanis Morissette', None),
(20, 'Alanis Morissette', 'My test comment'), (21, 'Apocalyptica', 'It is my favorite
artist'), (22, "Guns N' Roses", 'Funny guys'), (23, 'Iron Maiden', 'Test symbols v " v
\' v '), (24, 'The Clash', None), (25, 'The Cult', None), (26, 'The Doors', None), (2
7, 'The Police', None), (28, 'The Rolling Stones', None), (29, 'The Tea Party', None),
(30, 'New value two', None), (31, None, None), (32, 'AC/DC', None), (33, 'Aerosmith',
None), (34, 'Alanis Morissette', None), (35, 'Alanis Morissette', 'My test comment'),
(36, 'Apocalyptica', 'It is my favorite artist'), (37, "Guns N' Roses", 'Funny guys'),
(38, 'Iron Maiden', 'Test symbols v " v \' v '), (39, 'The Clash', None), (40, 'The Cu
lt', None), (41, 'The Doors', None), (42, 'The Police', None), (43, 'The Rolling Stone
s', None), (44, 'The Police', None), (45, 'The Rolling Stones', None), (46, 'The Tea P
arty', None), (47, 'The Police', None), (48, 'The Rolling Stones', None), (49, 'The Te
a Party', None)]
```

Управление транзакциями

Перед выполнением первого запроса автоматически запускается транзакция. Поэтому все запросы, изменяющие записи (INSERT, REPIACE, UPDATE и DELETE), необходимо завершать вызовом метода `commit()` объекта соединения.

- Если метод не вызвать и при этом закрыть соединение с базой данных, то все произведенные изменения будут отменены.

- Транзакция может автоматически завершаться при выполнении запросов CREATE TABLE, VACUUM и некоторых других. После выполнения этих запросов транзакция запускается снова.
- Если необходимо отменить изменения, то следует вызвать метод `rollback()` объекта соединения.

In [94]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
cur = con.cursor() #  НЕ создаем объект-курсор
# SQL выражение

sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val = ('New value',)

try:
    cur.execute(sql, val)
    con.rollback()
    cur.execute('SELECT * FROM Artist WHERE Name = ?', val)
    print('Результат вставки', cur.fetchall())

except sqlite3.DatabaseError as err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
    con.commit() # Завершаем транзакцию
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
Результат вставки []
Запрос успешно выполнен
```

Управлять транзакцией можно с помощью параметра `isolation _level` в функции `connect()` , а также с помощью атрибута `isolation _level` объекта соединения. Допустимые значения:

- "DEFERRED"
- "IMMEDIATE"
- "EXCLUSIVE"
- пустая строка
- `None` .

Первые три значения передаются в инструкцию BEGIN. Если в качестве значения указать `None` , то транзакция запускаться не будет. В этом случае нет необходимости вызывать метод `commit()` . Все изменения будут сразу сохраняться в базе данных.

In [58]:
```
sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val = ('New value two',)

con = sqlite3.connect("catalog.db",  isolation_level=None)
# Другой способ установки уровня изоляции:
# con.isolation_level = None # Отключение запуска транзакции
print('con.isolation_level: ', con.isolation_level)
cur = con.cursor()

cur.execute (sql, val)
con.close()

con  =  sqlite3.connect("catalog.db")
cur = con.cursor()

cur.execute('SELECT * FROM Artist WHERE Name = ?', val)

print(cur.fetchall())
con.close()
```

```
con.isolation_level:  None
[(16, 'New value two', None), (30, 'New value two', None), (50, 'New value two', Non
e)]
```

Обработка исключений

Модуль sqlites поддерживает следующую иерархию исключений:

- Exception
  - Warning
  - Error
    - InterfaceError
    - DatabaseError
      - DataError
      - OperationalError
      - IntegrityError
      - InternalError
      - ProgrammingError
      - NotSupportedError

Базовым классом самого верхнего уровня является класс `Exception` . Все остальные исключения определены в модуле sqlite3. Поэтому при указании исключения except следует предварительно указать названИе модуля sqlite3. DatabaseError).

Исключения возбуждаются в следующих случаях:

- warning - при наличии важных предупреждений; в инструкции (например,
- Error - базовый класс для всех остальных исключений, возбуждаемых в случае ошибки. Если указать этот класс в инструкции except, то будут перехватываться все ошибки:
- InterfaceError - при ошибках, которые связаны с интерфейсом базы данных, а не с самой базой данных;
- DatabaseError - базовый класс для исключений, которые связаны с базой данных;
- DataError - при ошибках, возникающих при обработке данных;
- OperationalError - вызывается при ошибках, которые связаны с операциями в базе данных, например, при синтаксической ошибке в SQL-запросе, несоответствии количества полей в

инструкции INSERT, отсутствии поля с указанным именем и т. д. Иногда не зависит от правильиости SQL-запроса;
- IntegrityError - при наличии проблем с внешними ключами или индексами;
- InternalError - при внутренней ошибке в базе данных;
- ProgrammingError - возникает при ошибках программирования. Например, количество переменных, указанных во втором параметре метода execute(), не совпадает с количеством специальных символов в SQL-запросе;
- NotSupportedError - при использовании методов, не поддерживаемых базой данных.

In [ ]:

Язык Python поддерживает протокол менеджеров контекста with. Этот протокол гарантирует выполнение завершающих действий вне зависимости от того, произошло исключение внутри блока кода или нет. В модуле sqlite3 объект соединения поддерживает этот протокол. Если внутри блока with не произошло исключение, то автоматически вызывается метод `commit()` . В противном случае все изменения отменяются с помощью метода `rollback()` .

In [23]:
```
# import  sqlite3
con = sqlite3.connect("catalog.db")
cur = con.cursor() #  НЕ создаем объект-курсор
# SQL выражение

sql = '''
INSERT INTO Artist (Name)
VALUES (?)'''

val = ('New value 3',)

try:
    with con:
        con.execute(sql, val)

except sqlite3.DatabaseError as err:
    print("Ошибка:", err)
else:
    print ("Запрос успешно выполнен")
#     con.commit() # Завершаем транзакцию
    cur.close() # Закрываем объект-курсор
    con.close() # Закрываем соединение
```

```
Запрос успешно выполнен
```

Нормальные формы

-

- к оглавлению

Устранение дублирования информации важно по двум причинам: − устранив дублирование, можно добиться существенной экономии памяти; − если некоторое значение поля повторяется несколько раз, то при корректировке данных необходимо менять содержимое всех этих полей, в противном случае нарушится целостность данных.

Нормализация отношений – формальный аппарат ограничений на формирование отношений (таблиц), который позволяет устранить дублирование, обеспечивает непротиворечивость хранимых в базе данных, уменьшает трудозатраты на ведение (ввод, корректировку) базы данных.

При неправильно спроектированной схеме реляционной БД могут возникнуть аномалии выполнения операций модификации данных.

Аномалия обновления может возникнуть в том случае, когда информация дублируется. Другие аномалии возникают тогда, когда две и более сущности объединены в одно отношение. Например:

- Аномалия обновления: изменился адрес поставщика. Если от него было несколько поставок, то придется менять несколько записей.
- Аномалия удаления: при удалении в архив записей обо всех поставках определённого поставщика все данные об этом поставщике (название, адрес) будут утеряны.
- Аномалия добавления: нельзя добавить сведения о поставщике, пока от него нет ни одной поставки. Для решения проблемы аномалии модификации данных при проектировании РБД проводится нормализация отношений.

Первая нормальная форма

ID	Code	Theme	Author	Title	Editor	Type	Year	Pg
20	22.18	МК	Бочков С. Субботин Д.	Язык програм-мирования СИ	Садчиков П. Седов П.	учебник	1990	384
10	22.18	МК	Джехани Н.	Язык АДА	Красилов А. Перминов О.	учебник	1988	552
35	32.97	ВТ	Соловьев Г. Никитин В.	Операционные системы ЭВМ		учебное пособие	1992	208
11	32.81	Кибер-нетика	Попов Э.В.	Общение с ЭВМ на есте-ственном языке	Некрасов А.	учебник	1982	360
44	32.97	ВТ		ПУ для ПЭВМ	Витенберг Э.	спра-вочник	1992	208
89	32.973	ЭВМ	Коутс Р. Влейминк И.	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501

Пример содержимого таблицы "Книги" в ненормализованном виде

Первая нормальная форма

Введём понятие простого и сложного атрибута:

- **Простой атрибут** – это атрибут, значения которого атомарны (т.е. неделимы). Неделимость поля означает, что содержащиеся в нем значения не должны делиться на более мелкие или представлять из себя повторяющиеся группы.
- **Сложный атрибут** может иметь значение, представляющее собой конкатенацию нескольких значений одного или разных доменов.

Первая нормальная форма (1НФ) - отношение приведено к 1НФ, если все его атрибуты простые (каждый его кортеж содержит только одно значение для каждого из атрибутов).

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ.

Для нахождения в 1НФ таблица должна удовлетворять следующим пяти условиям:

1. Нет упорядочивания строк сверху вниз (другими словами, порядок строк не несет в себе никакой информации).
2. Нет упорядочивания столбцов слева направо (другими словами, порядок столбцов не несет в себе никакой информации).
3. Нет повторяющихся строк.

4. Каждое пересечение строки и столбца содержит ровно одно значение из соответствующего домена (и больше ничего).
    - что здесь **домен**?
5. Все столбцы являются обычными

ID	*Code*	*Theme*	***Author***	*Title*	***Editor***	*Type*	*Year*	*Pg*
20	22.18	МК	Бочков С.	Язык програм-мирования СИ	Садчиков П.	учебник	1990	384
20	22.18	МК	Субботин Д.	Язык програм-мирования СИ	Седов П.	учебник	1990	384
10	22.18	МК	Джехани Н.	Язык АДА	Красилов А.	учебник	1988	552
10	22.18	МК	Джехани Н.	Язык АДА	Перминов О.	учебник	1988	552
35	32.97	ВТ	Соловьев Г.	Операционные системы ЭВМ		учебное пособие	1992	208
35	32.97	ВТ	Никитин В.	Операционные системы ЭВМ		учебное пособие	1992	208
11	32.81	Кибер-нетика	Попов Э.В.	Общение с ЭВМ на есте-ственном языке	Некрасов А.	учебник	1982	360
44	32.97	ВТ		ПУ для ПЭВМ	Витенберг Э.	спра-вочник	1992	208
89	32.973	ЭВМ	Коутс Р.	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501
89	32.973	ЭВМ	Влейминк И.	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501

Приведение таблицы "Книги" к 1НФ

Вторая нормальная форма

Отношение находится во 2НФ, если:

- оно приведено к 1НФ
- каждый неключевой атрибут **функционально полно зависит** от составного ключа.
- Понятие **функциональной зависимости**:
    - Пусть X и Y – атрибуты (группы атрибутов) некоторого отношения.
    - Говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X=x соответствует единственное значение Y=y: X->Y (икс определяет зет).
    - При этом любому значению Y=y может соответствовать несколько значений X=(x1, x2,…)).

Смысл определения в том, что если Y функционально зависит от X, то каждый из кортежей, имеющих одно и то же значение X, должен иметь также одно и то же значение Y. Значения X и Y могут изменяться время от времени, но при этом должны изменяться так, чтобы каждое уникальное значение X имело только одно уникальное значение Y, связанное с ним.

- **Полная функционально зависимость** означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.
    - Если потенциальный ключ является **простым**, то есть состоит из единственного атрибута, то любая функциональная зависимость от него является полной.
    - Если потенциальный ключ является **составным**, то, согласно определению 2НФ, в отношении **не должно быть неключевых атрибутов, зависящих от части составного потенциального ключа**.
- Вторая нормальная форма запрещает наличие неключевых атрибутов, которые вообще не зависят от потенциального ключа. Т.е. **запрещается создавать отношения как несвязанные (хаотические, случайные) наборы атрибутов**.

Приведение ко 2НФ:

- построить проекцию, исключив атрибуты, которые не находятся в функционально полной зависимости от составного ключа;
- построить дополнительные проекции на часть составного ключа и атрибуты, функционально зависящие от этой части ключа.

ID	Code	Theme	Title	Type	Year	Pg
20	22.18	МК	Язык программирования СИ	учебник	1990	384
10	22.18	МК	Язык АДА	учебник	1988	552
35	32.97	ВТ	Операционные системы ЭВМ	учебное пособие	1992	208
11	32.81	Кибернетика	Общение с ЭВМ на естественном языке	учебник	1982	360
44	32.97	ВТ	ПУ для ПЭВМ	справочник	1992	208
89	32.973	ЭВМ	Интерфейс «человек-компьютер»	учебник	1990	501

Приведение таблицы "Книги" к 2НФ: отношение "Книги"

ID	Author	Editor
20	Бочков С.	Садчиков П.
20	Субботин Д.	Седов П.
10	Джехани Н.	Красилов А.
10		Перминов О.
35	Соловьев Г.	
35	Никитин В.	
11	Попов Э.В.	Некрасов А.
44		Витенберг Э.
89	Коутс Р.	Шаньгин В.
89	Влейминк И.	

Приведение таблицы "Книги" к 2НФ: отношение "Книги-Авторы-Редакторы"

Частичные функциональные зависимости (отсутствие приведения к 2НФ) приводят к **аномалиям**:

- избыточнму дублированию данных
  - Например заголовок книги повторяется много раз для книг с несколькими авторами.
- проблеме контроля правильности данных в случае их изменения
  - Например изменение кода одной книги потребует внесения изменений в нескольких кортежах.
- проблеме добавления: нельзя добавить сведения в случае отсутсвия связанных с ним факта.
  - Например, нельзя добавить данные о поставщике, пока от него нет ни одной поставки.
- проблема удаления записей: удаление кортежа может привести к потере сведений, относящихся не только к этому кортежу
  - Удаление записи о книге может привести к удаелнию информации о наименовании темы.

Третья нормальная форма

Функциональная зависимость множества атрибутов Z от множества атрибутов X **является транзитивной**, если существует такое множество атрибутов Y, что:

- X->Y и Y->Z
- при этом ни одно из множеств X, Y и Z не является подмножеством другого, то есть функциональные зависимости X->Z, X->Y и Y->Z не являются тривиальными
- отсутствует функциональная зависимость Y->X.

Третья нормальная форма (3НФ): отношение находится в 3НФ, если:

- оно находится во 2НФ
- в нем отсутствуют транзитивные зависимости неключевых атрибутов от ключа.

Исключение: если для атрибутов X,Y,Z есть транзитивная зависимость X->->Z, и при этом Y->X или Z->Y, то такая зависимость не требует декомпозиции отношения.

Например : для отношения АВТОМОБИЛИ с первичным ключом Государственный номерной знак и полями № кузова и № двигателя очевидно, что номера кузова и двигателя зависят как друг от друга, так и от первичного ключа. Но эта зависимость взаимно однозначная, поэтому декомпозиция отношения не нужна.

ID	*Code*	*Title*	*Type*	*Year*	*Pg*
20	22.18	Язык программирования СИ	учебник	1990	384
10	22.18	Язык АДА	учебник	1988	552
35	32.97	Операционные системы ЭВМ	учебное пособие	1992	208
11	32.81	Общение с ЭВМ на естественном языке	учебник	1982	360
44	32.97	ПУ для ПЭВМ	справочник	1992	208
89	32.973	Интерфейс «человек-компьютер»	учебник	1990	501

Отношения "Книги", приведенные к 3НФ

Code	*Theme*
22.18	МК
32.97	ВТ
32.973	ЭВМ
32.81	Кибернетика

Приведение к 3НФ отношения "Книги": введение отношения "Рубрикатор"

Чтобы устранить транзитивную зависимость и привести отношение в 3НФ, необходимо:

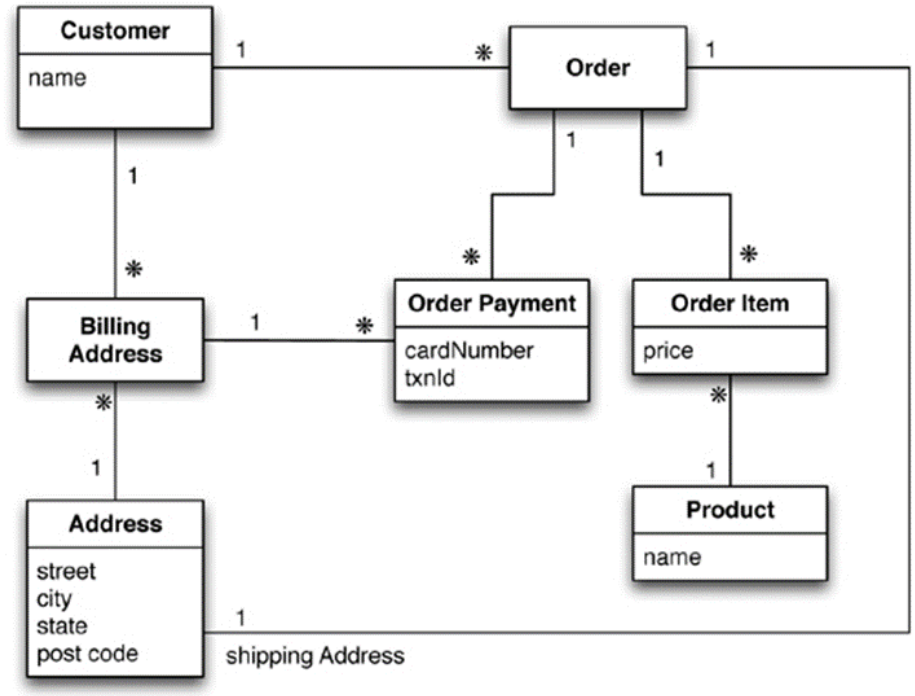
- построить проекцию без атрибутов, находящихся в транзитивной зависимости от ключа;
- построить отдельные проекции на неключевые атрибуты, между которыми имеется функциональная зависимость;
- получившиеся отношения с одинаковыми ключами соединить.

Наличие транзитивных зависимостей порождает **аномалии** следующего характера:

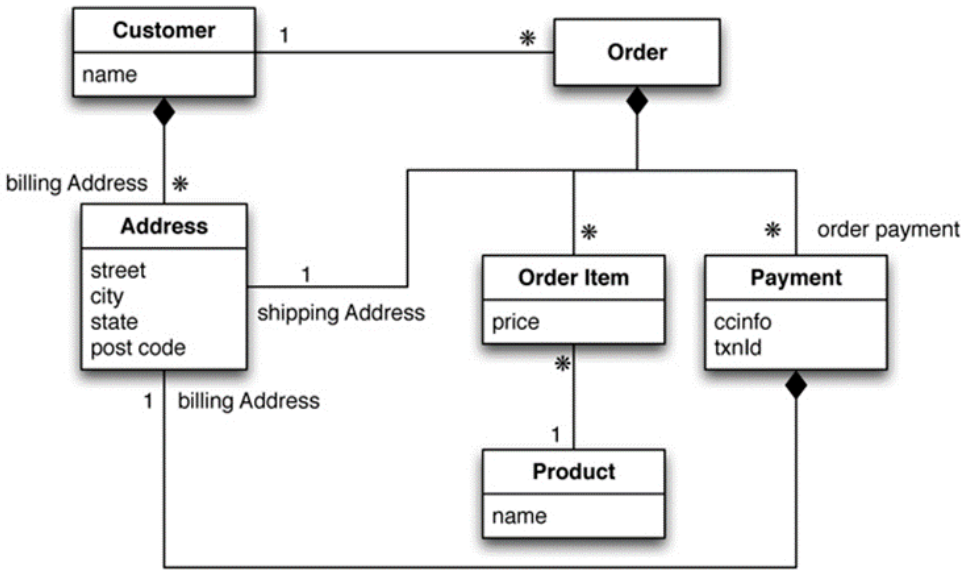
- избыточно дублирование информации
- сложное редактирование данных: изменение, требует поиска и замены его во всех кортежах где эти данные могут дублироваться;
- проблема удаления записей: удаление кортежа может привести к потере сведений о связанных категориях
- проблема добавления записей: невозможно дбавить сведения, если не добавлены связанные категории

Анализ применимости нормализации представления данных

Сравнение агрегированного и нормализованного представления:



Нормализованное представление



Агрегированное представление

Нормализация данных

- **Плюсы:**
  - **+** Целостность даннх при обновлении (изменяем данные в одной таблице, а не в нескольких)
  - **+** Ориентированность на широкий спектр запросов к данным
- **Минусы:**
  - **-** Низкая скорость чтения при использовании объединений (joins)
  - **-** Несоотвествие объектной модели приложения физической структуре данных
  - **-** Усложенение структуры БД при высокой степени нормализации
  - **-** Неэффективность в распределенной среде

Данные в виде агрегатов

- **Плюсы:**
  - **+** Высокая скорость чтения при отсутствии объединений (joins)
  - **+** Лучший способ добиться большой скорости на чтение в распределенной среде

    - **+** Возможность хранить физические объекты в том виде, в котором с ними работает приложение (легче кодировать и меньше ошибок при преобразовании)
- **Минусы:**
    - **-** Оптимизация только под определенный вид запросов.
    - **-** Сложность при обновлении денормализованных данных.

Организация работы Join

- Введение в язык SQL.ppt (слайды 20+)

TODO: Индексы

Подход NoSQL

- NoSQL — ряд подходов, к реализации хранилищ баз данных, имеющих существенные отличия от реляционных СУБД.
- NoSQL = Not ONLY SQL. Подход NoSQL не является отрицанием реляционного подхода (SQL), а рассматривает его как важный, но не универсальный инструмент.

Черты, присущие подходам NoSQL (к некоторым подходам в рамках NoSQL относятся не все свойства):

- Является большим хранилищем сериализованных объектов
    - Поиск информации по ID
- В общем случае сложные запросы к данным не поддерживаются
- Не имеют структурированной (а подчас и вообще какой-либо) схемы (нет реляционной модели)
- Ориентированы на работу с денормализованными данными
- Являются готовыми решениями для создания распределенных хранилищ данных (на основе кластеров) из-за этого не поддерживают требований ACID (Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Стойкость)
- Любой узел распределенного хранилища может отвечать на любой запрос
- Любое изменение (добавление) информации может выполнятся для любого узла хранилища и со временем распространится на другие узлы

Документо-ориентированные базы

Документо-ориентированные базы похожи на Key-Value базы, но отличаются тем, что БД знает, что из себя представляют значения. Обычно, значением является некоторый документ или объект, к структуре которого можно делать запросы.

Представители (см.: https://db-engines.com/en/ranking/document+store (https://db-engines.com/en/ranking/document+store)):

- MongoDB
- Couchbase
- Firebase.

MongoDB — документоориентированная СУБД с открытым исходным кодом, не требующая описания схемы таблиц, класса NoSQL, использует JSON-подобные документы и схему базы данных.

- Fast introduction: https://www.youtube.com/watch?v=EE8ZTQxa0AM (https://www.youtube.com/watch?v=EE8ZTQxa0AM)
- Система поддерживает ad-hoc-запросы: они могут возвращать конкретные поля документов.
- Поддерживается JavaScript в запросах, пользовательские JavaScript-функции в функциях агрегации (аналогах map-reduce)
- Поддерживается поиск по регулярным выражениям

- База данных MongoDB распространена в системах хранения и регистрация событий в системах управления документами и контентом.
- Система может работать с набором реплик, то есть, содержать две или более копии данных на различных узлах.
- Система масштабируется горизонтально, используя sharding объектов баз данных

- Краткое руководство по MongoDB: https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo (https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo)
- Tutorial MongoDB на Python: https://realpython.com/introduction-to-mongodb-and-python/ (https://realpython.com/introduction-to-mongodb-and-python/)
- https://www.w3schools.com/python/python_mongodb_getstarted.asp (https://www.w3schools.com/python/python_mongodb_getstarted.asp)