Оленчикова Т.Ю.

**Лабораторная работа 1.** Документоориентированная СУБД MongoDB

## Цель

1. Ознакомиться с нереляционной документоориентированной СУБД MongoDB
2. Получить навыки работы с БД из среды Python.
3. Получить навыки ETL трансформации данных при переносе их из одной БД в другую.

## Задание

**Внимание!** Работа выполняется с данными, загруженными в PostgreSQL БД, созданную на занятии ПК3, для вашей предметной области.

1. Проанализировать вашу предметную область и определить, какие данные Вы загружаете в MongoDB
2. В БД PostgreSQL написать запрос на выборку необходимых данных.
3. На Python написать программу:
4. создать подключение к PostgreSQL;
5. выполнить выборку необходимых данных, преобразование их в формат JSON.
6. создать подключение к СУБД MongoDB,
7. создать коллекцию документов, выполняя запросы к PostgreSQL
8. загрузить эти документы в MongoDB, добавить в коллекцию не менее четырех различных документов.
9. Вывести список документов в коллекции со всеми атрибутами, выполнив запрос к базе данных.
10. Вывести основные атрибуты части документов коллекции, удовлетворяющих некоторому условию (использовать условия «меньше», «больше»)

## Требования к оформлению отчета.

Отчет по ЛР состоит из

**а)** Титульный лист: название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания; в) UML-диаграмму классов (создаете в Umlet):

г) Исходный текст программы;

д) Скрины выполнения;

е) Выводы.

## Критерии оценивания.

1. Задание выполнено полностью. Созданы все необходимые модули – 1 балл; созданы не все модули – 0 баллов;

2. Приложение отлажено и работоспособно -2 балла; имеются ошибки или реализованы не все функции – 1 балл; приложение не рабоспособно – 0 баллов

3. Отчет оформлен в соответствии с требованиями – 1 балл, иначе – 0 баллов

4. Полные и верные ответы на контрольные вопросы – 1 балл, иначе – 0 баллов

Итого – максимум 5 баллов.

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%.

## Контрольные вопросы

1. Что является отличительной особенностью NoSQL?
2. В каком случае стоит применять NoSQL хранилища?
3. Что, согласно теореме САР, возможно обеспечить в любой реализации распределённых вычислений?
4. Какое свойство означает, что транзакции не нарушают согласованность данных, то есть они переводят базу данных из одного корректного состояния в другое?
5. Какой способ хранения данных используется в MongoDB?
6. Что относится к плюсам репликации?
7. Что относится к преимуществам нереляционных БД?
8. На какие три группы подразделяют пользователей в MongoDB?
9. Что такое шардинг?
10. Какие три свойства фигурируют в определении теоремы CAP?

## Приложение 1 Базы данных в Python: как подключить PostgreSQL

Во время разработки приложений часто нужно подключать и использовать базы данных для хранения информации. Самая распространенная база данных — PostgreSQL, поэтому мы расскажем, как работать в Python именно с ней. Для этого существует множество модулей, например:

* Psycopg2
* py-postgresql
* pg8000

Мы расскажем именно про модуль Psycopg2. И выбрали мы его по таким причинам:

* **Распространенность** — Psycopg2 использует большинство фреймворков Python
* **Поддержка** — Psycopg2 активно развивается и поддерживает основные версии Python
* **Многопоточность** — Psycopg2 позволяет нескольким потокам поддерживать одно и то же соединение

## Установка Psycopg2

Для начала работы с модулем достаточно установить пакет при помощи pip:

pip install psycopg2-binary

Если в вашем проекте используется poetry, то при первоначальной настройке проекта нужно добавить psycopg2-binary в зависимости. Для добавления в уже существующий проект воспользуйтесь командой:

poetry add psycopg2-binary

## Использование Psycopg2

## ****Подключение к БД:****

Для подключения к существующей базе данных необходимо знать основную информацию о вашей БД. Если вы не знаете, где ее взять, то пройдите сначала наш большой курс по [Основам баз данных](https://ru.hexlet.io/courses/rdb-basics?promo_name=courses&promo_position=article-body&promo_type=link):

* Username — имя пользователя, которое вы используете для работы с PostgreSQL
* Password — пароль, который используется пользователем
* Host Name — имя сервера или IP-адрес, на котором работает PostgreSQL
* Database Name — имя базы данных, к которой мы подключаемся.

Для подключения к базе данных мы используем метод connect(), которому в качестве аргументов передаются вышеперечисленные данные:

**import** psycopg2

**try**:

*# пытаемся подключиться к базе данных*

conn **=** psycopg2.connect(dbname**=**'test', user**=**'postgres', password**=**'secret', host**=**'host')

**except**:

*# в случае сбоя подключения будет выведено сообщение в STDOUT*

**print**('Can`t establish connection to database')

Также подключение к базе данных может осуществляться с помощью Connection URI:

**import** psycopg2

**try**:

*# пытаемся подключиться к базе данных*

conn **=** psycopg2.connect('postgresql://user:password@host:port/database\_name')

**except**:

*# в случае сбоя подключения будет выведено сообщение  в STDOUT*

**print**('Can`t establish connection to database')

## Взаимодействие Python с PostgreSQL

Итак, подключение к базе данных успешно выполнено. Дальше мы будем взаимодействовать с ней через объект cursor, который можно получить через метод cursor() объекта соединения. Он помогает выполнять SQL-запросы из Python.

*# получение объекта курсора*

cursor **=** conn.cursor()

С помощью cursor происходит передача запросов базе данных:

*# Получаем список всех пользователей*

cursor.execute('SELECT \* FROM users')

all\_users **=** cursor.fetchall()

cursor.close() *# закрываем курсор*

conn.close() *# закрываем соединение*

Для получения результата после выполнения запроса используются следующие команды:

* cursor.fetchone() — вернуть одну строку
* cursor.fetchall() — вернуть все строки
* cursor.fetchmany(size=10) — вернуть указанное количество строк

Хорошей практикой при работе с базой данных является закрытие объекта cursor и соединения с базой. Для автоматизации этого процесса удобно взаимодействовать через контекстный менеджер, используя конструкцию with :

**with** conn.cursor **as** curs:

curs.execute('SELECT \* FROM users')

all\_users **=** curs.fetchall()

В тот момент, когда объект cursor выходит за пределы конструкции with, происходит его закрытие и освобождение связанных с ним ресурсов.

По умолчанию результат возвращается в виде кортежа. Такое поведение возможно изменить, передав параметр cursor\_factory в момент открытия объекта cursor, например, использовать NamedTupleCursor. Это вернет данные в виде именованного кортежа:

**from** psycopg2.extras **import** NamedTupleCursor

*# …*

**with** conn.cursor(cursor\_factory**=**NamedTupleCursor) **as** curs:

curs.execute('SELECT \* FROM users WHERE name=%s', (name**=**'Alfred'))

alfred **=** curs.fetchone()

alfred *# (id=10, name='Alfred', age='90')*

*# …*

## Выполнение запросов

Psycopg2 преобразует переменные Python в SQL значения с учетом их типа. Все стандартные типы Python адаптированы для правильного представления в SQL.

Передача параметров в SQL-запрос происходит с помощью подстановки плейсхолдеров %s и цепочки значений в качестве второго аргумента функции:

**with** conn.cursor() **as** curs:

curs.execute('SELECT id, name FROM users WHERE name=%s', ('John',))

curs.fetchall()

*# …*

**with** conn.cursor() **as** curs:

curs.execute(INSERT INTO users (name, age) VALUES (**%**s, **%**s), ('John', 19))

*# …*

conn.close()

Подстановка значений в SQL-запрос используется для того, чтобы избежать атак типа SQL Injection. Также несколько полезных советов по построению запросов:

* Плейсхолдер должен быть %s даже если тип подставляемого значения отличается от строки
* Не заключайте плейсходер в кавычки
* Если в запросе используется знак %, он должен быть указан как %%

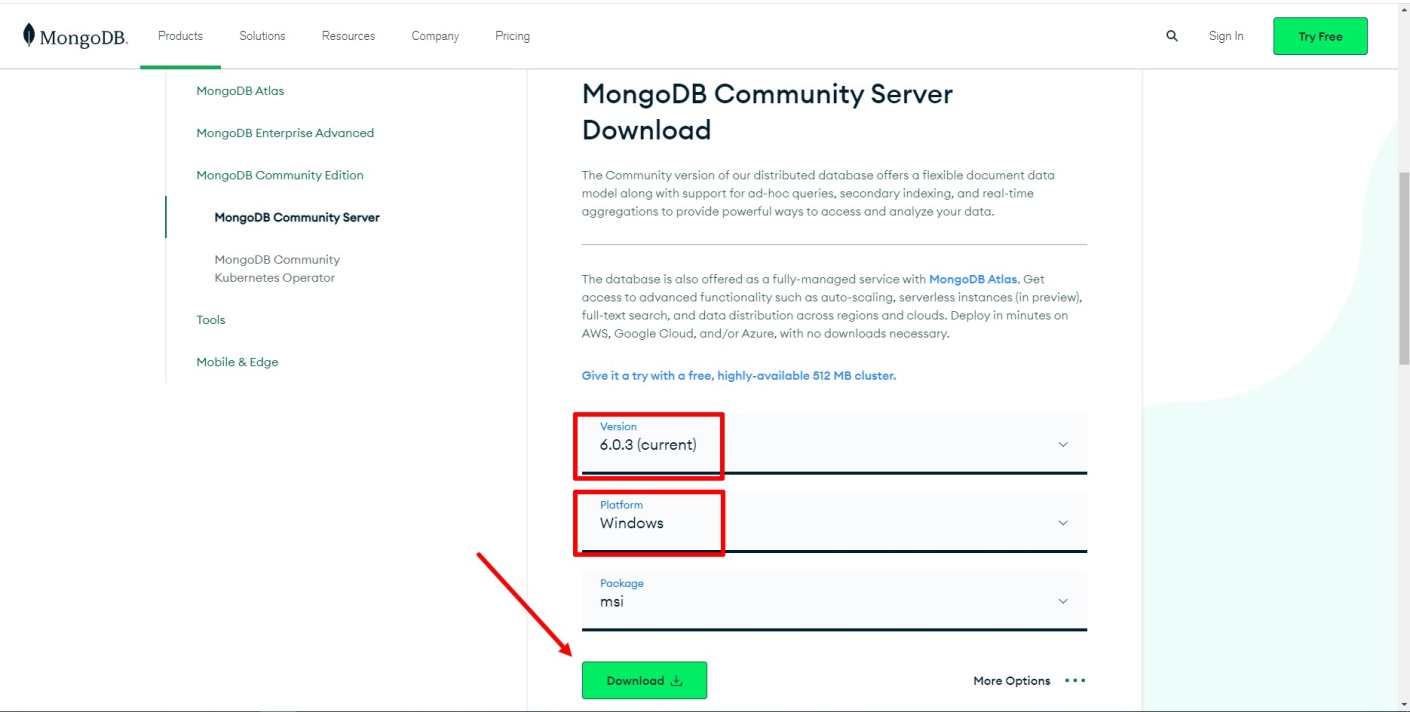
Использованы материалы с сайта <https://ru.hexlet.io/blog/posts/python-postgresql>

## Приложение 2 Базы данных в Python: как подключить MongoDB

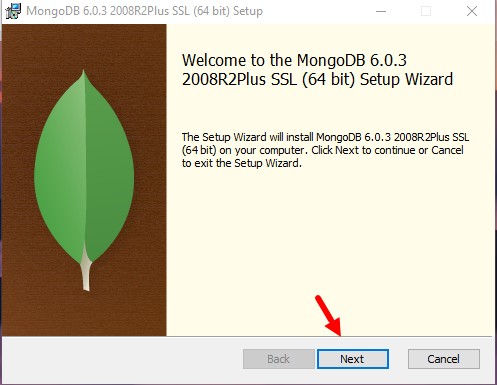
познакомимся с идеальной базой данных для не структурированной информации. То есть мы избавимся от строгого синтаксиса и сможем легко дополнять записи новой информацией. В этом огромный плюс NoSQL-базы для написания различных ботов, которые в дальнейшем будут масштабироваться.

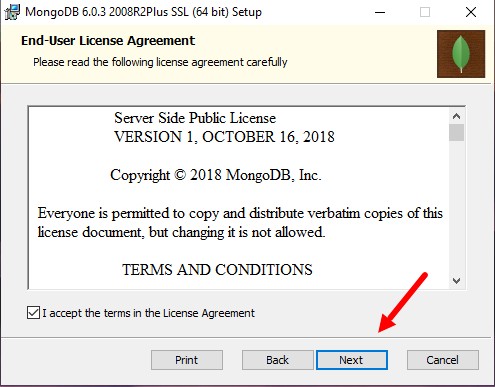
## Подготовка

Сначала установим саму MongoBD: переходим на официальный сайт Mongo по [**ссылке**](https://www.mongodb.com/try/download/community)и скачиваем сервер версии 6.0.3 под вашу операционную систему.

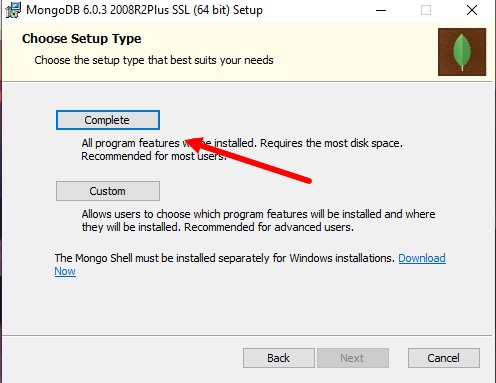


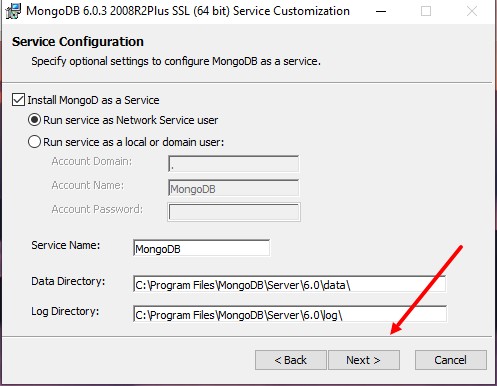
Для корректной установки следуйте примерам на скриншотах:



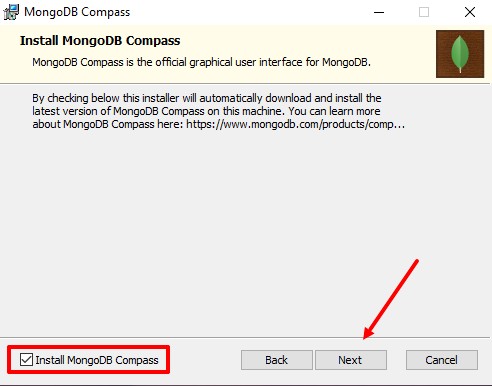


Важно тут выбрать пункт “Complete”. Этот пункт поставит все стандартные настройки и модули, что для начала нам подойдет. Потом, если будете изучать эту БД, сможете настраивать под себя:

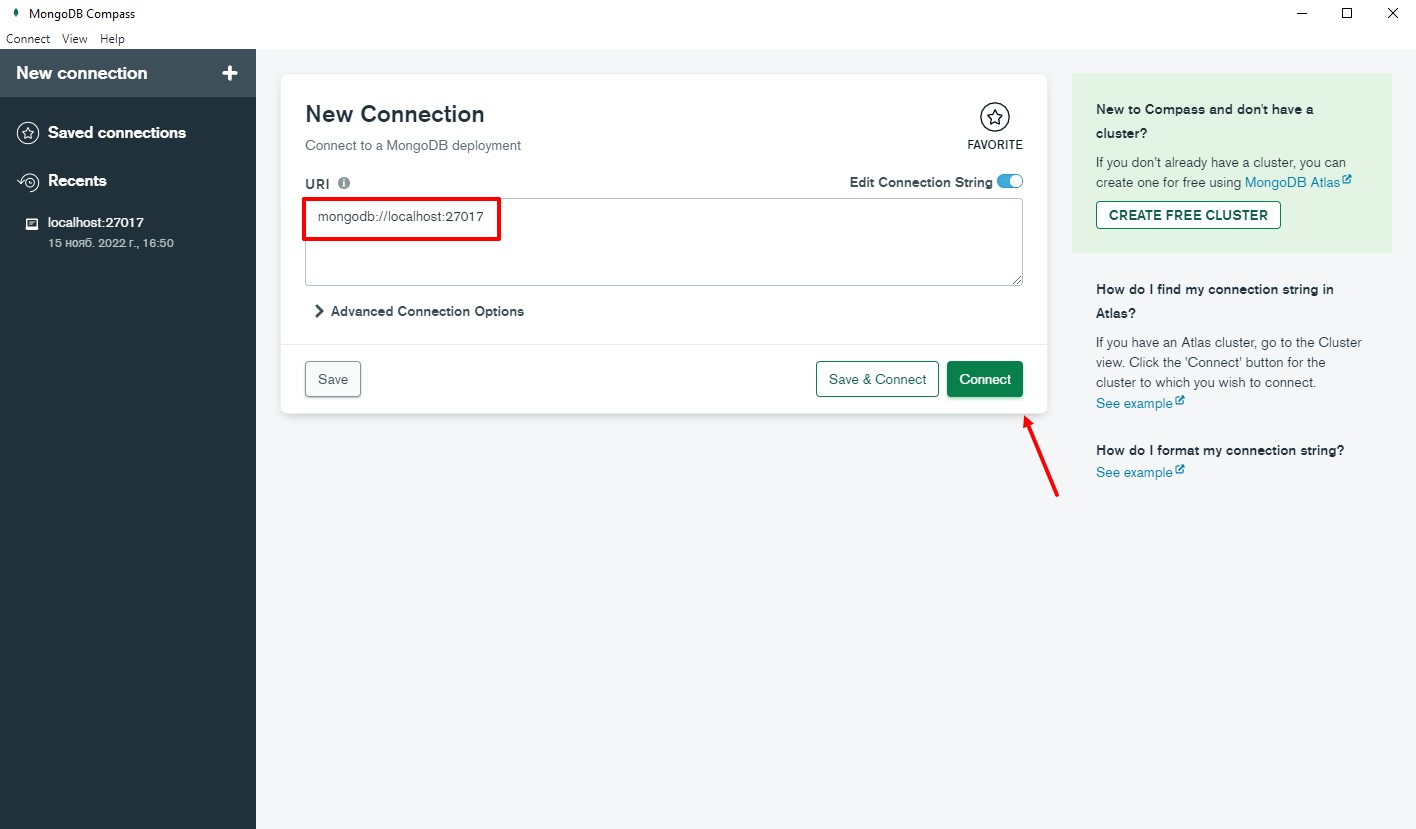




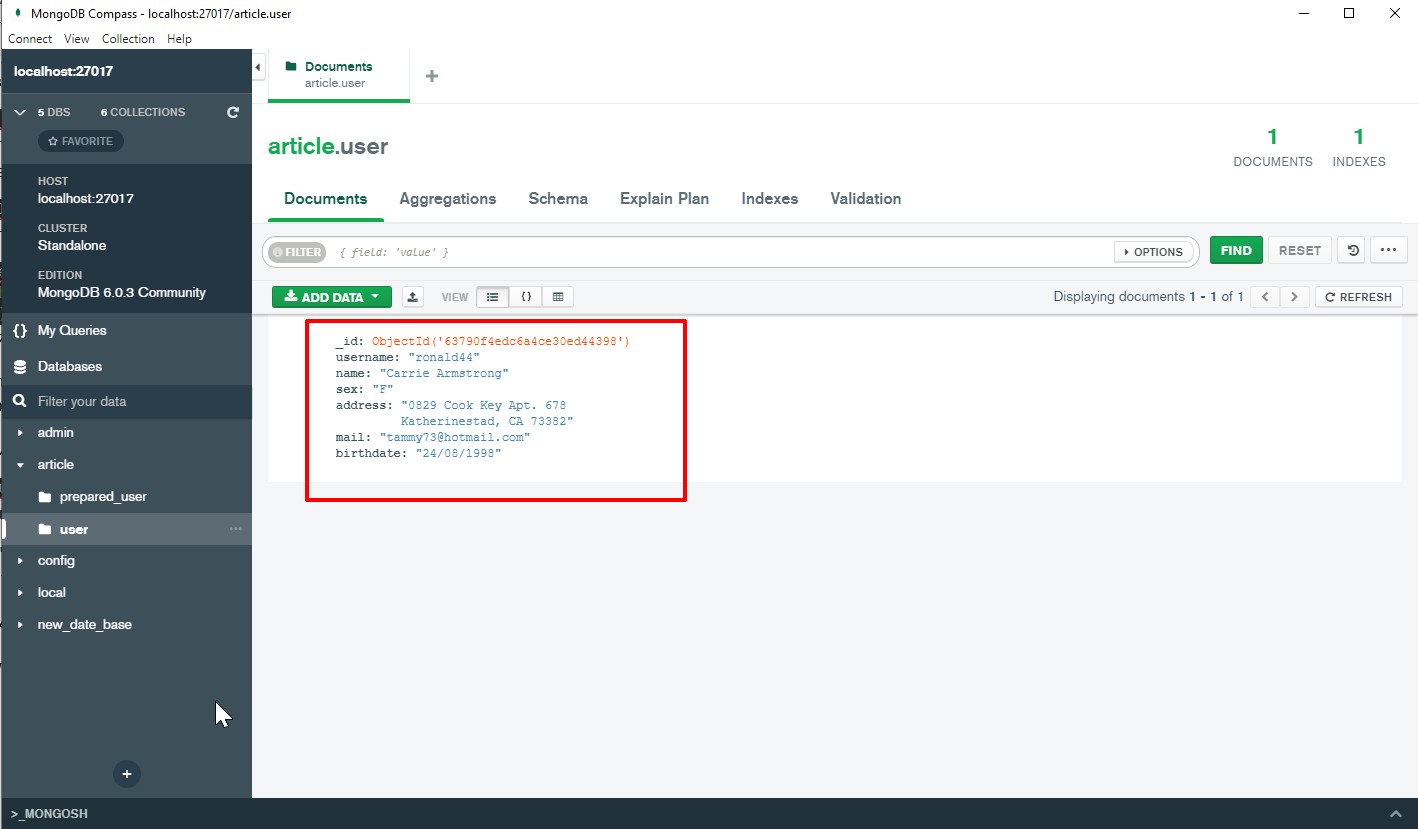
В этом пункте важно поставить галочку на “Install MongoDB Compass” - на компьютер установится графический интерфейс для взаимодействия с СУБД.



После установки нам открывается программа MongoDB Compass, где уже вставлены данные для подключения к установленному серверу. Нажимаем кнопку "connect", и мы в нем:



Ниже пример как можно создать базу данных и коллекцию. Коллекции – это аналог таблиц в SQL-базах. В них мы храним информацию, которая нам нужна.



## Установка MongoClient и код на Python

Для Windows:

pip install pymongo faker

Для MacOS:

python pip install pymongo faker

Для знакомства с СУБД MongoDB нам хватит сделать несколько методов, таких как: добавление пользователя, получить всю коллекцию, изменить какое-то значение, поиск, по заданным критериям и на этом все. Если вам будет интересна эта СУБД,  вы сможете самостоятельно ознакомиться с другими её возможностями.

У нас будет два файла - **pymongoAPI.py** и **main.py** (призываю вас всегда разделять проект на логические модули – это правильный тон в мире программистов Python). Первый файл будет отвечать за взаимодействие с БД, а во втором мы будем пытаться добавлять новые значения и что-то менять.

**Импорты:**

from pymongo import MongoClient

Для всего взаимодействия с MongoDB потребуется только сам клиент, который и импортируем.

**Класс взаимодействия с MongoDB:**

class MongoDB(object):

def \_\_init\_\_(self, host: str = 'localhost',

port: int = 27017,

db\_name: str = None,

collection: str = None):

self.\_client = MongoClient(f'mongodb://{host}:{port}')

self.\_collection = self.\_client[db\_name][collection]

def create\_user(self, user: dict):

try:

if self.\_collection.find\_one({"username": user.get('username')}) == None:

self.\_collection.insert\_one(user)

print(f"Added New user: {user.get('username')}")

else:

print(f"User: {user.get('username')} in collection")

except Exception as ex:

print("[create\_user] Some problem...")

print(ex)

def get\_all\_users(self):

try:

data = self.\_collection.find()

print("Get all users")

return data

except Exception as ex:

print("[get\_all] Some problem...")

print(ex)

def find\_by\_username(self, username: str):

try:

data = self.\_collection.find\_one({"username": username})

print("Get user by username")

return data

except Exception as ex:

print("[find\_by\_username] Some problem...")

print(ex)

def change\_user(self, username: str, key: str, value: str):

try:

if self.\_collection.find\_one({"username": user.get('username')}) is not None:

self.\_collection.update\_one({"username": username}, {"$set": {key: value}})

else:

print(f'User: {username} not find')

except Exception as ex:

print("[change\_user] Some problem...")

print(ex)

Разберёмся что за чем идет.

**\_\_init\_\_**— это конструктор, он принимает в себя ip адрес сервера, порт, название базы данных и название коллекции. Он создает подключение к серверу и записывает в отдельную внутреннюю переменную. Дальше отдельно вытягивает в переменную нашу базу данных. Часто спрашивают зачем ставить нижней прочерк перед полем – это нужно, чтобы показать другим разработчикам, который показывает, что его трогать не надо (так принято показывать в Python приватные переменные).

**Create\_user**– «кушает» словарь, это связанно с тем, что мы будем сразу передавать нового пользователя. Можно формировать его и внутри метода, но так проще. Сразу скажу, что все запросы к базам данных следует оборачивать в try… except, потому что в случае ошибки у нас не будет «падать» программа. В реальной жизни если из-за этой ошибки пользователи не смогу воспользоваться функциями, у вас могут быть проблемы.

Теперь переходим к сути. Обращаемся к методу **find\_one**для проверки, существует ли пользователь в коллекции. Внутрь мы передаем словарь ключ (искомое поле) и значение. Если нам в ответ вернется None, то значит такого значения не существует, ну и если словарь, то значит пользователь существует (сразу отмечу, что username у нас будут индивидуальным значением, то есть не будет повторяться).

Дальше обращаемся к **insert\_one**и передаем словарь пользователя. Чтобы видеть ошибки, мы делаем выводы в консоль о выполненном действии и в случае ошибки показываем модуль с ошибкой.

**Get\_all\_users**– с помощью этого метода будем выводить все записи в коллекции. Обращаемся к объекту коллекции и применяем метод find, без каких-либо значений. В ответ мы получаем массив значений и возвращаем его из метода.

**Find\_by\_username** – теперь будем искать по уникальному значению в виде username. Принимаем искомого пользователя и обращаемся к методу коллекции **find\_one**. Он будет выводить первое совпадение и возвращать его словарем, если пользователя не существует, то вернется None.

**Change\_user** – последний метод на сегодня, он будет менять значение. Принимает в себя username, key (изменяемое значение) и value (новое значение). В начале идет проверка на наложение пользователя в коллекции, потом используется **update\_one**.

Что же это такое, метод, предоставляемый объектом коллекции для обновления записи? Первым значением идет словарь для поиска по значение, потом второй с служебным оператором **$set**с вложенным вторым словарем, внутри которого находится ключ и его новое значение (подробнее об служебных операторах по [**ссылке**](https://www.mongodb.com/docs/manual/reference/operator/update/)).

## Main.py

Теперь идем проверять наш класс. Будем рассматривать код по строчкам, чтобы можно было посмотреть все по очереди и отслеживать, что меняется в БД. Импортируем наш MongoDB из файла mongoAPI. Так же библиотеку для создания несуществующих пользователей.

from mongoAPI import MongoDB

import faker

Дальше создаем объект dbase на основе нашего класса, передаем в него название БД и коллекции с которой будем работать:

dbase = MongoDB(db\_name='article', collection='user')

Так же не стоит забывать про объект для создание «фейковых» пользователей:

faker\_obj = faker.Faker()

Создаем наш фейковый профиль и посмотрим вывод в консоль:

user\_profile = faker\_obj.simple\_profile()

user\_profile['birthdate'] = user\_profile['birthdate'].strftime('%d/%m/%Y')

print(user\_profile)

Словарь состоит из логина, имени, пола, адреса, почты и даты рождения:

*{'username': 'dparker', 'name': 'Megan Morris', 'sex': 'F', 'address': '15074 Conner Lane\nDavidshire, ME 67230', 'mail': 'leslie72@gmail.com', 'birthdate': '24/04/2017'}*

Пробуем добавить его в коллекцию:

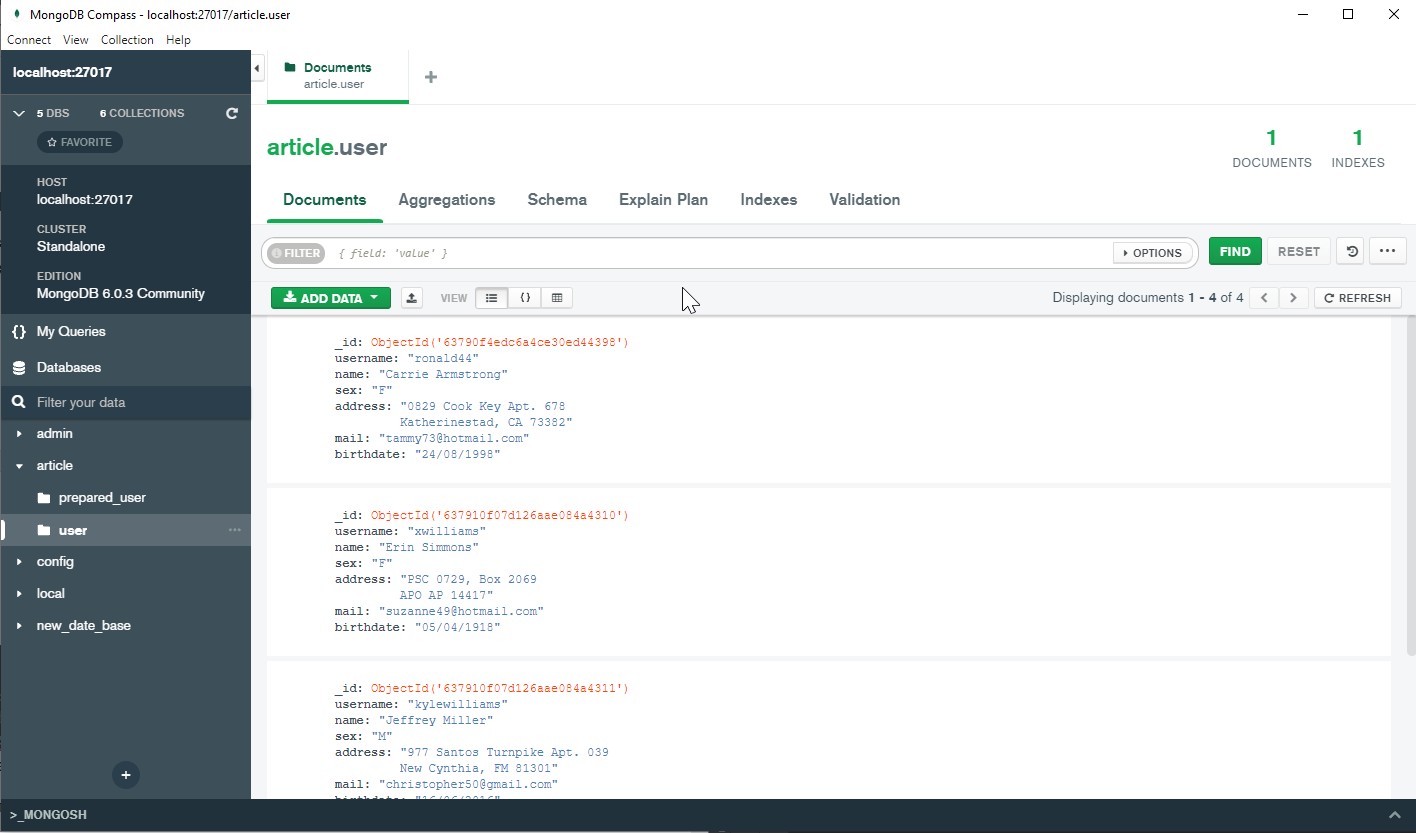
dbase.create\_user(user\_profile)

 Выводы в консоль:

*{'username': 'ronald44', 'name': 'Carrie Armstrong', 'sex': 'F', 'address': '0829 Cook Key Apt. 678\nKatherinestad, CA 73382', 'mail': 'tammy73@hotmail.com', 'birthdate': '24/08/1998'}*

*Added New user: ronald44*

Как выглядит все в коллекции:



Тут может возникнуть вопрос, что за значение \_id? Это уникальный ID, который автоматически создает MongoDB. Можно провести аналогию с Primary Key в MySQL.

Давайте добавим пару случайных пользователей с помощью библиотеки faker и цикла for. Выглядит все так:

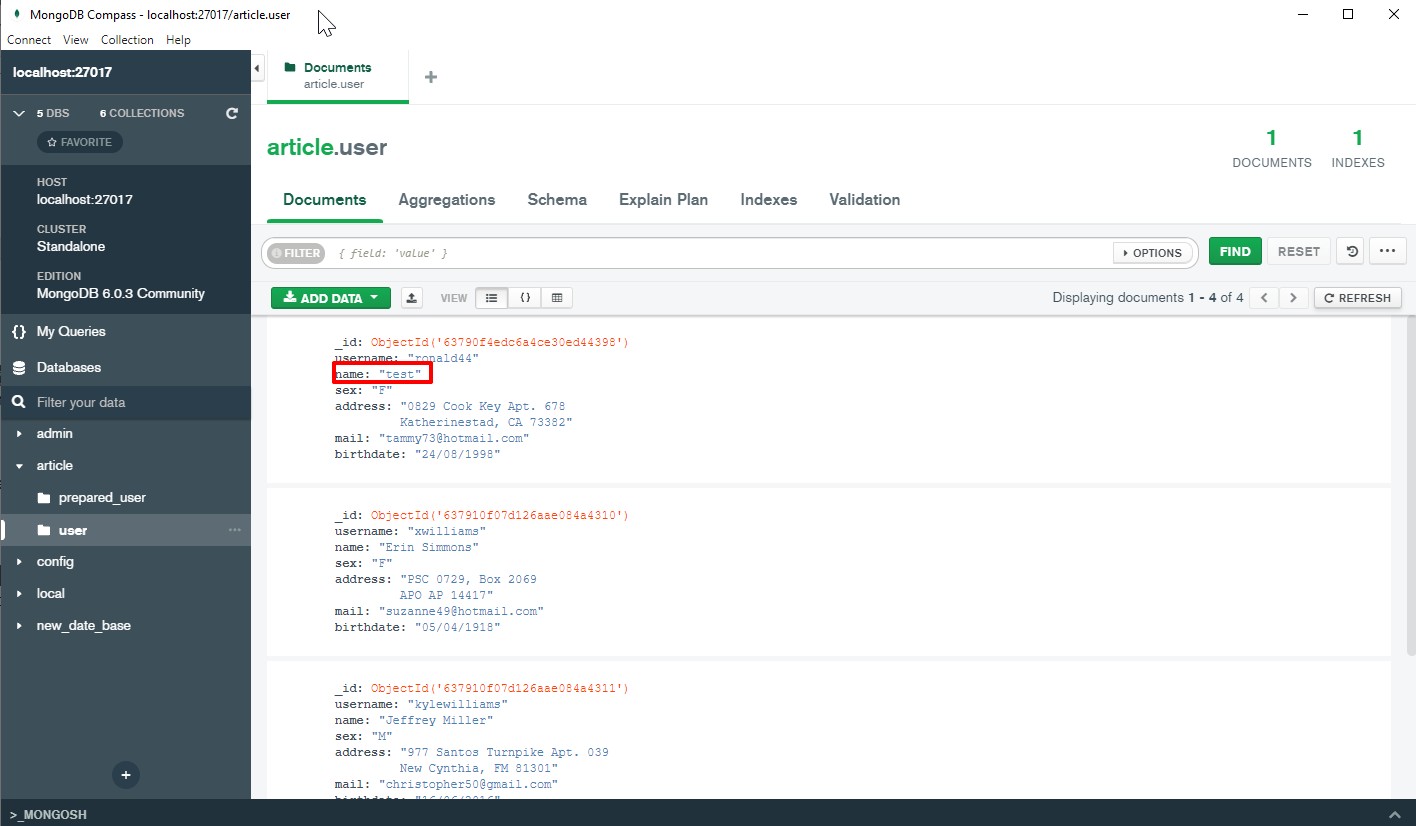
for \_ in range(3):

user\_profile = faker\_obj.simple\_profile()

user\_profile['birthdate'] = user\_profile['birthdate'].strftime('%d/%m/%Y')

dbase.create\_user(user\_profile)

Добавили пользователей: xwilliams, kylewilliams и obrowning. Так это выглядит в коллекции:



Чтобы посмотреть полную сводку из коллекции надо будет воспользоваться циклом for. Это связано с тем, что в ответ нам идет объект Cursor:

result = dbase.get\_all\_users()

for i in result:

print(i)

 Вывод в консоль:

*Get all users*

*{'\_id': ObjectId('63790f4edc6a4ce30ed44398'), 'username': 'ronald44', 'name': 'Carrie Armstrong', 'sex': 'F', 'address': '0829 Cook Key Apt. 678\nKatherinestad, CA 73382', 'mail': 'tammy73@hotmail.com', 'birthdate': '24/08/1998'}*

*{'\_id': ObjectId('637910f07d126aae084a4310'), 'username': 'xwilliams', 'name': 'Erin Simmons', 'sex': 'F', 'address': 'PSC 0729, Box 2069\nAPO AP 14417', 'mail': 'suzanne49@hotmail.com', 'birthdate': '05/04/1918'}*

*{'\_id': ObjectId('637910f07d126aae084a4311'), 'username': 'kylewilliams', 'name': 'Jeffrey Miller', 'sex': 'M', 'address': '977 Santos Turnpike Apt. 039\nNew Cynthia, FM 81301', 'mail': 'christopher50@gmail.com', 'birthdate': '16/06/2016'}*

*{'\_id': ObjectId('637910f07d126aae084a4312'), 'username': 'obrowning', 'name': 'Amy Thompson', 'sex': 'F', 'address': '041 Leah Locks Suite 280\nReedport, MI 56259', 'mail': 'vargassarah@hotmail.com', 'birthdate': '25/08/1915'}*

Поиск человека по логину:

print(dbase.find\_by\_username('username'))

 Вывод в консоль:

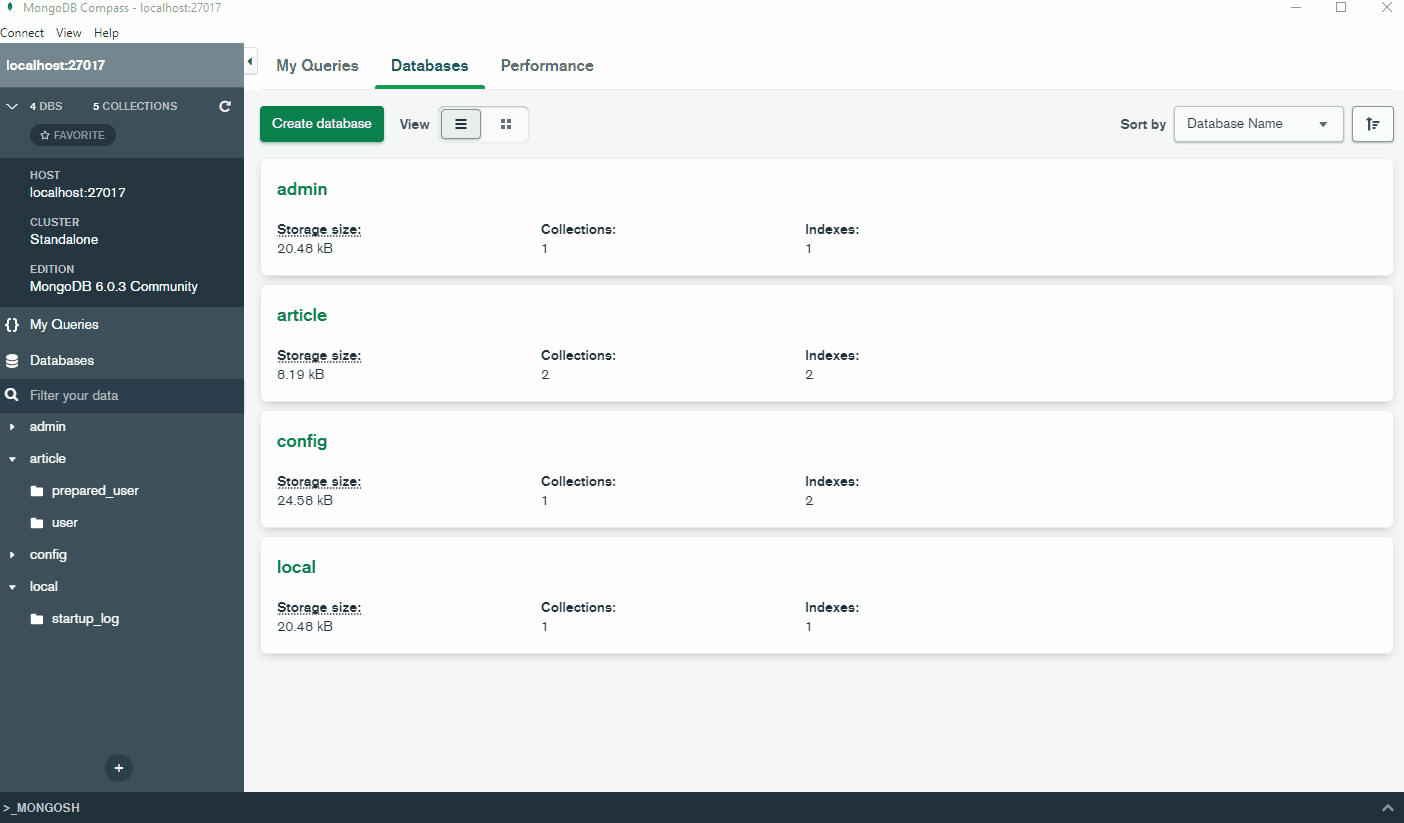
*Get user by username*

*{'\_id': ObjectId('63790f4edc6a4ce30ed44398'), 'username': 'ronald44', 'name': 'Carrie Armstrong', 'sex': 'F', 'address': '0829 Cook Key Apt. 678\nKatherinestad, CA 73382', 'mail': 'tammy73@hotmail.com', 'birthdate': '24/08/1998'}*

Теперь изменим пользователю имя на "тест":

dbase.change\_user('ronald44', 'name', 'test')

После запуска мы увидим в коллекции следующее:



## Заключение

Сегодня мы тестировали СУБД MongoDB - сделали тестовое добавление коллекции и попробовали вносить в базу изменения.

Как вы видите, Mongo очень удобна и гибка, что позволяет легко масштабировать из маленького проекта в большой. Советую вам изучить её более подробно.

## Полезные ссылки

Документации библиотеки pymongo: <https://pymongo.readthedocs.io/en/stable/>

Подробный курс на ютубе: <https://www.youtube.com/playlist?list=PL6plRXMq5RABbVCM0dn23PTKO13WcXnbf>

Материал взят с сайта: http://directprobi.ru/blogs/mongo-python-nosql-baza-dannyh-pymongo-api-database-mongodb-subd/

## Приложение 3. Примеры документов

1. Библиотека. Читательский билет с записями о книгах, находящихся у читателя.
2. Аптека. Товарный чек с информацией о купленных лекарствах.
3. Больница. Информация о лечении: пациент, врач, дата начала, дата окончания, виды лечения (список).
4. Предприятие. Информация о сотруднике и истории его назначений.
5. Курьерская доставка. Информация о посылке: отправитель (Ф.И.О., адрес отправления, паспортные данные), получатель (Ф.И.О, адрес получения, паспортные данные), масса и габариты посылки, информация о курьере и датах отправления и получения.
6. Повышение квалификации. Список курсов университета: Даты проведения, информация о преподавателе, наименование, количество часов, записанные на курсы сотрудники (до 3).
7. Научная конференция. Программа конференции: дата и наименование конференции, несколько секций со списком статей в каждой секции.
8. Прокат велосипедов. Информация об использовании велосипеда: дата начала, дата окончания, марка (тип), информация о клиенте, стоимость.
9. Футбольный турнир (РФПЛ). Игры: команда 1, основной состав (достаточно указать 3 игроков), команда 2, основой состав (достаточно указать 3 игроков), счет, информация о голах.
10. Пиццерия. Информация о пиццерии со списком продаваемых пицц.
11. Кулинарная книга. Рецепт.
12. Поликлиника. Информация о посещениях: дата, врач, посетитель, диагноз.
13. Сессия. Результаты экзаменов с указанием информации о преподавателе и списке студентов с оценками (до 5 студентов).
14. Расписание занятий. Расписание занятий на 1 день (для нескольких групп)
15. Социальная сеть. Изобразите диаграмму сущность связь для социальной сети Сообщения с главной страницы: дата, текст, комментарии (автор, текст).
16. Интернет-магазин мебели. Товарный чек с информацией о товарах, покупателе и магазине.
17. Онлайн кинотеатр. Информация о пользователе: Имя, предпочтения, список просмотренных фильмов (жанр, наименование, дата просмотра, длительность).
18. Бронирование авиабилетов. Информация о заказанных билетах: пункт А в пункт Б, дата, вес багажа, тип меню.
19. Музыкальные альбомы. Музыкальные альбомы.
20. Бронирование авиабилетов. Брони.
21. Курсы и студенты. Проекты: дата начала и окончания, информация о сотрудниках в проекте (до трех): имя, идентификационный номер, адрес, зарплата и дата рождения.
22. Галереи. Несколько групп со списком произведений в каждой группе.
23. Сеть отелей. Информация об истории бронирования номеров: дата начала, окончания, список постояльцев, стоимость.
24. Прокат автомобилей. Список автомобилей с историей их использования.
25. КХЛ. Информация об играх: команда 1, основной состав (достаточно указать 3 игроков), команда 2, основой состав (достаточно указать 3 игроков), счет, информация о голах.
26. Книжный онлайн-магазин. Товарный чек.
27. Мастерская. Коллекция документов с информациях о заказах: дана начала и окончания, имя мастера, имя клиента, техника, ее вид, список работ.
28. ЖЭК. Список заявок: дана начала и окончания работ, адрес клиента, бригада специалистов со списком специалистов.
29. Парки. Информация о трех парках, с указанием наименования, площади, место адреса и списка павильонов (до трех): наименование, тип (кафе, продуктовый, развлекательный, прокат вещей), занимаемая площадь.