**GeekBrains**

Факультет: Разработчик

Специализация: Аналитик

Дипломный проект

Формирование базы данных

(сбор, фильтрация данных) и синхронизация баз

Автор проекта: Федорова Марина Александровна

2023

Оглавление

[Введение 3](#_Toc138595212)

[Глава 1. Анализ базы данных и ее установка 4](#_Toc138595213)

[1.1. База https://world.openfoodfacts.org/data, характеристики 4](#_Toc138595214)

[1.2. Установка сервера 5](#_Toc138595215)

[1.3. Идентификатор документа 7](#_Toc138595216)

[Глава 2. Разворачивание базы на сервере 7](#_Toc138595217)

[Глава 3. Приведение данных к единому формату 9](#_Toc138595218)

[Глава 4. Запуск MySQL сервера локально 12](#_Toc138595219)

[Глава 5. Анализ базы MongoDB 14](#_Toc138595220)

[Глава 6. Написание скрипта для выбора и записи данных их MomgoDB в SQL 14](#_Toc138595221)

[6.1. Создание виртуального окружения. 14](#_Toc138595222)

[6.2. Подключаемся к базе MySQL 18](#_Toc138595223)

[6.3. Реализация запроса к Mongodb 19](#_Toc138595224)

[Заключение 21](#_Toc138595225)

# Введение

**Описание задачи**

Разработано Приложение, которое будет определять, есть ли в составе продукта запрещенные для человека ингредиенты. .Но пока база маленькая, нужно ее увеличить.

**Цель проекта**

Сделать общую базу, которую можно будет использовать для Приложения из двух доступных баз - из Призмы и OpenFoodFacts

**План работы**:

1. Выяснить, какой тип имеет база <https://world.openfoodfacts.org/data>
2. Развернуть базу локально, чтобы иметь возможность с ней работать.
3. Преобразовать и отфильтровать данные из базы
4. Влить данные из базы в существующую на данный момент базу.

**Проблема, которую будет решать проект**

Увеличение объема база для корректной работы Приложения.

**Специализация**: аналитик

**Необходимый для работы опыт:** работа с базами данных, программирование на Python

**Инструменты для работы**: MongoDB, MYSQL, Python

**Состав команды**: реализовала проект в качестве разработчика базы данных, аналитика данных

# Глава 1. Анализ базы данных и ее установка

## База <https://world.openfoodfacts.org/data>, характеристики

“База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД).

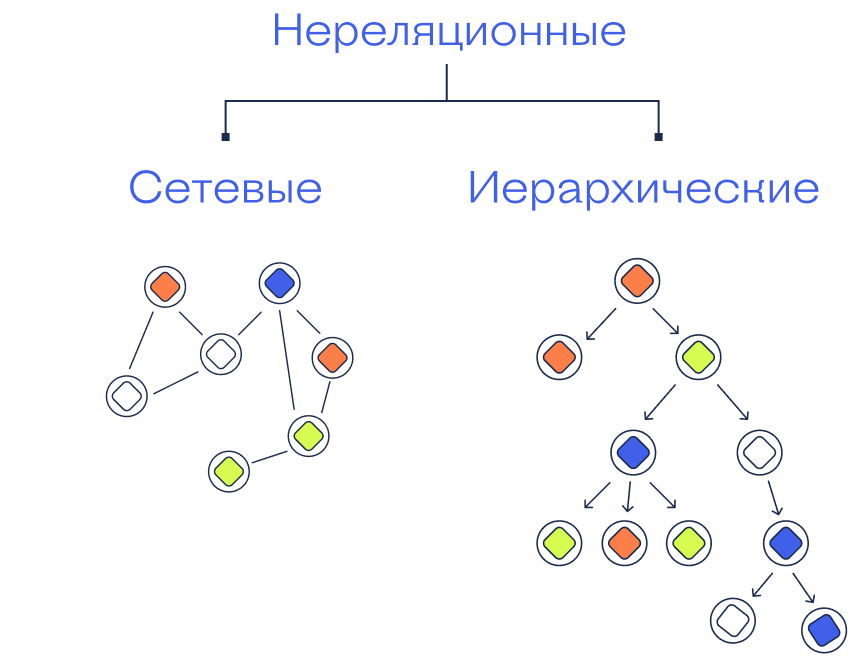
Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать”

При анализе базы OpenFoodFacts выяснилось, что OpenFoodFacts - база MongoDB.

В процессе учебы мы знакомились только с реляционными базами данных. Сейчас мы столкнулись с Нереляционной , NoSQL — not only SQL.

Здесь нет таблиц, строк и столбцов. В NoSQL данные оптимизируются более тонко, под конкретную задачу, а внутри имеют гибкую нетипизированную схему. Понадобилось добавить в документ или поле рандомные данные? Пожалуйста. Такой подход позволяет эффективнее работать с разреженными данными. Нереляционные БД можно поделить на два типа: сетевые и иерархические.

* **Плюсы:** большая скорость обработки данных, работа с неструктурированными данными (какие угодно), децентрализация систем, легче и дешевле масштабироваться, чем при SQL, преимущества в шаринге и репликации;
* **Минусы**: привязка к конкретной СУБД, ограниченность языков запросов и API, в процессе разработки базы данных могут случаться непредвиденные ошибки (так как схема БД не требуется изначально),



* миграция с одной БД NoSQL на другую БД NoSQL может стать квестом с препятствиями, понадобятся проприетарные инструменты для работы с БД;

Отличие от реляционных баз данных MongoDB не использует табличное устройство с четко заданным количеством столбцов и типов данных. MongoDB является документо-ориентированной системой, в которой центральным понятием является документ. Документ представляет набор пар ключ-значение. Например, в выражении "name": "Tom" "name" представляет ключ, а "Tom" - значение.

Ключи представляют строки. Значения же могут различаться по типу данных. В данном случае у нас почти все значения также представляют строковый тип, и лишь один ключ (company) ссылается на отдельный объект. Всего имеется следующие типы значений:

String: строковый тип данных, (для строк используется кодировка UTF-8)

Array (массив): тип данных для хранения массивов элементов

Binary data (двоичные данные): тип для хранения данных в бинарном формате

Boolean: булевый тип данных, хранящий логические значения TRUE или FALSE, например, {"married": FALSE}

Date: хранит дату в формате времени Unix

Double: числовой тип данных для хранения чисел с плавающей точкой

Integer: используется для хранения целочисленных значений размером 32 бита, например, {"age": 29}

Long: используется для хранения целочисленных значений размером 64 бита

JavaScript: тип данных для хранения кода javascript

Min key/Max key: используются для сравнения значений с наименьшим/наибольшим элементов BSON

Null: тип данных для хранения значения Null

Object: объект, который содержит набор свойств

ObjectId: тип данных для хранения id документа

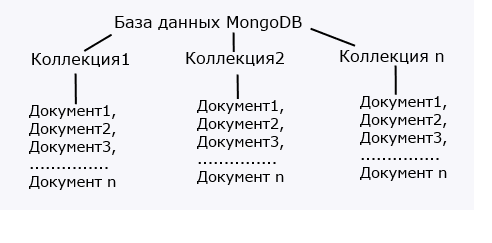
Regular expression: применяется для хранения регулярных выражений

Decimal128: тип данных для хранения десятичных дробных чисел размером 128 бит, которые позволяют решить проблемы с проблемой точности вычислений при использовании дробных чисел, которые представляют тип Double.

Timestamp: применяется для хранения времени

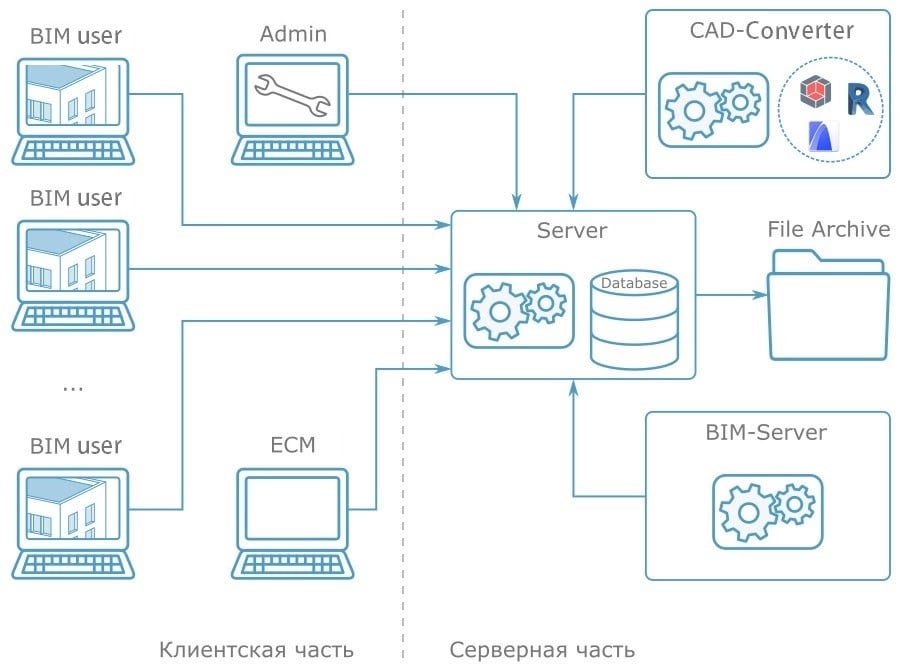
Из документации выяснилось, что если в реляционных бд содержимое составляют таблицы, то в mongodb база данных состоит из коллекций (рис 1.1.)

***Рисунок 1.1. Структура базы данных***

. [](https://github.com/MarinaFedorovaok/BD/blob/main/pictures/mango.png)

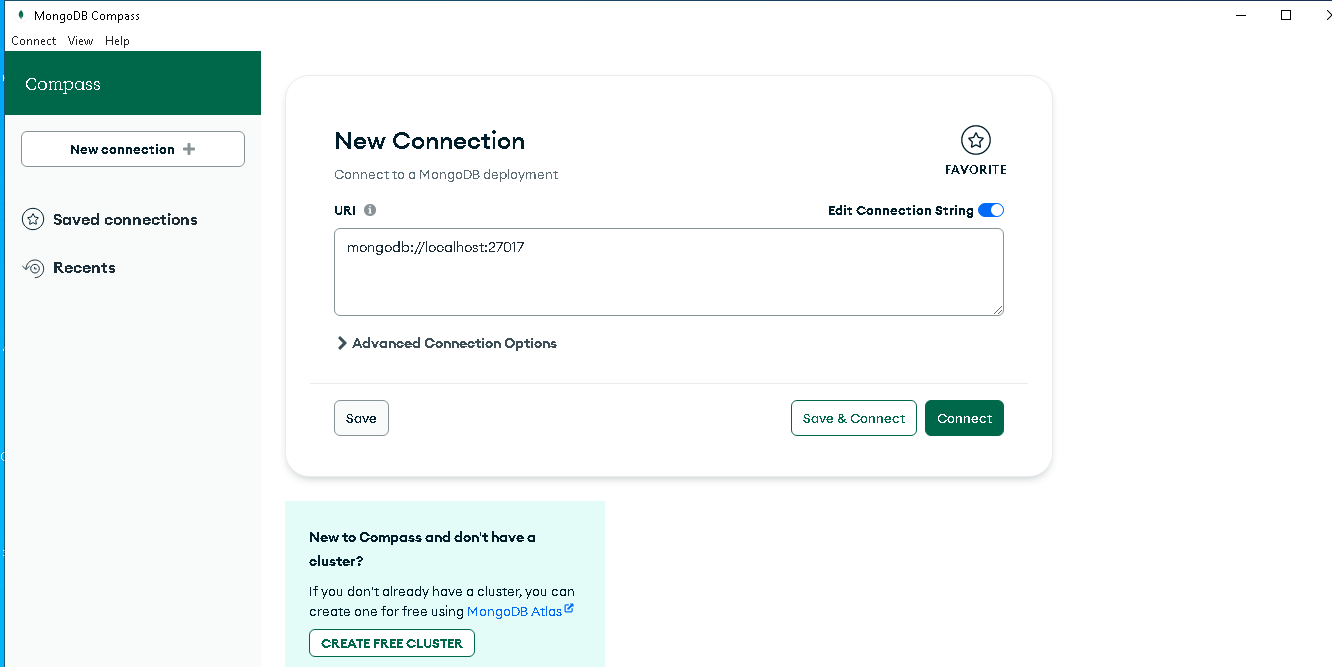
## Установка сервера

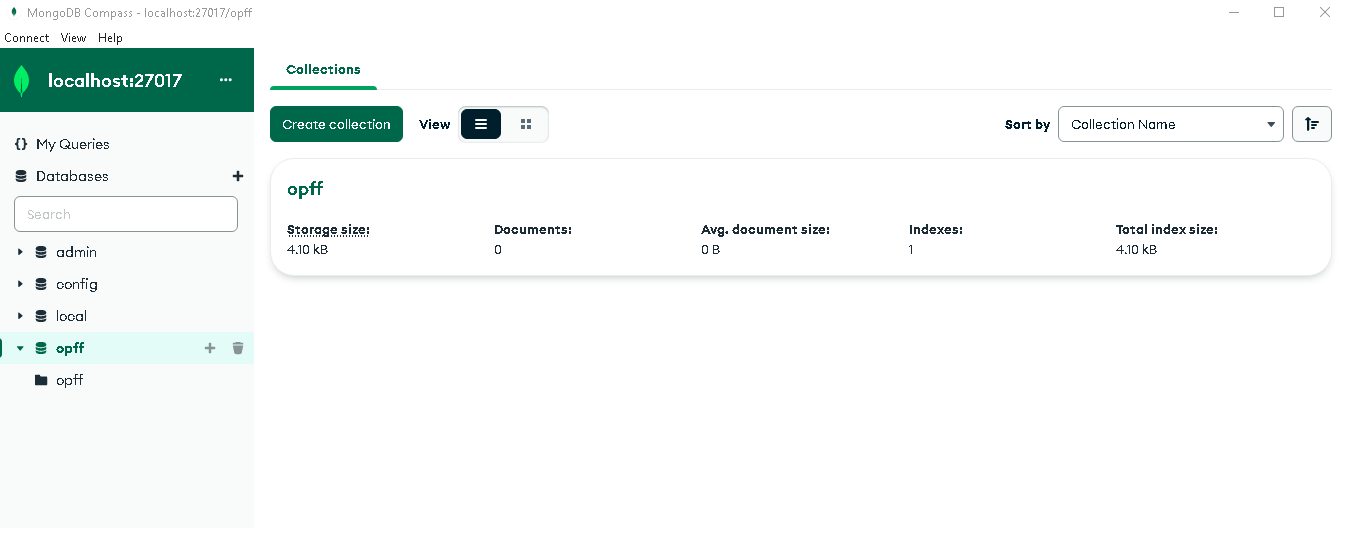
В работе обычно используется клиент-серверная система баз данных. Рассмотрим на примере BIM-системы.



В центре у нас сервер (Server), который связывает пользователей, BIM-систему и все данные, создаёт бэкапы, предоставляет и (или) ограничивает доступ к клиентским приложениям и многое другое.

Сервер было решено скачивать с официального сайта <https://www.mongodb.com/try/download/community> Скачиваем установщик с расширением msi Это установщик Windows. При установке выбираем нужный диск и нам нужно решить, будет ли Mongo сервисом. В моем случае - выбираю что будет. После установки видим:

***Рисунок 1.2. Установка сервера***[](https://github.com/MarinaFedorovaok/BD/blob/main/pictures/mongo1.png)

Нажимает connect и соединимся с локалхостом. [](https://github.com/MarinaFedorovaok/BD/blob/main/pictures/mongo2.png)

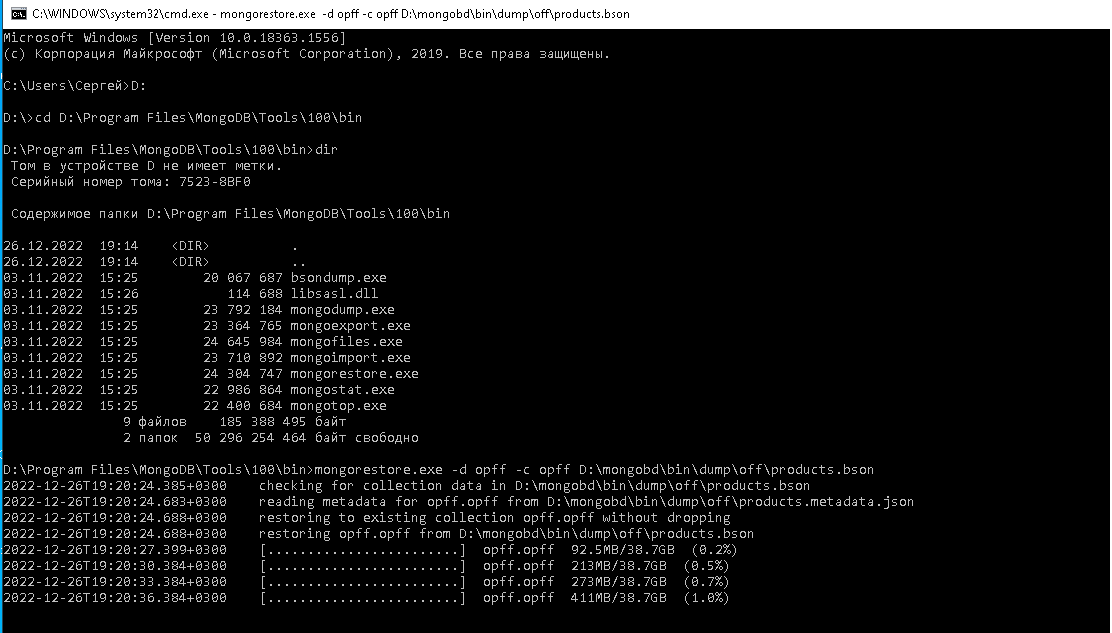
## ****Идентификатор документа****

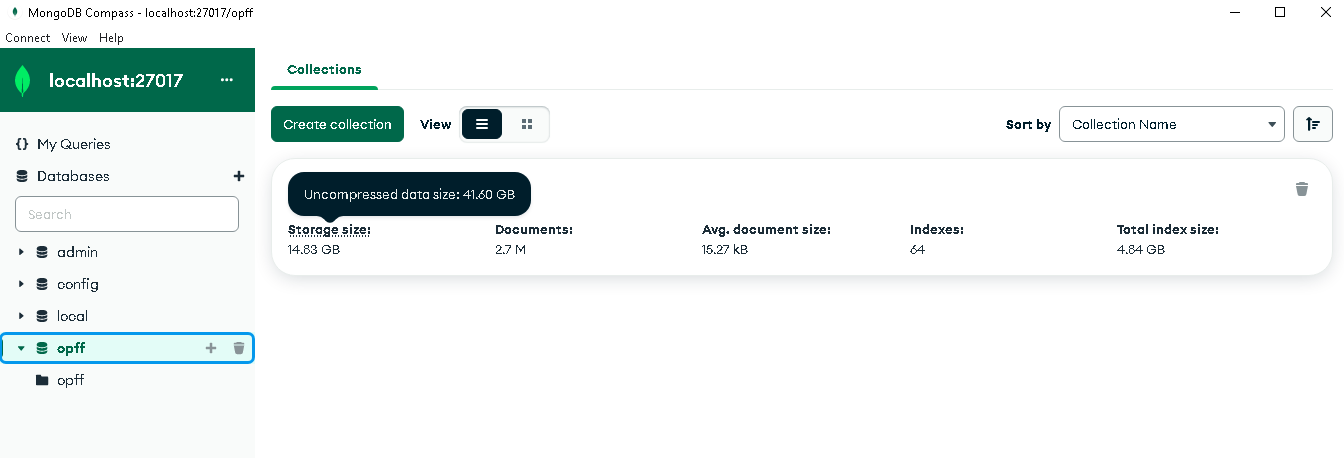
Для каждого документа в MongoDB определен уникальный идентификатор, который называется \_id. При добавлении документа в коллекцию данный идентификатор создается автоматически.

Документ можно представить как объект, хранящий некоторую информацию. В нашем случае документ - json, в то время как скачали мы dump базы.

# Глава 2. Разворачивание базы на сервере

Чтобы развернуть базу (сделать из .bson .json) нужно в командной строке ввести: mongorestore -d db\_name -c collection\_name path/file.bson

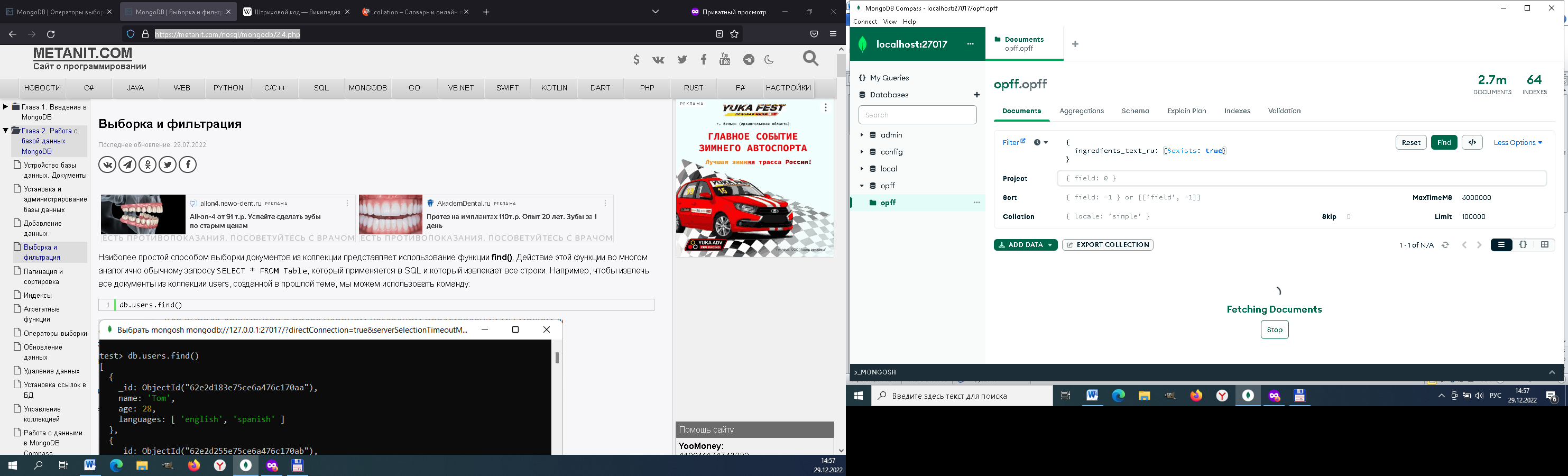
Существуют инструменты MondoDB командной строки. Скачиваем их с официального сайта https://www.mongodb.com/try/download/database-tools ***Рис 2.1. Разворачивание базы на сервере:*** [](https://github.com/MarinaFedorovaok/BD/blob/main/pictures/dump_restore.png)

***Рис 2.2. Вид базы после перезагрузки и запуска MongoDB Compass*** [](https://github.com/MarinaFedorovaok/BD/blob/main/pictures/data_restored.png)

Глава 3. Приведение данных к единому формату.

**Подготовка данных**

Создаем коллекцию с нужным нам фильтром – с наличием ингридиентов на русском языке.

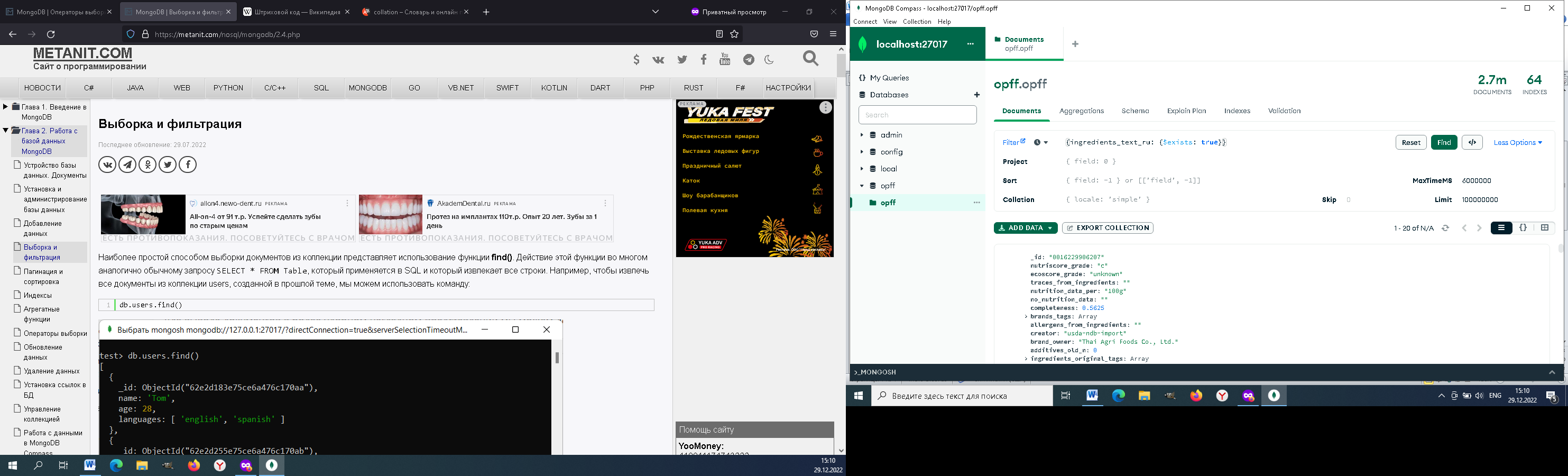


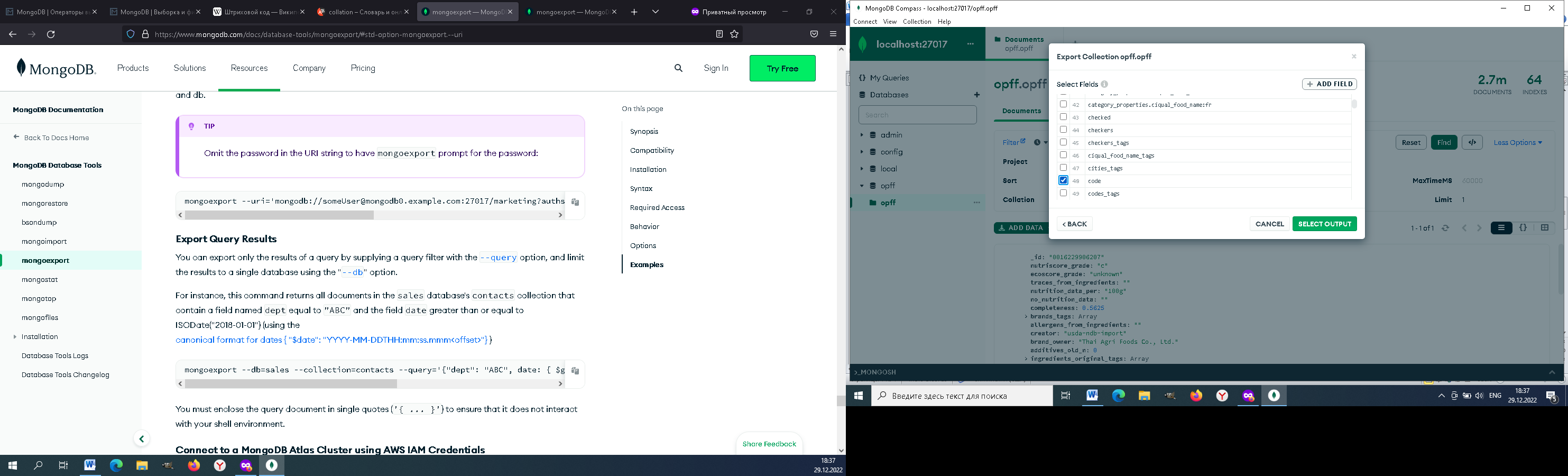
Синтаксис фильтров к базе: <https://metanit.com/nosql/mongodb/2.4.php>

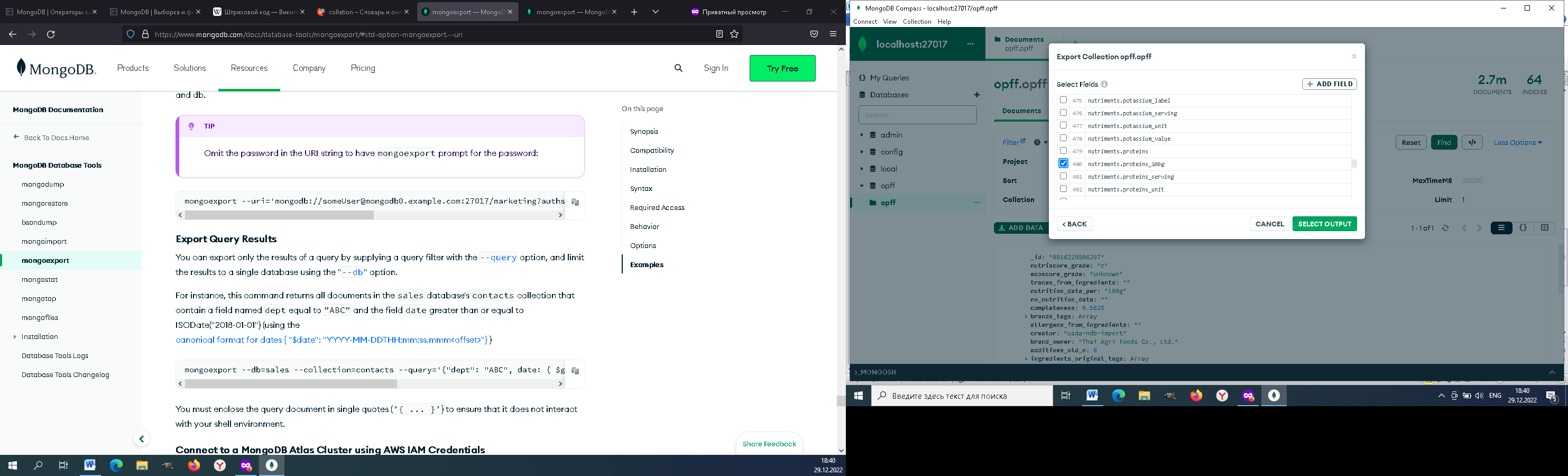
Дальше нам нужно экспортировать данные в csv

Экспортируем только те поля, которые нам нужны, а именно:

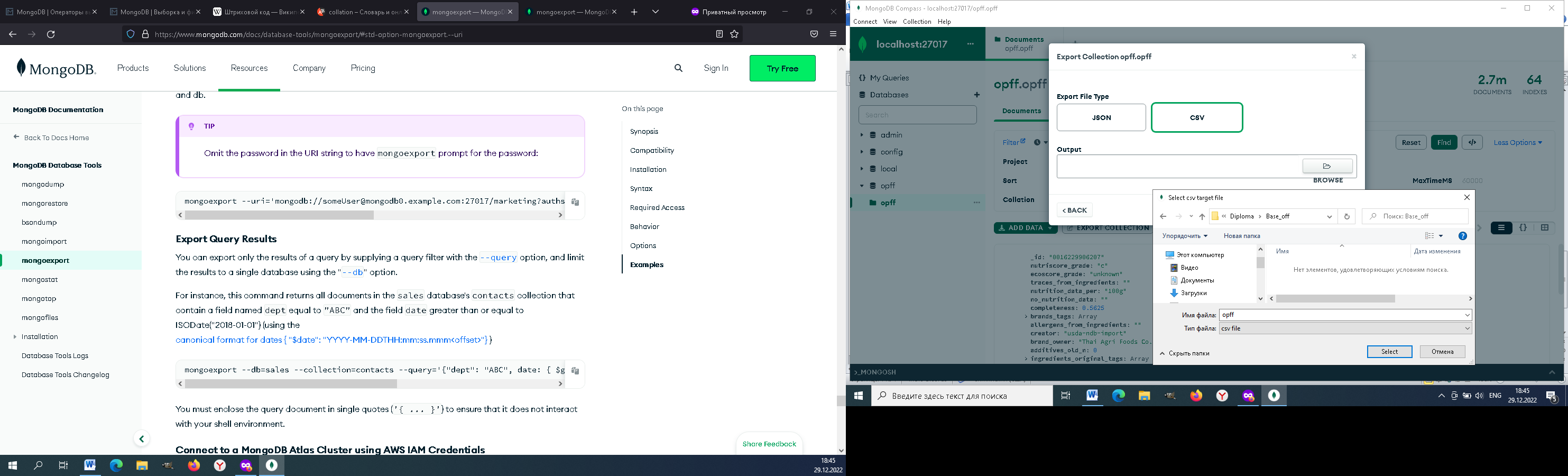
1. Название продукта
2. Баркод
3. Из состава продукта (nutrients) выбираем на 100 грамм – калорийность, белки, жиры, углеводы







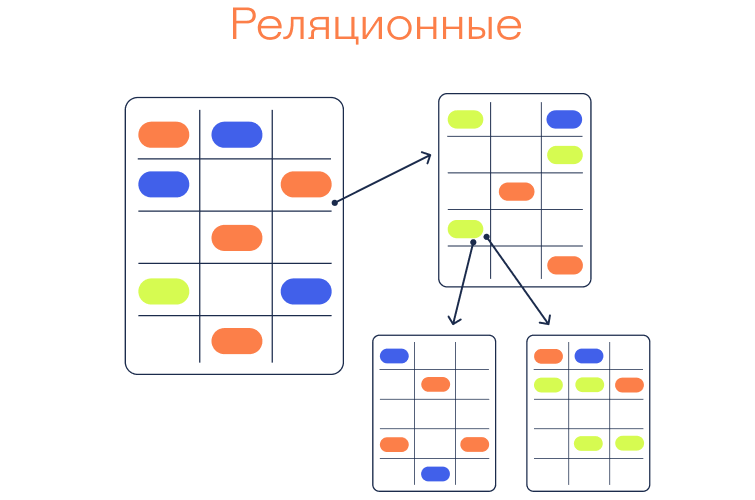
Выбираем формат данных и куда экспортируем



В итоге получаем данные в csv файле

# Глава 4. Разворачивание SQL базы, Запуск MySQL сервера локально

### Реляционная база данных, SQL — Structured Query Language.



Все данные связаны между собой и распределены по заранее созданным таблицам (книжная стойка). У всех таблиц есть столбцы (книжные полки) и строки (книги). В общем, всё буквально разложено по полочкам. Поэтому такие БД часто называют табличными.

* Плюсы: целостность, сохранность (соответствие требованиям ACID) и структурированность данных, богатый бэкграунд SQL даёт огромные функциональные возможности, не зависит от конкретной СУБД;
* Минусы: сложность, плохая оптимизация в работе с разреженными данными, проблемы с многократными выполнением одинаковых запросов (N+1);
* Примеры: SQLite, MySQL, PostgreSQL, Microsoft SQL Server.

**Принцип работы MySQL-серверов**

Он такой же, как в любых клиент-серверных моделях. Одно устройство делает запрос, а второе отвечает. Запрашивающих может быть больше одного, все зависит от сервера, сети и поставленных задач.

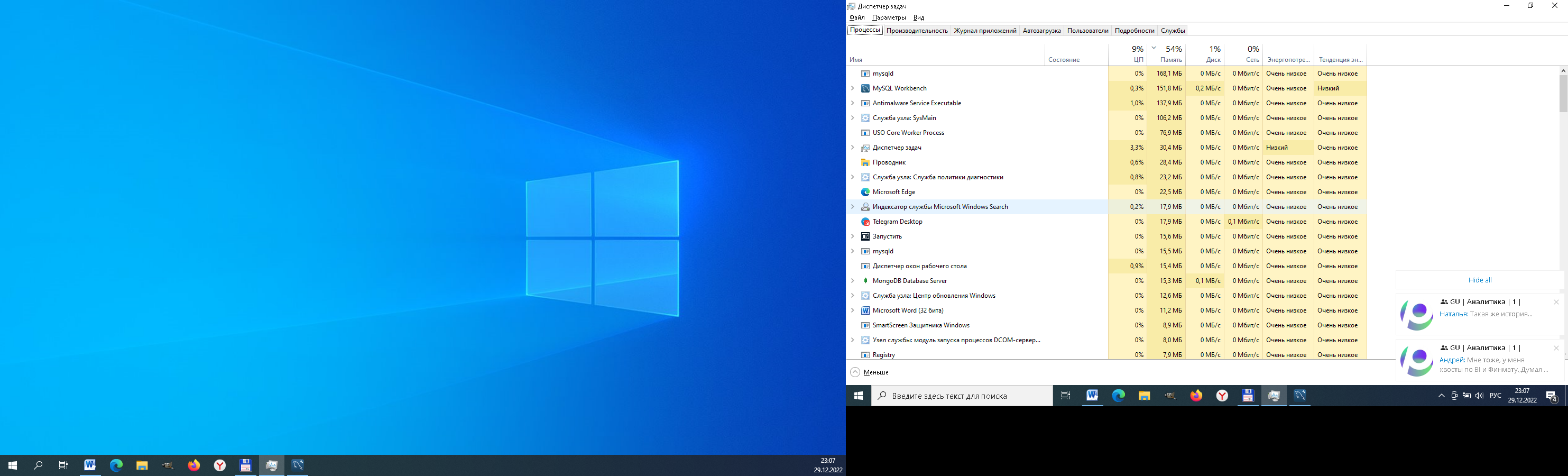
Технически немного иные, но по своей сути идентичные процессы происходят в среде MySQL:

* Система создает базу данных для хранения информации (ее сортировки, идентификации и т.п.).
* Клиенты (другие компьютеры в сети) подают запросы к базе с помощью специфичных для SQL команд.
* Серверное приложение обрабатывает запрос и выдает ответ клиенту (выдает запрашиваемые данные).

Для взаимодействия с MySQL-сервером используются соответствующие утилиты. Некоторые работают только в командной строке. Некоторые награждены графическим интерфейсом. Популярные решения – WorkBench, SequelPro, SQL Studio, TablePlus. Правда, большинство вебмастеров предпочитает [phpMyAdmin](https://timeweb.com/ru/community/articles/kak-ustanovit-phpmyadmin), так как та входит в LAMP и работает в браузере.

Скачиваем и устанавливаем MYSQL Server[**https://dev.mysql.com/downloads/mysql/**](https://dev.mysql.com/downloads/mysql/)

После установки он запустится. Это можно увидеть в диспетчере задач



Устанавливаем Workbrench,запускаем и соединяемся с сервером

[**https://www.mysql.com/products/workbench/**](https://www.mysql.com/products/workbench/)

## Операционная среда серверов

Примеры серверов БД: SQL SERVER (Microsoft), SQL BASE SERVER, Oracle SERVER (Oracle Corporation), IBM DB2, Informix.Каждый сервер БД может работать на определенных типах компьютеров и сетей. Операционными системами серверов могут быть MSDOS, OS/2, Xenix,Unix, Dec VMS/ Рабочие станции пользователей обычно работают под управление MSDOS, OS/2, Xenix, Unix.  
Существуют возможности смешанного использования различных ОС. Большая часть SQL-серверов может хранить описание БД в системном каталоге, который обычно бывает доступен пользователям. Для обращения к этому каталогу используются SQL-запросы. Реляционные СУБД могут использовать информацию, хранящуюся в системном каталоге для оптимизации SQL-запросов.

## Ссылочная целостность реляционных БД

Большинство SQL-серверов поддерживают ссылочную целостность реляционных БД, состоящих из отдельных таблиц, которые могут быть объединены на основе общей информации Рассмотрим на следующем примере: база данных содержит таблицу клиентов и таблицу заказов, которые связаны полем номера клиента, содержащимся в обеих таблицах. Поскольку может быть более одного заказа от одного клиента, соотношение таблиц — «один-ко-многим». Когда таблицы соединены, то таблица клиентов является родительской, а таблица заказов — дочерней. Если запись-родитель стирается, а соответствующие ей дочерние записи — нет, то говорят, что дочерние записи «осиротели». Ссылочная целостность означает, что ни в одной таблице не допустимы записи-«сироты». Запись может осиротеть тремя способами:

1)родительская запись удалена;

2)родительская запись изменена таким образом, что связь между «родителем» и «потомками» потеряна;

3)введена дочерняя запись без соответствующей родительской.

Поддержание ссылочной целостности возможно несколькими способами:

1)Через ключи, хранящиеся в таблицах БД (родительские таблицы содержат первичные ключи, представляющие собой комбинации внешних ключей, которые могут быть найдены внутри каждой из дочерних таблиц).

2)Использование присоединенных процедур — процедурная ссылочная целостность. Присоединенные программы обеспечивают ссылочную целостность за счет автоматического выполнения предложений SQL всякий раз, когда встречается одно из предложений UPDATE/INSERT или DELETE (либо запрещается удаление родительской записи, либо стираются все дочерние записи).

## Транзакции и целостность БД

[Транзакция](https://dic.academic.ru/dic.nsf/ruwiki/1697844) — совокупность логически взаимосвязанных запросов, направленных на согласованное изменение некоторого множества строк в одной или нескольких таблицах БД. Обычно при выполнении транзакций обновляется несколько таблиц и индексов, связанных с этими таблицами. Для того чтобы гарантировать синхронизацию обновления и целостность данных, в серверах обычно используется принцип «все или ничего», означающий, что в БД вносятся либо все обновления или ни одно из них. С этой целью ведется журнал транзакций, в котором регистрируется информация обо всех затребованных изменениях. Этот журнал обеспечивает возможность «прокрутить назад» совершенных транзакции и восстановить предыдущее состояние БД. Это становится важно, когда изменения в БД, предусмотренные в одной транзакции, реализованы лишь частично, например, из-за сбоя аппаратуры.

## Согласованность чтения

Характерна для многопользовательских СУБД. Для её реализации серверы обладают средствами автоматической блокировки.  
Уровни, на которых блокируется таблица во время обновлений:  
- вся таблица;  
- страница (физический блок размером от 1 до 4 Кбайт, содержащий несколько записей).

## Тупиковые ситуации

Серверы БД должны иметь средства определения состояния взаимоблокировки (dead lock). При возникновении такой ситуации выполнение одной из транзакций прерывается, выводя другую транзакцию из состояния вечного ожидания. Прерванная транзакция после исключения возможности её блокировки выполняется сначала.

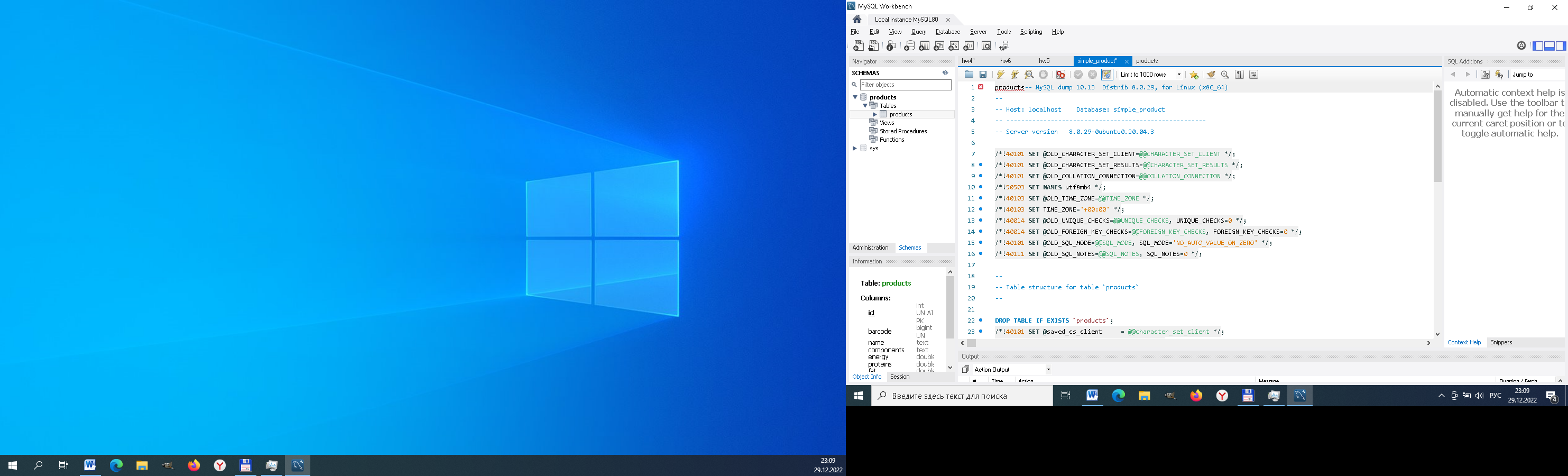
## Схемы оптимизации работ на языке SQL

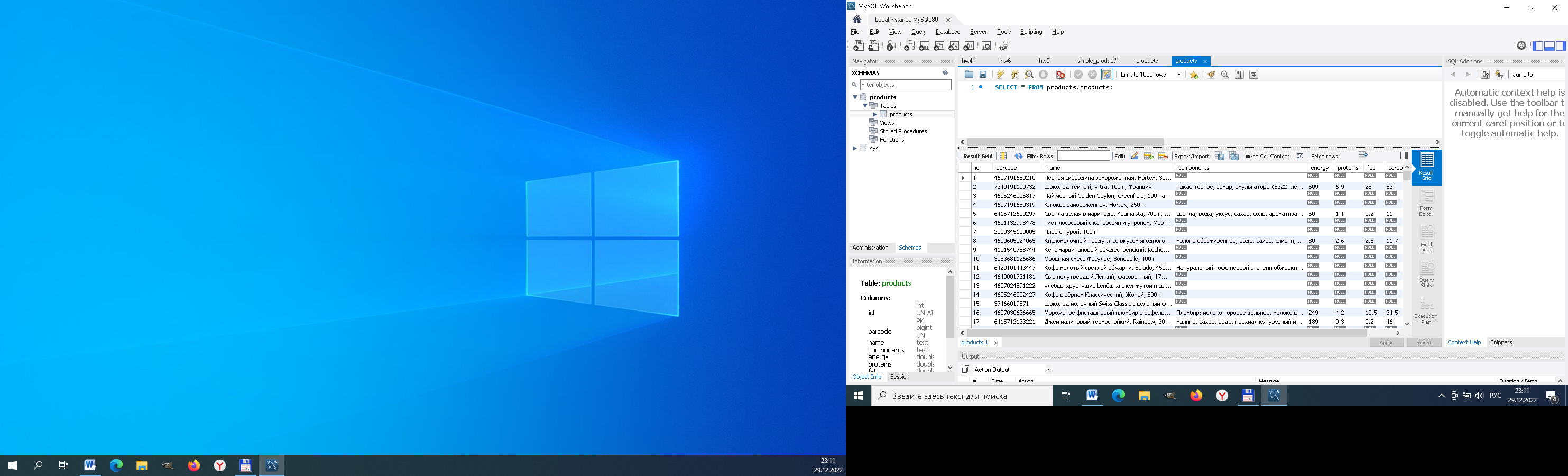
Цель оптимизации состоит в обеспечении как можно более быстрого получения ответа на запрос с минимальным числом обращений к БД  
Существует два типа оптимизации на языке SQL  
-оптимизация по синтаксису;  
-оптимизация по затратам;  
Оптимизация по синтаксису использует тот факт, что в языке SQL эффективность запроса зависит от того, как он сформулирован. В данном случае оптимизация зависит от квалификации программиста. При оптимизации по затратам происходит сбор сведений о БД — числе таблиц, числе строк, типе данных в каждой строке, доступности индексирования для конкретного столбца и т. д. Оптимизатор использует эту информацию для выработки наилучшего плана обработки запросов.  
Преимущества метода оптимизации по затратам: задача определения наилучшего способа выполнения запроса перекладывается с пользователя на процессор БД.  
Недостаток: нахождение оптимального метода само по себе может занять много времени.

После запуска создаем базу, в которой будем разворачивать из дампа сохраненную ранее бд.

Из базы открываем файл дампа и видим, что это текстовый файл с sql кодом. Выполняем код и видим, что база развернулась

Рис. 4.1. SQL база, с которой можно работать





# Глава 5. Анализ базы MongoDB

Открываем Mongo. Нам нужно выбрать объекты, которые соответствуют критерию:

- наличие поля "ingredients\_text\_ru"

{"ingredients\_text\_ru" : {$exists:true}}

Выбрать коды, начинающиеся не с 200. Будем использовать регулярные выражения [**https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html**](https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html)

## 5.1. Обзор регулярных выражений

Регулярное выражение задает шаблон поиска, используя метасимволы (которые являются операторами или принадлежат им) и символьные литералы (описаны в [**https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html**](https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html).).

Схема поиска может быть сложной. Например, это регулярное выражение соответствует любой строке, начинающейся с f или ht, за которой следует tp, необязательно за которой следует s, после которой следует двоеточие (:):

(f | ht)tps?:

Метасимволами (которые также являются операторами) в предыдущем примере являются круглые скобки, символ канала (|) и знак вопроса (?). Символьными литералами являются f, ht, tp s, : и двоеточие (,,).

Круглые скобки группируют несколько элементов шаблона в один элемент. Символ канала (|) указывает на выбор между элементами по обе стороны от него, f и ht. Знак вопроса (?) указывает, что предыдущий элемент s является необязательным. Таким образом, предыдущее регулярное выражение соответствует этим строкам:

* http:
* https:
* ftp:
* ftps:

Регулярные выражения являются мощным компонентом языков программирования Java и PERL для обработки текста. Например, скрипт PERL может считывать содержимое каждого HTML-файла в каталоге в единственную строковую переменную, а затем использовать регулярное выражение для поиска URL-адресов в этой строке. Эта надежная функциональность сопоставления с шаблоном является одной из причин, по которой многие разработчики приложений используют PERL.

Поддержка регулярных выражений базой данных позволяет разработчикам приложений реализовывать сложную логику сопоставления с образцом в базе данных, что полезно по этим причинам:

* Централизуя логику сопоставления с образцом в базе данных, вы избегаете интенсивной обработки строк наборов результатов SQL приложениями среднего уровня.

Функции регулярных выражений SQL приближают логику обработки к данным, тем самым обеспечивая более эффективное решение.

* Используя регулярные выражения на стороне сервера для обеспечения соблюдения ограничений, вы избегаете дублирования логики проверки на нескольких клиентах.

MongoDB предоставляет функциональность для поиска шаблона в строке во время запроса путем написания [регулярного выражения](https://translated.turbopages.org/proxy_u/en-ru.ru.5783680f-6499a5ef-d37e30e6-74722d776562/https/www.geeksforgeeks.org/write-regular-expressions/). Регулярное выражение - это обобщенный способ сопоставления шаблонов с последовательностями символов. MongoDB использует Perl-совместимые регулярные выражения (PCRE) версии 8.42 вместе с поддержкой UTF-8. В MongoDB мы можем выполнять сопоставление с шаблоном двумя различными способами:

* С оператором $regex
* Без оператора $regex

## Сопоставление с шаблоном с помощью оператора $regex

Этот оператор предоставляет возможности регулярного выражения для сопоставления с образцом в запросах. Или, другими словами, этот оператор используется для поиска заданной строки в указанной коллекции. Это полезно, когда мы не знаем точного значения поля, которое ищем в документе. Например, коллекция, содержащая 3 документа, т.е.,

{

name: "Tony",

position: "Backend developer"

}

{

name: "Bruce",

position: "frontend developer"

}

{

name: "Nick",

position: "HR Manager"

}

и мы ищем информацию о разработчике. Итак, с помощью оператора $regex мы создаем шаблон (т.Е. {position: {$regex: “developer”}}), который будет возвращать только те документы, которые содержат строку разработчика.

**Важные моменты:**

* Вам не разрешается использовать оператор $regex внутри оператора $in.
* Если вы хотите добавить регулярное выражение внутри списка условий запроса, разделенного запятыми, то вам придется использовать оператор $regex .
* Если вы хотите использовать опции x и s, то вам придется использовать выражение оператора $ regex с $options.
* Начиная с последней версии MongoDB (т. Е. 4.0.7), вам разрешено использовать оператор $not с выражением оператора $regex.
* Для запросов регулярных выражений с учетом регистра, если доступен индекс указанного поля, MongoDB сопоставляет регулярное выражение со значениями в индексе. Это самый простой способ сопоставления, а не сканирования всех коллекций на предмет запросов регулярных выражений без учета регистра, они неэффективно используют индекс.
* Если вы хотите использовать регулярные выражения, совместимые с Perl, поддерживающие регулярные выражения, которые не поддерживаются в JavaScript, тогда вы должны использовать оператор $regex .

**Синтаксис:**

{ <field>: { $regex: /pattern/, $options: '<options>' } }

{ <field>: { $regex: 'pattern', $options: '<options>' } }

**параметры $:**

В MongoDB доступны следующие <параметры> для использования с регулярным выражением:

* **я:** Чтобы соответствовать шаблону как в нижнем, так и в верхнем регистре в строке.
* **m:** Включить ^ и $ в шаблон при сопоставлении, т. е. специально выполнить поиск ^ и $ внутри строки. Без этой опции эти привязки совпадают в начале или конце строки.
* **x:** Игнорировать все пробелы в шаблоне $regex.
* **s:** Разрешить символу точки “.” соответствовать всем символам, включая символы новой строки.

## Сопоставление с шаблоном без использования оператора $regex

В MongoDB мы можем выполнять сопоставление с шаблоном без использования оператора $regex . Просто используя объект regular expression для указания регулярного выражения. Кроме того, используя объект регулярного выражения, вам разрешается использовать регулярное выражение внутри оператора $ in.

**Синтаксис:**

{ <field>: /pattern/<options> }

Здесь // означает указать ваши критерии поиска между этими разделителями.

## 5.2. Используемое регулярное выражение:

Нам нужно выбрать объекты, которые соответствуют критерию:

- наличие поля "ingredients\_text\_ru"

{"ingredients\_text\_ru" : {$exists:true}}

коды, начинающиеся не с 200

{

$and : [

"ingredients\_text\_ru" : {$exists:true},

"code" : {$exists:true},

"code" : {$not : {$regex : "^200"}}

]

}

# Глава 6. Выбор технологий создания скрипта для выбора и записи данных их MomgoDB в SQL

Нужно написать cкрипт, который будет вынимать из базы mongо нужные данные и записывать в базу SQL

Будем использовать библиотеку Pymongo [**https://pymongo.readthedocs.io/en/stable/**](https://pymongo.readthedocs.io/en/stable/)

## 6.1. Создание виртуального окружения.

Виртуальное окружение в Python — способ изолировать зависимости (пакеты) для определённого проекта.

## Создание виртуального окружения

Создаётся через модуль venv, который идёт в поставке Python 3.

Используется команда "python −m venv" и название директории, в которой будет создано виртуальное окружение.

mkdir project

cd project

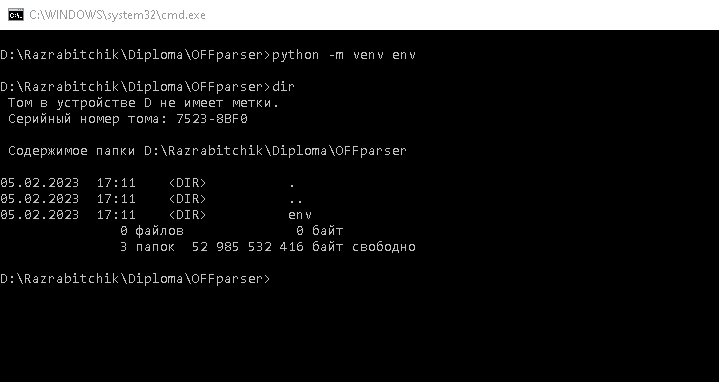
python -m venv env

## Использование:

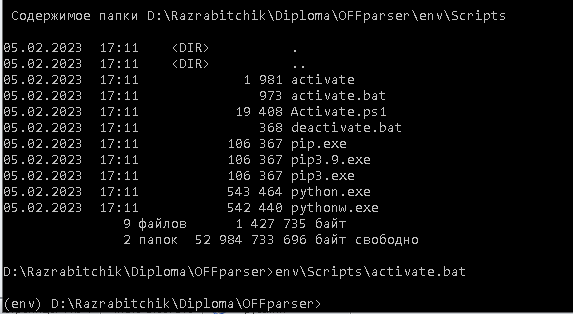
Для активации окружение надо запустить скрипт activate.bat внутри директории с виртуальным окружением. Затем выполнить необходимые операции, допустим установить пакет. По окончанию работы запусить скрипт deactivate.bat

Создаем виртуальное окружение для нашего проекта

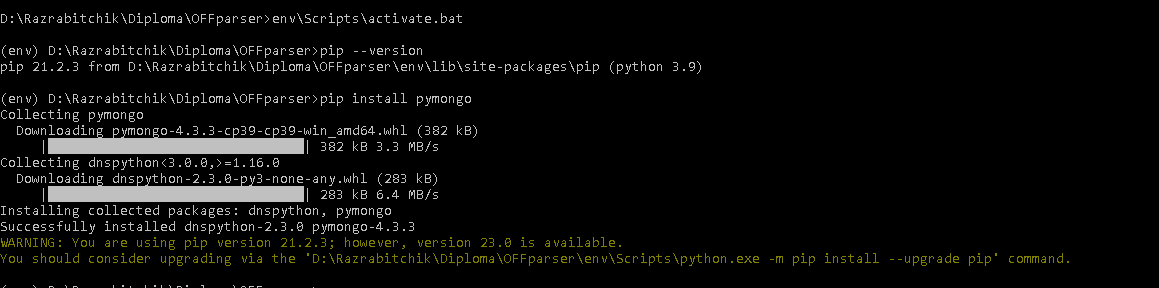
<https://tyapk.ru/blog/post/python-virtual-environment-windows>



Запускаем скрипт, активирующий виртуальное окружение



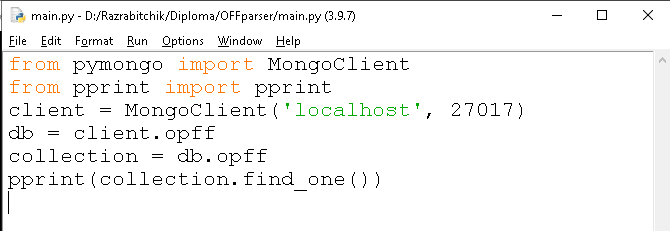
Устанавливаем библиотеку

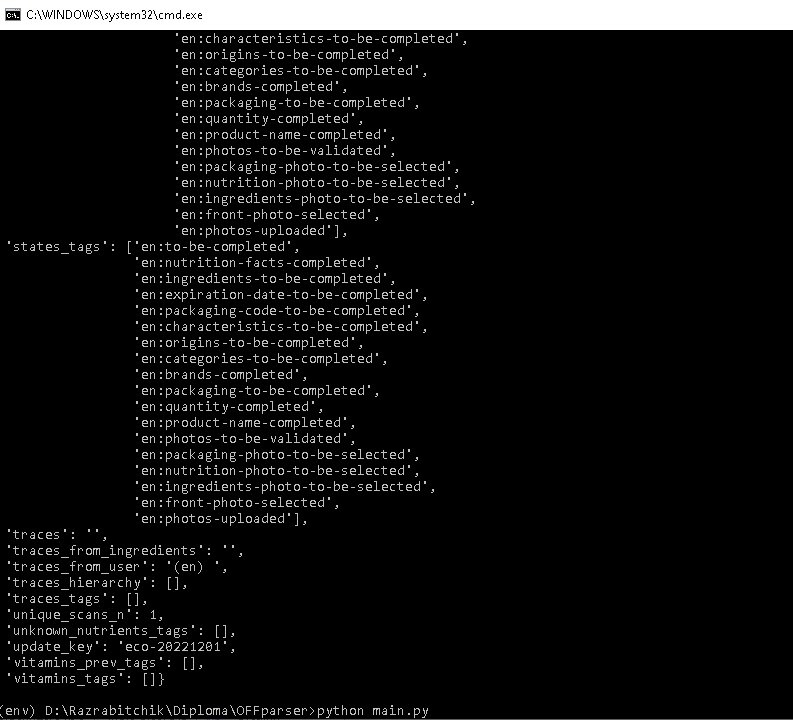


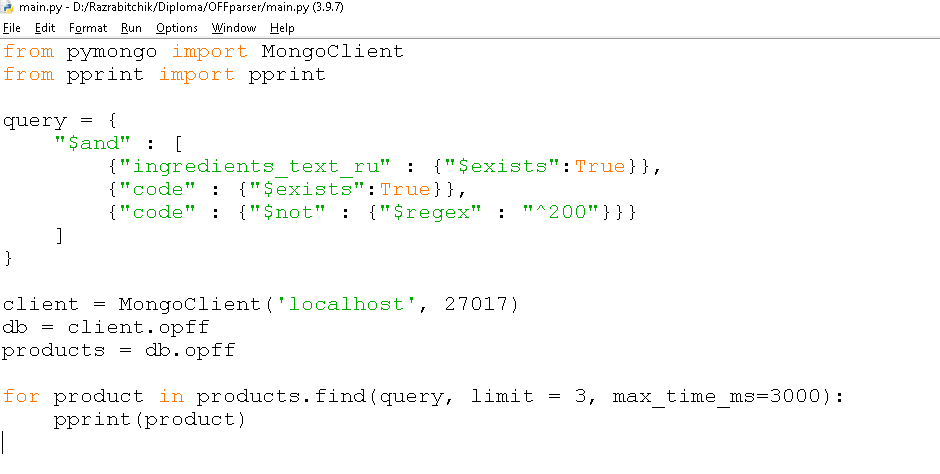
У нас запущен демон MongoDB, пытаемся подключиться используя MongoClient

from pymongo import MongoClient

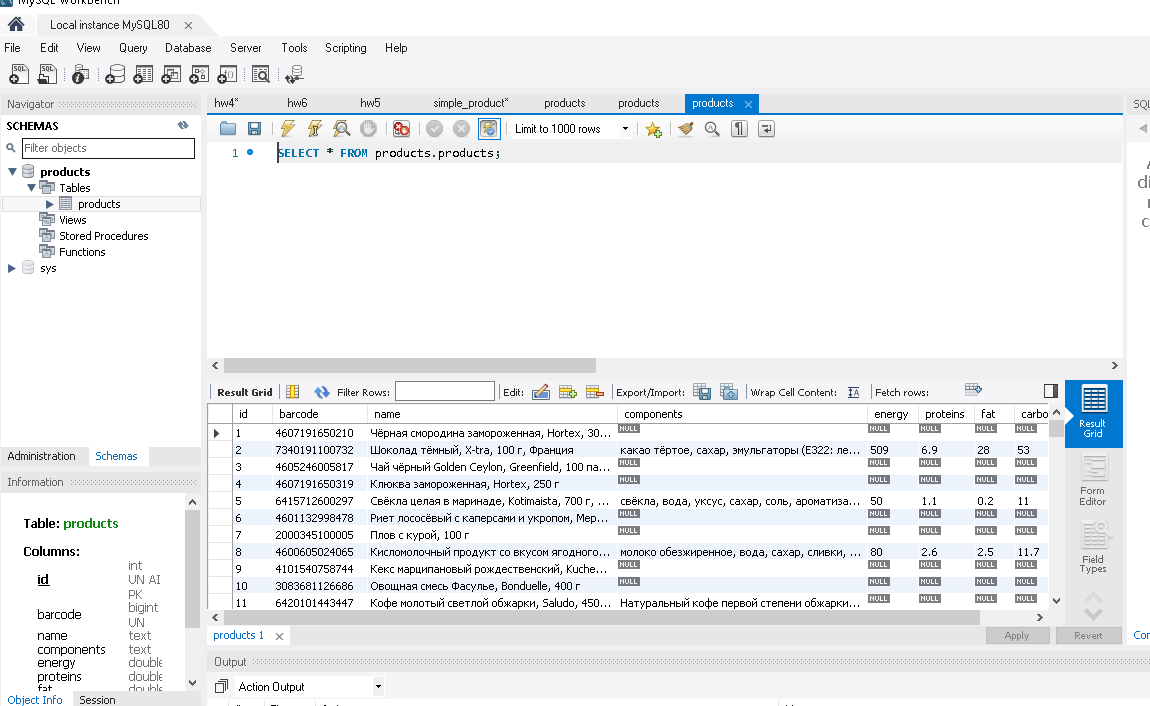
client = MongoClient('localhost', 27017**)**



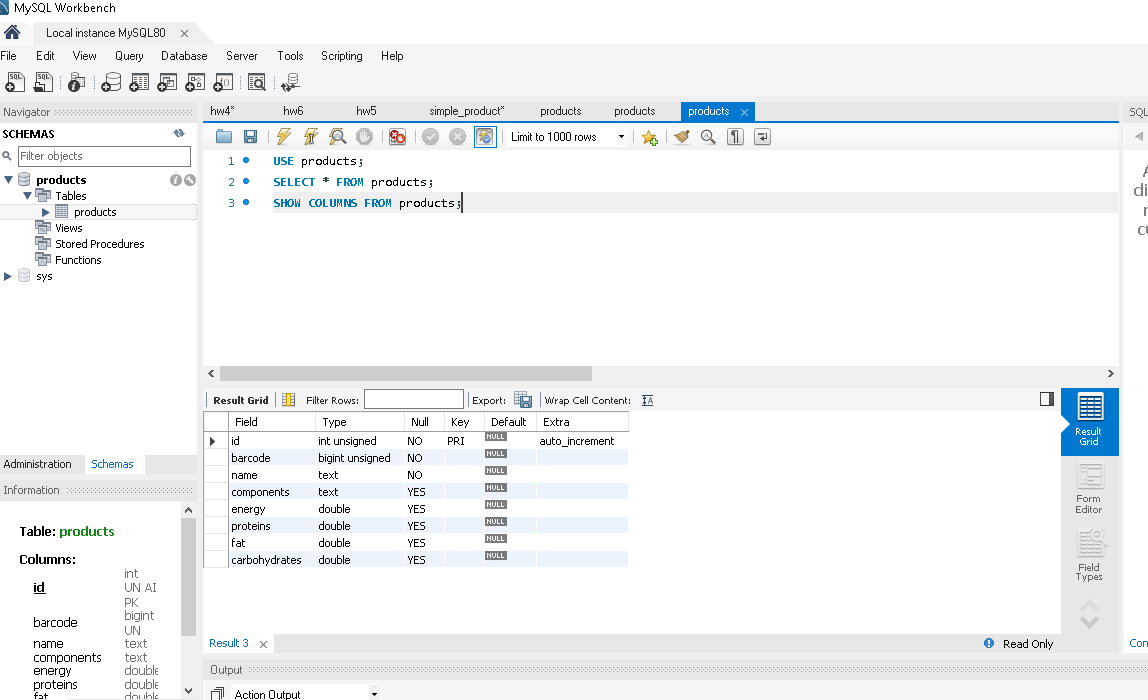
***Рисунок 6.1. Результат подключения к баз MongoDB.*** 



## 6.2. Подключаемся к базе MySQL



***Рис. 6.2. Структура базы MYSQL***



MySQL состоит из двух частей: серверной и клиентской.   
**Клиентская**

Клиентская может состоять, например, из **скрипта PHP**, который будет принимать, обрабатывать, отсылать или, наоборот, брать данные из БД.  
**Серверная (структура данных)**

**База данных** - это основа. Физически файлы БД и таблиц имеют расширения frm, MYD, MYI.  
В каждой БД есть одна или несколько **таблиц**. Особо структурированные данные с полями и записями.  
В таблице в свою очередь есть одна и более **записей**.

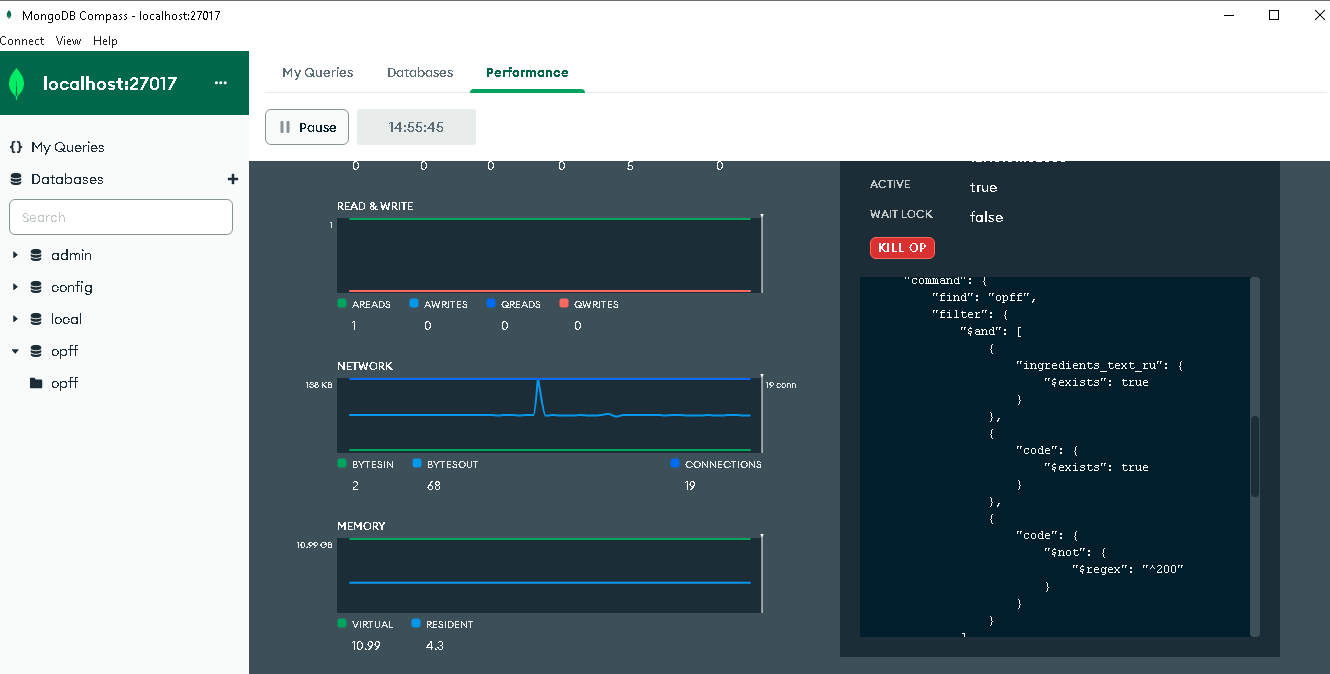
## Базы данных, таблицы, поля, строки в MySQL

С тем, что такое База Данных мы уже разобрались. Рассмотрим подпункты. Как говорилось выше - БД состоит из **таблиц**. Таблицы, как правило, содержат в себе тематическую информацию

Таблицы состоят из **полей**.

А вот дальше идут **записи**. Запись - это строка в таблице, где каждая отдельная ячейка содержит значение соответствующее определённому полю.

## 6.3. Реализация запроса к Mongodb



Как было сказано выше, mongo явялется noSQL базой данных, части полей может просто не быть. Мы вынуждены каждый раз проверять их наличие.Будем делать это при помощи обработки исключений в Python. <https://www.w3schools.com/python/python_try_except.asp>

## 6.4. Обработка исключений

Программа, написанная на Python, останавливается сразу как обнаружит ошибку. Ошибки могут быть (как минимум) двух типов:

* Синтаксические ошибки — возникают, когда написанное выражение не соответствует правилам языка (например, написана лишняя скобка);
* Исключения — возникают во время выполнения программы (например, при делении на ноль).

Синтаксические ошибки исправить просто (если вы используете IDE, он их подсветит). А вот с исключениями всё немного сложнее — не всегда при написании программы можно сказать возникнет или нет в данном месте исключение. Чтобы приложение продолжило работу при возникновении проблем, такие ошибки нужно перехватывать и обрабатывать с помощью блока try/except.

## Как устроен механизм исключений

В Python есть встроенные исключения, которые появляются после того как приложение находит ошибку. В этом случае текущий процесс временно приостанавливается и передает ошибку на уровень вверх до тех пор, пока она не будет обработано. Если ошибка не будет обработана, программа прекратит свою работу (а в консоли мы увидим Traceback с подробным описанием ошибки).

💁‍♂️ Пример: напишем скрипт, в котором функция ожидает число, а мы передаём сроку (это вызовет исключение "TypeError"):

def b(value):

print("-> b")

print(value + 1) # ошибка тут

def a(value):

print("-> a")

b(value)

a("10")

> -> a

> -> b

> Traceback (most recent call last):

> File "test.py", line 11, in <module>

> a("10")

> File "test.py", line 8, in a

> b(value)

> File "test.py", line 3, in b

> print(value + 1)

> TypeError: can only concatenate str (not "int") to str

В данном примере мы запускаем файл "test.py" (через консоль). Вызывается функция "a", внутри которой вызывается функция "b". Все работает хорошо до сточки print(value + 1). Тут интерпретатор понимает, что нельзя конкатенировать строку с числом, останавливает выполнение программы и вызывает исключение "TypeError".

Далее ошибка передается по цепочке в обратном направлении: "b" → "a" → "test.py". Так как в данном примере мы не позаботились обработать эту ошибку, вся информация по ошибке отобразится в консоли в виде Traceback.

Traceback (трассировка) — это отчёт, содержащий вызовы функций, выполненные в определенный момент. Трассировка помогает узнать, что пошло не так и в каком месте это произошло.

Traceback лучше читать снизу вверх ↑

Пример Traceback в Python

В нашем примере Traceback содержится следующую информацию (читаем снизу вверх):

1. TypeError — тип ошибки (означает, что операция не может быть выполнена с переменной этого типа);
2. can only concatenate str (not "int") to str — подробное описание ошибки (конкатенировать можно только строку со строкой);
3. Стек вызова функций (1-я линия — место, 2-я линия — код). В нашем примере видно, что в файле "test.py" на 11-й линии был вызов функции "a" со строковым аргументом "10". Далее был вызов функции "b". print(value + 1) это последнее, что было выполнено — тут и произошла ошибка.
4. most recent call last — означает, что самый последний вызов будет отображаться последним в стеке (в нашем примере последним выполнился print(value + 1)).

В Python ошибку можно перехватить, обработать, и продолжить выполнение программы — для этого используется конструкция try ... except ....

## Как обрабатывать исключения в Python (try except)

В Python исключения обрабатываются с помощью блоков try/except. Для этого операция, которая может вызвать исключение, помещается внутрь блока try. А код, который должен быть выполнен при возникновении ошибки, находится внутри except.

Например, вот как можно обработать ошибку деления на ноль:

try:

a = 7 / 0

except:

print('Ошибка! Деление на 0')

Здесь в блоке try находится код a = 7 / 0 — при попытке его выполнить возникнет исключение и выполнится код в блоке except (то есть будет выведено сообщение "Ошибка! Деление на 0"). После этого программа продолжит свое выполнение.

[PEP 8](https://www.python.org/dev/peps/pep-0008/) рекомендует, по возможности, указывать конкретный тип исключения после ключевого слова except (чтобы перехватывать и обрабатывать конкретные исключения):

try:

a = 7 / 0

except ZeroDivisionError:

print('Ошибка! Деление на 0')

Однако если вы хотите перехватывать все исключения, которые сигнализируют об ошибках программы, используйте тип исключения Exception:

try:

a = 7 / 0

except Exception:

print('Любая ошибка!')

### As — сохраняет ошибку в переменную

Перехваченная ошибка представляет собой объект класса, унаследованного от "BaseException". С помощью ключевого слова as можно записать этот объект в переменную, чтобы обратиться к нему внутри блока except:

try:

file = open('ok123.txt', 'r')

except FileNotFoundError as e:

print(e)

> [Errno 2] No such file or directory: 'ok123.txt'

В примере выше мы обращаемся к объекту класса "FileNotFoundError" (при выводе на экран через print отобразится строка с полным описанием ошибки).

У каждого объекта есть поля, к которым можно обращаться (например если нужно логировать ошибку в собственном формате):

import datetime

now = datetime.datetime.now().strftime("%d-%m-%Y %H:%M:%S")

try:

file = open('ok123.txt', 'r')

except FileNotFoundError as e:

print(f"{now} [FileNotFoundError]: {e.strerror}, filename: {e.filename}")

> 20-11-2021 18:42:01 [FileNotFoundError]: No such file or directory, filename: ok123.txt

### Finally — выполняется всегда

При обработке исключений можно после блока try использовать блок finally. Он похож на блок except, но команды, написанные внутри него, выполняются обязательно. Если в блоке try не возникнет исключения, то блок finally выполнится так же, как и при наличии ошибки, и программа возобновит свою работу.

Обычно try/except используется для перехвата исключений и восстановления нормальной работы приложения, а try/finally для того, чтобы гарантировать выполнение определенных действий (например, для закрытия внешних ресурсов, таких как ранее открытые файлы).

В следующем примере откроем файл и обратимся к несуществующей строке:

file = open('ok.txt', 'r')

try:

lines = file.readlines()

print(lines[5])

finally:

file.close()

if file.closed:

print("файл закрыт!")

> файл закрыт!

> Traceback (most recent call last):

> File "test.py", line 5, in <module>

> print(lines[5])

> IndexError: list index out of range

Даже после исключения "IndexError", сработал код в секции finally, который закрыл файл.

p.s. данный пример создан для демонстрации, в реальном проекте для работы с файлами лучше использовать менеджер контекста with.

Также можно использовать одновременно три блока try/except/finally. В этом случае:

* в try — код, который может вызвать исключения;
* в except — код, который должен выполниться при возникновении исключения;
* в finally — код, который должен выполниться в любом случае.

def sum(a, b):

res = 0

try:

res = a + b

except TypeError:

res = int(a) + int(b)

finally:

print(f"a = {a}, b = {b}, res = {res}")

sum(1, "2")

> a = 1, b = 2, res = 3

### Else — выполняется когда исключение не было вызвано

Иногда нужно выполнить определенные действия, когда код внутри блока try не вызвал исключения. Для этого используется блок else.

Допустим нужно вывести результат деления двух чисел и обработать исключения в случае попытки деления на ноль:

b = int(input('b = '))

c = int(input('c = '))

try:

a = b / c

except ZeroDivisionError:

print('Ошибка! Деление на 0')

else:

print(f"a = {a}")

> b = 10

> c = 1

> a = 10.0

В этом случае, если пользователь присвоит переменной "с" ноль, то появится исключение и будет выведено сообщение "'Ошибка! Деление на 0'", а код внутри блока else выполняться не будет. Если ошибки не будет, то на экране появятся результаты деления.

**Несколько блоков except**

В программе может возникнуть несколько исключений, например:

1. Ошибка преобразования введенных значений к типу float ("ValueError");
2. Деление на ноль ("ZeroDivisionError").

В Python, чтобы по-разному обрабатывать разные типы ошибок, создают несколько блоков except:

try:

b = float(input('b = '))

c = float(input('c = '))

a = b / c

except ZeroDivisionError:

print('Ошибка! Деление на 0')

except ValueError:

print('Число введено неверно')

else:

print(f"a = {a}")

> b = 10

> c = 0

> Ошибка! Деление на 0

> b = 10

> c = питон

> Число введено неверно

Теперь для разных типов ошибок есть свой обработчик.

**Несколько типов исключений в одном блоке except**

Можно также обрабатывать в одном блоке except сразу несколько исключений. Для этого они записываются в круглых скобках, через запятую сразу после ключевого слова except. Чтобы обработать сообщения "ZeroDivisionError" и "ValueError" в одном блоке записываем их следующим образом:

try:

b = float(input('b = '))

c = float(input('c = '))

a = b / c

except (ZeroDivisionError, ValueError) as er:

print(er)

else:

print('a = ', a)

При этом переменной er присваивается объект того исключения, которое было вызвано. В результате на экран выводятся сведения о конкретной ошибке.

**Raise — самостоятельный вызов исключений**

Исключения можно генерировать самостоятельно — для этого нужно запустить оператор raise.

min = 100

if min > 10:

raise Exception('min must be less than 10')

> Traceback (most recent call last):

> File "test.py", line 3, in <module>

> raise Exception('min value must be less than 10')

> Exception: min must be less than 10

Перехватываются такие сообщения точно так же, как и остальные:

min = 100

try:

if min > 10:

raise Exception('min must be less than 10')

except Exception:

print('Моя ошибка')

> Моя ошибка

Кроме того, ошибку можно обработать в блоке except и пробросить дальше (вверх по стеку) с помощью raise:

min = 100

try:

if min > 10:

raise Exception('min must be less than 10')

except Exception:

print('Моя ошибка')

raise

> Моя ошибка

> Traceback (most recent call last):

> File "test.py", line 5, in <module>

> raise Exception('min must be less than 10')

> Exception: min must be less than 10

### Как пропустить ошибку

Иногда ошибку обрабатывать не нужно. В этом случае ее можно пропустить с помощью pass:

try:

a = 7 / 0

except ZeroDivisionError:

pass

## 6.5. Коннектор SQL Python

Большинство приложений в той или иной форме взаимодействует с данными. Поэтому языки программирования (Python не исключение), предоставляют инструменты хранения источников данных и доступа к ним

## Установка MySQL Connector/Python

Драйвер базы данных — программное обеспечение, позволяющее приложению подключаться и взаимодействовать с СУБД. Такие драйверы обычно поставляются в виде отдельных модулей. Сандартный интерфейс, которому должны соответствовать все драйверы баз данных Python, описан в [PEP 249](https://www.python.org/dev/peps/pep-0249/). Драйверы баз данных Python, такие как [sqlite3](https://docs.python.org/3/library/sqlite3.html) для SQLite, [psycopg](https://www.psycopg.org/docs/) для PostgreSQL и [MySQL Connector/Python](https://github.com/mysql/mysql-connector-python) для MySQL, следуют этим правилам.

Для установки драйвера (коннектора) воспользуемся менеджером пакетов pip:

pip install mysql-connector-python

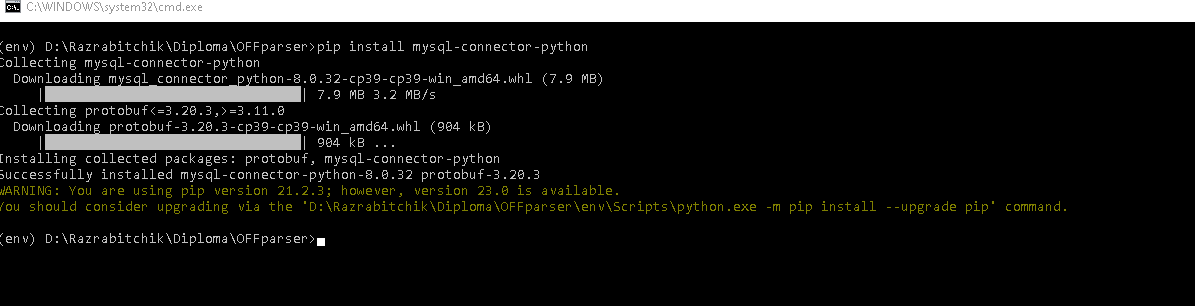
pip установит коннектор в текущую активную среду. Чтобы работать с проектом изолированным образом, мы рекомендуем [настроить виртуальную среду](https://proglib.io/p/kak-sozdat-virtualnoe-okruzhenie-v-python-i-perestat-dumat-o-zavisimostyah-2020-03-10).

Проверим результат установки, запустив в терминале Python следующую команду:

import mysql.connector

Если инструкция импорта выполняется без ошибок, значит mysql.connector успешно установлен и готов к использованию.

Устанавливаем коннектор SQL python

[**https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html**](https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html)

# Подключаемся к серверу MySQL из Python

Чтобы установить соединение, используем connect() из модуля mysql.connector. Эта функция принимает параметры host, user и password, а возвращает объект MySQLConnection. Учетные данные можно получить в результате ввода от пользователя:

from getpass import getpass

from mysql.connector import connect, Error

try:

with connect(

host="localhost",

user=input("Имя пользователя: "),

password=getpass("Пароль: "),

) as connection:

print(connection)

except Error as e:

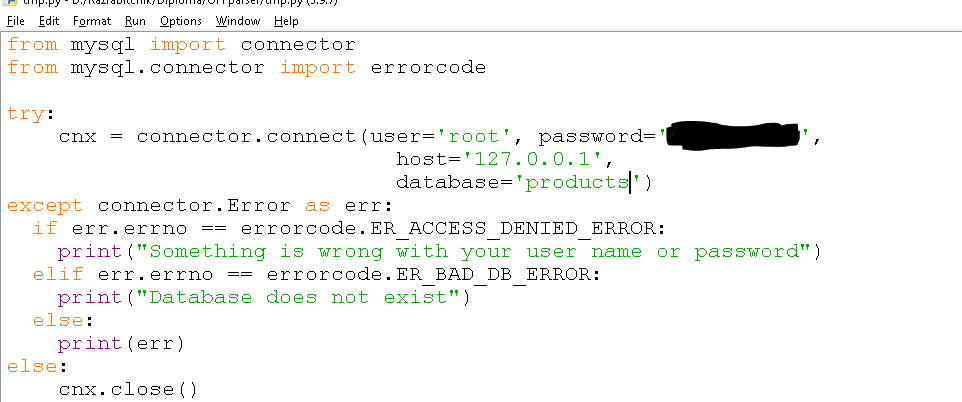
print(e)

Объект MySQLConnection хранится в переменной connection, которую мы будем использовать для доступа к серверу MySQL. Несколько важных моментов:

* Все соединения с базой данных оборачивайтев блоки try ... except. Так будет проще перехватить и изучить любые исключения.
* Не забывайте закрывать соединение после завершения доступа к базе данных. Неиспользуемые открытые соединения приводят к неожиданным ошибкам и проблемам с производительностью. В коде для этого используется диспетчер контекста (with ... as ...).
* Никогда не следует встраивать учетные данные (имя пользователя и пароль) в строковом виде в скрипт Python. Это плохая практика для развертывания, которая представляет серьезную угрозу безопасности. Приведенный код запрашивает для входа учетные данные. Для этого используется встроенный модуль getpass, чтобы скрыть вводимый пароль. Хотя это лучше, чем жесткое кодирование, но есть и другие, более безопасные способы хранения конфиденциальной информации, например, использование переменных среды.

Итак, мы установили соединение между нашей программой и сервером MySQL.

Обрабатываем исключения



# Глава 7 Скрипт для выбора и записи данных их MomgoDB в SQL

from pymongo import MongoClient

from pprint import pprint

from mysql import connector

from mysql.connector import errorcode

MySQL\_config = {

'user': 'root',

'password': 'Letme4682!',

'host': '127.0.0.1',

'database': 'products',

'raise\_on\_warnings': True

}

mongo\_query = {

"$and" : [

{"ingredients\_text\_ru" : {"$exists":True}},

{"code" : {"$exists":True}},

{"code" : {"$not" : {"$regex" : "^200"}}}

]

}

add\_product = ("INSERT INTO products "

"(barcode, name, components, energy, proteins, fat, carbohydrates) "

"VALUES (%(barcode)s, %(name)s, %(components)s, "

"%(energy)s, %(proteins)s, %(fat)s, %(carbohydrates)s)")

def get\_data\_for\_SQL(product):

data = {}

data["name"] = product.get("product\_name\_ru")

data["barcode"] = product.get("code")

data["components"] = product.get("ingredients\_text\_ru")

try:

nutriments = product.nutriments

data["proteins"] = nutrimnets.get("proteins\_100g")

data["fat"] = nutrimnets.get("fat\_100g")

data["carbohydrates"] = nutrimnets.get("carbohydrates\_100g")

data["energy"] = nutrimnets.get("energy\_100g")

except:

data["proteins"] = None

data["fat"] = None

data["carbohydrates"] = None

data["energy"] = None

return data

try:

# connect to mongo

client = MongoClient('localhost', 27017)

db = client.opff

products = db.opff

#connect to MySQL

cnx = connector.connect(\*\*MySQL\_config)

cursor = cnx.cursor()

for product in products.find(mongo\_query, limit = max\_time\_ms=7200000):

data = get\_data\_for\_SQL(product)

pprint(data)

# Insert new products

cursor.execute(add\_product, data)

cnx.commit()

cursor.close()

except connector.Error as err:

if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:

print("Something is wrong with your user name or password")

elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:

print("Database does not exist")

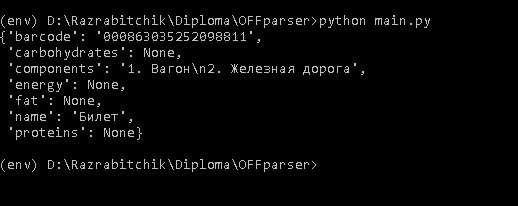
else:

print(err)

else:

cnx.close()

Проверяем работу main.py на примере 1элемента



# Заключение

В процессе работы над Дипломным проектом нам удалось долить необходимые данные из базы MongoDB в существующую SQL базу

В процессе работы я познакомилась с работой no SQL базы, развернула 2 базы на сервере, написала скрипт на python, который корректно отработал и дописал в базу необходимые мне значения.

# Источники и ресурсы:

1. Рабочий репозиторий: <https://github.com/MarinaFedorovaok/BD.git>
2. Приложение [https://github.com/MarinaFedorovaok/goodmeals](https://checklink.mail.ru/proxy?es=qXE34OmEFBUq%2F2PHzaBVnhBKNM7xPF08N3ouAOiz9cM%3D&egid=SilSqvTEVtT36tVrlRQH8GEPCWHRzTEpdx68qxmiCtc%3D&url=https%3A%2F%2Fclick.mail.ru%2Fredir%3Fu%3Dhttps%253A%252F%252Fgithub.com%252FMarinaFedorovaok%252Fgoodmeals%26c%3Dswm%26r%3Dhttp%26o%3Dmail%26v%3D3%26s%3D9b2399bf58817c4f&uidl=16724002230225527156&from=diplom%40geekbrains.ru&to=fedma001%40mail.ru&email=fedma001%40mail.ru)
3. <https://world.openfoodfacts.org/data>
4. <https://www.mongodb.com/try/download/community>
5. <https://www.mongodb.com/try/download/database-tools>
6. <https://metanit.com/nosql/mongodb/2.4.php>
7. <https://dev.mysql.com/downloads/mysql/>
8. <https://www.mysql.com/products/workbench/>
9. <https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html>
10. <https://pymongo.readthedocs.io/en/stable/>
11. [https://tyapk.ru/blog/post/python-virtual-environmen t-windows](https://tyapk.ru/blog/post/python-virtual-environmen%09t-windows)

<https://www.w3schools.com/python/python_try_except.asp>

1. <https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html>
2. Oracle.com