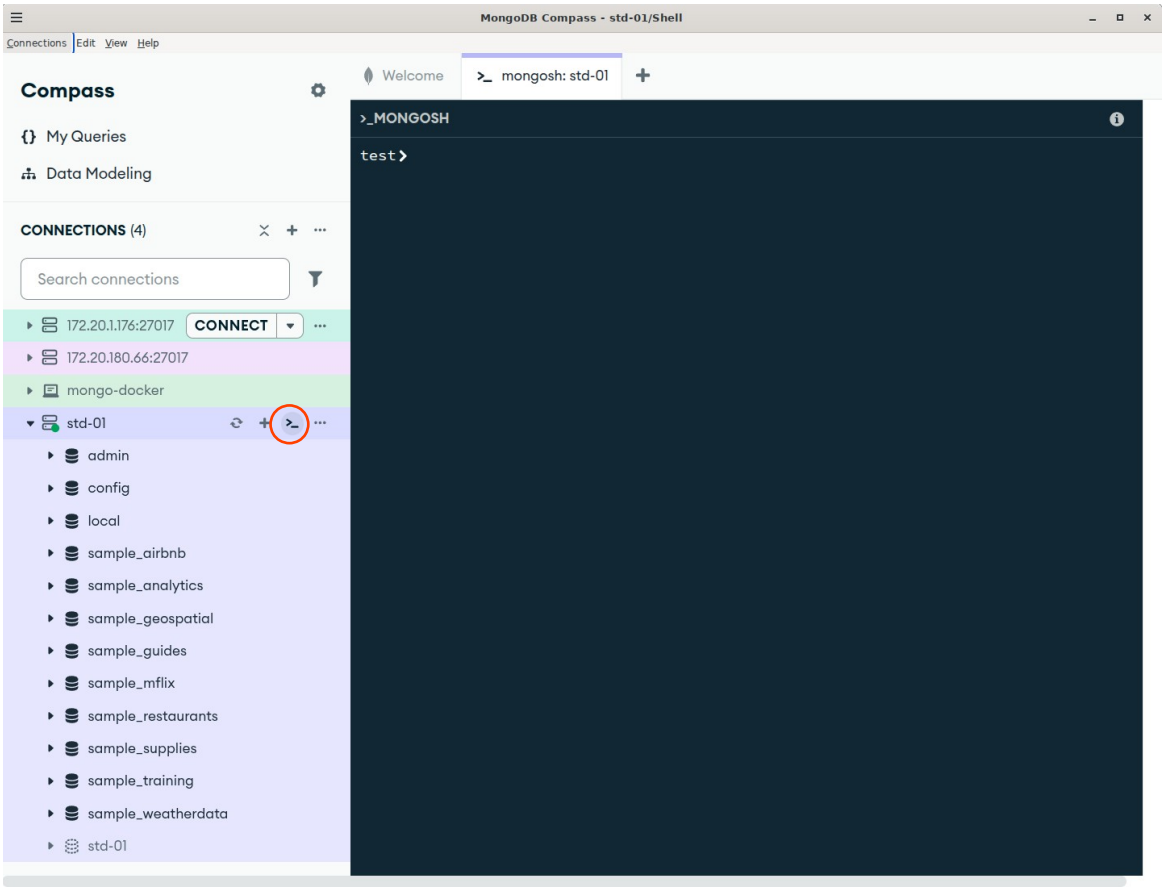
Лабораторная работа № 1.

В данной работе представлена пошаговая инструкция по работе с MongoDB. Вы будете использовать клиентское приложение *MongoDB Compass* для создания базы данных, добавления коллекции, вставки и удаления документов и выполнения базовых CRUD-операций по управлению данными.

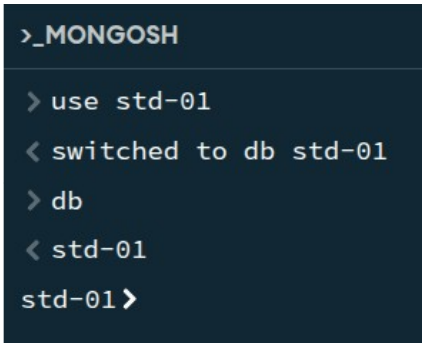
1. Введение.

Убедитесь, что на вашем компьютере имеется приложение *MongoDB Compass* и запустите его. Добавьте новое соединение: адрес сервера 172.20.1.176, порт 27017, имя соединения указываете любое, на вкладке Advanced Connection Options/Authentication указываете имя пользователя и пароль, которые получили у преподавателя. В поле Authentication Database указываете имя вашей базы данных, которое *совпадает с вашим логином*. Для этой базы у вас будут максимальные привилегии, к остальным — доступ только на чтение. Если компьютер является общедоступным, созданный вами аккаунт может быть быстро скомпрометирован, поэтому в конце занятия *имеет смысл его удалить*, чтобы никто не смог случайно или намеренно повредить результаты вашей работы.

Если вы всё сделали правильно, после нажатия кнопки "*Save & Connect*" установится соединение с базой данных. В левой панели окна нажмите на кнопку "*Open MongoDB Shell*" (показана красным кружком на скриншоте), в результате запустится терминальное окно *mongosh*, в котором вы и будете работать.



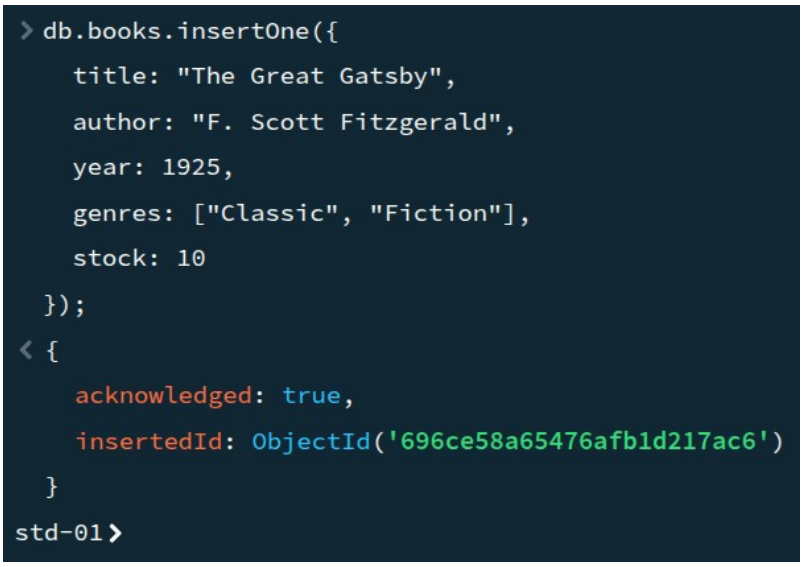
Подсказка **test>** указывает, что вы в настоящее время подключены к базе данных по умолчанию. Чтобы переключиться на вашу базу данных, используйте команду **use** *имяБД*. Ваша база данных в левой панели окна показана серым цветом, поскольку в ней нет ещё ни одной коллекции. В данном тексте работа с MongoDB будет проиллюстрирована от лица пользователя с логином *std-01*.



Если на компьютере отсутствует (или нет возможности установить) приложение *MongoDB Compass*, используйте программу *mongosh*. Она инсталлируется даже под старые версии операционных систем и реализует исключительно интерфейс командной строки к СУБД MongoDB. Строка подключения *mongosh* к СУБД будет выглядеть следующим образом: *mongosh mongodb://имя:пароль@172.20.1.176:27017/имя*. Под словом "имя" понимается ваш логин, оно же — имя вашей базы данных.

2. Создание коллекции и вставка документа.

Коллекция — это группа документов MongoDB, примерно эквивалентная таблице в реляционной базе данных. Метод **insertOne();** добавляет один документ в коллекцию. Введите в коллекцию *books* информацию о книге так, как представлено на скриншоте, при этом коллекция будет создана автоматически.



Вставленный документ — это всегда BSON-объект, который представляет собой бинарно-кодированный текст в формате, очень похожем на JSON. После успешной вставки документа MongoDB возвращает информацию, подтверждающую операцию и содержащую уникальный

идентификатор (*_id*) только что вставленного документа. Посмотреть, какие коллекции существуют в базе данных, можно с помощью команды **show collections**.

Для одновременного добавления нескольких документов MongoDB предоставляет метод **insertMany()**. Это известно как операция пакетной вставки (bulk insert), которая сокращает количество сетевых обращений к базе данных. Добавьте ещё информацию о трёх книгах (три документа) в коллекцию *books*:

```
std-01> db.books.insertMany([
          {
            title: "1984",
            author: "George Orwell",
            year: 1949,
            genres: ["Dystopian", "Science Fiction"],
            stock: 15
          },
          {
            title: "To Kill a Mockingbird",
            author: "Harper Lee",
            year: 1960,
            genres: ["Classic", "Fiction"],
            stock: 5
          },
          {
            title: "Pride and Prejudice",
            author: "Jane Austen",
            year: 1813,
            genres: ["Romance", "Classic"],
            stock: 12
          }
        ]);
```

! Здесь и далее цветом и знаком восклицания выделены задания, которые должны быть сделаны самостоятельно, а полученные результаты представлены в отчёте по работе. Введите в коллекцию *book* ещё несколько документов, содержащих информацию о любых выбранных вами книгах так, чтобы общее количество книг было больше десяти.

Подсчитайте общее количество документов в коллекции с помощью команды
db.books.countDocuments();

3. Просмотр введённых данных.

Для этой цели MongoDB предоставляет мощный метод **find()**. Чтобы извлечь все документы из коллекции *books*, используйте метод **find()** без каких-либо аргументов:

```
db.books.find();
```

Чаще всего требуется находить документы, соответствующие определенным критериям. Например, чтобы найти все книги, опубликованные до 1950 года, вы можете использовать фильтр запроса с оператором **$lt** (меньше чем):

```
db.books.find({ year: { $lt: 1950 } });
```

Соответственно, условию «больше» соответствует конструкция **$gt**, «больше либо равно» — **$gte**, равно — **$eq**. Иногда нужно получить только определенные поля из документов, а не весь документ целиком. Это называется *проекцией*. Чтобы получить только *title* (название) и *author* (автор) всех книг, опубликованных до 1950 года, вы можете добавить документ проекции в качестве второго аргумента к *find():*

```
db.books.find({ "year": {$lt: 1950}}, {"title": 1, "author": 1, "_id": 0});
```

В описании проекции **1** означает "включить это поле", а **0** — "исключить это поле". По умолчанию поле _id всегда включается, поэтому мы явно исключаем его с помощью *_id: 0*.

! Создайте и проверьте синтаксис команды find(), с помощью которой можно вывести названия книг, написанных в жанре антиутопии.

4. Простое обеспечение корректности данных.

Один из способов обеспечить ее — предотвратить дублирование записей. Например, логично предположить, что в коллекции *book* не должно быть двух книг с одинаковым названием. Для этого создадим уникальный индекс по полю *title*.

```
db.books.createIndex({ title: 1 }, { unique: true });
```

После этого попробуем ввести в коллекцию документ-дубликат:

```
db.books.insertOne({
  title: "1984",
  author: "George Orwell",
```

```
  year: 1949,
  genres: ["Dystopian", "Classic"],
  stock: 20
});
```

Эта операция завершится неудачей. MongoDB выдаст MongoBulkWriteException с кодом ошибки E11000, который указывает на нарушение ограничения уникальности ключа. Это ожидаемое поведение подтверждает, что наш уникальный индекс работает правильно.

Попробуем ввести в коллекцию формально правильный документ, но содержащий ошибочную информацию, которую впоследствии придётся удалить либо исправить:

```
db.books.insertOne({
  title: "Странная книга",
  author: "И автор странный",
  year: 2026,
  genres: ["Classic"],
  stock: 1200
});
```

Удалим этот документ с помощью команды **deleteOne()**, используя в качестве фильтра (с целью минимизации возможных ошибок) значение *_id* этого документа:

```
> let doc = db.books.findOne({title: "Странная книга"})
> print(doc._id)
< ObjectId('696cfab065476afb1d217acc')
> db.books.deleteOne({_id: doc._id})
< {
    acknowledged: true,
    deletedCount: 1
  }
std-01>
```

Данный фрагмент кода показывает, что в терминале *mongosh* мы вполне можем использовать синтаксические конструкции языка JavaScript, который является основным языком для работы с СУБД MongoDB.

Помимо команды **deleteOne()**, команда **deleteMany();** может удалить несколько (или все) документы из коллекции. Используйте эти команды с осторожностью. Полностью удалить коллекцию можно с помощью команды **drop()**.

5. Импорт и обработка коллекции из другой базы данных.

Попробуем сделать копию коллекции из базы данных *sample_guides*, которая находится на том же сервере, что и ваша база данных, и к которой вы имеете доступ только на чтение данных. Для этого сначала перейдём в БД *sample_guides* и посмотрим список её доступных коллекций:

```
use sample_guides;

show collections;
```

Обнаружилась одна коллекция по имени *planet*, которую вы и скопируете в свою базу данных с помощью следующей команды:

```
db.planets.aggregate([{$out: {db: "std-01", coll: "planets"}}]);
```

Как вы понимаете, в данной команде вместо "*std-01*" вы указываете имя своей базы данных. Если импорт прошёл успешно, вернитесь обратно в свою базу данных и посмотрите результат.

```
sample_guides> use std-01;

std-01> show collections;
< books
< planets
```

! Полностью удалите из только что созданной коллекции *planets* описания тех планет, у которых средняя температура на поверхности выше 150 °C. Не забудьте в создаваемом выражении заключить иерархический путь в кавычки. Покажите полученный результат.

6. Создание схемы данных для контроля целостности данных.

No-SQL документная модель MongoDB отличается гибкой схемой представления данных: например, в одной коллекции могут быть документы самой разной структуры. Однако если предметная область вашей базы данных требует соблюдения одинаковой структуры документов в коллекции, это можно обеспечить с помощью создания схемы данных для проверки структурной целостности (schema validation). Это гарантирует, что все документы в коллекции соответствуют определенному набору правил, что ведёт к уменьшению ошибок ввода данных и обеспечивает согласованность данных.

Добавим валидатор к коллекции *books*. Этот валидатор потребует, чтобы каждый документ имел поля *title, author и year,* а также будет обеспечивать соблюдение определенных типов данных и диапазонов для этих полей. Используйте команду **runCommand**, которая запустит изменение коллекции (**collMod**) для добавления валидатора (**validator**) в коллекцию. Текст команды представлен на скриншоте на следующей странице.

Протестируем валидатор, попытавшись вставить документ, нарушающий правила, например, такой:

```
db.books.insertOne({
  title: "The Hobbit",
  author: "J.R.R. Tolkien",
  year: 7260
});
```

Вставить это документ в коллекцию не удастся, СУБД выдаст ошибку: MongoServerError: Document failed validation. Это подтверждает, что наше правило проверки схемы активно и эффективно защищает целостность данных.

```
std-01> db.runCommand({
          collMod: "books",
          validator: {
            $jsonSchema: {
              bsonType: "object",
              required: ["title", "author", "year"],
              properties: {
                title: {
                  bsonType: "string",
                  description: "must be a string and is required"
                },
                author: {
                  bsonType: "string",
                  description: "must be a string and is required"
                },
                year: {
                  bsonType: "int",
                  minimum: 1000,
                  maximum: 2024,
                  description: "must be an integer between 1000 and 2024"
                }
              }
            }
          }
        });
```

! Попробуйте ввести ещё несколько документов с заведомыми нарушениями структуры либо диапазона значений полей. Проверьте, что валидатор работает и в этих случаях.

7. Контрольное задание.

Создайте в вашей базе данных новую коллекцию с именем *employees*, содержащую следующий набор документов:

```
[
  { name: "John Doe", age: 25, department: "Sales", salary: 50000 },
  { name: "Jane Smith", age: 35, department: "Marketing", salary: 60000 },
  { name: "Bob Johnson", age: 45, department: "Sales", salary: 55000 },
  { name: "Alice Brown", age: 30, department: "HR", salary: 52000 }
]
```

Напишите команду, которая удалит из этой коллекции сотрудников отдела "Sales", чья зарплата меньше 55 000. Приведите результаты работы этой команды.