Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

«Реализация БД с использованием СУБД MongoDB. Запросы к базе данных»

по дисциплине «Проектирование и реализация баз данных»

Обучающийся *Коваленко Евгений Юрьевич*
Факультет прикладной информатики
Группа K*3241*
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

6.1. ВВЕДЕНИЕ В СУБД MONGODB. УСТАНОВКА MONGODB. НАЧАЛО РАБОТЫ С БД

Цель: овладеть практическими навыками установки СУБД MongoDB.

Введение

Вопросы для самоконтроля

1. *Какую модель данных поддерживает СУБД MongoDB?*

Документо-ориентированную (само это слово мне ни о чем не говорит, зато по методичке!)

2. *Какой формат данных поддерживается в MongoDB?*

BSON (binary JSON - как обычный JSON, только занимает больше места, но зато быстрее).

3. *Из чего состоит база данных в MongoDB?*

Коллекции - аналогия для таблиц, документы - аналогия для строк.

4. *Имеет ли БД в MongoDB жесткую структуру?*

Нет.

Работа с БД

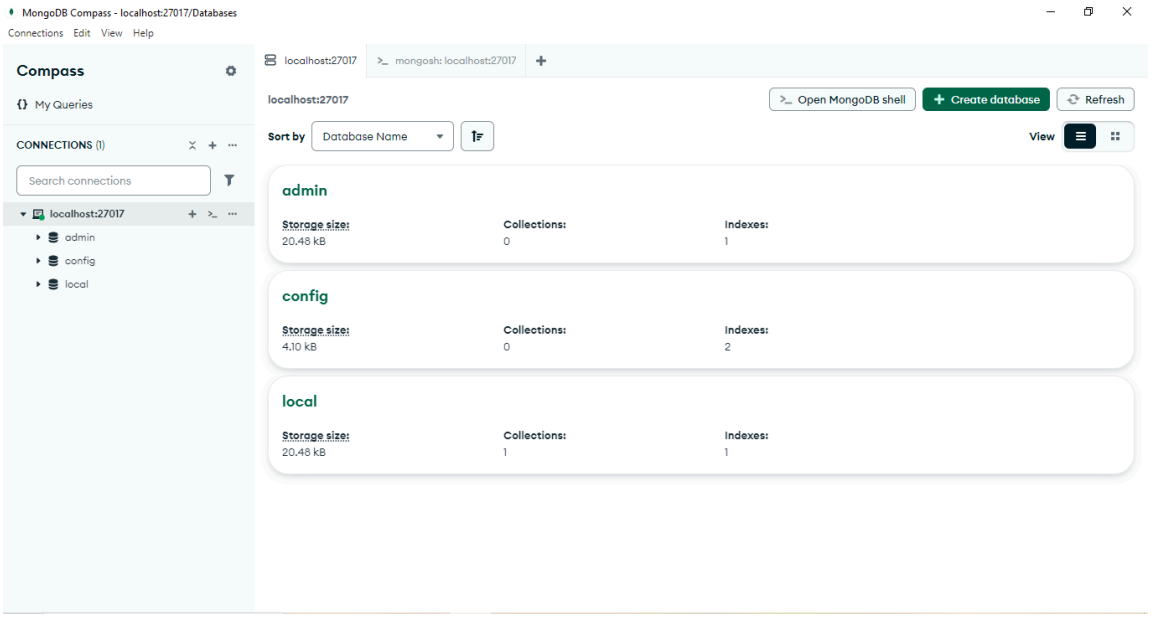
Практическое задание

1. *Установите MongoDB для обеих типов систем (32/64 бита).*

С вашего позволения этот шаг я не буду показывать… Все сделано в соответствии с методичкой - скачан файл `.msi` и распакован в папку C:/mongodb.

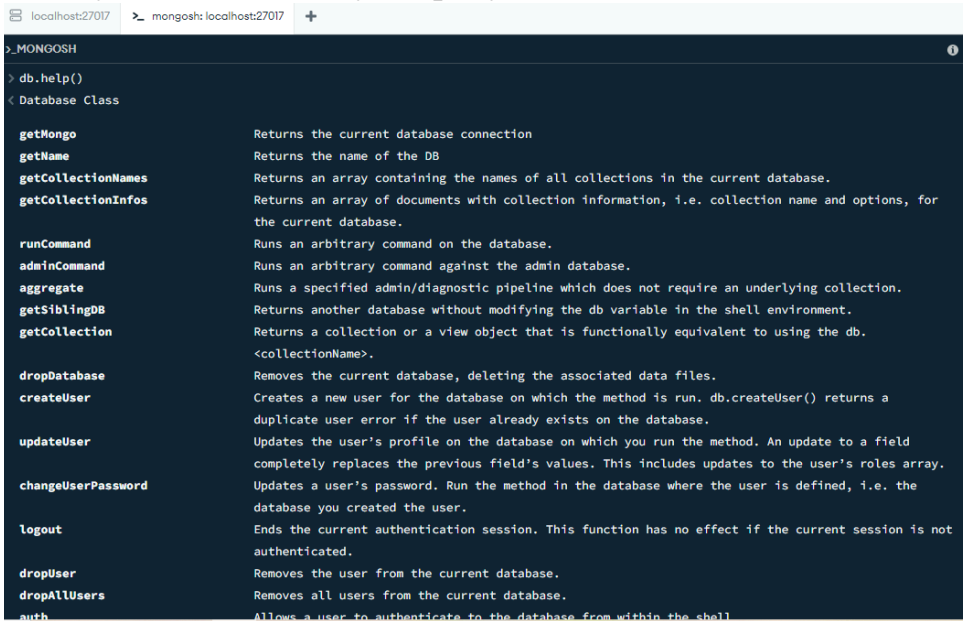
2. *Проверьте работоспособность системы запуском клиента mongo.*

Не получилось. Тут мне после всяких тыков по различным элементам папки помогло лишь приложение “MongoDBCompass”, автоматически установившееся на рабочий стол. Для подключения к консоли нужно нажать среднюю кнопку рядом со строкой “localhost” в левой панели:



3. *Выполните методы:*
    a. `db.help()`

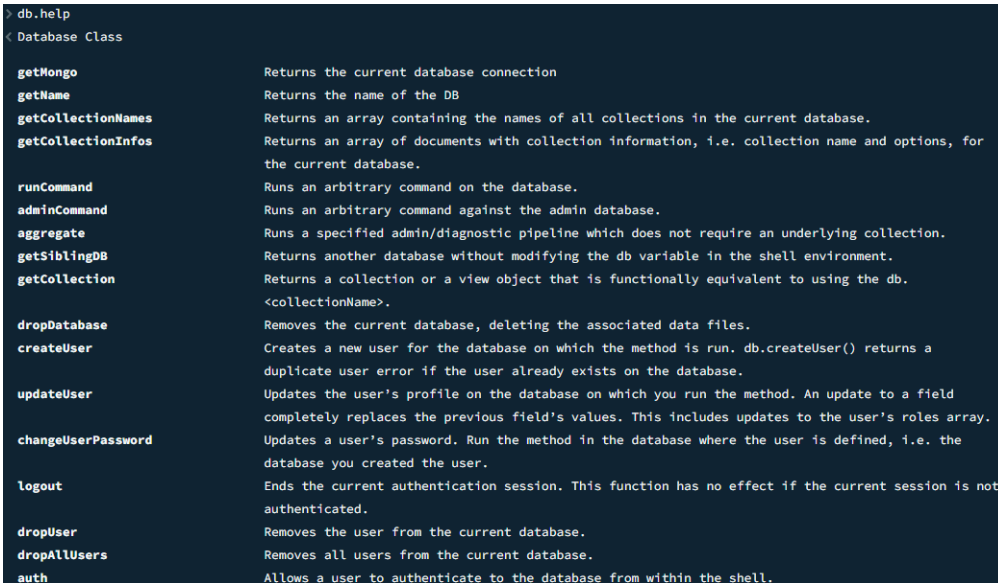
Вот что получилось по этому запросу:



То есть этот метод выводит список доступных методов с описанием того, что каждый из методов делает.

b. `db.help`

Аналогично:



Методичка наврала, выводится не внутреннее представление этого метода.

c. `db.stats()`

Выводится общая статистика по базе данных:

```
> db.stats()
< {
    db: 'test',
    collections: Long('0'),
    views: Long('0'),
    objects: Long('0'),
    avgObjSize: 0,
    dataSize: 0,
    storageSize: 0,
    indexes: Long('0'),
    indexSize: 0,
    totalSize: 0,
    scaleFactor: Long('1'),
    fsUsedSize: 0,
    fsTotalSize: 0,
    ok: 1
  }
```

4. *Создайте БД learn.*
   Для созданий БД мы используем команду `use`.

```
> use learn
< switched to db learn
```

5. *Получите список доступных БД.*
   Для этого используется команда `show dbs`:

```
> show dbs
< admin   40.00 KiB
  config  92.00 KiB
  local   40.00 KiB
```

Тут мы видим, что БД learn нет. Это все потому, что БД должна создаться автоматически при добавлении в одну из ее коллекций нового документа.

6. *Создайте коллекцию unicorns, вставив в нее документ* `{name: 'Aurora', gender: 'f', weight: 450}`.

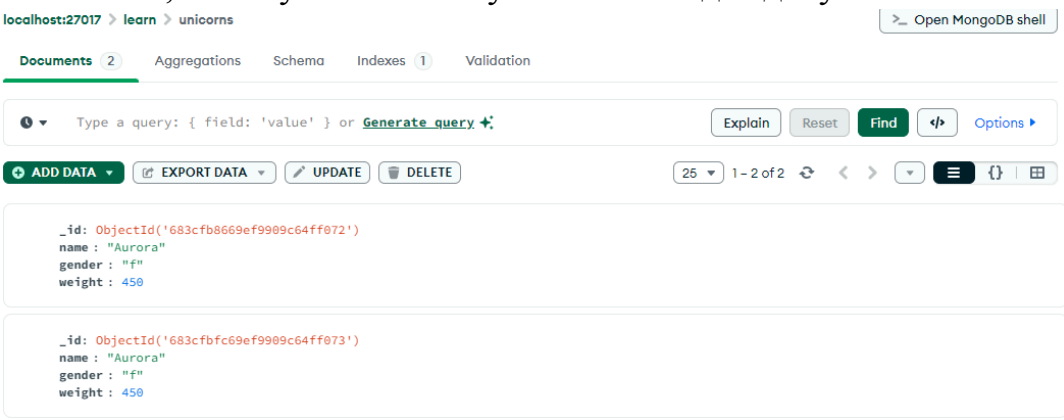
Для этого, судя по методичке, используется команда `db.<collection>.insert()`:

```
> db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})
< DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683cfb8669ef9909c64ff072')
    }
  }
```

И здесь обманули… Тогда воспользуемся методом `insertOne()`:

```
> db.unicorns.insertOne({name: 'Aurora', gender: 'f', weight: 450})
< {
    acknowledged: true,
    insertedId: ObjectId('683cfbfc69ef9909c64ff073')
  }
```

Хотя ладно, видимо, метод `insert()` просто какой-то старый и не предпочтительный, потому что по итогу вставилось два документа:



7. *Просмотрите список текущих коллекций.*

Для просмотра списка текущих коллекций можно воспользоваться одним из двух методов: `db.getCollectionNames()` или `show collections`. Мы воспользуемся вторым, в нем как-то поменьше писать:

```
> show collections
< unicorns
```

8. *Переименуйте коллекцию unicorns.*

Для переименования коллекции предназначен метод `renameCollection()`:

```
> db.unicorns.renameCollection("aboba")
< { ok: 1 }
```

9. *Просмотрите статистику коллекции.*

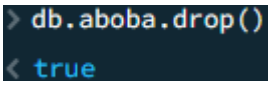
Метод очевиден:

```
> db.aboba.stats()
< {
    ok: 1,
    capped: false,
    wiredTiger: {
      metadata: { formatVersion: 1 },
      creationString: 'access_pattern_hint=none,allocation_size=4KB,app_metadata=(formatVersion=1),assert=(commit_timestamp=none,dural
      type: 'file',
      uri: 'statistics:table:collection-7-16078707285402663180',
      LSM: {
        'bloom filter false positives': 0,
        'bloom filter hits': 0,
        'bloom filter misses': 0,
        'bloom filter pages evicted from cache': 0,
        'bloom filter pages read into cache': 0,
        'bloom filters in the LSM tree': 0,
        'chunks in the LSM tree': 0,
        'highest merge generation in the LSM tree': 0,
        'queries that could have benefited from a Bloom filter that did not exist': 0,
        'sleep for LSM checkpoint throttle': 0,
        'sleep for LSM merge throttle': 0,
        'total size of bloom filters': 0
      },
      autocommit: {
        'retries for readonly operations': 0,
```

Там много вывелось, все вставлять не буду.

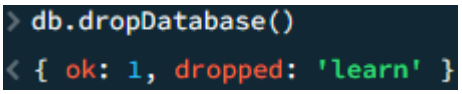
10. *Удалите коллекцию.*

Для удаления коллекции нужно применить к ней метод `drop()`:



11. *Удалите БД learn.*

А вот для удаления всей БД уже нужен метод `dropDatabase()`:



6.2. Работа с БД в СУБД MongoDB

Цель: овладеть практическими навыками работы с CRUD-операциями, с вложенными объектами в коллекции базы данных MongoDB, агрегации и изменения данных, со ссылками и индексами в базе данных MongoDB.

ПРАКТИЧЕСКАЯ ЧАСТЬ
2 CRUD-ОПЕРАЦИИ В СУБД MONGODB. ВСТАВКА ДАННЫХ. ВЫБОРКА ДАННЫХ
2.1 ВСТАВКА ДОКУМЕНТОВ В КОЛЛЕКЦИЮ

Практическое задание 2.1.1

1. *Создайте базу данных learn.*

```
> use learn
< switched to db learn
```

2. *Заполните коллекцию единорогов unicorns:*

```
db.unicorns.insert({name: 'Horny', loves: ['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});
db.unicorns.insert({name: 'Aurora', loves: ['carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});
db.unicorns.insert({name: 'Unicrom', loves: ['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});
db.unicorns.insert({name: 'Roooooodles', loves: ['apple'], weight: 575, gender: 'm', vampires: 99});
db.unicorns.insert({name: 'Solnara', loves:['apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});
db.unicorns.insert({name:'Ayna', loves: ['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});
db.unicorns.insert({name:'Kenny', loves: ['grape', 'lemon'], weight: 690, gender: 'm', vampires: 39});
db.unicorns.insert({name: 'Raleigh', loves: ['apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});
db.unicorns.insert({name: 'Leia', loves: ['apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});
db.unicorns.insert({name: 'Pilot', loves: ['apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});
db.unicorns.insert({name: 'Nimue', loves: ['grape', 'carrot'], weight: 540, gender: 'f'});
```

```
> db.unicorns.insert({name: 'Horny', loves: ['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});
  db.unicorns.insert({name: 'Aurora', loves: ['carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});
  db.unicorns.insert({name: 'Unicrom', loves: ['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});
  db.unicorns.insert({name: 'Roooooodles', loves: ['apple'], weight: 575, gender: 'm', vampires: 99});
  db.unicorns.insert({name: 'Solnara', loves:['apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});
  db.unicorns.insert({name:'Ayna', loves: ['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});
  db.unicorns.insert({name:'Kenny', loves: ['grape', 'lemon'], weight: 690,  gender: 'm', vampires: 39});
  db.unicorns.insert({name: 'Raleigh', loves: ['apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});
  db.unicorns.insert({name: 'Leia', loves: ['apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});
  db.unicorns.insert({name: 'Pilot', loves: ['apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});
  db.unicorns.insert({name: 'Nimue', loves: ['grape', 'carrot'], weight: 540, gender: 'f'});
< DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683d00fa35ad23deff149f50')
    }
  }
```

3. *Используя второй способ, вставьте в коллекцию единорогов документ:* `{name: 'Dunx', loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165}`

Во втором способе нам сначала надо определить документ, а потом вставить его тем же методом:

```
document=(...)
db.<collection>.insert(document)
```

```
> document=({name: 'Dunx', loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165})
< {
    name: 'Dunx',
    loves: [ 'grape', 'watermelon' ],
    weight: 704,
    gender: 'm',
    vampires: 165
  }
> db.unicorns.insert(document)
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683d02a135ad23deff149f51')
    }
  }
```

4. *Проверьте содержимое коллекции с помощью метода* `find`.

Тут просто вывелись все документы из коллекции:

2.2 ВЫБОРКА ДАННЫХ ИЗ БД

Практическое задание 2.2.1

1. *Сформируйте запросы для вывода списков самцов и самок единорогов. Ограничьте список самок первыми тремя особями. Отсортируйте списки по имени.*

Первый запрос - для самцов с сортировкой по имени, второй запрос - для самок с сортировкой по имени и выводом первых трех особей.

`db.unicorns.find({gender: 'm'}).sort({name: 1})` (там действительно имена отсортированы в алфавитном порядке):

`db.unicorns.find({gender: 'f'}).sort({name: 1}).limit(3)` (здесь действительно 3 особи в алфавитном порядке по именам):

```
> db.unicorns.find({gender: 'f'}).sort({name: 1}).limit(3)
< {
    _id: ObjectId('683d00fa35ad23deff149f47'),
    name: 'Aurora',
    loves: [
      'carrot',
      'grape'
    ],
    weight: 450,
    gender: 'f',
    vampires: 43
  }
  {
    _id: ObjectId('683d00fa35ad23deff149f4b'),
    name: 'Ayna',
    loves: [
      'strawberry',
      'lemon'
    ],
    weight: 733,
    gender: 'f',
    vampires: 40
  }
  {
    _id: ObjectId('683d00fa35ad23deff149f4e'),
```

2. *Найдите всех самок, которые любят carrot. Ограничьте этот список первой особью с помощью функций* `findOne` *и* `limit`.

   В целом запросы очевидны:

```
> db.unicorns.findOne({gender: 'f', loves: 'carrot'})
< {
    _id: ObjectId('683d00fa35ad23deff149f47'),
    name: 'Aurora',
    loves: [
      'carrot',
      'grape'
    ],
    weight: 450,
    gender: 'f',
    vampires: 43
  }
```

```
> db.unicorns.find({gender: 'f', loves: 'carrot'}).limit(1)
< {
    _id: ObjectId('683d00fa35ad23deff149f47'),
    name: 'Aurora',
    loves: [
      'carrot',
      'grape'
    ],
    weight: 450,
    gender: 'f',
    vampires: 43
  }
```

Практическое задание 2.2.2

Модифицируйте запрос для вывода списков самцов единорогов, исключив из результата информацию о предпотениях и поле.

Тут нам нужно использовать метод `find()`, в качестве второго аргумента которого в виде JSON-структуры передать 0 тем свойствам документов, которые не нужно отображать:

```
> db.unicorns.find({gender: 'm'}, {loves: 0, _id: 0})
< {
    name: 'Horny',
    weight: 600,
    gender: 'm',
    vampires: 63
  }
  {
    name: 'Unicrom',
    weight: 984,
    gender: 'm',
    vampires: 182
  }
  {
    name: 'Roooooodles',
    weight: 575,
    gender: 'm',
    vampires: 99
  }
  {
    name: 'Kenny',
    weight: 690,
    gender: 'm',
    vampires: 39
  }
```

Практическое задание 2.2.3

Вывести список единорогов в обратном порядке добавления.

Здесь можно использовать параметр `$natural`, который позволяет задать сортировку: документы передаются в том порядке, в каком они были добавлены в коллекцию (тогда ему нужно задать значение 1), либо в обратном порядке (значение -1).

Действительно, единорога Dunx я добавлял вторым способом, т.е. после добавления остальных, значит, сортировка действительно произведена в обратном порядке:

Практическое задание 2.2.4

Вывести список единорогов с названием первого любимого предпочтения, исключив идентификатор.

Используем вот такую команду: `db.unicorns.find({}, {_id: 0, loves: {$slice: 1}})`

Здесь мы помним, что первый аргумент метода `find()` - это то, что мы ищем (здесь нас в принципе интересует вся коллекция), второй аргумент -

то, что нужно или не нужно отображать (убираем id и с помощью параметра `$slice` указываем, что нам нужен только первый элемент из предпочтений).

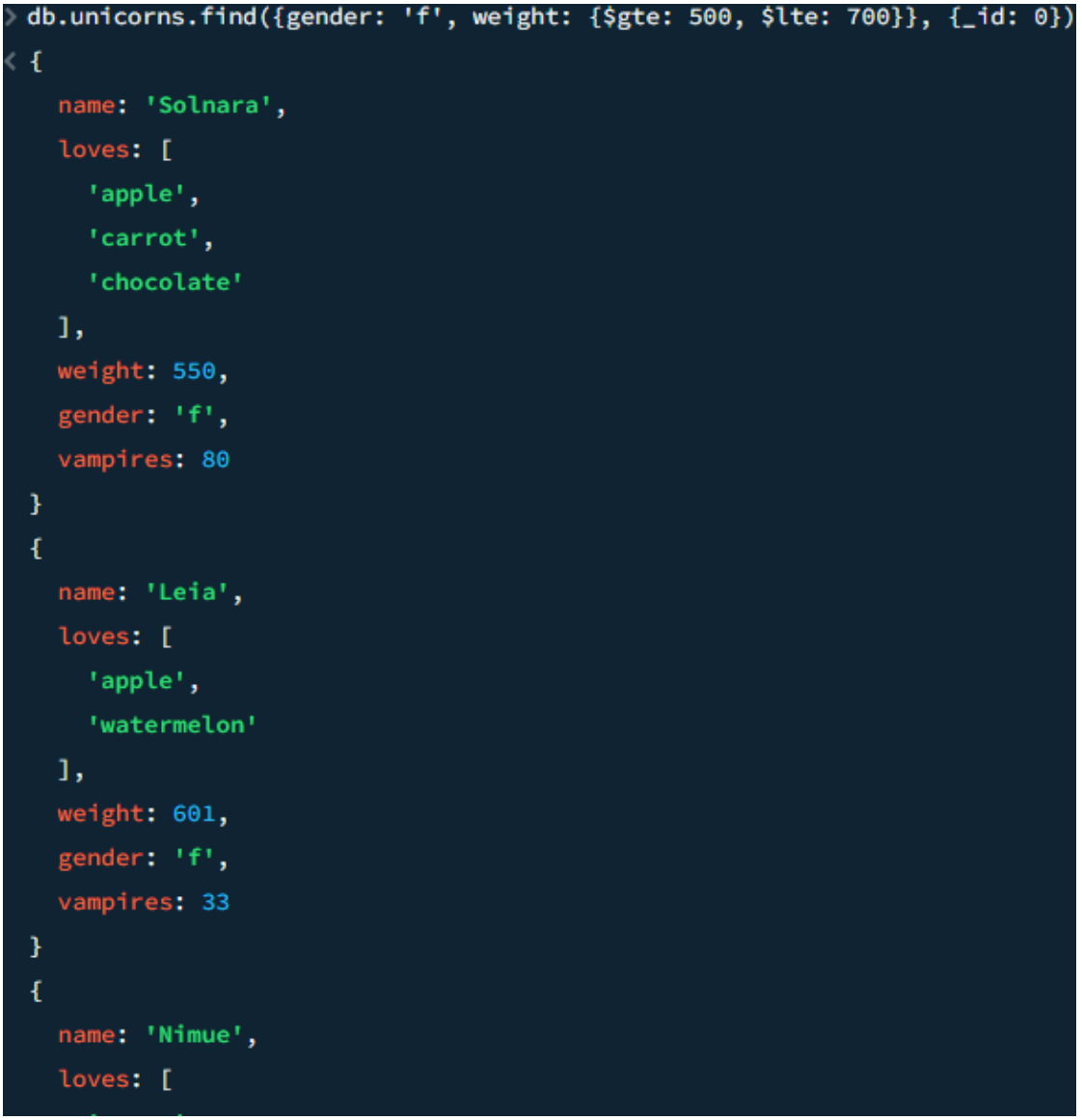
2.3 ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Практическое задание 2.3.1

Вывести список самок единорогов весом от полутонны до 700 кг, исключив вывод идентификатора.

Команда такая: `db.unicorns.find({gender: 'f', weight: {$gte: 500, $lte: 700}}, {_id: 0})`

Здесь мы выбираем нужный пол единорога, ищем вес с помощью параметров `$gte` (эквивалент “>=”) и `$lte` (эквивалент “<=”), а потом убираем id с помощью второго аргумента метода `find()`.



Практическое задание 2.3.2

Вывести список самцов единорогов весом от полутонны и предпочитающих grape и lemon, исключив вывод идентификатора.

Команда: `db.unicorns.find({gender: 'm', weight: {$gte: 500}, loves: {$in: ['grape', 'lemon']}}, {_id: 0})`

Здесь у нас появляется новый параметр - `$in` - с его помощью можно передать список того, что должно быть в атрибуте `loves`.

```
> db.unicorns.find({gender: 'm', weight: {$gte: 500}, loves: {$in: ['grape', 'lemon']}}, {_id: 0})
< {
    name: 'Kenny',
    loves: [
      'grape',
      'lemon'
    ],
    weight: 690,
    gender: 'm',
    vampires: 39
  }
  {
    name: 'Dunx',
    loves: [
      'grape',
      'watermelon'
    ],
    weight: 704,
    gender: 'm',
    vampires: 165
  }
```

Практическое задание 2.3.3

Найти всех единорогов, не имеющих ключ `vampires`.

Команда: `db.unicorns.find({vampires: {$exists: false}})`

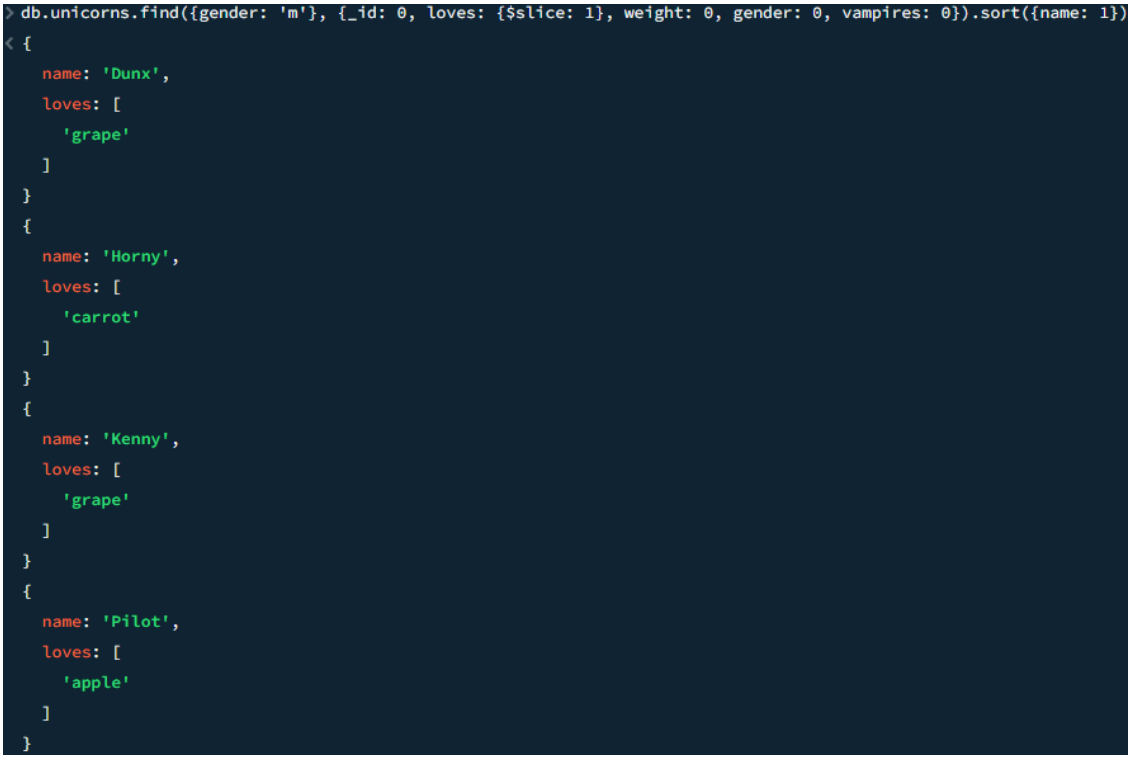
Здесь мы используем параметр `$exists`, ему устанавливаем значение `false`.

```
> db.unicorns.find({vampires: {$exists: false}})
< {
    _id: ObjectId('683d00fa35ad23deff149f50'),
    name: 'Nimue',
    loves: [
      'grape',
      'carrot'
    ],
    weight: 540,
    gender: 'f'
  }
```

Практическое задание 2.3.4

Вывести список упорядоченный список имен самцов единорогов с информацией об их первом предпочтении.

Используем команду из задания 2.2.4, только добавим сортировку и выбор пола и сделаем так, чтобы отображались только имена и первые предпочтения: `db.unicorns.find({gender: 'm'}, {_id: 0, loves: {$slice: 1}, weight: 0, gender: 0, vampires: 0}).sort({name: 1})`



3 ЗАПРОСЫ К БАЗЕ ДАННЫХ MONGODB. ВЫБОРКА ДАННЫХ. ВЛОЖЕННЫЕ ОБЪЕКТЫ. ИСПОЛЬЗОВАНИЕ КУРСОРОВ. АГРЕГИРОВАННЫЕ ЗАПРОСЫ. ИЗМЕНЕНИЕ ДАННЫХ

3.1 ЗАПРОС К ВЛОЖЕННЫМ ОБЪЕКТАМ

Практическое задание 3.1.1

1. *Создайте коллекцию towns, включающую следующие документы:*

```
{name: "Punxsutawney ",
populatiuon: 6200,
last_sensus: ISODate("2008-01-31"),
famous_for: [""],
mayor: {
   name: "Jim Wehrle"
}}
{name: "New York",
populatiuon: 22200000,
last_sensus: ISODate("2009-07-31"),
famous_for: ["status of liberty", "food"],
mayor: {
   name: "Michael Bloomberg",
party: "I"}}
{name: "Portland",
populatiuon: 528000,
last_sensus: ISODate("2009-07-20"),
famous_for: ["beer", "food"],
mayor: {
   name: "Sam Adams",
party: "D"}}
```

Используем метод `insert()`:

```
> db.towns.insert({name: "Punxsutawney ",
  populatiuon: 6200,
  last_sensus: ISODate("2008-01-31"),
  famous_for: [""],
  mayor: {
     name: "Jim Wehrle"
  }});
  db.towns.insert({name: "New York",
  populatiuon: 22200000,
  last_sensus: ISODate("2009-07-31"),
  famous_for: ["status of liberty", "food"],
  mayor: {
     name: "Michael Bloomberg",
  party: "I"}});
  db.towns.insert({name: "Portland",
  populatiuon: 528000,
  last_sensus: ISODate("2009-07-20"),
  famous_for: ["beer", "food"],
  mayor: {
     name: "Sam Adams",
  party: "D"}});
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683d1a6535ad23deff149f54')
```

2. *Сформировать запрос, который возвращает список городов с независимыми мэрами (`party="I"`). Вывести только название города и информацию о мэре.*

Используем такую команду: `db.towns.find({"mayor.party": "I"}, {_id: 0, name: 1, mayor: 1})` (я попробовал не выставлять `_id: 0`, но он, видимо, всегда по умолчанию стоит; еще при работе с вложенными атрибутами нужно заключать их в кавычки).

```
> db.towns.find({"mayor.party": "I"}, {_id: 0, name: 1, mayor: 1})
< {
    name: 'New York',
    mayor: {
      name: 'Michael Bloomberg',
      party: 'I'
    }
  }
```

3. *Сформировать запрос, который возвращает список беспартийных мэров (`party` отсутствует). Вывести только название города и информацию о мэре.*

Используем похожую команду, но с уже известными нам параметром `$exists`: `db.towns.find({"mayor.party": {$exists: false}}, {_id: 0, name: 1, mayor: 1})`

```
> db.towns.find({"mayor.party": {$exists: false}}, {_id: 0, name: 1, mayor: 1})
< {
    name: 'Punxsutawney ',
    mayor: {
      name: 'Jim Wehrle'
    }
  }
```

КУРСОРЫ

Практическое задание 3.1.2

1. *Сформировать функцию для вывода списка самцов единорогов.*

Вот функция: `function getMaleUnicorns() { return db.unicorns.find({gender: 'm'}).sort({name: 1}); }`

```
> function getMaleUnicorns() { return db.unicorns.find({gender: 'm'}).sort({name: 1}); }
< [Function: getMaleUnicorns]
```

2. *Создать курсор для этого списка из первых двух особей с сортировкой в лексикографическом порядке.*

`var maleCursor = getMaleUnicorns().limit(2);`

При его вводе в консоль ничего не произошло.

3. *Вывести результат, используя forEach.*

Раз уж мы работаем с первыми предпочтениями, то так и запишем вывод: `maleCursor.forEach(function(unicorn) { printjson({name: unicorn.name, loves: unicorn.loves[0], weight: unicorn.weight}); });`

```
> maleCursor.forEach(function(unicorn) { printjson({name: unicorn.name, loves: unicorn.loves[0], weight: unicorn.weight}); });
< { name: 'Dunx', loves: 'grape', weight: 704 }
< { name: 'Horny', loves: 'carrot', weight: 600 }
```

3.2 АГРЕГИРОВАННЫЕ ЗАПРОСЫ

Практическое задание 3.2.1

Вывести количество самок единорогов весом от полутонны до 600 кг.

Используем метод `count()`: `db.unicorns.count({gender: 'f', weight: {$gte: 500, $lte: 600}})`

```
> db.unicorns.count({gender: 'f', weight: {$gte: 500, $lte: 600}})
< DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
< 2
```

Но в будущем, конечно, придется использовать метод `countDocument()`, чтобы консоль не ругалась!

Практическое задание 3.2.2

Вывести список предпочтений.

Используем метод `distinct()`: `db.unicorns.distinct("loves")`

```
> db.unicorns.distinct("loves")
< [
    'apple',      'carrot',
    'chocolate',  'energon',
    'grape',      'lemon',
    'papaya',     'redbull',
    'strawberry', 'sugar',
    'watermelon'
  ]
```

Практическое задание 3.2.3

Посчитать количество особей единорогов обоих полов.

Команда: `db.unicorns.aggregate([{$group: {_id: "$gender", count: {$sum: 1}}}])`

Здесь мы используем метод `aggregate()`, который имеет три параметра (конечно же, скопировал из методички): `$group` - агрегатор, который вернет новый документ, `_id` - указывает на ключ, по которому надо проводить группировку (`$<field_name>`), `$sum` - оператор для вычисления.

```
> db.unicorns.aggregate([{$group: {_id: "$gender", count: {$sum: 1}}}])
< {
    _id: 'f',
    count: 5
  }
  {
    _id: 'm',
    count: 7
  }
```

3.3 РЕДАКТИРОВАНИЕ ДАННЫХ

Практическое задание 3.3.1

1. *Выполнить команду:*

`db.unicorns.save({name: 'Barny', loves: ['grape'], weight: 340, gender: 'm'})`

Меня обманула методичка:

```
> db.unicorns.save({name: 'Barny', loves: ['grape'], weight: 340, gender: 'm'})
✖ ▸ TypeError: db.unicorns.save is not a function
```

Давайте тогда через `insert()` вставим (`id` не указан, так что выполнится вставка):

```
> db.unicorns.insert({name: 'Barny', loves: ['grape'], weight: 340, gender: 'm'})
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683d3c0135ad23deff149f55')
    }
  }
```

2. *Проверить содержимое коллекции* `unicorns`.

Проверим с помощью команды `db.unicorns.find()`. Единорог Barny вставился самым последним:

```
>_MONGOSH
    ],
    weight: 540,
    gender: 'f'
  }
  {
    _id: ObjectId('683d02a135ad23deff149f51'),
    name: 'Dunx',
    loves: [
      'grape',
      'watermelon'
    ],
    weight: 704,
    gender: 'm',
    vampires: 165
  }
  {
    _id: ObjectId('683d3c0135ad23deff149f55'),
    name: 'Barny',
    loves: [
      'grape'
    ],
    weight: 340,
    gender: 'm'
  }
```

Практическое задание 3.3.2

1. *Для самки единорога Аупа внести изменения в БД: теперь ее вес 800, она убила 51 вапмира.*

Используем метод `update()`, в котором первый аргумент - информация для поиска нужного документа, а второй аргумент - ключи, значения которых нужно изменить. Есть также опциональный аргумент - `upsert` (если он `true`, то создастся новый документ, если ничего не

найдется по запросу из первого аргумента, а если `false`, то не создастся). Кстати, это не работает:

```
> db.unicorns.update({name: 'Ayna'}, {weight: 800, vampires: 51})
< DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
✖ ▸ MongoInvalidArgumentError: Update document requires atomic operators
> db.unicorns.update({name: 'Ayna'}, {weight: 800, vampires: 51}, {upsert: true})
✖ ▸ MongoInvalidArgumentError: Update document requires atomic operators
```

Так что забежим вперед и используем оператор `$set`. Вот так будет выглядеть команда: `db.unicorns.update({name: 'Ayna'}, {$set: {weight: 800, vampires: 51}}, {upsert: true})`

```
> db.unicorns.update({name: 'Ayna'}, {$set: {weight: 800, vampires: 51}}, {upsert: true})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Но, возможно, дело было не в `upsert`.

2. *Проверить содержимое коллекции `unicorns`.*

Обновилось:

```
{
  _id: ObjectId('683d00fa35ad23deff149f4b'),
  name: 'Ayna',
  loves: [
    'strawberry',
    'lemon'
  ],
  weight: 800,
  gender: 'f',
  vampires: 51
}
```

Практическое задание 3.3.3

1. *Для самца единорога Raleigh внести изменения в БД: теперь он любит рэдбул.*

Используем известный нам из предыдущего задания синтаксис: `db.unicorns.update({name: 'Raleigh'}, {$set: {loves: ['redbull']}})`

```
> db.unicorns.update({name: 'Raleigh'}, {$set: {loves: ['redbull']}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

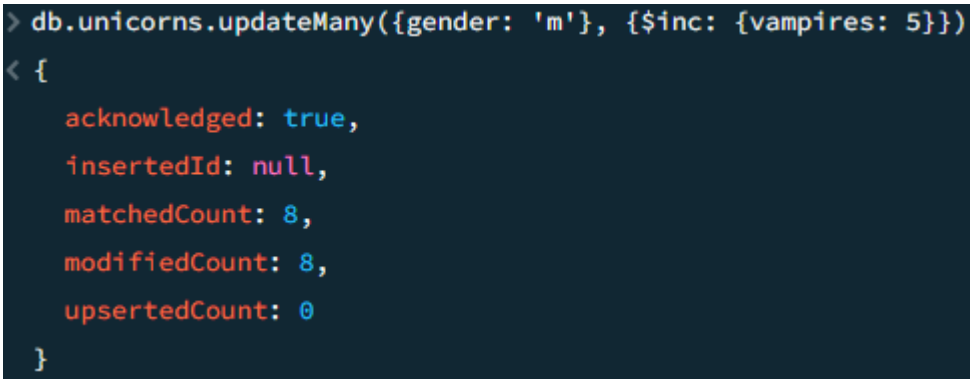
2. *Проверить содержимое коллекции* `unicorns`.

Данные об этом единороге обновились:

Практическое задание 3.3.4

1. *Всем самцам единорогов увеличить количество убитых вампиров на 5.*

Для увеличения числового значения на определенное число можно использовать параметр `$inc`. Так как нужно обновить значения всех самцов (т.е. нескольких документов), то с использованием метода `updateMany()` получится вот такая команда: `db.unicorns.updateMany({gender: 'm'}, {$inc: {vampires: 5}})`



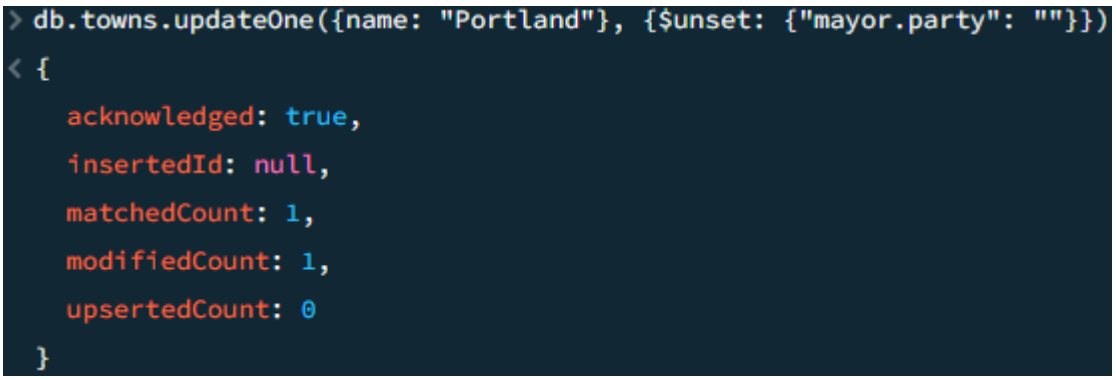
2. *Проверить содержимое коллекции* `unicorns`.

У тех самцов, у которых не было ключа `vampires`, появилось его значение `5`, у остальных увеличилось на 5:

Практическое задание 3.3.5

1. *Изменить информацию о городе Портланд: мэр этого города теперь беспартийный.*

Тут нам нужно удалить партию. Для этого можно использовать оператор `$unset`. Получится следующая команда: `db.towns.updateOne({name: "Portland"}, {$unset: {"mayor.party": ""}})`



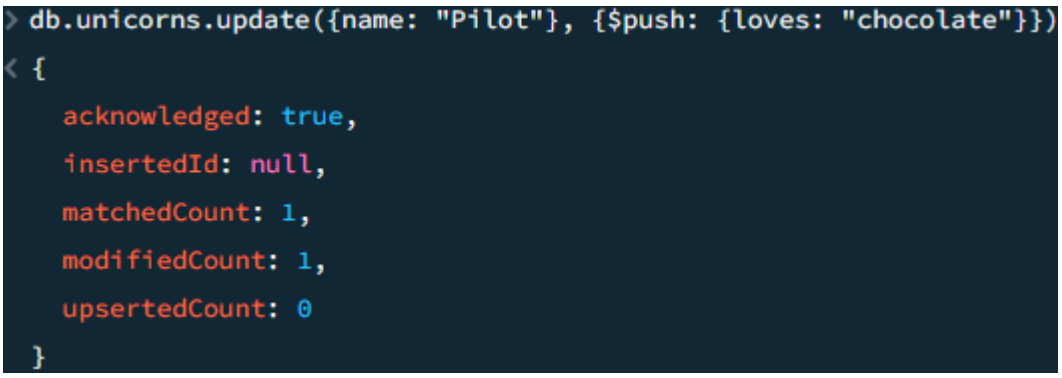
2. *Проверить содержимое коллекции `towns`.*

Партия успешно удалилась:

Практическое задание 3.3.6

1. *Изменить информацию о самце единорога Pilot: теперь он любит и шоколад.*

Используем оператор `$push`, который позволяет добавить элемент в значение-список: `db.unicorns.update({name: "Pilot"}, {$push: {loves: "chocolate"}})`



2. *Проверить содержимое коллекции* `unicorns`.

Обновилось:

Практическое задание 3.3.7

1. *Изменить информацию о самке единорога Aurora: теперь она любит еще и сахар, и лимоны.*

Здесь нам нужно использовать оператор `$each`, который позволяет добавить несколько элементов в список, но к нему до кучи идет еще и оператор `$addToSet` (зачем он именно тут, я не знаю, но вообще он создаст список, если изначально введенному ключу никакого значения не соответствовало): `db.unicorns.update({name: "Aurora"}, {$addToSet: {loves: {$each: ["sugar", "lemon"]}}})`

```
> db.unicorns.update({name : "Aurora"}, {$addToSet: {loves: {$each: ["sugar", "lemon"]}}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

2. *Проверить содержимое коллекции* `unicorns`.

Обновили Аврору:

```
{
  _id: ObjectId('683d00fa35ad23deff149f47'),
  name: 'Aurora',
  loves: [
    'carrot',
    'grape',
    'sugar',
    'lemon'
  ],
  weight: 450,
  gender: 'f',
  vampires: 43
}
```

3.4 УДАЛЕНИЕ ДАННЫХ ИЗ КОЛЛЕКЦИИ

Практическое задание 3.4.1

1. *Создайте коллекцию* `towns`*, включающую следующие документы:*

```
{name: "Punxsutawney ",
popujatiuon: 6200,
last_sensus: ISODate("2008-01-31"),
famous_for: ["phil the groundhog"],
mayor: {
   name: "Jim Wehrle"
}}
{name: "New York",
popujatiuon: 22200000,
last_sensus: ISODate("2009-07-31"),
```

```
famous_for: ["status of liberty", "food"],
mayor: {
   name: "Michael Bloomberg",
party: "I"}}
{name: "Portland",
popujatiuon: 528000,
last_sensus: ISODate("2009-07-20"),
famous_for: ["beer", "food"],
mayor: {
   name: "Sam Adams",
party: "D"}}
```

Для начала удалим предыдущую коллекцию с таким же названием с помощью команды `db.towns.remove({})`:

```
> db.towns.remove({})
< DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
< {
    acknowledged: true,
    deletedCount: 6
  }
```

(Да, их тут 6, потому что я сначала вставил, а потом понял, что надо создать новую коллекцию.)

Теперь создадим:

```
last_sensus: ISODate("2008-01-31"),
famous_for: ["phil the groundhog"],
mayor: {
   name: "Jim Wehrle"
}});
db.towns.insert({name: "New York",
popujatiuon: 22200000,
last_sensus: ISODate("2009-07-31"),
famous_for: ["status of liberty", "food"],
mayor: {
   name: "Michael Bloomberg",
party: "I"}});
db.towns.insert({name: "Portland",
popujatiuon: 528000,
last_sensus: ISODate("2009-07-20"),
famous_for: ["beer", "food"],
mayor: {
   name: "Sam Adams",
party: "D"}});
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('683d4ad035ad23deff149f5b')
  }
}
```

2. *Удалите документы с беспартийными мэрами.*
   Для удаления используем метод `remove()`: `db.towns.remove({"mayor.party": {$exists: false}})`

```
> db.towns.remove({"mayor.party": {$exists: false}})
< {
    acknowledged: true,
    deletedCount: 1
  }
```

3. *Проверьте содержание коллекции.*
   Беспартийный мэр удален (он был первым):

```
> db.towns.find()
< {
    _id: ObjectId('683d4be935ad23deff149f60'),
    name: 'New York',
    popujatiuon: 22200000,
    last_sensus: 2009-07-31T00:00:00.000Z,
    famous_for: [
      'status of liberty',
      'food'
    ],
    mayor: {
      name: 'Michael Bloomberg',
      party: 'I'
    }
  }
  {
    _id: ObjectId('683d4be935ad23deff149f61'),
    name: 'Portland',
    popujatiuon: 528000,
    last_sensus: 2009-07-20T00:00:00.000Z,
    famous_for: [
      'beer',
      'food'
    ],
    mayor: {
```

4. *Очистите коллекцию.*
   Уже знакомая команда:

```
> db.towns.remove({})
< {
    acknowledged: true,
    deletedCount: 2
  }
```

5. *Просмотрите список доступных коллекций.*

Коллекция `towns` не удалилась, она просто пустая, так что она тоже теоретически доступна:

```
> show collections
towns
unicorns
```

Контрольные вопросы

1. Как используется оператор точка?

Он применяется для вложенных документов, причем если мы ее используем, то этот вложенный документ мы должны заключить в кавычки.

2. Как можно использовать курсор?

Курсор - это указатель на результат запроса, который получился с помощью метода `find()`. По нему можно пробегаться с помощью циклов `while` и `forEach`, по нему можно сортировать с помощью `sort()`, можно пропустить какие-то значения с помощью `skip()`.

3. Какие возможности агрегирования данных существуют в MongoDB?

Агрегация происходит с помощью метода `aggregate()`. Можно группировать с помощью `$group` и сортировать с помощью `$sort`. Вообще это аналог `group by` в SQL.

4. Какая из функций save или update более детально позволит настроить редактирование документов коллекции?

`update` (она хотя бы работает).

5. Как происходит удаление документов из коллекции по умолчанию?

По умолчанию второй параметр метода `remove()` равен `false`, т.е. удаляются все найденные по первому параметру данные.

4 ССЫЛКИ И РАБОТА С ИНДЕКСАМИ В БАЗЕ ДАННЫХ MONGODB

4.1 ССЫЛКИ В БД

Практическое задание 4.1.1

1. *Создайте коллекцию зон обитания единорогов, указав в качестве идентификатора кратко название зоны, далее включив полное название и описание.*

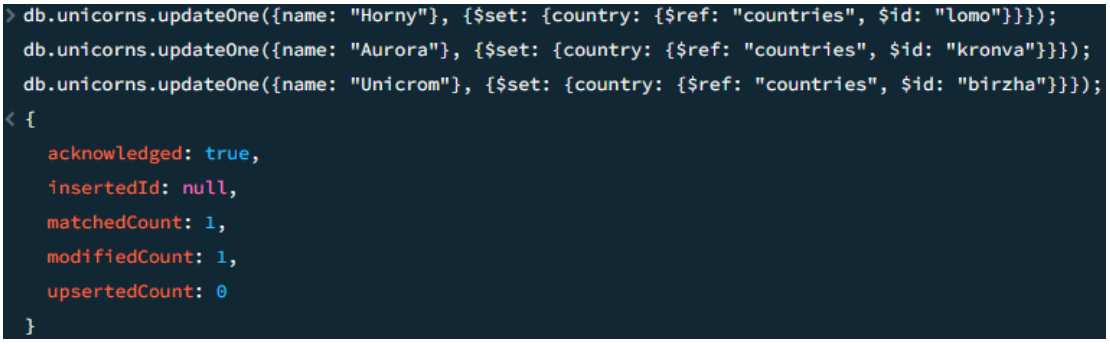
Создадим коллекцию с помощью метода `insertMany()`:

```
> db.countries.insertMany([
  {_id: "lomo", fullName: "lomonosova, 9", description: "a classroom that will never have db"},
  {_id: "kronva", fullName: "kronverkskij, 49", description: "a classroom that will have db lectures"},
  {_id: "birzha", fullName: "birzhevaya, 14-16", description: "a classroom that will have db practices"}
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': 'lomo',
      '1': 'kronva',
      '2': 'birzha'
    }
  }
```

2. *Включите для нескольких единорогов в документы ссылку на зону обитания, используя второй способ автоматического связывания.*

Для первых трех единорогов (Horny, Aurora, Unicrom) добавим ссылки. Изменение документов можно сделать с помощью метода `updateOne()` и оператора `$set`, а добавление ссылок - с помощью операторов `$ref` и `$id`. Вот такие команды получились:

```
db.unicorns.updateOne({name: "Horny"}, {$set: {country: {$ref: "countries", $id: "lomo"}}});
db.unicorns.updateOne({name: "Aurora"}, {$set: {country: {$ref: "countries", $id: "kronva"}}});
db.unicorns.updateOne({name: "Unicrom"}, {$set: {country: {$ref: "countries", $id: "birzha"}}});
```



3. *Проверьте содержание коллекции единорогов.*

Ссылки вставились (для остальных двух единорогов тоже):

4.2 НАСТРОЙКА ИНДЕКСОВ

Практическое задание 4.2.1

Проверьте, можно ли задать для коллекции `unicorns` индекс для ключа `name` с флагом `unique`.

С помощью метода `ensureIndex()` создадим индекс для имени единорогов по возрастанию и флагом *unique*: `db.unicorns.ensureIndex({"name": 1}, {"unique": true})`

```
> db.unicorns.ensureIndex({"name": 1}, {"unique": true})
< [ 'name_1' ]
```

То есть установить уникальное значение можно.

4.3 УПРАВЛЕНИЕ ИНДЕКСАМИ

Практическое задание 4.3.1

1. *Получите информацию о всех индексах коллекции* `unicorns`.

   Это можно сделать с помощью метода `getIndexes()`:

```
> db.unicorns.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { name: 1 }, name: 'name_1', unique: true }
  ]
```

2. *Удалите все индексы, кроме индекса для идентификатора.*

   У нас помимо индекса для идентификатора есть только индекс имени - его и удалим. Для этого воспользуемся методом `dropIndex()`:

```
> db.unicorns.dropIndex("name_1")
< { nIndexesWas: 2, ok: 1 }
```

3. *Попытайтесь удалить индекс для идентификатора.*

   Возвращается ошибка о невозможности удаления:

```
> db.unicorns.dropIndex("_id_")
✖ ▸ MongoServerError[InvalidOptions]: cannot drop _id index
```

Это логично, поскольку это системный индекс, создаваемый автоматически для каждого документа (он гарантирует уникальность поля `_id`).

4.4 ПЛАН ЗАПРОСА

Практическое задание 4.4.1

1. *Создайте объемную коллекцию numbers, задействовав курсор:*

   `for(i = 0; i < 100000; i++){db.numbers.insert({value: i})}`

```
> for(i = 0; i < 100000; i++){db.numbers.insert({value: i})}
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('683d657435ad23deff162601')
    }
  }
```

   Это выполняется ужасно долго - несколько минут…

2. *Выберите последних четыре документа.*

   Чтобы отобрать последние 4 документа, мы должны отсортировать по id в обратном порядке, т.е. получится вот такая команда: `db.numbers.find().sort({_id: -1}).limit(4)`

3. *Проанализируйте план выполнения запроса 2. Сколько потребовалось времени на выполнение запроса? (по значению параметра `executionTimeMillis`)*

Для получения информации о запросе используется метод `explain()`, а для того, чтобы он был детализированный, - параметр `executionStats`. После этого метода нужно ввести запрос, который мы анализируем. Получится вот такая команда: `db.numbers.explain("executionStats").find().sort({_id: -1}).limit(4)`

Тут очень много статистики:

```
> db.numbers.explain("executionStats").find().sort({_id: -1}).limit(4)
< {
    explainVersion: '1',
    queryPlanner: {
      namespace: 'learn.numbers',
      parsedQuery: {},
      indexFilterSet: false,
      queryHash: 'AB494614',
      planCacheShapeHash: 'AB494614',
      planCacheKey: 'AF8F2A5C',
      optimizationTimeMillis: 0,
      maxIndexedOrSolutionsReached: false,
      maxIndexedAndSolutionsReached: false,
      maxScansToExplodeReached: false,
      prunedSimilarIndexes: false,
      winningPlan: {
        isCached: false,
        stage: 'LIMIT',
        limitAmount: 4,
        inputStage: {
          stage: 'FETCH',
          inputStage: {
            stage: 'IXSCAN',
            keyPattern: {
              _id: 1
```

Значение параметра `executionTimeMillis: 2`.

4. *Создайте индекс для ключа `value`.*
   Индексы мы уже умеем вставлять:

```
> db.numbers.ensureIndex({value: 1})
< [ 'value_1' ]
```

5. *Получите информацию о всех индексах коллекции `numbers`.*
   Получать информацию об индексах тоже:

```
> db.numbers.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { value: 1 }, name: 'value_1' }
  ]
```

6. *Выполните запрос 2.*
   Ничего не поменялось в выводе (удивительно!):

7. *Проанализируйте план выполнения запроса с установленным индексом. Сколько потребовалось времени на выполнение запроса?*

Полную статистику вставлять не буду. Значение параметра `executionTimeMillis: 0`.

8. *Сравните время выполнения запросов с индексом и без. Дайте ответ на вопрос: какой запрос более эффективен?*

Более эффективен запрос с индексом. Он во много (во сколько точно - посчитать не могу, точно не в $\frac{5}{0} = \infty$) раз быстрее. Индекс делает сортировку практически мгновенной, так как данные уже упорядочены.

Контрольные вопросы

1. *Назовите способы связывания коллекций в MongoDB.*

Есть ручная установка ссылок (втупую) и автоматическое связывание (с помощью `DBRef` или с помощью использования операторов `$set` и `$ref`).

2. *Сколько индексов можно установить на одну коллекцию в БД MongoDB?*

64.

3. *Как получить информацию о всех индексах базы данных MongoDB?*

Обо всех индексах БД можно получить информацию с помощью метода `db.system.indexes.find()`, поскольку все индексы БД хранятся в системной коллекции `system.indexes`. Команда `db.numbers.getIndexes()` возвращает информацию об индексах в коллекции.

Выводы

Установка и настройка MongoDB выполнена успешно, несмотря на первоначальные сложности с запуском клиента `mongo`. Для работы использовался графический интерфейс MongoDB Compass.

CRUD-операции (вставка, чтение, обновление, удаление) отработаны на практике. Я выяснил, что для вставки предпочтительны методы `insertOne()` и `insertMany()` вместо устаревшего `insert()` (то же самое можно сказать про `updateOne()` и `updateMany()`, `deleteOne()` и `deleteMany()`).

Проанализировав производительность с помощью `explain("executionStats")`, я выяснил, что создание индексов ускоряет поиск и сортировку. Например, индекс для поля `value` в коллекции `numbers` из 10^6 элементов сократил время выполнения запроса с 2 мс до чуть более 0 мс. Но, кроме того, операция создания коллекции и добавления в нее нового документа занимает очень много времени.

В БД MongoDB можно создать максимум 64 индекса на коллекцию. Системный индекс `_id` нельзя удалить.

При использовании ручных ссылок поле `_id` одного документа сохраняется в другом, а при автоматическом связывании происходит подвязывание ссылки на документ из другой коллекции (при изменении документа ссылка не изменится, а данные в “подвязывающей” коллекции обновятся).

В целом операции и синтаксис MongoDB эквивалентен SQL (можно группировать, агрегировать и отбирать данные, выставлять уникальные значения и т.п.), но мне MongoDB все равно не понравилась. Может быть, это и удобное NoSQL решение, но, на мой взгляд, в промышленности удобнее использовать БД, поддерживающие SQL. И выполняются операции там быстрее. И 3.5 Гб памяти, которые занимает MongoDB, все-таки немало. Но поковыряться было интересно.

Теперь мнение о лабораторной, а не о теме. Главный минус - устаревшая методичка. Что делать с методом `save()` (кроме как использовать уже изученный устаревший, но работающий, `insert()`) я так и не разобрался. И 4 балла за такую работу (она хоть и легкая, но достаточно объемная) - очень мало.

Спасибо!