ЛАБОРАТОРНАЯ РАБОТА № 1

Введение в MongoDB

1. ***Цель работы.***

1. Изучение документов JSON.
2. Изучение создания баз данных в MongoDB.
3. Изучение создания коллекций в MongoDB.
4. Изучение переименования коллекций.
5. Изучение удаления коллекций.

2. ***Теоретическая часть.***

Название MongoDB происходит от слова humongous — «огромный», поскольку разработчики предполагали, что их новый продукт будет поддерживать чрезвычайно большие наборы данных. Она разработана для следующего.

1. Высокая доступность.
2. Высокая масштабируемость.
3. Высокая производительность.

Будучи базой данных документов, MongoDB не имеет схем и поддерживает агрегирование. Отсутствие схемы означает, что все документы не обязаны соответствовать одной и той же структуре, и структуру документов не нужно объявлять заранее. Поддержка агрегирования означает, что документы инкапсулируют все соответствующие данные, относящиеся к центральному объекту, в одном документе. Данные хранятся в документах, документы аналогичного типа хранятся в коллекциях, а связанные коллекции хранятся в базе данных. Документы форматируются с использованием для хранения JSON, который представляет собой двоичное представление JSON, но с поддержкой дополнительных функций, таких как более широкий диапазон типов данных. Для пользователей документы отображаются в виде файлов JSON, что упрощает их чтение и управление ими на различных языках программирования. Понимание базовой структуры документа JSON необходимо для работы с MongoDB.

Документы JSON

Нотация объектов JavaScript (JSON) — это формат обмена данными, который представляет данные как логический объект. Объекты заключаются в фигурные скобки {}, содержащие пары «ключ — значение». При обсуждении документов JSON пары «ключ — значение» обычно записываются в формате «ключ: значение», поэтому мы будем следовать этому соглашению при обсуждении MongoDB. В разных обстоятельствах вы можете увидеть использование терминологии «ключ: значение», «тег: значение», «поле: значение», «id: значение» или «свойство: значение», но все они означают одно и то же, поэтому не допускайте, чтобы эти изменения в терминологии запутали вас. Такое разнообразие терминологии является свидетельством широкого спектра контекстов и языков программирования, которые приняли формат JSON.

Один объект JSON может содержать множество пар «ключ: значение», разделенных запятыми. Простой документ JSON для хранения данных о книге может выглядеть так:

```
{"_id": 101, "title": "Большие данные и NoSQL базы данных"}
```

Этот документ содержит две пары «ключ: значение»:

1) «_id» — это ключ со значением 101 в качестве связанного значения;

2) «title» — это ключ с «Большие данные и NoSQL базы данных» в качестве связанного значения.

Компонент значения может иметь несколько значений, подходящих для данного ключа. В предыдущем примере добавление пары «ключ: значение» для авторов могло иметь значения «Мамедли» и «Казиахмедов». Если для одного ключа имеется несколько значений, ис-

пользуется массив. Массивы в формате JSON заключаются в квадратные скобки [ ]. Например, приведенный ранее документ можно расширить до:

```
{"_id": 101, "название": "Большие данные и NoSQL базы данных", "автор": ["Мамедли", " Казиахмедов "]}
```

Когда документы JSON предназначены для чтения людьми, они часто отображаются с каждой парой «ключ: значение» на отдельной строке, чтобы улучшить читаемость, например:

```
{
      "_id": 101,
      "название": "Большие данные и NoSQL базы данных",
      "автор":
            [
                  "Мамедли",
                  " Казиахмедов"
            ]
}
```

Объекты также могут иметь другие объекты, встроенные в качестве значения. Если документ JSON имеет встроенный объект, этот внедренный объект часто называют вложенным документом. Рассмотрим еще один простой документ с данными об издательстве, связанном с книгой из предыдущего примера:

```
{
      "название": "Издательство НВГУ",
      "адрес": "улица Дзержинская, 11",
      "город": "Нижневартовск",
      "область": "ХМАО-Югра"
}
```

Документ книги и документ издательства связаны. Помните, что базы данных документов поддерживают коллекции, поэтому документы обычно группируются вокруг центрального объекта. Если мы создаем документы, в которых книга является центральной сущностью, мы можем захотеть включить данные об издательстве в этот документ. Чтобы включить информацию об издательстве в книгу, мы могли бы встроить его в качестве вложенного документа в документ книги следующим образом:

```
{
      "_id": 101,
      "название": "Большие данные и NoSQL базы данных",
      "автор":
      [
            "Мамедли",
            " Казиахмедов"
      ],
      "издательство":
      {
            "название": "Издательство НВГУ",
            "улица": "Дзержинская, 11",
            "город": "Нижневартовск",
            "область": "ХМАО-Югра"
      }
}
```

В этом случае мы избежали ситуации, которая потребовала бы соединения в реляционной среде. В реляционной среде мы бы использовали таблицу КНИГА и таблицу ИЗДАТЕЛЬСТВО с отношением 1:М. По сути, мы заранее объединили данные о книге и издательстве в один документ. Хотя это увеличивает избыточность, базы данных NoSQL часто жертвуют избыточность для улучшения масштабируемости. Помните, что с помощью баз данных документов мы пытаемся избежать необходимости в соединениях, делая документы независимыми друг от друга, чтобы их можно было легко масштабировать на множество компьютеров в кластере.

Помимо предварительного объединения данных, вложенные документы могут быть полезны, когда значение состоит из более мелких значимых компонентов. В предыдущем примере пары улица, город и область вместе составляют адрес. Поэтому можно создать объект адреса, включающий в себя следующие свойства:

```
{
      "_id": 101,
      "название": "Большие данные и NoSQL базы данных",
      "автор":
      [
            "Мамедли",
            " Казиахмедов"
      ],
      "издательство":
      {
            "название": "Издательство НВГУ",
            "адрес":
            {
                  "улица": "Дзержинская, 11",
                  "город": "Нижневартовск",
                  "область": "ХМАО-Югра"
            }
      }
}
```

Этот документ организован вокруг книги. Документ книги имеет четыре пары «ключ: значение». Третья пара «ключ: значение» (автор) имеет массив из нескольких значений. Четвертая пара (издательство) имеет объект в качестве значения. Объект публикации состоит из двух пар «ключ: значение», вторая из которых (адрес) — это объект, состоящий из трех пар «ключ: значение». Формат JSON позволяет использовать неопределенное конечное количество объектов, которые могут быть встроены друг в друга и, возможно, даже иметь массивы объектов. Например, массив авторов в нашем примере можно расширить как:

```
{
      "_id": 101,
      "название": "Большие данные и NoSQL базы данных",
      "автор":
            [
                  {
                        "фамилия": "Мамедли",
                        "электронная_почта": "prog-nv@mail.ru",
                        "телефон": "89876543210"
                  },
                  {
                        "фамилия": " Казиахмедов",
                        "электронная_почты": "ktofik@yandex.ru",
                        "кабинет": "208"
                  }
            ],
      "издательство":
            {
                  "название": "Издательство НВГУ",
                  "адрес":
                        {
                              "улица": "Дзержинская, 11",
                              "город": "Нижневартовск",
                              "область": "ХМАО-Югра"
                        }
            }
}
```

Формат JSON очень удобный, гибкий и мощный. Он может представлять данные разными способами, поэтому программисты могут найти представление, которое лучше всего

соответствует потребностям каждого отдельного приложения. JSON хорошо подходит для моделей данных без схемы, поскольку нет необходимости, чтобы каждый документ в коллекции имел одинаковые пары «ключ: значение». Обратите внимание, например, что существуют различия в парах «ключ: значение» для двух объектов автора в массиве автора на нашей иллюстрации.

По мере того, как вы освоитесь работы с базами данных документов, может оказаться полезным установление связей с реляционными темами, с которыми вы уже знакомы. Коллекцию в MongoDB можно рассматривать как аналогичную таблице в реляционной базе данных, поскольку обе они содержат данные по определенной теме. Очевидно, что данные в коллекции учитывают агрегирование, поэтому это совсем другой набор данных, чем тот, который отображается в реляционной таблице. Документ похож на строку данных в реляционной таблице. Пары «ключ: значение» можно рассматривать как столбец и значение в столбце. Поскольку каждый документ (строка) может иметь разные пары «ключ: значение», это все равно, что разрешить каждой строке таблицы иметь разные столбцы (таким образом, базы данных документов не имеют схемы). Массив в паре «ключ: значение» — это способ обработки многозначных атрибутов в базе данных документов. Встроенные объекты в документе могут соответствовать составным атрибутам или предварительно объединенным таблицам. Использование массивов и встроенных объектов позволяет базе данных документов учитывать агрегирование и избегать использования нескольких таблиц, для которых потребовалось бы объединение объединений для целей отчетности.

Создание баз данных и коллекций в MongoDB

Доступ к реляционным базам данных возможен через широкий спектр языков прикладного программирования, таких как JavaScript, Java, Python и PHP, а также и MongoDB. MongoDB предоставляет встроенную программу-оболочку MongoDB, которая предоставляет интерфейс для непосредственной работы с данными. С помощью оболочки данные запрашиваются с использованием языка запросов Mongo (MQL). MQL основан на JavaScript, и многие программные функции JavaScript, такие как операторы IF, объявление и манипулирование переменными, а также циклы, доступны непосредственно в MongoDB. Хотя полное программирование на JavaScript выходит за рамки этой книги, важно рассмотреть основы манипулирования данными в MQL.

Базы данных MongoDB содержат коллекции документов. На каждом сервере MongoDB может размещаться множество баз данных. На сервере MongoDB (также называемом хостом) каждая база данных представляет собой тип объекта. Как мы знаем из предыдущего обсуждения формата JSON, объекты могут содержать другие объекты. Объект базы данных содержит коллекции. Коллекции также являются объектами. Объекты коллекции содержат объекты документа. Помимо содержания данных, объект также может иметь методы — запрограммированные функции для управления объектом. При подключении к серверу MongoDB первая задача — указать, с каким объектом базы данных вы хотите работать. Список доступных на сервере баз данных можно получить командой:

```
show dbs
```

```
>_MONGOSH

> show dbs
< admin    40.00 KiB
  config   72.00 KiB
  local    40.00 KiB
  учебная   8.00 KiB
```

Рис. 19

Базы данных MongoDB

По умолчанию новая установка MongoDB включает базу данных администратора и локальную базу данных, как показано на рисунке 19. База данных администратора используется для записи данных о проблемах администрирования базы данных, таких как пользователи, роли и привилегии для баз данных, размещенных на этом сервере. Локальная база данных используется для хранения данных о процессе запуска сервера и роли сервера в операциях сегментирования. Напомним, что базы данных, поддерживающие агрегирование, пытаются разбить данные на части, называемые сегментами. По мере увеличения объема базы данных MongoDB может корректировать способ распределения сегментов между узлами кластера, чтобы сбалансировать рабочую нагрузку. Если вы не администрируете сервер MongoDB, вам, скорее всего, не придется беспокоиться об администраторских и локальных базах данных, поскольку их никогда не следует использовать для хранения данных конечных пользователей.

Все команды манипулирования данными в MongoDB должны быть направлены в конкретную базу данных. Создать новую базу данных в MongoDB так же просто, как ввести команду use. Команда use сообщает серверу, какая база данных должна быть целью последующих команд. Если существует база данных с указанным именем, то эта база данных будет использоваться для последующих команд. Если базы данных с таким именем нет, она создается автоматически. Например, следующая команда создает базу данных с именем «учебная», которую мы будем использовать в некоторых последующих примерах кода:

```
use учебная
```

Эта команда создает базу данных с именем «учебная» и присваивает специальной переменной MongoDB с именем db значение «учебная». В последующих командах мы будем использовать переменную db для указания базы данных, на которую нацелены наши команды. Имя используемой базы данных можно отобразить с помощью метода getName() для переменной db следующим образом:

```
db.getName()
```

Хотя между коллекциями и реляционными таблицами существует концептуальное сходство, различия становятся гораздо более очевидными при создании коллекции. Создание таблицы в реляционной базе данных требует указания большого количества метаданных. Имя и тип данных для каждого атрибута, любые ограничения столбцов, ограничения первичного ключа, ограничения внешнего ключа и проверочные ограничения указываются во время создания таблицы. Поскольку базы данных документов не имеют схемы и не имеют предопределенных структур, ни один из этих параметров не является обязательным. Для создания коллекции требуется лишь указать базу данных, которой она принадлежит, и имя коллекции. Использование метода createCollection() с переменной db создает коллекцию с указанным именем. Следующая команда создает коллекцию «страны» внутри ранее определенной демонстрационной базы данных:

```
db.createCollection("страны")
```

Чтобы отобразить коллекции, существующие в базе данных, можно использовать следующую команду, как показано на рисунке 20:

```
show collections
```

Теперь, когда демонстрационная база данных имеет хотя бы одну коллекцию, она отображается как одна из доступных баз данных, возвращаемых командой show dbs. Хотя при создании коллекции не было необходимости указывать параметры, существует несколько доступных опций. Например, существует опция autoIndexID, которая действует как суррогатный ключ. Если для коллекции autoIndexID установлено значение true, СУБД автоматически добавляет к каждому документу поле «_id», которое действует как уникальный идентификатор и автоматически создавать индекс для этого поля, чтобы ускорить поиск. По умолчанию для AutoIndexID установлено значение true.

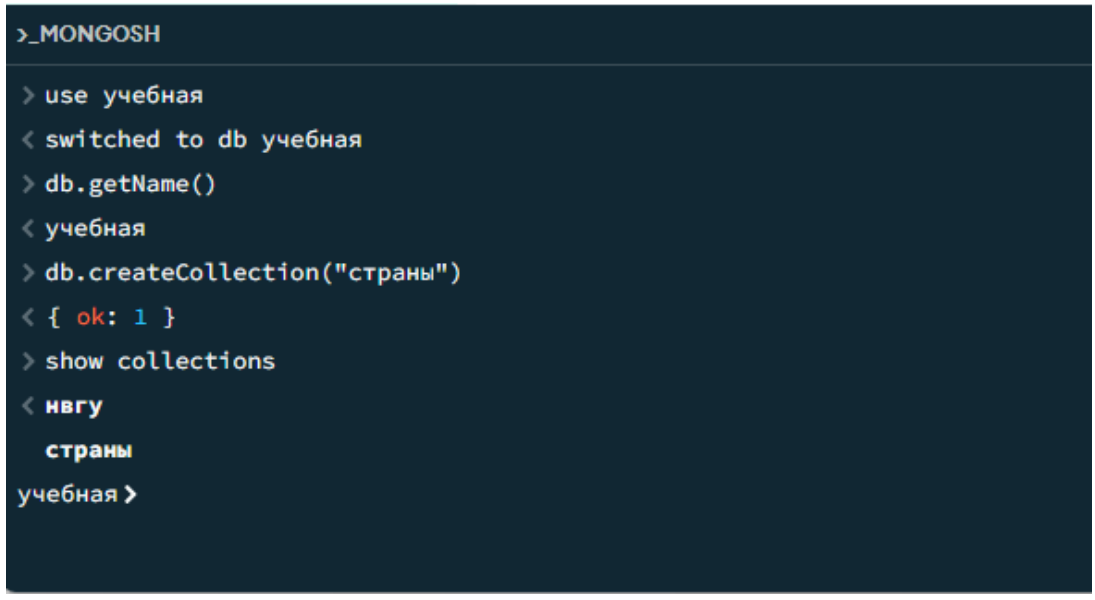

Рис. 20

Коллекции MongoDB

Используя параметры создания коллекции, в MongoDB также можно создать ограниченную коллекцию. Ограниченная коллекция — это коллекция MongoDB, которая может вырасти только до указанного максимального размера; затем документы из него автоматически удаляются. Когда ограниченная коллекция достигает максимального размера и добавляются новые документы, самые старые документы автоматически удаляются, чтобы освободить место для новых документов. Ограниченные коллекции часто используются для файлов журналов, где интерес представляет только ограниченное количество последних действий. Ограниченные коллекции могут быть ограничены по общему размеру хранилища и/или количеству документов. Например, следующая команда создает коллекцию с именем `userlog`, ограниченную 1000 документами и размером 1 мегабайт:

```
db.createCollection("userlog", {capped: true, size: 1048576, max: 1000})
```

Как вы могли заметить, большинство команд MQL заимствуют свой синтаксис из JavaScript и формата JSON.

Переименование и удаление коллекций

Иногда может возникнуть необходимость переименовать коллекции в MongoDB. Чтобы переименовать коллекцию, вы используете метод коллекции `renameCollection()` и указываете новое имя коллекции в качестве параметра. Например, следующая команда переименовывает коллекцию «Страны» в «Страна»:

```
db.страны.renameCollection("страна")
```

С помощью команды `showcollections` вы можете увидеть, что имя коллекции изменилось (см. рис. 21).

При удалении коллекции удаляются все документы в коллекции и все индексы, созданные с помощью этой коллекции. Удаление коллекции осуществляется с помощью метода коллекции `drop()`. Метод `drop()` не требует каких-либо опций или параметров. Он удаляет любую коллекцию, с которой он был вызван. Например, следующая команда удаляет коллекцию пользовательских журналов, созданную ранее: `db.userlog.drop()`.

3. ***Задание.***

1. Создать базы данных `«Деканат»`.
2. Создать коллекцию `«Сотрудники»`.

3. Переименовать коллекцию «Сотрудники» на «Сотрудник».
4. Показать все коллекции.

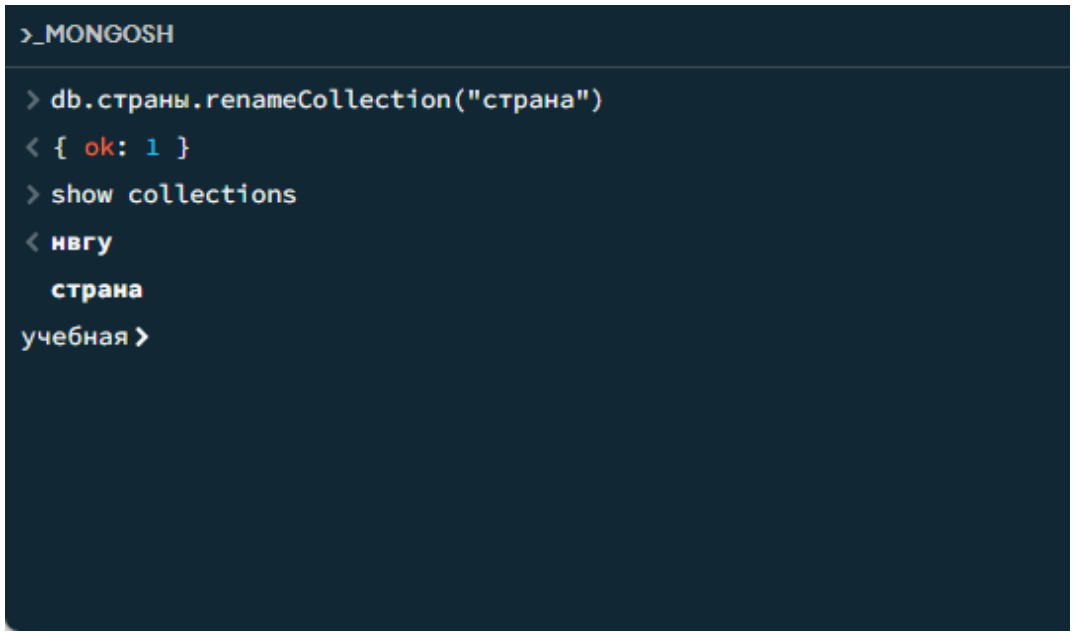

Рис. 21

Переименование коллекций

ЛАБОРАТОРНАЯ РАБОТА № 2

Редактирование документов в MongoDB

1. *Цель работы.*

1. Вставка документов в MongoDB.
2. Обновление документов в MongoDB.
3. Обновление массивов с помощью `$push`, `$pull` и `$addToSet`.
4. Обновление массивов с использованием `$each` и `$pullAll`.
5. Переименование ключей с помощью `$rename`.
6. Удаление документов в MongoDB.

2. *Теоретическая часть.*

Вставка документов в MongoDB

После создания коллекции в нее можно добавлять документы. Для добавления в коллекцию могут использоваться два метода:

1) `insertOne()`: добавляет один документ;

2) `insertMany()`: добавляет несколько документов.

Синтаксис метода `insertOne()` в MongoDB следующий:

```
db.<имя коллекции>.insertOne({document})
```

Рассмотрим следующий документ JSON:

```
{"название":"Россия","столица":"Москва","площадь":1
7151442,"население":146544700,"континент":"Европа"}
```

Чтобы вставить предыдущий документ в коллекцию стран, которую мы создали ранее, мы вызываем метод коллекции `insertOne()` и предоставляем документ в качестве параметра следующим образом:

```
db.страна.insertOne(
{
"название":"Россия",
"столица":"Москва",
"площадь":17151442,
"население":146544700,
"континент":"Европа"
})
```

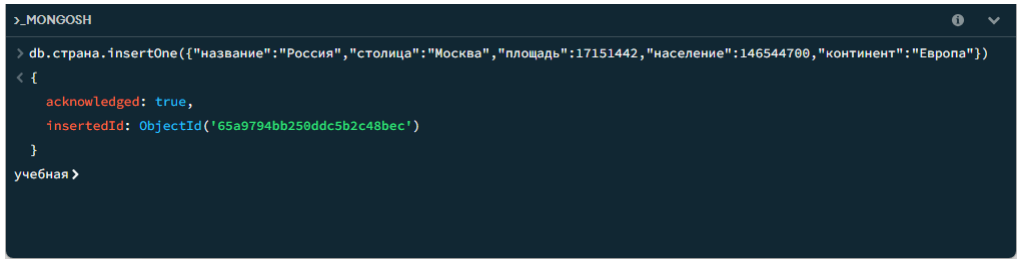

Рис. 22

Вставка документов в MongoDB

Для получения документов в MongoDB из нашей коллекции используем метод `find()`. Добавление метода `pretty()`, о котором пойдет речь далее, улучшает читаемость возвращаемого документа. Следующая команда отображает все документы в коллекции продуктов, как показано на рисунке 23:

```
db.страна.find()
```

Обратите внимание, что каждый документ имеет ключ с именем `_id`, который содержит `ObjectID`. Как обсуждалось ранее, по умолчанию для новых коллекций свойство

autoIndexID имеет значение true, поэтому столбец _id генерируется автоматически при вставке документов. Значения _id, сгенерированные в ваших коллекциях, будут отличаться от показанных на рисунках, поскольку характеристики главного компьютера кодируются как часть поля _id, но они всегда будут уникальными в вашей коллекции.

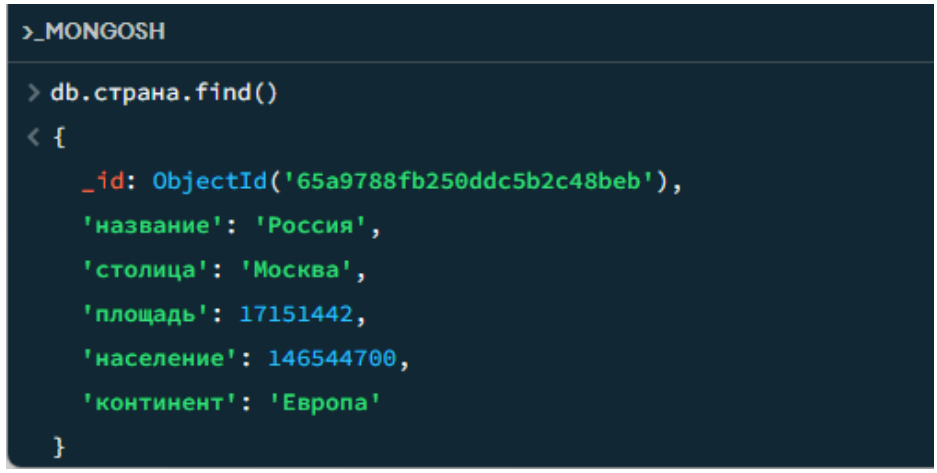

Рис. 23

Получение документов

Далее вставим в ту же коллекцию еще один документ, но с немного другой структурой. Помните, что базы данных документов не имеют схемы, поэтому все документы в коллекции не обязательно должны иметь одинаковую структуру.

```
db.страна.insertOne({
"ISO":"AZ",
"название":"Азербайджан",
"столица":"Баку",
"площадь":"86600",
"население":10103512,
"континент":"Азия",
"домен":".az",
"соседи":
[
"GE",
"IR",
"AM",
"TR",
"RU"
],
"деньги":"манат",
"телефон":"994"
})
```

Метод insertOne() можно использовать с документами JSON любой сложности. Предыдущий пример включает массив значений для ключа «соседи».

Обновление документов в MongoDB

Существует множество вариантов обновления документов в MongoDB. На самом базовом уровне обновления могут быть либо обновлениями оператора, либо заменяющими обновлениями. Обновление оператора вносит изменения в часть содержимого документа, но оставляет остальную часть документа неизменной. Это похоже на обновление реляционной базы данных. Обновление замены полностью удаляет исходный документ и заменяет его новым документом, сохраняя при этом только значение первичного ключа _id. Для этого применяются функции updateOne() (обновляет только один документ) и updateMany() (обновляет множество документов). Для обновления отдельных полей в этих функциях применяется оператор $set. Если документ не содержит обновляемое поле, то оно создается.

Следующая команда выполняет обновление документа «Россия», который был первый вставлен в коллекцию стран:

```
db.страна.updateOne(
{
"название":"Россия"
},
{
$set:
{
"название":"Российская Федерация"
}
})
```

```
>_MONGOSH
> db.страна.updateOne({"название":"Россия"},{$set:{"название":"Российская Федерация"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.страна.find()
< {
    _id: ObjectId('65a97bb9b250ddc5b2c48bed'),
    'название': 'Российская Федерация',
    'столица': 'Москва',
    'площадь': 17151442,
    'население': 146544700,
    'континент': 'Европа'
  }
  {
```

Рис. 24

Обновление документа

Если посмотреть на документ, показанный на рисунке 24, видно, что документ «Россия» был заменен на «Российская Федерация». Сравнение с рисунком 22 показывает, что _id документа остается прежним.

Объект изменения может включать в себя несколько пар «ключ: значение», включая сложные значения, такие как массивы и встроенные объекты.

Обновление массивов с помощью $push, $pull и $addToSet

Изменение значений в массиве может немного отличаться. Чтобы изменить значения в массиве, можно использовать оператор $set, но весь массив должен быть установлен одновременно. Документ «Азербайджан», который был вставлен ранее, содержал ключ «соседи», который содержит массив [«GE», «IR», «AM», «TR», «RU»]. Дополнительное ключевое слово «TM» можно добавить к массиву с помощью оператора $set следующим образом:

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$set:
{
"соседи":
[
"GE",
```

```
"IR",
"AM",
"TR",
"RU",
"TM"
]
}
})
```

Хотя это и дает требуемый результат, но не является нормальной практикой. В большинстве случаев программист пытается добавить или удалить значения из массива, а не заменить каждое значение в массиве. Использование `$set` в этой ситуации неэффективно. Это удобно, потому что сначала потребуется получить весь массив, чтобы узнать, какие значения уже существуют в массиве, чтобы их можно было включить в операцию обновления. Обычно значения добавляются в массив или удаляются из него без необходимости сначала получать массив. Это делается с помощью операторов `$push` и `$pull`.

Оператор `$push` используется для добавления значения в массив. Оператор `$pull` используется для удаления значения из массива. Так же, как `$set` и `$unset`, `$push` и `$pull` принимают параметр объекта, описывающий вносимые изменения. Например, чтобы добавить соседа «KZ» в документ «Азербайджан», можно использовать следующую команду (результат показан на рис. 25):

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$push:
{
"соседи":"KZ"
}
})
```

```
>_MONGOSH
> db.страна.updateOne({"название":"Азербайджан"},{$push:{"соседи":"KZ"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.страна.find().pretty()
  {
    _id: ObjectId('65aacb28b250ddc5b2c48bf0'),
    ISO: 'AZ',
    'название': 'Азербайджан',
    'столица': 'Баку',
    'площадь': '86600',
    'население': 10103512,
    'континент': 'Азия',
    'домен': '.az',
    'соседи': [
      'GE',
      'IR',
      'AM',
      'TR',
      'RU',
      'KZ'
    ],
    'деньги': 'манат',
    'телефон': '994'
  }
```

Рис. 25

Изменение массива

Объект запроса сообщает методу `updateOne()` найти документ со значением «Азербайджан» в ключе имени. Объект изменения сообщает методу, что нужно добавить или добавить значение «KZ» в массив значений для ключа соседи.

Оператор `$pull` имеет тот же синтаксис, что и оператор `$push`. Единственное отличие состоит в том, что `$pull` удаляет значение из массива, а не добавляет его в массив. Ключевое слово «KZ» можно удалить из массива с помощью оператора `$pull` следующим образом (результаты показаны на рисунке 26):

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$pull:
{
"соседи":"KZ"
}
})
```

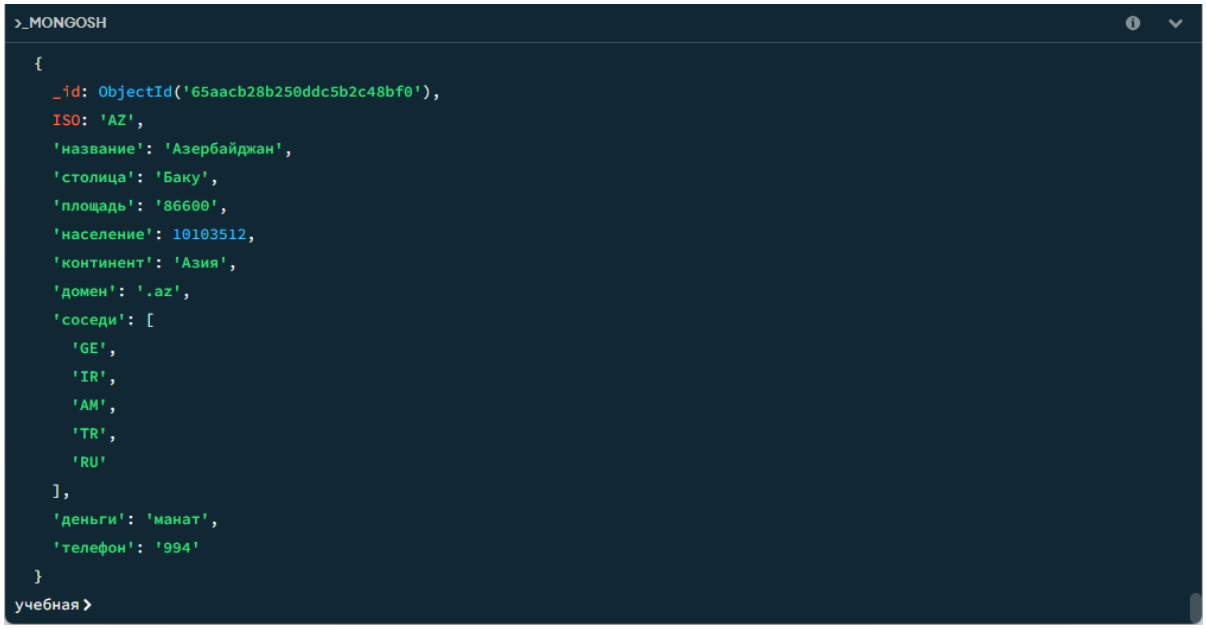

Рис. 26

Удаление из массива

Изучая результаты обновления на рисунке 26, обратите внимание, что другие значения в массиве не были затронуты.

Возможно, а в некоторых случаях желательно, чтобы массив содержал повторяющиеся значения. Если массив содержит список веб-сайтов, посещенных пользователем, нам может потребоваться знать каждый раз, когда пользователь посещал данный веб-сайт, даже если это означает, что веб-сайт появляется в массиве несколько раз. В других случаях мы не хотим, чтобы в массиве появлялись дубликаты. В соседях для «Азербайджана» нет смысла хранить соседа «KZ» несколько раз. Оператор `$push` позволяет дублировать значения массива. Если бы вы повторно выполнили предыдущую команду, чтобы добавить слово «KZ» в массив соседей, значение появилось бы более одного раза. Чтобы позволить программистам добавлять значение в массив только в том случае, если оно еще не существует в массиве, используется оператор `$addToSet`. Оператор `$addToSet` использует тот же синтаксис и работает так же, как и оператор `$push`, за исключением того, что он не добавляет значение в массив, если это значение уже существует в массиве. Следующую команду можно использовать для добавления значения «TM» в массив ключевых слов:

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$addToSet:
{
"соседи":"TM"
}
})
```

Эта команда добавляет соседа «TM» в массив соседей, поскольку значения «TM» еще не было в массиве. Если вы используете оператор $addToSet со значением, которое уже существует в массиве, значение не будет добавлено снова, как показано ниже.

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$addToSet:
{
"соседи":"KZ"
}
})
```

Если значение появляется в массиве несколько раз, оператор $pull удалит все вхождения этого значения из массива.

Обновление массивов с использованием $each и $pullAll

Предыдущие операторы очень хороши при работе с отдельными значениями в массивах, будь то одно простое значение или значение объекта. Однако, поскольку массивы предназначены для хранения нескольких значений, нам часто хочется изменить содержимое массива с несколькими значениями одновременно. Невозможно обновить массив значений с помощью $push, $pull и $addToSet. К счастью, для выполнения этой задачи необходимы лишь небольшие модификации. Чтобы удалить массив значений из существующего массива, используйте оператор $pullAll вместо оператора $pull. Синтаксис оператора $pullAll аналогичен оператору $pull, за исключением того, что он принимает массив значений вместо одного значения. Следующая команда удаляет значения «KZ» и «TM» из массива соседей, как показано на рисунке 27.

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$pullAll:
{
"соседи":
[
"KZ",
"TM"
]
}
})
```

$push и $addToSet можно использовать вместе с оператором $each. Оператор $each модифицирует $push и $addToSet для перебора или цикла по набору значений и воздействия на каждое из них отдельно. Например, если мы хотим добавить значения «KZ» и «TM» в массив «соседи», следующая команда объединяет $push и $each для выполнения этой задачи, как показано на рисунке 28.

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$push:
{
"соседи":
{
$each:
[
"KZ",
"TM"
]
}
}
})
```

```
>_MONGOSH
{
  _id: ObjectId('65aacb28b250ddc5b2c48bf0'),
  ISO: 'AZ',
  'название': 'Азербайджан',
  'столица': 'Баку',
  'площадь': '86600',
  'население': 10103512,
  'континент': 'Азия',
  'домен': '.az',
  'соседи': [
    'GE',
    'IR',
    'AM',
    'TR',
    'RU'
  ],
  'деньги': 'манат',
  'телефон': '994'
}
учебная >
```

Рис. 27

Множественное удаление из массива

```
>_MONGOSH
{
  _id: ObjectId('65aacb28b250ddc5b2c48bf0'),
  ISO: 'AZ',
  'название': 'Азербайджан',
  'столица': 'Баку',
  'площадь': '86600',
  'население': 10103512,
  'континент': 'Азия',
  'домен': '.az',
  'соседи': [
    'GE',
    'IR',
    'AM',
    'TR',
    'RU',
    'KZ',
    'TM'
  ],
  'деньги': 'манат',
  'телефон': '994'
}
```

Рис. 28

Добавление нескольких значений в массив

Как и раньше, оператор `$push` принимает параметр объекта. При добавлении одного значения в массив параметр объекта представляет собой простую пару «ключ: значение». При добавлении массива значение в паре «ключ: значение» представляет собой объект, который содержит собственную пару «ключ: значение» с оператором `$each` в качестве ключа и массивом в качестве значения. Этот синтаксис также используется, когда оператор `$each` используется с оператором `$addToSet`. Следующая команда пытается добавить значения «KZ» и «UZ» в массив соседей Азербайджана, результаты показаны на рисунке 29.

```
db.страна.updateOne(
{
"название":"Азербайджан"
},
{
$addToSet:
{
"соседи":
{
$each:
[
"KZ",
"UZ"
]
}
}
})
```

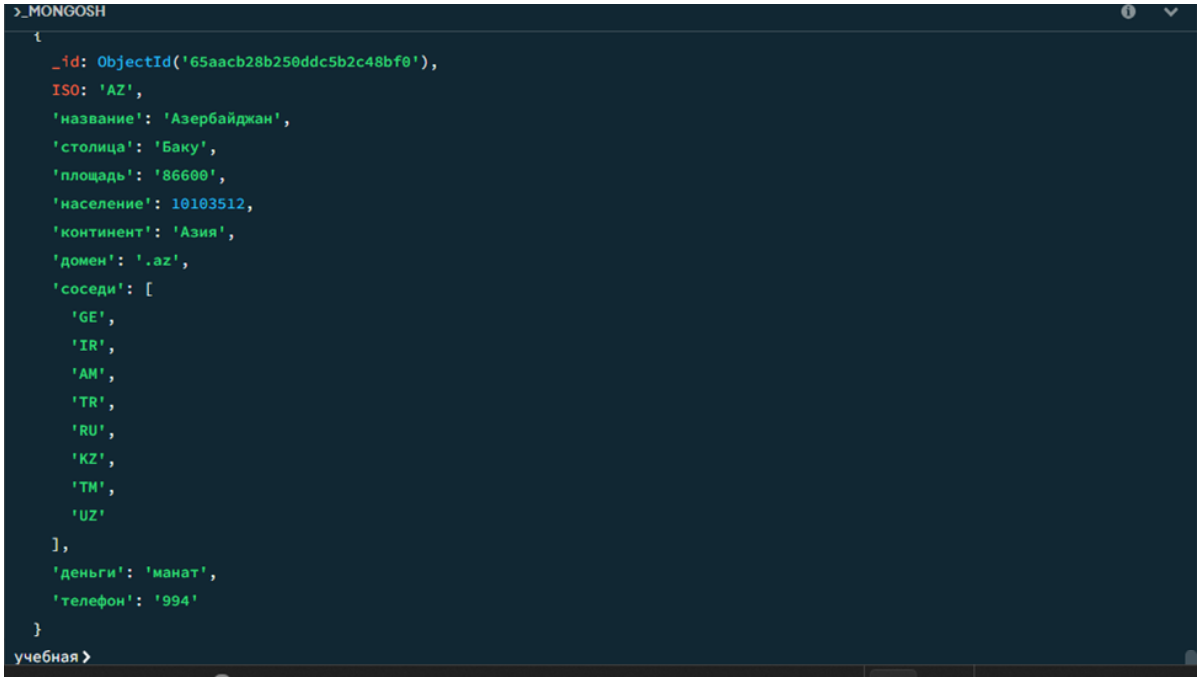

Рис. 29

Добавление уникальных значений в массив

Глядя на рисунок 29, обратите внимание, что значение «KZ» появляется только один раз, поскольку `$addToSet` не создает повторяющиеся значения в массиве.

Переименование ключей с помощью `$rename`

Все предыдущие операции обновления работали над изменением значений в парах «ключ: значение». Также можно изменить ключ в паре «ключ: значение» с помощью оператора `$rename`. Оператор `$rename` также использует параметр объекта. В этом случае параметр объекта содержит простую текстовую пару «ключ: значение», в которой ключ содержит имя ключа, который нужно переименовать, а значение представляет собой текст, на который

следует изменить имя ключа, например {<старое_имя>:<"новое_имя">}. Компонент значения, содержащий новое имя, считывается MongoDB как значение при проверке синтаксиса команды. Поэтому новое значение должно быть заключено в кавычки, поскольку это текстовое значение. Например, следующая команда переименовывает ключ «континент» на «часть_света» для всех документов в коллекции, как показано на рисунке 30:

```
db.страна.updateMany(
{},
{
$rename:
{
"континент": "часть_света"
}
} )
```

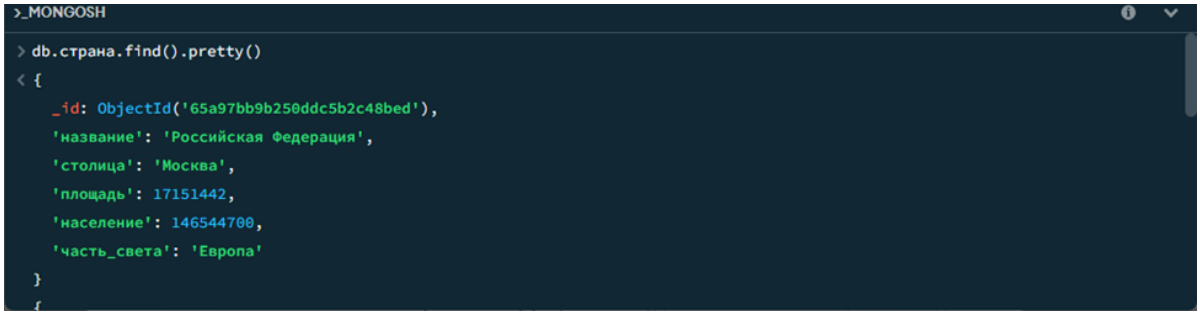

Рис. 30

Результат изменения ключа

Обратите внимание, что объект запроса в этой команде пуст. Включение объекта запроса является обязательным, но объект может быть пустым. Пустой объект запроса соответствует каждому документу, поэтому операция обновления выполняется для каждого документа в коллекции.

Удаление документов в MongoDB

Удаление документов из коллекции MongoDB выполняется просто и имеет несколько вариантов. Для удаления документов в MongoDB предусмотрены функции `deleteOne()` — удаляет один документ и `deleteMany()` — позволяет удалить несколько документов. В качестве параметра в эти функции передается фильтр удаляемых документов. Как и в случае с методом `updateMany()`, объект запроса может быть пустым объектом, который удалит каждый документ в коллекции, но это редко используется на практике. Следующий код удаляет документ «Российская Федерация»:

```
db.страна.deleteOne(
{
"название": "Российская Федерация"
} )
```

3. *Задание.*

1. Создать коллекцию «Страна».
2. Добавить в коллекцию «Страна» следующие документы:

```
{
        "ISO":"AD",
        "Название":"Andorra",
        "Столица":"Andorra la Vella",
        "Площадь":"467.63",
        "Население":82887,
        "Континент":"EU",
        "Домен":".ad",
        "Деньги":"Euro",
        "Телефон":"376"
```

```
},
{
      "ISO":"AE",
      "Название":
      "United Arab Emirates",
      "Столица":"Abu Dhabi",
      "Площадь":"82880",
      "Население":4975593,
      "Континент":"AS",
      "Домен":".ae",
      "Деньги":"Dirham",
      "Телефон":"971"
},
{
      "ISO":"AF",
      "Название":"Afghanistan",
      "Столица":"Kabul",
      "Площадь":"647500",
      "Население":29121286,
      "Континент":"AS",
      "Домен":".af",
      "Деньги":"Afghani",
      "Телефон":"93"
},
{
      "ISO":"AG",
      "Название":"Antigua and Barbuda",
      "Столица":"St. John's",
      "Площадь":"443",
      "Население":86754,
      "Континент":"NA",
      "Домен":".ag",
      "Деньги":"Dollar",
      "Телефон":"+1-268"
}
```


3. Для документа «Afghanistan» добавить «соседи»: «TM», «CN», «IR», «TJ», «PK», «UM».

4. «Население» для документа «United Arab Emirates» исправить на 10207863.

5. Для документа «Afghanistan» изменить «соседи», исправляя «UM» на «UZ».

ЛАБОРАТОРНАЯ РАБОТА № 3

Выбор данных в MongoDB

1. ***Цель работы.***

1. Запрос документов в MongoDB.
2. Выбор и ограничение с помощью метода `find()`.
3. Разбивка документа на страницы.
4. Сортировка документов с помощью `sort()`.
5. Объединение критериев с помощью `$and` и `$or`.

2. ***Теоретическая часть.***

Запрос документов в MongoDB

Коллекция документов, используемых для демонстрации запросов MongoDB, основана на данных, использованных в предыдущих лабораторных работах. Используемая здесь часть модели состоит из документов, центральным объектом которых является страна. Документы имеют следующую структуру:

```
{_id: <сгенерированный системой ObjectID,
название: <название страны, которое будет отображаться>,
столица: <название столицы>,
площадь: <площадь страны, вещественное число>,
население: <население страны, целое число>,
континент: <часть света, где находится страна>
}
```

По умолчанию все поисковые запросы в MongoDB чувствительны к регистру. Чтобы сделать запрос нечувствительным к регистру, обычно требуется использование метода программирования, называемого регулярными выражениями, который может очень отрицательно сказаться на производительности MongoDB. Поэтому на практике принято хранить текстовые значения, в которых заглавные буквы имеют значение дважды: один раз с заглавной буквы так, как она должна отображаться, и один раз с прописными или строчными буквами для упрощения запросов.

Выбор и ограничение с помощью метода `find()`

Основным методом, используемым для получения документов в MongoDB, является метод `find()`. Метод коллекции `find()` извлекает из коллекции документы, соответствующие предоставленным ограничениям. Метод принимает два параметра объекта: объект запроса, определяющий критерии, которым документы должны соответствовать для включения, и объект проекции, определяющий, какие ключи будут включены в возвращаемые документы. Оба параметра являются необязательными. Синтаксис метода `find()` следующий:

```
find(
{
<запрос>
},
{
<проекция>
})
```

Параметр объекта запроса действует аналогично предложению WHERE запроса SQL `SELECT`. Предоставление простой пары «ключ: значение» в параметре запроса интерпретируется как условие равенства. Например, следующая команда извлекает страну с отображаемым названием «`Российская Федерация`», как показано на рисунке 31:

```
db.страна.find(
{
название: "Российская Федерация"
```

```
})
```

```
>_MONGOSH
> db.страна.find({название: "Российская Федерация"})
< {
    _id: ObjectId('65ad61b0b250ddc5b2c48bf1'),
    'название': 'Российская Федерация',
    'столица': 'Москва',
    'площадь': 17151442,
    'население': 146544700,
    'континент': 'Европа'
  }
учебная >
```

Рис. 31

Выбор страны с названием «Российская Федерация»

В предыдущей команде параметр объекта запроса — `{название: "Российская Федерация"}`. Это эквивалент команды `WHERE название = "Российская Федерация"` в запросе SQL `SELECT`.

Параметр объекта проекции используется для указания того, какие пары «ключ: значение» возвращаются. Объект проекции может содержать одну или несколько пар «ключ: значение». Ключи в объектах проецирования — это ключи проецируемого документа. Значением каждого ключа в объекте проекции является значение 0 или 1. Указание значения 1 с помощью ключа указывает на то, что эта пара «ключ: значение» из документа должна быть включена в результаты. Указание значения 0 с помощью ключа указывает на то, что эту пару «ключ: значение» из документа следует опустить. Например, следующая команда извлекает документ для страны «Российская Федерация», но возвращает только _id, название и население, как показано на рисунке 32:

```
db.страна.find(
{
название: "Российская Федерация"
},
{
название:1,
население:1
})
```

```
>_MONGOSH
> db.страна.find({название: "Российская Федерация"},{название:1, население:1})
< {
    _id: ObjectId('65ad61b0b250ddc5b2c48bf1'),
    'название': 'Российская Федерация',
    'население': 146544700
  }
учебная >
```

Рис. 32

Выбор страны с названием и населением

Обратите внимание, что ключ `_id` возвращается по умолчанию, даже если он не указан. Чтобы вернуть все ключи, кроме указанных, используйте значение 0 в объекте проекции. Например, следующая команда возвращает все ключи в документе «Российская Федерация», кроме континента, как показано на рисунке 33:

```
db.страна.find(
{
название: "Российская Федерация"
},
{
континент:0
```

```
})
```

```
>_MONGOSH
> db.страна.find({название: "Российская Федерация"},{континент:0})
< {
    _id: ObjectId('65ad61b0b250ddc5b2c48bf1'),
    'название': 'Российская Федерация',
    'столица': 'Москва',
    'площадь': 17151442,
    'население': 146544700
  }
учебная >
```

Рис. 33

Выбор страны со всеми ключами, кроме континента

Как правило, невозможно указать включение и исключение ключей в одном запросе. Исключением является ключ `_id`, который возвращается по умолчанию. Если при указании ключей для включения программист желает подавить ключ `_id`, его можно явно исключить. Следующая команда возвращает только название страны, как показано на рисунке 34:

```
db.страна.find(
{
название: "Российская Федерация"
},
{
название:1,
_id:0
})
```

```
>_MONGOSH
> db.страна.find({название: "Российская Федерация"},{название:1, _id:0})
< {
    'название': 'Российская Федерация'
  }
учебная >
```

Рис. 34

Выбор названия без `_id`

Документ JSON возвращается в плотном формате, где каждая пара «ключ: значение» разделена пробелом. Это представление документов по умолчанию, и оно подходит, когда документы очень простые или когда документ возвращается в приложение, а не для удобства чтения человеком. Чтобы улучшить читаемость возвращаемого документа, можно использовать метод `pretty()`. Метод `pretty()` в MongoDB предназначен для улучшения читаемости документов путем размещения пар «ключ: значение» в отдельных строках. Метод `pretty()` можно добавить к методу `find()`, и он не принимает никаких параметров. Следующая команда объединяет методы `find()` и `pretty()`:

```
db.страна.find(
{
название: "Российская Федерация"
}).pretty()
```

Напомним, что параметр объекта запроса и параметр объекта проекции являются необязательными. Если оба параметра опущены, метод `find()` записывается без параметров. Если параметр объекта запроса необходим, а параметр объекта проекции — нет, то можно опустить только объект проекции. Однако если параметр объекта запроса не нужен, а объект проекции нужен, то программист не может просто пропустить объект запроса. Если указан только один параметр объекта, MongoDB предположит, что этот параметр предназначен для использования в качестве параметра объекта запроса. В этих случаях для объекта запроса можно использовать пустой объект, аналогично тому, как ранее пустой объект использовал-

ся для обновления всех документов. Например, следующая команда извлекает только название каждого документа, как показано на рисунке 35:

```
db.страна.find(
{},
{
название:1,
_id:0
}).pretty()
```

Рис. 35

Название всех стран

При работе в оболочке MongoDB, если запрос возвращает более 20 документов, на экране отображаются только первые 20, после чего следует приглашение ввести «it» для отображения следующих 20. В этих случаях, все документы найдены и возвращены, но оболочка выполняет простую манипуляцию, чтобы уменьшить прокрутку экрана с данными.

Разбивка документа на страницы с помощью `limit()` и `skip()`

Поведение оболочки MongoDB, заключающееся в отображении только нескольких документов одновременно, может быть полезным, но оно довольно ограничено и может не соответствовать потребностям программиста. Другие методы можно использовать с результатами метода `find()`, чтобы улучшить поток документов как при их интерактивном просмотре в оболочке, так и при выводе их в приложение.

Когда к одной коллекции применяется несколько методов, это называется цепочкой методов. Использование метода `pretty()` с функцией `find()` в предыдущем примере было простым примером цепочки методов. Другие методы также могут быть объединены с методом `find()`. Среди наиболее распространенных — `limit()`, `skip()` и `sort()`. Метод `limit()` используется для ограничения количества возвращаемых документов. Функция `limit()` принимает один числовой параметр, определяющий количество документов, которым необходимо ограничиться. В отличие от параметра объекта запроса метода `find()`, `limit()` не является запросом на основе содержимого документов. После того как метод `find()` определил, какие документы следует вернуть, метод `limit()` просто ограничивает набор результатов указанным количеством документов. Например, следующая команда находит все документы для азиатских стран, но ограничивает результаты только первыми двумя документами.

```
db.страна.find(
{
континент: "Азия"
},
{
название:1
}).limit(2)
```

`limit()` часто используется с приложениями, которые извлекают данные для отображения пользователям, чтобы приложение или веб-сайт могли разбивать данные на блоки. Функция `skip()` также используется для поддержки нумерации страниц, позволяя извлекать последующие фрагменты документов. Например, если мы извлекаем страны, чтобы их мож-

но было отобразить для демонстрации, предыдущая команда `limit()` указывает, что мы используем фрагменты двух документов одновременно. Чтобы получить следующую страницу двух документов, нам нужно пропустить первые два документа, поскольку они уже были отображены, а затем отобразить следующие два документа. Следующая команда сделает это, как показано на рисунке 37:

```
db.страна.find(
{
континент: "Азия"
},
{
название:1
})
.skip(2)
.limit(2)
```

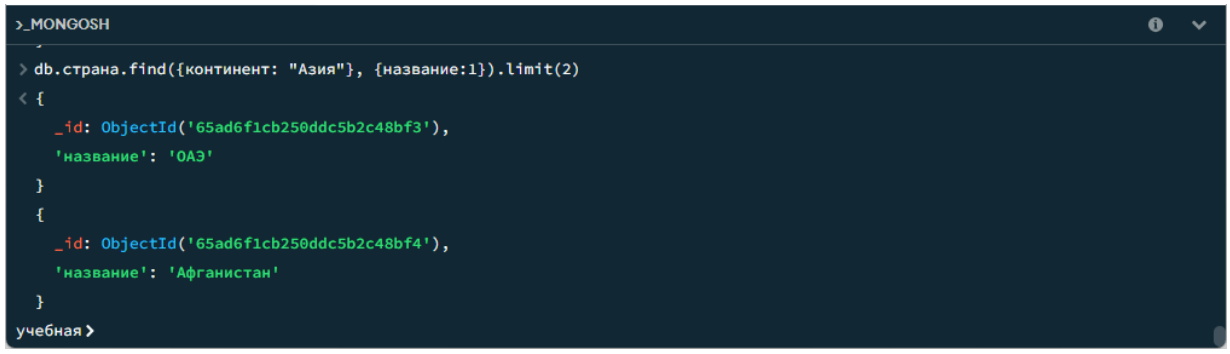

Рис. 36

Выбор первых двух стран

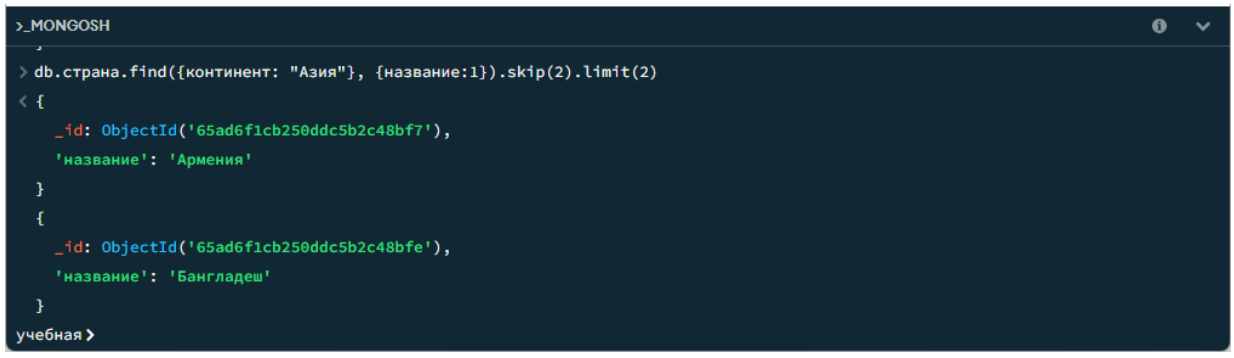

Рис. 37

Использование `skip()` и `limit()` вместе

Чтобы получить третий фрагмент документов с разбивкой на страницы, параметр пропуска необходимо увеличить до 4.

Сортировка документов с помощью `sort()`

До сих пор мы не интересовались порядком возвращаемых документов. Благодаря механизмам внутреннего хранения, используемым MongoDB, документы часто возвращаются в том порядке, в котором они были введены в коллекцию. Однако, как и в случае с запросами SELECT, мы часто хотим контролировать порядок сортировки возвращаемых данных. Метод `sort()` позволяет нам это сделать. Метод `sort()` коллекции упорядочивает полученные документы на основе значений одного или нескольких ключей. Подобно параметру объекта проекции метода `find()`, метод `sort()` принимает объект, состоящий из одного или нескольких ключей в паре с числовыми значениями. В случае проекции значения были 0 или 1. При сортировке значения равны 1 или –1. Если ключ связан со значением 1 в методе `sort()`, то документы будут отсортированы по возрастанию значений этого ключа. Если

ключ связан со значением –1 в методе `sort()`, то документы будут отсортированы в порядке убывания. Как и в случае с предложением `ORDER BY` запроса SQL `SELECT`, если указано несколько атрибутов сортировки, результаты сортируются на основе атрибутов слева направо. Например, следующая команда извлекает документы стран, проецирует название и столица, затем сортирует результаты в порядке убывания по континентам, а затем в пределах совпадающих значений континентам документы сортируются в порядке возрастания по населению, как показано на рисунке 38:

```
db.страна.find(
{},
{
название:1,
столица:1,
_id:0
})
.sort(
{
континент:-1,
население:1
})
```

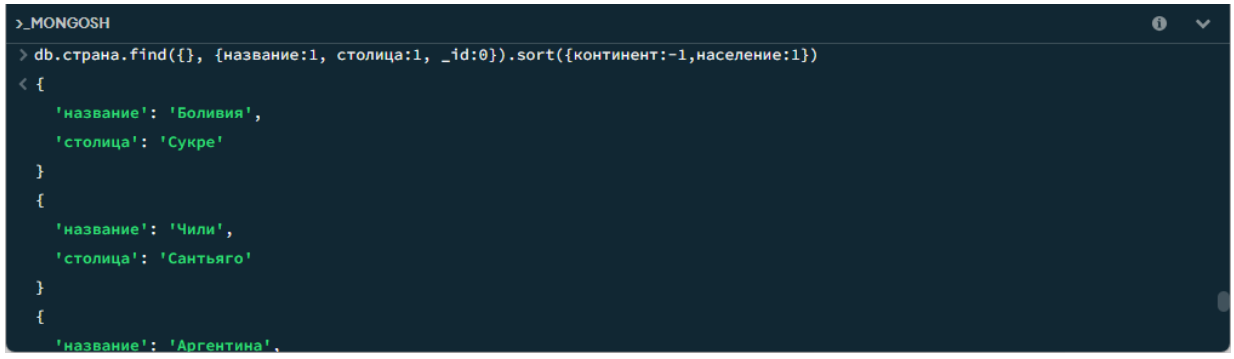

Рис. 38

Сортировка документов

Интересно, что MongoDB не заботится о том, в каком порядке методы `limit()`, `skip()` и `sort()` располагаются после метода `find()`. Таким образом, следующие три команды возвращают один и тот же результат:

```
db.страна.find(
{},
{
название:1,
столица:1,
_id:0
})
.sort(
{
континент:-1,
население:1
})
.limit(2)
.skip(4)
db.страна.find(
{},
{
название:1,
столица:1,
_id:0
})
.sort(
```

```
{
континент:-1,
население:1
})
.skip(4)
.limit(2)
db.страна.find(
{},
{
название:1,
столица:1,
_id:0
})
.skip(4)
.limit(2)
.sort(
{
континент:-1,
население:1
})
```

Запросы с использованием неравенств в MongoDB

На данный момент все рассмотренные нами запросы используют не более одного критерия в объекте запроса с простым условием равенства. Неравенства, такие как «больше», «меньше» и «не равно», также возможны, но требуют использования функций. В таблице 4 перечислены функции, используемые в MongoDB для сравнения неравенств.

Таблица 4

Функции сравнения MongoDB

Функция	Описание
$gt	Больше чем >
$gte	Больше чем или равно ≥
$lt	Меньше чем <
$lte	Меньше чем или равно ≤
$ne	Не равно ≠

Все функции неравенства работают одинаково и имеют одинаковый синтаксис. Функции используются в качестве ключа для пары «ключ: значение». Например, функция `$gt` используется для сравнения значений, превышающих указанную. Следовательно, `{$gt:20}` будет означать «больше 20». Пара «ключ: значение» функции используется в качестве значения объекта для ключа, с которым функция должна работать. Если мы хотим ограничиться документами, в которых `площадь > 200 000`, то мы бы поместили объект функции `{$gt: 200 000}` как значение в пару с ключом площадь, например `площадь: { $gt:200 000}` в параметре объекта запроса. Аналогично, если мы хотим найти документы, население которых ≤ `300 000`, критерием будет запись `население:{ $lte:300 000}`, потому что `$lte` — это функция «меньше или равная», `300 000` — целевое значение для функции. Следующая команда извлекает название и население из документов для стран с население `300000` и меньше, как показано на рисунке:

```
db.страна.find(
{
население:
{
$lte:300000
}
},
{
название:1,
```

```
население:1,
_id:0
})
.sort(
{
население:1
})
```

```
>_MONGOSH
> db.страна.find({население:{$lte:300000}}, {название:1, население:1, _id:0}).sort({население:1})
< {
    'название': 'Андорра',
    'население': 82887
  }
  {
    'название': 'Антигуа и Барбуда',
    'население': 93581
  }
  {
    'название': 'Барбадос',
    'население': 290604
  }
учебная >
```

Рис. 39

Использование неравенства

Объединение критериев с помощью `$and` и `$or`

Как и следовало ожидать, MongoDB предоставляет возможность комбинировать несколько критериев в параметре объекта запроса метода `find()`, используя логические `AND` и `OR` операторы. Объединение критериев с логическим оператором может осуществляться как неявно, так и явно. Для разделения критериев запятыми в объекте запроса используется неявный оператор `AND`. Например, следующий запрос находит страны с населением больше `100 000 000`, у которых площадь меньше `1 000 000` км^2. Чтобы улучшить читабельность, результат форматируется с помощью метода pretty(), как показано на рисунке 40.

```
db.страна.find(
{
население:
{
$gte:100000000
},
площадь:
{
$lte:1000000
}
})
.pretty()
```

```
>_MONGOSH
> db.страна.find({население:{$gte:100000000},площадь:{$lte:1000000}}).pretty()
< {
    _id: ObjectId('65ad6f1cb250ddc5b2c48bfe'),
    'название': 'Бангладеш',
    'столица': 'Дакка',
    'площадь': 148460,
    'население': 171674893,
    'континент': 'Азия'
  }
учебная >
```

Рис. 40

Неявный оператор AND

Предыдущая команда содержала два критерия: `население:{$gte:100 000 000}` и `площадь:{$lte:1 000 000}`. Поскольку эти критерии разделены запятой, MongoDB объединяет их с неявным логическим `AND`.

Явная логика AND требует использования функции `$and`. На практике явное использование функции `$and` имеет тенденцию быть предпочтительным. При использовании неявного все критерии в списке всегда оцениваются как `true` или `false`. При явном использовании `$and`, как только какой-либо критерий для документа оказывается ложным, остальные критерии для этого документа пропускаются. Это делает работу явной `$and` быстрой и эффективной. Таким образом, он более ценен, чем неявный, для большинства операций. Функция `$and` появляется как ключ в паре «ключ: значение», которая принимает массив объектов в качестве целевого значения. Например, если *x* и *y* являются критериями, то синтаксис `$and` будет `$and: [{x}, {y}]`. Обратите внимание, что каждый критерий заключен в `{}`, чтобы указать, что он является объектом. Следующая команда извлекает все документы, в которых есть страна на континенте `Азия`, с населением меньше `100 000 000` и с площадью `100 000` км^2, как показано на рисунке 41.

```
db.страна.find(
{
$and:
[
{
континент:"Азия"
},
{
население:
{
$lte:100000000
}
},
{
площадь:
{
$gte:100000
}
}
]
})
.pretty()
```

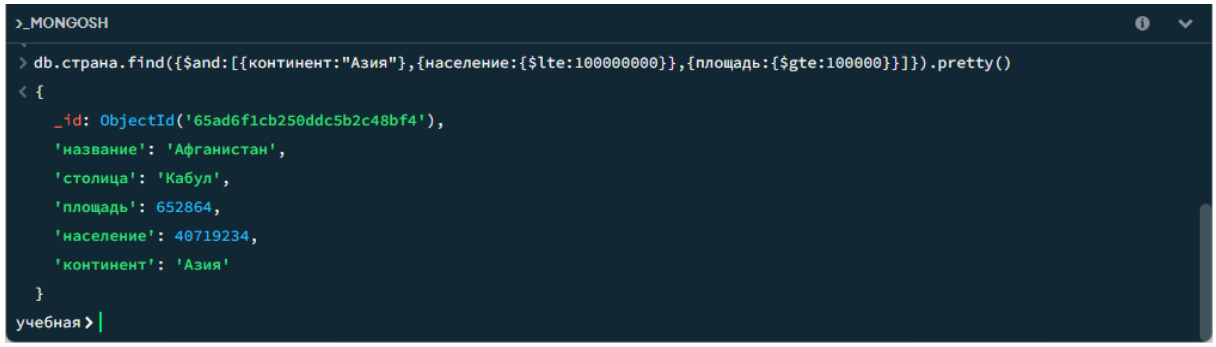

Рис. 41

Явный оператор `$and`

Опять же, обратите внимание, что каждый критерий заключен в свой собственный набор скобок, так что он является самостоятельным объектом. Объекты критериев разделяются запятой.

Все логические `OR` операции являются явными. Функция `$or` имеет тот же синтаксис, что и функция `$and`. Следующая команда извлекает страны `Азии` или `Африки` с населением

больше `10 000 000`. Будут отображаться только `название`, `население` и `площадь`, а результаты будут отсортированы `по населению`, как показано на рисунке 42:

```
db.страна.find(
{
$and:
[
{
$or:
[
{
континент:"Азия"
},
{
континент:"Африка"
}
]
},
{
население:
{
$gt:10000000
}
}
]
},
{
название:1,
население:1,
площадь:1,
_id:0
})
.sort(
{
население:1
})
.pretty()
```

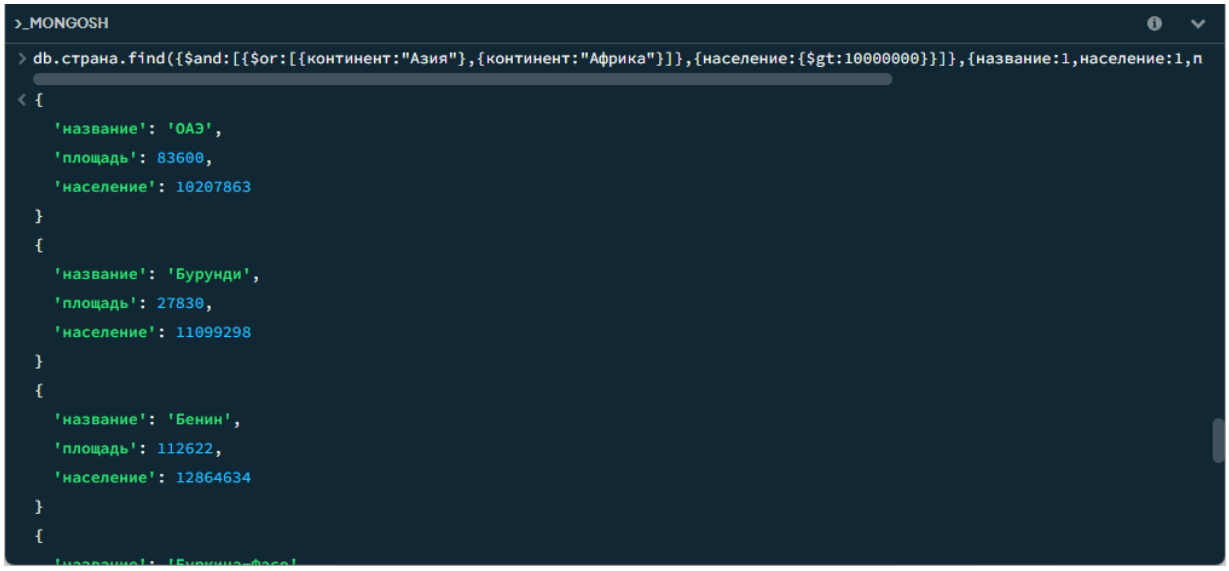

Рис. 42

Составной MQL-запрос

Поскольку запросы MQL становятся более сложными, они могут показаться запутанными из-за множества знаков препинания. Если разобрать предыдущий запрос, символы станут немного понятнее.

1. Запрос основан на методе `db.страна.find()`.

2. Внутри метода `find()` есть два объекта: объект запроса и объект проекции.

3. Объект запроса: `{$and: [{$or: [{континент: "Азия"}, {континент: "Африка"}]}, {население: {$gt: 10000000}}]}`.

4. Объект запроса содержит одну пару «ключ: значение». `$and` — это ключ, а `[{$or: [{континент:"Азия"}, {континент: "Африка"}]}, {население: {$gt:10000000}}]` — это значение.

5. Значением ключа `$and` является массив, состоящий из двух объектов. Первый объект — `{$or: [{континент:"Азия"}, {континент: "Африка"}]`, а второй — `{население: {$gt:10000000}}`.

6. Значением ключа `$or` является массив, состоящий из двух объектов. Первый объект — `{континент: "Азия"}`, а второй — `{континент: "Африка"}`.

7. Второй объект в массиве значений оператора `$and` представляет собой пару «ключ: значение», где ключом является `население`, а значением — объект. Значение объекта — `{$gt:10000000}`, которое вызывает функцию «больше чем» с `10 000 000` в качестве целевого значения функции.

8. Объект проекции содержит список пар «ключ: значение».

9. Объект проекции: `{название:1, население:1, площадь:1, _id:0}`.

10. Ключи объекта, связанные со значением 1, включаются в результат. Ключ объекта, связанный со значением 0, исключается из результата.

11. Метод `sort()` связан с методом `find()`, чтобы указать порядок сортировки выходных данных.

12. Метод `sort()` содержит объект, состоящий из простых пар «ключ: значение».

13. Ключи в объекте сортировки, связанные со значением 1, сортируются в порядке возрастания. Если какой-либо ключ был связан со значением –1, он был бы отсортирован в порядке убывания.

3. *Задание.*

1. Вывести название и столицу пяти наибольших стран по площади.

2. Вывести список африканских стран, население которых не превышает 1 млн чел.

3. Вывести список стран, население которых больше 5 млн чел., а площадь меньше 100 тыс. км^2, и они расположены не в Европе.

4. Вывести список стран Северной и Южной Америки, население которых больше 20 млн чел., или стран Африки, у которых население больше 30 млн чел.

5. Вывести список стран, население которых составляет от 10 до 100 млн чел., а площадь не больше 500 тыс. км^2.

6. Вывести список стран Африки, Северной и Южной Америки.