**Лабораторные работы №6-7**

**Часть 1**

ЗАПРОСЫ К БАЗЕ ДАННЫХ MONGODB.

ВЫБОРКА ДАННЫХ. ВЛОЖЕННЫЕ ОБЪЕКТЫ. ИСПОЛЬЗОВАНИЕ КУРСОРОВ. АГРЕГИРОВАННЫЕ ЗАПРОСЫ. ИЗМЕНЕНИЕ ДАННЫХ

1. ЗАПРОС К ВЛОЖЕННЫМ ОБЪЕКТАМ

Рассмотренные ранее запросы применялись к простым документам. Документы могут иметь сложную структуру и содержать вложенные документы.

use towns

db.towns.insert({"name": "Punxsutawney ", "populatiuon": 6200,"last\_sensus": ISODate("2022-01-31"), "famous\_for": [""], mayor: { "name": "Jim Wehrle"}})

db.towns.insert({"name": "New York ", "populatiuon": 22200000,"last\_sensus": ISODate("2022-01-31"), "famous\_for": ["status of liberty", "food"], mayor: { "name": "Michael Bloomberg", "party": "I"}})

db.towns.insert({"name": "Portland", "populatiuon": 528000,"last\_sensus": ISODate("2022-07-20"), "famous\_for": ["beer", "food"], mayor: { "name": "Sam Adams", "party": "D"}})

Пусть коллекцию users добавлен следующий документ:

> db.users.insert({"name": "Alex", "age": 28, company: {"name":"microsoft", "country":"USA"}})

*Примечание. Содержание коллекции users:*

> db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})

> db.users.insert({"name": "Bill", "age": 32, languages: ["english", "french"]})

> db.users.insert({"name": "Tom", "age": 32, languages: ["english", "german"]})

Здесь определяется вложенный объект с ключом company. Чтобы найти все документы, у которых в ключе company вложенное свойство name=microsoft, нужно использовать оператор точку:

> db.users.find({"company.name": "microsoft"})

**Практическое задание 1:**

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

{name: "Punxsutawney ",

populatiuon: 6200,

last\_sensus: ISODate("2022-01-31"),

famous\_for: [""],

mayor: {

name: "Jim Wehrle"

}}

{name: "New York",

populatiuon: 22200000,

last\_sensus: ISODate("2022-07-31"),

famous\_for: ["status of liberty", "food"],

mayor: {

name: "Michael Bloomberg",

party: "I"}}

{name: "Portland",

populatiuon: 528000,

last\_sensus: ISODate("2022-07-20"),

famous\_for: ["beer", "food"],

mayor: {

name: "Sam Adams",

party: "D"}}

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

db.towns.find({"mayor.party":"I"},{name:1,mayor:1,\_id:0})

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

db.towns.find({"mayor.party":{$exists:false}},{name:1,mayor:1,\_id:0})

1. ИСПОЛЬЗОВАНИЕ JAVASCRIPT

MongoDB предоставляет возможность создавать запросы, используя язык JavaScript. Например, создать запрос, возвращающий те документы, в которых name=Tom. Для этого сначала объявляется функция:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | > fn = function() { return this.name=="Tom"; }  > db.users.find(fn)  Этот запрос эквивалентен следующему:   |  |  | | --- | --- | |  | > db.users.find("this.name=='Tom'") |   Собственно, только запросами область применения JavaScript в консоли mongo не ограничена. Например, можно создать какую-нибудь функцию и применять ее:   |  |  | | --- | --- | |  | > function sqrt(n) {return n\*n;}  > sqrt(5)  25 | |

1. КУРСОРЫ

Результат выборки, получаемой с помощью функции find, называется курсором:

|  |  |
| --- | --- |
|  | > var cursor = db.users.find(); null; |

Чтобы получить курсор и сразу же не выводить все содержащиеся в нем данные, после метода find() через точку с запятой нужно задать выражение null;.

Курсоры инкапсулируют в себе наборы, получаемых из базы объектов. Используя синтаксис языка javascript и методы курсоров, можно вывести полученные документы на экран и обработать их. Например:

|  |  |
| --- | --- |
|  | > var cursor = db.users.find();null;  > while(cursor.hasNext()){  ... obj = cursor.next();  ... print(obj["name"]);  ... } |

Курсор обладает методом hasNext, который показывает при переборе, имеется ли еще в наборе документ. Метод next извлекает текущий документ и перемещает курсор к следующему документу в наборе. В итоге в переменной obj оказывается документ, к полям которого можно получить доступ.

Для перебора документов в курсоре в качестве альтернативы можно использовать конструкцию итератора javascript - forEach:

> var cursor = db.users.find()

> cursor.forEach(function(obj){

... print(obj.name);

... })

Чтобы ограничить размер выборки, используется метод limit, принимающий количество документов:

> var cursor = db.users.find();null;

> cursor.limit(3);null;

> cursor.forEach(function(obj){

... print(obj.name);

... })

Используя метод sort(), можно отсортировать документы в курсоре:

> var cursor = db.users.find();null;

null

> cursor.sort({name:1});null;

null

> cursor.forEach(function(obj){

... print(obj.name);

... })

Выражение cursor.sort({name:1}) сортирует документы в курсоре по полю name по возрастанию. Если нужно отсортировать по убыванию, то вместо 1 использовать -1: cursor.sort({name:-1})

Метод skip() позволяет пропустить при выборке определенное количество документов:

> var cursor = db.users.find();null;

> cursor.skip(2);null;

> cursor.forEach(function(obj){

... print(obj.name);

... })

В данном случае пропускается два документа.

Можно объединять все эти методы в цепочки:

> var cursor = db.users.find();null;

> cursor.sort({name:1}).limit(3).skip(2);null;

> cursor.forEach(function(obj){

... print(obj.name);

... })

**Практическое задание 2:**

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

*-*

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

*-*

1. *Вывести результат, используя forEach.*

*var cursor = db.unicorns.find({gender:"m"});null;cursor.sort({name:1}).limit(2);null; cursor.forEach(function(obj){ print(obj.name); })*

1. *Содержание коллекции 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', 44), 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: 'Dunx', loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165})

1. АГРЕГИРОВАННЫЕ ЗАПРОСЫ

С помощью функции **count()** можно получить число элементов в коллекции:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | > db.users.count()  Эта функция является самым простым агрегатом.  Можно группировать параметры поиска и функцию count, чтобы подсчитать количество документов, удовлетворяющих критерию, например, у которых name=Alex: | |  | > db.users.find({name: "Alex"}).count()  Можно создавать цепочки функций, чтобы конкретизировать условия подсчета:   |  |  | | --- | --- | |  | > db.users.find({name: "Tom"}).skip(2).count(true) |   Здесь нужно отметить, что по умолчанию функция count не используется с функциями limit и skip. Чтобы их использовать, как в примере выше, в функцию count надо передать булевое значение true. | |  |

**Практическое задание 3:**

*Вывести количество самок единорогов весом от 500 до 600 кг.*

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

Функция **distinct()** позволяет найти уникальные различающиеся значения для одного или нескольких полей документа.

Например, в нескольких документах определено name: "Tom". Нужно найти только уникальные различающиеся значения для одного из полей документа. Для этого можно воспользоваться функцией distinct:

|  |  |
| --- | --- |
|  | > db.users.distinct("name")  ["Tom", "Bill", "Alex"] |

**Практическое задание 4:**

*Вывести список предпочтений.*

db.unicorns.distinct("loves")

Использование метода aggregate аналогично применению выражения GROUP BY в SQL. Метод group принимает три параметра:

* $group: агрегатор, который вернет новый документ
* \_id: указывает на ключ, по которому надо проводить группировку ($+назване поля)
* $sum: оператор для вычисления.

Например:

> db.users.aggregate({"$group":{\_id:"$name",count:{$sum:1}}})

Пояснение. Параметр \_id указывает, что группировка будет проводиться по ключу name: \_id:"$name".

Значение параметра $sum инициализирует начальное значение поля count. Это поле будет представлять количество элементов для группы. На рисунке 1 приведен пример подсчет количества документов с разными именами name.

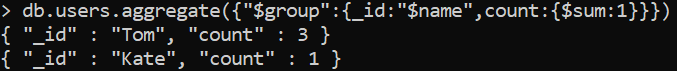


Рисунок 1 – Пример использования метода aggregate

(<https://docs.mongodb.com/manual/aggregation/>).

**Практическое задание 5:**

*Посчитать количество особей единорогов обоих полов.*

db.unicorns.aggregate({"$group":{\_id:"$gender",count:{$sum:1}}})

5 РЕДАКТИРОВАНИЕ ДАННЫХ

СУБД MongoDB предоставляет несколько возможностей для обновления данных.

Метод **save** является наиболее простым. В качестве фактического параметра метод принимает документ. В этот документ в качестве поля можно передать параметр \_id. Если метод находит документ с таким значением \_id, то документ обновляется. Если же с подобным \_id нет документов, то документ вставляется.

Если параметр \_id не указан, то документ вставляется, а параметр \_id генерируется автоматически как при обычном добавлении через функцию insert:

|  |  |
| --- | --- |
|  | > db.users.save({name: "Eugene", age : 29,  languages: ["english", "german", "spanish"]}) |

В качестве результата функция возвращает объект WriteResult. Например, при успешном сохранении получится результат:

|  |  |
| --- | --- |
|  | WriteResult({"nInserted" : 1 })  **Практическое задание 6:**   1. *Выполнить команду:*   > db.unicorns.save({name: 'Barny', loves: ['grape'],  weight: 340, gender: 'm'})   1. *Проверить содержимое коллекции unicorns.* |

Функция **update** позволяет более **детально настроить обновление**.

Функция принимает три параметра:

* query: принимает запрос на выборку документа, который нужно обновить;
* objNew: представляет документ с новой информацией, который заместит старый при обновлении;
* options: определяет дополнительные параметры при обновлении документов и может принимать два аргумента: upsert и multi.

Если параметр upsert имеет значение true, что mongodb будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если же он имеет значение false, то mongodb не будет создавать новый документ, если запрос на выборку не найдет ни одного документа.

Параметр multi указывает, должен ли обновляться первый элемент в выборке (используется по умолчанию, если данный параметр не указан) или же должны обновляться все документы в выборке.

Например:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {name: "Tom", age : 25, married : false},  {upsert: true}) |

Теперь документ, найденный запросом {name : "Tom"}, будет перезаписан документом {"name": "Tom", "age" :"25", "married" : false}.

Функция update() также возвращает объект WriteResult. Например:

|  |  |
| --- | --- |
|  | WriteResult({"nMatched" : 1, "nUpserted": 0, "nModified": 1}) |

В данном случае результат говорит о том, что найден один документ, удовлетворяющий условию, и один документ был обновлен.

**Практическое задание 7:**

1. *Для самки единорога* Ayna *внести изменения в БД: теперь ее вес 800, она убила 51 вампира.*
2. db.unicorns.update({name: "Ayna"},{$set: {weight: 800, vampires: 51}})
3. *Проверить содержимое коллекции unicorns.*

**Для обновления значения только одного из ключей** используется оператор **$set.**

Если документ не содержит обновляемое поле, то оно создается.

|  |  |
| --- | --- |
|  | > db.users.update({name : "Eugene", age: 29}, {$set: {age : 30}}) |

В данном случае обновлялся только один документ, первый в выборке. Указав значение multi:true, можно обновить все документы выборки:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"},  {$set: {name: "Tom", age : 25, married : false}}, {multi:true})  **Практическое задание 8:**   1. *Для самца единорога* Raleigh *внести изменения в БД: теперь он любит рэдбул.*   *Для пары lowers:*  db.unicorns.update({name: "Raleigh"},{$set: {loves: ["apple", "lemon"]}})  *Для изменения одной:*  db.unicorns.update({name: "Raleigh"},{$set: {loves: "redbul"}})   1. *Проверить содержимое коллекции unicorns.* |
|  |  |

**Для простого увеличения значения числового поля на определенное количество единиц** применяется оператор **$inc**. Если документ не содержит обновляемое поле, то оно создается. Данный оператор применим только к числовым значениям.

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$inc: {salary:100}}) |

**Практическое задание 9:**

1. *Всем самцам единорогов увеличить количество убитых вапмиров на 5.*
2. db.unicorns.update({gender:'m'},{$inc:{ vampires:5}},{multi:true} )

*updateMany*

1. *Проверить содержимое коллекции unicorns.*

**Для удаления отдельного ключа** используется оператор **$unset**:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$unset: {salary: 1}}) |

Если подобного ключа в документе не существует, то оператор не оказывает никакого влияния.

Можно удалять сразу несколько ключей:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}}) |

**Практическое задание 10:**

1. *Изменить информацию о городе Портланд: мэр этого города теперь беспартийный.*
2. db.towns.update({name:"Portland"},{$unset:{"mayor.party":1}})

*(если надо обратно вернуть, то вот запрос:* db.towns.update({name:"Portland"},{$set:{"mayor.party":"D"}})

1. *Проверить содержимое коллекции towns.*

Оператор **$push** позволяет **добавить еще одно значение к уже существующему**. Например, если ключ в качестве значения хранит массив:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$push: {languages: "russian"}}) |

Если ключ, для которого нужно добавить значение, не представляет массив, то получится ошибка Cannot apply $push/$pushAll modifier to non-array.

**Практическое задание 11:**

1. *Изменить информацию о самце единорога Pilot: теперь он любит и шоколад.*
2. db.unicorns.update({name:"Pilot"},{$push:{loves:"chocolate"}})
3. *Проверить содержимое коллекции unicorns.*

Оператор **$addToSet** подобно оператору **$push** **добавляет объекты в массив**. Отличие состоит в том, что $addToSet добавляет данные, если их еще нет в массиве:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$addToSet: {languages: "russian"}}) |

Используя оператор **$each**, можно **добавить сразу несколько значений**:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"},  {$addToSet: {languages: {$each: ["russian", "spanish", "italian"]}}}) |

**Практическое задание 12:**

1. *Изменить информацию о самке единорога* Aurora: *теперь она любит еще и сахар, и лимоны.*
2. db.unicorns.update({name:"Aurora"},{$addToSet:{loves:{$each:["lemon","sugar"]}}})
3. *Проверить содержимое коллекции unicorns.*

Оператор **$pop** позволяет **удалять элемент из массива**:

> db.users.update({name : "Tom"}, {$pop: {languages: 1}})

Указывая для ключа languages значение 1, можно удалить первый элемент с конца. Чтобы удалить первый элемент с начала массива, нужно передать отрицательное значение:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$pop: {languages: -1}}) |

Оператор **$pull** **удаляет каждое вхождение элемента в массив**. Например, через оператор $push можно добавить одно и то же значение в массив несколько раз. Далее с помощью $pull удалить его:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"}, {$pull: {languages: "english"}}) |

Если нужно удалить не одно значение, а сразу несколько, тогда можно применить оператор $pullAll:

|  |  |
| --- | --- |
|  | > db.users.update({name : "Tom"},  {$pullAll: {languages: ["english", "german", "french"]}}) |

1. УДАЛЕНИЕ ДАННЫХ ИЗ КОЛЛЕКЦИИ

Для удаления документов в MongoDB предусмотрен метод remove:

> db.users.remove({name : "Tom"})

Метод remove() возвращает объект WriteResult. При успешном удалении одного документа результат будет следующим:

WriteResult({"nRemoved" : 1})

В итоге все найденные документы с name=Tom будут удалены. Причем, как и в случае с find, можно задавать условия выборки для удаления различными способами (в виде регулярных выражений, в виде условных конструкций и т.д.):

> db.users.remove({age: {$lt : 30}})

Метод remove также может принимать второй необязательный параметр булевого типа, который указывает, надо удалять один элемент или все элементы, соответствующие условию. Если этот параметр равен true, то удаляется только один элемент. По умолчанию он равен false:

> db.users.remove({name : "Tom"}, true)

Чтобы удалить все документы из коллекции, нужно оставить пустым параметр запроса:

> db.users.remove({})

*Примечание. Содержание коллекции users:*

> db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})

> db.users.insert({"name": "Bill", "age": 32, languages: ["english", "french"]})

> db.users.insert({"name": "Tom", "age": 32, languages: ["english", "german"]})

**Практическое задание 13:**

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

{name: "Punxsutawney ",

popujatiuon: 6200,

last\_sensus: ISODate("2022-01-31"),

famous\_for: ["phil the groundhog"],

mayor: {

name: "Jim Wehrle"

}}

{name: "New York",

popujatiuon: 22200000,

last\_sensus: ISODate("2022-07-31"),

famous\_for: ["status of liberty", "food"],

mayor: {

name: "Michael Bloomberg",

party: "I"}}

{name: "Portland",

popujatiuon: 528000,

last\_sensus: ISODate("2022-07-20"),

famous\_for: ["beer", "food"],

mayor: {

name: "Sam Adams",

party: "D"}}

1. *Удалите документы с беспартийными мэрами.*

db.towns.remove({"mayor.party":{$exists:false}})

1. *Проверьте содержание коллекции.*
2. *Очистите коллекцию.*

db.towns.remove({})

1. *Просмотрите список доступных коллекций.*

db.getCollectionNames()

**Контрольные вопросы:**

1. Как используется оператор точка?
2. Как можно использовать курсор?
3. Какие возможности агрегирования данных существуют в MongoDB?
4. Какая из функций save или update более детально позволит настроить редактирование документов коллекции?
5. Как происходит удаление документов из коллекции по умолчанию?

**Список источников:**

1. MоngoDB CRUD Operations [Электронный ресурс] // mongoDB. Documentation: официальный сайт MоngoDB. URL: <https://docs.mongodb.com/manual/crud/> (дата обращения: 02.11.2022).
2. MоngoDB – Краткое руководство [Электронный ресурс] // CoderLessons.com. Уроки по программированию, DevOps и другим IT-технологиям: сайт, 2021. URL: <https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo> (дата обращения: 02.11.2022).
3. Кайл Б. MongoDB в действии [Электронный ресурс] // Доступ в ЭБС «Лань». Режим доступа: <https://e.lanbook.com/book/4156> (дата обращения: 05.11.2022).
4. Онлайн-руководство по MongoDB [Электронный ресурс] // METANIT.COM. Сайт о программировании. URL: <https://metanit.com/nosql/mongodb/> (дата обращения: 05.11.2022).
5. Эрик Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. [Электронный ресурс]/Р. Эрик, Р.У. Джим. Электрон. дан. М.: ДМК Пресс, 2013. 384с. Доступ из ЭБС «Лань». URL: [http://e.lanbook.com/book/58690](http://e.lanbook.com/book/58690%20)

**Часть 2**

ССЫЛКИ И РАБОТА С ИНДЕКСАМИ В БАЗЕ ДАННЫХ MONGODB

1. ССЫЛКИ В БД

Ручная установка ссылок

Ручная установка ссылок сводится к присвоению значения поля \_id одного документа полю другого документа. Пусть есть коллекции companies и persons, представляющие компании и работников, работающих в этих компаниях. В коллекции companies есть документ, представляющий компанию:

> db.companies.insert({"\_id" : "microsoft", year: 1974})

Пусть в коллекции persons есть документ, представляющий работника. В этом документе будет поле company, представляющее компанию, где работает работник. И очень важно, что в качестве значения для этого поля устанавливается не объект company, а значение ключа \_id добавленного выше документа:

> db.users.insert({name: "Tom", age: 28, company: "microsoft"})

Вывести документ из коллекции users:

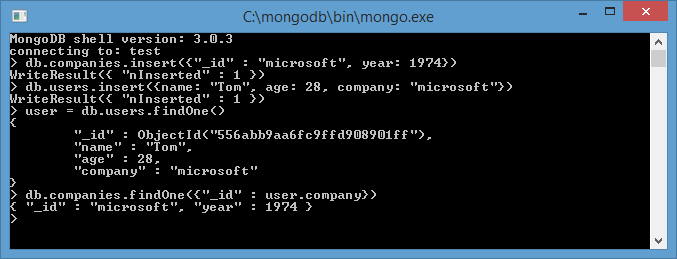
> user = db.users.findOne()

В данном случае имеется в виду, что выше добавленный элемент будет единственным в коллекции.

После этого консоль выводит полученный документ. Далее найти ссылку на его компанию в коллекции companies:

> db.companies.findOne({\_id: user.company})

Если документ с таким идентификатором обнаружен, он отображается на консоли:



Автоматическое связывание

1 способ. Используя функциональность DBRef, можно установить автоматическое связывание между документами.

Пример применение данной функциональности: добавить новый документ в коллекцию companies:

> apple=({"name" : "apple", "year": 1976})

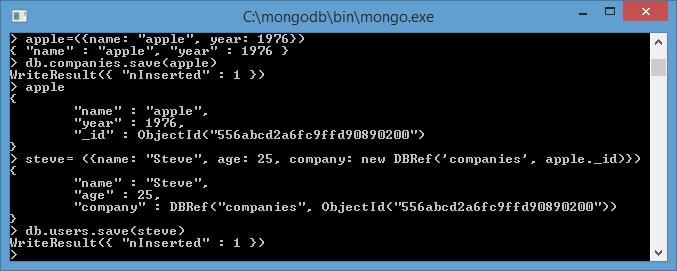
> db.companies.save(apple)

В данном случае сохранение идет с помощью метода save, не insert. Метод save при добавлении нового документа генерирует \_id. После сохранения можно вывести документ на консоль: > apple

Далее создать новый документ для коллекции person, у которого ключ company свяжем с только что добавленным документом apple:

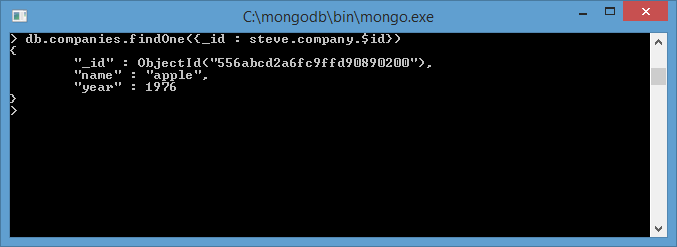
> steve = ({"name": "Steve", "age": 25, company: new DBRef('companies', apple.\_id)})

> db.users.save(steve)



Протестировать:

> db.companies.findOne({\_id: steve.company.$id})



Посмотрев на примере, можно разобрать организацию ссылок между документами. Для связывания с документом apple использовалось следующее выражение company: new DBRef('companies', apple.\_id). Формальный синтаксис DBRef следующий:

**{ "$ref" : название\_коллекции, "$id": значение [, "$db" : название\_бд ]}**

При тестировании в качестве запроса на выборку указывается выражение \_id: steve.company.$id. Так как person.company представляет теперь объект new DBRef('companies', apple.\_id), то нужно конкретизировать параметр steve.company.$id.

2 способ. Пусть пользователя нужно связать со страной.

Используется коллекция countries, хранящая в качестве идентификатора аббревиатуру и наименование, например:

> db.countries.insert({\_id:"us", name:"US"})

Необходимо добавить в коллекцию пользователей ссылку на страну:

>db.users.update({\_id:ObjectId("573d7468bfe2c1a9875562e6")},{$set:

{country:{$ref:"countries", $id: "us"}}})

Теперь можно извлечь из коллекции данные о Томе:

>var tom=db.users.findOne({"\_id":ObjectId("573d7468bfe2c1a9875562e6")})

Можно также получить сведения о стране, в которой находится пользователь, опросив коллекцию стран, передав сохраненный ранее идентификатор $id:

> db.countries.findOne({\_id:tom.country.$id})

Можно непосредственно запросить у документа о пользователе имя коллекции, хранящейся в поле ссылки:

db[tom.country.$ref].findOne({\_id:tom.country.$id})

Последние два запроса эквивалентны, но второй в большей степени управляется данными.

**Практическое задание 1:**

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

db.place.insert({\_id:"tw", name:"town", description:"Cold and dirty"})

db.place.insert({\_id:"se",name:"sea",description:"Blue and warm"})

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

db.unicorns.update({\_id:ObjectId("636e3bdc3c6dae7bb1e6b5ee")},{$set:

{place:{$ref:"place", $id: "sa"}}})

db.unicorns.update({\_id:ObjectId("6371ff1702a5e0966061956a")},{$set:

{place:{$ref:"place", $id: "sa"}}})

db.unicorns.update({\_id: ObjectId(

"6371fab202a5e0966061955f")},{$set:{place:{$ref:"place",$id: "tw"}}})

db.unicorns.update({\_id: ObjectId("6371fafc02a5e09660619560")},{$set:{place:{$ref:"place",$id:"fr"}}})

1. *Проверьте содержание коллекции единорогов.*
2. *Содержание коллекции 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', 44), 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: 'Dunx', loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165}

2 НАСТРОЙКА ИНДЕКСОВ

Когда коллекции содержат миллионы документов, а нужно сделать выборку по определенному полю, то поиск нужных данных может занять некоторое время, которое может оказаться критичным для задачи. В этом случае могут помочь индексы.

Индексы позволяют упорядочить данные по определенному полю, что впоследствии ускорит поиск. Например, если в приложении или задаче, как правило, выполняется поиск по полю name, то можном индексировать коллекцию по этому полю:

> db.users.ensureIndex({"name" : 1})

Значение 1 сообщает, что индекс следует выполнить в порядке возрастания.

*Примечание. Содержание коллекции users:*

> db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})

> db.users.insert({"name": "Bill", "age": 32, languages: ["english", "french"]})

> db.users.insert({"name": "Tom", "age": 32, languages: ["english", "german"]})

Таким образом с помощью метода ensureIndex устанавливается индекс по полю name. MongoDB позволяет установить до 64 индексов на одну коллекцию.

Создать индекс можно также с помощью метода createIndex().

Если нужно просто определить индекс для коллекции, например, db.users.ensureIndex({"name" : 1}), то можно добавлять в коллекцию документы с одинаковым значением ключа name. Однако, если потребуется, чтобы в коллекцию можно было добавлять документ с одним и тем же значением ключа только один раз, нужно установить флаг unique:

|  |  |
| --- | --- |
|  | > db.users.ensureIndex({"name" : 1}, {"unique" : true})  Теперь, если попытаться добавить в коллекцию два документа с одним и тем же значением name, то мы получится ошибка.  **Практическое задание 2:**   1. *Проверьте, можно ли задать для коллекции unicorns индекс для ключа name с флагом* unique*.*   db.unicorns.ensureIndex({"name" : 1}, {"unique" : true})   1. *Содержание коллекции единорогов unicorns:*   db.unicorns.insert({name: 'Horny', dob: new Date(1992,2,13,7,47), loves: ['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});  db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0), loves: ['carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});  db.unicorns.insert({name: 'Unicrom', dob: new Date(1973, 1, 9, 22, 10), loves: ['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});  db.unicorns.insert({name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44), loves: ['apple'], weight: 575, gender: 'm', vampires: 99});  db.unicorns.insert({name: 'Solnara', dob: new Date(1985, 6, 4, 2, 1), loves:['apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});  db.unicorns.insert({name:'Ayna', dob: new Date(1998, 2, 7, 8, 30), loves: ['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});  db.unicorns.insert({name:'Kenny', dob: new Date(1997, 6, 1, 10, 42), loves: ['grape', 'lemon'], weight: 690, gender: 'm', vampires: 39});  db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57), loves: ['apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});  db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53), loves: ['apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});  db.unicorns.insert({name: 'Pilot', dob: new Date(1997, 2, 1, 5, 3), loves: ['apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});  db.unicorns.insert ({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15), loves: ['grape', 'carrot'], weight: 540, gender: 'f'});  db.unicorns.insert {name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18), loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165  В тоже время здесь особенности. Так, документ может не иметь ключа name. В этом случае для добавляемого документа автоматически создается ключ name со значением null. Поэтому при добавлении второго документа, в котором не определен ключ name, будет выброшено исключение, так как ключ name со значением null уже присутствует в коллекции.  Также можно задать уникальный индекс сразу для двух полей:  > db.users.ensureIndex({"name" : 1, "age" : 1}, {"unique" : true})  Однако в этом случае все добавляемые документы должны иметь уникальные значения для обоих полей.  Ограничения: значение поля, по которому идет индексация, не должно быть больше 1024 байт. |
|  |  |

1. УПРАВЛЕНИЕ ИНДЕКСАМИ

Все индексы базы данных хранятся в системной коллекции system.indexes. Обратившись к ней, можно получить все индексы и связанную с ними информацию:

> db.system.indexes.find()

Данная команда возвращает информацию об индексах:

> db.numbers.getIndexes()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "learn.numbers"

}

]

Здесь для коллекции users определен 1 индекс: id. Поле key используется для поиска максимального и минимального значений, для различных операций, где нужно применять данный индекс. Поле name применяется в качестве идентификатора для операций администрирования, например, для удаления индекса:

> db.users.dropIndex("value")

Здесь value равно значению поля "name" для удаляемого индекса.

**Практическое задание 3:**

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

db.unicorns.getIndexes()

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

*db.unicorns.dropIndex("name\_1")*

1. *Попытайтесь удалить индекс для идентификатора.*

*-*

1. ПЛАН ЗАПРОСА

С помощью метода explain()  можно получить информацию о выполнении запроса. Метод explain() возвращает JSON-структуру с планом выполнения запроса.

Для детализации плана запроса нужно указать параметр "executionStats" для метода explain():

db.users.explain("executionStats").find({}})

**Практическое задание 4:**

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

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

var cursor = db.numbers.find();null;

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

1. *Выберите последних четыре документа.*

db.numbers.find().limit(4).sort({value: -1})

1. *Проанализируйте план выполнения запроса 2. Сколько потребовалось времени на выполнение запроса? (по значению параметра executionTimeMillis)*
2. *Создайте индекс для ключа value.*
3. *Получите информацию о всех индексах коллекции numbres.*
4. *Выполните запрос 2.*
5. *Проанализируйте план выполнения запроса с установленным индексом. Сколько потребовалось времени на выполнение запроса?*
6. *Сравните время выполнения запросов с индексом и без. Дайте ответ на вопрос: какой запрос более эффективен?*

**Контрольные вопросы:**

1. Назовите способы связывания коллекций в MongoDB.
2. Сколько индексов можно установить на одну коллекцию в БД MongoDB.
3. Как получить информацию о всех индексах базы данных MongoDB?

**Список источников:**

1. MоngoDB CRUD Operations [Электронный ресурс] // mongoDB. Documentation: официальный сайт MоngoDB. URL: [https://docs.mongodb.com/manual/](https://docs.mongodb.com/manual/crud/) (дата обращения: 02.11.2022).
2. MоngoDB – Краткое руководство [Электронный ресурс] // CoderLessons.com. Уроки по программированию, DevOps и другим IT-технологиям: сайт, 2019. URL: <https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo> (дата обращения: 02.11.2022).
3. Кайл Б. MongoDB в действии [Электронный ресурс] // Доступ в ЭБС «Лань». Режим доступа: <https://e.lanbook.com/book/4156> (дата обращения: 05.11.2022).
4. Онлайн-руководство по MongoDB [Электронный ресурс] // METANIT.COM. Сайт о программировании. URL: <https://metanit.com/nosql/mongodb/> (дата обращения: 05.11.2022).
5. Эрик Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. [Электронный ресурс]/Р. Эрик, Р.У. Джим. Электрон. дан. М.: ДМК Пресс, 2013. 384с. Доступ из ЭБС «Лань». URL: [http://e.lanbook.com/book/58690](http://e.lanbook.com/book/58690%20) (дата обращения: 05.11.2022).

**Структура отчета:**

1. Наименование работы.
2. Цель работы.
3. Практическое задание.
4. Выполнение.

*Указание: привести результаты выполнения практических заданий (номер задания, формулировка, команда, лог (скриншот) результата, вывод (при необходимости)).*

1. Выводы.