**Операторы выборки**

**Условные операторы**

Условные операторы задают условие, которому должно соответствовать значение поля документа:

* **$eq** (равно)
* **$ne** (не равно)
* **$gt** (больше чем)
* **$lt** (меньше чем)
* **$gte** (больше или равно)
* **$lte** (меньше или равно)
* **$in** определяет массив значений, одно из которых должно иметь поле документа
* **$nin** определяет массив значений, которые не должно иметь поле документа

Например, найдем все документы, у которых значение ключа age меньше 30:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$lt : 30}}) |

Аналогично будет использование других операторов сравнения. Например, тот же ключ, только больше 30:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$gt : 30}}) |

Обратите внимание, что сравнение здесь проводится над целочисленными типами, а не строками. Если ключ age представляет строковые значения, то соответственно надо проводить сравнение над строками: db.users.find ({age: {$gt : "30"}}), однако результат будет тем же.

Но представим ситуацию, когда нам надо найти все объкты со значением поля age больше 30, но меньше 50. В этом случае мы можем комбинировать два оператора:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$gt : 30, $lt: 50}}) |

Найдем пользователей, возраст которых равен 22:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$eq : 22}}) |

По сути это аналогия следующего запроса:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: 22}) |

Обратная операция - найдем пользователей, возраст которых НЕ равен 22:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$ne : 22}}) |

Оператор $in определяет массив возможных выражений и ищет те ключи, значение которых имеется в массиве:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$in : [22, 32]}}) |

Противоположным образом действует оператор $nin - он определяет массив возможных выражений и ищет те ключи, значение которых отсутствует в этом массиве:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$nin : [22, 32]}}) |

**Логические операторы**

Логические операторы выполняются над условиями выборки:

* **$or**: соединяет два условия, и документ должен соответствовать одному из этих условий
* **$and**: соединяет два условия, и документ должен соответствовать обоим условиям
* **$not**: документ должен НЕ соответствовать условию
* **$nor**: соединяет два условия, и документ должен НЕ соответстовать обоим условиям

**Оператор $or**

Оператор **$or** представляет логическую операцию ИЛИ и определяет набор пар ключ-значение, которые должны иметься в документе. И если документ имеет хоть одну такую пару ключ-значение, то он соответствует данному запросу и извлекается из бд:

|  |  |
| --- | --- |
| 1 | > db.users.find ({$or : [{name: "Tom"}, {age: 22}]}) |

Это выражение вернет нам все документы, в которых либо name=Tom, либо age=22.

Другой пример вернет нам все документы, в которых name=Tom, а age равно либо 22, либо среди значений languages есть "german":

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom", $or : [{age: 22}, {languages: "german"}]}) |

В подвыраженях or можно применять условные операторы:

|  |  |
| --- | --- |
| 1 | > db.users.find ({$or : [{name: "Tom"}, {age: {$gte:30}}]}) |

В данном случае мы выбираем все документы, где name="Tom" или поле age имеет значение от 30 и выше.

**Оператор $and**

Оператор **$and** представляет логическую операцию И (логическое умножение) и определяет набор критериев, которым обязателньо должен соответствовать документ. В отличие от оператора $or документ должен соответствовать всем указанным критериям. Например:

|  |  |
| --- | --- |
| 1 | > db.users.find ({$and : [{name: "Tom"}, {age: 32}]}) |

Здесь выбираемые документы обязательно должны имееть имя Tom и возраст 32 - оба этих признака.

**Поиск по массивам**

Ряд операторов предназначены для работы с массивами:

* **$all**: определяет набор значений, которые должны иметься в массиве
* **$size**: определяет количество элементов, которые должны быть в массиве
* **$elemMatch**: определяет условие, которым должны соответствовать элементы в массиве

**$all**

Оператор $all определяет массив возможных выражений и требует, чтобы документы имели весь определяемый набор выражений. Соответственно он применяется для поиску по массиву. Например, в документах есть массив languages, хранящий иностранные языки, на которых говорит пользователь. И чтобы найти всех людей, говорящих одновременно и по-английски, и по-французски, мы можем использовать следующее выражение:

|  |  |
| --- | --- |
| 1 | > db.users.find ({languages: {$all : ["english", "french"]}}) |

**Оператор $elemMatch**

Оператор **$elemMatch** позволяет выбрать документы, в которых массивы содержат элементы, попадающие под определенные условия. Например, пусть в базе данных будет коллекция, которая содержит оценки пользователей по определенным курсам. Добавим несколько документов:

|  |  |
| --- | --- |
| 1  2 | > db.grades.insertMany([{student: "Tom", courses:[{name: "Java", grade: 5},  {name: "MongoDB", grade: 4}]},  {student: "Alice", courses:[{name: "C++", grade: 3}, {name: "MongoDB",  grade: 5}]}]) |

Каждый документ имеет массив courses, который в свою очередь состоит из вложенных документов.

Теперь найдем студентов, которые для курса MongoDB имеют оценку выше 3:

|  |  |
| --- | --- |
| 1 | > db.grades.find({courses: {$elemMatch: {name: "MongoDB", grade: {$gt: 3}}}}) |

**Оператор $size**

Оператор $size используется для нахождения документов, в которых массивы имеют число элементов, равным значению $size. Например, извлечем все документы, в которых в массиве laguages два элемента:

|  |  |
| --- | --- |
| 1 | > db.users.find ({languages: {$size:2}}) |

Такой запрос будет соответствовать, например, следующему документу:

|  |  |
| --- | --- |
| 1 | {"name": "Tom", "age": 32, languages: ["english", "german"]} |

**Оператор $exists**

Оператор $exists позволяет извлечь только те документы, в которых определенный ключ присутствует или отсутствует. Например, вернем все документы, в который есть ключ company:

|  |  |
| --- | --- |
| 1 | > db.users.find ({company: {$exists:true}}) |

Если мы укажем у оператора $exists в качестве параметра false, то запрос вернет нам только те документы, в которых не определен ключ company.

**Оператор $type**

Оператор **$type** извлекает только те документы, в которых определенный ключ имеет значение определенного типа, например, строку или число:

|  |  |
| --- | --- |
| 1  2 | > db.users.find ({age: {$type:"string"}})  > db.users.find ({age: {$type:"number"}}) |

**Оператор $regex**

Оператор $regex задает регулярное выражение, которому должно соответствовать значение поля. Например, пусть поле name обязательно имеет букву "b":

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: {$regex:"b"}}) |

Важно понимать, что $regex принимает не просто строки, а именно регулярные выражения, например: name: {$regex:"om$"} - значение name должно оканчиваться на "om".

**Настройка запросов и сортировка**

MongoDB представляет ряд функций, которые помогают управлять выборкой из бд. Одна из них - функция **limit**. Она задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку тремя документами:

|  |  |
| --- | --- |
| 1 | > db.users.find().limit(3) |

В данном случае мы получим первые три документа (если в коллекции 3 и больше документов). Но что, если мы хотим произвести выборку не сначала, а пропустив какое-то количество документов? В этом нам поможет функция **skip**. Например, пропустим первые три записи:

|  |  |
| --- | --- |
| 1 | > db.users.find().skip(3) |

MongoDB предоствляет возможности отсортировать полученный из бд набор данных с помощью функции **sort**. Передавая в эту функцию значения 1 или -1, мы можем указать в каком порядке сортировать: по возрастанию (1) или по убыванию (-1). Во многом эта функция по действию аналогична выражению ORDER BY в SQL. Например, сортировка по возрастанию по полю name:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({name: 1}) |

Ну и в конце надо отметить, что мы можем совмещать все эти функции в одной цепочке:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({name: 1}).skip(3).limit(3) |

**Поиск одиночного документа**

Если все документы извлекаются функцией find, то одиночный документ извлекается функцией findOne. Ее действие аналогично тому, как если бы мы использовали функцию limit(1), которая также извлекает первый документ коллекции. А комбинация функций skip и limit извлечет документ по нужному местоположению.

**Параметр $natural**

Если вдруг нам надо отсортировать ограниченную коллекцию, то мы можем воспользоваться параметром $natural. Этот параметр позволяет задать сортировку: документы передаются в том порядке, в каком они были добавлены в коллекцию, либо в обратном порядке.

Например, отберем последние пять документов:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({ $natural: -1 }).limit(5) |

**Оператор $slice**

$slice является в некотором роде комбинацией функций limit и skip. Но в отличие от них $slice может работать с массивами.

Оператор $slice принимает два параметра. Первый параметр указывает на общее количество возвращаемых документов. Второй параметр необязательный, но если он используется, тогда первый параметр указывает на смещение относительно начала (как функция skip), а второй - на ограничение количества извлекаемых документов.

Например, в каждом документе определен массив languages для хранения языков, на которых говорит человек. Их может быть и 1, и 2, и 3 и более. И допустим, ранее мы добавили следующий объект:

|  |  |
| --- | --- |
| 1 | > db.users.insert({"name": "Tom", "age": "32", languages: ["english",  "german"]}) |

И мы хотим при выводе документов сделать так, чтобы в выборку попадал только один язык из массива languages, а не весь массив:

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

Данный запрос при извлечении документа оставит в результате только первый язык из массива languages, то есть в данном случае english.

Обратная ситуация: нам надо оставить в массиве также один элемент, но не с начала, а с конца. В этом случае необходимо передать в параметр отрицательное значение:

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom"}, {languages: {$slice : -1}}); |

Теперь в массиве окажется german, так как он первый с конца в добавленном элементе.

Используем сразу два параметра:

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom"}, {languages: {$slice : [-1, 1]}}); |

Первый параметр говорит начать выборку элементов с конца (так как отрицательное значение), а второй параметр указывает на количество возвращаемых элементов массива. В итоге в массиве language окажется "german"

**Курсоры**

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

|  |  |
| --- | --- |
| 1 | > var cursor = db.users.find(); null; |

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

Курсоры инкапсулируют в себе наборы получаемых из бд объектов. Используя синтаксис языка javascript и методы курсоров, мы можем вывести полученные документы на экран и как-то их обработать. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5 | > 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. А метод **next** извлекает текущий документ и перемещает курсор к следующему документу в наборе. В итоге в переменной obj оказывается документ, к полям которого мы можем получить доступ.

Также для перебора документов в курсоре в качестве альтернативы мы можем использовать конструкцию итератора javascript - **forEach**:

|  |  |
| --- | --- |
| 1  2  3  4 | > var cursor = db.users.find()  > cursor.forEach(function(obj){  ... print(obj.name);  ... }) |

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | > var cursor = db.users.find();null;  null  > cursor.limit(5);null;  null  > cursor.forEach(function(obj){  ... print(obj.name);  ... }) |

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | > 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 мы хотим отсортировать по убыванию, то вместо 1 используем -1: cursor.sort({name:-1})

И еще один метод skip() позволяет пропустить при выборке определенное количество документов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | > var cursor = db.users.find();null;  null  > cursor.skip(2);null;  null  > cursor.forEach(function(obj){  ... print(obj.name);  ... }) |

В данном случае пропускаем два документа.

И кроме того, можно объединять все эти методы в цепочки:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | > var cursor = db.users.find();null;  null  > cursor.sort({name:1}).limit(3).skip(2);null;  null  > cursor.forEach(function(obj){  ... print(obj.name);  ... }) |