一、查询

find 方法

db.collection_name.find();

查询所有的结果：

select * from users;

db.users.find();

指定返回那些列（键）：

select name, skills from users;

db.users.find({}, {'name' : 1, 'skills' : 1});

补充说明： 第一个{} 放 where 条件 第二个{} 指定那些列显示和不显示 （0 表示不显示 1 表示显示)

where 条件：

1.简单的等于:

select name, age, skills from users where name = 'hurry';

db.users.find({'name' : 'hurry'},{'name' : 1, 'age' : 1, 'skills' : 1});

2.使用 and

select name, age, skills from users where name = 'hurry' and age = 18;

db.users.find({'name' : 'hurry', 'age' : 18},{'name' : 1, 'age' : 1, 'skills' : 1});

3.使用 or

select name, age, skills from users where name = 'hurry' or age = 18;

db.users.find({ '$or' : [{'name' : 'hurry'}, {'age' : 18}] },{'name' : 1, 'age' : 1, 'skills' : 1});

4.<, <=, >, >= ($lt, $lte, $gt, $gte)

select * from users where age >= 20 and age <= 30;

db.users.find({'age' : {'$gte' : 20, '$lte' : 30}});

5.使用 in, not in ($in, $nin)

select * from users where age in (10, 22, 26);

db.users.find({'age' : {'$in' : [10, 22, 26]}});

6.匹配 null

select * from users where age is null;

db.users.find({'age' : null);

7.like (mongoDB 支持正则表达式)

select * from users where name like "%hurry%";

db.users.find({name:/hurry/});

select * from users where name like "hurry%";

db.users.find({name:/^hurry/});

8.使用 distinct

select distinct (name) from users;

db.users.distinct('name');

9.使用 count

select count(*) from users;

db.users.count();

10.数组查询 （mongoDB 自己特有的）

如果 skills 是 ['java','python']

db.users.find({'skills' : 'java'}); 该语句可以匹配成功

$all

db.users.find({'skills' : {'$all' : ['java','python']}}) skills 中必须同时包含 java 和 python

$size

db.users.find({'skills' : {'$size' : 2}}) 遗憾的是$size 不能与$lt 等组合使用

$slice

db.users.find({'skills' : {'$slice : [1,1]}})

两个参数分别是偏移量和返回的数量

11.查询内嵌文档

12.强大的$where 查询
db.foo.find();
{ "_id" : ObjectId("4e17ce0ac39f1afe0ba78ce4"), "a" : 1, "b" : 3, "c" : 10 }
{ "_id" : ObjectId("4e17ce13c39f1afe0ba78ce5"), "a" : 1, "b" : 6, "c" : 6 }

如果要查询 b = c 的文档怎么办？

```
> db.foo.find({"$where":function(){
    for(var current in this){
        for(var other in this){
            if(current != other && this[current] == this[other]){
                return true;
            }
        }
    }
    return false;

}});
```

{ "_id" : ObjectId("4e17ce13c39f1afe0ba78ce5"), "a" : 1, "b" : 6, "c" : 6 }

1) . 大于，小于，大于或等于，小于或等于
$gt:大于
$lt:小于
$gte:大于或等于
$lte:小于或等于
例子：
db.collection.find({ "field" : { $gt: value } }); // greater than : field > value
db.collection.find({ "field" : { $lt: value } }); // less than : field < value
db.collection.find({ "field" : { $gte: value } }); // greater than or equal to : field >= value
db.collection.find({ "field" : { $lte: value } }); // less than or equal to : field <= value

如查询 j 大于 3,小于 4:

```
db.things.find({j : {$lt: 3}});
db.things.find({j : {$gte: 4}});
```

也可以合并在一条语句内:

```
db.collection.find({ "field" : { $gt: value1, $lt: value2 } } ); // value1 < field < value
```

2) 不等于 $ne

例子：

```
db.things.find( { x : { $ne : 3 } } );
```

3) in 和 not in ($in $nin)

语法：

```
db.collection.find( { "field" : { $in : array } } );
```

例子：

```
db.things.find({j:{$in: [2,4,6]}});
```

```
db.things.find({j:{$nin: [2,4,6]}});
```

4) 取模运算$mod

如下面的运算：

```
db.things.find( "this.a % 10 == 1")
```

可用$mod 代替：

db.things.find({ a : { $mod : [10 , 1] } })

5) $all
$all 和$in 类似，但是他需要匹配条件内所有的值：
如有一个对象：
{ a: [1, 2, 3] }

下面这个条件是可以匹配的：

db.things.find({ a: { $all: [2, 3] } });

但是下面这个条件就不行了：

db.things.find({ a: { $all: [2, 3, 4] } });

6) $size
$size 是匹配数组内的元素数量的，如有一个对象：{a:["foo"]}，他只有一个元素：
下面的语句就可以匹配：
db.things.find({ a : { $size: 1 } });

官网上说不能用来匹配一个范围内的元素，如果想找$size<5 之类的，他们建议创建一个字段来保存元素的数量。

You cannot use $size to find a range of sizes (for example: arrays with more than 1 element). If you need to query for a range, create an extra size field that you increment when you add elements.

7）$exists

$exists 用来判断一个元素是否存在：

如：

db.things.find({ a : { $exists : true } }); // 如果存在元素 a,就返回

db.things.find({ a : { $exists : false } }); // 如果不存在元素 a，就返回

8) $type

$type 基于 bson type 来匹配一个元素的类型，像是按照类型 ID 来匹配，不过我没找到 bson 类型和 id 对照表。

db.things.find({ a : { $type : 2 } }); // matches if a is a string
db.things.find({ a : { $type : 16 } }); // matches if a is an int

9）正则表达式

mongo 支持正则表达式，如：

db.customers.find({ name : /acme.*corp/i }); // 后面的 i 的意思是区分大小写

10) 查询数据内的值

下面的查询是查询 colors 内 red 的记录，如果 colors 元素是一个数据,数据库将遍历这个数组的元素来查询。

db.things.find({ colors : "red" });

11) $elemMatch

如果对象有一个元素是数组，那么$elemMatch 可以匹配内数组内的元素：

> t.find({ x : { $elemMatch : { a : 1, b : { $gt : 1 } } } })
{ "_id" : ObjectId("4b5783300334000000000aa9"),
"x" : [{ "a" : 1, "b" : 3 }, 7, { "b" : 99 }, { "a" : 11 }]
}

$elemMatch : { a : 1, b : { $gt : 1 } } 所有的条件都要匹配上才行。

注意，上面的语句和下面是不一样的。

> t.find({ "x.a" : 1, "x.b" : { $gt : 1 } })

$elemMatch 是匹配{ "a" : 1, "b" : 3 }，而后面一句是匹配{ "b" : 99 }, { "a" : 11 }

12) 查询嵌入对象的值

db.postings.find({ "author.name" : "joe" });

注意用法是 author.name，用一个点就行了。更详细的可以看这个链接： dot notation

举个例子：

> db.blog.save({ title : "My First Post", author: {name : "Jane", id : 1}})

如果我们要查询 authors name 是 Jane 的, 我们可以这样：

> db.blog.findOne({"author.name" : "Jane"})

如果不用点，那就需要用下面这句才能匹配：

db.blog.findOne({"author" : {"name" : "Jane", "id" : 1}})

下面这句：

db.blog.findOne({"author" : {"name" : "Jane"}})

是不能匹配的，因为 mongodb 对于子对象，他是精确匹配。

13) 元操作符 $not 取反

如：

db.customers.find({ name : { $not : /acme.*corp/i } });

db.things.find({ a : { $not : { $mod : [10 , 1] } } });

mongodb 还有很多函数可以用，如排序，统计等，请参考原文。

mongodb 目前没有或(or)操作符，只能用变通的办法代替，可以参考下面的链接：

http://www.mongodb.org/display/DOCS/OR+operations+in+query+expressions

二、更新

mongodb 更新有两个命令：
1).update()命令

db.collection.update(criteria, objNew, upsert, multi)

criteria : update 的查询条件，类似 sql update 查询内 where 后面的
objNew ：update 的对象和一些更新的操作符（如$,$inc...）等，也可以理解为 sql update 查询内 set 后面的
upsert ：这个参数的意思是，如果不存在 update 的记录，是否插入 objNew,true 为插入，默认是 false，不插入。
multi ：mongodb 默认是 false,只更新找到的第一条记录，如果这个参数为 true,就把按条件查出来多条记录全部更新。

例：
db.test0.update({ "count" : { $gt : 1 } } , { $set : { "test2" : "OK"} }); 只更新了第一条记录
db.test0.update({ "count" : { $gt : 3 } } , { $set : { "test2" : "OK"} },false,true); 全更新了
db.test0.update({ "count" : { $gt : 4 } } , { $set : { "test5" : "OK"} },true,false); 只加进去了第一条
db.test0.update({ "count" : { $gt : 5 } } , { $set : { "test5" : "OK"} },true,true); 全加进去了
db.test0.update({ "count" : { $gt : 15 } } , { $inc : { "count" : 1} },false,true);全更新了
db.test0.update({ "count" : { $gt : 10 } } , { $inc : { "count" : 1} },false,false);只更新了第一条

2).save()命令

db.collection.save(x)

x 就是要更新的对象，只能是单条记录。

如果在 collection 内已经存在一个和 x 对象相同的"_id"的记录。mongodb 就会把 x 对象替换 collection 内已经存在的记录，否则将会插入 x 对象，如果 x 内没有_id,系统会自动生成一个再插入。相当于上面 update 语句的 upsert=true,multi=false 的情况。
例：
db.test0.save({count:40,test1:"OK"}); #_id 系统会生成
db.test0.save({_id:40,count:40,test1:"OK"}); #如果 test0 内有_id 等于 40 的，会替换，否则插入。

mongodb 的更新操作符：
1) $inc
用法：{ $inc : { field : value } }
意思对一个数字字段 field 增加 value，例：
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 16, "test1" : "TESTTEST", "test2" : "OK", "test3" : "TESTTEST", "test4" : "OK", "test5" : "OK" }

> db.test0.update({ "_id" : 15 } , { $inc : { "count" : 1 } });
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 17, "test1" : "TESTTEST", "test2" : "OK", "test3" : "TESTTEST", "test4" : "OK", "test5" : "OK" }

> db.test0.update({ "_id" : 15 } , { $inc : { "count" : 2 } });

```
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 19, "test1" : "TESTTEST", "test2" : "OK", "test3" : "TESTTEST",
"test4" : "OK", "test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $inc : { "count" : -1 } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : "TESTTEST", "test2" : "OK", "test3" : "TESTTEST",
"test4" : "OK", "test5" : "OK" }
```

2) $set

用法：{ $set : { field : value } }

就是相当于 sql 的 set field = value，全部数据类型都支持$set。例：

```
> db.test0.update( { "_id" : 15 } , { $set : { "test1" : "testv1","test2" : "testv2","test3" : "testv3","test4" : "testv4"
} } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : "testv1", "test2" : "testv2", "test3" : "testv3", "test4" :
"testv4", "test5" : "OK" }
```

3) $unset

用法：{ $unset : { field : 1} }

顾名思义，就是删除字段了。例：

```
> db.test0.update( { "_id" : 15 } , { $unset : { "test1":1 } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test2" : "testv2", "test3" : "testv3", "test4" : "testv4", "test5" :
"OK" }

> db.test0.update( { "_id" : 15 } , { $unset : { "test2": 0 } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test3" : "testv3", "test4" : "testv4", "test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $unset : { "test3":asdfasf } } );
Fri May 14 16:17:38 JS Error: ReferenceError: asdfasf is not defined (shell):0

> db.test0.update( { "_id" : 15 } , { $unset : { "test3":"test" } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test4" : "testv4", "test5" : "OK" }
```

没看出 field : 1 里面的 1 是干什么用的，反正只要有东西就行。

4) $push

用法：{ $push : { field : value } }

把 value 追加到 field 里面去，field 一定要是数组类型才行，如果 field 不存在，会新增一个数组类型加进去。例：

```
> db.test0.update( { "_id" : 15 } , { $set : { "test1" : ["aaa","bbb"] } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb" ], "test4" : "testv4", "test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $push : { "test1": "ccc" } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb", "ccc" ], "test4" : "testv4", "test5" : "OK" }
```

```
> db.test0.update( { "_id" : 15 } , { $push : { "test2": "ccc" } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb", "ccc" ], "test2" : [ "ccc" ], "test4" :
"testv4", "test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $push : { "test1": ["ddd","eee"] } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb", "ccc", [ "ddd", "eee" ] ], "test2" : [ "ccc" ],
"test4" : "testv4", "test5" : "OK" }5) $pushAll
```

5) $pushAll

用法：{ $pushAll : { field : value_array } }

同$push,只是一次可以追加多个值到一个数组字段内。例：

```
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb", "ccc", [ "ddd", "eee" ] ], "test2" : [ "ccc" ],
"test4" : "testv4", "test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $pushAll : { "test1": ["fff","ggg"] } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "aaa", "bbb", "ccc", [ "ddd", "eee" ], "fff", "ggg" ],
"test2" : [ "ccc" ], "test4" : "testv4", "test5" : "OK" }
```

6) $addToSet

用法：{ $addToSet : { field : value } }

增加一个值到数组内，而且只有当这个值不在数组内才增加。例：

```
> db.test0.update( { "_id" : 15 } , { $addToSet : { "test1": {$each : ["444","555"] } } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18,


  "test1" : ["aaa","bbb","ccc",["ddd","eee"],"fff","ggg",["111","222"],"444","555"],


  "test2" : [ "ccc" ], "test4" : "testv4", "test5" : "OK"


 }
> db.test0.update( { "_id" : 15 } , { $addToSet : { "test1": {$each : ["444","555"] } } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18,


  "test1" : ["aaa","bbb","ccc",["ddd","eee"],"fff","ggg",["111","222"],"444","555"], "test2" : [ "ccc" ],


  "test4" : "testv4", "test5" : "OK"
```

}
> db.test0.update({ "_id" : 15 } , { $addToSet : { "test1": ["444","555"] } });
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 18,

"test1" : ["aaa","bbb","ccc",["ddd","eee"],"fff","ggg",["111","222"],"444","555",["444","555"]], "test2" :
["ccc"],

"test4" : "testv4", "test5" : "OK"

}
> db.test0.update({ "_id" : 15 } , { $addToSet : { "test1": ["444","555"] } });
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : ["aaa","bbb","ccc",["ddd","eee"],"fff","ggg",
["111","222"],"444","555",["444","555"]], "test2" : ["ccc"],

"test4" : "testv4", "test5" : "OK"

}

7) $pop
删除数组内的一个值
用法：
删除最后一个值：{ $pop : { field : 1 } }删除第一个值：{ $pop : { field : -1 } }
注意，只能删除一个值，也就是说只能用 1 或-1，而不能用 2 或-2 来删除两条。mongodb 1.1 及以后的版本才可以用，
例：
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 18,

"test1" : ["bbb","ccc",["ddd","eee"],"fff","ggg",["111","222"],"444"],

"test2" : ["ccc"], "test4" : "testv4", "test5" : "OK"

}
> db.test0.update({ "_id" : 15 } , { $pop : { "test1": -1 } });
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 18,

"test1" : ["ccc",["ddd","eee"],"fff","ggg",["111","222"],"444"],

"test2" : ["ccc"], "test4" : "testv4", "test5" : "OK"

}
> db.test0.update({ "_id" : 15 } , { $pop : { "test1": 1 } });
> db.test0.find({ "_id" : 15 });
{ "_id" : { "floatApprox" : 15 }, "count" : 18,

"test1" : ["ccc", ["ddd", "eee"], "fff", "ggg", ["111", "222"]], "test2" : ["ccc"], "test4" : "testv4",
"test5" : "OK"

}

8) $pull

用法：$pull : { field : value } }

从数组 field 内删除一个等于 value 值。例：

```
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "ccc", [ "ddd", "eee" ], "fff", "ggg", [ "111", "222" ] ],
"test2" : [ "ccc" ], "test4" : "testv4",
"test5" : "OK" }

> db.test0.update( { "_id" : 15 } , { $pull : { "test1": "ggg" } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "ccc", [ "ddd", "eee" ], "fff", [ "111", "222" ] ], "test2" :
[ "ccc" ], "test4" : "testv4", "test5"
: "OK" }
```

9) $pullAll

用法：{ $pullAll : { field : value_array } }

同$pull,可以一次删除数组内的多个值。例：

```
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ "ccc", [ "ddd", "eee" ], "fff", [ "111", "222" ] ], "test2" :
[ "ccc" ], "test4" : "testv4", "test5"
: "OK" }

> db.test0.update( { "_id" : 15 } , { $pullAll : { "test1": [ "ccc" , "fff" ] } } );
> db.test0.find( { "_id" : 15 } );
{ "_id" : { "floatApprox" : 15 }, "count" : 18, "test1" : [ [ "ddd", "eee" ], [ "111", "222" ] ], "test2" : [ "ccc" ],
"test4" : "testv4", "test5" : "OK" }
```

10) $ 操作符

$是他自己的意思，代表按条件找出的数组里面某项他自己。呵呵，比较坳口。看一下官方的例子：
> t.find()
{ "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" : [{ "by" : "joe", "votes" : 3 }, { "by" : "jane", "votes" : 7 }] }

> t.update({'comments.by':'joe'}, {$inc:{'comments.$.votes':1}}, false, true)

> t.find()
{ "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" : [{ "by" : "joe", "votes" : 4 }, { "by" : "jane", "votes" : 7 }] }

需要注意的是，$只会应用找到的第一条数组项，后面的就不管了。还是看例子：
> t.find();
{ "_id" : ObjectId("4b9e4a1fc583fa1c76198319"), "x" : [1, 2, 3, 2] }
> t.update({x: 2}, {$inc: {"x.$": 1}}, false, true);
> t.find();

还有注意的是$配合$unset 使用的时候，会留下一个 null 的数组项，不过可以用{$pull:{x:null}}删除全部是 null 的数组项。例：
> t.insert({x: [1,2,3,4,3,2,3,4]})
> t.find()
{ "_id" : ObjectId("4bde2ad3755d00000000710e"), "x" : [1, 2, 3, 4, 3, 2, 3, 4] }
> t.update({x:3}, {$unset:{"x.$":1}})
> t.find()
{ "_id" : ObjectId("4bde2ad3755d00000000710e"), "x" : [1, 2, null, 4, 3, 2, 3, 4] }

{ "_id" : ObjectId("4b9e4a1fc583fa1c76198319"), "x" : [1, 3, 3, 2] }

========== 数组元素操作示例 ==========

> db.arraytest.insert({id:2, name:'leon', comments:[{id:'011', content:'cmt11'}, {id:'012', content:'cmt12'}, {id:'013', content:'cmt13'}]})

1. 数组内的元素可以直接查询

> db.arraytest.find({'comments.id':'002'})

2. 更新数组中的某个节点的值,用$符号

db.arraytest.update({'comments.id':'012'}, {$set: {'comments.$.content':'cmtttt012'}})

3. 删除数组中的某一列，变成 null

> db.arraytest.update({'comments.id':'012'}, {$unset: {'comments.$':1}})

4. 向数组中添加一个元素，如果之前没有元素则会新建数组

> db.arraytest.update({'comments.id':'112'}, {$push: {'comments.$.reply': {'rid':'r21', content:'reply22'}}})
========== 数组元素操作示例 ==========