MONGODB PROJECTION & PROJECTION OPERATORS EXPLAINED

In MongoDB, we use the **find()** method to retrieve data. However, find() retrieves all the fields in a document without any filtering.

MongoDB projection solves this problem by enabling the find() function to be used with data filtering arguments, which allow users to extract only the necessary data fields from a document.

In this hands-on article, we'll show you:

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

How MongoDB projection works

MongoDB projection is a powerful tool that can be used to extract only the fields you need from a document—not all fields. It enables you to:

- Project concise and transparent data
- Filter the data set without impacting the overall database performance

Because MongoDB projection is built on top of the existing **find()** method, you can use any

projection query without significant modifications to the existing functions. Plus, projection is a key factor when finding user-specific data from a given data set.

Basic Syntax of MongoDB Projection

```
db.collection_name.find({},{<field> : <value>})
```

MongoDB projection uses the same find syntax, but we also add a set of parameters to the find function. This set of parameters informs the MongoDB instance of which fields to be returned.

Consider the following collection called “vehicleinformation”

```
mongos> db.vehicleinformation.find().pretty()
{
        "_id" : ObjectId("5faaf5541139223fe5c19a01"),
        "make" : "Nissan",
        "model" : "GTR",
        "year" : 2016,
        "type" : "Sports",
        "reg_no" : "EDS 5578"
}
{
        "_id" : ObjectId("5faaf7131139223fe5c19a02"),
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
{
        "_id" : ObjectId("5faaf7221139223fe5c19a03"),
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
{
        "_id" : ObjectId("5faaf7291139223fe5c19a04"),
        "make" : "Audi",
        "model" : "RS3",
        "year" : 2018,
        "type" : "Sports",
        "reg_no" : "RFD 7866"
}
{
        "_id" : ObjectId("5faaf72f1139223fe5c19a05"),
        "make" : "Ford",
        "model" : "Transit",
        "year" : 2017,
        "type" : "Van",
        "reg_no" : "TST 9880"
}
{
        "_id" : ObjectId("5faaf7371139223fe5c19a06"),
        "make" : "Honda",
        "model" : "Gold Wing",
        "year" : 2018,
        "type" : "Bike",
        "reg_no" : "LKS 2477"
}
mongos>
```

We will try to retrieve the model year of the vehicle make Toyota (make: 'Toyota')

Normal find() statement

```
db.vehicleinformation.find({make : "Toyota"}).pretty()
```

Result:

```
mongos> db.vehicleinformation.find({make : "Toyota"}).pretty()
{
        "_id" : ObjectId("5faaf7221139223fe5c19a03"),
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
```

In this statement, we receive all the data within the document where make is Toyota. To get only the year, we can use projection, as shown below.

MongoDB projection statement

```
db.vehicleinformation.find({make : "Toyota"}, {year : 1}).pretty()
```

Result:

```
mongos> db.vehicleinformation.find({make : "Toyota"}, {year : 1}).pretty()
{ "_id" : ObjectId("5faaf7221139223fe5c19a03"), "year" : 2019 }
mongos>
```

In the projection statement, we use the year field with the Boolean value 1 (True) to indicate that we only require the "year" field. The limitation is that MongoDB will return the object ID (_id) of each document unless we explicitly specify that we do not require the object ID.

MongoDB projection statement with "_id" excluded

```
db.vehicleinformation.find({make : "Toyota"}, {year : 1, _id : 0}).pretty()
```

Result:

```
mongos> db.vehicleinformation.find({make : "Toyota"}, {year : 1, _id : 0}).pretty()
{ "year" : 2019 }
mongos>
```

In this statement, we pass 0 as the Boolean operator to indicate that we do not require the object ID field.

Operators in MongoDB Projection

Using the MongoDB projection method to retrieve specific data from a document will positively impact database performance because it reduces the workload of the find query, minimizing resource usage.

To reduce the workload, MongoDB offers the below operators that can be used within a projection query:

- $
- $elemMatch
- $slice
- $meta

$ Operator

The $ operator is used to limit the contents of an array to project the first element that matches the query condition on the array field. Starting from MongoDB 4.4, if no query condition is present, the first element will be returned in the specified array.

Syntax:

```
db.collection.find( { <array>: <condition> ... }, { "<array>.$": 1 } )
```

Limitations of the $ operator:

- Only a single $ operator can be used in a single query.
- The query must only consist of a single condition on the array field where it will be applied.
- The sort() function in the find() method will be applied before the $ operator. This function may cause the sort order not to be represented correctly.
- The $ operator can only be applied at the end of a field path. This restriction was introduced in MongoDB 4.4 to mitigate any formatting issues.
- The $slice projection expression cannot be used with $ operator as a part of the same expression.

We will use the “studentgrades” Collection to demonstrate the $ operator.

```
db.studentgrades.find({},{_id:0}).pretty()
```

Result:

```
mongos> db.studentgrades.find({},{_id:0}).pretty()
{ "stud_id" : 1, "year" : 1, "grades" : [ 99, 85, 90 ] }
{ "stud_id" : 2, "year" : 2, "grades" : [ 56, 66, 34 ] }
{ "stud_id" : 3, "year" : 2, "grades" : [ 76, 80, 78 ] }
{ "stud_id" : 4, "year" : 2, "grades" : [ 90, 95, 79 ] }
{ "stud_id" : 5, "year" : 3, "grades" : [ 76, 65, 81 ] }
{ "stud_id" : 6, "year" : 3, "grades" : [ 75, 100, 90 ] }
mongos>
```

Using the “studentgrades” collection, we search if a student has received grades equal to or greater than 80.

Let’s see what happens when we don’t use the $ operator. The below code will result in the query returning all the values in the array as the output if any item is equal or greater than 80.

```
db.studentgrades.find( {grades: { $gte: 80}}, {"grades": 1})
```

Result:

```
mongos> db.studentgrades.find( {grades: { $gte: 80}}, {"grades": 1})
{ "_id" : ObjectId("5fab14641139223fe5c19a12"), "grades" : [ 99, 85, 90 ] }
{ "_id" : ObjectId("5fab14701139223fe5c19a14"), "grades" : [ 76, 80, 78 ] }
{ "_id" : ObjectId("5fab14761139223fe5c19a15"), "grades" : [ 90, 95, 79 ] }
{ "_id" : ObjectId("5fab147c1139223fe5c19a16"), "grades" : [ 76, 65, 81 ] }
{ "_id" : ObjectId("5fab1eb5efd219c12eff4aab"), "grades" : [ 75, 100, 90 ] }
mongos>
```

Using the $ operator will only return the first item where it is equal or greater than 80.

```
db.studentgrades.find( {grades: { $gte: 80}}, {"grades.$": 1})
```

Result:

```
mongos> db.studentgrades.find( {grades: { $gte: 80}}, {"grades.$": 1})
{ "_id" : ObjectId("5fab14641139223fe5c19a12"), "grades" : [ 99 ] }
{ "_id" : ObjectId("5fab14701139223fe5c19a14"), "grades" : [ 80 ] }
{ "_id" : ObjectId("5fab14761139223fe5c19a15"), "grades" : [ 90 ] }
{ "_id" : ObjectId("5fab147c1139223fe5c19a16"), "grades" : [ 81 ] }
{ "_id" : ObjectId("5fab1eb5efd219c12eff4aab"), "grades" : [ 100 ] }
mongos>
```

$elemMatch Operator

$elemMatch operator will limit the contents of an array to the first element that matches a given condition. This condition differs from the $ operator because the $elemMatch projection operator requires an explicit condition argument.

Let's see the syntax of using $elemMatch in find() method.

Syntax:

```
db.collection.find( { <array>: <condition> ... }, { "<array>.$elemMatch": ($elemMatch operator) } )
```

Limitations of the $elemMatch operator

- Regardless of the ordering of fields in the document, the field to which $elemMatch projection is applied will be returned as the last field of the document.
- find() operations done on MongoDB views do not support $elemMatch projection operator.
- $text query expressions are not supported with the $elemMatch operator.

We'll use the following "schooldata" collection to demonstrate the $elemMatch operator.

```
db.schooldata.find()
```

Result:

```
mongos> db.schooldata.find()
{ "_id" : ObjectId("5fab2329efd219c12eff4aaf"), "shool_id" : 1, "address" : "Londo
n", "students" : [ { "name" : "barry", "age" : "10" }, { "name" : "jane", "age" :
"12" }, { "name" : "harry", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2330efd219c12eff4ab0"), "shool_id" : 2, "address" : "Paris
", "students" : [ { "name" : "toby", "age" : "14" }, { "name" : "matt", "age" : "1
0" }, { "name" : "jake", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2337efd219c12eff4ab1"), "shool_id" : 3, "address" : "New Y
ork", "students" : [ { "name" : "selena", "age" : "8" }, { "name" : "penny", "age"
 : "9" }, { "name" : "nate", "age" : "9" } ] }
{ "_id" : ObjectId("5fab233cefd219c12eff4ab2"), "shool_id" : 4, "address" : "Londo
n", "students" : [ { "name" : "henry", "age" : "10" }, { "name" : "alex", "age" :
"12" }, { "name" : "seth", "age" : "15" } ] }
mongos>
```

Now we'll run this on a single field. With the $elemMatch operator, we will search for schools in London with students who are ten years old. The pretty() method is used to format the output.

```
db.schooldata.find({address: "London"}, {"students" : {$elemMatch: {"age":
```

```
"10"}}}).pretty()
```

Result:

```
mongos> db.schooldata.find({address: "London"}, {"students" : {$elemMatch: {"age":
 "10"}}}).pretty()
{
        "_id" : ObjectId("5fab2329efd219c12eff4aaf"),
        "students" : [
                {
                        "name" : "barry",
                        "age" : "10"
                }
        ]
}
{
        "_id" : ObjectId("5fab233cefd219c12eff4ab2"),
        "students" : [
                {
                        "name" : "henry",
                        "age" : "10"
                }
        ]
}
mongos>
```

In the below example, we'll check multiple fields. Here, we check for both the age and name of the students. We check if a student named "barry" who is ten years old is attending any school in London.

```
db.schooldata.find({address: "London"}, {"students" : {$elemMatch: {name:
"barry", "age": "10"}}}).pretty()
```

Result:

```
mongos> db.schooldata.find({address: "London"}, {"students" : {$elemMatch: {name: "barry",
 "age": "10"}}}).pretty()
{
        "_id" : ObjectId("5fab2329efd219c12eff4aaf"),
        "students" : [
                {
                        "name" : "barry",
                        "age" : "10"
                }
        ]
}
```

$slice Projection Operator

The $slice operator specifies the number of elements that should be returned as the output of a query.

Syntax:

```
db.collection.find( <query>,  { <array Field>: { $slice: <number> } })
```

Limitations in $slice operator:

- With the introduction of MongoDB 4.4, the $slice operator will only return the sliced element. It will not return any other item in a nested array.
- The $slice operator does not support the find() operation done on MongoDB views.
- The $slice operator cannot be used in conjunction with the $ projection operator due to the MongoDB restriction, where top-level fields can't consist of $ (dollar sign) as a part of the field name.
- Queries can't contain the $slice of an array and a field embedded in the array as part of the same statement to eliminate path collisions from MongoDB 4.4.

Now we'll use the "schooldata" collection to demonstrate the $slice operator.

```
db.schooldata.find()
```

Result:

```
mongos> db.schooldata.find( )
{ "_id" : ObjectId("5fab2329efd219c12eff4aaf"), "shool_id" : 1, "address" : "Londo
n", "students" : [ { "name" : "barry", "age" : "10" }, { "name" : "jane", "age" :
"12" }, { "name" : "harry", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2330efd219c12eff4ab0"), "shool_id" : 2, "address" : "Paris
", "students" : [ { "name" : "toby", "age" : "14" }, { "name" : "matt", "age" : "1
0" }, { "name" : "jake", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2337efd219c12eff4ab1"), "shool_id" : 3, "address" : "New Y
ork", "students" : [ { "name" : "selena", "age" : "8" }, { "name" : "penny", "age"
 : "9" }, { "name" : "nate", "age" : "9" } ] }
{ "_id" : ObjectId("5fab233cefd219c12eff4ab2"), "shool_id" : 4, "address" : "Londo
n", "students" : [ { "name" : "henry", "age" : "10" }, { "name" : "alex", "age" :
"12" }, { "name" : "seth", "age" : "15" } ] }
mongos>
```

Let's slice the first element in an array. To achieve this result, we use the $slice operator to slice only the first element from the student's array.

```
db.schooldata.find({}, {students: {$slice : 1}})
```

Result:

```
mongos> db.schooldata.find({}, {students: {$slice : 1}})
{ "_id" : ObjectId("5fab2329efd219c12eff4aaf"), "shool_id" : 1, "address" : "London", "stu
dents" : [ { "name" : "barry", "age" : "10" } ] }
{ "_id" : ObjectId("5fab2330efd219c12eff4ab0"), "shool_id" : 2, "address" : "Paris", "stud
ents" : [ { "name" : "toby", "age" : "14" } ] }
{ "_id" : ObjectId("5fab2337efd219c12eff4ab1"), "shool_id" : 3, "address" : "New York", "s
tudents" : [ { "name" : "selena", "age" : "8" } ] }
{ "_id" : ObjectId("5fab233cefd219c12eff4ab2"), "shool_id" : 4, "address" : "London", "stu
dents" : [ { "name" : "henry", "age" : "10" } ] }
mongos>
```

Now, we'll slice the last element in the array. Use the $slice operator with "-1" to indicate the last record.

```
db.schooldata.find({}, {students: {$slice : -1}})
```

Result:

```
mongos> db.schooldata.find({}, {students: {$slice : -1}})
{ "_id" : ObjectId("5fab2329efd219c12eff4aaf"), "shool_id" : 1, "address" : "London", "stu
dents" : [ { "name" : "harry", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2330efd219c12eff4ab0"), "shool_id" : 2, "address" : "Paris", "stud
ents" : [ { "name" : "jake", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2337efd219c12eff4ab1"), "shool_id" : 3, "address" : "New York", "s
tudents" : [ { "name" : "nate", "age" : "9" } ] }
{ "_id" : ObjectId("5fab233cefd219c12eff4ab2"), "shool_id" : 4, "address" : "London", "stu
dents" : [ { "name" : "seth", "age" : "15" } ] }
mongos>
```

To skip elements, we have to define the elements to be skipped and the number of elements to be returned. The following examples demonstrate this scenario.

Skip the first element and return the next two elements:

```
db.schooldata.find({}, {students: {$slice : }})
```

Result:

```
mongos> db.schooldata.find({}, {students: {$slice : [1, 2]}})
{ "_id" : ObjectId("5fab2329efd219c12eff4aaf"), "shool_id" : 1, "address" : "London", "stu
dents" : [ { "name" : "jane", "age" : "12" }, { "name" : "harry", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2330efd219c12eff4ab0"), "shool_id" : 2, "address" : "Paris", "stud
ents" : [ { "name" : "matt", "age" : "10" }, { "name" : "jake", "age" : "15" } ] }
{ "_id" : ObjectId("5fab2337efd219c12eff4ab1"), "shool_id" : 3, "address" : "New York", "s
tudents" : [ { "name" : "penny", "age" : "9" }, { "name" : "nate", "age" : "9" } ] }
{ "_id" : ObjectId("5fab233cefd219c12eff4ab2"), "shool_id" : 4, "address" : "London", "stu
dents" : [ { "name" : "alex", "age" : "12" }, { "name" : "seth", "age" : "15" } ] }
mongos>
```

In this statement, we slice the "students" array by skipping the first element and returning the remaining two elements. These were defined inside the square brackets []. The first value is the element to be ignored, while the second value is how many remaining elements to be returned. If the array has less than two elements after skipping the first element, it will return all the remaining elements.

$meta Operator

The $meta operator is used to obtain the metadata associated with a document. The $meta operator can be used with both MongoDB Aggregation and MongoDB projection. This section will only cover the basics of the MongoDB projection and how the $meta operator will be used in projection.

The $meta operator uses the following syntax.

```
{ $meta: <metaDataKeyword> }
```

For the "metaDataKeyword", you can associate these values:

- **"textScore" Keyword.** The "textScore" will return the score associated with the given $text string. This keyword essentially finds how exactly the search term is matched with the given field in a document. $text query is mandatory when using the "textScore" keyword from MongoDB 4.4.
- **"indexKey" Keyword.** Introduced in MongoDB 4.4 as a preferred alternative to the

cursor.returnKey() function. This function returns an index key for a document where a non-text index is used. This keyword is for debugging purposes only, not to be used in production environments.

Usage and Limitations of $meta operator in projection:

- The {$meta: "$textScore"} expression can be a part of a projection query in a document to obtain the text score. The expression can be either an inclusion or exclusion in the projection statement. The only limitation is that if the expression is set to an existing field in the document, the projected metadata value will overwrite the current value in the field.
- You can't define "text score" in the find() method. Hence, the best choice is to use MongoDB aggregation.
- The $meta expression can be used within a sort() function. The "textScore" metadata will be sorted in descending order.
- When {$meta: "$textScore"} expression is included in both the projection and sort functions, each function can have different field names for the expression. The query system will disregard the field name in sort().

Now, let's look at a few examples:

For {$meta : "textScore"}, we will be using the food collection.

```
db.food.find().pretty()
```

Result:

```
mongos> db.food.find().pretty()
{
        "_id" : ObjectId("5fab4f65efd219c12eff4ab3"),
        "f_id" : 1,
        "food_desc" : "pizza"
}
{
        "_id" : ObjectId("5fab4f6aefd219c12eff4ab4"),
        "f_id" : 2,
        "food_desc" : "burger"
}
{
        "_id" : ObjectId("5fab4f6fefd219c12eff4ab5"),
        "f_id" : 3,
        "food_desc" : "pizza and coke"
}
{
        "_id" : ObjectId("5fab4f75efd219c12eff4ab6"),
        "f_id" : 4,
        "food_desc" : "a slice of pizza"
}
{
        "_id" : ObjectId("5fab4f7aefd219c12eff4ab7"),
        "f_id" : 5,
        "food_desc" : "chicken wings"
}
{
        "_id" : ObjectId("5fab4f80efd219c12eff4ab8"),
        "f_id" : 6,
        "food_desc" : "large pizza"
}
mongos>
```

Let's create an index for "food" collection. We will be using the "food_desc" field to create a text index.

```
db.food.createIndex({"food_desc": "text"})
```

Result:

```
mongos> db.food.createIndex({"food_desc": "text"})
{
        "raw" : {
                "ShardReplSet/10.10.10.58:27018" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1605062797, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1605062797, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Now let's find the text score. We use the word "pizza" as the search term and use the $meta operator to obtain the text score for each of the documents. The (_id: 0) field is used to omit the object id of each document for a more straightforward output.

```
db.food.find( {$text : {$search: "pizza"}}, {score: {$meta: "textScore"}, _id: 0})
```

Result:

```
mongos> db.food.find( {$text : {$search: "pizza"}}, {score: {$meta: "textScore"}, _id: 0})
{ "f_id" : 1, "food_desc" : "pizza", "score" : 1.1 }
{ "f_id" : 4, "food_desc" : "a slice of pizza", "score" : 0.75 }
{ "f_id" : 3, "food_desc" : "pizza and coke", "score" : 0.75 }
{ "f_id" : 6, "food_desc" : "large pizza", "score" : 0.75 }
mongos>
```

Now onto {$meta: "indexKey"}. As "indexKey" is strictly for debugging purposes only, the official MongoDB documentation states the following.

```
NOTE
The { $meta: "indexKey" } expression is for debugging purposes only and not for application logic. MongoDB returns the value associated with the index chosen by the query system. The system can choose a different index upon subsequent execution.
For the selected index, the return value depends on how the database decides to represent values in an index and may change across versions. The represented value may not be the actual value for the field.
```

The following examples are created using the "vehiclesales" collection. Using {_id:0} field, object ids are omitted from the output.

```
db.vehiclesales.find({},{_id:0})
```

Result:

```
mongos> db.vehiclesales.find({},{_id:0})
{ "make" : "Audi", "model" : "RS3", "price" : 47000, "type" : "Sports" }
{ "make" : "BMW", "model" : "X3", "price" : 35000, "type" : "SUV" }
{ "make" : "Audi", "model" : "A1", "price" : 25000, "type" : "Compact" }
{ "make" : "Nissan", "model" : "GTR", "price" : 55000, "type" : "Sports" }
{ "make" : "Toyota", "model" : "Yaris", "price" : 21000, "type" : "Compact" }
{ "make" : "Audi", "model" : "RS5", "price" : 77000, "type" : "Sports" }
mongos>
```

Here, we create a compound index using the fields “make” and “type”.

```
db.vehiclesales.createIndex({make : 1, type: 1})
```

Result:

```
mongos> db.vehiclesales.createIndex({make : 1, type: 1})
{
        "raw" : {
                "ShardReplSet/10.10.10.58:27018" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1605065050, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1605065050, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

Next let’s do a Projection with an index. We search for documents in the “vehiclesales” collection where the type is equal to “Sports” and use the $meta operator to obtain the index key for each document.

```
db.vehiclesales.find({type: "Sports"},{idxKey: {$meta: "indexKey"}}).pretty()
```

Result:

```
mongos> db.vehiclesales.find({type: "Sports"},{idxKey: {$meta: "indexKey"}}).prett
y()
{
        "_id" : ObjectId("5fab57beefd219c12eff4ab9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : 47000,
        "type" : "Sports",
        "idxKey" : {
                "type" : "Sports",
                "make" : "Audi"
        }
}
{
        "_id" : ObjectId("5fab57e0efd219c12eff4abe"),
        "make" : "Audi",
        "model" : "RS5",
        "price" : 77000,
        "type" : "Sports",
        "idxKey" : {
                "type" : "Sports",
                "make" : "Audi"
        }
}
{
        "_id" : ObjectId("5fab57d5efd219c12eff4abc"),
        "make" : "Nissan",
        "model" : "GTR",
        "price" : 55000,
        "type" : "Sports",
        "idxKey" : {
                "type" : "Sports",
                "make" : "Nissan"
        }
}
mongos>
```

What about projections without an index? If we request a field that is not part of the index, the query will not return any value as the index will be null. In this instance, the index will not be returned as the price field is not a part of the index.

```
db.vehiclesales.find({price: {$gte: 47000}},{idxKey: {$meta:
"indexKey"}}).pretty()
```

Result:

```
mongos> db.vehiclesales.find({price: {$gte: 47000}},{idxKey: {$meta: "indexKey"}})
.pretty()
{
        "_id" : ObjectId("5fab57beefd219c12eff4ab9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : 47000,
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fab57d5efd219c12eff4abc"),
        "make" : "Nissan",
        "model" : "GTR",
        "price" : 55000,
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fab57e0efd219c12eff4abe"),
        "make" : "Audi",
        "model" : "RS5",
        "price" : 77000,
        "type" : "Sports"
}
mongos>
```

That concludes this informational tutorial.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Guide, a series of articles and tutorials
- MongoDB: The Mongo Shell & Basic Commands
- Data Storage Explained: Data Lake vs Warehouse vs Database