Create a Sample Database

We will create a database with name *myDB*

```
> use myDB
```

Create a Collection

We will create a collection with name *orders*.

```
> db.createCollection("orders")
```

MongoDB doesn't use the rows and columns. It stores the data in a document format. A collection is a group of documents.

Check all Collections

We can check all collections in a database by using the following statement.

```
>show collections
orders
```

Insert One Document in Collection

```
db.orders.insertOne(
        {
                Customer: "abc",
                Address:{"City":"Jaipur","Country":"India"},
                PaymentMode:"Card",
                Email:"abc@mail.in",
                OrderTotal: 1000.00,
                OrderItems:[
                        {"ItemName":"notebook","Price":"150.00","Qty":10},
                        {"ItemName":"paper","Price":"10.00","Qty":5},
                        {"ItemName":"journal","Price":"200.00","Qty":2},
                        {"ItemName":"postcard","Price":"10.00","Qty":500}
        })
```

Insert Many Documents Together in Collection

```
db.orders.insertMany([
        {

                Customer: "xyz",
                Address:{"City":"Delhi","Country":"India"},
                PaymentMode:"Cash",
                OrderTotal: 800.00,
                OrderItems:[
                    {"ItemName":"notebook","Price":"150.00","Qty":5},
                    {"ItemName":"paper","Price":"10.00","Qty":5},
                    {"ItemName":"postcard","Price":"10.00","Qty":500}
                ]
        },
        {
                Customer: "ron",
                Address:{"City":"New York","Country":"USA"},
                PaymentMode:"Card",
                Email:"ron@mail.com",
                OrderTotal: 800.00,
                OrderItems:[
                    {"ItemName":"notebook","Price":"150.00","Qty":5},
                    {"ItemName":"postcard","Price":"10.00","Qty":00}
                ]
        }
])
```

A document is the equivalent of an RDBMS row. It doesn't need to have the same schema in each document. It means a document might not have any field that doesn't have any value.

Query Documents

find() method

We need to use the **find()** method to query documents from MongoDB collections. The following statement will retrieve all documents from the collection.

```
> db.orders.find()
```

The result will be:

```
{
                "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                Customer: "abc",
                Address:{"City":"Jaipur","Country":"India"},
                PaymentMode:"Card",
                Email:"abc@mail.com",
                OrderTotal: 1000.00,
                OrderItems:[
                    {"ItemName":"notebook","Price":"150.00","Qty":10},
                    {"ItemName":"paper","Price":"10.00","Qty":5},
                    {"ItemName":"journal","Price":"200.00","Qty":2},
                    {"ItemName":"postcard","Price":"10.00","Qty":500}
                ]
        },
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
		Customer: "xyz",
		Address:{"City":"Delhi","Country":"India"},
		PaymentMode:"Cash",
		OrderTotal: 800.00,
		OrderItems:[
			{"ItemName":"notebook","Price":"150.00","Qty":5},
			{"ItemName":"paper","Price":"10.00","Qty":5},
			{"ItemName":"postcard","Price":"10.00","Qty":500}
		]
	},
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
		Customer: "ron",
		Address:{"City":"New York","Country":"USA"},
		PaymentMode:"Card",
		Email:"ron@mail.com",
		OrderTotal: 600.00,
		OrderItems:[
			{"ItemName":"notebook","Price":"150.00","Qty":5},
			{"ItemName":"postcard","Price":"10.00","Qty":00}
		]
	}
```

Projection

If we want to fetch only selected fields then we can use the projection. Following statement will fetch only *Customer* and *Email* field.

```
> db.orders.find( { }, { Customer: 1, Email: 1 })
```

The result will be

```
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607534c")
		Customer: "abc",
		Email:"abc@mail.com"
	},
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
		Customer: "xyz"
	},
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
		Customer: "ron",
		Email:"ron@mail.com"
```

```
            }
```

Filter the Documents by Specifying a Condition

Now we will learn how we can fetch the documents that match a specified condition. MongoDB provides many comparison operators for this.

1. $eq Operator

The $eq operator checks the equality of the field value with the specified value. To fetch the order where *PaymentMode* is 'Card' we can use the following statement

>db.orders.find({ PaymentMode: { $eq: "Card" } })

The query can also be written as:

>db.orders.find({ PaymentMode: "Card" })

Example
>db.orders.find({ PaymentMode: "Card" }, { Customer: 1, PaymentMode: 1 })

```
            {
                        "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                        Customer: "abc",
                        PaymentMode:"Card"
            },
            {
                        "_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
                        Customer: "ron",
                        PaymentMode:"Card"
            }
```

$eq Operator with embedded document

We may have noticed that we inserted an embedded document *Address* in the *Orders* collection. If we want to fetch the order where *Country* is 'India' we can use a dot notation like the following statement.

>db.orders.find({ "Address.Country": { $eq: "India" } })

This query can be written also like below

>db.orders.find({ "Address.Country":"India" })

Example

>db.orders.find({ "Address.Country": { $eq: "India" } } , { Customer: 1, Address: 1 })

The result will be as follows:

```
            {
                        "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                        Customer: "abc",
                    Address:{"City":"Jaipur","Country":"India"}
          },
          {
                    "_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
                    Customer: "xyz",
                    Address:{"City":"Delhi","Country":"India"}
          }
```

$eq Operator with array

$eq operator will retrieve all the documents if the specified condition is true for any item in an array. We have an *OrderItems* array in the document. If we want to filter the documents where 'paper' were also ordered then the statement would be as follows.

```
>db.orders.find( { "OrderItems.ItemName": { $eq: "paper" } } )
```

This query can be written also like below

```
>db.orders.find( { "OrderItems.ItemName": "paper"  } )
```

Example

```
>db.orders.find( { "OrderItems.ItemName": { $eq: "paper" } } , { Customer: 1, OrderItems: 1 })
```

The result will be as follows:

```
{
                    "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                    Customer: "abc",
                    OrderItems:[
                              {"ItemName":"notebook","Price":"150.00","Qty":10},
                              {"ItemName":"paper","Price":"10.00","Qty":5},
                              {"ItemName":"journal","Price":"200.00","Qty":2},
                              {"ItemName":"postcard","Price":"10.00","Qty":500}
                    ]
          },
          {
                    "_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
                    Customer: "xyz",
                    OrderItems:[
                              {"ItemName":"notebook","Price":"150.00","Qty":5},
                              {"ItemName":"paper","Price":"10.00","Qty":5},
                              {"ItemName":"postcard","Price":"10.00","Qty":500}
                    ]
          }
```

2. $gt Operator

We can use the $gt operator to retrieve the documents where a field’s value is greater than the specified value. The following statement will fetch the documents where *OrderTotal* is greater than 800.

```
>db.orders.find( { OrderTotal: { $gt: 800.00 } } )
```

Example

```
>db.orders.find( { "OrderTotal": { $gt: 800.00 } } , { Customer: 1, OrderTotal: 1 })
```

The result will be as follows:

```
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                Customer: "abc",
                OrderTotal: 1000.00
        }
```

3. $gte Operator

We can use the $gte operator to retrieve the documents where a field’s value is greater than or equal to the specified value. The following statement will fetch the documents where *OrderTotal* is greater than or equal to 800.

```
>db.orders.find( { OrderTotal: { $gte: 800.00 } } )
```

Example

```
>db.Orders.find( { "OrderTotal": { $gte: 800.00 } } , { Customer: 1, OrderTotal: 1 })
```

The result will be as follows:

```
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607534c")
                Customer: "abc",
                OrderTotal: 1000.00
        },
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
                Customer: "xyz",
                OrderTotal: 800.00
        }
```

4. $lt Operator

We can use the $lt operator to retrieve the documents where a field’s value is less than the specified value. The following statement will fetch the documents where *OrderTotal* is less than 800.

```
>db.orders.find( { OrderTotal: { $lt: 800.00 } } )
```

Example

```
>db.orders.find( { "OrderTotal": { $lt: 800.00 } } , { Customer: 1, OrderTotal: 1 })
```

The result will be as follows:

```
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
                Customer: "ron",
                OrderTotal: 600.00
        }
```

4. $lte Operator

We can use the $lte operator to retrieve the documents where a field's value is less than or equal to the specified value. Following statement will fetch the documents where *OrderTotal* is less than or equal to 800.

```
>db.orders.find( { OrderTotal: { $lte: 800.00 } } )
```

Example

```
>db.orders.find( { "OrderTotal": { $lte: 800.00 } } , { Customer: 1, OrderTotal: 1 })
```

The result will be as follow:

```
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
                Customer: "xyz",
                OrderTotal: 800.00
        },
        {
                "_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
                Customer: "ron",
                OrderTotal: 600.00
        }
```

5. $ne Operator

We can use the $ne operator to retrieve the documents where a field's value is not equal to the specified value.

```
>db.orders.find( { PaymentMode: { $ne: "Card" } } )
```

Example

```
>db.orders.find( { "PaymentMode": { $ne: "Card" } } , { Customer: 1, PaymentMode: 1 })
```

The result will be as follow:

```
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
		Customer: "xyz",
		PaymentMode":"Cash"
	}
```

6. $in Operator

We can use the $in operator to retrieve the documents where a field’s value is equal to any value in the specified array.

```
>db.orders.find( { OrderItems.ItemName: { $in: ["journal","paper"] } } )
```

Example

>db.orders.find({ OrderItems.ItemName: { $in: ["journal","paper"] } } , { Customer: 1, OrderItems: 1 })

The result will be as follow:

```
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607534c")
		Customer: "abc",
		OrderItems:[
			{"ItemName":"notebook","Price":"150.00","Qty":10},
			{"ItemName":"paper","Price":"10.00","Qty":5},
			{"ItemName":"journal","Price":"200.00","Qty":2},
			{"ItemName":"postcard","Price":"10.00","Qty":500}
		]
	},
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607544c"),
		Customer: "xyz",
		OrderItems:[
			{"ItemName":"notebook","Price":"150.00","Qty":5},
			{"ItemName":"paper","Price":"10.00","Qty":5},
			{"ItemName":"postcard","Price":"10.00","Qty":500}
		]
	}
```

7. $nin Operator

We can use the $nin operator to retrieve the documents where a field’s value is not equal to any value in the specified array. It will also select the documents where the field does not exist.

```
>db.orders.find( { OrderItems.ItemName: { $nin: ["journal","paper"] } } )
```

Example

```
>db.orders.find( { OrderItems.ItemName: { $nin: ["journal","paper"] } } , { Customer: 1, OrderItems: 1 })
```

The result will be as follow:

```
	{
		"_id" : ObjectId("5dd4e2cc0821d3b44607644c"),
		Customer: "ron",
		OrderItems:[
			{"ItemName":"notebook","Price":"150.00","Qty":5},
			{"ItemName":"postcard","Price":"10.00","Qty":00}
		]
	}
```

Aggregate Functions

Operator	**Meaning**
$count	Calculates the quantity of documents in the given group.
$max	Displays the maximum value of a document’s field in the collection.
$min	Displays the minimum value of a document’s field in the collection.
$avg	Displays the average value of a document’s field in the collection.
$sum	Sums up the specified values of all documents in the collection.

> db.orders.aggregate([{$group:{ _id:”PaymentMode”, total:{$**count**: “OrderTotal”}}}])

> db.orders.aggregate([{$group:{ _id:”PaymentMode”, total:{$**max**: “OrderTotal”}}}])

> db.orders.aggregate([{$group:{ _id:”PaymentMode”, total:{$**min**: “OrderTotal”}}}])

> db.orders.aggregate([{$group:{ _id:”PaymentMode”, total:{$**avg**: “OrderTotal”}}}])

> db.orders.aggregate([{$group:{ _id:”PaymentMode”, total:{$**sum**: “OrderTotal”}}}])

To find Distinct Results:

```
>db.orders.distinct(“OrderItems.ItemName”)
```

To update a particular value:

```
>db.orders.updateMany({‘Address.Country’:’India’},{$set:{ ‘Address.Country’:’Bharat’}})
```

To rename a collection:

```
>db.orders.renameCollection(‘OrderDetails’)
```

To delete the entry from the collection

```
>db.OrderDetails.deleteOne({‘Address.City’:’Delhi’})
```

To delete multiple entries from the collection:

```
>db.OrderDetails.deleteMany({‘PaymentMode’:’Card’’})
```

To check version of the database

```
>db.version()
```

To list MongoDB Commands

```
>db.help()
```

To get database Statistics:

```
>db.stats()
```

To drop database

```
>db.dropDatabase()
```