MongoDB

Day2

- Query operators
- Update operators
- Projections & Cursor
- Aggregation & map–reduce

Query operators

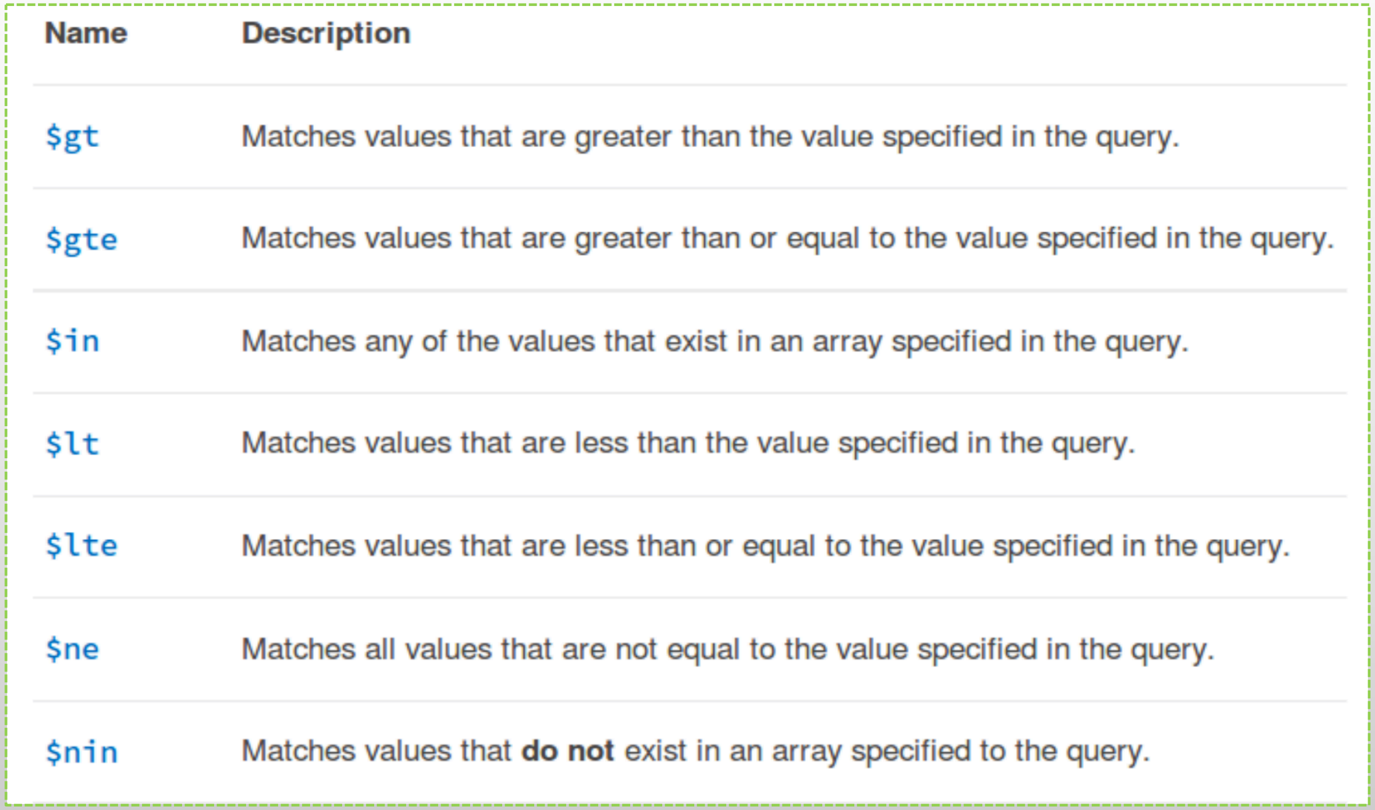
Comparison

Logical

Arrays

Element

Comparison operators



Name	Description
`$gt`	Matches values that are greater than the value specified in the query.
`$gte`	Matches values that are greater than or equal to the value specified in the query.
`$in`	Matches any of the values that exist in an array specified in the query.
`$lt`	Matches values that are less than the value specified in the query.
`$lte`	Matches values that are less than or equal to the value specified in the query.
`$ne`	Matches all values that are not equal to the value specified in the query.
`$nin`	Matches values that **do not** exist in an array specified to the query.

Comparison operators

```
db.collection.find({property_name:{$operator:value}})
```

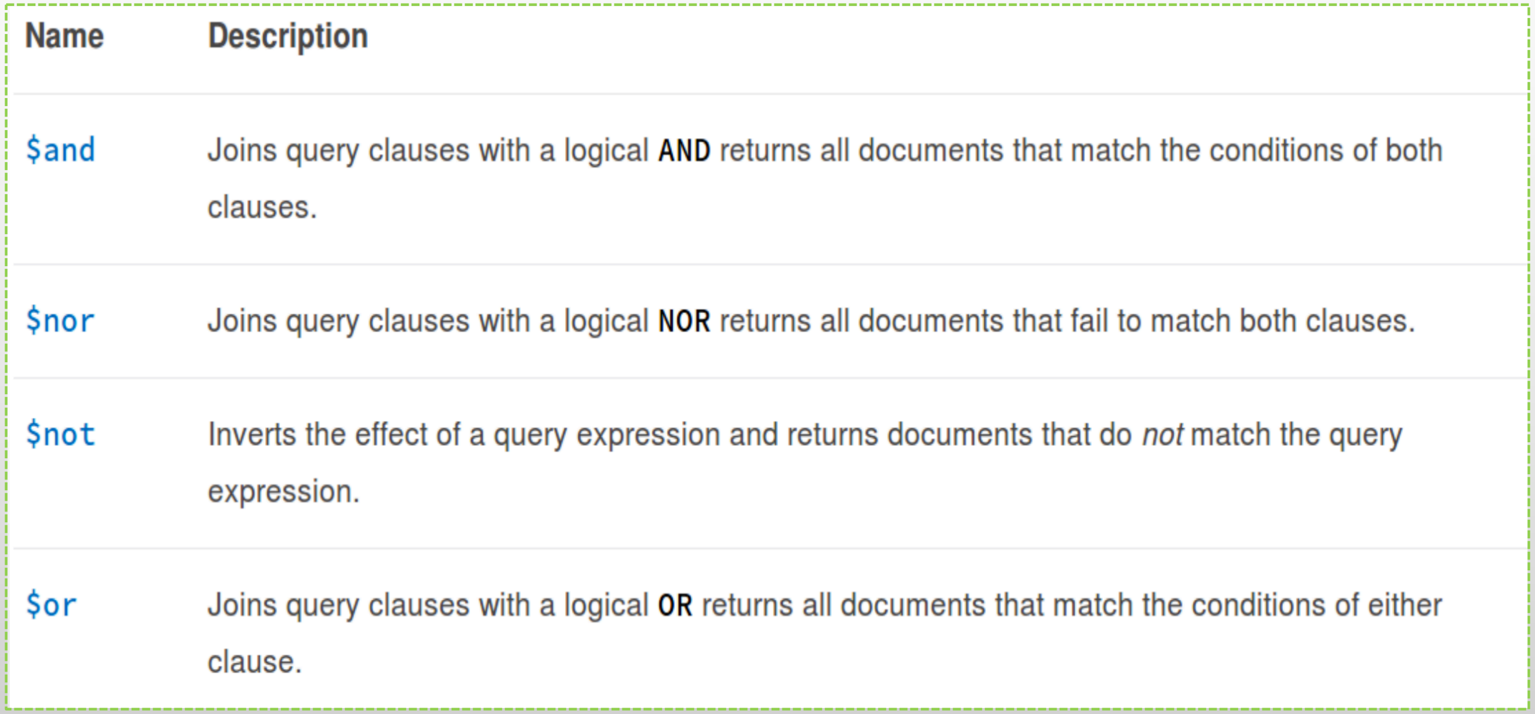
Example :

```
db.employee.find({role:{$in: ["designer","programmer"]}})
```

```
db.employee.find({salary:{$gt:1200}})
```

```
db.employee.find({_id:{$ne:10}})
```

Logical operators



Name	Description
`$and`	Joins query clauses with a logical **AND** returns all documents that match the conditions of both clauses.
`$nor`	Joins query clauses with a logical **NOR** returns all documents that fail to match both clauses.
`$not`	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
`$or`	Joins query clauses with a logical **OR** returns all documents that match the conditions of either clause.

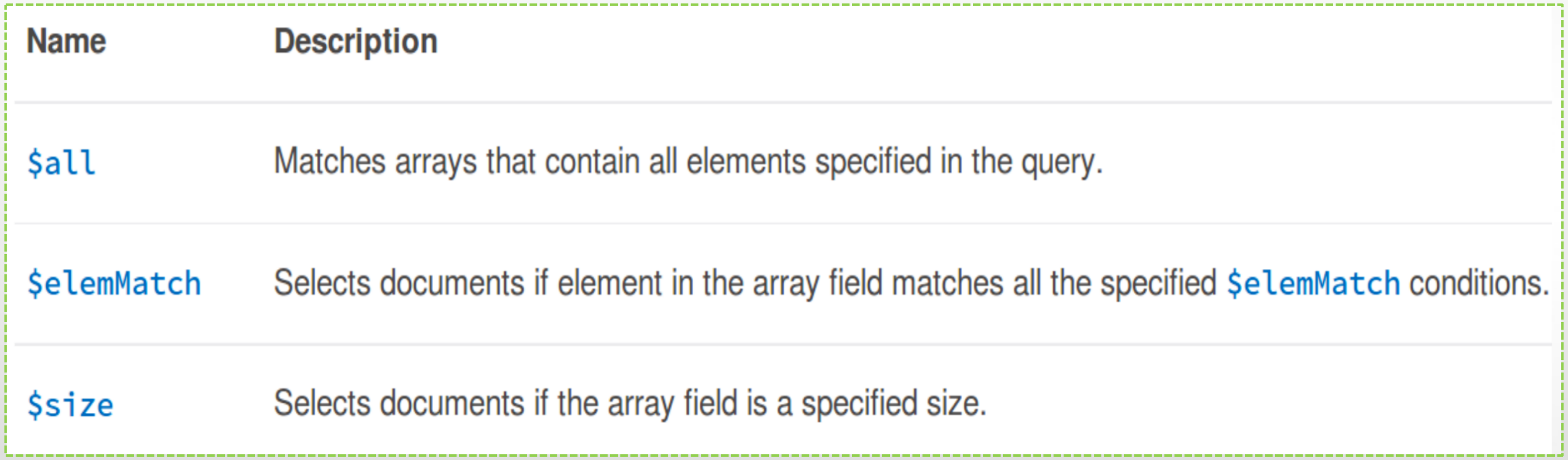
Logical operators

Example :

```
db.employee.find({$and:[{fname: "ahmed"},{"lname":"mahmoud"}] })
```

```
db.employee.find({salary:{$not:{$gt:1200} }})
```

Array operators



Name	Description
`$all`	Matches arrays that contain all elements specified in the query.
`$elemMatch`	Selects documents if element in the array field matches all the specified `$elemMatch` conditions.
`$size`	Selects documents if the array field is a specified size.

Array operators

Example :

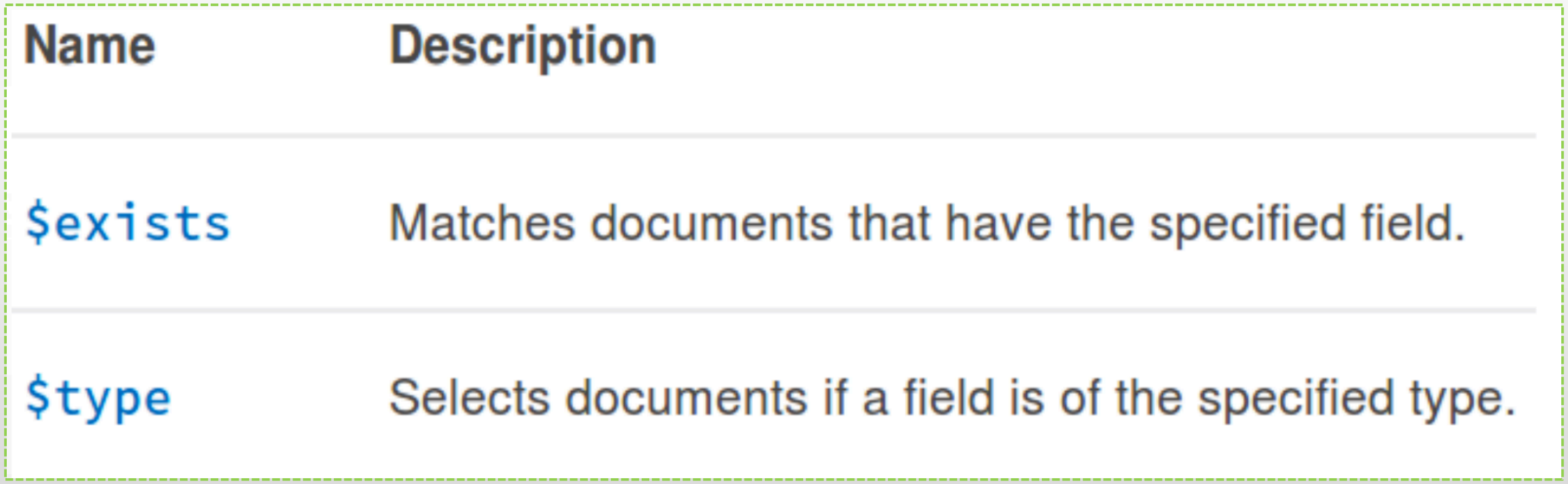
db . scores . find({ results : { **$elemMatch** : { $gte : 80, $lt : 85 } } })

- The results is an array and should contains at least one element matches this query (i . e . 82)

db . topic . find({ tags : { **$all** : [“php”,”ssl”] } })

- The tags field value is an array and should contain ssl, php elements

Element operators



Name	Description
`$exists`	Matches documents that have the specified field.
`$type`	Selects documents if a field is of the specified type.

Element operators

Example :

```
db.inventory.find( {qty: {$exists: true, $nin: [ 5, 15 ] } })
```

- This query will select all documents in the inventory collection where the qty field exists and its value does not equal 5 or 15.

```
db.topic.find( {tags:{ $type: 2 }})
```

- This will list all documents containing a tags field that is either a string or an array holding at least one string.

Update Operation

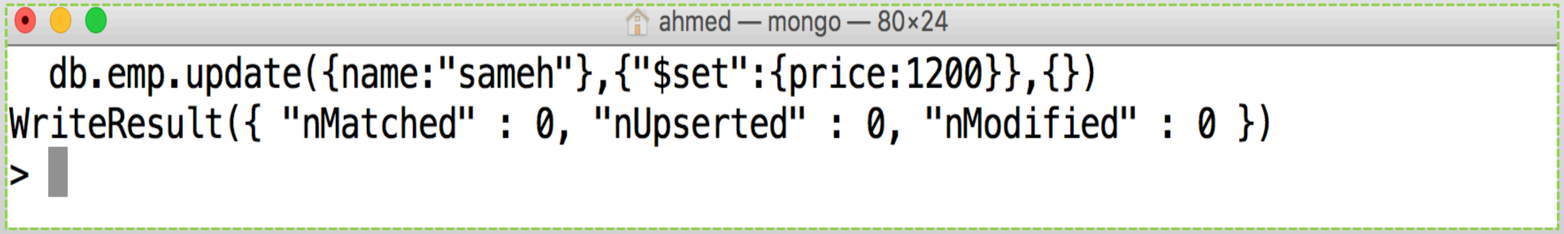
MongoDB provides the update() method to update the documents of a collection .

The method accepts as its parameters :

1- an update conditions document to match the documents to update,

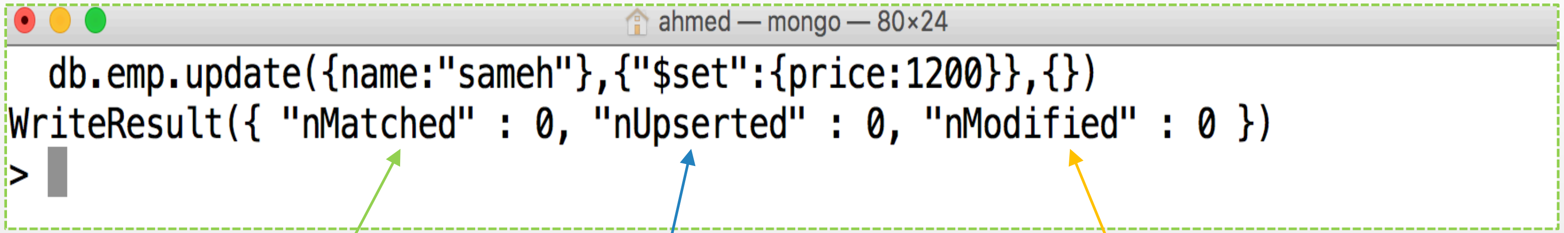
2-an update document to specify the modification to perform, and

3-an options document



```
  db.emp.update({name:"sameh"},{"$set":{price:1200}},{})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
>
```

Update Result



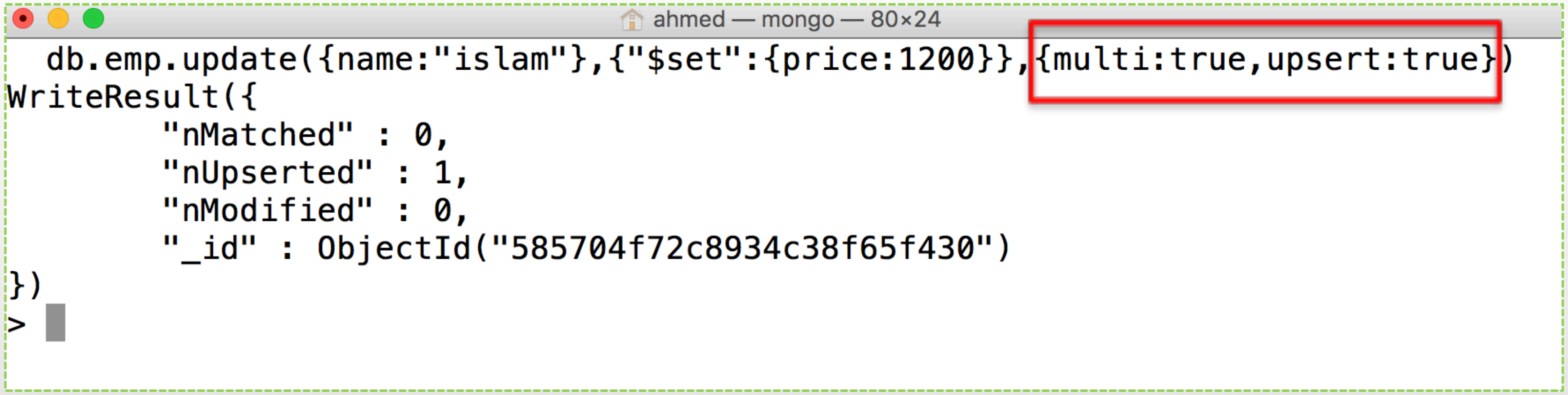
```
  db.emp.update({name:"sameh"},{"$set":{price:1200}},{})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> 
```

Number of documents matched

Number of documents that were created

Number of documents modified

Multi & Upsert

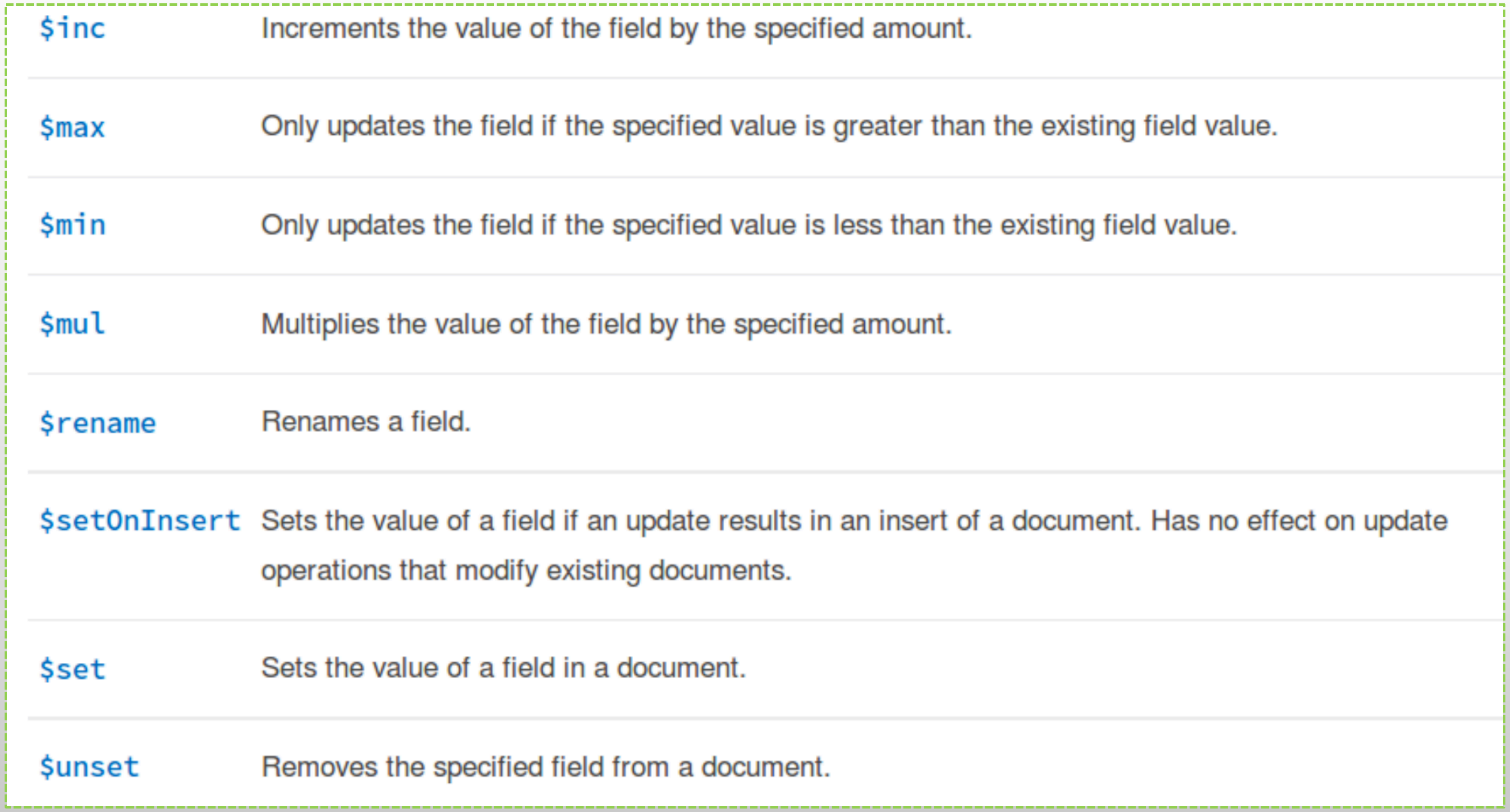


```
  db.emp.update({name:"islam"},{"$set":{price:1200}},{multi:true,upsert:true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("585704f72c8934c38f65f430")
})
> 
```

When multi is true, the update modifies all matching documents

when upsert is true , If the field doesn’t exist, it gets created with the value
Creates a document using the values from the query and update parameter

Update Operators



`$inc`	Increments the value of the field by the specified amount.
`$max`	Only updates the field if the specified value is greater than the existing field value.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$mul`	Multiplies the value of the field by the specified amount.
`$rename`	Renames a field.
`$setOnInsert`	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
`$set`	Sets the value of a field in a document.
`$unset`	Removes the specified field from a document.

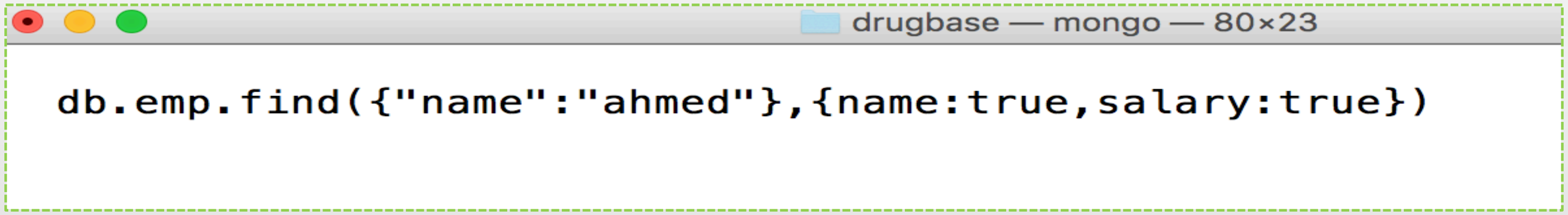
Update Operators For Array

- $pop
- $push
- $addToSet
- $pull
- “$” placeholder operator

Projection

Projection : Only retrieve what 's needed

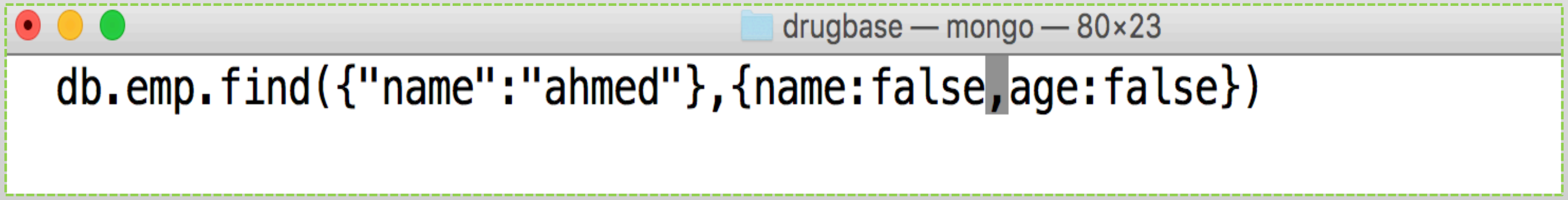
find() takes a second parameter called a "projection" that we can use to specify the exact fields we want back by setting their value to true .



drugbase — mongo — 80×23

```
db.emp.find({"name":"ahmed"},{name:true,salary:true})
```

When selecting fields, all other fields but the __id are automatically set to false



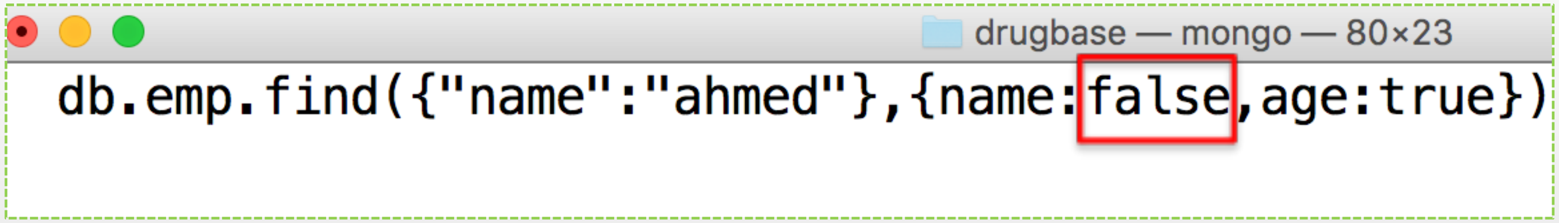
drugbase — mongo — 80×23

```
db.emp.find({"name":"ahmed"},{name:false,age:false})
```

When excluding fields, all fields but those set to false are defaulted to true

Projection

Projection : Only retrieve what ’s needed



```
db.emp.find({"name":"ahmed"},{name:false,age:true})
```

“$err” : "Can't canonicalize query : BadValue Projection cannot have a mix of inclusion and exclusion **except _id** . "

Cursor

Whenever we search for documents, an object is returned from the find method called a "cursor object."

```
db.emp.find({"name": "AHMED"})
result
{"__id": ObjectId(...), ... }
{"__id": ObjectId(...), ... }
{"__id": ObjectId(...), ... }
```

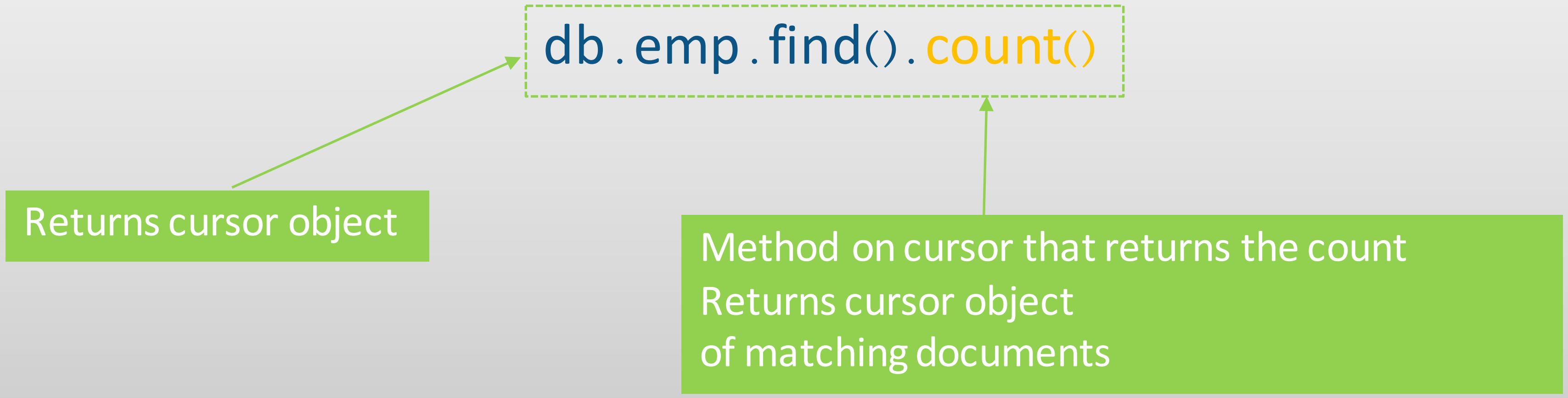
By default, the first 20 documents are printed out

type "it" for more We'll continue being prompted until no documents are left

Cursor Methods

Since the cursor is actually an **object**, we can chain methods on it .

Cursor methods always come after find() since it returns the cursor object .



```
db . emp . find() . count()
```

Cursor Methods – Sort

We can use the ***sort()*** cursor method to sort documents .

Cursor methods always come after find() since it returns the cursor object .

```
db . emp . find() . sort({“salary” : 1})
```

Returns cursor object

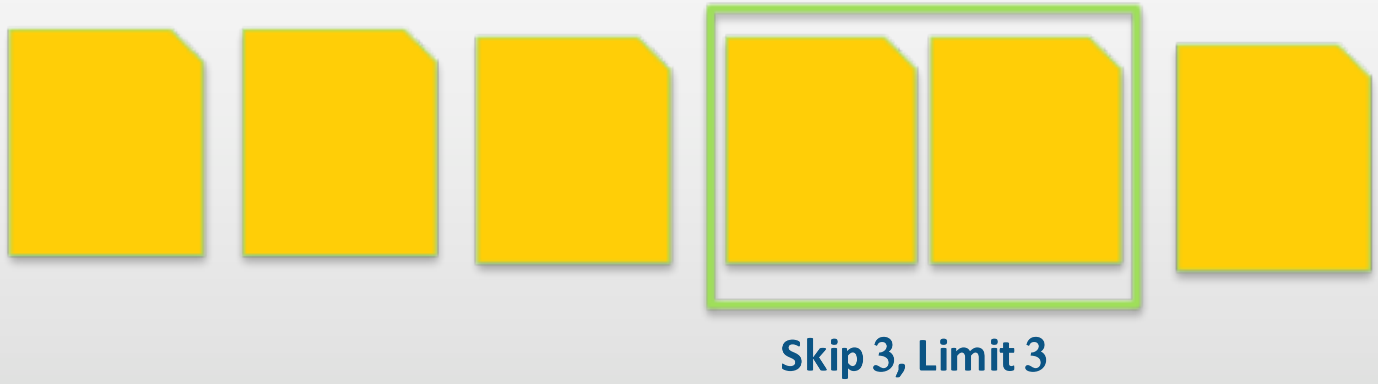
Field to Sort “salary”

–1 to order descending

1 to order ascending

Cursor Methods – Pagination

We can implement basic pagination by **limiting** and **skipping** over documents.



```
db.emp.find().skip(3).limit(2)
```

Aggregation

"Aggregate" is a fancy word for combining data.

Time for an audit, We need to know how many employees we have per dep.

We could manually pull all the data and count everything, but it's better to have MongoDB handle that for us.

The aggregation framework allows for advanced computations.

Using Aggregation Framework

This is known as the “group key” and is required

```
db.emp.aggregate([{“$group”: {“__id”: “$dep__id”}}])
```

Stage operator that ’s used to data by any field we specify

Go within an array

Field names that begin with a “$” are called “field paths” and are links to a field in a document

Using Accumulators

Anything specified after the group key is considered an accumulator. Accumulators take a single expression and compute the expression for grouped documents.

```
db.emp.aggregate([{"$group":{"_id": "$dep_id","total":{"$sum":1}}}])
```

Accumulator

Will add 1 for each matching document

Result :

```
{"_id": "6", "total": 2},
{"_id": "4", "total": 3}
```

Field Paths Vs . Operators

When values begin with a “$”, they represent **field paths** that point to the **value**

When fields begin with a “$”, they are **operators** **that perform a task**

```
db.emp.aggregate([{“$group”: {“__id”: “$dep__id”,”gSalary”:{“$sum”: $salary}}}] )
```

field path

Result :

```
{"__id": “6”, ”gSalary": 10000},
{"__id": “4”, ”gSalary": 7500}
```

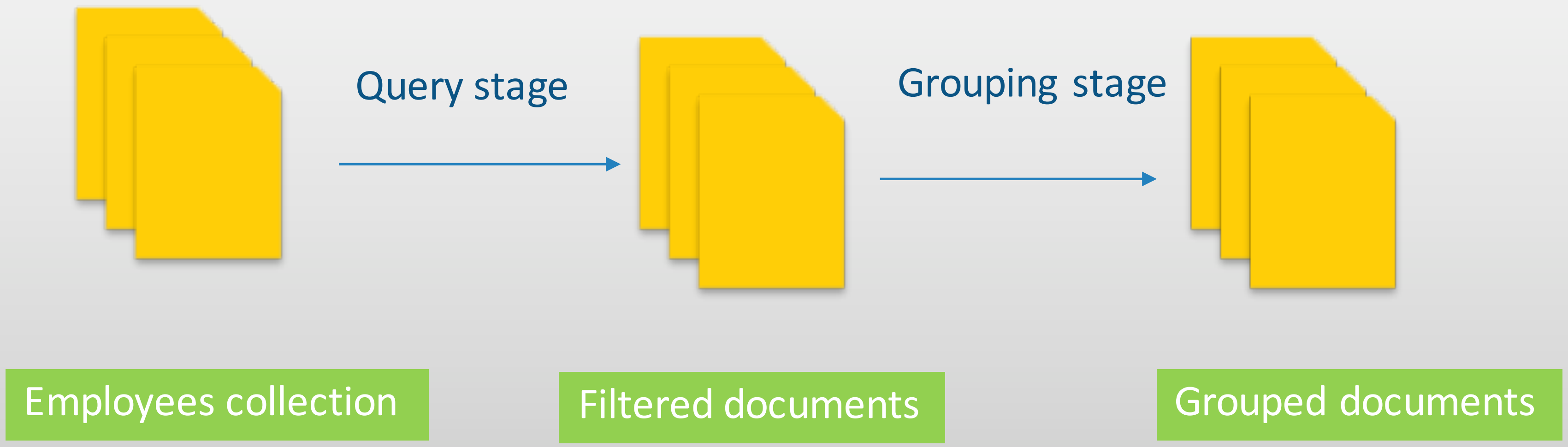
Sums the salary values for employee in their department

Other Accumulators

- $avg
- $min
- $max

Aggregation Pipeline

The aggregate method acts like a pipeline, where we can pass data through many stages in order to change it along the way.



Aggregation Pipeline

Example :

1) Query for employee with a salary less than 1500

2) Group employees by department and average their salaries

3) Sort the results by salary average

4) Limit results to only 3 department

Aggregation Pipeline

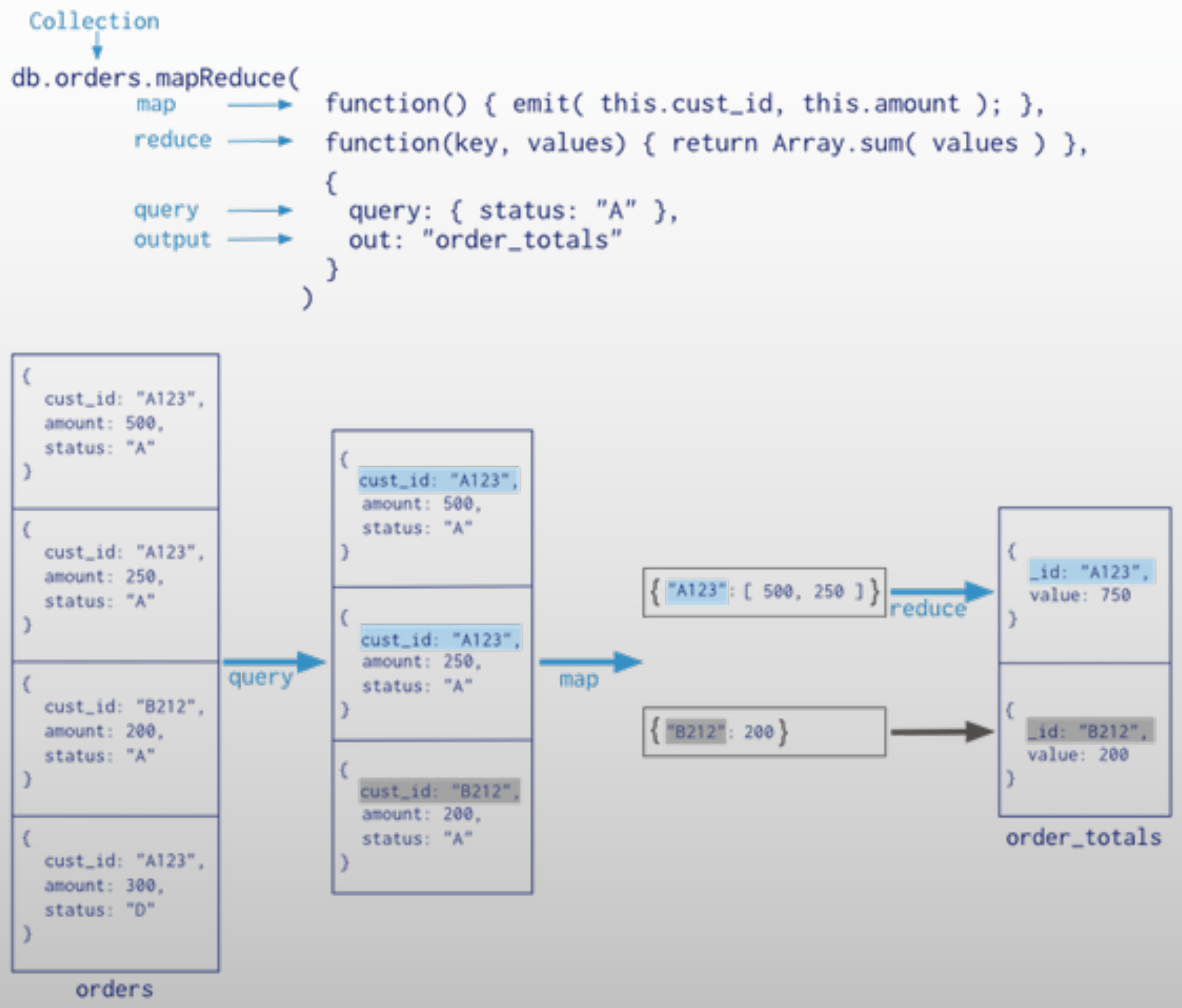
```
db.emp.aggregate([
{"$match": {"salary": {"$lt": 1500}}},
{"$project": {"__id": false, "dep__id": true, "salary": true}},
{"$group": {"__id": "$dep__id","avgS": {"$avg": "$salary"}}},
{"$sort": {"avgS": -1}},
{"$limit": 3}
 ])
```

Map – Reduce

Map–reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results

In this map–reduce operation, MongoDB applies the map phase to each input document The map function emits key–value pairs. For those keys that have multiple values, MongoDB applies the reduce phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection. Optionally, the output of the reduce function may pass through a finalize function to further condense or process the results of the aggregation.

Map – Reduce



```
   Collection
      ↓
db.orders.mapReduce(
           map    ──→   function() { emit( this.cust_id, this.amount ); },
           reduce ──→   function(key, values) { return Array.sum( values ) },
                        {
           query  ──→     query: { status: "A" },
           output ──→     out: "order_totals"
                        }
                      )
```

{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [500, 250] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals

Thank You

E-mail Address :

ahmedcs2012@gmail.com