
mongoDB

# Lecture 2

# Agenda

- Update and upsert
- The limit( ) method
- The sort( ) method
- Projection
- Cursor
- Mongo indexes
- Export and Import
- Aggregation
- Lab 2
- Report

# Update and upsert

**MongoDB provides the update() method to update the documents of a collection. The method accepts as its parameters:**

- **an update conditions document to match the documents to update,**
- **an update document to specify the modification to perform, and**
- **an options document.**

**To specify the update condition, use the same structure and syntax as the query conditions.**

# Update and upsert

**MongoDB provides the update() method to update the documents of a collection. The method accepts as its parameters:**

- **an update conditions document to match the documents to update,**
- **an update document to specify the modification to perform, and**
- **an options document.**

**To specify the update condition, use the same structure and syntax as the query conditions.**

**db.collection.update({where condition },{ update statement }, {optional })**

# Update and upsert

## 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 and upsert

- db.students.update( {“age”:15 },
  { $set: {“name”:”Ali”} },
  {multi: 1})

- db.students.update( {“age”:15 },
  { $rename: {“name”:”firstName”} },
  {multi: true})

# Update and upsert

```
db.inventory.update(
    { item: "MNO2" },
          {
          $set: {
               category: "apparel",
               details: { model: "14Q3", manufacturer: "XYZ Company" }
          },
    }
)
```

# Update and upsert

- db.students.update( {“age”:15 },
  { $inc: {“age”:20} },
  {multi: 1})

- db.students.update( {“age”:15 },
  { $mul: {“age”:20} },
  {multi: 1})

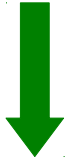


mongoDB

# Update and upsert

```
for(var i=0 ; i<20 ; i++){db.emp.insert({“salary”:i*5}) }
```

- db.emp.update( { },
  { $min: {“salary”:70} },
  {multi: 1})



- db.emp.update( { },
  { $max: {“salary”:50} },
  {multi: 1})

# Update and upsert

- db.emp.update( {salary:60 },
  { $set: {“name”:”Ali”} },
  {multi: 1 , upsert : 1 })

- db.emp.update( {salary: { $gt: 100 } },
  { $setOnInsert: {“name”:”Ali”} },
  {multi: 1 , upsert : 1 })

# Update and upsert

- db.new.update( {salary:{$gt:700 }},
  { $set: {“name”:”Ali”} },
  {multi: 1 , upsert : 1 })

- db.emp.update( {salary: { $gt: 100 } },
  { $setOnInsert: {“name”:”Ali”} },
  {multi: 1 , upsert : 1 })

# The limit( ) method

- To limit the records in MongoDB, you need to use limit() method. The method accepts one number of documents that you want to be displayed.

- db.collection_name.find( ).limit(number)

  db.students.find( ).limit(2)

# The sort( ) method

- To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

- db.collection_name.find( ).sort(number)

  db.students.find( ).sort({ “_id”: -1 })

# 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.**

```
db.students.find({"_id":5},{"name":1, "_id":1})

db.students.find({"_id":5},{"name":1})

db.students.find({"_id":5},{"name":1, "_id":0})

db.students.find({"_id":5},{"name":1, "age":0})
```

**Error**

# 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**



mongoDB

# 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.students.find().count()
```

**Returns cursor object**

**Method on cursor that returns the count**
**Returns cursor object**
**of matching documents**

# 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.students.find().sort({“salary”:1})
```

**Returns cursor object**

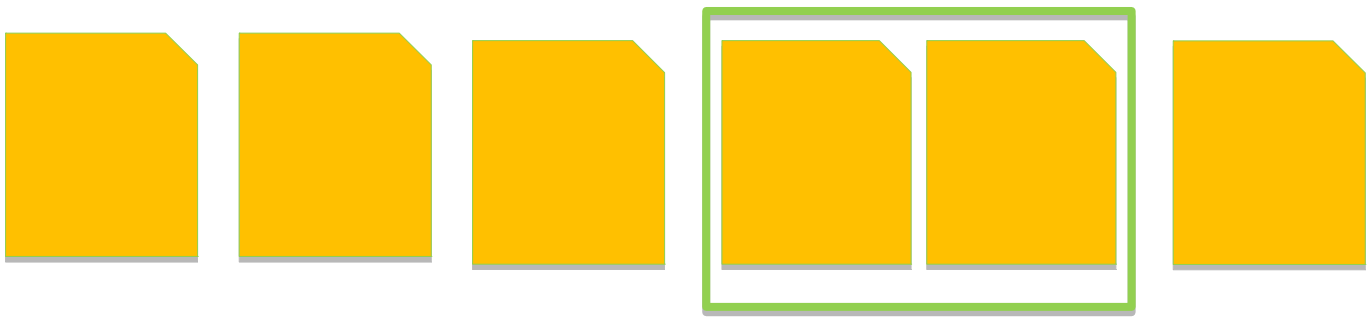
Field to Sort “salary”

-1 to order descending

1 to order ascending

# Cursor Methods - Pagination

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



**Skip 3, Limit 2**

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

# Mongo indexes

**Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect**.

The index stores the **value of a specific field or set of fields**, ordered by the value of the field.

MongoDB defines indexes at the collection level and supports indexes on any **field or sub-field** of the documents in a MongoDB collection.

# Mongo indexes

- **Before Index**

  **db.students.find({},{name:1, _id:0}).skip(6).limit(2).explain()**

- **Create Index**

  **db.students2.createIndex({name:1})**

  db.collection_name.**createIndex**( <key and index type specification>, <options> )

# Mongo indexes

- **Create Index**

```
db.students2.createIndex({name:1})
    {
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
    }
```

- db.students2.find({"name":"Ahmed"}).explain()

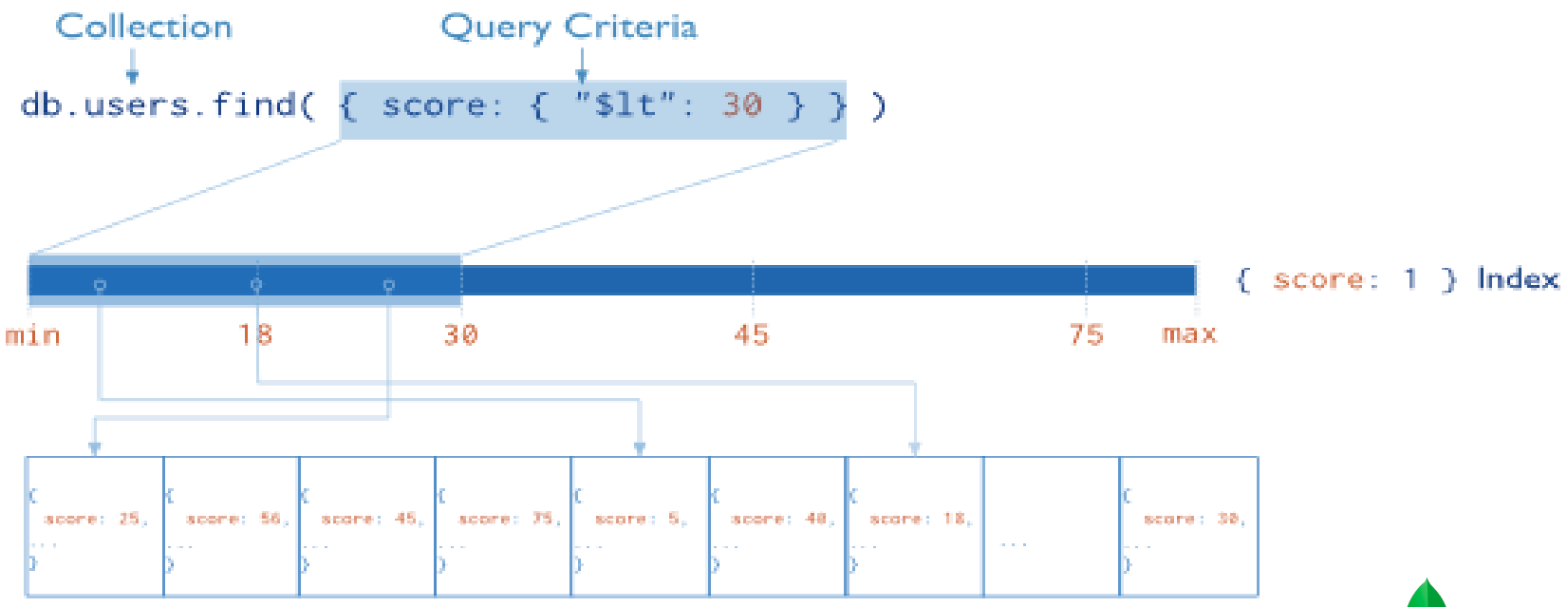
# Mongo indexes

**Default _id Index**

**MongoDB creates a unique index on the _id field during the creation of a collection. The _id index prevents clients from inserting two documents with the same value for the _id field. You cannot drop this index on the _id field.**

# Mongo indexes

## db.users.createIndex({ score: 1 })



# Mongo indexes

## Show Collection Indexes

**db.collection.getIndexes()**

**db.students2.getIndexes()**

```
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "iti.students2"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "name_1",
                "ns" : "iti.students2"
        }
]
```



mongoDB

# Modify and Drop indexes

- db.students.dropIndex({key:”index_key”})
- db.students.dropIndexes()
- You can modify index using createIndex but there are some cases you will must remove index and rebuild it

# Index Types

```
db.records.insert ( { “_id” :1 ,
                      “score”:1034,
                      “location” :{ state: “NY”, city: “New York”}})
```

## 1) Single Field

```
db.records.createIndex({ score : 1 })
db.records.find({ score : { $gt : 10 } })
```

## 2) Embedded Field

```
db.records.createIndex({ location.state : 1 })
db.records.find({ “location.state” : “NY” })
```

# Index Types

```
db.records.insert ( { “_id” :1 ,
                      “score”:1034,
                      “location” :{ state: “NY”, city: “New York”}})
```

## 3) Embedded Document

```
db.records.createIndex({ location : 1 })
```

## 4) Compound indexes

```
db.records.createIndex({location:-1,score:1})
```

# Export and Import

## Export :

```
$ mongodump --db Inventory --collection products  --out  dir_path
```

## Import :

```
$mongorestore --db Inventory  path_to_dump_directory
```

# Aggregation

- Aggregation operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- MongoDB provides three ways to perform aggregation:
  - The aggregation pipeline.
  - The map-reduce function.
  - Single purpose aggregation methods.



mongoDB

# Aggregation Pipeline

- The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines.
- Documents enter a multi-stage pipeline that transforms the documents into aggregated results.

# Aggregation Pipeline

**Example:**

**Restaurant Database has orders collections:**

**{ _id:1 , cust_id: “abc1” , status: “A” , amount: 50}**

**{ _id:2 , cust_id: “xyz1” , status: “A” , amount: 100}**

**{ _id:3 , cust_id: “xyz1” , status: “D” , amount: 25}**

**{ _id:4 , cust_id: “xyz1” , status: “D” , amount: 125}**

**{ _id:5 , cust_id: “abc1” , status: “A” , amount: 25}**

**Total amount for each customer that has status A?**



mongoDB

# Aggregation Pipeline

1) Select documents with status equal to “A”,

{ _id:1 , cust_id: “abc1” , status: “A” , amount: 50}

{ _id:2 , cust_id: “xyz1” , status: “A” , amount: 100}

{ _id:5 , cust_id: “abc1” , status: “A” , amount: 25}

2) Group the matching documents by the cust_id field,

{ _id: “abc1” , status: “A” , amount: 50 , amount: 25}

{ _id: “xyz1” , status: “A” , amount: 100}

# Aggregation Pipeline

3) Calculate the total for each cust_id field from the amount field.

4) Sort the results by the total field in descending order.

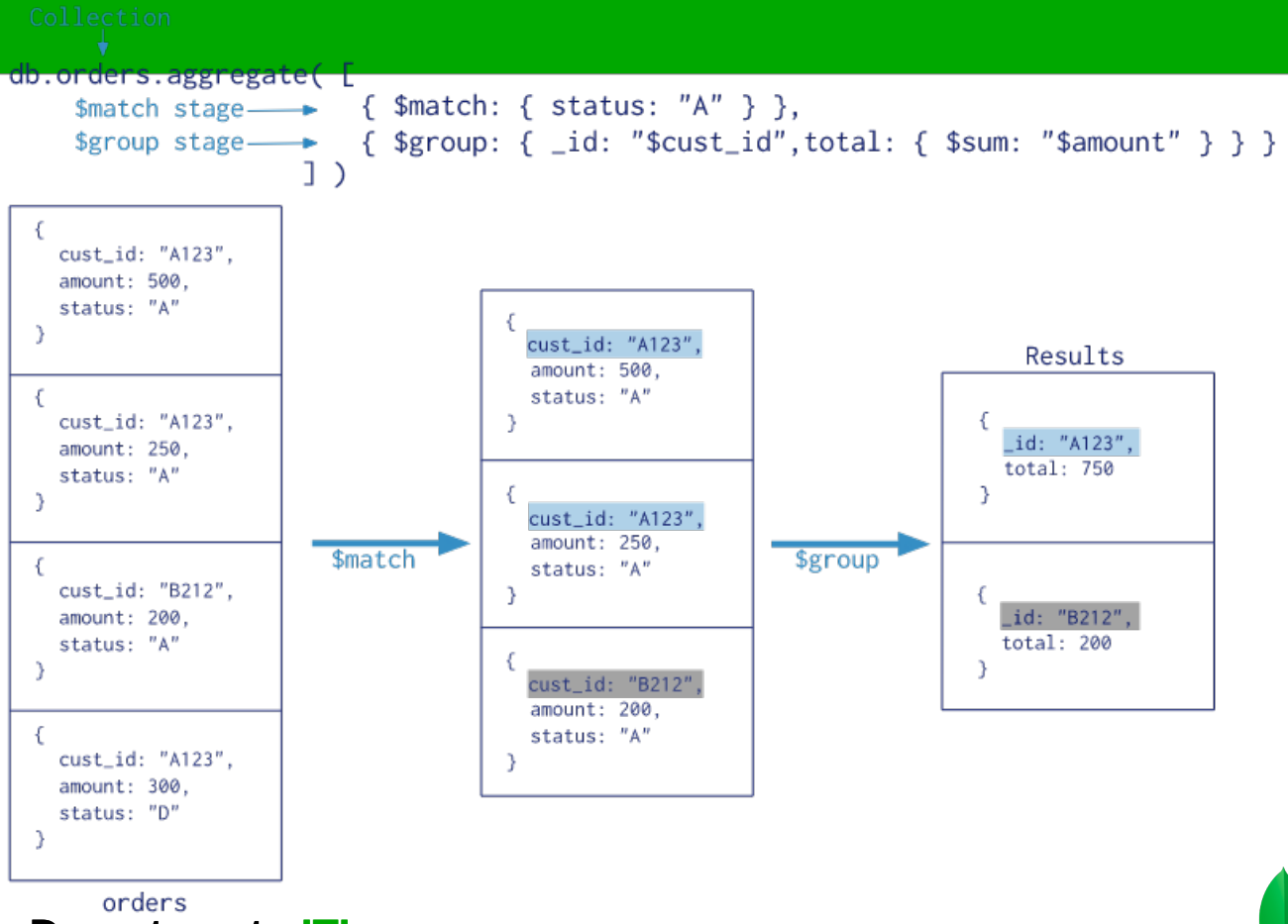
{ _id: “xyz1” , status: “A” , total: 100}

{ _id: “abc1” , status: “A” , total: 75}

# Aggregation Pipeline

```
db.orders.aggregate ( [
                        { $match: { status: “A”} },
                        { $group: {_id : “$cust_id”,
                                   total: { $sum: “$amount” } } },
                        { $sort: { total : -1 } } ] )
```

# Aggregation Pipeline



# Aggregation Pipeline

**Example:**

**Library Database has tutorials collection**

**{ _id:1 , title: “Mongo” , by_user: “user1” , likes: 100}**

**{ _id:2 , title: “NoSQL” , by_user: “user1” , likes: 10}**

**{ _id:3 , title: “Neo4j” , by_user: “user2” , likes: 70}**

**Display a list starting how many tutorials are written by each user.**

# Aggregation Pipeline

```
db.orders.aggregate ( [
                    { $group: {_id : “$by_user”,
                              num_tutorial: { $sum: 1 } } },
                  ] )
```

**We have grouped documents by field by_user and on each occurrence of by_user previous value of sum is incremented.**

**{ _id: “user1” , num_tutorial : 2}**

**{ _id: “user2” , num_tutorial : 1}**

# Aggregation Pipeline

## Aggregation Operators

| | |
|---|---|
| `$avg` | Returns an average for each group. Ignores non-numeric values. |
| `$first` | Returns a value from the first document for each group. Order is only defined if the documents are in a defined order. |
| `$last` | Returns a value from the last document for each group. Order is only defined if the documents are in a defined order. |
| `$max` | Returns the highest expression value for each group. |
| `$min` | Returns the lowest expression value for each group. |
| `$push` | Returns an array of expression values for each group. |
| `$sum` | Returns a sum for each group. Ignores non-numeric values. |

# Aggregation Pipeline

## Return all city with a population greater than 10 million

```
db.orders.aggregate ( [
    { $group: {_id : "$city",
               totalPop: { $sum: "$pop" } } },
    { $match:
        { totalPop : { $gte: 10*1000*1000} } }
    ] )
```

```
{"id":1
"city":"NewYork"
"state":"NX",
"pop":5574,
}
```



mongoDB

# Aggregation Pipeline

## $sort

**Sorts all input documents and returns them to the pipeline in sorted order.**

## $match

**Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.**

# Aggregation Pipeline

## $group

- Groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.
- The output documents contain an _id field which contains the distinct group by key.
- The output documents can also contain computed fields that hold the values of some accumulator expression grouped by the $group‘s _id field.
- $group does not order its output documents.



mongoDB

# Aggregation Pipeline

**$out**

**Takes the documents returned by the aggregation pipeline and writes them to a specified collection.**

**The $out operator must be the last stage in the pipeline.**

**$skip**

**Skips over the specified number of documents that pass into the stage and passes the remaining documents to the next stage in the pipeline.**

# Aggregation Pipeline

## $limit

**Limits the number of documents passed to the next stage in the pipeline.**

## $project ?

# Lab 2

**mongorestore --db Inventory path_to_Inventory_folder.**

**1) Display products which available in 4 stocks at the same time.**

**2) Increase all products by 500 EGP.**

**3) Replace stock #40 with #60 in all products.**

**4) Remove stock 70 from all products.**

**5) Display only product name and vendor phone number.**

**6) Rename vendor filed to shop.**

**7)Insert new document with name vodafone , category, price and quantity fields if the price is less than 100.**

**8)How to create user?**

# Lab 2

**9) Display number of products per category.**

**10) Display max category products price.**

**11) Display the most expensive product.**

**12) Import Blog Database using this command in terminal**

**mongorestore --db Blog path_to_Blog_folder**

**13) Create text index for post content field.**

**14)Retrieve all posts which have “hello” word**

# Lab 3

15)Retrieve all posts which have not “hello” word.

16)Retrieve all post which have “hello world ” exactly.

17) Export Blog database.

18)Drop blog database.

19)What is the maximum number of Indexes per collection allowd?

20)What is the maximum size of an Index key?

21)What is collation in indexes.

22)Which of the following are stages of aggregation:

( $sort , $skip , $drop , $limit )

23)Mention the command to list all the indexes on a particular collection.

# Report

- **Group 1**

  **Add users.**

  **Manage users and roles.**

  **change your password.**
- **Group 2**

  **Text Search.**

  **BulkWrite Operations.**
- **Group 3**

  **Data Models.**
- **Group 4**

  **Replication.**
- **Group 5**

  **Sharding.**

  **Geospatial Queries.**
- **Group 6**

  **Map-Reduce.**