Prof R. Madana Mohana

BIG DATA ANALYTICS

MONGODB

Aggrgation

https://www.youtube.com/c/RASINENIMADANAMOHANA

MongoDB: Aggregation

MongoDB: *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.
- In **SQL count(*)** and with **group by** is an equivalent of **MongoDB aggregation**.

MongoDB: *Aggregation*

The aggregate() Method:

For the aggregation in MongoDB, we should use **aggregate()** method.

The aggregate() Method: *Syntax*

Basic syntax of **aggregate()** method is as follows:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

MongoDB: *Aggregation*

The aggregate() Method: *Example*

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
mydb       0.000GB
sampleDB   0.000GB
test       0.000GB
> use test
switched to db test
```

MongoDB: *Aggregation*

The aggregate() Method: *Example*

```
> show collections
Employee
EmployeeDetails
Softwareemployee
bdacollection
blogspost
courses
mongodbcollection
mycollection
softwareemployee
```

MongoDB: *Aggregation*

The aggregate() Method:

Example

```
> db.Employee.find().pretty()
```

Command Prompt - mongo

```
> db.Employee.find().pretty()
{
        "_id" : ObjectId("634bfe4112c650f5e845ddc9"),
        "First_Name" : "Rasineni Madana",
        "Last_Name" : "Mohana",
        "Date_Of_Birth" : "1980-12-26",
        "e_mail" : "rmmnaidu@yahoo.co.in",
        "phone" : "0123456789",
        "Age" : "35"
}
{
        "_id" : ObjectId("634c03ef12c650f5e845ddca"),
        "First_Name" : "Rasineni Madana",
        "Last_Name" : "Mohana",
        "Date_Of_Birth" : "1981-06-15",
        "e_mail" : "rmmnaidu@gmail.com",
        "phone" : "0123456789"
}
{
        "_id" : ObjectId("634c03ef12c650f5e845ddcb"),
        "First_Name" : "Madan",
        "Last_Name" : "Mohan",
        "Date_Of_Birth" : "1980-12-26",
        "e_mail" : "rmmnaidu@gmail.com",
        "phone" : "9876543210",
        "Age" : "35"
}
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

```
> db.Employee.aggregate([{$group : {_id :
"$First_Name", num_persons: {$sum : 1}}}])
{ "_id" : "Rasineni Madana", "num_persons" : 2 }
{ "_id" : "Madan", "num_persons" : 1 }
{ "_id" : "Madana Mohana", "num_persons" : 1 }
```

SQL equivalent query for the a*bove use case* will be

```
select First_Name, count(*) from Employee group by
First_Name
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

```
> db.Employee.aggregate([{$group : {_id : "$Age",
num_persons: {$sum : 1}}}])
{ "_id" : "35", "num_persons" : 2 }
{ "_id" : null, "num_persons" : 2 }
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

```
> db.Employee.aggregate([{$group : {_id : "$Age",
num_persons: {$avg : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

```
> db.Employee.aggregate([{$group : {_id : "$Age",
num_persons: {$min : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

```
> db.Employee.aggregate([{$group : {_id : "$Age",
num_persons: {$max : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
```

MongoDB: *Aggrgation*

The aggregate() Method: *Example*

Command Prompt - mongo

```
> db.Employee.aggregate([{$group : {_id : "$First_Name", num_persons: {$sum : 1}}}])
{ "_id" : "Rasineni Madana", "num_persons" : 2 }
{ "_id" : "Madan", "num_persons" : 1 }
{ "_id" : "Madana Mohana", "num_persons" : 1 }
> db.Employee.aggregate([{$group : {_id : "$Age", num_persons: {$sum : 1}}}])
{ "_id" : "35", "num_persons" : 2 }
{ "_id" : null, "num_persons" : 2 }
> db.Employee.aggregate([{$group : {_id : "$Age", num_persons: {$avg : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
> db.Employee.aggregate([{$group : {_id : "$Age", num_persons: {$min : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
> db.Employee.aggregate([{$group : {_id : "$Age", num_persons: {$max : 1}}}])
{ "_id" : "35", "num_persons" : 1 }
{ "_id" : null, "num_persons" : 1 }
```

MongoDB: *Aggregation*

Following is a list of available **aggregation** expressions:

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	`db.Employee.aggregate([{$group : {_id : "$First_Name", num_persons: {$sum : "$Age"}}}])`
$avg	Calculates the average of all given values from all documents in the collection.	`db.Employee.aggregate([{$group : {_id : "$First_Name", num_persons: {$avg : "$Age"}}}])`
$min	Gets the minimum of the corresponding values from all documents in the collection.	`db.Employee.aggregate([{$group : {_id : "$First_Name", num_persons: {$min : "$Age"}}}])`
$max	Gets the maximum of the corresponding values from all documents in the collection.	`db.Employee.aggregate([{$group : {_id : "$First_Name", num_persons: {$max : "$Age"}}}])`

MongoDB: *Aggregation*

Following is a list of available **aggregation** expressions:

Expression	Description	Example
$push	Inserts the value to an array in the resulting document.	`db.Employee.aggregate([{$group : {_id : "$First_Name", person_age: {$push: "$Age"}}}])`
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	`db.Employee.aggregate([{$group : {_id : "$First_Name", new_person: {$addToSet: "$First_Name"}}}])`
$first	Gets the first document from the source documents according to the grouping. Typically, this makes only sense together with some previously applied "$sort"-stage.	`db.Employee.aggregate([{$group : {_id : "$First_Name", first_person: {$first : "$First_Name"}}}])`
$last	Gets the last document from the source documents according to the grouping. Typically, this makes only sense together with some previously applied "$sort"-stage.	`db.Employee.aggregate([{$group : {_id : "$First_Name", last_person: {$last : "$First_Name"}}}])`

MongoDB: *Aggregation*

Pipeline Concept:

- In **UNIX** command, **shell pipeline** means the possibility to execute an operation on some input and use the output as the input for the next command and so on.
- **MongoDB** also supports same concept in **aggregation** framework.

MongoDB: *Aggregation*

Pipeline Concept:

- There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting **JSON** document at the end of the pipeline).
- This can then in turn be used for the next stage and so on.

MongoDB: *Aggregation*

Pipeline Concept:

Following are the *possible stages* in **aggregation** framework:

- **$project:** Used to select some specific fields from a collection.
- **$match:** This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- **$group:** This does the actual aggregation as discussed above.

MongoDB: *Aggregation*

Pipeline Concept:

Following are the *possible stages* in **aggregation** framework:

- **$sort:** Sorts the documents.
- **$skip:** With this, it is possible to skip forward in the list of documents for a given amount of documents.
- **$limit:** This limits the amount of documents to look at, by the given number starting from the current positions.

MongoDB: *Aggregation*

Pipeline Concept:

Following are the *possible stages* in **aggregation** framework:

- **$unwind**: This is used to unwind (undo) document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.