executionStats – this mode includes all the information provided by the queryPlanner, plus the statistics. Statistics include details such as the number of documents examined and returned, the execution time in milliseconds, and so on.

db.collection.find({query}).explain("executionStats")

Q. Is any default index for our collection?

For every collection default index is avaialble, and it is based on _id field.

db.employees.getIndexes()
[{ "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" }]

Aggregation Framework:

```
db.employees.insertOne({eno:100,ename:"Sunny",esal:1000,eaddr:"Mumbai"})
db.employees.insertOne({eno:200,ename:"Bunny",esal:2000,eaddr:"Hyderabad"})
db.employees.insertOne({eno:300,ename:"Chinny",esal:3000,eaddr:"Hyderabad"})
db.employees.insertOne({eno:400,ename:"Vinny",esal:4000,eaddr:"Mumbai"})
db.employees.insertOne({eno:500,ename:"Pinny",esal:5000,eaddr:"Chennai"})
```

```
db.employees.insertOne({eno:600,ename:"Zinny",esal:6000,eaddr:"Chennai"})
db.employees.insertOne({eno:700,ename:"Yinny",esal:7000,eaddr:"Hyderabad"})
```

db.employees.find().pretty()
db.employees.find({eaddr:"Hyderabad"}).pretty()

Total salary of all employees irrespective of city
Total salary of all employees city wise
Average salary of all employees city wise
minimum salary of employees city wise
maximum salary of employees city wise
What is the min salary of all employees
What is the max salary of all employees

etc

Such type of requirements won't be fullfilled by find() methods.

find() methods will provide existing data as it is. If we want to process documents and to provide results in our customized format then we should go for aggregation framework.

We can implement aggregation by using aggregate() method.
aggregate() method is more powerful than find() method.

aggregate() vs find():

---

aggregate() method can perform some processing and provide results in our customized format.

But find() method will always provide data as it is without performing any processing and in the existing format only.

eg-1: To find total salary of all employees?

db.employees.aggregate([

```
{ $group: {_id:null,totalsalary:{$sum:"$esal"}}}
])

{ "_id" : null, "totalsalary" : 28000 }
```

Note:

$group stage:

---

1. It is the most important stage.
2. It can be used to group documents based on required fields. It merges different documents into new documents.
   eg: group all documents city wise
   group all documents department wise
3. The first parameter in $group stage is always _id.
4. We should use _id to specify field based on which we have to perform grouping. ie _id defines by which fields we can group.
5. If we want to process all records then we have to provide null value to _id field.
6. $group is exactly same as group by phrase in relational databases.

Accumulator operators:

These operators can be used for accumulation purpose.

The following various accumulator operators:

1. $sum: Returns a sum of numerical values. Ignores non-numeric values.
2. $avg: Returns an average of numerical values. Ignores non-numeric values.
3. $max: Returns the highest expression value for each group.
4. $min: Returns the lowest expression value for each group.

etc

eg-2: To find average salary of all employees?

```
db.employees.aggregate([
{ $group: {_id:null,averagesalary:{$avg:"$esal"}}}
])

{ "_id" : null, "averagesalary" : 4000 }
```

eg-3: To find max salary of all employees?

```
db.employees.aggregate([
{ $group: {_id:null,maxsalary:{$max:"$esal"}}}
])

{ "_id" : null, "maxsalary" : 7000 }
```

eg-4: To find min salary of all employees?

```
db.employees.aggregate([
{ $group: {_id:null,minsalary:{$min:"$esal"}}}
])

{ "_id" : null, "minsalary" : 1000 }
```

eg-5: To find max salary city wise?

We have to group documents city wise. ie based on eaddr field.

```
db.employees.aggregate([
  {$group: {_id:"$eaddr",maxSalary:{$max:"$esal"}}}
])
```

o/p:

```
{ "_id" : "Mumbai", "maxSalary" : 4000 }
{ "_id" : "Hyderabad", "maxSalary" : 7000 }
{ "_id" : "Chennai", "maxSalary" : 6000 }
```

eg-6: To find city wise total salary?

```
db.employees.aggregate([
  {$group: {_id:"$eaddr",totalSalary:{$sum:"$esal"}}}
])
```

o/p:

```
{ "_id" : "Mumbai", "totalSalary" : 5000 }
{ "_id" : "Hyderabad", "totalSalary" : 12000 }
{ "_id" : "Chennai", "totalSalary" : 11000 }
```

eg-7: To find city wise average salary?

```
db.employees.aggregate([
  {$group: {_id:"$eaddr",averageSalary:{$avg:"$esal"}}}
])
```

o/p:

```
{ "_id" : "Mumbai", "averageSalary" : 2500 }
{ "_id" : "Hyderabad", "averageSalary" : 4000 }
{ "_id" : "Chennai", "averageSalary" : 5500 }
```

eg-8: To find total number of employees?

```
db.employees.aggregate([
  {$group: {_id:null,employeecount:{$sum:1}}}
])
```

For every document add 1 to the employeecount.

o/p: `{ "_id" : null, "employeecount" : 7 }`

eg-9: To find total number of employees city wise?

```
db.employees.aggregate([
  {$group: {_id:"$eaddr",employeecount:{$sum:1}}}
])
```

o/p:

{ "_id" : "Mumbai", "employeecount" : 2 }

{ "_id" : "Hyderabad", "employeecount" : 3 }

{ "_id" : "Chennai", "employeecount" : 2 }

Aggregation Pipeline:

We can define multiple stages in the aggregation and all these stages will form pipeline, which is known as aggregation pipeline.

```
db.collection.aggregate([
{stage-1},
{stage-2},
{stage-3},
{stage-4},
{stage-5}
...
])
```

All these stages will be executed one by one.
The output of previous stage will become input to next stage.

Pipeline will take collection as input and provides aggregated results in our required format.

Note: In Documentation just explore as many stages as possible.
reference-->Operators--->Aggregation Pipeline stages