Week2

Running mongodB commands in Mongodb shell

In this tutorial you will learn how to query mongdb using the mongodb shell. It is important to understand this commands because in our programming exercise we will be using this commands. But for this tutorial you DO NOT need to do any programming just practice these commands. The mongodb command discussed in this tutorial are:

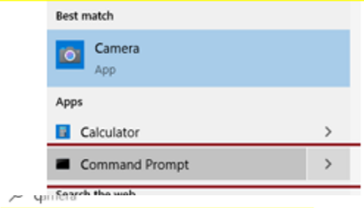
db.collection.insertOne()	Inserts a single document into a collection.
db.collection.insertMany()	db.collection.insertMany() inserts multiple documents into a collection.
db.collection.insert()	db.collection.insert() inserts a single document or multiple documents into a collection.
db.collection.UpdateOne()	Update a single document into a collection.
db.collection.UpdateMany()	Update multiple documents into a collection.
db.collection.deleteMany()	Deletes a single document into a collection.
db.collection.deleteOne()	Deletes multiple documents into a collection.
db.collection.find()	Find documents for a given condition
db.collection.Aggegrate()	Perform pipeline of match and group the data

Opening the Mongodb Shell for running these commands

1. How to open mongodb shell
   a. Open the command prompt

Opening the Mongodb Shell for running these commands

1. How to open mongodb shell
   a. Open the command prompt



b. Go to the directory where you installed mongodb usually it

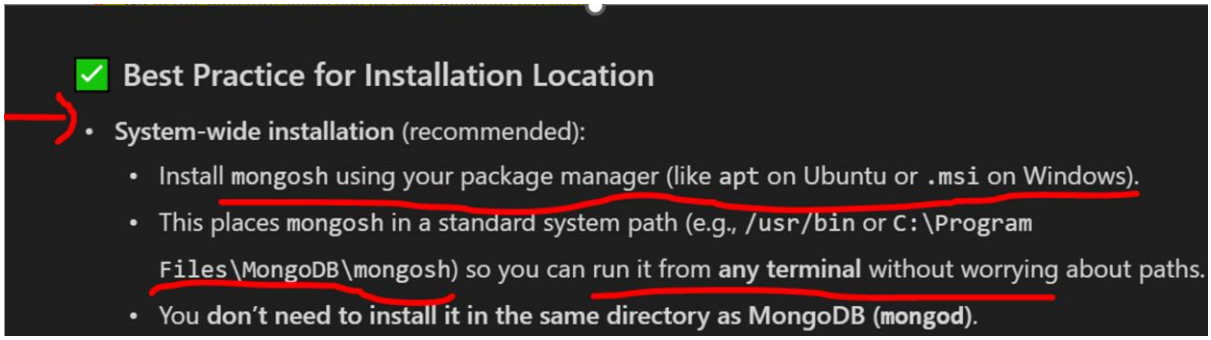
   b. Go to the directory where you installed mongodb usually it is **C:\Program Files\MongoDB\Server\8.2\bin**
   c. On your command prompt write cd **C:\Program Files\MongoDB\Server\8.2\bin**
   d. Write mongo as shown in the figure below:mongod –version , it will show you the installed Mongodb version

```
C:\Program Files\MongoDB\Server\8.2\bin>mongod --version
db version v8.2.0
Build Info: {
    "version": "8.2.0",
    "gitVersion": "13e629eeccd63f00d17568fc4c12b7530fa34b54",
    "modules": [],
    "allocator": "tcmalloc-gperf",
    "environment": {
        "distmod": "windows"
    }
}

C:\Program Files\MongoDB\Server\8.2\bin>
```

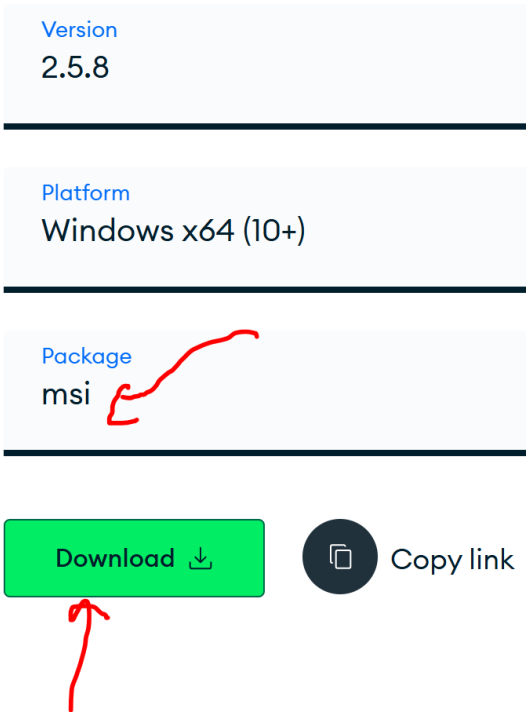
Cong MongoDB is installed on your system. Now we will connect with Mongshell. you are in the mongodb shell you can see the > prompt which is mongodb shell prompt

- **If mongo shell is not installed on your system follow this**



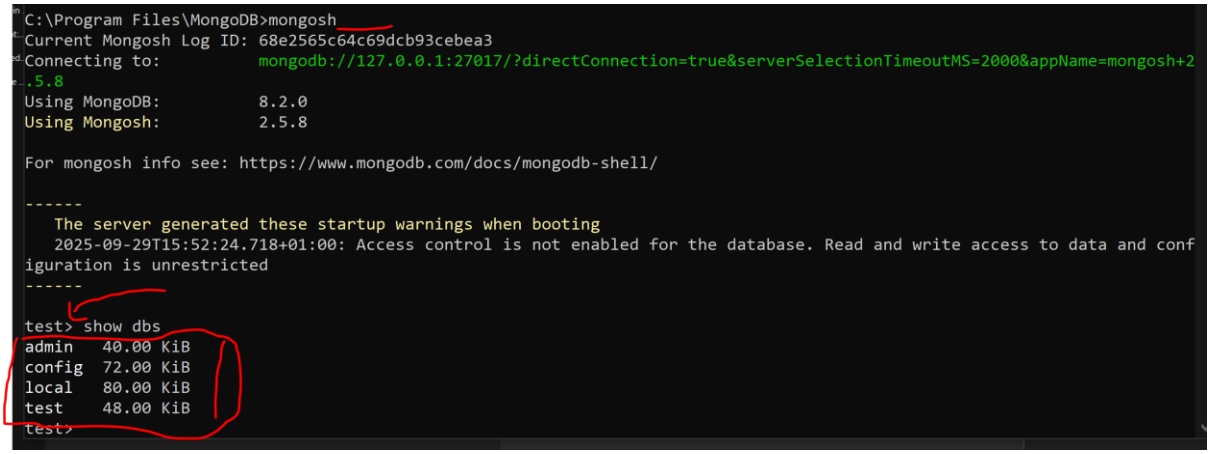
- **.msi is recommended**
- **Visit**

https://www.mongodb.com/try/download/shell



To install MongoDB Shell (mongosh) on Windows:

1. **Download the `.msi` installer** from the official MongoDB website.
2. **Run the installer** and follow the setup instructions.
3. **Choose the installation directory** during setup.
   ✅ *It is recommended to install mongosh in the same directory where MongoDB is installed* — this helps keep all MongoDB tools organized and simplifies environment configuration.



The `mongo` shell provides various help. The following table displays some common help methods and commands:

Help Methods and Commands	Description
`help`	Show help.
`db.help()`	Show help for database methods.
`db.<collection>.help()`	Show help on collection methods. The `<collection>` can be the name of an existing collection or a non-existing collection.
`show dbs`	Print a list of all databases on the server. The operation corresponds to the `listDatabases` command. If the deployment runs with access control, the operation returns different values based on user privileges. See listDatabases Behavior for details.
`use <db>`	Switch current database to `<db>`. The `mongo` shell variable `db` is set to the current database.
`show collections`	Print a list of all collections for current database. TIP **See also:** `show collections`
`show users`	Print a list of users for current database.
`show roles`	Print a list of all roles, both user-defined and built-in, for the current database.
`show profile`	Print the five most recent operations that took 1 millisecond or more. See documentation on the database profiler for more information.
`show databases`	Print a list of all available databases. The operation corresponds to the `listDatabases` command. If the deployment runs with access control, the operation returns different values

- How to show all the databases in your mongodb server
  - On your prompt displayed as > in above figure , write the command **"Show dbs"** it will show name of all the databases in your mongoDB server . On my system when I wrote show dbs on the prompt it displayed following databases.

```
> show dbs
Group3          0.000GB
LabWeek4        0.000GB
People          0.000GB
People_db       0.000GB
Peopledb        0.000GB
admin           0.000GB
config          0.000GB
local           0.000GB
mydatabase      0.000GB
testdatabase    0.000GB
tutorialkart    0.000GB
```

So in my system the name of databases are Group3,Labweek4,People ,People_db etc.

a. How to select a database.

b. **Using the `use` Command in MongoDB Shell**

c. The `use` command is used to select or switch to a specific database in MongoDB. The general syntax is:

```
use [database_name]
```

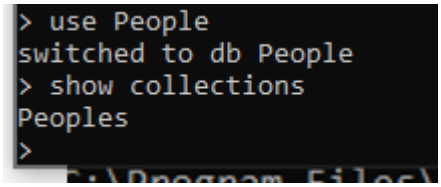
For example, if you want to work with a database named `people`, you would enter:

```
use people
```

```
> use People
switched to db People
```

d. How to list the collections in the selected database?

After you have selected a database using use people , then display all the collection in the opened database using following command show collections or db.getCollectionNames() as shown in the figure below:



Inserting one document in people collection

- Make sure you are using the correct database , in my system I am using people database which contain a collection or table called as Peoples. Now in order to add a record in to this collection Mongodb provide three commands :

db.collection.insertOne()	**Inserts a single document into a collection.**
db.collection.insertMany()	**db.collection.insertMany() inserts multiple documents into a collection.**
db.collection.insert()	**db.collection.insert() inserts a single document or multiple documents into a collection.**

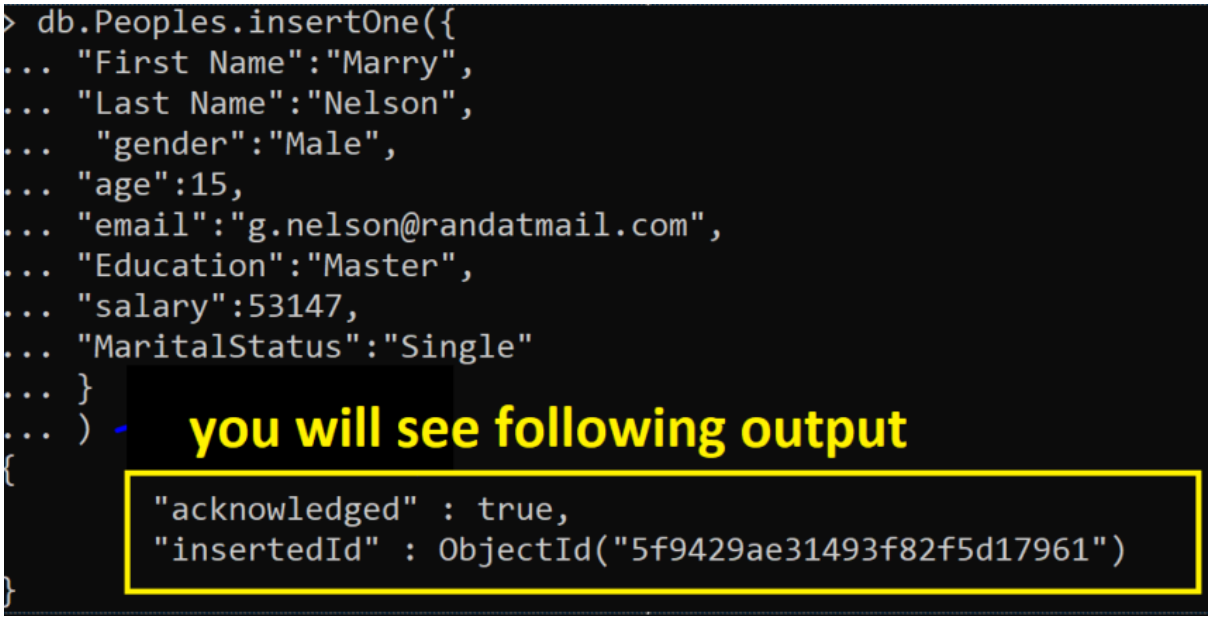
Where
collection is the name of the collection . for example if we are using People's collection then it will be **db.Peoples.insertOne()**

- Note. If the collection does not exist already, then using the command db.peoplesinsertOne() will create the People collection automatically .
- In the parenthesis of the command db.collection.insertOne(), The JSON data is given in {}with key value pair. Each key represents the field in the collection and its corresponding value is present after.Each key value pair is separated by a comma:

- Use the *CodeBlock* below to insert a document using db.collection.insertOne()

```
db.Peoples.insertOne({
        "First Name":"Marry",
        "Last Name":"Nelson",
         "gender":"Male",
        "age":15,
        "email":"g.nelson@randatmail.com",
        "Education":"Master",
        "salary":53147,
        "MaritalStatus":"Single"
        }
)
```

A JSON response will be returned to the console, displaying the inserted id.



Inserting multiple documents

Rather than inserting documents one method at a time, you can utilise the insertMany() command. Follow the steps below –

1. Using the command `db.Peoples.insertMany()` will create the people collection automatically if it does not exist. Between the parenthesis is where the array of JSON data should go. Note this time you have an array starting from [and ending] and with this [ ] there are two documents marked with different colors for your understanding

```
db.Peoples.insertMany(
```

db.Peoples.insertMany([
{"firstName":"Daryl","lastName":"Johnson","gender":"Male","age":20,"email":"d.johnson@randatmail.com","education":"Upper Secondary","salary":4450,"maritalStatus":"Married"},

{"firstName":"Justin","lastName":"West","gender":"Male","age":27,"email":"j.west@randatmail.com","education":"Doctoral","salary":5783,"maritalStatus":"Married"}

])

2. A JSON response will be returned to the console, displaying the inserted ids. See Figure below.

```
> db.Peoples.insertMany( [ {"firstName":"Daryl","lastName":"Johnson","gender":"Male","
age":20,"email":"d.johnson@randatmail.com","education":"Upper Secondary","salary":4450
,"maritalStatus":"Married"},
... {"firstName":"Justin","lastName":"West","gender":"Male","age":27,"email":"j.west@r
andatmail.com","education":"Doctoral","salary":5783,"maritalStatus":"Married"}
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f942ade31493f82f5d17962"),
                ObjectId("5f942ade31493f82f5d17963")
        ]
}
```

More Multiple Insert :

```
db.Peoples.insertMany([
 {
  firstName: "Ayesha",
  lastName: "Khan",
  gender: "Female",
  age: 28,
  email: "ayesha.khan@example.com",
  education: "Bachelor",
  salary: 5200,
  maritalStatus: "Single"
 },
 {
  firstName: "Omar",
  lastName: "Farooq",
  gender: "Male",
  age: 35,
  email: "omar.farooq@example.com",
  education: "Master",
  salary: 6100,
  maritalStatus: "Married"
 },
 {
  firstName: "Sara",
```

```
  lastName: "Iqbal",
  gender: "Female",
  age: 42,
  email: "sara.iqbal@example.com",
  education: "Master",
  salary: 5800,
  maritalStatus: "Single"
 },
 {
  firstName: "Ali",
  lastName: "Raza",
  gender: "Male",
  age: 23,
  email: "ali.raza@example.com",
  education: "Bachelor",
  salary: 4700,
  maritalStatus: "Single"
 }
])
```

Updating documents (data)

Updating a document uses a method called updateOne() which requires multiple parameters in the format of JSON. The general syntax is :

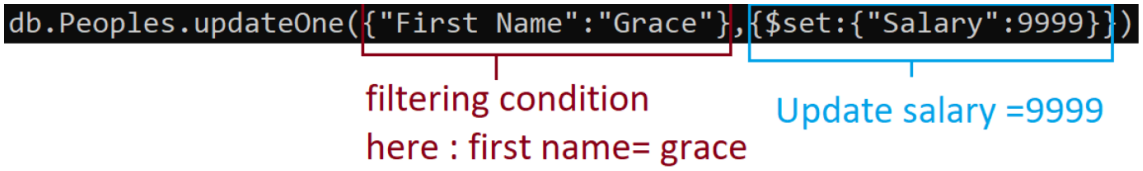
db.collection.updateOne(<filter>, <update>, <options>)

The updating command takes two parameter

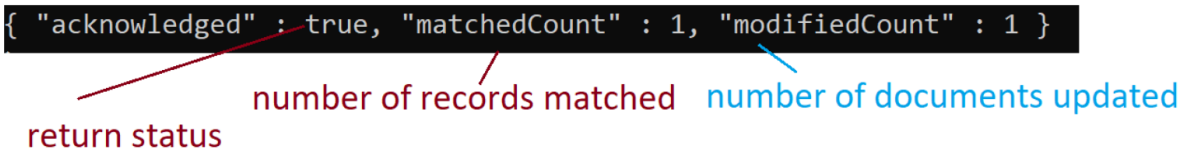
The first is the filter criteria or search term to find the document in question, the second is the values we want to set. $set command is used to set the new value

And third options are the optional parameter

Suppose in the Peoples collection you wish to change the salary of the person whose first name in the Peoples is “Grace” following is the command you will be using on the command prompt



- The response returned should give you an acknowledgment that the document was modified.



Command:

```
db.Peoples.updateOne(
  { firstName: "Daryl", lastName: "Johnson" },
  { $set: { salary: 5000 } }
)
```

```
People> db.Peoples.updateOne(
...   { firstName: "Daryl", lastName: "Johnson" },
...   { $set: { salary: 5000 } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Deleting documents (data)

Delete command follows the same formate as of update

db.collection.deleteOne(<filter>, <update>, <options>)

Enter the following command to delete the document that has the email of j.west@randatmail.com – db.people.deleteOne({email: “j.west@randatmail.com” })

Mongo should give a JSON response notifying the deletion was successful

Example : db.Peoples.deleteOne({ firstName: "Justin", lastName: "West" })

```
People> db.Peoples.deleteOne({ firstName: "Justin", lastName: "West" })
{ acknowledged: true, deletedCount: 1 }
People>
```

Finding total number of records in the collection :

db.Peoples.countDocuments()

```
People> db.Peoples.countDocuments()
1
```

Filtering Documents

i) Filtering documents : db.collectionname.find({}) ; with the {} specified the filtering condition having field:value in Json formate for example filtering data for Education=”Bachelor”

db.Peoples.find()

db.Peoples.find({"education":"Bachelor"}) returns all the documents having Education=Bachelor.

```
db.Peoples.find({"education":"Bachelor"})
```

```
People> db.Peoples.find({ maritalStatus: "Married" })
[
  {
    _id: ObjectId('68e277caf2b61c2bcacebea5'),
    firstName: 'Daryl',
    lastName: 'Johnson',
    gender: 'Male',
    age: 20,
    email: 'd.johnson@randatmail.com',
    education: 'Upper Secondary',
    salary: 5000,
    maritalStatus: 'Married'
  }
]
People>
```

ii) Finding total number of documents for a given criteria.

db.Peoples.find({“Education”:”Bachelor”}).count()

db.Peoples.find({ maritalStatus: "Married" }).count()

```
db.Peoples.find({"education":"Bachelor"}).count()
```

iii) **Select people who have Bachelor degree and have age less than 26. It will take two fields i.e. Education and Age. Note value of Age is without quotes because it is number field. The less than equal to is the operation mentioned as lte however a $ sign is placed before lteі.e. $lte**

db.Peoples.find({"education":"Bachelor", "age":{$lte:26} })

iv) Limiting number of the documents returned using limit method for example if you wish that only three documents should be displayed for a given filtering criteria :

db.Peoples.find({"Education":"Bachelor", "age":{$lte:26} }).limit(3)

```
> db.Peoples.find({"Education":"Bachelor", "Age":{$lte:26}   } ).limit(3)
{ "_id" : ObjectId("5f900e7a6f8e8e47f098996a"), "First Name" : "Grace", "Last Name" : "Nelson", "Gender" : "Fem
le", "Age" : 21, "Email" : "g.nelson@randatmail.com", "Education" : "Bachelor", "Salary" : 9999, "Marital Statu
" : "Single" }
{ "_id" : ObjectId("5f900e7a6f8e8e47f0989974"), "First Name" : "Rebecca", "Last Name" : "Douglas", "Gender" : "
emale", "Age" : 20, "Email" : "r.douglas@randatmail.com", "Education" : "Bachelor", "Salary" : 8283, "Marital S
atus" : "Single" }
{ "_id" : ObjectId("5f900e7a6f8e8e47f098997c"), "First Name" : "Naomi", "Last Name" : "Spencer", "Gender" : "Fe
ale", "Age" : 26, "Email" : "n.spencer@randatmail.com", "Education" : "Bachelor", "Salary" : 5907, "Marital Sta
us" : "Married" }
```

v) Note there are many fields such as First Name, Last Name, Age , Email etc in the output , however you can select the fields through placing a 1 before the field you want to show.

db.Peoples.find({"education":"Bachelor", "age":{$lte:26}} ,{"education":1,"age":1})

```
People> db.Peoples.find({"education":"Bachelor", "age":{$lte:26}} ,{"education":1,"age":1})
[
  {
    _id: ObjectId('68e27ad9f2b61c2bcacebeaa'),
    age: 23,
    education: 'Bachelor'
  }
]
```

Note limit(3) , it restricts the number of documents to be displayed in returned data set.

vi) Also find total number of document using db.Peoples.find({age:{$gte:25}}).count()

Applying more than one criterion on find :

Applying Boolean operator

vii) Applying boolean operator on filtering , suppose we want to filter the documents for gender = female and age not equal to 30. The general syntax is

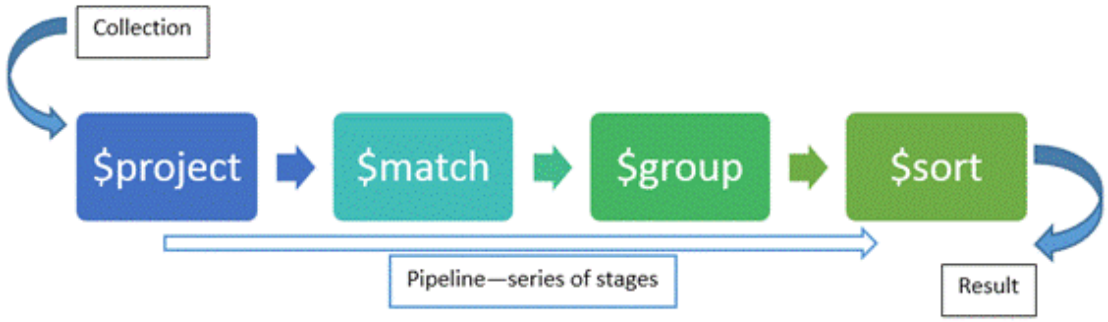
db.Peoples.find({ $and: [{field1:condition},{field2:condition}] })

db.Peoples.find({ $and: [{ age: { $ne: 30 } }, {Gender: “Female”}] })

Aggergrate functions:

Aggregation in MongoDB returns the computed results grouping the data from one or multiple documents. Aggregations can be used to apply a sequence of query-operations to the documents in a collection, reducing and transforming them.

Documents are processed through the stages in sequence, with each stage applying to each document individually. When calling aggregate on a collection, we pass a list of stage operators.



$match

The first stage of a pipeline is matching, and that allows to filter out the documents so that we are only manipulating the documents that we care about. In order to achieve the best performance of the $match stage, use it early in the aggregation process.

```
{ $match: { <query> } }
```

$group

Groups input documents by the specified _id expression and for each distinct grouping, outputs a document. The _id field of each output document contains the unique group by value. The output documents can also contain computed fields that hold the values of some accumulator expression.

```
{
  $group:
    {
      _id: <expression>, // Group By Expression
      <field1>: { <accumulator1> : <expression1> },
      ...
    }
 }
```

```
> db.Peoples.aggregate([    { $match: { Gender: "Female" } },    { $group: { _id: "$Education", AverageSalary:
{ $avg: "$Salary" } } } ])
{ "_id" : "Master", "AverageSalary" : 4207.923076923077 }
{ "_id" : "Doctoral", "AverageSalary" : 4510.3125 }
{ "_id" : "Lower secondary", "AverageSalary" : 5515.181818181818 }
{ "_id" : "Primary", "AverageSalary" : 4359.05 }
{ "_id" : "Bachelor", "AverageSalary" : 5656.9 }
{ "_id" : "Upper secondary", "AverageSalary" : 5368 }
```

Counting on aggergrate

```
db.Peoples.aggregate([    { $match: { Gender: "Female" } },    { $group: { _id: "$Education", count: {
$sum: 1} } } ])
```

```
db.Peoples.aggregate([    { $match: { Gender: "Female" } },    { $group: { _id: "$Education", count: {
$sum: 1} } ,AverageSalary:{}}


 ]

)
```

Query Selectors

Comparison ¶

For comparison of different BSON type values, see the specified BSON comparison order.

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Logical

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

Element

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

Aggerate Expressions

	Expression	Description |
1	$sum	Summates the defined values from all the documents in a collection
2	$avg	Calculates the average values from all the documents in a collection
3	$min	Return the minimum of all values of documents in a collection
4	$max	Return the maximum of all values of documents in a collection
5	$addToSet	Inserts values to an array but no duplicates in the resulting document
6	$push	Inserts values to an array in the resulting document
7	$first	Returns the first document from the source document
8	$last	Returns the last document from the source document