University of East London

WEEK 2

CN5006: Web and Mobile Application Development

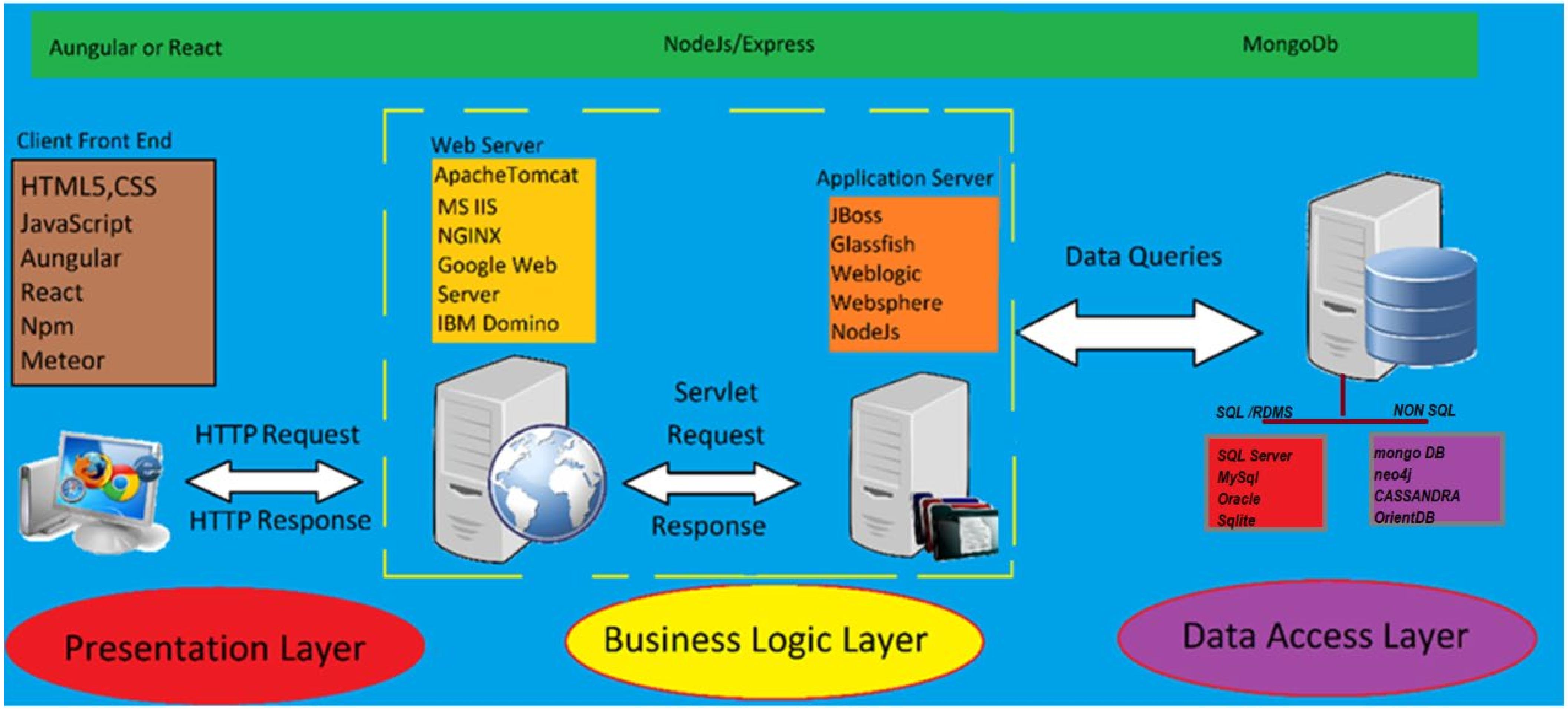
Dr. Nadeem Qazi

Introduction to MongoDB

Learning OutCome

- Databases
- Non SQL Databases
- Connecting with MongoDB
  - using MongoDB Compus
  - MongoDB shell commands
- CRUD and Aggregation functions:

N-Tier Application



Overall view of Database

Database: a very large, integrated and interconnected collection of data.

Models a real-world *enterprise*

– Entities (e.g., University,Departments, games)

– Relationships (e.g., IT is a Computer Science Branch)

– More recently, also includes active components , often called “business logic”. (e.g., the BCS ranking system)

A *Database Management System (DBMS)* is a software system designed to store, manage, and facilitate access to databases

Database Management System (DBMS)

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases can be very large.
- Databases touch all aspects of our lives

• SQL/RDBMS/Relational Databases (Structured)

1. Presents Data in structured Form, use a scheme in the form of collection of 2D tables of Rows and Columns
2. Each Table represents the fixed attributes and data types,with associated keys to identify specific column or rows of a table.
3. Provides functionality for reading, creating, updating, and deleting data,through Structured Query Language (SQL) statements.
4. Provides data integrity through constraints to ensure that the data contained in tables are reliable and accurate.
5. Relational databases and related management systems (RDBMS) are more widely known and understood than their unstructured NoSQL cousin

RDBMSs don't work well — or at all — with unstructured or semi-structured data due to schema and type constraints. This makes them ill-suited for large analytics or IoT event loads.

Popular RDMS Database

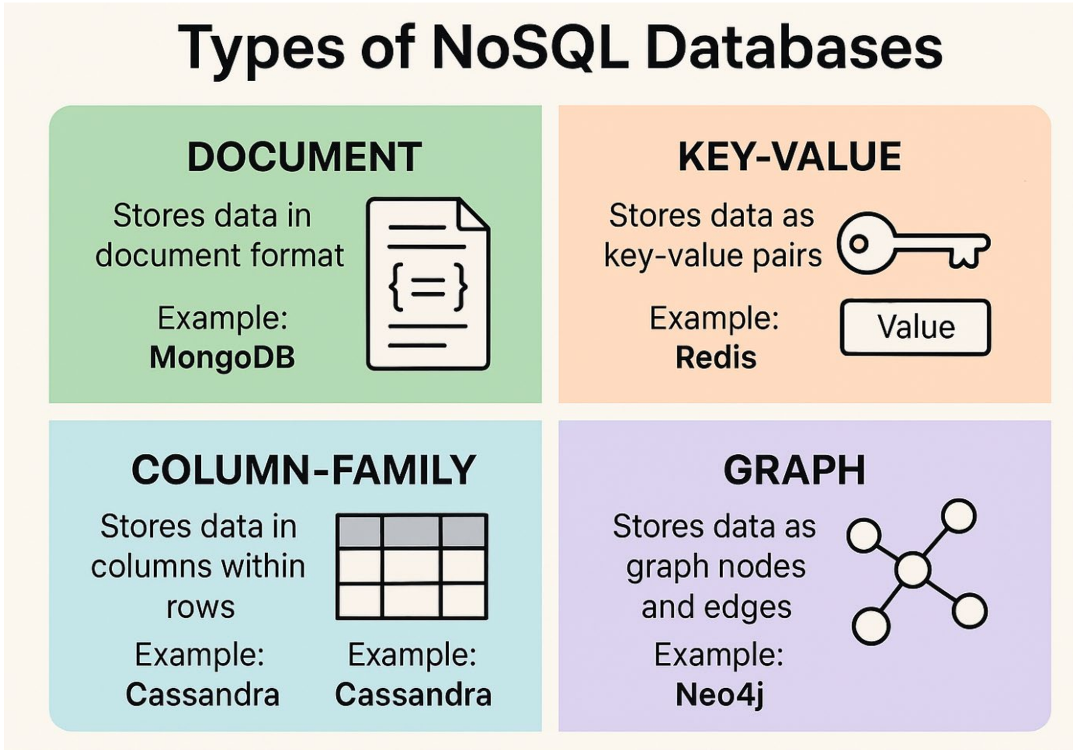
- **Oracle**: Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a multi-model database management system produced and marketed by Oracle Corporation.
- **MySQL**: MySQL is an open-source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX, and Windows.
- **Microsoft SQL Server**: Microsoft SQL Server is an RDBMS that supports a wide variety of transaction processing, business intelligence, and analytics applications in corporate IT environments.
- **PostgreSQL**: PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance.
- **DB2**: DB2 is an RDBMS designed to store, analyze, and retrieve data efficiently

NoSQL Databases

No SQL databases are non-relational databases designed to handle unstructured, semi-structured, and structured data. They offer flexibility, scalability, and high performance, making them ideal for modern applications like real-time analytics, IoT, and cloud-native systems.

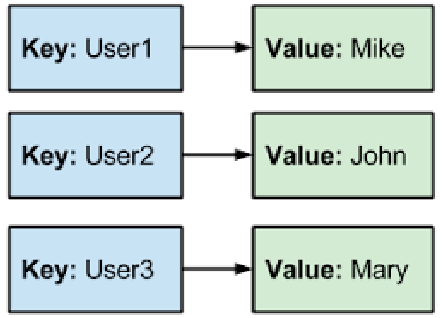
Does not follow the relational model, More scalable and flexible than RDMS

- **Handles unstructured data that doesn't fit neatly into rows and columns**
- **Typically open source**



https://youtu.be/wzcb10_lQqA

NoSQL Databases

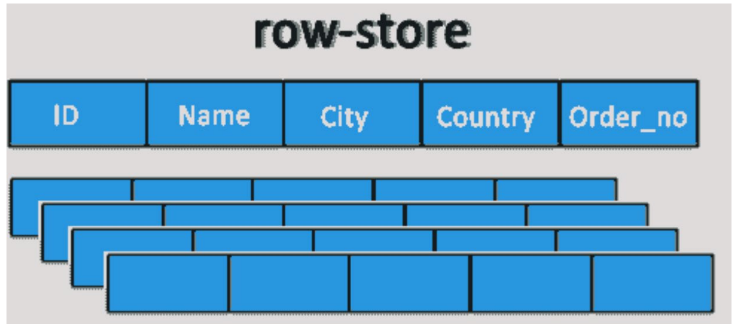


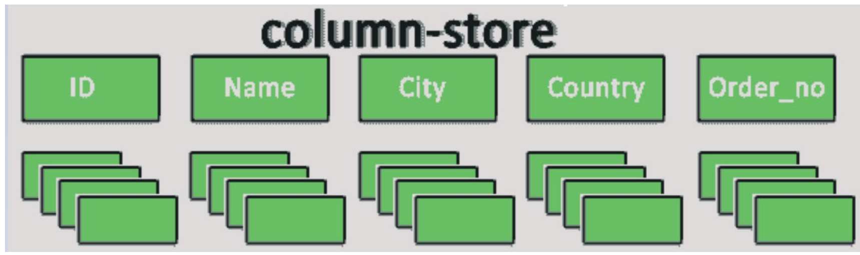
- **Key-value stores**
- Store only key-value pairs ,provide basic functionality for retrieving the value
- associated with a known key.
- Data can be stored in native data type of Programing language
- Suited to embedded Non Complex databases and speed is of paramount importance.
- Examples Redis and Amazon DynamoDB,
- **Wide column Database**
- organizes data storage into flexible columns rather than rows, that can be spread across multiple

  servers or database nodes, using multi-dimensional mapping to reference

data by column, row, and timestamp.

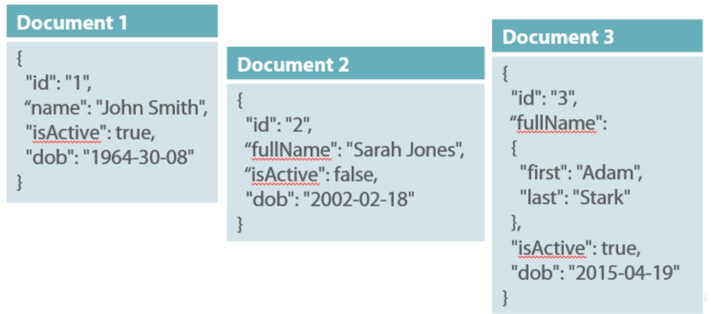
Cassandra, Scylla, and HBase,





•

NoSQL Databases

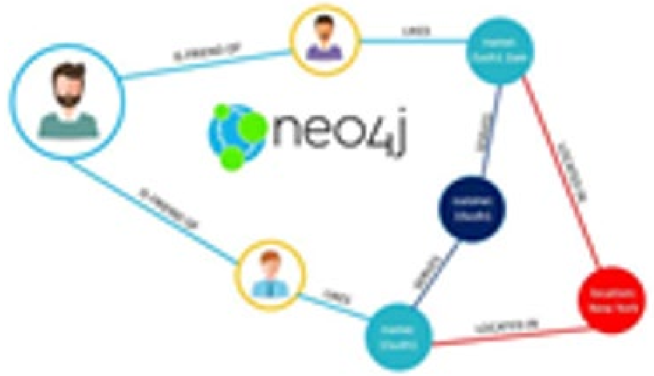


Document 1
```
{
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
}
```

Document 2
```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3
```
{
  "id": "3",
  "fullName":
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

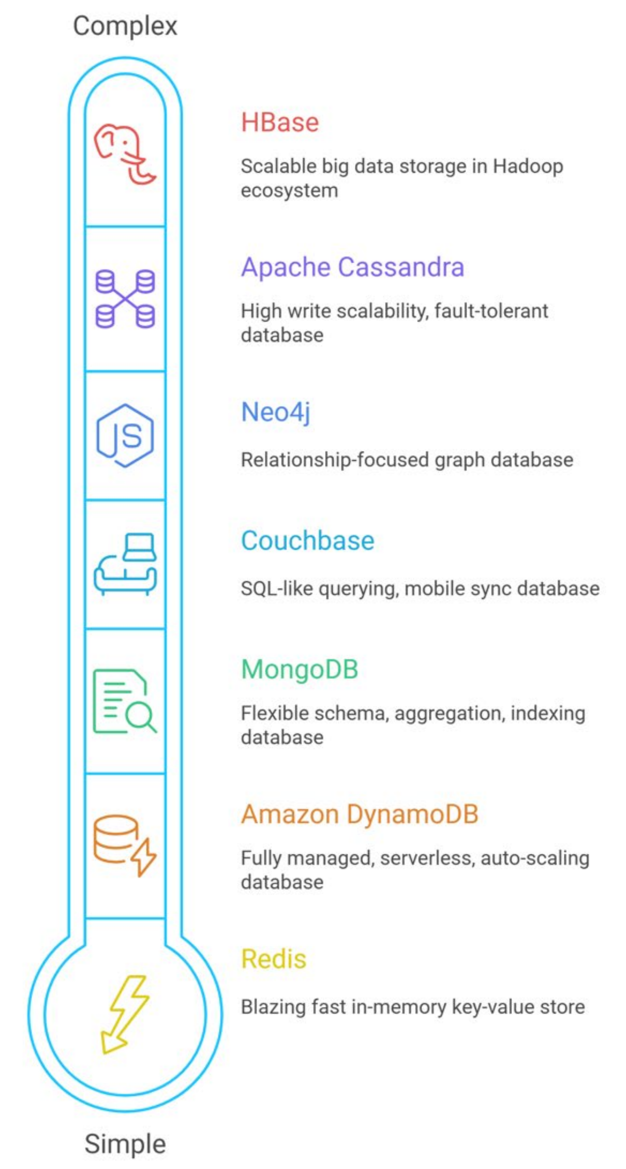
- **Document Database**
  - These databases store semi-structured information in a document format such as JSON or XML
  - This format makes updating and creating programs easy.
  - document-based stores are often used for financial services and content management data. mongoDB
- **Graph Database**
  - Data is stored data in nodes and corresponding relationship in vertices. A node may have multiple relationship
  - Optimal for searching social network data



No SQL Database Comparison

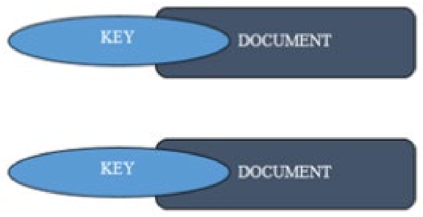
Feature/Database	Type	Features	Use Cases	Ease of Use
MongoDB	Document	JSON-like BSON documents, flexible schema, aggregation, indexing	Web apps, CMS, APIs	⭐⭐⭐⭐
Redis	Key-Value (In-Memory)	Ultra-fast, supports data structures like lists, sets	Caching, session storage, real-time analytics	⭐⭐⭐⭐⭐
Apache Cassandra	Wide-Column	High write scalability, fault-tolerant, distributed	IoT, social media, time-series data	⭐⭐
Amazon DynamoDB	Key-Value & Document	Fully managed, serverless, auto-scaling	Mobile apps, gaming, IoT	⭐⭐⭐⭐⭐
Couchbase	Document & Key-Value	SQL-like querying, mobile sync, high performance	Hybrid storage, enterprise apps	⭐⭐⭐
Neo4j	Graph	Relationship-focused, Cypher query language	Fraud detection, recommendation engines	⭐⭐⭐
HBase	Wide-Column	Hadoop ecosystem, scalable big data storage	Analytics, finance apps	⭐⭐

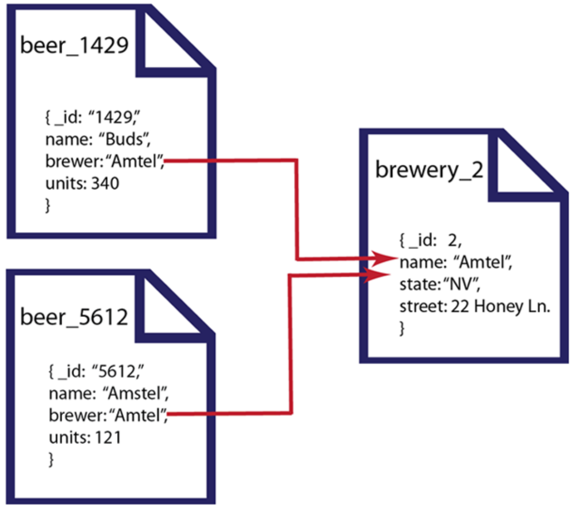
Database ease of use ranges from simple to complex.


Complex
HBase
Scalable big data storage in Hadoop ecosystem
Apache Cassandra
High write scalability, fault-tolerant database
Neo4j
Relationship-focused graph database
Couchbase
SQL-like querying, mobile sync database
MongoDB
Flexible schema, aggregation, indexing database
Amazon DynamoDB
Fully managed, serverless, auto-scaling database
Redis
Blazing fast in-memory key-value store
Simple

NoSQL Databases:MongoDB

- **DocumentDatabase** , are schema-free systems that store data in the form of JSON documents.
- Similar to key-value, Document name is the key and the contents of the document, are the value.
- In a document store, individual records do not require a uniform structure, can contain many different value types, and can be nested.

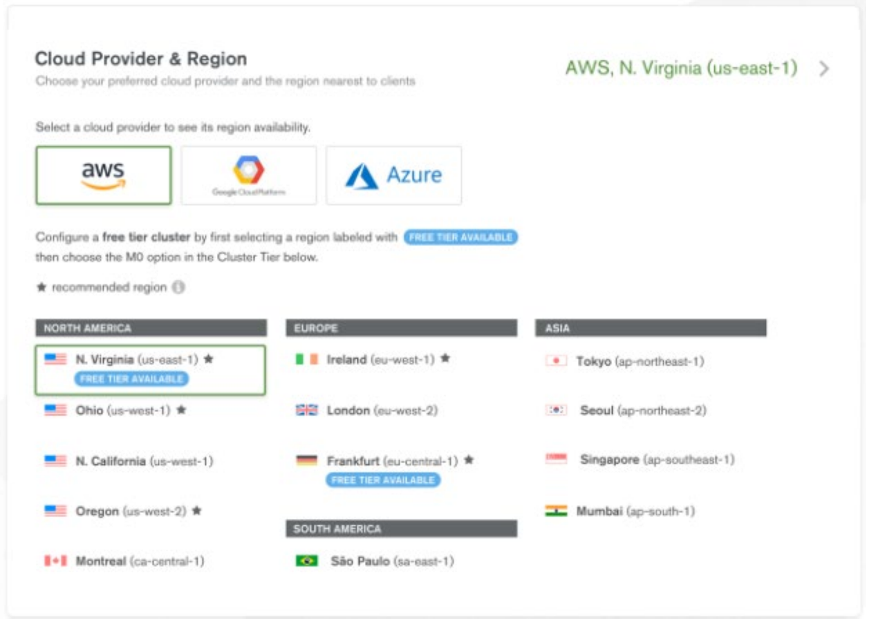




MongoDB	RDMS
Database	Database
Collection	Table
Document	Row
SchemaLess	Schema

Atlas: Cloud base MongoDB

- Multi-cloud database service for MongoDB.
- Available on AWS, Google Cloud, and Azure.
- Go to website https://www.mongodb.com/cloud/atlas
- Sign in and start using it.



Cloud Provider & Region

Choose your preferred cloud provider and the region nearest to clients

AWS, N. Virginia (us-east-1)

Select a cloud provider to see its region availability.

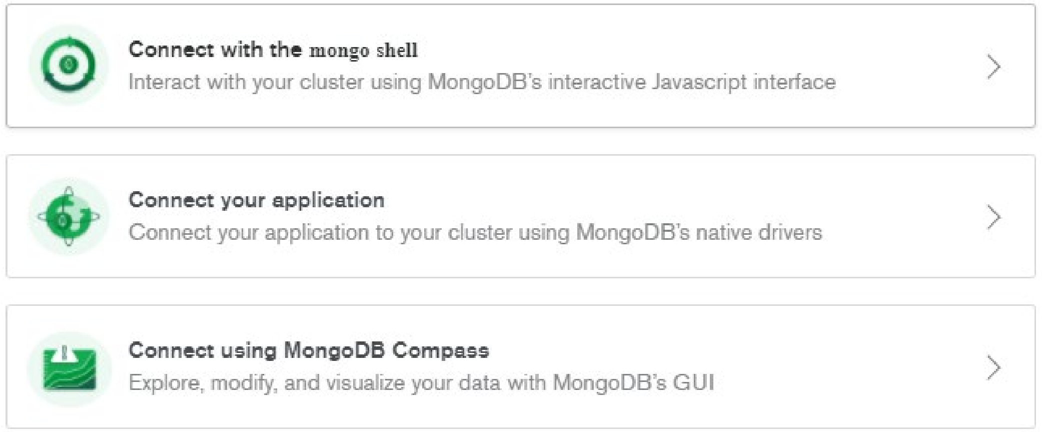
Configure a **free tier cluster** by first selecting a region labeled with FREE TIER AVAILABLE then choose the M0 option in the Cluster Tier below.

★ recommended region

NORTH AMERICA	EUROPE	ASIA
N. Virginia (us-east-1) ★ FREE TIER AVAILABLE	**Ireland** (eu-west-1) ★	**Tokyo** (ap-northeast-1)
Ohio (us-west-1) ★	**London** (eu-west-2)	**Seoul** (ap-northeast-2)
N. California (us-west-1)	**Frankfurt** (eu-central-1) ★ FREE TIER AVAILABLE	**Singapore** (ap-southeast-1)
Oregon (us-west-2) ★	SOUTH AMERICA	**Mumbai** (ap-south-1)
Montreal (ca-central-1)	**São Paulo** (sa-east-1)	

Multiple Ways to connect with MongoDB

1. **MongoDB shell (Needs installation on you laptop)**
2. **MongoDB Compass (Needs installation on you laptop)**
3. Connect your application using Node
   1. Repl.it (cloud base,Does Not need installation we will using Atlas
   2. **Visual Studio Code.**



Connect with the mongo shell
Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI

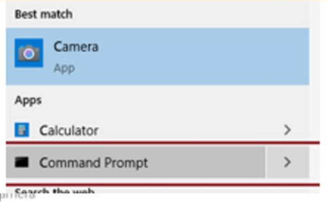
Mongo Shell

- If you are using Mongo DB version 6 or above chances are that you should be installing Mongodb shell, otherwise for earlier version mongo shell is automatically installed . Checked the directory
- Your installation directory/Server/bin
- Down load it from   https://www.mongodb.com/try/download/shell
- Download and unzip , copy the files(**mongosh.exe**) from unzip folder to the
- Your MongoDB Server installation directory/Server/bin
- C:\Program Files\MongoDB\Server\7.0\bin
- Mongosh is the new Mongo Shell includes improved features such as syntax highlighting, command history, and logging.
- It's compatible with: Red Hat Enterprise Linux (RHEL) 7, Amazon Linux 2, SUSE Linux Enterprise Server (SLES) 12, and Ubuntu 18.04.

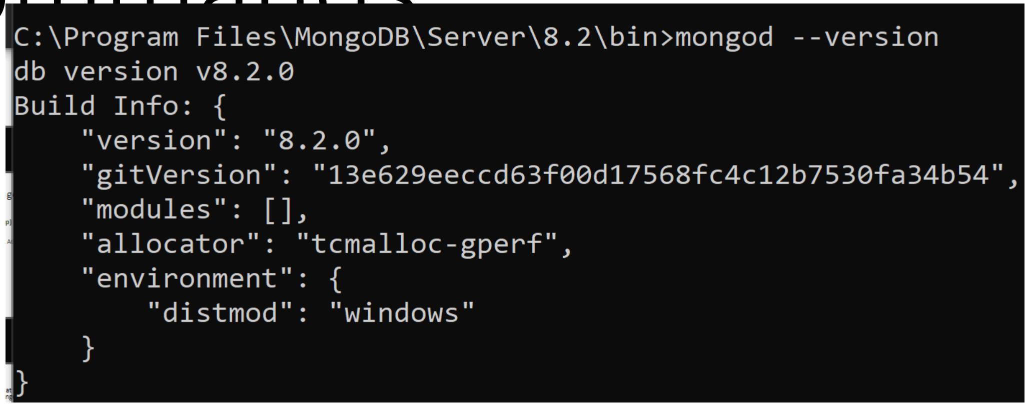
MonGODB Shell commands

Opening the Mongodb Shell for running these commands

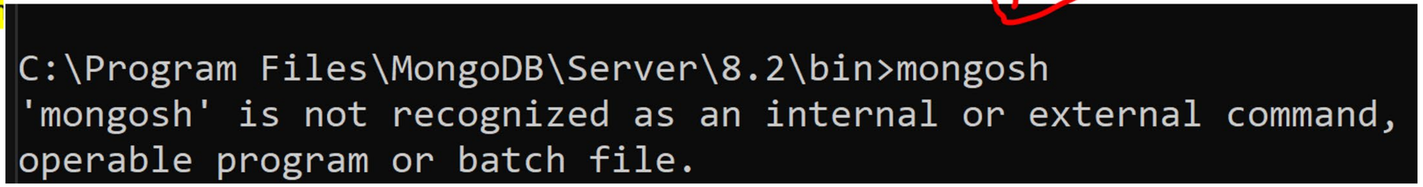
1. How to open mongodb shell
   a. Open the command prompt



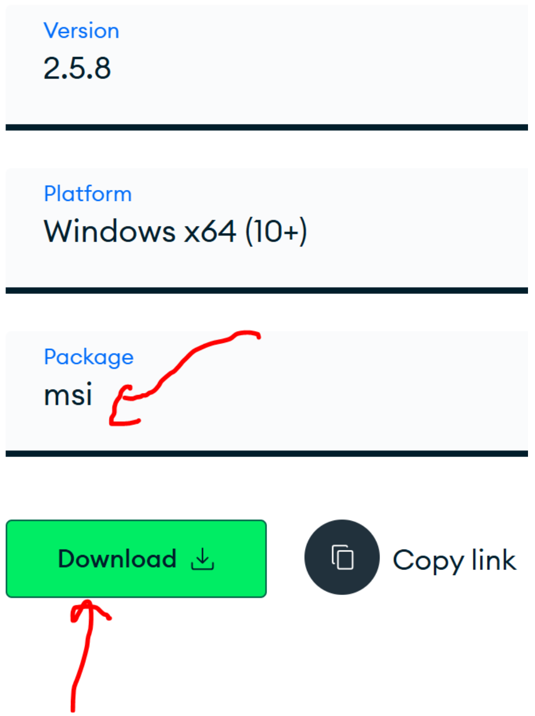
   b. Go to the directory wh



```
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>mongosh
'mongosh' is not recognized as an internal or external command,
operable program or batch file.
```

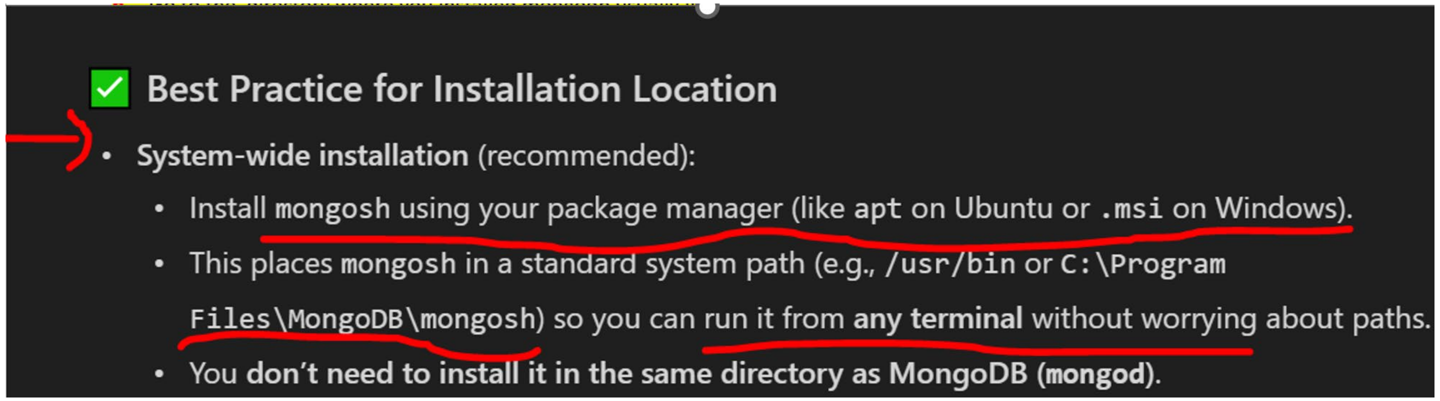


Version
2.5.8

Platform
Windows x64 (10+)

Package
msi

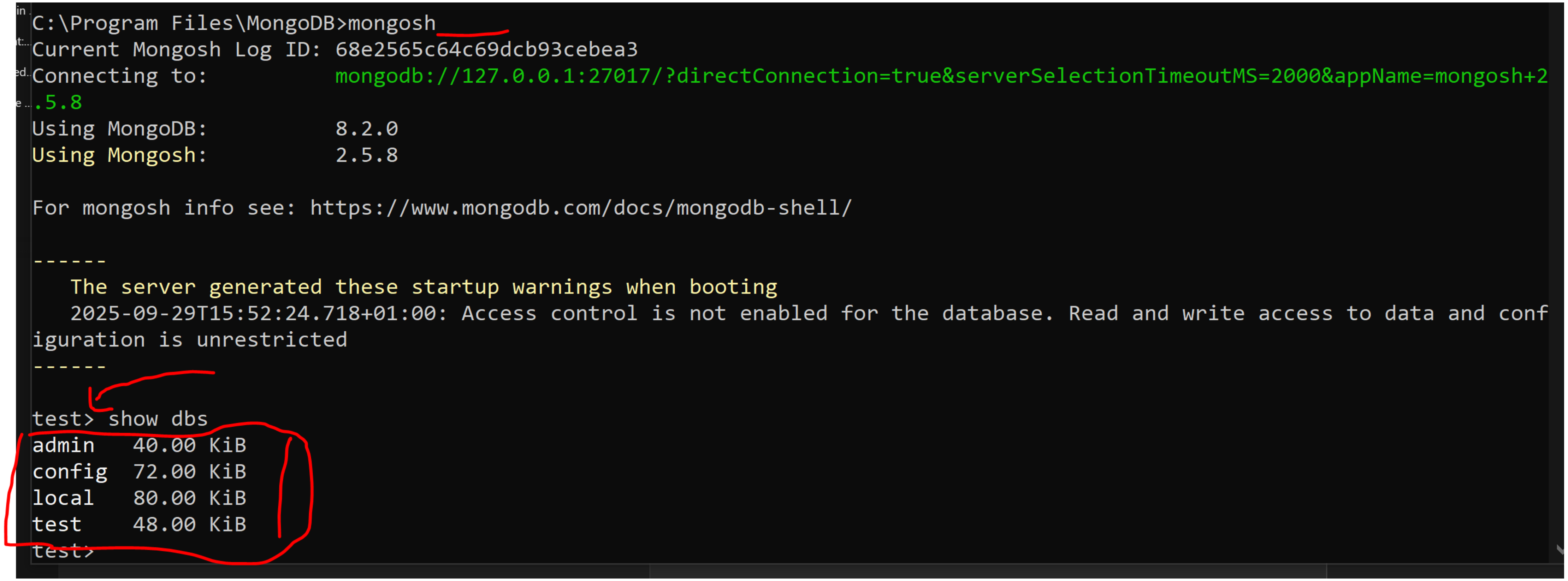
https://www.mongodb.com/try/download/shell



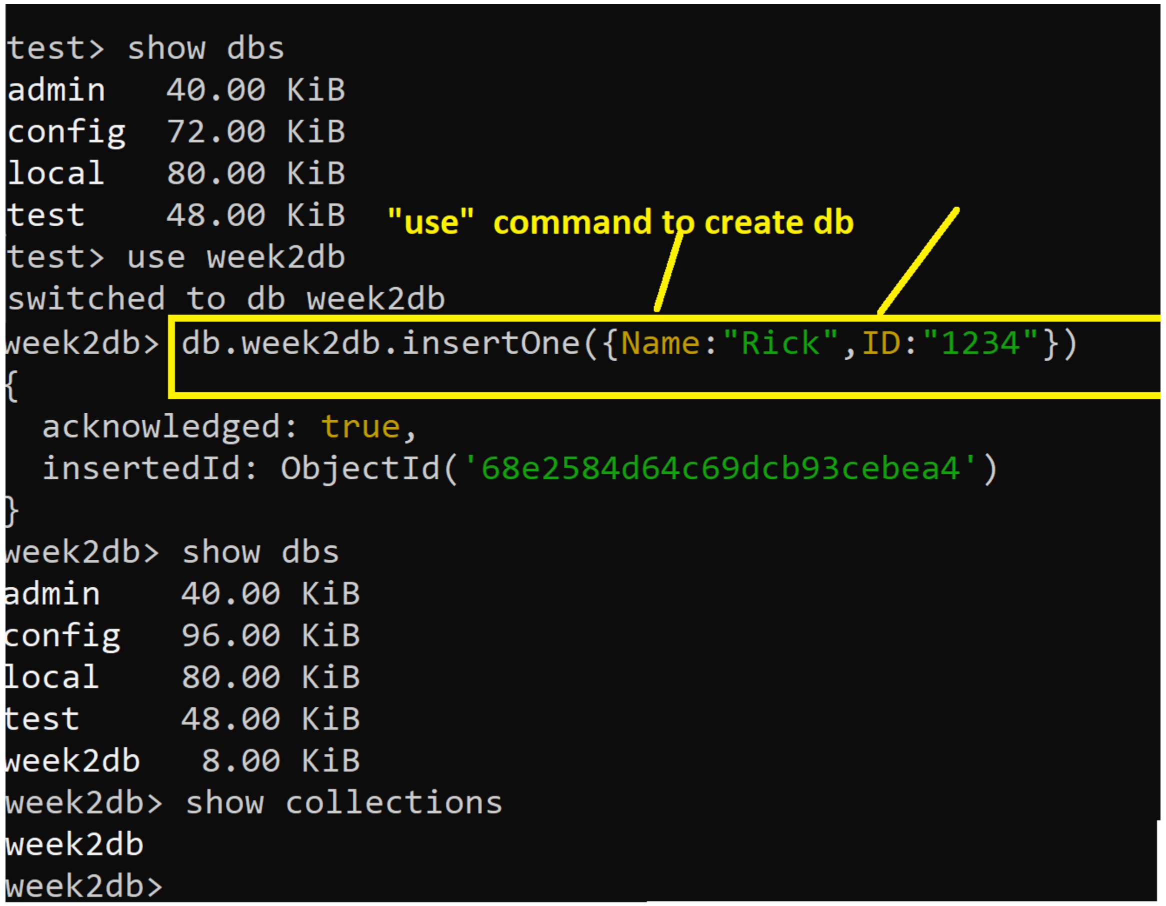
✅ **Best Practice for Installation Location**

- **System-wide installation** (recommended):
  - Install `mongosh` using your package manager (like `apt` on Ubuntu or `.msi` on Windows).
  - This places `mongosh` in a standard system path (e.g., `/usr/bin` or `C:\Program Files\MongoDB\mongosh`) so you can run it from **any terminal** without worrying about paths.
  - You **don't need to install it in the same directory as MongoDB (`mongod`)**.

Mongosh



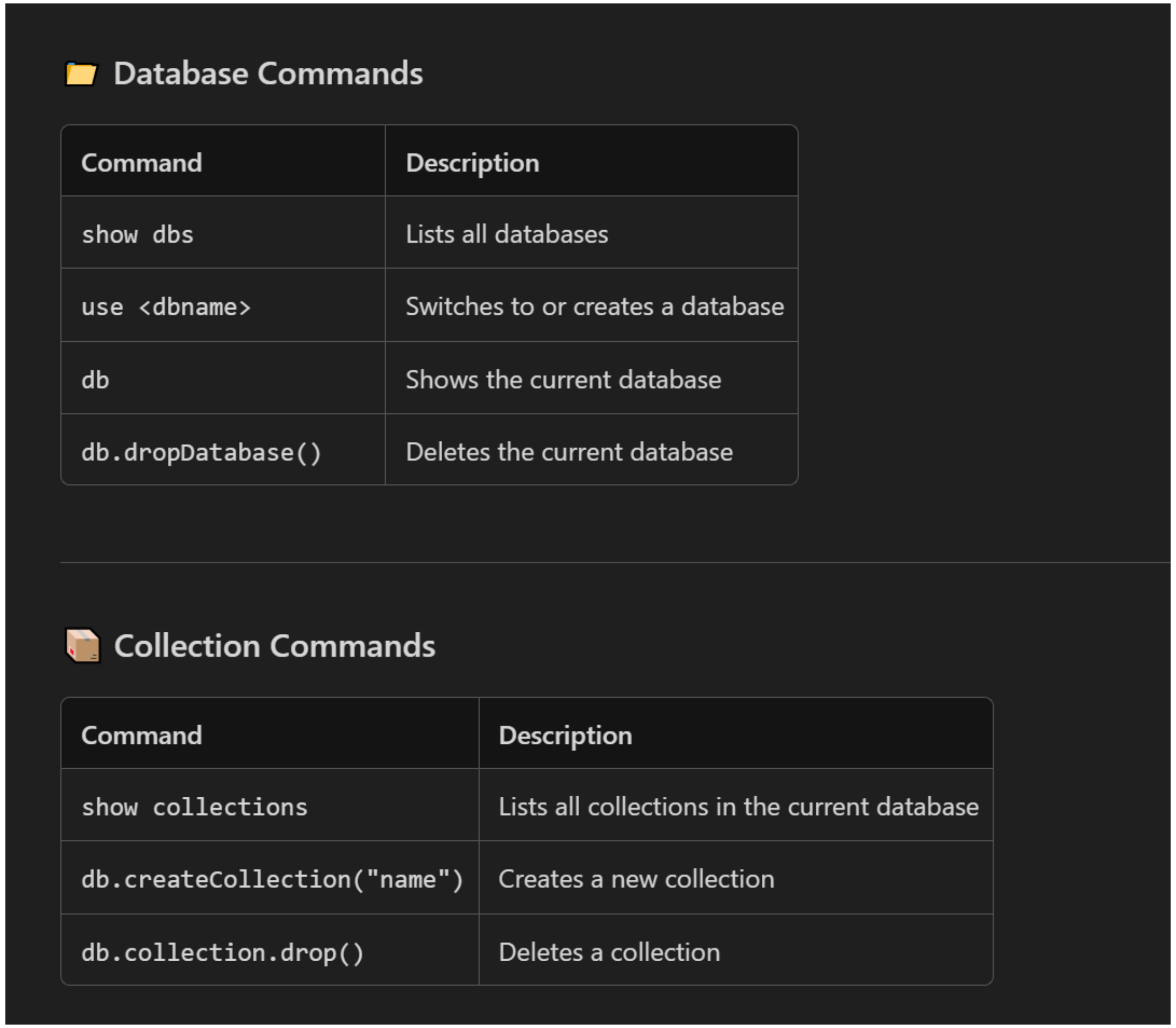
```
C:\Program Files\MongoDB>mongosh
Current Mongosh Log ID: 68e2565c64c69dcb93cebea3
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.8
Using MongoDB:          8.2.0
Using Mongosh:          2.5.8

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-29T15:52:24.718+01:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test> show dbs
admin   40.00 KiB
config  72.00 KiB
local   80.00 KiB
test    48.00 KiB
test>
```



```
test> show dbs
admin   40.00 KiB
config  72.00 KiB
local   80.00 KiB
test    48.00 KiB
test> use week2db
switched to db week2db
week2db> db.week2db.insertOne({Name:"Rick",ID:"1234"})
{
  acknowledged: true,
  insertedId: ObjectId('68e2584d64c69dcb93cebea4')
}
week2db> show dbs
admin     40.00 KiB
config    96.00 KiB
local     80.00 KiB
test      48.00 KiB
week2db    8.00 KiB
week2db> show collections
week2db
week2db>
```

There is no "create" command in the MongoDB Shell. In order to create a database, you will first need to switch the context to a non-existing database using the use command:

Common Mongodb Commands

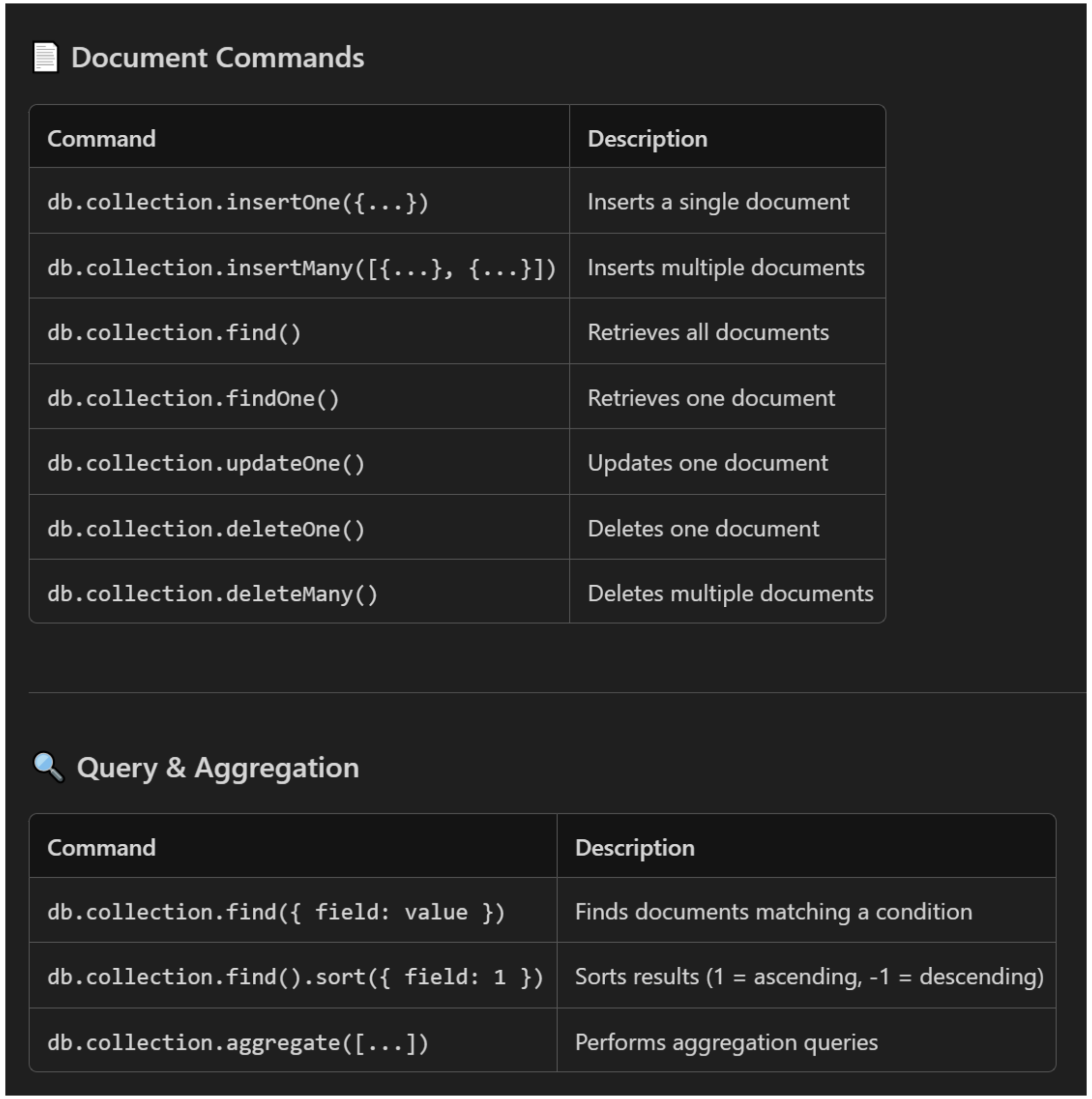


📁 Database Commands

Command	Description
`show dbs`	Lists all databases
`use <dbname>`	Switches to or creates a database
`db`	Shows the current database
`db.dropDatabase()`	Deletes the current database

📦 Collection Commands

Command	Description
`show collections`	Lists all collections in the current database
`db.createCollection("name")`	Creates a new collection
`db.collection.drop()`	Deletes a collection



📄 Document Commands

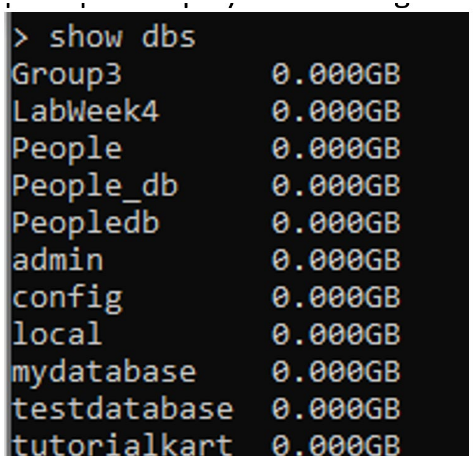
Command	Description
`db.collection.insertOne({...})`	Inserts a single document
`db.collection.insertMany([{...}, {...}])`	Inserts multiple documents
`db.collection.find()`	Retrieves all documents
`db.collection.findOne()`	Retrieves one document
`db.collection.updateOne()`	Updates one document
`db.collection.deleteOne()`	Deletes one document
`db.collection.deleteMany()`	Deletes multiple documents

🔍 Query & Aggregation

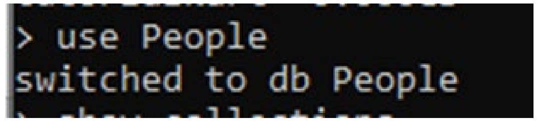
Command	Description
`db.collection.find({ field: value })`	Finds documents matching a condition
`db.collection.find().sort({ field: 1 })`	Sorts results (1 = ascending, -1 = descending)
`db.collection.aggregate([...])`	Performs aggregation queries

How to show all the databases in your mongo dB server

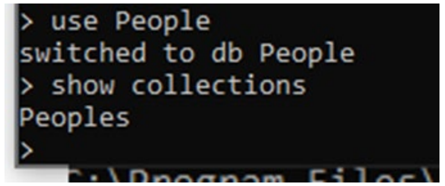
- Command: “Showdbs”
- **“Show dbs”** it will show name of all the databases in your mongoDB server .
- How to select a database.
- “use” is the command for selection a databases .
- the general syntax is use [name of the database].
- How to list the collections in the selected database?
- Show collections or db.getCollectionNames()



```
> 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
```



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



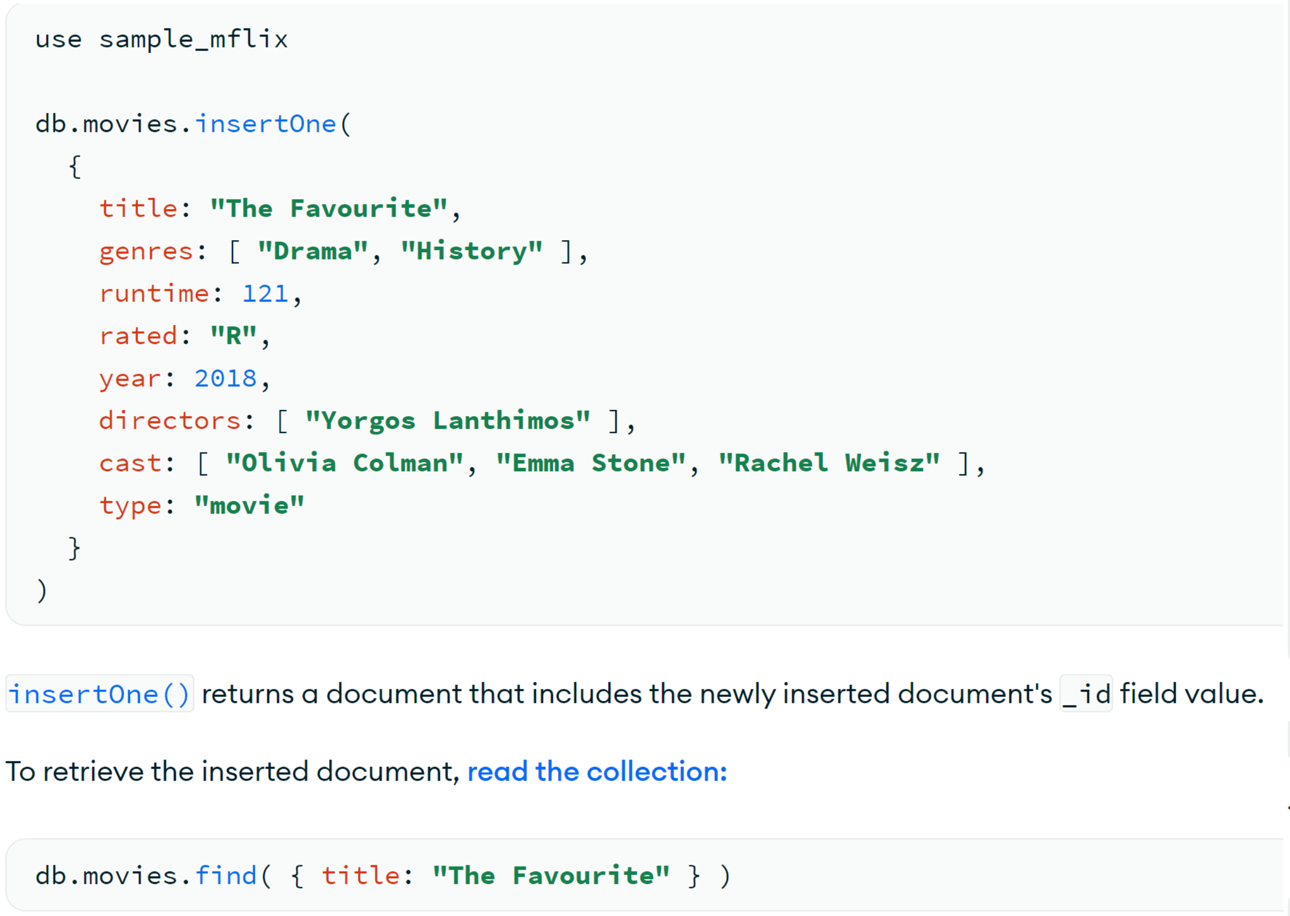
```
> use People
switched to db People
> show collections
Peoples
>
```

CRUDE command used for MongoDB

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.Aggregate()	Perform pipeline of match and group the data

https://www.mongodb.com/docs/mongodb-shell/crud/

Shell commands

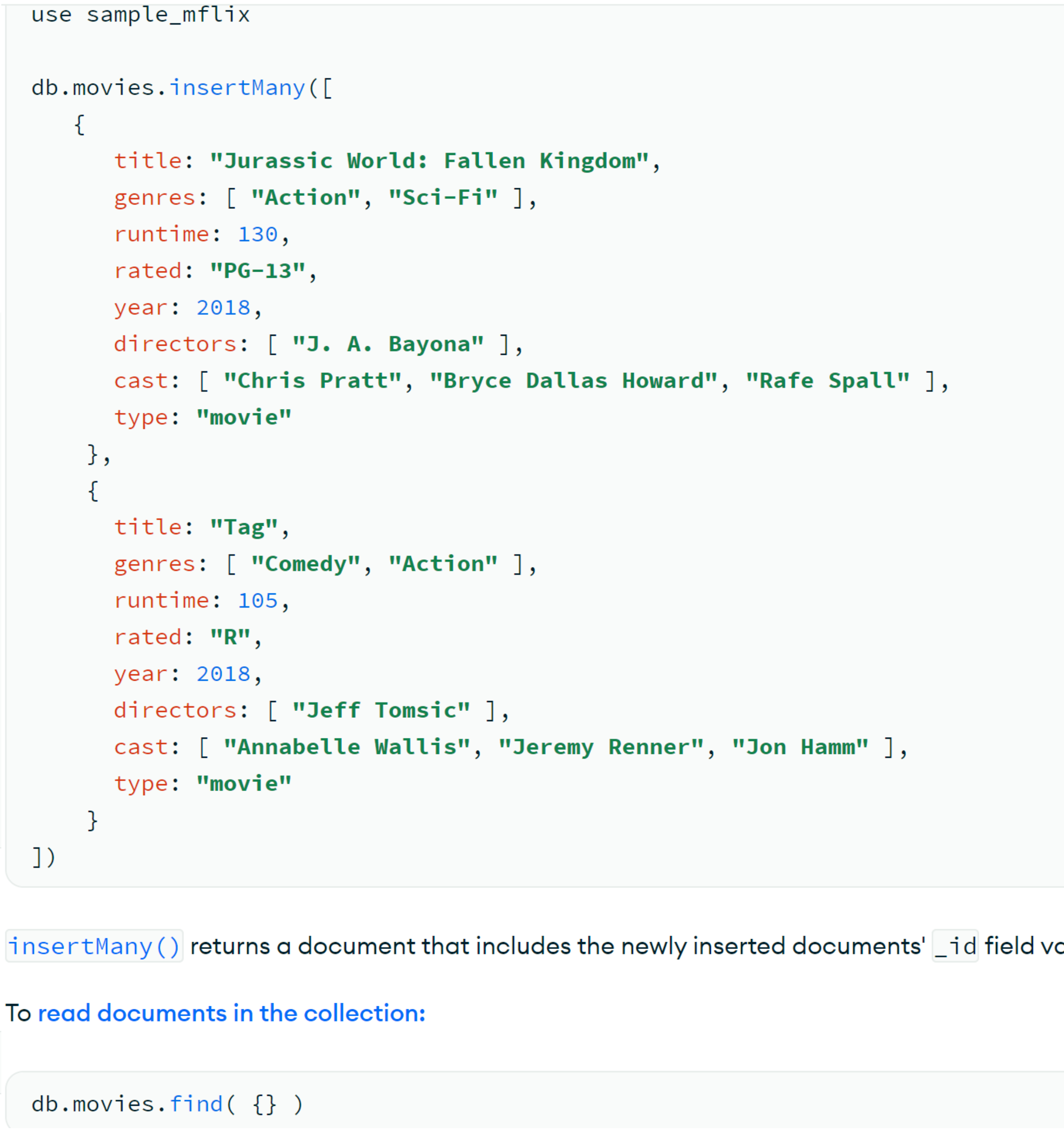


```
use sample_mflix

db.movies.insertOne(
  {
    title: "The Favourite",
    genres: [ "Drama", "History" ],
    runtime: 121,
    rated: "R",
    year: 2018,
    directors: [ "Yorgos Lanthimos" ],
    cast: [ "Olivia Colman", "Emma Stone", "Rachel Weisz" ],
    type: "movie"
  }
)
```

`insertOne()` returns a document that includes the newly inserted document's `_id` field value.

To retrieve the inserted document, read the collection:

```
db.movies.find( { title: "The Favourite" } )
```



```
use sample_mflix

db.movies.insertMany([
   {
      title: "Jurassic World: Fallen Kingdom",
      genres: [ "Action", "Sci-Fi" ],
      runtime: 130,
      rated: "PG-13",
      year: 2018,
      directors: [ "J. A. Bayona" ],
      cast: [ "Chris Pratt", "Bryce Dallas Howard", "Rafe Spall" ],
      type: "movie"
    },
    {
      title: "Tag",
      genres: [ "Comedy", "Action" ],
      runtime: 105,
      rated: "R",
      year: 2018,
      directors: [ "Jeff Tomsic" ],
      cast: [ "Annabelle Wallis", "Jeremy Renner", "Jon Hamm" ],
      type: "movie"
    }
])
```

`insertMany()` returns a document that includes the newly inserted documents' `_id` field va

To read documents in the collection:

```
db.movies.find( {} )
```

Inserting a record in a collection

- Mongodb provide three commands to add a record in the database

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

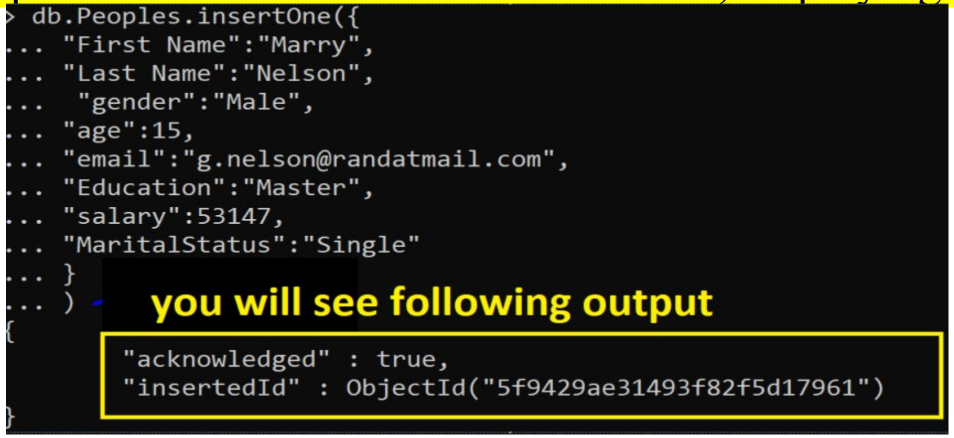
collection is the name of the collection . for example if we are using Peoples collection then it will be **db.Peoples.insertOne()**

- 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

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

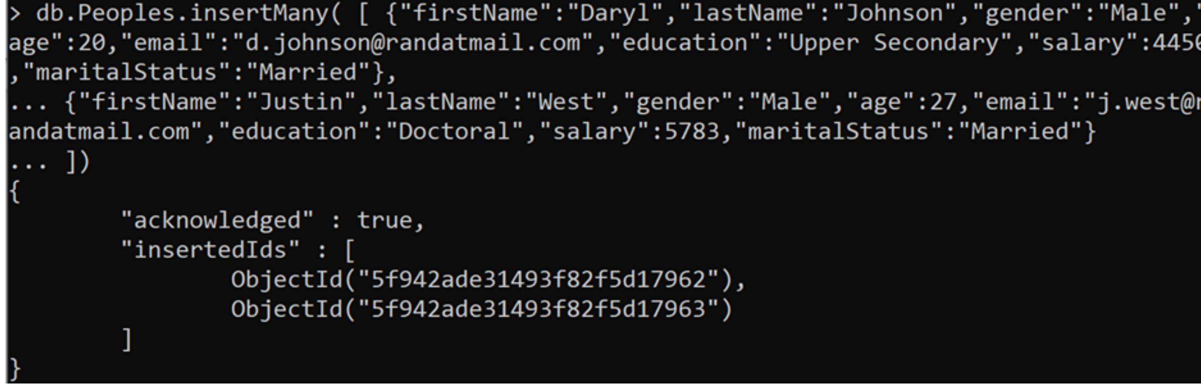
A JSON response will be returned to the console, displaying the inserted id.



```
> db.Peoples.insertOne({
...  "First Name":"Marry",
...  "Last Name":"Nelson",
...   "gender":"Male",
...  "age":15,
...  "email":"g.nelson@randatmail.com",
...  "Education":"Master",
...  "salary":53147,
...  "MaritalStatus":"Single"
...  }
...  )
{
     "acknowledged" : true,
     "insertedId" : ObjectId("5f9429ae31493f82f5d17961")
}
```

you will see following output

Inserting Multiple Records:

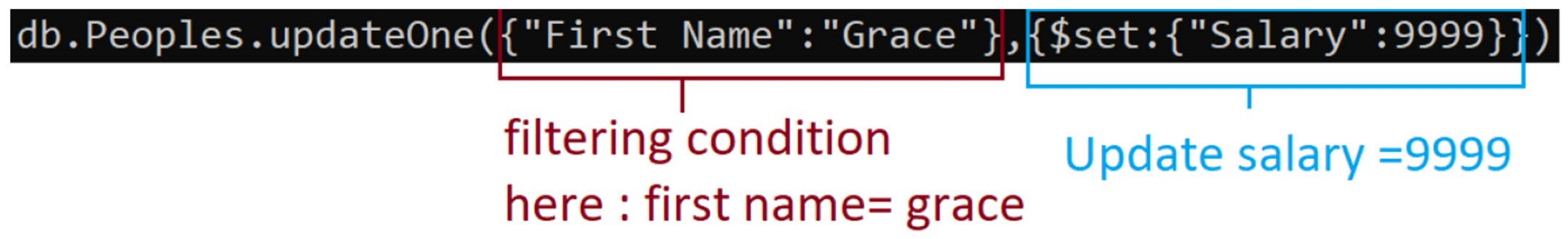


```
> db.Peoples.insertMany( [ {"firstName":"Daryl","lastName":"Johnson","gender":"Male",
age":20,"email":"d.johnson@randatmail.com","education":"Upper Secondary","salary":445
,"maritalStatus":"Married"},
... {"firstName":"Justin","lastName":"West","gender":"Male","age":27,"email":"j.west@r
andatmail.com","education":"Doctoral","salary":5783,"maritalStatus":"Married"}
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f942ade31493f82f5d17962"),
                ObjectId("5f942ade31493f82f5d17963")
        ]
}
```

- insertMany command
- 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([{record1},{record2}…..])
- 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"}
-])

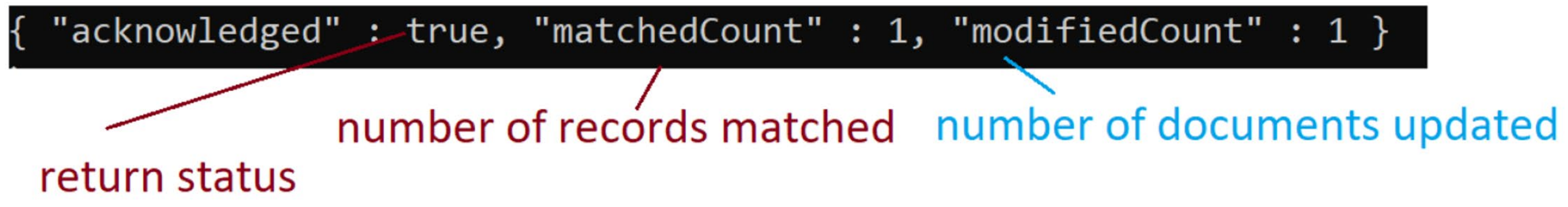
Updating documents (data)

- **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”



```
db.Peoples.updateOne({"First Name":"Grace"},{$set:{"Salary":9999}})
```

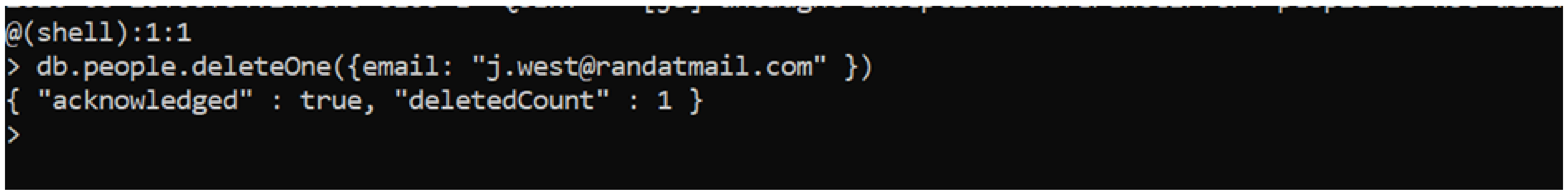
- The response returned should give you an acknowledgment that the document was modified.



```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Delete the record

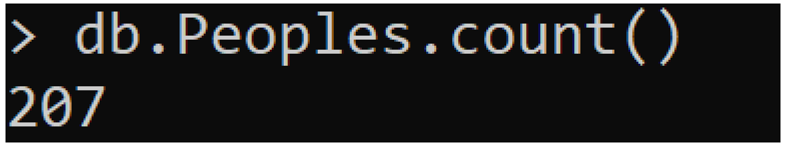
- db.collection.deleteOne(<filter>, <update>, <options>)
- to delete the document that has the email of j.west@randatmail.com – db.people.deleteOne({email: “j.west@randatmail.com” })



```
@(shell):1:1
> db.people.deleteOne({email: "j.west@randatmail.com" })
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

. Check to see if the document above still exists by using the following command –
people.find({email: “j.west@randatmail.com” }) and you should see nothing returned from mongo.

Finding total number of records in the collection :
db.collectionname.count for example for collection name Peoples u can check the total documents
db.peoples. count()

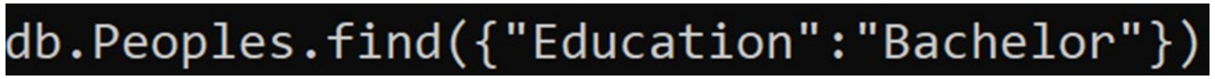


```
> db.Peoples.count()
207
```

Filtering the document

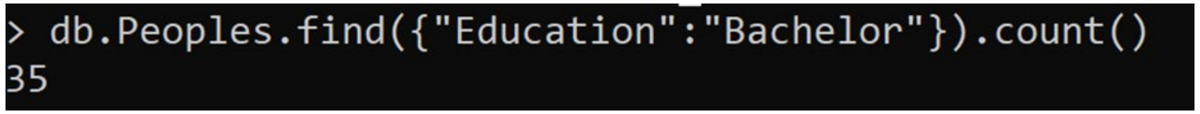
i) Filtering documents : db.collectionname.find({}) ; with the {} specified the filtering condition having field:value in Json formate for example filtering data for Education=”Bachelor”

- db.Peoples.find({“Education”:”Bachelor”}) returns all the documents having Education=Bachelor.



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

Finding total number of documents for a given criteria.
db.Peoples.find({“Education”:”Bachelor”}).count()



```
> db.Peoples.find({"Education":"Bachelor"}).count()
35
```

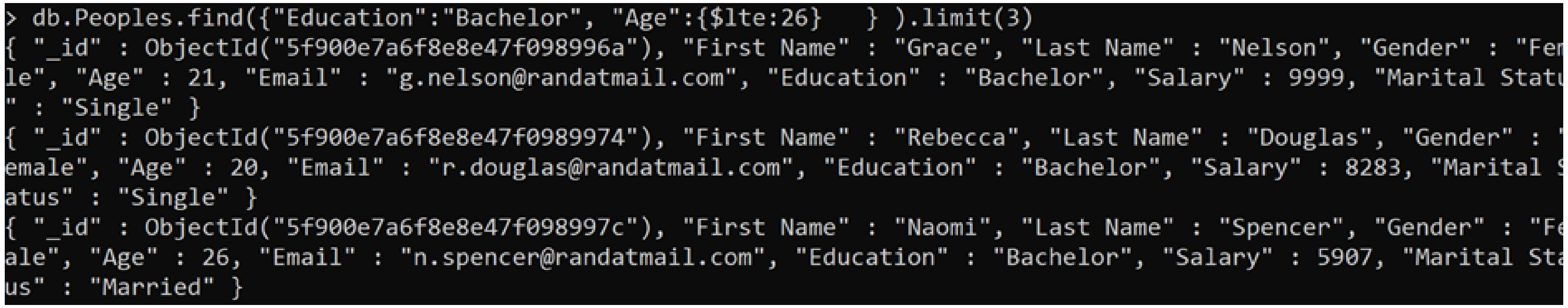
i) **Select people who have Bachelor degree and have age less than 26. It will take two fields**

- **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 ltei.e. $lte**

- db.Peoples.find({"Education":"Bachelor", "Age":{$lte:26} })

Limiting number of the documents

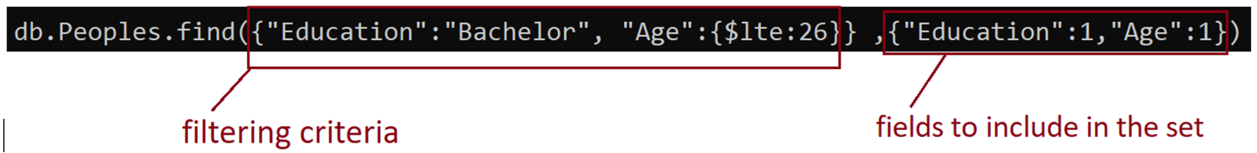
- Limiting number of the documents returned using limit method for example if you wish that only three documents should be displayed for a given filtering criteria :
- db.Peoples.find({"Education":"Bachelor", "Age":{$lte:26}   }).limit(3)



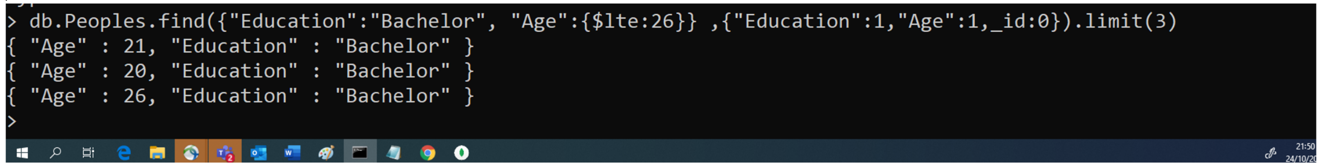
```
> db.Peoples.find({"Education":"Bachelor", "Age":{$lte:26}   } ).limit(3)
{ "_id" : ObjectId("5f900e7a6f8e8e47f098996a"), "First Name" : "Grace", "Last Name" : "Nelson", "Gender" : "Fem
le", "Age" : 21, "Email" : "g.nelson@randatmail.com", "Education" : "Bachelor", "Salary" : 9999, "Marital Statu
" : "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" }
```

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

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})
```

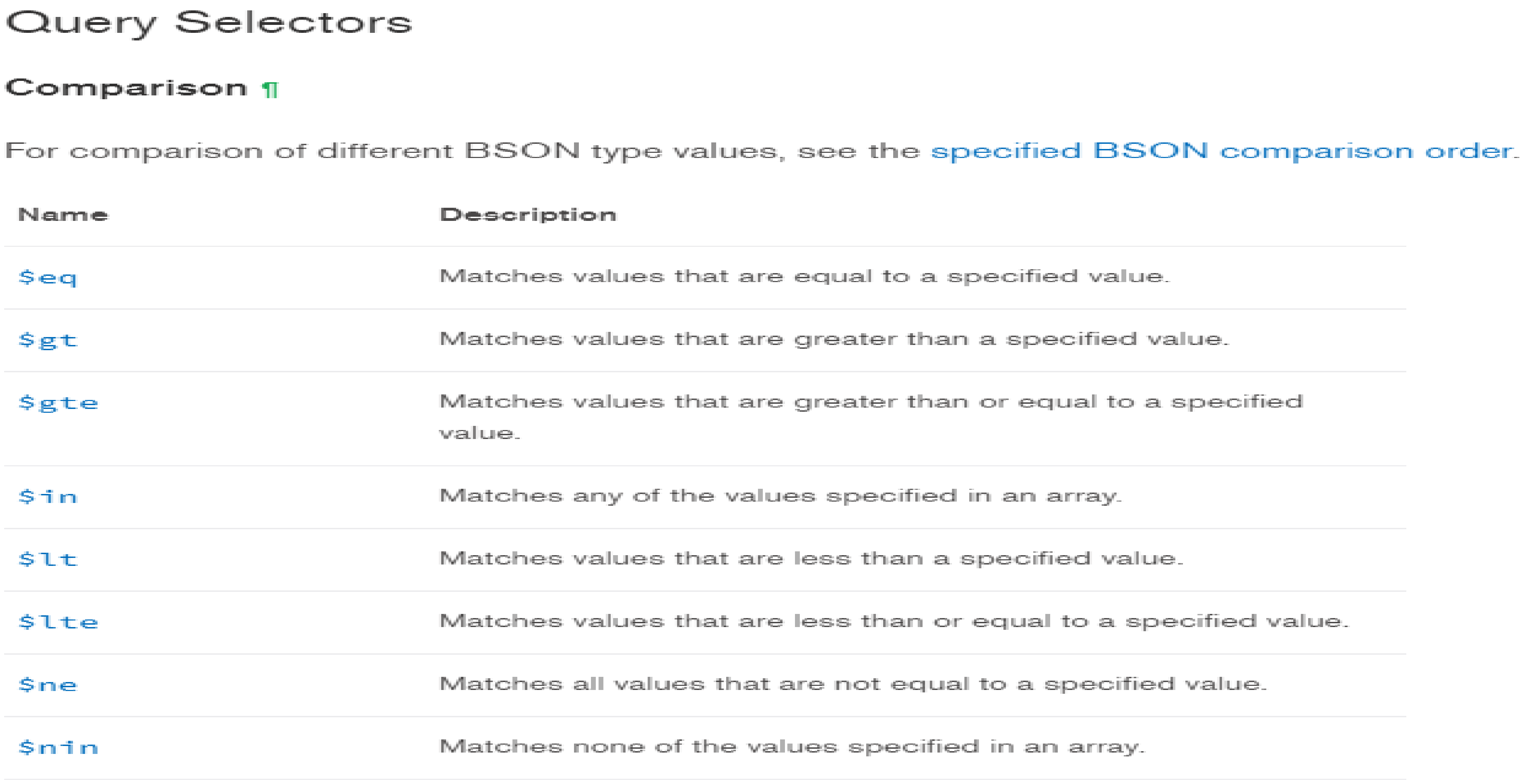


```
> db.Peoples.find({"Education":"Bachelor", "Age":{$lte:26}} ,{"Education":1,"Age":1,_id:0}).limit(3)
{ "Age" : 21, "Education" : "Bachelor" }
{ "Age" : 20, "Education" : "Bachelor" }
{ "Age" : 26, "Education" : "Bachelor" }
>
```

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

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

Query Selector



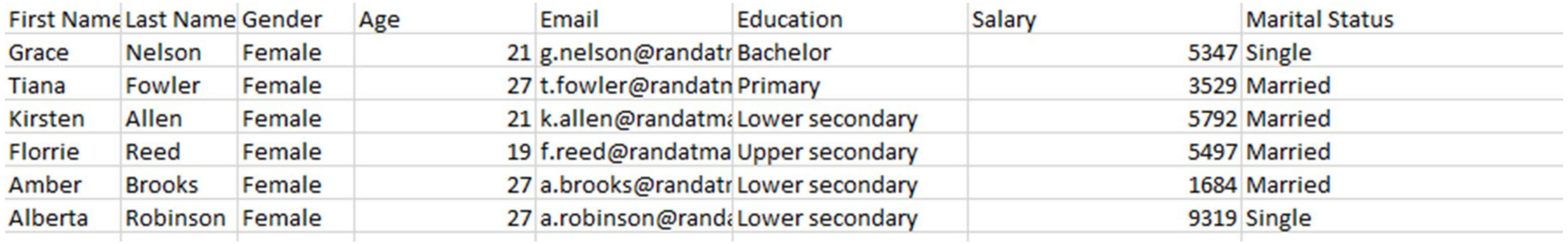
Query Selectors

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.

People CSV file



First Name	Last Name	Gender	Age	Email	Education	Salary	Marital Status
Grace	Nelson	Female	21	g.nelson@randatr	Bachelor	5347	Single
Tiana	Fowler	Female	27	t.fowler@randatn	Primary	3529	Married
Kirsten	Allen	Female	21	k.allen@randatma	Lower secondary	5792	Married
Florrie	Reed	Female	19	f.reed@randatma	Upper secondary	5497	Married
Amber	Brooks	Female	27	a.brooks@randatr	Lower secondary	1684	Married
Alberta	Robinson	Female	27	a.robinson@randa	Lower secondary	9319	Single

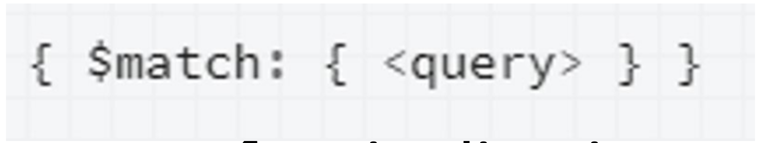
Aggregation operations in MongoDB(Group By in SQL

- Aggregation operations (Groupby) process data records and return computed results. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single purpose aggregation methods. In this tutorial we will discuss the aggregation pipeline only.

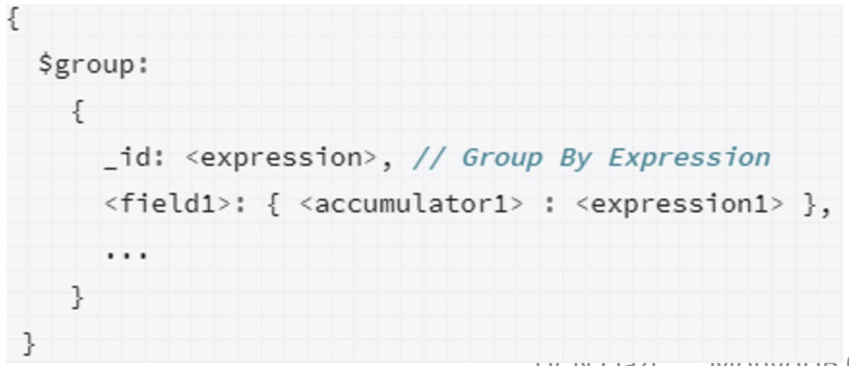
- Aggregation:
- Aggregation is performed in multiple stages to produce aggregated results from the documents. Aggregation pipelines are a composition of various stages that transform and filter the data. For example suppose we in our collection people we want to see:
- • How many people are there who has got the bachelor degree and are older than 21 years of age.
- • what is average age of Female and male in this group
- • what is the age salary of the Female and male in this group?
- • What is max age of male and female in this group
- • What is the min age of male and female in this group?
- This will be performed in two stages :
- 1. $match — This stage is filtering the people has bachelor degree and are older than 21 years
- 2. $group — This stage is grouping the people by their gender and creating the aggregate functions.

Aggregate functions:

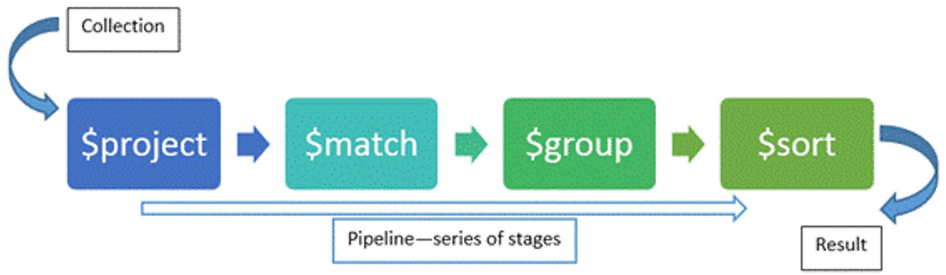
- Aggregation in MongoDB returns the computed results grouping the data from one or multiple documents.
- When calling aggregate on a collection, we pass a list of stage operators.

- $match 

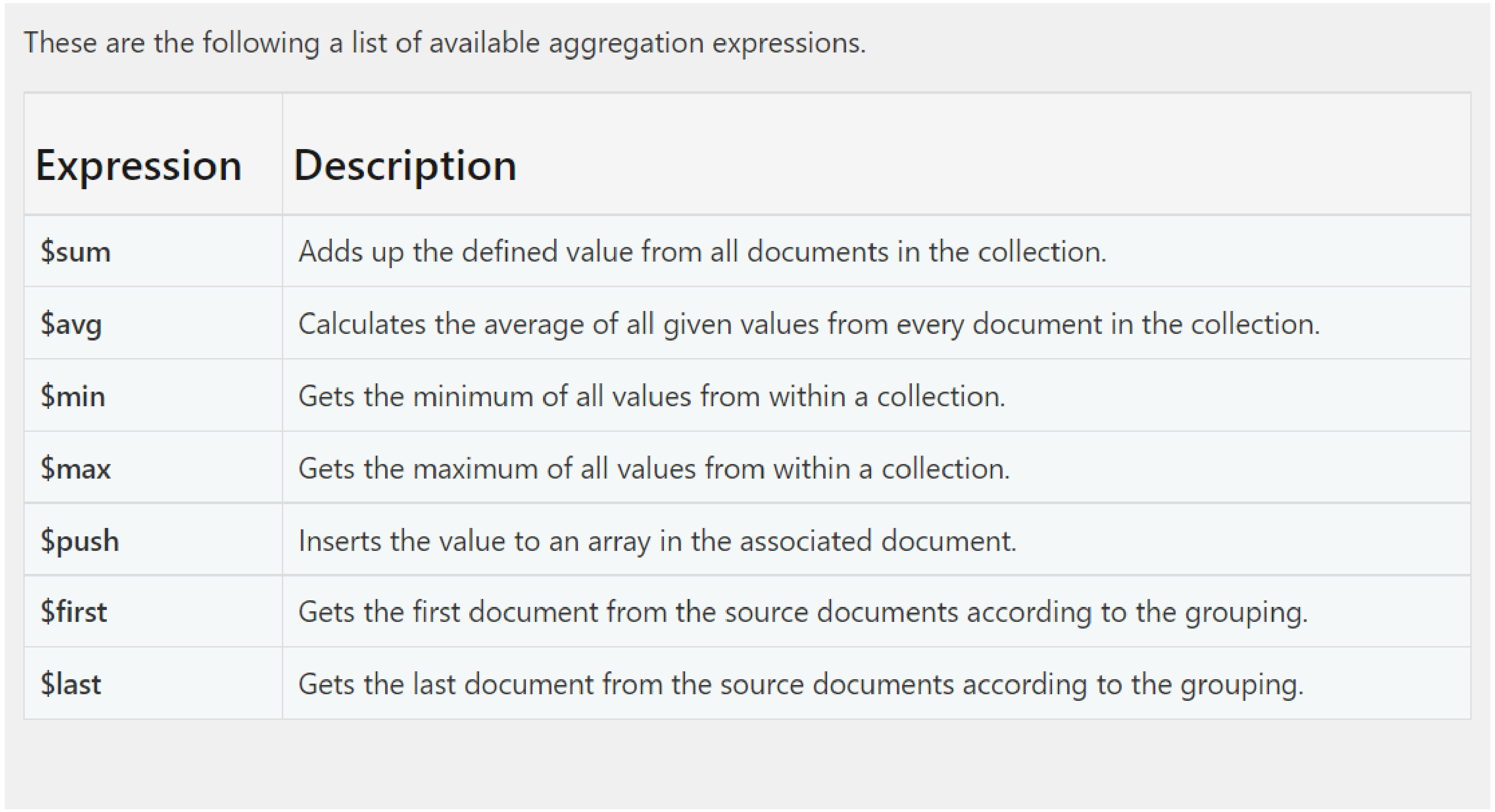
```
{ $match: { <query> } }
```

- The first stage of a pipeline is matching, and that allows to filter out the documents .which is then fed to grouping
- $group 

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



aggregation expressions.



These are the following a list of available aggregation expressions.

Expression	Description
$sum	Adds up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from every document in the collection.
$min	Gets the minimum of all values from within a collection.
$max	Gets the maximum of all values from within a collection.
$push	Inserts the value to an array in the associated document.
$first	Gets the first document from the source documents according to the grouping.
$last	Gets the last document from the source documents according to the grouping.

Counting on aggerate

- Count all the female grouping on their Education
- db.Peoples.aggregate(
- [
- { $match: { Gender: "Female" } },
- {
- $group: { _id: "$Education", count: { $sum: 1} }
- }
-])
- Here count is the name of text , $sum :1 to sum every row.

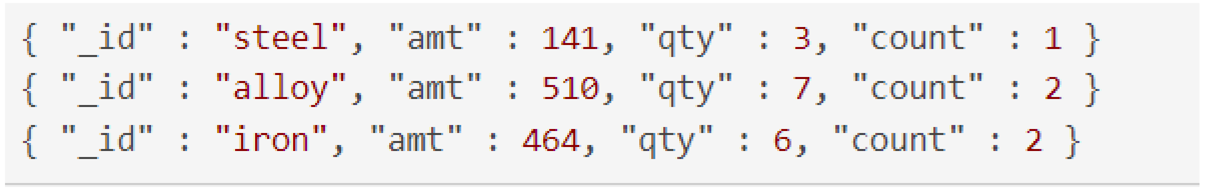
Aggregate : Average Salary of the female grouping education

- db.Peoples.aggregate([
- { $match: { Gender: "Female" } },
- { $group: {
- _id: "$Education", Average_salary: { $avg:"$Salary"}
- }
- }
-])

Another Example

- **Suppose we have the following 'product' collection.**
- **db.product.insert([**
- **{'item': 'iron', 'price': 120, 'quantity': 2},**
- **{'item':'alloy', 'price':90, 'quantity': 1},**
- **{'item':'steel', 'price':47, 'quantity': 3},**
- **{'item':'alloy', 'price':70, 'quantity': 6},**
- **{'item':'iron', 'price':56, 'quantity':4}**
- **]);**
- **Find sum of the price and quantity**

```
db.product.aggregate(
[
{ "$group" :
{ "_id": "$item",
amt: {$sum: {$multiply: ["$price", "$quantity"]}},
qty: {$sum: "$quantity"}, count: {$sum: 1}
}
}]);
```



```
{ "_id" : "steel", "amt" : 141, "qty" : 3, "count" : 1 }
{ "_id" : "alloy", "amt" : 510, "qty" : 7, "count" : 2 }
{ "_id" : "iron", "amt" : 464, "qty" : 6, "count" : 2 }
```

Write a query for average product amount and average quantity.

Solution

- db.product.aggregate(
- [{
-         "$group" : {
-                 "_id": "$item",
-                 avgAmt: { $avg: {$multiply: ["$price", "$quantity"]}},
-                 avgQty: { $avg: "$quantity"}
-         }
- }]
-)

Aggerate without match

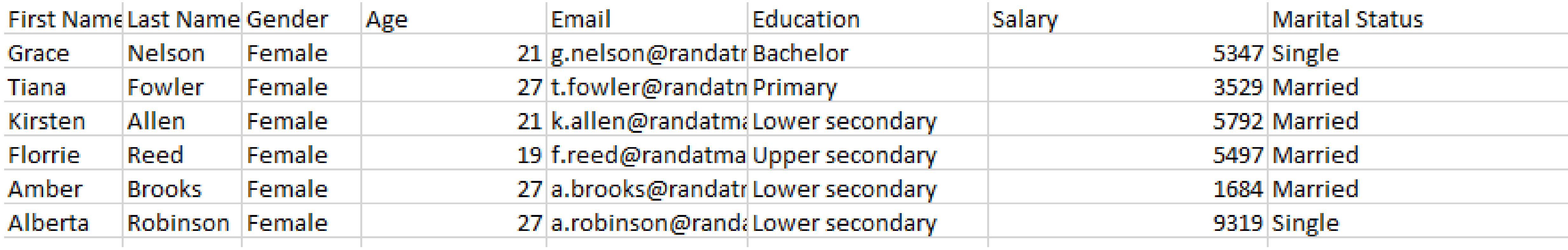
- Find total numbers of married and unmarried
- db.Peoples.aggregate([{ $group: { _id: "$Marital Status", count: { $sum: 1} } }])
- Find total married and unmarried for each of Education group.
- db.Nhqazi.aggregate([{ $group: { _id: {Edu:"$Education",Msts:"$Marital Status"}, count: { $sum: 1} } }])

MongoDB Compass

- What is MongoDB Compass
  - The GUI for MongoDB. Visually explore your data. Run ad hoc queries in seconds. Interact with your data with full CRUD functionality. View and optimize your query performance. Available on Linux, Mac, or Windows.
- Connecting mongo DB with mongo dB compass and perform CRUD queries without any programming
  - **Inserting a document**
  - **Updating a document**
  - **Deleting a document**
  - **Aggerate pipe line**

Lab Work

- Given a CSV file
- Import it using MongoDB compass. Instruction is provided in separate file/video
- Apply mongoDB crude Commands for insertion deletion update and aggerate function using compass and command line .
- The CSV file has following data:



First Name	Last Name	Gender	Age	Email	Education	Salary	Marital Status
Grace	Nelson	Female	21	g.nelson@randatr	Bachelor	5347	Single
Tiana	Fowler	Female	27	t.fowler@randatn	Primary	3529	Married
Kirsten	Allen	Female	21	k.allen@randatma	Lower secondary	5792	Married
Florrie	Reed	Female	19	f.reed@randatma	Upper secondary	5497	Married
Amber	Brooks	Female	27	a.brooks@randatr	Lower secondary	1684	Married
Alberta	Robinson	Female	27	a.robinson@randa	Lower secondary	9319	Single