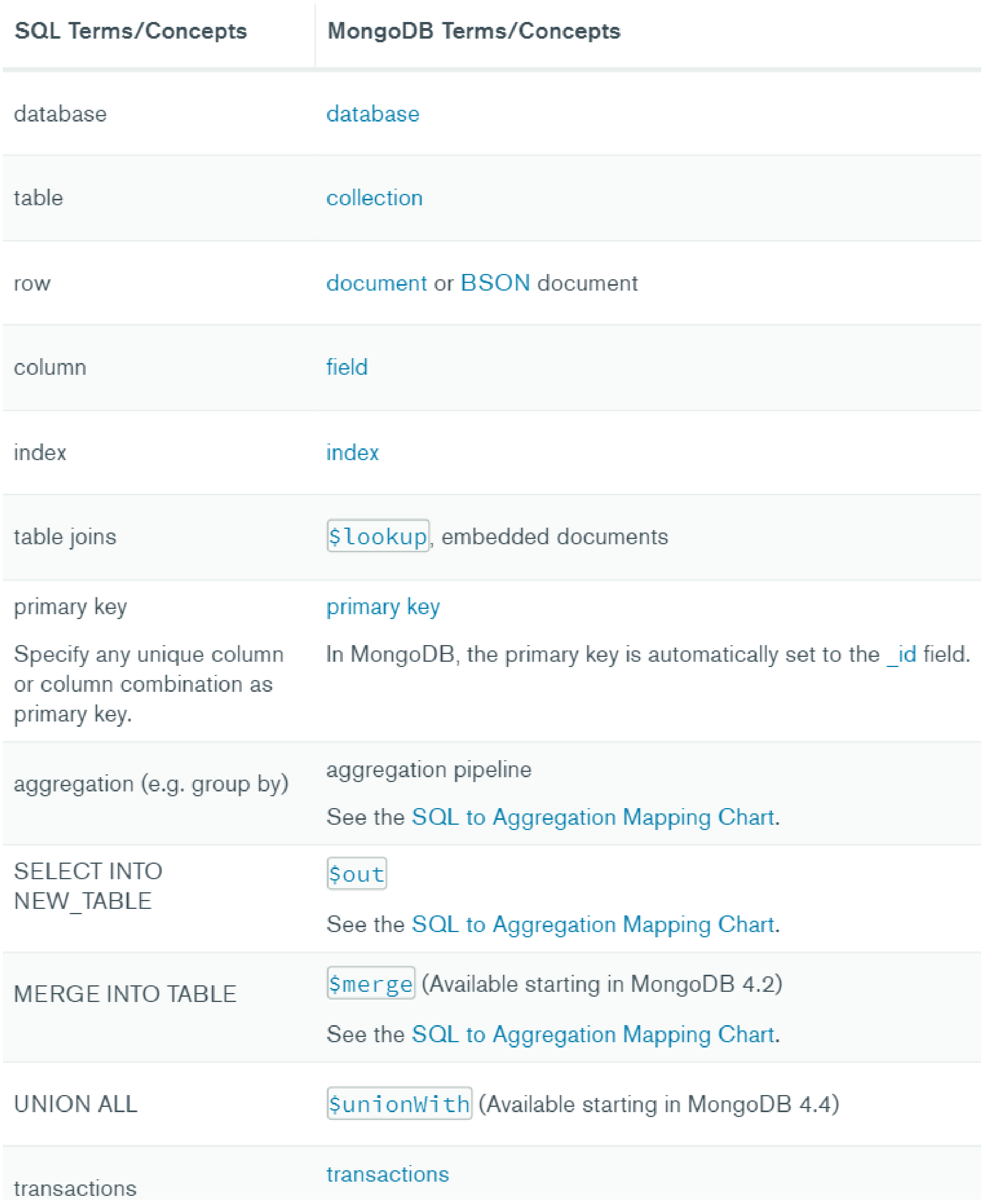
MONGO DB

mongod.exe --dbpath "C:\data"

Mongo.exe

Comparison between SQL and MongoDB (No SQL)-



SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	`$lookup`, embedded documents
primary key Specify any unique column or column combination as primary key.	primary key In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.
SELECT INTO NEW_TABLE	`$out` See the SQL to Aggregation Mapping Chart.
MERGE INTO TABLE	`$merge` (Available starting in MongoDB 4.2) See the SQL to Aggregation Mapping Chart.
UNION ALL	`$unionWith` (Available starting in MongoDB 4.4)
transactions	transactions

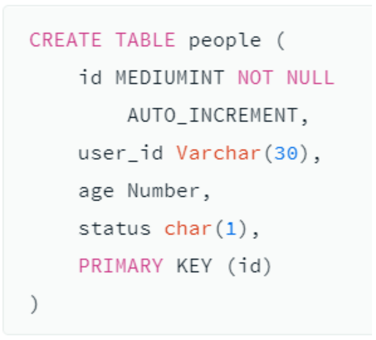
Show databases-

- show dbs
- OR show databases

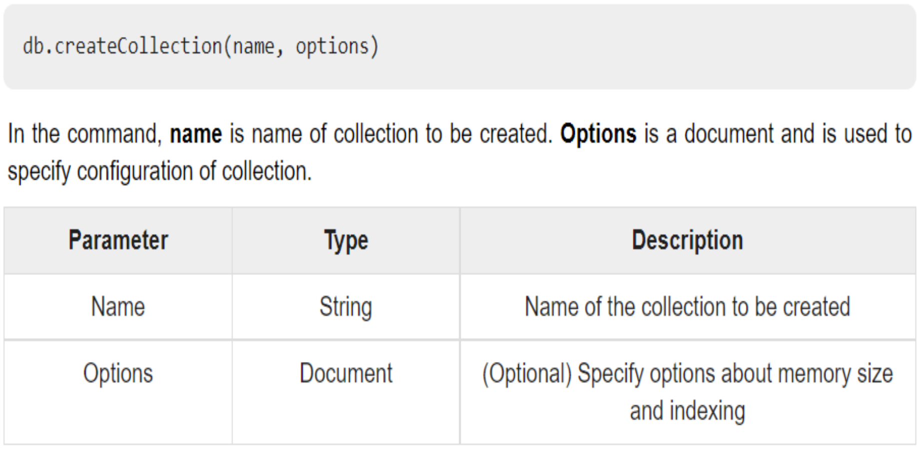
Select database-

- Use mydb

CREATE table (Collection)-



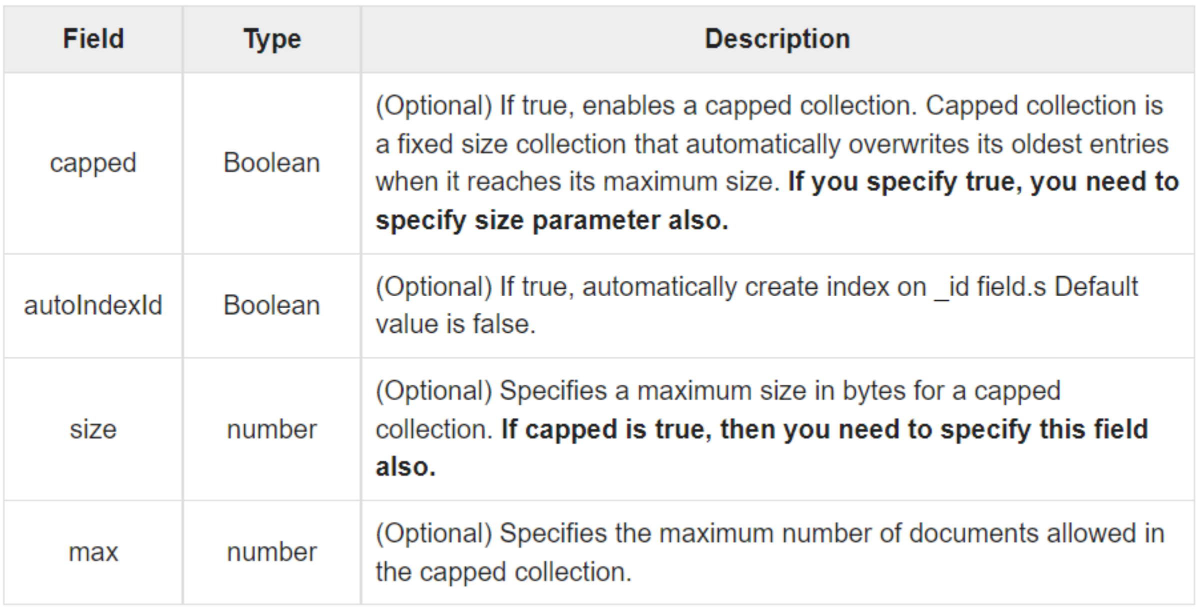
```
CREATE TABLE people (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```



```
db.createCollection(name, options)
```

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing



Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

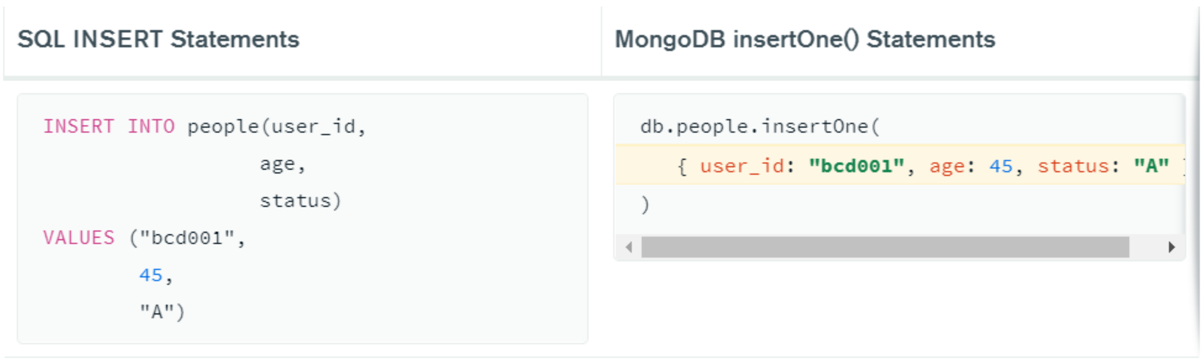
- db.createCollection("bajaj", {capped:true, autoIndexID: true, size: 6142800, max:10000})

DROP-

```
- db.COLLECTION_NAME.drop()
```

- show collections

INSERT-



SQL INSERT Statements	MongoDB insertOne() Statements

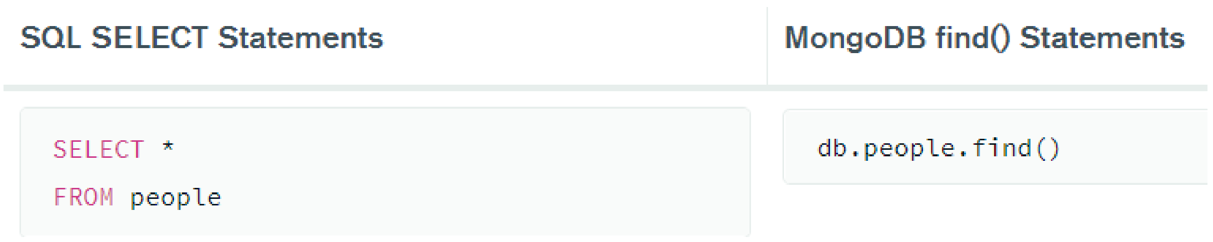
```
INSERT INTO people(user_id,
                  age,
                  status)
VALUES ("bcd001",
        45,
        "A")
```

```
db.people.insertOne(
   { user_id: "bcd001", age: 45, status: "A"
)
```

```
>db.COLLECTION_NAME.insert(document)
```

- db.bajaj.insertOne({ empid:10, empname:"Shashank" })
- db.bajaj.insertMany([{ empid:11, empname:"Mihir" }, { empid:12, empname:"Pranv" }])

SELECT OR FIND-



SQL SELECT Statements	MongoDB find() Statements

```
SELECT *
FROM people
```

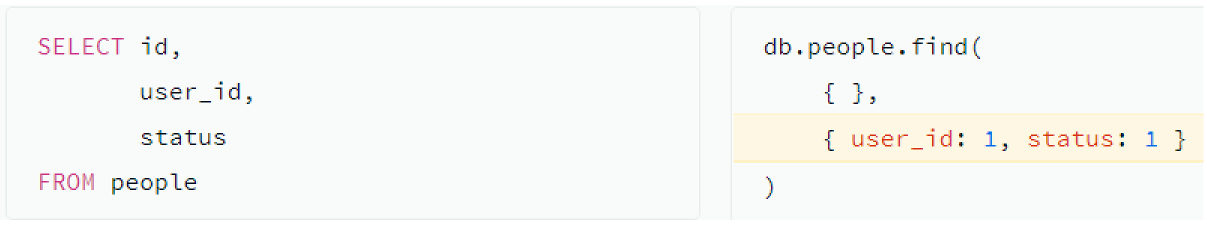
```
db.people.find()
```

```
>db.COLLECTION_NAME.find()

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

- db.bajaj.find()
- db.bajaj.find.pretty()

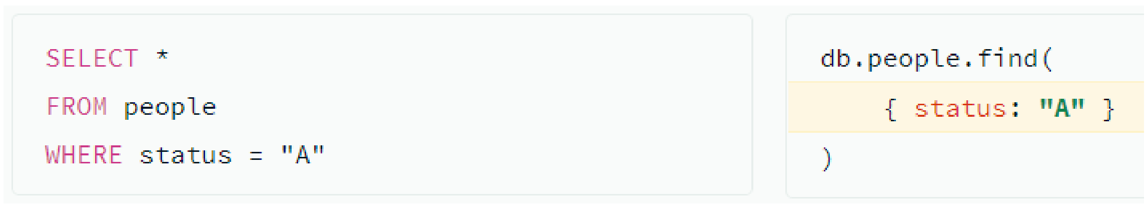
```
>db.COLLECTIONNAME.findOne()
```



```
SELECT id,
       user_id,
       status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1, status: 1 }
)
```

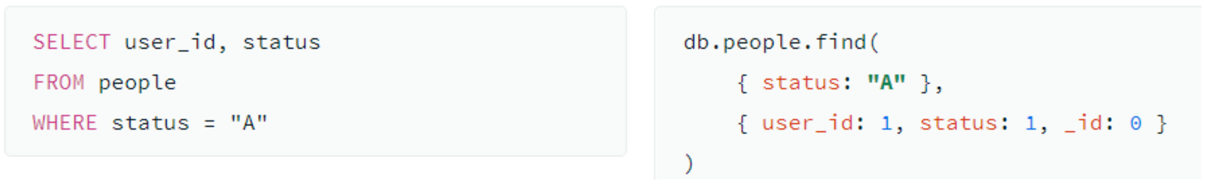
- db.bajaj.find({ }, {empid:1, empname:1})



```
SELECT *
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" }
)
```

- db.bajaj.find({empid:10})

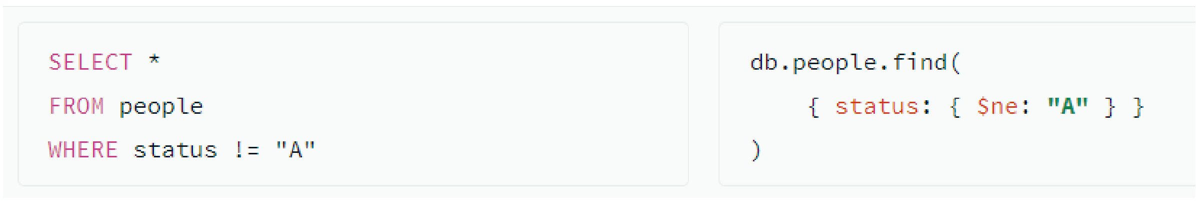


```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }
)
```

- db.bajaj.find({empid:10}, {empid:1, empname:1})

NOT EQUAL-

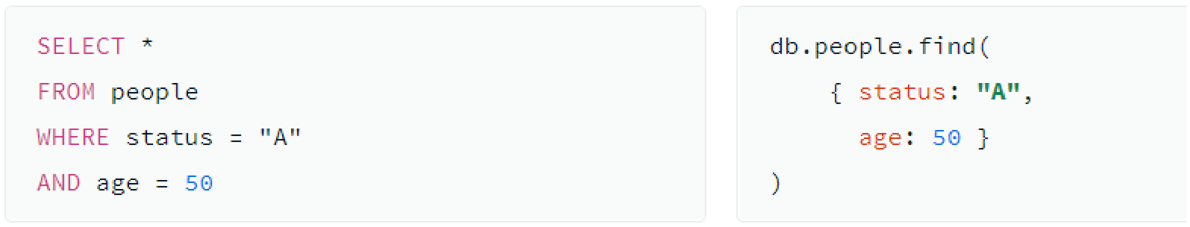


```
SELECT *
FROM people
WHERE status != "A"
```

```
db.people.find(
    { status: { $ne: "A" } }
)
```

- db.bajaj.find({empid: {$ne:10}})

AND-



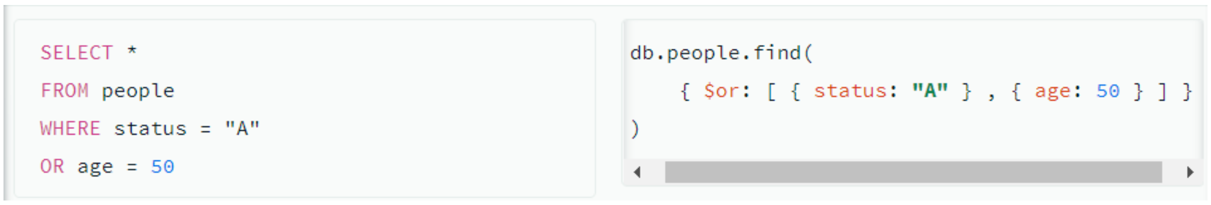
```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

- db.bajaj.find({empid:10, empname:"Shashank"})

- db.bajaj.find({$and: [{empid: {$ne: 10}}, {empid: {$ne: 11}}]})

OR-

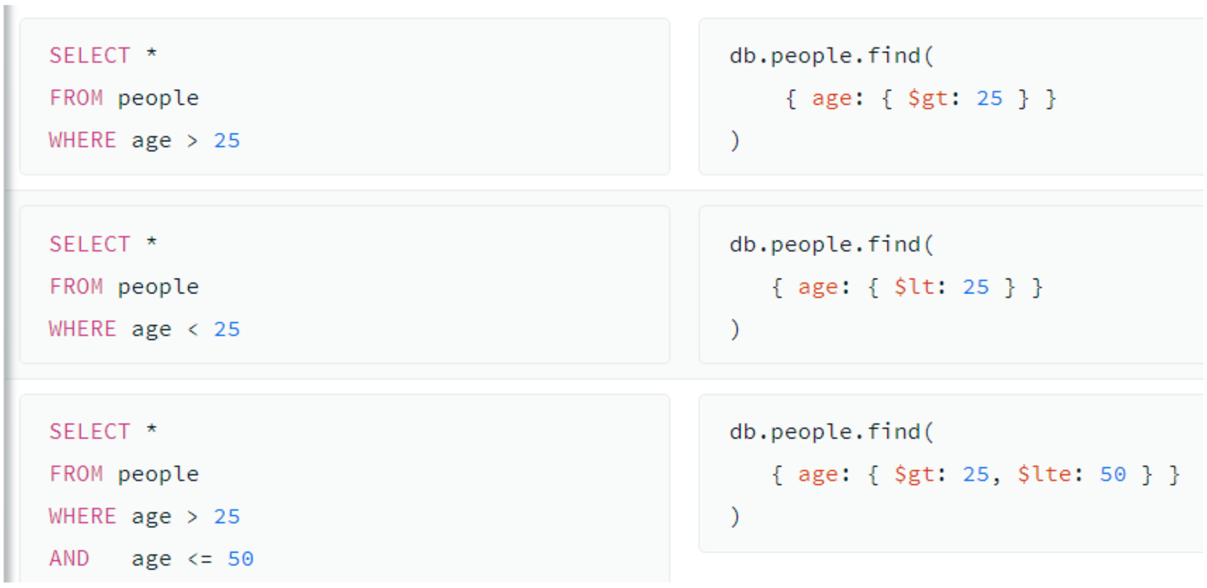


```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
    { $or: [ { status: "A" } , { age: 50 } ] }
)
```

- db.bajaj.find({$or: [{empid: {$ne: 10}}, {empid: {$ne: 11}}]})
- db.bajaj.find({$or: [{empid:10}, {empid:11}]})

GREATER THAN/LESS THAN-



```
SELECT *
FROM people
WHERE age > 25
```

```
db.people.find(
    { age: { $gt: 25 } }
)
```

```
SELECT *
FROM people
WHERE age < 25
```

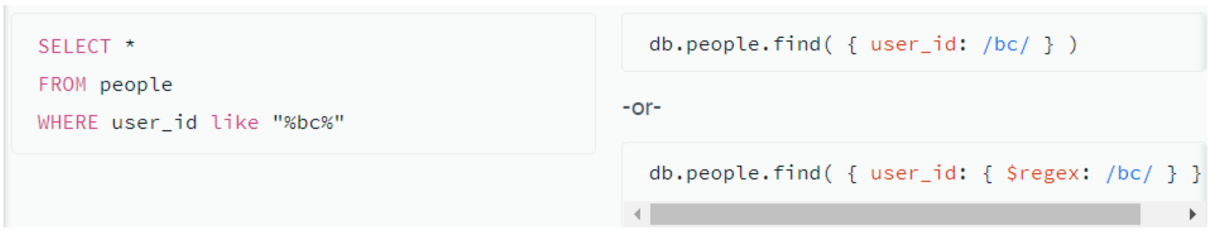
```
db.people.find(
   { age: { $lt: 25 } }
)
```

```
SELECT *
FROM people
WHERE age > 25
AND   age <= 50
```

```
db.people.find(
   { age: { $gt: 25, $lte: 50 } }
)
```

- db.bajaj.find({empid: {$gt: 11}})
- db.bajaj.find({empid: {$gt: 10, $lte: 12}})

LIKE-



```
SELECT *
FROM people
WHERE user_id like "%bc%"
```

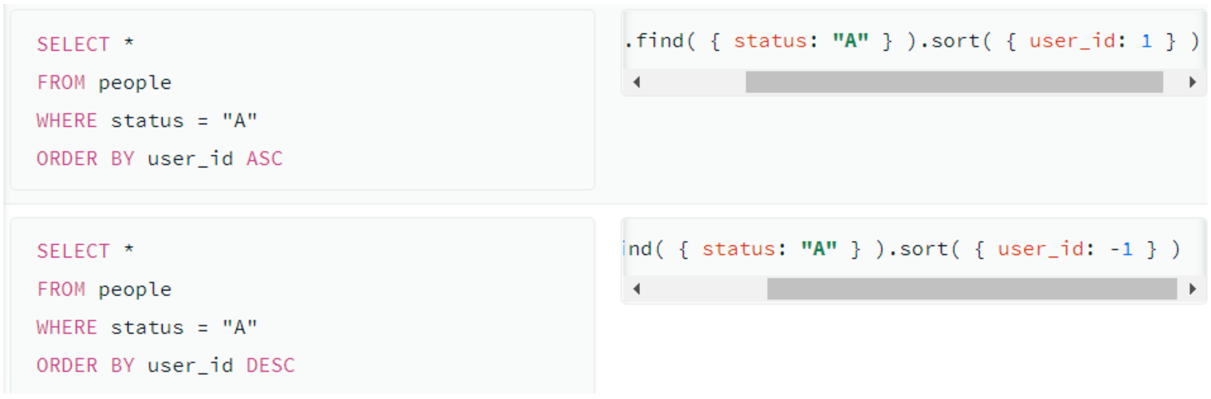
```
db.people.find( { user_id: /bc/ } )
```

-or-

```
db.people.find( { user_id: { $regex: /bc/ } }
```

- db.bajaj.find({empname: /sha/})
- db.bajaj.find({empname: {$regex: /sha/}})

ORDER BY-



```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

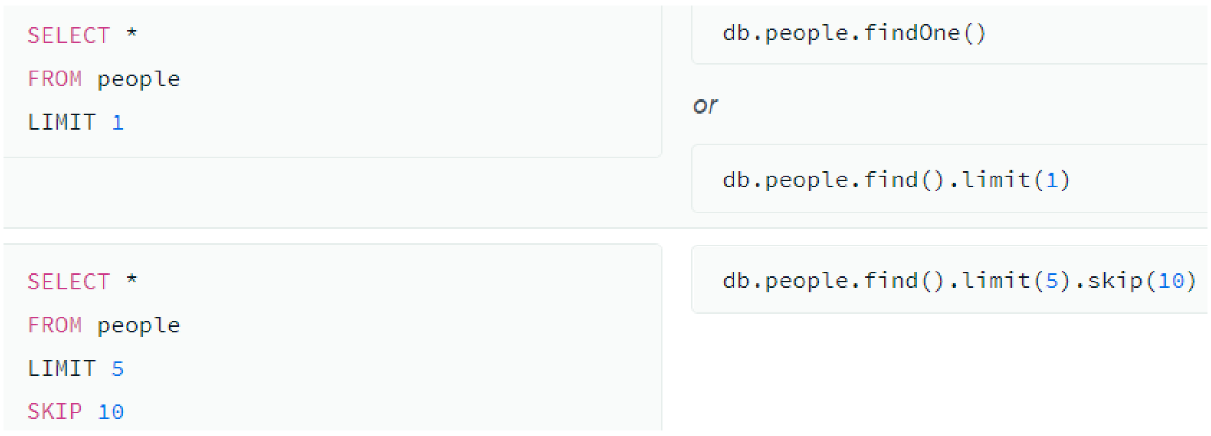
```
.find( { status: "A" } ).sort( { user_id: 1 } )
```

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
```

```
nd( { status: "A" } ).sort( { user_id: -1 } )
```

- db.bajaj.find().sort({empid:1})
- db.bajaj.find().sort({empid:-1})

LIMIT/FIND-



```
SELECT *
FROM people
LIMIT 1
```

```
db.people.findOne()
```

or

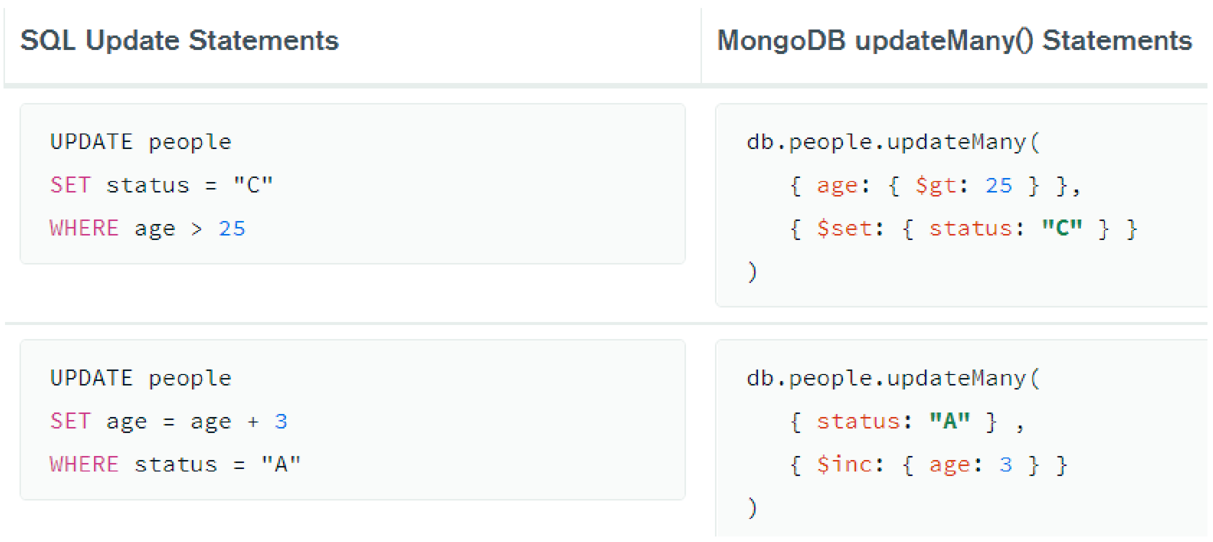
```
db.people.find().limit(1)
```

```
SELECT *
FROM people
LIMIT 5
SKIP 10
```

```
db.people.find().limit(5).skip(10)
```

- db.bajaj.findOne()
- db.bajaj.find().limit(1)
- db.bajaj.find().limit(1).skip(2)

UPDATE-

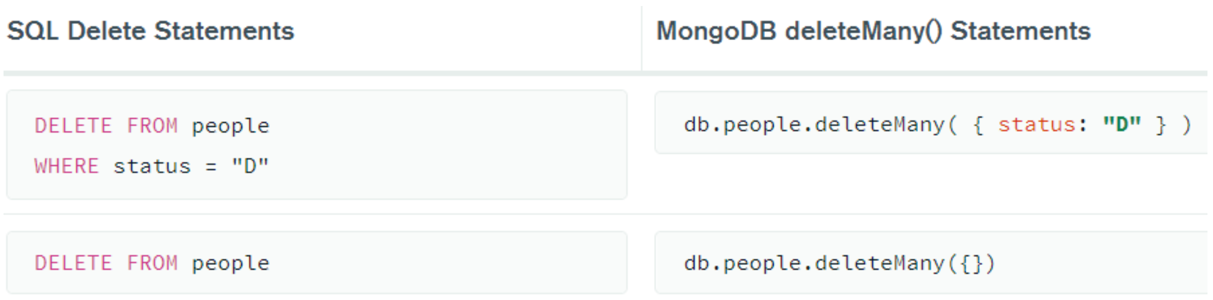


SQL Update Statements	MongoDB updateMany() Statements
`UPDATE people SET status = "C" WHERE age > 25`	`db.people.updateMany( { age: { $gt: 25 } }, { $set: { status: "C" } } )`
`UPDATE people SET age = age + 3 WHERE status = "A"`	`db.people.updateMany( { status: "A" } , { $inc: { age: 3 } } )`

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

- db.bajaj.updateMany({empid:13}, {$set:{salary:2000}})
- db.bajaj.updateMany({empid:13}, {$inc:{salary:2000}})

DELETE-

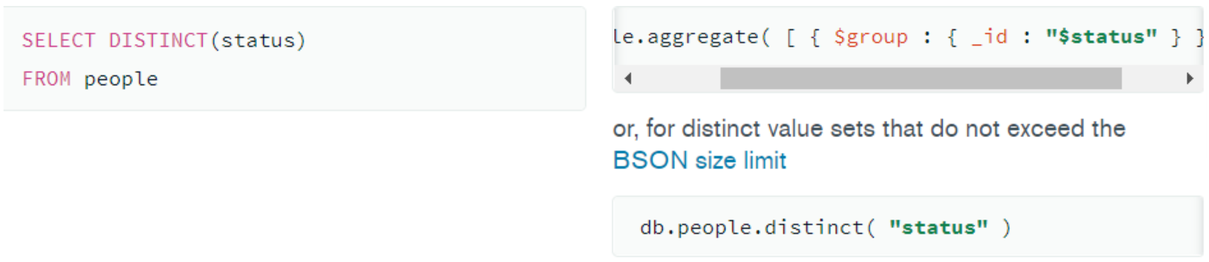


SQL Delete Statements	MongoDB deleteMany() Statements
`DELETE FROM people WHERE status = "D"`	`db.people.deleteMany( { status: "D" } )`
`DELETE FROM people`	`db.people.deleteMany({})`

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

AGGREGATE FUNCTIONS-

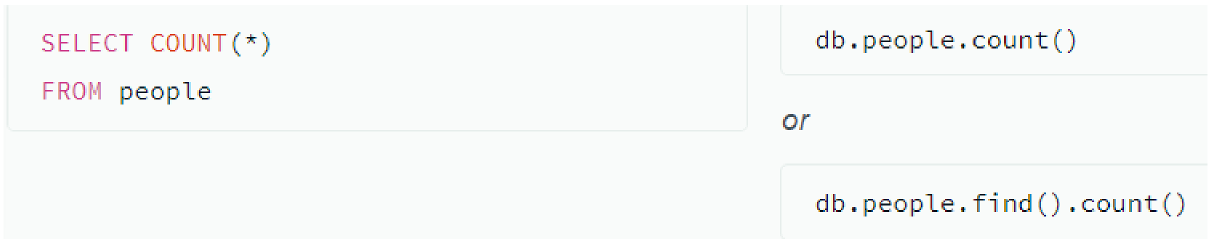
DISTINCT-



`SELECT DISTINCT(status) FROM people`	`le.aggregate( [ { $group : { _id : "$status" } }`
	 or, for distinct value sets that do not exceed the BSON size limit
	 `db.people.distinct( "status" )`

- db.bajaj.distinct("empid")

COUNT-



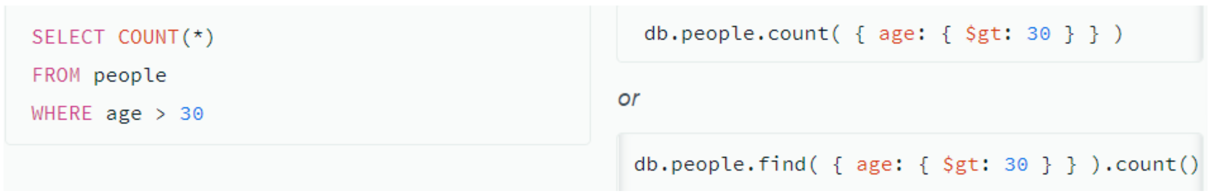
```
SELECT COUNT(*)
FROM people
```

```
db.people.count()
```

or

```
db.people.find().count()
```

- db.bajaj.find().count() -> will give only count



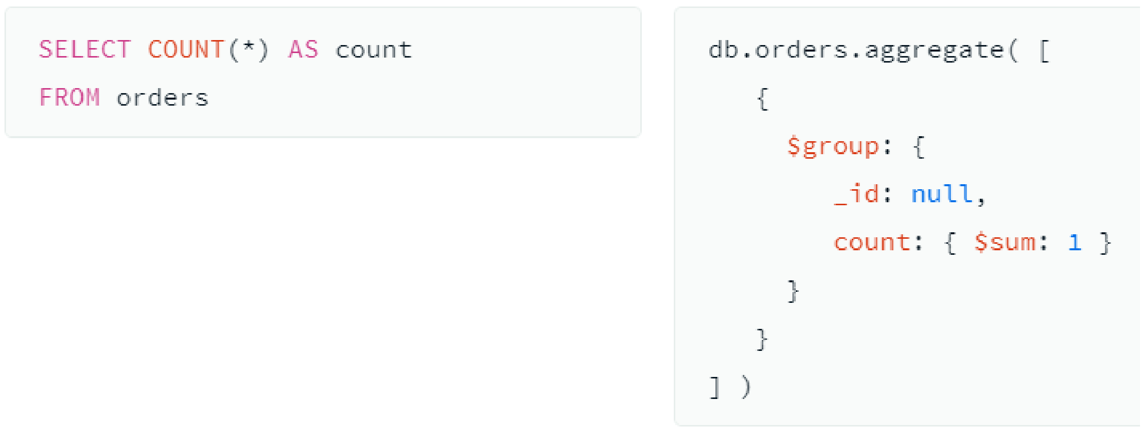
```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

```
db.people.count( { age: { $gt: 30 } } )
```

or

```
db.people.find( { age: { $gt: 30 } } ).count()
```

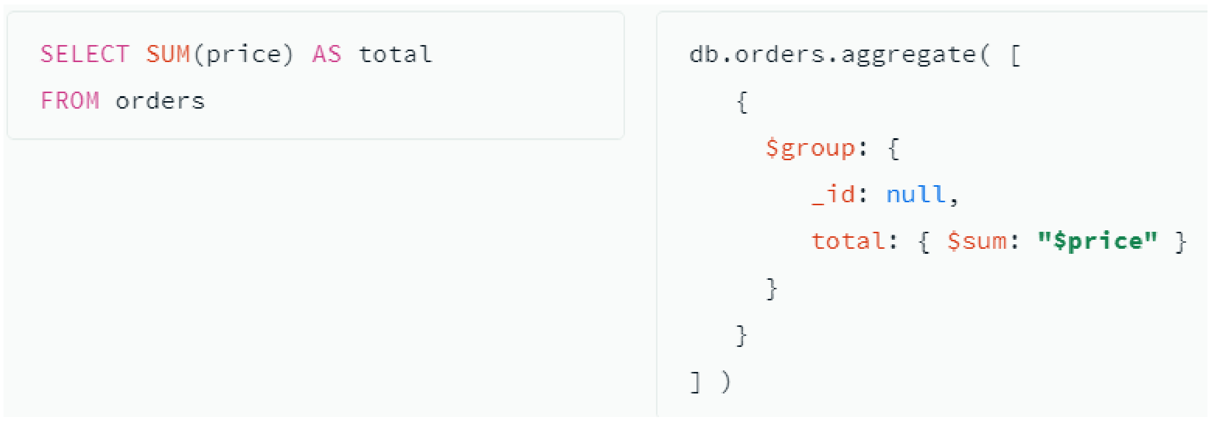
- db.bajaj.find({empid: {$gt:11}}).count() -> only count (select count(*))



```
SELECT COUNT(*) AS count
FROM orders
```

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

- db.bajaj.aggregate([{ $group: {_id:null, count: {$sum: 1}}}])

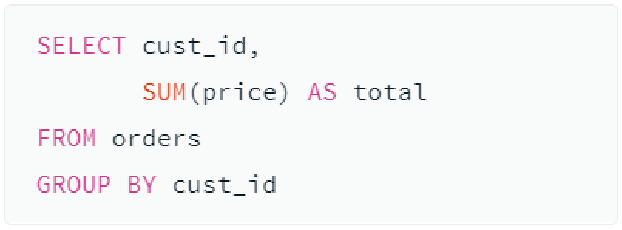


```
SELECT SUM(price) AS total
FROM orders
```

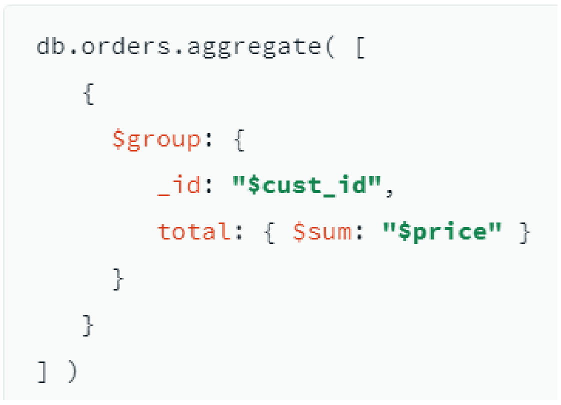
```
db.orders.aggregate( [
   {
     $group: {
        _id: null,
        total: { $sum: "$price" }
     }
   }
] )
```

- db.bajaj.aggregate([{$group: { _id:null, total: {$sum: "$empid"}}}])
-

GROUP BY-

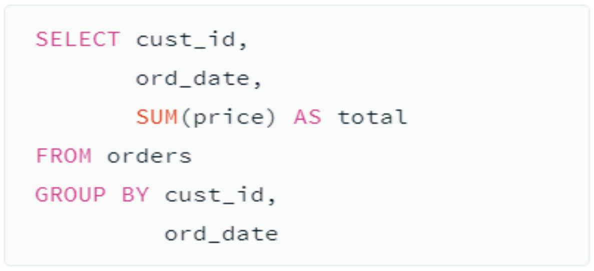


```
SELECT cust_id,
       SUM(price) AS total
FROM orders
GROUP BY cust_id
```

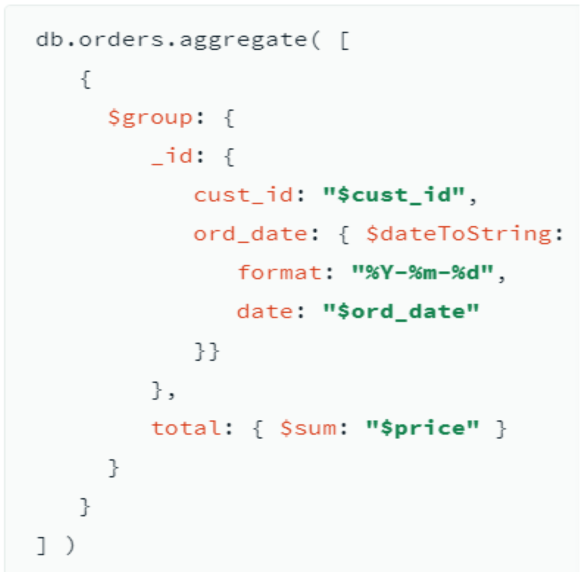


```
db.orders.aggregate( [
   {
     $group: {
        _id: "$cust_id",
        total: { $sum: "$price" }
     }
   }
] )
```

- db.bajaj.aggregate([{$group: {_id: "$empid", total: {$sum: "$empid"}}}])



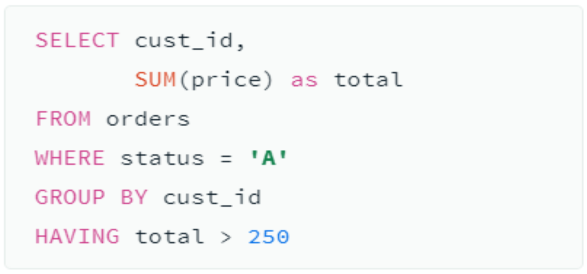
```
SELECT cust_id,
       ord_date,
       SUM(price) AS total
FROM orders
GROUP BY cust_id,
         ord_date
```



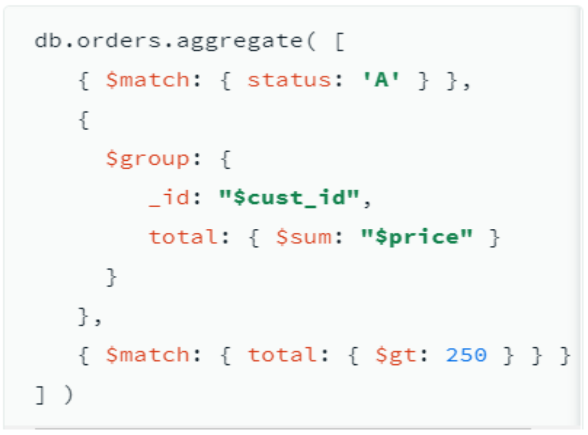
```
db.orders.aggregate( [
   {
     $group: {
        _id: {
           cust_id: "$cust_id",
           ord_date: { $dateToString: {
              format: "%Y-%m-%d",
              date: "$ord_date"
           }}
        },
        total: { $sum: "$price" }
     }
   }
] )
```

- db.bajaj.aggregate([{$group: {_id: {empid:"$empid", empname:"$empname"}, total: {$sum: "$empid"}}}])

HAVING-



```
SELECT cust_id,
       SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
HAVING total > 250
```



```
db.orders.aggregate( [
   { $match: { status: 'A' } },
   {
     $group: {
        _id: "$cust_id",
        total: { $sum: "$price" }
     }
   },
   { $match: { total: { $gt: 250 } } }
] )
```