MongoDB

what is mongoDB? open source data management system. NoSQL it is not the same as other DBMSs instead of storing data in tables, mongoDB orgnizes data as documents.

Instead of tables and rows the MongoDB structure consists of collections and document.

MongoDB features: data can be stored without defining a previous structure, Store in document form (json-like form).

SQL	NoSQL (MongDB)
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index

Collections

Show dbs or show databases To show all dbs you have

To create a database use database_name we can also use 'use' to move between databases

db to identify which db you are using

db.dropDatabase() it will drop the database you are currently using

Show collections to show the collections of the current db

db.createCollection("Collection_name") to create a new collection in the curren db

db.collection_name.drop() drop a collection

Documents (CRUD)

`db.collection_name.insertOne({Name:"s", Age:0})` add one document to a collection

`db.collection_name.insertMany([{Name:"b"},{Name:"c"},{Name:"d"}])` add many documents to a collection

`db.collection_name.find({Condition})` if you didn't specifi any conditions all of the document in the collection would be displayed

for example({Age:15}) it would display documents with the age = 15

`db.collection_name.find({}).pretty()` Pretty is used to display organized documents

`db.collection_name.findOne({})` return the first document in the collection (you can add a condition

Condition

`db.collection_name.find({},{Name:1,_id:0})` 1→ to display 0→ to hide id is displayed by defult you can hide it by giving it 0

Projection

`db.collection_name.updateOne({Name:"s"},{$set:{Age:20}})` updating one value

Condition — new value

`db.collection_name.updateMany({Name:"s"},{$set:{Age:20}})` updating more than one document value

we can use {$inc:{Age:-2}} in update to decrease age by 2 or use a positive number to increase the age

There is also {$unset:{Age:}} to delete the age value

Condition

`db.collection_name.deleteOne({Name:"b"})` deleting a document

Condition

`db.collection_name.deleteMany({Name:"b"})` deleting all document that matches the condition

Aggregation

Some aggregatio examples :

1. count : db.Collection_name.aggregate([{$count: "name" }]) → its only for display won't be saved in the collection → for all types of aggregation

Count how many documents in a collection

2- sort : db.Collection_name.aggregate([{$sort : "name: 1" }])

if 1 sort Asc (A-Z, 1-9) if -1 sort Desc (Z-A, 9-0)

3- limit : db.Collection_name.aggregate([{$limit : "3" }]) → number of documents you want to display it will display the first 3 documents

4- addfields : db.Collection_name.aggregate([{$addFields : { name : {$Sum : "$ name"}} }])

new field name

the name of the field you want to sum for each document

5- sortByCount : to sort and groub documents with the same value , then cant how many documents in each group

6- set : to count the total of a spesific document

7- unset : to exclude a field

8- sample : select a random documen from the collection

9- Project : if the document is large and we only need few data we can select want to display "0" for no "1" for yes

10- out : used at the end of the code to store the results in a new collection

11- skip : you can select how many documents you want to skip

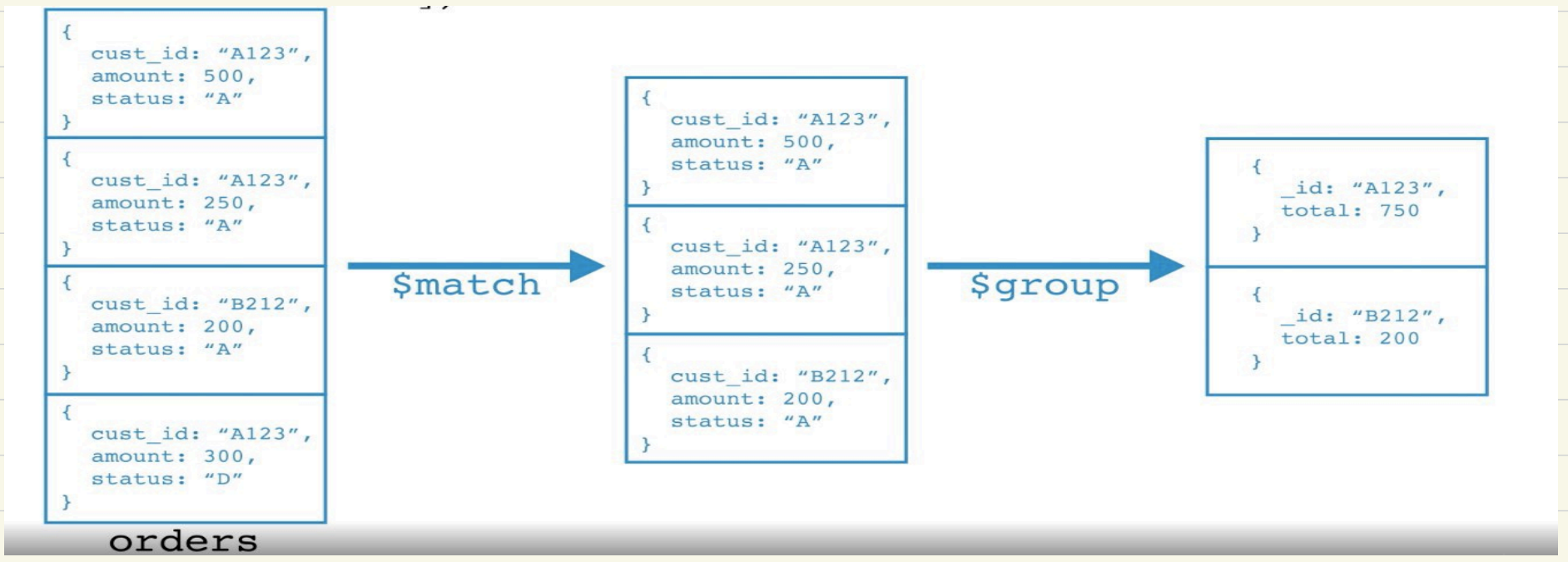
12- group : group documents results or for total

14- lookup : merge 2 collections

```
db.posts.aggregate([{$lookup:{from:'comments', localField: 'title',
foreignField:'Post Title', as :'comments'}}]).Pretty()
```

aggregation pipeline : db.orders.aggregate([

```
{$match: { status: "A" }},
{$group: { _id: "$cust_id", total: { $sum: "$amount" }}} ])
```



Relations

3 Types of relations:

1- one to one (one person has one car) there are two ways to connect

A - Refrenced: using first document ID in second

```
db.users.insertOne({street: "b", city: "Riyadh", user_id: ObjectId("5f")})
```
the second document ID

B - Embeded: inserting a document content in the other (results of one document no need of the other collection document field)

```
db.users.insertOne({name: "Rana", age: 30, adresse: {street: "b", city: "Riyadh"}})
```
second document

2- one to many (one user has more than one post) there are two ways to connect

A - Normalization: Simelar to refrenced using the ID of anthore document

```
{ _id: "5db", title: 'P', comments: ["5cc", "5aa"] }
```
array of IDs

B - Denormalization: similar to Embeded but in array

```
{ _id: "5db", title: 'P', comments: [{username: "s", text: "bbb"}, {username: "b", text: "ccc"}]}
```
array

3- many to many (one user can follow many users and users can follow many users back)

To connect collections with many to many relationship we need a new collection and use the two collections documents ids

```
db.follow.insertMany([{following: ObjectId("f1e"), follower: ObjectId("f1f")},
{following: ObjectId("f1e"), follower: ObjectId("f20")},
{following: ObjectId("f20"), follower: ObjectId("f1e")}])
```
S follows A

S follows B

B follows S

* We can use compass insted of typing code in mongosh (mongoshell)

you can export collections as json or csv

also you can import collections as json or csv