**MongoDB**

It is a new philosophy database

Structure of MongoDB

Database Shop-Site

Collections Users Orders

Documents {name:`Davi`,Age:20} {...}

{name:`David`,Age:22} {...}

{name:`Dravid`} {...}

It is also Schema-less, it is not defined how many entries we can make randomly inside an document,

we can have only `name` or we can have `name`,`password`,`address` ,it is not defined or not strict-ed

to use same structure of entries in Documents.

SQL is different is must define every type of entry

**Data stored**

In Document is BSON (**Binary JSON**) Format in background, but we will be using JSON

**Relations in No SQL**

1. Embedded Document/Nested

Here if one collection is dependent on

other collections we do not use `id` we

can embed data.

embed data means to directly copy the

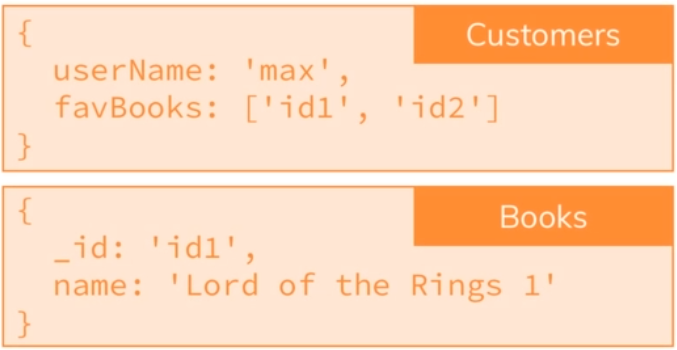
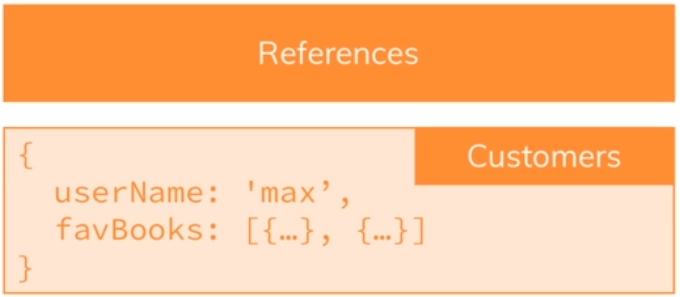
data into parent document



1. In some cases the data redundancy is more,

so in this case the data need to referred

from parent document to child document.



Embedding Data

To

Referencing Data via `id`

**Setup MongoDB in your cluster**

$ kubectl run mongorun --image mongo --env=`MONGO\_INITDB\_ROOT\_USERNAME=admin` --env`MONGO\_INITDB\_ROOT\_PASSWORD=password` --port=27017

login to Mongo via MongoDB shell as admin

(mongosh)

$ kubectl exec -it pod/mongorun -- mongosh -u admin -p password

or

$ kubectl exec -it pod/mongorun -- mongosh

Now Initiate Primary and Secondary cluster system

via Readme - rs\_initiate**.txt**

**Inserting Data**

**Json Stucture**

{

**`name`**:`mainName`,

**`interests`**: [`sports`,`computer`,`education`],

**`additionalDetails`**: [ {`name`:`n1`, `value`:`v1`} , {`name`:n2``,`value`:`v2`} ]

}

**Sample Practise Data Command**

db.users.insertMany([{name:`Davinder`,age:24, interests:[`sports`,`computer`,`education`],profession:`Computer Science Engineer`,additionalDetails:[{name:`telephone`, value:`7009362040`},{name:`parentsDetails`,value:[{name:`fatherName`,value:`RajeshKumar`},{name:`motherName`,value:`Pooja Rani`}]}]},{name:`Khushveer`, age:23,interests:[`sports`,`education`,`Movies`],profession:`Doctor`,additionalDetails:[{name:`telephone`, value:`98737283829`},{name:`parentsDetails`,value:[{name:`fatherName`,value:`KhusuFather`},{name:`motherName`,value:`MommyKaur1`}]}]},{name:`Rajnesh`, interests:[`sports`,`Traveling`,`education`],age:25,profession:`Manager`,additionalDetails:[{name:`telephone`, value:`98873748883`},{name:`parentsDetails`,value:[{name:`fatherName`,value:`RajiFather`},{name:`motherName`,value:`MommyKaur2`}]}]}])

**Commands**

admin Commands

* show dbs
* show users
* show roles

Collection Based Commands

**Inserting** document

* db.collection.**insertOne**({})
* db.collection.**insertMany**({})

**Tricky** in-build Functions

* db.collections.**count**()
* db.collections.**limit**()

**Finding** documents

* db.collection.**find**({},{})
* db.collection.**findOne**({},{})
  + firstArg - KeyValue Pair of FilteringField, secondArd - KeyValue Pair of shorting the view
    - eg db.users.**find**({name:`Davinder`},{name:1,interests:1})
* Operators used in find via advance filtering

KeyName ValueType

**$in** array -- alternate of $or:[{filter1},{filter2}]

**$nin** array -- not in

**$all** array -- alternate of and

eg **db.users.find({name: {$in:[`Davinder`,`Khushveer`]}})**

-- Either Davinder or Khushveer named Document Visible

eg **db.users.find({name: {$all:[`Davinder`,`Khushveer`]}})**

-- Only Davinder and Khushveer named Document Visible

* filtering through **Object** inside **ARRAY**

`array.objectKey`:`ObjectValue` -- is equals to .array[].objectKey==`objectValue`

eg - db.users.**find**({`additionalDetails.name`:`telephone`})

**Updating** document (simple)

1. **db**.**collection**.**updateOne**( {finding\_filter} , { **$set**:{setObject} } )
2. **db**.**collection**.**updateOne**( {finding\_filter} , { **$set**:{setObject} }, {**$inc**:{`key`:IncrementValue}} )
3. **db**.**collection**.**updateOne**(

{finding\_filter},

{**$set**:{`objectKey.arrayKey.**$[arg]**.chagingKey`:`value`}},

{**arrayFilters**:[{`**arg**.compareKey`:`comparingValue`}]}

)

Eg1.1 updatingVia\_Finding\_Name\_Set\_**common**Value\_to\_every\_key\_inside\_array

**db**.**users**.**updateOne**({name:`Davinder},{**$set**:{`additionalDetails.name`:`details`}})

Eg1.2 updatingVia\_Finding\_**ObjectId\_**Set\_new\_value\_to\_name

**db**.**users**.**updateOne**({\_id:ObjectID(658a509e0ea77e7f1ccbdd72)},{**$set**:{`name`:`Dravid`}})

Eg2.1 updatingVia\_Finding\_**Incrementing**\_Value\_of\_age

**db**.**users**.**updateOne**({name:`Davinder`},{**$inc**:{`age`:2}})

Eg2.2 updatingVia\_Finding\_**Decrement**\_Value\_of\_age

**db**.**users**.**updateOne**({name:`Davinder`},{**$inc**:{`age`:**-**2}})

Eg5 updatingVia\_Finding\_Name\_Then\_Set\_Value\_Inside\_ArrayTo\_Specific\_Object\_with\_name\_**telephone**

**db**.**users**.**updateOne**({name:'Davinder'},{**$set**:{'additionalDetails.**$[fil\_arg1]**.value':'9417233167'}},{**arrayFilters**: [{**'fil\_arg1**.name':'telephone'}]})

Eg6 **Pull** Operator for fetchingout elements of array

**db.users.update**({},{**"$pull"**:{additionalDetails:{name:'telephone'}}},{"**multi**": true})

Eg7 **Push** Operator for pushing elements of array

**db.users.updateMany**({},{**$push**:{'additionalDetails.**$[arg1]**.value':{'name':'Dadu','value':'Puranchand'}}},{**arrayFilters**:[{'**arg1**.name':'parentsDetails'}]},{**"multi"**:true})

Eg6.2 **Pull** Sepecific element present inside array of object contain key name:’Dadu’

**db.users.updateMany**({},{**$pull**:{'additionalDetails.**$[filadditional]**.value':{name:'Dadu'}}},{**arrayFilters**:[{**'filadditional**.name':'parentsDetails'}]})

Reference Link

netninja - mongodb playlist

https://www.youtube.com/watch?v=ExcRbA7fy\_A&list=PL4cUxeGkcC9h77dJ-QJlwGlZlTd4ecZOA