MongoDB

Prerequisites

- Start the mongoDB server using the command `mongod` in the terminal
- Start the mongoDB interactive shell using the command `mongo` on a separate terminal.
- > **NOTE** : Sometimes the mongod server is already running in the backgroud. So first just try running the command mentioned in the second step. If it doesn't work then start the server and the interactive shell one after another respectively.

After sucessfully entering the interactive shell try out the following stuff :

CREATE

Creating a database

- To create a database enter the command `use db_name` where *db*name_ is the name of the database you want to create.
- This command is also used to switch between databases
- Eg:-

```
use cars
```

List databases

- To view the various databases created enter :-

```
show dbs
```

- Result:

```
admin   0.000GB
config  0.000GB
local   0.000GB
temp    0.000GB
```

> **NOTE** : The *cars* db created in the earlier command isn't visible as it has no data in it. Once a collection is created it will be visible on executing the above command.

- To view the database you are currently using enter:-

```
db
```

Creating collections

- Collections are in `mongoDB` what tables are in `SQL` .
- The only difference is that you don't need to have a defined structure of the *column names* and the *datatype* those columns will have in `mongoDB` unlike `SQL` .
- You can directly create a collection without specifying any details of the data that will be entered in it.
- This can be done using the command `db.createCollection('collection_name')`
- Eg:-

```
db.createCollection('car_details')
```

Inserting documents

- In `mongoDB` when you make an entry in a collection that entry is called a *document* i.e. a *row* in terms of `SQL`
- You can insert a document in a collection using the command `db.collection_name.insert(data)`
- Structure of the data in the example below :-
  - brand
  - model_name
  - type
  - engine
    - fuel
    - cc
    - bhp
  - color
  - safety
    - airbag (if airbags isn't present then this field is omitted else its type is associated with the *key*)
    - sb_warn (seatbelt warning)
- Eg:-

```
db.car_details.insert(
    {
        'brand': 'chevrolet',
        'model_name': 'beat',
        'type': 'hatchback',
        'engine': {
            'fuel': 'petrol',
            'cc': 1000,
            'bhp': 100
        },
        'color': ['red', 'blue', 'green'],
        'safety': {
            'sb_warn': false
        },
        'price' : 400000
    }
)
```

- You can insert multiple entries in one go by using an array of objects i.e.

```
db.collection_name.insert([{},{},....])
```

- Copy and paste the contents of this file in your terminal.
  - The file contains an array of objects containing the car_details.

READ

- Fetching data from the documents is done using the `find()` function.
- Eg:-

```
db.car_details.find()
```

- This returns all the documents in the *car*details_ collection as **no** parameter was passed as to which filter(s) should be applied.
- To get a readable format of the results use the `pretty()` function
- Eg: -

```
db.car_details.find().pretty()
```

- This returns a properly formatted result of the `find()` output.

Filters

- To get the document(s) based on some filters enter the *key*and the corresponding *value* as the first parameter to the `find()` function.
- Eg:-
- Here we want to get the details of the cars of type **sedan**

```
db.car_details.find({ 'type' : 'sedan' }).pretty()
```

- You could relate this type of filtering to the *WHERE* clause in `SQL` .

Comparison Operators

- Comparison operators can be used to filter documents in monogDB using the following specifiers:-
  - **$gt** = greater than
  - **$gte** = greater than equal to
  - **$lt** = less than
  - **$lte** = less than equal to
  - **$ne** = not equal to
- Eg:- (**$lte**)

```
db.car_details.find({
    'price' : { '$lte' : 500000 }
}).pretty()
```

Logical Operators

- Logical operators can be used to filter documents in monogDB using the following specifications:-
  - **$or** = OR operation
  - **$and*** = AND operation
- Eg:- (AND)

```
db.car_details.find({'$and' : [
    { 'type' : 'sedan' },
    { 'price' : { '$gte' : 500000 } }
]}).pretty()
```

> Notice that the following command wil also output the same as the previous as the default operation in case of multiple filters is an *AND* operation.

- Eg:- (AND)(without the **$and** specifier)

```
db.car_details.find({
     'type' : 'sedan' ,
     'price' : { '$gte' : 500000 }
}).pretty()
```

- Eg:- (OR)

```
db.car_details.find({'$or': [
    { 'type' : 'sedan' },
    { 'price' : { '$gte' : 500000 } }
]}).pretty()
```

Functions

- **sort()**
  - Sort funtion can be used to sort the result of the query based on a key
  - For ascending set the value of the key to 1
    and for descending set it to -1
  - Eg:- SORT (desc)

```
db.car_details.find().sort(
    {'price' : -1}
).pretty()
```

- **limit()**
  - To limit the no of results of the query, send the no of results to be displayed as a parameter to the limit function
  - Eg:-

```
db.car_details.find().limit(3).pretty()
```

- **count()**
  - To just display the no of results that the query will output the `count()` function can be used
  - Eg:-

```
db.car_details.find().count()
```

Miscellaneous

- **$exists**
  - To check whether a key exists in the documents in the collections the $exists specifier
  - Eg:-

```
db.car_details.find({
    'safety.airbags': {'$exists' : true}
}).pretty()
```

- **$regex**
  - Regular expressions can also be used to find entries having a pattern using the $regex specifier
  - Eg:-

```
db.car_details.find({
    'model_name' : { $regex: /^e/ }
}).pretty()
```

UPDATE

- There are 3 functions associated to updation of documents in mongoDB :-
  - **updateOne()**
    This method is used to update the first matching document based on the filters provided

    Eg:-

```
db.car_details.updateOne(
    { 'model_name' : 'cruze' },
    { '$set' : { 'price' : 1500000 } }
)
```

- **updateMany()**
  This method is used to update all the matching documents based on the filters provided
  Eg:-

```
db.car_details.updateMany(
    { },
    { '$rename' : { 'body_type' : 'type' } }
)
```

- **replaceOne()**
  This method is used to update the first matching document based on the filters provided
  Eg:-

```
db.car_details.replaceOne(
  { 'model_name' : 'jazz' },
  {
      'brand': 'honda',
      'model_name': 'jazz',
      'type': 'hatchback',
      'engine': {
          'fuel': 'diesel',
          'cc': 1150,
          'bhp': 115
      },
      'color': ['silver','orange','white'],
      'safety': {
          'airbags' : 'front',
          'sb_warn': true
      },
      'price' : 750000
  }
)
```

DELETE

Deleting documents

- One or many documents can be deleted in mongoDb using the following commands:-

  - **deleteOne()**
    This method deletes the first document that matches the conditions specified
    Eg:-

```
db.car_details.deleteOne(
    { 'model_name' : 'city' },
)
```

- **deleteMany()**
  This method deletes the all the documents that match the conditions specified. If no conditions are specified the all documents in the collection are deleted.
  Eg:-

```
db.car_details.deleteMany(
     { 'body_type' : 'suv' },
)
```

Deleting collections

- To delete a collection enter `db.collection_name.drop()`
- Eg:-

```
db.car_details.drop()
```

Deleting databases

- To delete the database you are currently working on enter :-

```
db.dropDatabase()
```

Resources

- MongoDB official Documentation
- MongoDB Tutorials Point
- Marvels of MongoDB (pluralsight) slides

> **NOTE** : The official documentation is more than sufficient to get a hold of both basic and advanced concepts in mondoDB as they have an exhaustive no. of examples associated with every functionality. So, first have a look at the documentation before searching for a tutorial.