First, we use the test collection to insert some documents that include just an integer and a string as properties.

```
db.test.insert({i: 3, s: "something"})
```

"find()" method without an argument prints out the whole collection. (It's actually a cursor.)

```
db.test.find()
```

We can pupulate the collection with three more documents.

```
db.test.insert({i: 5, s: "anything"})

db.test.insert({i: 7, s: "nothing"})

db.test.insert({i: 9, s: "everything"})
```

Now, we can find a single document in the collection.

```
db.test.find({{i:1}})
```

Or, we can find a set of docements based on a constraint. The constraint is shown as an embedded document.

```
db.test.find({{i: {$gt : 0} }})
```

We can choose to enable or disable the automatically given _id fields by setting its value to 0 or 1 in a projection. A projection is the second argument in a find method.

```
db.test.find({{i: {$gt : 0} }}, { _id:0 })
```

In a projection, you can either enable a set of properties to 1 or disable a set of properties to 0.
(_id property is special and it can be separately set independent from others.)
(This feature is better visualized in the next collection.)

```
db.test.find({{i: {$gt : 0} }}, { _id:0, i:0 })

db.test.find({{i: {$gt : 0} }}, { _id:0, i:1 })
```

You can sort the results of the find method based on any of the fields.

```
db.test.find({{i: {$gt : 0} }}, { _id:0 }).sort( {i:1} )
```

We can enter a completely different document to the collection.

```
db.test.insert({i: 0, s: "null", x:"a new property"})

db.test.insert({a:3, b:4, c:"this document is fundamentally different"})
```

Now, we can import a sample collection for further examples. (Use the provided json file.)

```
mongoimport --db test --collection restaurants --drop --file
dataset.json
```

Insert example.
(This one also exemplifies the structure of a document.)

```
db.restaurants.insert(
   {
      "address" : {
         "street" : "2 Avenue",
         "zipcode" : "10075",
         "building" : "1480",
         "coord" : [ -73.9557413, 40.7720266 ],
      },
      "borough" : "Manhattan",
      "cuisine" : "Italian",
      "grades" : [
         {
            "date" : ISODate("2014-10-01T00:00:00Z"),
            "grade" : "A",
            "score" : 11
         },
         {
            "date" : ISODate("2014-01-16T00:00:00Z"),
            "grade" : "B",
            "score" : 17
         }
      ],
      "name" : "Vella",
      "restaurant_id" : "41704620"
   }
)
```

Find example. (A nice point to explain cursors & iterators. Also mention that mongodb is "near realtime" and the results may change while you're iterating through the results.)

```
db.restaurants.find( { "address.zipcode": "10075" } )
```

Operator example.

```
db.restaurants.find( { "grades.score": { $lt: 10 } } )
```

Commas are used as "AND" operator. OR operator has distinct structure.

```
db.restaurants.find(

   { $or: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }

)
```

Sorting based on multiple properties.

```
db.restaurants.find( { "cuisine": "Mexican" } ).sort( { "borough": 1,
"address.zipcode": 1 } )
```

Using projection. (Second one will be pretty useless)

```
db.restaurants.find( { "cuisine": "Mexican" } , {_id:0, "name": 1,
"borough": 1, "address.zipcode": 1}).sort( { "borough": 1,
"address.zipcode": 1 } )

db.restaurants.find( { "cuisine": "Mexican" } , {_id:0, "name": 0,
"borough": 0, "address.zipcode": 0}).sort( { "borough": 1,
"address.zipcode": 1 } )
```

Updating single embedded document.

```
db.restaurants.update(

  { "restaurant_id" : "41156888" },

  { $set: { "address.street": "East 31st Street" } }

)
```

Updating multiple embedded document. (So, you need to lock for atomic operation.)

```
db.restaurants.update(
  { "address.zipcode": "10016", cuisine: "Other" },
  {
    $set: { cuisine: "Category To Be Determined" },
    $currentDate: { "lastModified": true }
  },
  { multi: true}
)
```

Updating the whole document.

```
db.restaurants.update(
   { "restaurant_id" : "41704620" },
   {
     "name" : "Vella 2",
     "address" : {
              "coord" : [ -73.9557413, 40.7720266 ],
              "building" : "1480",
              "street" : "2 Avenue",
              "zipcode" : "10075"
     }
   }
)
```

Removing some documents based on a query.

```
db.restaurants.remove( { "borough": "Manhattan" } )
```

Matching everything while removing.

```
db.restaurants.remove( { } )
```

Dropping the collection.

```
db.restaurants.drop()
```

For neo4j, we use the given movie database in their local web interface.