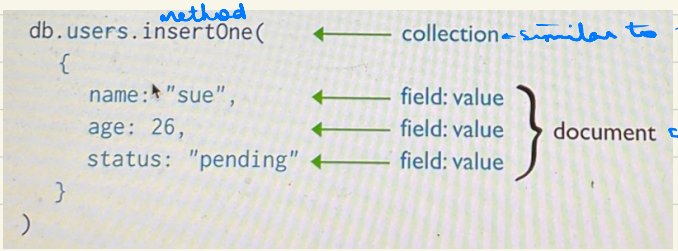
* CRUD

Create
~ mongod
on new terminal ~ mongosh

* show dbs (databases)

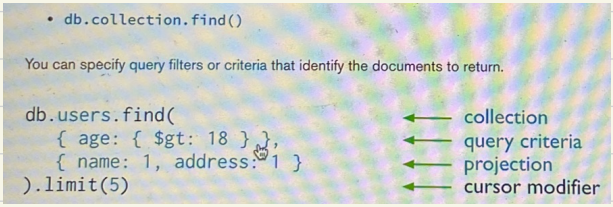
* use shopDB // create a database
* db // to see in which database



db.product.insertOne({_id: 1, name: "Pen", price: 1.20})
- show collection output: product
db output: shopDB

db.product.insertOne({_id: 2, name: "Pencil", price: 0.80})

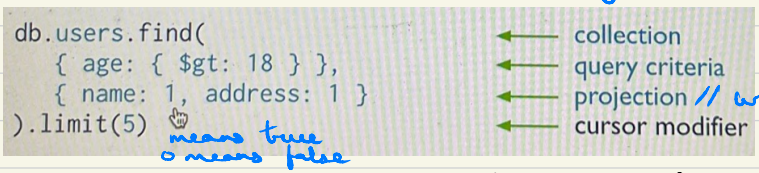
Read



* db.product.find() will get everything

* db.product.find({name: "Pencil"}) will bring data where name matches Pencil

* db.product.find({price: {$gt: 1}})
find product with price greater than 1



* db.product.find({_id:1}, {name: 1})
will return name of product with id 1 but you will also be returned id of the product by default so here we can use projection

* db.product.find({_id:1}, {name:1, _id:0})
just return name and dont return id

Update

db.product.UpdateOne({id:1}, {$set : {stock:32}})

which one I want to update — what you want to add with set

Delete

- `db.collection.deleteOne()` *New in version 3.2*
- `db.collection.deleteMany()` *New in version 3.2*

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

```
db.users.deleteMany(
   { status: "reject" }
)
```

collection

delete filter

db.products.deleteOne({_id:2})

will delete the full row of pencil

Relationships in MongoDB

```
db.products.insertOne(
    {
        _id: 3,
        name: "Rubber",
        price: 1.30,
        stock: 43,
        reviews: [
            {
                authorName: "Sally",
                rating: 5,
                review: "Best rubber ever"
            },
            {
                authorName: "John",
                rating: 5,
                review: "Best rubber"
            }
        ]
    }
)
```

```
db.product.insertOne({_id:2, name: "Pencil", price: 0.80, stock: 32, reviews: [{authorName: "Mohit", rating: 3, reviews: [{authorName: "Kartik", rating: 4, review: "Good"}]})
```

app.js

```
//jshint esversion:6

const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');

// Connection URL
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'fruitsdb';

// Create a new MongoClient
const client = new MongoClient(url, {useNewUrlParser : true});

// Use connect method to connect to the Server
client.connect(function(err) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  const db = client.db(dbName);

  insertDocuments(db, function() {
    client.close();
  });
});
```

lets do with testing
validates connection and entry to the database mongodb

to avoid warning

connects server and myproject

close only after inserting

once done inserting the documents will close the database

+ to insert & also find

```
insertDocuments(db, function() {
  findDocuments(db, function() {
    client.close();
  });
});
```

now close server only after finding

```
findDocuments(db, function() {
  client.close();
});
```

to always first run server on terminal

~ mongod

then run node app.js

```
const insertDocuments = function(db, callback) {
  // Get the documents collection
  const collection = db.collection('documents');
  // Insert some documents
  collection.insertMany([
    {a : 1}, {a : 2}, {a : 3}
  ], function(err, result) {
    assert.equal(err, null); // no errors while adding
    assert.equal(3, result.result.n); // we assume we have 3
    assert.equal(3, result.ops.length); // results inserted in our collection
    console.log("Inserted 3 documents into the collection");
    callback(result);
  });
}
```

fruits

```
{
  name : "Apple",
  score: 8,
  review: "Great fruit"
},
{
  name : "Orange",
  score: 6,
  review: "Kinda sour"
}, {
  name: "Banana",
  score: 9,
  review:"Great stuff!"
}
```

because of this 3 items

now when show dbs we can see fruitsdb

→ use fruitsdb

* db.fruits.find()

```
const findDocuments = function(db, callback) {
  // Get the documents collection
  const collection = db.collection('fruits');
  // Find some documents
  collection.find({}).toArray(function(err, fruits) {
    assert.equal(err, null);
    console.log("Found the following records");
    console.log(fruits);
    callback(fruits);
  });
}
```

from where we want to find

name of the array which will be formed from toArray

So that we can use our db collection to be accessed by Node JS

now when we run node app.js

output :

```
Connected successfully to server
Found the following records
[ { _id: 5bc06f3da46cc5cbd98cf45c,
    name: 'Apple',
    score: 8,
    review: 'Great fruit' },
  { _id: 5bc06f3da46cc5cbd98cf45d,
    name: 'Orange',
    score: 6,
    review: 'Kinda sour' },
  { _id: 5bc06f3da46cc5cbd98cf45e,
    name: 'Banana',
    score: 9,
    review: 'Great stuff!' },
  { _id: 5bc06f480e9a3acbdd7efbbb,
    name: 'Apple',
    score: 8,
    review: 'Great fruit' },
```