MongoDB

Paul Fodor

CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

NoSQL

- NoSQL databases are NOT Relational Databases
- No tables and no table normalization
- Allows non-structured data [Data may be structured as needed]
- Underlying storage format: Usually JSON
- Handles Big Data
- No predefined Schema [Data can be unstructured]
- Cheaper/Easier to manage
- Scaling [Horizontal / Scale Out]

Scaling out vs Scaling up

- Relational Databases are designed to be hosted on a single server
  - [It is possible to split a relational database to several servers but there are many adjustments that have to be made]
  - This forces scaling to be 'up', meaning acquiring a higher capacity/higher performance server ($$$)
- NoSQL databases can be distributed with replicated copies or slice of the data
  - When scaling is needed, you can 'scale out': simply add more servers

RDBMS Advantages

- Good for relational data
  - There is a structure to the data
  - Data items have set content schema/tables/columns
  - You know how data in tables is related (foreign keys)
- Normalization
  - Eliminates Redundancy
  - Minimizes Space usage ➔ Better performance
- Use SQL standard query language
- Data Integrity Checks though Constraints
- Supports Transactions: ACID – Atomicity, Consistency, Isolation, Durability
  - ➔ You can do this with NoSQL but not built into the system

Types of NoSQL DBs

- **Document Databases** [MongoDB, CouchDB]
  - Totally schemaless
- **Key-Value Stores** [Redis, Memcache] and **Cache Systems** [Couchbase Server]
  - Mimics Associative Array / Hash
  - Good for Huge datasets with very simple data
  - Fast
  - Not very customizable
- **Column Databases** [Apache Cassandra]
  - Optimized for reading columns rather than rows
  - Good for analytics
  - Reduces disk I/O requirements

Types of NoSQL DBs

- **Graph Databases** [Neo4J]
  - Not very common
  - Everything is a Node
  - Nodes can be related vis 'Edges'

NoSQL Drawbacks

- Data is duplicated
- Does not support ACID (specifically, has issues with consistency)
- Not read-optimized
  - NoSQL has to read all data in all blobs for a simple query
- Joins are manual and very difficult

MongoDB

- Mongo is a ***"document based"*** NoSQL database
- Support a shell interface as well as a GUI (Compass)
  - Install server on your own computer

  OR
  - Use the Atlas Cloud version

MongoDB Basic Commands

- **show dbs** – Show the databases known by Mongo
- **use db**
  - Select a db with which to work
  - If db does not exist, creates db
- **db** – returns the name of the db currently in use
- **db.<newCollectionName>.insert({record}) –** Create a new collection named <newCollectionName> and insert a record.
- **db.<collectionName>.drop()** – Delete the collection named <collectionName>

Inserting data

- **insert or insertOne()** – Insert a record into a collection
- **insertMany()** - inserts multiple records/documents
- Example:

```
use music;
db.albums.insert({title:'Different Light',
    artist:'The Bangles',
    genre: 'Rock',
    runtime: 39,
    publication: 1986,
    media: 'iTunes',
    songTitles: ['Manic Monday','In a Different Light','Walking Down Your Street', 'Walk Like an Egyption',
    'Standing in the Hallway', 'Return Post', 'If She Knew What She Wants', 'Let It Go','September Gurls',
    'Angels Dont Fall in Love','Following','Not Like You']});
```

- Inserts 1 document into the ***albums*** collection in the ***music*** database.
- The document includes attributes ***title, artist, genre, runtime, publication, media***, and ***songTitles*** which is an array of song tracks on the album.

Finding data

- **.find()** – Find all documents in collection and print them.
- **.find({attribute/value to match})** – Find documents matching the attribute/value given.
- **.findOne()** – Find a single document that matches criteria

Finding data

db.music.albums.find().pretty() - returns all documents in the collection

```
{
	"_id" : ObjectId("5e439e1574629038afac0eab"),
	"title" : "Blue Chip",
	"artist" : "Acoustic Alchemy",
	"genre" : "New Age",
	"runtime" : 44,
	"publication" : 1989,
	"media" : "iTunes",
	"songTitles" : [
		"Cataline Kiss",
		"The Blue Chip Bop",
		"Making Waves",
		"With You in Mind",
		"Bright Tiger",
		"Ariane",
		"Highland",
		"Boulder Coaster",
		"Hearts in Chains",
		"No More Nachos (Por Favor)"
	]
}
{
	"_id" : ObjectId("5e439e1574629038afac0eac"),
	"title" : "Natural Elements",
	"artist" : "Acoustic Alchemy",
	"genre" : "New Age",

   ...
```

.pretty() = Reformats output to be nicely readable.

_id is generated by MongoDB and is provided as a unique reference to the document.

Finding data

- db.music.albums.find({publication: 1988}).pretty()
  - Returns all documents with a publication attribute of 1988


```
{
	"_id" : ObjectId("5e439e1574629038afac0eac"),
	"title" : "Natural Elements",
	"artist" : "Acoustic Alchemy",
	"genre" : "New Age",
	"runtime" : 35,
	"publication" : 1988,
	"media" : "iTunes",
	"songTitles" : [
		"Drakes Drum",
		"Overnight Sleeper",
		"Natural Elements",
		"Casino",
		"If Only",
		"Ballad for Kay",
		"Evil the Weasel",
		"Late Night Duke Street"
	]
}
{
	"_id" : ObjectId("5e44a7d074629038afac0eb0"),
	"title" : "Greatest Hits Live",
	"artist" : "Carly Simon",
	"genre" : "Rock",
	"runtime" : 90,
	"publication" : 1988,
	"media" : "iTunes",
	"songTitles" : [
		"Nobody Does it Better",
		"Youre So Vain",
		"It Happens Every Day",
```

Finding data

- **db.music.albums.findOne({publication: 1988})**
  - Returns 1 document with a ***publication*** attribute of 1988

```
{
    "_id" : ObjectId("5e439e1574629038afac0eac"),
    "title" : "Natural Elements",
    "artist" : "Acoustic Alchemy",
    "genre" : "New Age",
    "runtime" : 35,
    "publication" : 1988,
    "media" : "iTunes",
    "songTitles" : [
        "Drakes Drum",
        "Overnight Sleeper",
        "Natural Elements",
        "Casino",
        "If Only",
        "Ballad for Kay",
        "Evil the Weasel",
        "Late Night Duke Street"
    ]
}
```

Do not need pretty() here. findOne() does that for you.

Finding data with Relative operators

- **.find({ <relop> attribute/value})** – Find documents value of attribute relative to a given value
  - $gt:
  - $gte:
  - $lt:
  - $lte:

Finding data with Relative operators

- **db.music.albums.find({runtime: {$lt: 39}}).pretty()**
  - Albums with less than 39 minutes runtime


```
{
        "_id" : ObjectId("5e439e1574629038afac0eac"),
        "title" : "Natural Elements",
        "artist" : "Acoustic Alchemy",
        "genre" : "New Age",
        "runtime" : 35,
        "publication" : 1988,
        "media" : "iTunes",
        "songTitles" : [
                "Drakes Drum",
                "Overnight Sleeper",
                "Natural Elements",
                "Casino",
                "If Only",
                "Ballad for Kay",
                "Evil the Weasel",
                "Late Night Duke Street"
        ]
}
{
        "_id" : ObjectId("5e439e1574629038afac0eab"),
        "title" : "Reg Chip",
        "artist" : "Acoustic Alchemy",
        "genre" : "New Age",
        "runtime" : 24,
```

Finding data with Relative operators

- **db.music.albums.find({publication: {$gt: 1988}}).pretty()**
  - Albums published after 1988


```
{

    "_id" : ObjectId("5e439e1574629038afac0eab"),
    "title" : "Blue Chip",
    "artist" : "Acoustic Alchemy",
    "genre" : "New Age",
    "runtime" : 44,
    "publication" : 1989,
    "media" : "iTunes",
    "songTitles" : [
        "Cataline Kiss",
        "The Blue Chip Bop",
        "Making Waves",
        "With You in Mind",
        "Bright Tiger",
        "Ariane",
        "Highland",
        "Boulder Coaster",
        "Hearts in Chains",
        "No More Nachos (Por Favor)"
    ]
}
```

Text Searches

- Add index:
  - **.createIndex({<fieldname>: 'text' })**
- Text Search command
  - **.find({ $text: { $search: " \"<string>\" " } })**

Text Searches

- First, add an index:

db.music.albums.createIndex({'artist': 'text' })

- Creates and index on the field ***artist***

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Text Searches

- db.music.albums.find({$text: { $search: " \"Bangles\" " } }).pretty()
  - Searches for and returns any document where the ***artist*** attribute includes ***Bangles***


```
{
        "_id" : ObjectId("5e44a7d074629038afac0eaf"),
        "title" : "Different Light",
        "artist" : "The Bangles",
        "genre" : "Rock",
        "runtime" : 39,
        "publication" : 1986,
        "media" : "iTunes",
        "songTitles" : [
                "Manic Monday",
                "In a Different Light",
                "Walking Down Your Street",
                "Walk Like an Egyption",
                "Standing in the Hallway",
                "Return Post",
                "If She Knew What She Wants",
                "Let It Go",
                "September Gurls",
                "Angels Dont Fall in Love",
                "Following",
                "Not Like You"
        ]
}
```

Modifying returned document set

- **.sort()** – sorts returned values on a field (ascending or descending)
- **.pretty()** – 'Pretty' prints the returned data
- **.count()** – return the count of records rather than the data
- **.limit(#)** – return only the number of results given by the number (#).

Modifying returned document set

- db.music.albums.find({artist:'The Alan Parsons Project'})
  .sort({publication:-1}).pretty()
  - Returns all documents with the ***artist*** '***The Alan Parsons Project***' sorted from latest ***publication*** year to oldest publication year.


```
{
	"_id" : ObjectId("5e439e1574629038afac0ead"),
	"title" : "Gaudi",
	"artist" : "The Alan Parsons Project",
	"genre" : "Rock",
	"runtime" : 39,
	"publication" : 1987,
	"media" : "iTunes",
	"songTitles" : [
		"La Sagrada Familia",
		"Too Late",
		"Closer to Heaven",
		"Standing on Higher Ground",
		"Money Talks",
		"Inside Looking Out",
		"Paseo de Gracia [Instrumental]"
	]
}
{
	"_id" : ObjectId("5e439e1574629038afac0eae"),
```

Modifying returned document set

- db.music.albums.find({artist:'The Alan Parsons Project'}).count()
  - Will return the number of albums from the ***artist 'The Alan Parsons Project'***. It does not return the data itself.

  2

Modifying returned document set

- **db.music.albums.find({artist:'The Alan Parsons Project'}).limit(1).pretty()**
  - This will return the first document from the artist 'The Alan Parsons Project'.


```
{
        "_id" : ObjectId("5e439e1574629038afac0eac"),
        "title" : "Natural Elements",
        "artist" : "Acoustic Alchemy",
        "genre" : "New Age",
        "runtime" : 35,
        "publication" : 1988,
        "media" : "iTunes",
        "songTitles" : [
                "Drakes Drum",
                "Overnight Sleeper",
                "Natural Elements",
                "Casino",
                "If Only",
                "Ballad for Kay",
                "Evil the Weasel",
                "Late Night Duke Street"
        ]
}
```

Special processing on results

- **.forEach()** – Preforms a function on each document returned by find()

```
db.music.albums.find({publication: {$lt: 1988}})
.sort({publication:1})
.forEach(function(doc) {
        print("Title: '" + doc.title + "'        ---- num tracks: "
                + doc.songTitles.length)
})
```

  - Find all documents with publication years before 1988, sort by publication year, and print the title and number of tracks on each album.

```
Title: "Stereotomy"        ---- num tracks: 9
Title: "Different Light"        ---- num tracks: 12
Title: "Gaudi"        ---- num tracks: 7
```

Altering data

- **.update()** – Update values in one or more documents
  - Replace whole document
  - Replace selected attributes
  - Operators:
    - $inc: { field: <incvalue>}
    - $set
    - $rename: {oldname: 'newname'}
  - Options
    - upsert

Altering data

- Add a new field:

db.music.albums.update({title: 'Stereotomy'}, {$set: {favorites: 'true'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.music.albums.find({title: 'Stereotomy'}).pretty()


```
{
	"_id" : ObjectId("5e439e1574629038afac0eae"),
	"title" : "Stereotomy",
	"artist" : "The Alan Parsons Project",
	"genre" : "Rock",
	"runtime" : 42,
	"publication" : 1985,
	"media" : "iTunes",
	"songTitles" : [
		"Stereotomy",
		"Beaujolais",
		"Urbania",
		"Limelight",
		"In the Real World",
		"Wheres the Walrus",
		"Light of the World",
		"Chinese Whispers",
		"Stereotomy Two"
	],
	"favorites" : "true"
}
```

Altering data

- Increment a field:

db.music.albums.update({title:'Different Light'}, {$inc: {runtime: 2}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.music.albums.find({title: 'Different Light'}).pretty()


```
{
	"_id" : ObjectId("5e44a7d074629038afac0eaf"),
	"title" : "Different Light",
	"artist" : "The Bangles",
	"genre" : "Rock",
	"runtime" : 41,
	"publication" : 1986,
	"media" : "iTunes",
	"songTitles" : [
		"Manic Monday",
		"In a Different Light",
		"Walking Down Your Street",
		"Walk Like an Egyption",
		"Standing in the Hallway",
		"Return Post",
		"If She Knew What She Wants",
		"Let It Go",
		"September Gurls",
		"Angels Dont Fall in Love",
		"Following",
		"Not Like You"
	]
}
```

Altering data

- Change a field name in 1 document:

db.music.albums. update({title: 'Greatest Hits Live'}, {$rename: {runtime: 'totalMinutes'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.music.albums.find({title: 'Greatest Hits Live'}).pretty()

```
{
        "_id" : ObjectId("5e44e69674629038afac0eb1"),
        "title" : "Greatest Hits Live",
        "artist" : "Carly Simon",
        "genre" : "Rock",
        "publication" : 1988,
        "media" : "iTunes",
        "songTitles" : [
                "Nobody Does it Better",
                "Youre So Vain",
                "It Happens Every Day",
                "Anticipation",
                "The Right Thing To Do",
                "Do the Walls Come Down",
                "You Belong to Me",
                "Two Hot Girls",
                "All I Want is You",
                "Coming Around Again",
                "Never Been Gone"
        ],
        "totalMinutes" : 90
}
```

Deleting Data

- **.remove()** – Delete the applicable document

```
db.music.albums.remove({artist: 'Carly Simon'})
```

  - Removes all documents with an ***artist*** **of** *Carly Simon*

```
WriteResult({ "nRemoved" : 1 })
```

Connecting to MongoDB

- Download and install the official MongoDB driver:

```
npm install mongodb
```

- Node.js can use this module to manipulate MongoDB databases:

```
var mongo = require('mongodb');
```

Connecting to MongoDB

- Create a database called "mydb":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

- In MongoDB, a database is not created until it gets content!
- MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

Connecting to MongoDB

- Creating a Collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Connecting to MongoDB

- Insert Into Collection:
  - A document in MongoDB is the same as a record/tuple in MySQL

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Connecting to MongoDB

- Insert Multiple Documents:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = [
    { name: 'John', address: 'Highway 71'},
    { name: 'Peter', address: 'Lowstreet 4'},
    { name: 'Amy', address: 'Apple st 652'},
    { name: 'Hannah', address: 'Mountain 21'},
    { name: 'Michael', address: 'Valley 345'},
    { name: 'Sandy', address: 'Ocean blvd 2'},
    { name: 'Betty', address: 'Green Grass 1'},
    { name: 'Richard', address: 'Sky st 331'},
    { name: 'Susan', address: 'One way 98'},
    { name: 'Vicky', address: 'Yellow Garden 2'},
    { name: 'Ben', address: 'Park Lane 38'},
    { name: 'William', address: 'Central st 954'},
    { name: 'Chuck', address: 'Main Road 989'},
    { name: 'Viola', address: 'Sideway 1633'}
  ];
  dbo.collection("customers").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("Number of documents inserted: " + res.insertedCount);
    db.close();
  });
});
```

Connecting to MongoDB

- The Result Object:

```
{
  result: { ok: 1, n: 14 },
  ops: [
    { name: 'John', address: 'Highway 71', _id: 58fdbf5c0ef8a50b4cdd9a84 },
    { name: 'Peter', address: 'Lowstreet 4', _id: 58fdbf5c0ef8a50b4cdd9a85 },
    { name: 'Amy', address: 'Apple st 652', _id: 58fdbf5c0ef8a50b4cdd9a86 },
    { name: 'Hannah', address: 'Mountain 21', _id: 58fdbf5c0ef8a50b4cdd9a87 },
    { name: 'Michael', address: 'Valley 345', _id: 58fdbf5c0ef8a50b4cdd9a88 },
    { name: 'Sandy', address: 'Ocean blvd 2', _id: 58fdbf5c0ef8a50b4cdd9a89 },
    { name: 'Betty', address: 'Green Grass 1', _id: 58fdbf5c0ef8a50b4cdd9a8a },
    { name: 'Richard', address: 'Sky st 331', _id: 58fdbf5c0ef8a50b4cdd9a8b },
    { name: 'Susan', address: 'One way 98', _id: 58fdbf5c0ef8a50b4cdd9a8c },
    { name: 'Vicky', address: 'Yellow Garden 2', _id: 58fdbf5c0ef8a50b4cdd9a8d },
    { name: 'Ben', address: 'Park Lane 38', _id: 58fdbf5c0ef8a50b4cdd9a8e },
    { name: 'William', address: 'Central st 954', _id: 58fdbf5c0ef8a50b4cdd9a8f },
    { name: 'Chuck', address: 'Main Road 989', _id: 58fdbf5c0ef8a50b4cdd9a90 },
    { name: 'Viola', address: 'Sideway 1633', _id: 58fdbf5c0ef8a50b4cdd9a91 } ],
  insertedCount: 14,
  insertedIds: [
    58fdbf5c0ef8a50b4cdd9a84,
    58fdbf5c0ef8a50b4cdd9a85,
    58fdbf5c0ef8a50b4cdd9a86,
    58fdbf5c0ef8a50b4cdd9a87,
    58fdbf5c0ef8a50b4cdd9a88,
    58fdbf5c0ef8a50b4cdd9a89,
    58fdbf5c0ef8a50b4cdd9a8a,
    58fdbf5c0ef8a50b4cdd9a8b,
    58fdbf5c0ef8a50b4cdd9a8c,
    58fdbf5c0ef8a50b4cdd9a8d,
    58fdbf5c0ef8a50b4cdd9a8e,
    58fdbf5c0ef8a50b4cdd9a8f
    58fdbf5c0ef8a50b4cdd9a90,
    58fdbf5c0ef8a50b4cdd9a91 ]
}
```

Connecting to MongoDB

- The _id Field:
  - If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.
  - If you do specify the _id field, the value must be unique for each document.

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = [
    { _id: 154, name: 'Chocolate Heaven'},
    { _id: 155, name: 'Tasty Lemon'},
    { _id: 156, name: 'Vanilla Dream'}
  ];
  dbo.collection("products").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log(res);
    db.close();
  });
});
```

Connecting to MongoDB

- The findOne() method returns the first occurrence in the selection
  - The first parameter of the findOne() method is a query object
    - empty query object selects all documents in a collection (but returns only the first document)

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").findOne({}, function(err, result) {
    if (err) throw err;
    console.log(result.name);
    db.close();
  });
});
```

Company Inc.

Connecting to MongoDB

- The second parameter of the find() method is the projection object that describes which fields to include in the result
  - Return the fields "name" and "address" of all documents in the customers collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find({}, { projection: { _id: 0, name: 1, address: 1 }
}).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

```
[
  { name: 'John', address: 'Highway 71'},
  { name: 'Peter', address: 'Lowstreet 4'},
  { name: 'Amy', address: 'Apple st 652'},
```

Connecting to MongoDB

- This example will exclude "address" from the result:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find({}, { projection:
        {address: 0 } }).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Connecting to MongoDB

The Result Object

Return the address of the third document:

```
console.log(result[2].address);
```

Connecting to MongoDB

- When finding documents in a collection, you can filter the result by using a query object:
  - Find documents with the address "Park Lane 38":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var query = { address: "Park Lane 38" };
  dbo.collection("customers").find(query).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Connecting to MongoDB

- Filter With Regular Expressions:
  - To find only the documents where the "address" field starts with the letter "S", use the regular expression /^S/:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var query = { address: /^S/ };
  dbo.collection("customers").find(query).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

```
[
  { _id: 58fdbf5c0ef8a50b4cdd9a8b , name: 'Richard', address: 'Sky st 331' },
  { _id: 58fdbf5c0ef8a50b4cdd9a91 , name: 'Viola', address: 'Sideway 1633' }
]
```

Connecting to MongoDB

- Sort the Result:
  - The sort() method takes one parameter, an object defining the sorting order.
    - Sort the result alphabetically by name:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var mysort = { name: 1 }; // Use the value -1 in the sort object to sort descending.
  dbo.collection("customers").find().sort(mysort).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

```
[
  { _id: 58fdbf5c0ef8a50b4cdd9a86, name: 'Amy', address: 'Apple st 652'},
  { _id: 58fdbf5c0ef8a50b4cdd9a8e, name: 'Ben', address: 'Park Lane 38'},
  { _id: 58fdbf5c0ef8a50b4cdd9a8a, name: 'Betty', address: 'Green Grass 1'},
```

Connecting to MongoDB

- Delete Document:
  - Delete the document with the address "Mountain 21":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: 'Mountain 21' };
  dbo.collection("customers").deleteOne(myquery, function(err, obj) {
    if (err) throw err;
    console.log("1 document deleted");
    db.close();
  });
});
```

Connecting to MongoDB

- Delete all documents were the address starts with the letter "O":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: /^O/ };
  dbo.collection("customers").deleteMany(myquery, function(err, obj) {
    if (err) throw err;
    console.log(obj.result.n + " document(s) deleted");
    db.close();
  });
});
```

Connecting to MongoDB

- Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });
});
```

Connecting to MongoDB

- Update Only Specific Fields:
  - When using the $set operator, only the specified fields are updated
    - Update the address from "Valley 345" to "Canyon 123":

```
...
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: { address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues,
 function(err, res) {
...
```

Connecting to MongoDB

- Limit the result to only return 5 documents:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find().limit(5).toArray(
   function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Connecting to MongoDB

- Limit the result to only return 5 documents:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find().limit(5).toArray(
   function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Join

- MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.
  - The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.
  - Consider you have a "orders" collection and a "products" collection:

```
orders
[
  { _id: 1, product_id: 154, status: 1 }
]
products
[
  { _id: 154, name: 'Chocolate Heaven' },
  { _id: 155, name: 'Tasty Lemons' },
  { _id: 156, name: 'Vanilla Dreams' }
]
```

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection('orders').aggregate([
    { $lookup:
       {
         from: 'products',
         localField: 'product_id',
         foreignField: '_id',
         as: 'orderdetails'
       }
     }
    ]).toArray(function(err, res) {
    if (err) throw err;
    console.log(JSON.stringify(res));
    db.close();
  });
});
```

```
[
  { "_id": 1, "product_id": 154, "status": 1, "orderdetails": [
    { "_id": 154, "name": "Chocolate Heaven" } ]
  }
```