**10917 MongoDB for Node.js Developers**

**Using Shell to access Mongo in Terminal**

* mongod
* 'database' : 'mongodb://localhost/Insta'

**Using Shell via Terminal**

* show dbs | use <dbName> | show collections

**Create, Insert, Update, Find, Remove**

> db.createCollection('<Name>')

> db.<Name>.insert({ title: 'car 1', make: 'honda', year: '2010' });

> db.categories.update({name:'Technology'}, {$set: {description: 'learn more about tech stuff'}})

> db.users.update({email: "[me2@me.com](mailto:me2@me.com)"}, { $set: {"role" : "admin"}}, {$upsert: true});

> db.users.update({email: 'your@email.addy'}, {$set:{email:'schroe9@gmail.com'}})

> db.things.find() //connector to interact with database //Using my DB find all the things

> db.things.remove ({})

> db.things.insert({ "a" : 1, "b" : 2, "c" : 3 })

> db.things.find({ "a" : 1 }) //finds documents with only specified field

> db.things.find().pretty() //prints a more readable data set

> for (var i = 0; i<10; i++) { db.things.insert({ "x" : i }) } //goes through 10 docs inserts 'x'

> db.test\_users.remove( {"\_id": ObjectId("563e24b14dc3382c17c7d006")}); - **remove by ID**

**Comparison Query Operators**

$gt greater than

$gte greater than or equal to

$in exists in

$lt less than

$lte less than or equal to

$ne not equal to

$nin does not exist

**What is MongoDB?**

* Non-relational, Stores in JSON documents, Horizontally Scaled
* Schemaless -> 2 docs don't need to have the same schema
* Brings together: Scalability & Peformance + Functionality
* **Document** is instance of the model

**Does Not Support**:

* Joins and Transactions (across multiple documents)

**The Process**

* Client makes a request that goes to server. Node.JS processes and returns something. If your app neds to store persistent data then use Mongo. App makes a request and Mongo responds when completed.

**Mongo Shell** (administration interface)

* Like mongo & node it is a C++ controlled using V8. You can open a terminal and make requests to mongo which will get a reponse. Useful for doing things like: asking questions about configuration, seeing what is stored or debugging app.

**Driver**

* This is what is used to communicate between Node and Mongo. Use API to move/insert doc

**Intro to JSON**

* Basic Structure -- {"a" : 1}
* Value Types -- string, bool, number
* Basic Nesting -- arrays, nested objects
* Deep Nesting -- combines arrays with objects

> var course = db.deepnested.findOne()

> course //access the JSON object

> course.students

> course.students[0].name //call the element course['students'][0].name

> course.students[0].name = "Sue"

**Asynchronous vs Synchronous** (Mongo is Sync, and Node is Async)

* Names for handling operations that don't require active work on running thread
* When using Asynchronous your able to run calls to the app parallel

**Node.js**

function(err, doc) - - common convention for the callback -> take err as 1st param, value as 2nd

> node app.js // 'Called findOne!' { name: 'MongoDB' }

> npm install // installs the modules located in package.json (node\_modules directory)

> npm remove X | npm insall express@3.x

**Creating/Accessing files**

* make the folder and add/edit source code as .JS file. Navigate via cd + folder.
* > node app.js //or location of the file
* 'Hello World' app.js file creates a server and tells it what to do when receiving request

**Using Commands**

* **cd** (+ folder opens directory) | **ls** (shows files in folder) | **cls** (clears prompt)
* **mkdir** (+ name creates folder) | **ctrl+C** (if cmd froze) |

**Express**

* 2 ways Express can get data from client side of an application: use GET and SET

**MongoDB is Schemaless**

-- Don't need to have same keys in same document, provides flexibility to add/subtract/change

**JSON Revisited**

* Arrays - list of things []
* Dictionaries - associated maps {}
* {name: "value", city: "value", interests: [1, 2, 3]

**Modeling our Blog in Mongo**

posts: {title:"free classes", body:"...", author:"me", date:"8/13", comments:[{name:"Joe",

email: "[me@me](mailto:me@me)", comment: "hi"{....}],

{tags: ["cycling", "education", "startups"]}

authors: {\_id: "myId", password:"234"}

**Week 2**

**CRUD** - create = Insert, read = Find, update = Update, delete = Delete

-- Operation exists as methods/functions

**Connecting to the Mongo Shell**

> cd \Program Files\MongoDb 2.6 Standard\bin

-- Use **^** arrow to reload previous statement. Use **Ctrl + A** go to beginning, Home or End

**Inserting Docs**

doc = {'name' : 'smith', 'age' : '30', 'profession' : 'hacker'} //use this in place of doc to direct insert

> db.people.insert( doc ) //interprets as name of collection inside current database

> db.people.find() //see what's there

-- You get back a '\_id' : ObjectId -- which is the primary field

> db.people.findOne({'name' : 'jones' }) //finds document with named parameters

> db.people.findOne({'name' : true, '\_id' : false}) //specifying certain fields to retrieve

**The find field**

> db.scores.find({ student: 19 , type : "essay" }, {"score" : true , "\_id" : false });

**Query Operators**

> db.scores.find({score : { $gt : 95 }}); //score should be greater then 95

> db.scores.find({score : { $gte : 95, $lte : 98 } , type : "essay"}); //between 95-98 w/ type essay

> db.people.find({ name : { $lt : "D" }});

**Using $exists, $type + true or false** - Use to return a list of users meet parameters

> db.people.find ({ profession : { $exists : false }}); //say true or false

> db.people.find ({ name : { $type : 2 }}); //type for 'string' is 2 – search bson types

**Regular Expressions** - Allows you to do complicated string matching inside of fields

> db.people.find({ name : { $regex : "a"}}); //find all people with an 'a' in their name

> db.people.find({ name : { $regex : "$e"}}); //ends with the letter 'e'

> db.people.find({ name : { $regex : "^A" }}); // ^ must begin with whatever follows

**Using $or** - Prefix operator (takes an **array** as an operand) Matches any that meet criteria

> db.people.find({$or : [{ name : { $regex : 'e$' }}, {age : { $exists : true }}]});

**Using $and**

> db.people.find({$and : [{name : { $gt : 'C' }}, {name : { $regex : "a"}}]});

**Querying inside Arrays**

> db.accounts.insert({name : "howard", favorites : ["pretzels", "beer"]});

> db.accounts.insert({ favorites : 'pretzels, name : {$gt : 'H'}});

**Using $in and $all -** Takes in an array //Returns documents with the Field queried

> db.accounts.find({favorites : {$all : ['pretzels', 'beer']}}); //matches All elements in the field

> db.accounts.find({name : {$in : ['Howard', 'John"]}}); //returns docs with at least 1 element

-- If not including the $ operator the order must be the same as in the DB

**Dot Notation**

> db.users.find({"email.work" : "[richard@10gen.com](mailto:richard@10gen.com)"});

> db.catalog.find( { price : { $gt : 10000 } , "reviews.rating" : { $gte : 5 } } );

**Querying Cursors**

> cur.next() //steps through to the next document

**Methods**: hasNext(), next(), limit(), sort(), skip()

> cur.sort({name : -1 }); null; //shows names from Z-A then #'s

> cur.sort({name : -1}).limit(3); null;

> cur.sort({name : -1}).limit(3).skip(2); null;

> db.scores.find({type : 'exam"}).sort({score : -1}).skip(50).limit(20)

**Counting** - Counting up all documents that match a criteria

> db.scores.count({ type : 'exam'});

> db.scores.count({type : 'essay', score : {$gt : 90}});

**Updating Documents**  -- Methods on a collection, replaces 1st with 2nd param

> db.people.update({ name : 'Smith'}, {name : 'Thompson' , salary : 50000}); //replaces smith

**Using $set Command**

-- Find all instances with Alice and set her age to 30

({ name : "Alice" }, { $set : { age : 30 }})

Use **$inc** will increment by 1 ({ name : "Alice" }, { $inc : { age : 1 }}) //makes the age 31

\*\* You can **$inc** by any number (20)

**Using the $unset Command** //removes field

> db.people.update ({name : "Jones" }, {$unset : {profession : 1}})

Using **$push, $pop, $pull, $pushAll, $pullAll, $addToSet**

> db.arrays.update ({ \_id : 0}, {$push : {a : 6}}) // will add 6 to the end of array

> db.arrays.update ({ \_id : 0}, {$pop : {a : 1}}) // will remove last element in array

> db.arrays.update ({ \_id : 0}, {$pop : {a : -1}}) // will remove first element in array

> db.arrays.update ({ \_id : 0}, {$pushAll : {a : [7, 8, 9]}}) //adds 7, 8, 9 to end

> db.arrays.update ({ \_id : 0}, {$pull : {a : 5}}) //removes value no matter location

> db.arrays.update ({ \_id : 0}, {$pullAll : {a : [7, 8, 9]}}) //removes 7, 8, 9 from the array

> db.arrays.update ({ \_id : 0}, {$addToSet : {a : 5}}) // will add 5 to the array if doesn't exist

**Using $upsert** //updates existing record or adds a new one

>db.people.update({ name : 'George' }, {$set : {age : 40}}, {upsert : true})

**Multi-Update** //if you don't supply {multi} then only affects 1 document

> db.people.update({}, {$set : {title : 'Dr'}}, {multi : true}) //open paren updates all docs w/ title

**Removing from the Database**

> db.remove({ name : 'Alice'})

**>** db.people.remove ({})//empty removes all docs from collection

> db.people.drop() //faster way of removing All collections

**Importing Files**

> mongoimport -d course -c grades grades.json //start mongod, import file to node cmd

> mongo //connect to mong

> use course //switched to db course collection

> db.grades.find() //prints out collection items

**Examples**:

findOne() - returns the first document that meets requirements

find (toArray) - returns all students with 100 using the **toArray** method

find\_cursors (each) - finds students with 100 and prints message

find\_lt\_gt - find docs with grades between x and y

**Importing JSON from other sites**

-- request = require('request');

-- request('[http://reddit.json](http://reddit.json/)', function(error, response, body)

var obj = JSON.parse(body);

var stories = obj.data.children.map(function (story) { return story.data; });

db.collection('reddit').insert(stories, function (err, data) {

**Using $regex**

> db.reddit.find({ 'title' : { '$regex' : 'NSA' }}, {'title' : 1, '\_id' : 0}); //finds all docs with NSA in title

**Sort, Limit, Skip** //Order doesn't matter

cursor.sort([['grade', 1], ['student', -1]]);

var options = { 'skip' : 1,

'limit' : 4,

'sort' : [['grade', 1], ['student', -1]] };

var cursor = grades.find({}, {}, options); //empty, projector, call var options

**Find and Modify** // find docs matching query, update and show result

var query = { 'name' : 'comments' };

var sort = [];

var operator = { '$inc' : { 'counter' : 1 } };

var options = { 'new' : true }; //we want the new doc to be returned

db.collection('counters').findAndModify(query, sort, operator, options, function(err, doc) {

**Week 3**

* Rich Documents
* Pre Join / Embed Data
* No Constraints
* Atomic Operations
* No Declared Schema (similar structure)

**Week 4**

* **Indexes**
* -> indexes are collected as top level arrays which can be called in by order when doing query, sort or find.
* (name, hair\_color, dob) - start with name then optional use hair\_color

**Create an Index**

> db.students.ensureIndex({student\_id:1,class:-1}) //makes student ascending and class descend

**Find an Index**

-- You can only use find() and/or sort() on a db when asking for first index

> db.system.indexes.**find()**

> db.students.**getIndexes()** -- Displays the indexes

> db.students.dropInxex({'student\_id':1}) //remove index

**Multikey Indexes - an index w/ an array**

{ name: 'Andrew',

tags: ['cycling', 'tennis', 'football'],

color: 'red'

locations: ['NY', 'CA']}

* You can create a **compound index** on tags/color. But can't on 2 indexes that contain arrays

**Adding a Unique Index**

> db.students.ensureIndex({student\_id:1, class\_id:1}, {unique:true})

**Dropping Duplicate Index**

> db.things.ensureIndex({thing: 1}, {unique: true, dropDups: true})

**Explain** -- provides a break down of the data returned related to query

> db.foo.find({c:1}).**explain()**

**Geospatial Indexes**  - 2d Index

-- 'location' : [x, y]

-- ensureIndex({"location" : '2d', type : 1 }) -- db.stores.ensureIndex (creates index)

> find({location:{**$near:**[x, y]}}).limit(20) -- returns in increasing distance

**Geospatial Spherical** - 2dsphere

* Uses geoJSON (Point, Polygon)
* longitude, latitude

> db.places.ensureIndex({'location': '2dsphere'})

"location" : {

"type" : "Point",

"coordinates" : [-122.169129, 37.4434854]

},

"type" : "Retail"

}

> db.places.find({loc:{$near:{$geometry:{type:"point",coordinates:[122,33]},$maxDistance:2000}}}).pretty()

**Text Searches**

> db.sentences.find({ $text: {$search:'dog' }}) //implies a logical OR operator

**Find 'best' match first**

> db.sentences.find({$text:{$search:'dog tree obsidian'}}, {score:{$meta: 'textScore'}}).sort({score:{$meta:'textScore'}})

**Simple Aggregation**

> db.products.aggregate([{$group:{"\_id":"$category", "num\_products":{"$sum":1}}}])

**Aggregation Pipeline**

$project - reshape - 1 : 1

$match - filter - n : 1

$group - aggregate - n : 1

$sort - sort - 1 : 1

$skip - skips - n : 1

$limit - limits - n : 1

$unwind - normalize - 1 : n (expands items in array)

$out - output - 1 : 1