**WEEK1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**What is mongoDB**

Non relational database- no tables KEY VALUE

Json : javascript object notation {“name”: “document”}

{“a”: 4,”b”:5,”c”,7} {a:6, b:7, fruit: [“apple”, “pear”, “banana”]}

**schemaless:**

{a:3,b:7}

{a:7,b:5,c:6}

**Mongodb relative to relational**

* does not support joints not scalable
* does not support transactions because difficult to support transactions in a system that is horizontally scalable

**Asynchronous vs synchronous**

Handling post request

\* app.use(express.bodyParser())—middleware for express – parse the body

**mongodb schema???**

**INTRO TO NPM**

**Json revisited**

[] arrays- list of thinga

dictionaries- associate maps

{keyword:value,}

{name:”fodo”, city:”value”, tenats[{dictionary},{dictionary}]}

{“address”: {“street\_address” : “23 Elm Drive”, “city”:”Palo Alto”,“state”:”California”, “zipcode”:”94305”}}

**Modeling our blog in Mongo**

**Posts collection**

{title:”free online class”,body:”…”, author:”gianni”,date:”…”, comments:[{name:”joe bidon”, email:”joe@gmail.gov”,comment:”…”},

{………}],

tags:[“cycling”,”education”,”startups”]}

**Authors collection**

{\_id:”evildean”, password:”……”}

**Intro to schema design**

To embed or not to embed

16mb data allowed per document/object: {}

**Hello world using express**

Var express = require(‘express’),

App = express();

App.get(‘/’, function (req, res) {

Res.send(“Hello, World!”);

});

App.get(‘\*’, function (req, res) {

Res.send(“Page not found, 404 error”);

});

app.listen(8080);

**HELLO WORLD USING EXPRESS AND SWIG**

Var express = require(‘express’),

App = express(),

Cons = require(‘consolidate’);

App.engine(‘html’, cons.swig);

App.set(‘view engine’, ‘html’);

App.set(‘views’, \_\_dirname + “/views”);

App.get(‘/’, function (req, res) {

Res.render(‘hello’, {‘name’: ‘swig’});

});

App.get(‘\*’, function (req, res) {

Res.send(“Page not found, 404 error”);

});

app.listen(8080);

1. make directory for html pages. views

**HELLO WORLD USING EXPRESS SWIG AND MONGODB**

Var express = require(‘express’),

App = express(),

Cons = require(‘consolidate’),

MongoClient = require(‘mongodb’).MongoClient,

Server = require(‘mongodb’).Server;

App.engine(‘html’, cons.swig);

App.set(‘view engine’, ‘html’);

App.set(‘views’, \_\_dirname + “/views”);

Var mongoclient = new MongoClient(new Server(‘localhost’, 27017,

{‘native\_parser’: true}));

Var db = mongoclient.db(‘course’);

App.get(‘/’, function (req, res) {

Db.collection(‘hello\_mongo\_express’).findOne({}, function(err, doc){

Res.render(‘hello’, doc);

});

});

App.get(‘\*’, function (req, res) {

Res.send(“Page not found, 404 error”);

});

mongoclient.open(function(err, mongoclient){

if (err) throw err;

app.listen(8080);

console.log(“express server 8080”);

});

**EXPRESS : HANDLING GET REQUESTS**

url parameters:

get variables: /:name puts the get request into name and if loading page :8080 will result in cannot get / because you have to specify the url to :8080/gio.

**WEEK2 CRUD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

BSON introduced

Obj = {“a”:1, “b”:”hello”,”c”:[“apples”,tomatoes]}

INSERTING DOCS

Doc = {“name”: “smith”, “age”:30, “profession”: “hacker”}

Documents live inside collections

Db.people.insert – people is the collection, insert is a method takes variable doc

Db.people.find()

INTRODUCTION TO FINDONE

Db.people.findOne()- brings back 1 random document from collection

Db.people.findOne({“name”:”jones”})-brings back name with jones

Db.people.findOne({“name”:”jones”},{“name”:true,”\_id”:false}) brings back name only without \_id

QUERUING INSIDE ARRAYS

Db.accounts.insert({name:”howard”, favorites: [“pretzels”,”beer”]})

Db.accounts.find({favorites: “pretzels”});

USING $IN AND $ALL

Db.accounts.find({favorites:{$all:[“pretzels”,”beer”]}})- brings back everything in any order

Db.accounts.find({name:{$in:[“Howard”,”John”]}})

QUERIES WITH DOT NOTATION

Db.users.find({email:{work:[email@gmail.com](mailto:email@gmail.com), personal:”email@gmail.com”}})

Db.users.find({“email.work”: “emailwork@gmail.com”}) allow to reach inside nested document. Fixed depth path expression in bson.

QUERYING, CURSORS

Db.people.find() – cursors iterates all elements and prints them to shell

Cur = db.people.find(); null;

Cur.hasNext(); - true

Cur.limit(5) –returns 5 documents

Cur.sort({name:- 1 }) returns results in reverse order by name

Cur.skip(2) – skips over first 2

COUNTING RESULTS

Db.users..count({type:”exam”}) calls how many documents there are and returns number

WHOLESALE UPDATING OF A DOCUMENT

Db.people.update({name:”smith”}, {name::”thompson”, salary:50000}) updates everything but replaces everything so its kinda dangerous

USING THE $SET COMMAND

Db.people.update({name:”Alice”}, {$set:{age:30}}) updates or adds the age field

USING THE $UNSET COMMAND

Dp.people.update({name:”Jones”},{$unset:{profession:1}})

db.users.update({\_id:"jimmy"},{$unset:{interests:1}})

USING $PUSH $POP $PULL $PUSHALL $PULLALL $ADDTOSET

Db.array.update({\_id:0}, {$push:{a:6}}) adds element to right hand side of array and extemds

Db.array.update({\_id:0}, {$pop:{a:1}}) removes right most operator with $pop

Db.array.update({\_id:0}, {$pushAll:{a:[7,8,9]}}) adds all to array full on array

Db.array.update({\_id:0}, {$pull:{a:5}}) pulls out number 5

Db.array.update({\_id:0}, {$push:{a:[2,4,8]}}) removes these from array

Db.array.update({\_id:0}, {$addToSet:{a:5}}) is like push

UPSERTS

Db.people.update({name:”George”}, {$set:{age:40}},{upsert:true}) – update record if does exists or add a new one if doesn’t

MULTI-UPDATE

Db.people.update({}, {$set:{title:”Dr”}},{multi:true}) traverse every document in collection and add a new field

Db.score.update({‘score’: {$lt:70}},{$inc:{score20}},{mulit:true})

**REMOVING DATA**

Db.people.remove({name: “Alice”})

Db..people.remove({})

Db.people.drop() – evauluates to true if they have been dropped

**NODEJS DRIVER: FIND FINDONE AND CURSORS**

mongoimport –d course –c grades grades.json = -d:database, -c:collection & file.json to import a file into mongo

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//query for students that got 100

var query = {'grade':100};

//perform findOne query against mongodb and log 1 document to the console.

db.collection('grades').findOne(query, function(){

if(err) throw err;

console.dir(doc);

db.close();

});

//log the documents to the console in an array.

db.collection('grades').find(query).toArray(function(err, docs){

if(err) throw err;

console.dir(docs);

db.close();

});

});

cursors: Do not perform a query until it gets a callback, basically putting your query into a variable and will return when doing the below functions.

.toArray\/\/\/\/

cursor.each(function(err, doc){

if(err) throw err;

If(doc == null) {

Return db.close()

}

console.log(doc.student + “got a good grade!”);

})

**NODEJS DRIVER:USING FIELD PROJECTION**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//query for students that got 100

var query = {'grade':100};

var projection = {'student': 1, '\_id':0};

//

db.collection('grades').find(query, projection).toArray(function(err, docs){

if(err) throw err;

//iterating through all documents

docs.forEach(function (doc){

console.dir(doc);

console.dir(doc.student + " got a good grade!");

});

db.close();

});

});

**NODEJS DRIVER:USING $GT AND $LT**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//query projection

var query = {'student': 'Joe', 'grade':{'$gt':80, '$lt':95}};

db.collection('grades').find(query).each(function(err, doc){

if(err) throw err;

if(doc == null){

return db.close();

}

console.dir(doc);

});

});

**IMPORTING REDDIT**

Brings .json from reddit page and we do a query to bring back the child data set

**NODEJS DRIVER USING $REGEX**

Db.reddit.find({‘title’:’NSA’})

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//brings back title with NSA in it

var query = {'title':{'$regex': 'NSA'}};

var projection = {'title':1, '\_id':0};

db.collection('reddit').find(query, projection).each(function(err, doc){

if(err) throw err;

if(doc == null) {

return db.close();

}

console.dir(doc.title);

});

});

**NODEJS DRIVER: USING DOT NOTATION**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//queries something nested inside media - json

var query = {'media.oembed.type': 'video'};

//remove all fields on top level and in oembed reuturns only url

var projection = {'media.oembed.url': 1, '\_id': 0};

db.collection('reddit\_front').find(query, projection).each(function(err, doc){

if(err) throw err;

if(doc == null) {

return db.close();

}

console.dir(doc);

});

});

**NODEJS DRIVER: SKIP LIMIT AND SORT**

First we sort then we skip then we limit

Sort-skip-limit – very strict positioning

Before retrieving documents\*

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

var grades = db.collection('grades');

var cursor = grades.find({});

cursor.skip(1);

cursor.limit(4);

//ascending order sort -1 = descending

cursor.sort('grade',1);

cursor.each(function(err, doc){

if(err) throw err;

if(doc == null) {

return db.close();

}

console.dir(doc);

})

});

**NODEJS DRIVER INSERTING \_ID**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

var doc {'student': 'Calvin', 'age': 6};

db.collection('students').insert(function(err, inserted){

if(err) {

console.log(err.message);

return db.close();

};

console.dir("Successfully inserted: " + JSON.stringify(inserted));

return db.close();

});

});

**NODEJS DRIVER: UPDATES**

Update –replacement – replace the entire doc

Update – in place – use update operators to modify doc without completely replacing it.

Update – multi – apply to large number of docs using single operation.

//replace

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

var query = {'assignment': 'hw1'};

db.collection('students').findOne(query, function(err, doc){

if(err) throw err;

if(!doc){

console.log('No docuements for assignment ' + query.assignment + ' found!');

return db.close();

}

query['\_id'] = doc['\_id'];

doc['date\_returned'] = new Date();

db.collection('grades').update(query, doc, function(err, updated){

if(err) throw err;

console.dir("Successfully updated" + updated + " docuement");

return db.close();

});

});

});

//in place replacement

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

var query = {'assignment': 'hw1'};

var operator = {'$set': {'date\_returned': new Date()}};

db.collection('grades').update(query, operator, function(err, updated){

if(err) throw err;

console.dir("Successfully updated " + updated + " document");

return db.close();

});

});

//multi updates

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

var query = { };

var operator = {'$unset': {'date\_returned': ''}};

var options = {'multi': true};

db.collection('grades').update(query, operator, options, function(err, updated){

if(err) throw err;

console.dir("Successfully updated " + updated + " document");

return db.close();

});

});

**NODEJS DRIVER: UPSERTS**

**Upsert vs save**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//insert then update

var query = {'student': 'frank', 'assignment':'hw1' };

var operator = {'student': 'frank', 'assignment':'hw1', 'grade':100};

var options = {'upsert': true};

db.collection('grades').update(query, operator, options, function(err, upserted){

if(err) throw err;

console.dir("Successfully upserted " + upserted + " document");

return db.close();

});

});

**NODEJS DRIVER: FINDANDMODIFY**

Query -> update : replaces doc when updating

findAndModify: atomically updates and returns the document. Specified if you wanna return the new or old document.

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//insert then update

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

var sort = [];

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

var options = {'new': true};

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

if(err) throw err;

if (!doc) {

console.log("No counter found for comments");

} else {

console.log("Number of comments " + doc.counter)

}

return db.close();

});

});

**NODEJS DRIVER: REMOVE**

var MongoClient = require('mongodb').MongoClient;

//connects to database course

MongoClient.connect('mongodb://localhost:27017/course', function(err,db){

//if there is an error connecting throw err

if(err) throw err;

//insert then update

var query = {'assignment': 'hw3'};

db.collection('grades').remove(query function(err, removed){

if (err) throw err;

console.dir("Successfully removed " + " documents");

return db.close();

});

});

**BUILDING A BLOG ON NODEJS**

App.js<-routes<<<-- sessions.js,posts,js,users.js

WEEK3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MONGODB SCHEMA DESIGN**

\*Rich documents -> store array of items, value of key for entire document, prejoin data for fast access.

\* pre join data / embed data –

\* No Mongo Joins –

\* No constraints

\* Atomic operations – organize data to support atomic operations

\* No Declared Schema – is going to have a schema (similar structure).

Single most important factor in designing schema is to match the data access patterns in your application.

**RELATIONAL NORMALIZATION**

mongo

* (free database of modification anomalies ) – worry about these
* (Minimize the redesign when extending the database) – worry about these
* Avoid bias toward any particular access pattern. – this is the one that mongodb solves

**MONGO DESIGN FOR BLOG HOMeWORK Help**

**ALTERNATIVE SCHEMA FOR BLOG**  & Living without constraints

Foreign key constraint – store something into a collection and pull it into another collection.

**LIVING WITHOUT TRANSACTIONS**

Atomic operation do exist in mongodb – updating everything at once instead of doing things 1 by 1.

Benefits:

1. Restructure code so that your working in 1 document
2. Implement locking and software
3. To tolerate a little inconsistency that is someone temporary

**ONE TO ONE RELATIONS**

1to 1 relation: relations where each item corresponds to one other item.

Considerations:

Frequency of access – if it’s a large collection, keep them in separate collections because it is barely used.

Size of items – point beyond which documents need to move into their own collection.

Atomicity in data – if I want to update everything at same time, I may want to embed the document into same collection.

**ONE TO MANY RELATIONSHIPS**

Where 2 entities or many entities that match to 1 entity.

Use true linking – (many 8mill)people collection{name:Andrew, city: NY} LINKS INTO city collection (few){\_id:NYC}

one to few – if there aren’t many than embedding it is OK.

**MANY TO MANY RELATIONS**

Books:Authors – students:teachers **FEW:FEW**(BOOKS TO AUTHORS)

Link by creating an array in document

**MULTIKEYS**

Mulitkey indexes are useful because

Db.students.ensureIndex({‘teachers’:1})

Db.students.find({‘teachers’:{$all:[0,1]}}).explain()

**BENFITS OF EMBEDDING**

Improved read performance one round trip to DB

**TREES**

Be able to list ancestors or children easier than relational database. – Solution: to list the ancestors

**DENORMAILIZATION**

Why you normalize data in relation world want to avoid anomalies in data

Why in mongodb do we denormalize

**WEEK4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Storage Engines introduction:**

What is a storage engine: the interface between the persistence storage and the database itself.

2 main storage engines :

MMAP:

&

WiredTiger

**STORAGE ENGINES MMAPV1**

If we have a 100gb hard disk storage MMAP changes it into virtual memory which would be using a 64 bit computer.

Collection level locking: if you have 2 different operations going on at the same time one is going to have to wait for another if they both are right, but if operations are happening in different collections it s happening simultaneously

**WIRED TIGER STORAGE ENGINE:**

1. Document level concurrency
2. Offers compression of data and indexes
3. No Implace

To start with this:

Killall mongo

Mkdir WT

Mongod –dbpath WT --storageEngine wiredTiger

Mongo

To check if wired or mmap: db.foo.stats()

**INDEXES**

Indexing slows down writes but reads faster

**CREATING INDEXES**

**DISCOVERING AND DELETING INDEXES**

Db.students.getIndexed();

**MULTIKEY INDEXES**

No parallel indexes so cannot have two arrays when put into