Databases & Schema Design

Intro to MongoDB, Object Database Mappers and Schema Design

Outline

- History of Database
- Why MongoDB
- Mongoose — Connecting MongoDb and Node.js
- Tooling
- Workshop

What is a Database?

- Excel
- Cellphone directory
- Inbox
- Folder System
- The Web

Database Management Sys. (DBMS)

- How to define records (Data Definition)
- Creating/Updating/Deleting Records
- Querying/Retrieval of Records
- Administering the database system

Progression of Databases

- Navigational (< 1970s)
- Relational (> 1970s)
- NoSQL (> 2000s)

MongoDB History

- Founded in 2007 as 10gen
- Engine for Google Apps
- Open Source, but driven by the company MongoDB
- Puts the M in MEAN

NoSQL: MongoDB

- **Not Only SQL — Can have SQL-like queries, but:**
- **Does not restrict to storing info in tables (relations)**
- **Stores information in BSON (Binary JSON) *documents***
  - JSON (JavaScript Object Notation) is a subset of actual JS objects:
    - ```
      {
        "firstName" : "Frodo",
        "age" : 33,
        "possessions" : ["ring", "sword"]
      }
      ```
  - Must use double quotes, including around key names.
  - BSON is a superset of JSON which is stored as binary values, including some additional data types (e.g. integers, dates) and metadata (number of bits)
      ```

Collections vs. Tables

- The Mongo equivalent of a table is a *collection*
- Collections don't have schemas — docs can vary

Documents vs. Rows

- Each piece of data in Mongo is called a Document
- Not PDF or DOC files, but like JS objects or associative arrays
- A Document is a JSON object, like JavaScript objects
- Each Document can have its own design

Mongo vs. SQL vs. Excel

Excel	SQL	Mongo
Excel File	Database	Database
Worksheet	Table	Collection
Row	Row	Document
Column	Column	Field
	Index	Index
VLOOKUP	Join	Embedding
	Primary Key: id	Primary Key: _id

SQL to NoSQL

SQL Term	Mongo NoSQL Term
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index
Join	Embedding & Linking
Primary Key	_id field
Group By	Aggregation

Talking to Mongo using JS

- Talk to Mongo using JavaScript functions
- The “db” object is the pathway to the DB

SQL	Mongo Statement
CREATE TABLE users (name varchar, age int)	Done automatically while inserting
INSERT INTO users (name, age) VALUES (“Nicole”, 24)	db.users.insert({“name”:”Nicole”, “age”:24});
SELECT * FROM users;	db.users.find()
SELECT * FROM users WHERE age=24;	db.users.find({“age”:24})
SELECT * FROM users WHERE name=“Jay” ORDER BY name ASC;	db.users.find({“name”:”Jay”}).sort(“name”:1)

MongoDB Strengths

- **Schema-less (no enforced document structure)**
  - Collections can have mixed docs!
  - Agile development - changing, flexible
  - Capturing unstructured information
- **Preplanning for scale**
- **Use cases:**
  - Archiving / event logging (Hummingbird)
  - Document / content management (Forbes)
  - Gaming (EA FIFA)
  - Mobile & location-based (Foursquare)
  - Realtime statistics (Crittercism)

MongoDB Features

- **Geospatial data type — Foursquare**
  - Spherical searches
  - Inclusion search (within polygons)
- **Horizontal scaling (replica sets, sharding)**
- **Aggregation (map-reduce)**
- **Stored JS functions**
- **Low JS impedance**
- **Rich query capabilities**

MongoDB Weaknesses

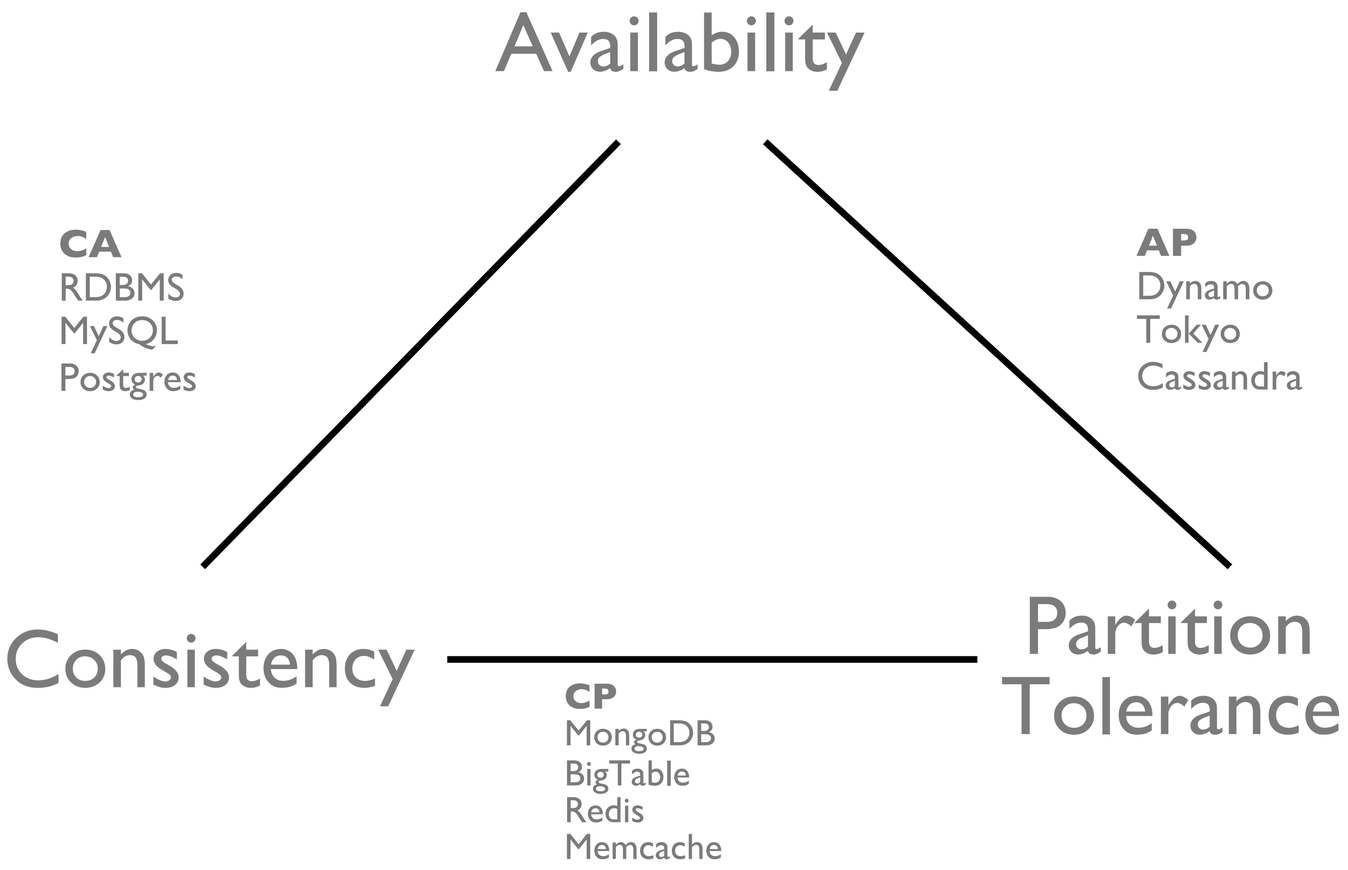
- **MongoDB is consistent at the *document* level, but does not have ACID-compliant multi-document *transactions***
- **ACID (key in banking / accounting)**
  - Atomic
  - Consistent
  - Isolated
  - Durable
- **Transactions**
  - Cannot change multiple docs at the same time in an all-or-none fashion
- **Maybe you *want* SQL**

CAP "Theorem"

- Consistency: all clients see the same data
- Availability: each client can always read or write
- Partition Tolerance: system works even if network splits

- "Pick 2"
  - (Reality is more complex…)

CAP, metaphorically speaking


Availability
CA
RDBMS
MySQL
Postgres
AP
Dynamo
Tokyo
Cassandra
Consistency
Partition Tolerance
CP
MongoDB
BigTable
Redis
Memcache

CRUD in Mongo

- Create
- Read / Query
- Update
- Delete

Create/Insert

```
use demodatabase;
db.users.insert({
  firstName:  "David",
  lastName:   "Yang",
  username:   "davidyang",
  password:   "74824h234b280412312fdgsdgae"
});
```

Read

```
> db.users.find()
[{
  "_id" : ObjectId("52f1014ace52721056688d7a"),
  "firstName" : "David",
  "lastName" : "Yang",
  "username" : "davidyang",
  "password" : "74824h234b280412312fdgsdgae"
}]

> db.users.findOne()
{
  "_id" : ObjectId("52f1014ace52721056688d7a"),
  "firstName" : "David",
  "lastName" : "Yang",
  "username" : "davidyang",
  "password" : "74824h234b280412312fdgsdgae"
}
```

Update

```
db.users.update(queryObject, updateCommandObject);

db.users.update(
  { "username": "david" }, // query object
  { "username": "david", "favorite_movies" : [
    "Braveheart", "Terminator 2"
  ]} // update command
)

db.users.update(
  { "username": "david" },
  { $set: { "favorite_movies": ["Braveheart", "Johnny Bravo"] } }
)
```

Delete

```
db.users.remove( {_id: ObjectId("52f1014ace52721056688d7a")} )

> db.users.count()
0
```

Querying

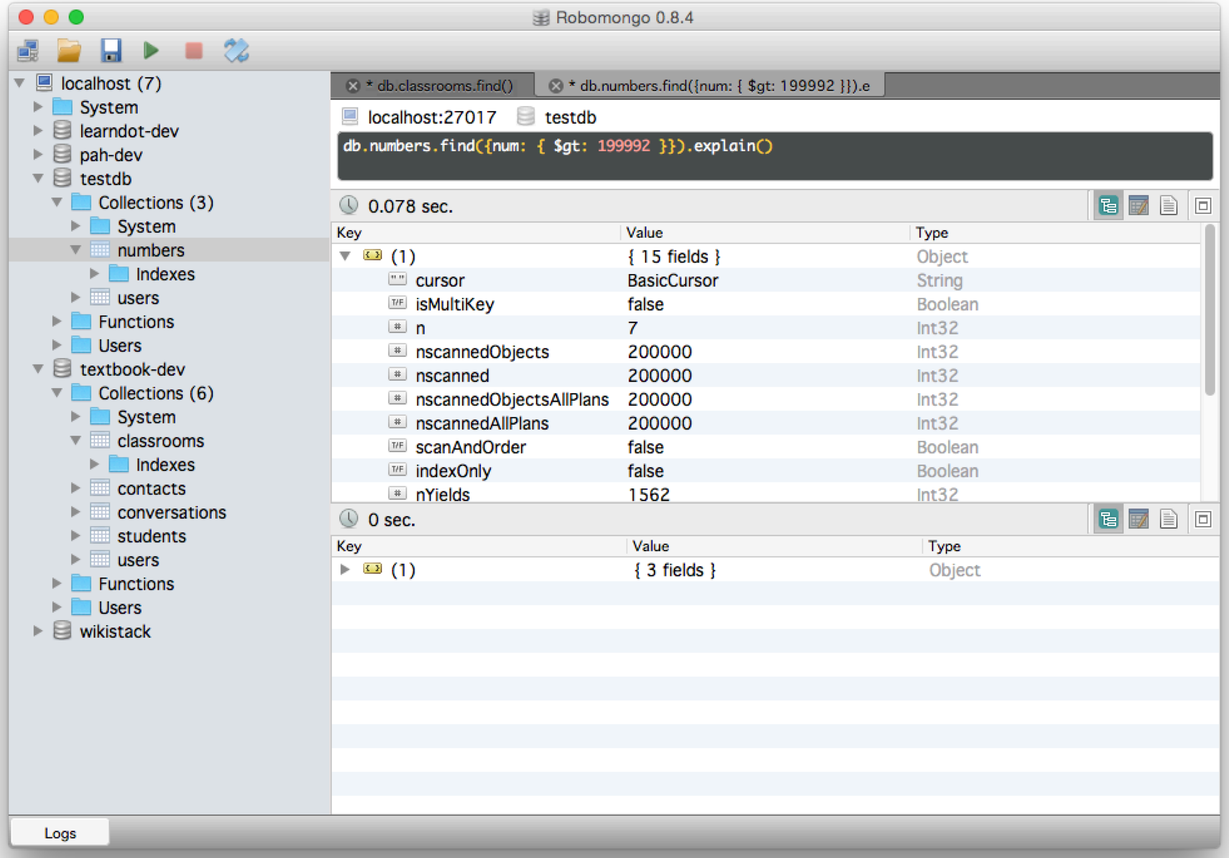
- Matchers
- Range ($gt, $lt, $gte, $lte)
- Set Operators ($in, $nin, $all)
- Boolean Expressions ($or,
- Arrays
- Regular Expressions (/regex/)
- JavaScript (functions)

Indexes

- Indexes are the primary tool DB administrators have to improve performance of querying
- Like the index of book, without one, any time you want to find something, you have to scan the entire table

Tooling

- Robomongo



Tooling

- Mongohacker

This repository Search Explore Gist Blog Help davidy

TylerBrock / mongo-hacker Watch 46 S

MongoDB Shell Enhancements for Hackers http://tylerbrock.github.io/mongo-hacker

193 commits 2 branches 4 releases 26 contributors

branch: master mongo-hacker / +

Merge pull request #113 from tobsn/patch-1 ...

TylerBrock authored on Jan 9 latest commit 3e776b455b

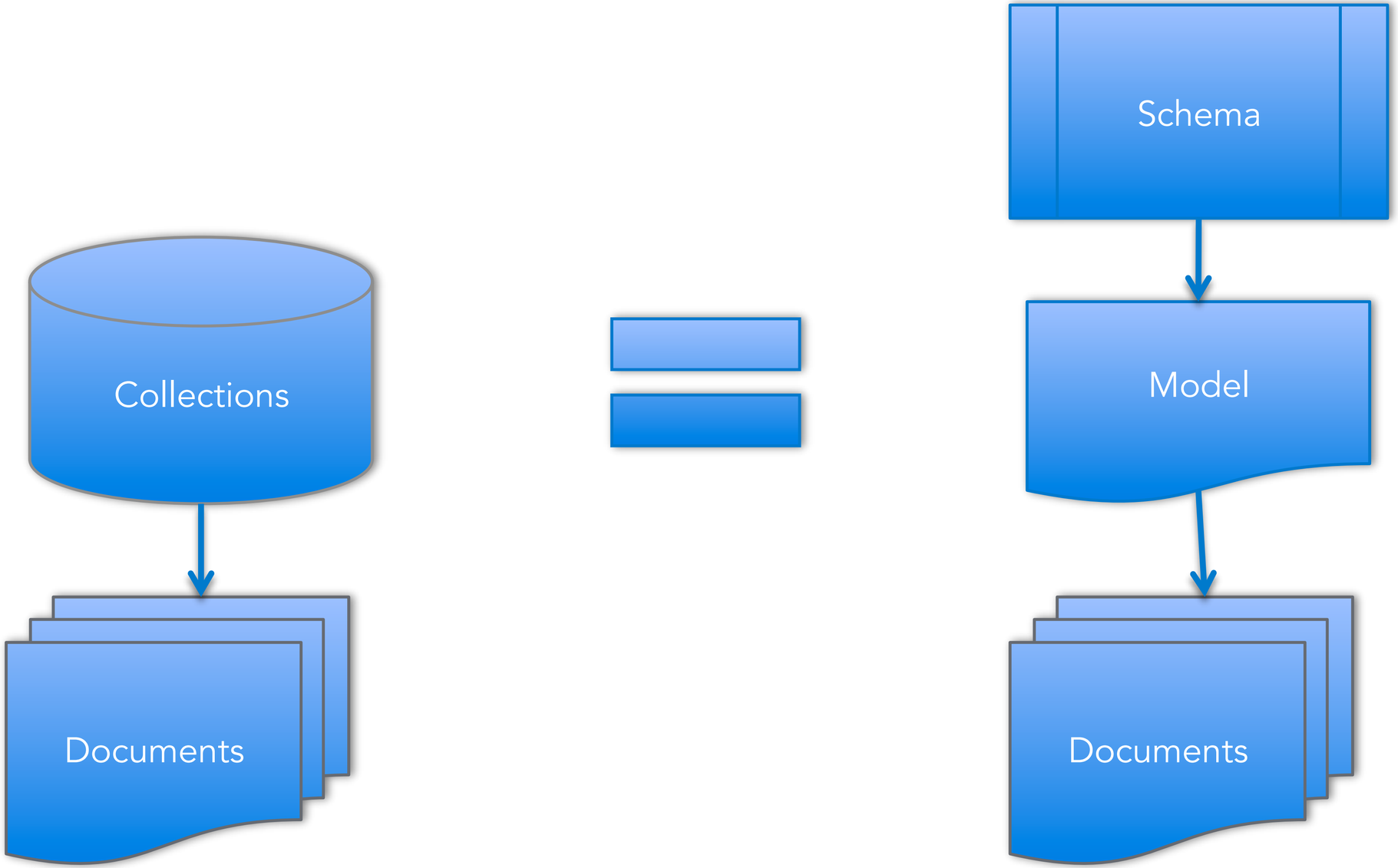
hacks	changed n._limit to n / n is an integer and has no _limit attribute	2 months ago
.gitignore	adding built mongo_hacker output to gitignore	2 years ago
LICENSE	Added license file	5 months ago
Makefile	minor: restore original .mongorc.js on uninstall	4 months ago
README.md	Update README.md	2 months ago
base.js	Fix #76: Add Support for windows	5 months ago
config.js	change default key color to grey to increase visibility	4 months ago
package.json	bump 0.0.4	2 months ago

README.md

MongoDB Shell Enhancements

Mongoose

- **Mongoose is an Object-Document Mapper that makes it easy to access MongoDB from Node.js**
- **Mongoose Features:**
  - Schema modeling/validation
  - Data casting (convert mongo types to JS types)
  - Query building
  - Hooks (code that runs pre/post save/delete/update)
  - Class and instance methods of models
  - Getters and setters


Collections
Documents
Schema
Model
Documents

Mongoose Basics

- Make a Schema (interactive blueprint object)
- Extend the Schema with Hooks, Methods, Statics, Virtuals, etc.
- Use the completed Schema to build a Model (object that allows interacting with a Mongo collection — also, can act a constructor function)
- Use the Model (Collection) to create/find Instances
- Use the Instances (individual Docs) to save/update/delete

Create a Schema

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  author: String,
  body: String,
  comments: [{body: String, date: Date }],
  date: {type: Date, default: Date.now },
  hidden: Boolean,
  meta: {votes: Number, favs: Number }
});
```

Build Model from Schema

```
var Blog = mongoose.model('Blog', blogSchema);
```

Model & Instance usage

```
var myBlog = new Blog({
  title: "David's Blog",
  author: "David"
});

myBlog.save(function(err, myBlog) {
  if(err) console.log("Error saving", err);
  console.log(myBlog);
});

Blog.find({ title: /David/ });
```

Mongoose

- Lives inside Node.js process
- knows how to communicate via the MongoDB Binary Protocol to the MongoDB Server

Schema Design

- **How do we store data in Mongo?**
  - Non-relational
  - No support for Database Joins
- **Since MongoDB documents are just JSON documents they support array and object embedding**
  - New choice in MongoDB: do I embed this document or reference?
  - MongoDB doesn't support JOINs, you have to do it in your JavaScript application

Background: SQL & Joins

- **Three Basic Principles in SQL**
  - Data is tabular
  - Each cell of the table contains one value (no arrays/collections/objects)
  - No redundancy of data (each row represents the single-source of truth about that data)
- **Storing multiple data points (relations)**
  - Avoid redundancy / multiple columns / complex updates using join tables
  - Cost is that performing a join is slow

Embedding

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  body: String,
  comments: [{
    body: String,
    date: Date
  }]
});
```

OR

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var commentSchema = new Schema({
  body: String,
  date: Date
});

var blogSchema = new Schema({
  title: String,
  body: String,
  comments: [commentSchema]
});
```

Referencing

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var commentSchema = new Schema({
  body: String,
  date: Date
});
var Comment = mongoose.model('Comment', commentSchema);
var blogSchema = new Schema({
  title: String,
  body: String,
  comments: [{
    type: Schema.Types.ObjectId,
    ref: 'Comment'
  }]
});
```

Embedding vs. Referencing

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  body: String,
  comments: [{
    body: String,
    date: Date
  }]
});
```

```
comments: [{
  body: String,
  date: Date
}]
```

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var commentSchema = new Schema({
  body: String,
  date: Date
});
var Comment = mongoose.model('Comment', commentSchema);
var blogSchema = new Schema({
  title: String,
  body: String,
  comments: [{
    type: Schema.Types.ObjectId,
    ref: 'Comment'
  }]
});
```

```
comments: [{
  type: Schema.Types.ObjectId,
  ref: 'Comment'
}]
```

Schema Choices

- **Cardinality**
  - 1-1
  - 1-Many
  - Many-Many
- **Foreign Keys?**

One to One

- Almost always embed 1-1 relationships in MongoDB
  - User and UserProfile
  - User and AccountPreferences
  - User and Photo
- Allows us to get all the info with one query

One to Many

- **Split into One-to-Few and One-to-Many**
- **One-to-Few**
  - Prefer embedding
  - Examples: User and Addresses, Blog Post to Category
- **One-to-Many**
  - Prefer References
  - Examples: User and Posts, Blog Post to Comments
  - Who gets the reference? Typically the "many" (posts/comments in this example)

Many to Many

- Social Networking's Many to Many problem
- Can embed followers or followees as references
- What happens on twitter when you have more than 16MB of references?
  - One document can reference the next document of followers
  - Linked list

SQL to Mongo Conversion

Movie Genres

Movies

Roles

Actors

Movies Directors

Directors

Directors Genres

Movies to MovieGenres

- Few to Many
- From movies perspective, it's One to Few
- Recommendation: embed as string array

Movies to Directors

- Many to Many
- From both movies and directors perspective, only a few
- Recommendation: embed in movies (why?)

Movies to Actors

- Many to Many
- From both movies and directors perspective, only a few
- Where does the role information go?
- Recommendation: embed both in movies (why?)

Workshop

- **WikiStack**
  - Walk you through installing and using MongoDB
  - Wikipedia for Fullstackers
  - Application of everything we've learned so far
  - We will do intermediate reviews to ensure everyone is on track