MongoDB w/ Some Node.JS Sprinkles

Niall O'Higgins
Author “MongoDB and Python” O'Reilly

@niallohiggins on Twitter
niallo@beyondfog.com

MongoDB Overview

- Non-relational (NoSQL) document-oriented database
- Rich query language
- Flexible data model (JSON)
- Auto-sharding and replication with automatic-failover
- No Transactions nor joins

npm install mongodb

- Automatic re-connect with Replica Sets
- Failover works very well
- Asynchronous by design
- Higher level ODMs available:
  - Mongoose
- Or try Mongolian Deadbeef

MongoDB vs MySQL vs BDB

Feature Comparison

	MongoDB	MySQL	BDB
Master-Slave Replication	Yes	Yes	No
Automatic Master Failover	Yes	No	No
Ad-hoc Schemas	Yes	No	No
Manual Index Specification	Yes	Yes	Yes
Rich Query Language	Yes	Yes	No
Joins	No	Yes	No
Transactions	No	Yes	No
Auto-Sharding	Yes	No	No

MongoDB: How NoSQL is it?

- Closer to MySQL than to BDB/Tyrant/Redis
- Less emphasis on Google-scale scalability *at this point* compared to e.g. Cassandra/HBase
- Today, MongoDB scales slightly better than MySQL out of the box IMHO.
- Main advantages over RDBMS are ease of development, flexible data model!

MongoDB + JavaScript

- Think in documents
- A document is a JSON object
- JavaScript driver exposes these as JS Objects

  E.g. db.mycollection.insert({myprop:”foo!”});
- Anything you can do in JSON, you can do in MongoDB
- Documents are grouped logically in ”Collections” - analogous to a table in SQL.
- **Unlike SQL, not all documents in a Collection must have same schema!**

MongoDB Data Model Example

Concept	SQL	MongoDB
One User	One Row	One Document
All Users	Users Table	Users Collection
One Username Per User (1-to-1)	Username Column	Username Property
Many Emails Per User (1-to-many)	SQL Join With Emails Table	Embed In User Document
Many Items Owned By Many Users (Many-to-Many)	SQL Join With Items Table	Code Join With Items Collection

MongoDB Data Model Example II

- Add a new user

  db.users.insert({”username”:”niallo”, ”emails”: [”niallo@beyondfog.com”]});

- Add a new email to user

  db.users.update({”username”:”niallo”}, {”$push”: {”emails”:”niallo@niallohiggins.com”}});

- Add item owned by niallo and jill – data is string

  db.items.insert({”owners”:[”niallo”, ”jill”], ”data”:”foo”});

- Get all of niallo's items

  var items = db.items.find({”owner”:”niallo”});

MongoDB Data Model Example III

- Get jill's items where data is ”foo”

  var foo_items = db.items.find({”owner”:”jill”,”data”:”foo”});
- Add an item owned by jill - data is number 321

  db.items.insert({”owner”:”jill”, ”data”:321});
- Get jill's items where data is greater than 2

  var gt_2 = db.items.find({”owner”:”jill”,”data”:{”$gt”:2}});
- Get all items owned by either niallo or jill

  var niallo_jill_items = db.items.find({”owner”:{”$in”:[”niallo”,”jill”]}});

- All these examples can make use of an index!

More on Querying MongoDB

- $lt, $lte, $gt, $gte === <, <=, >, =>
- $ne === <>, !=
- $in === IN
- $or === OR
- Sort() === ORDER BY
- Limit() === LIMIT
- Skip() === OFFSET
- Group() === GROUP BY

MongoDB Gotchas

- Size limit per document: 16MB but increasing.
- Cap on number of indexes per collection.
- No transactions, although good support for atomic operations.
- → Not the right fit for every system!
- → Think through your schema!

Node.JS with Mongoose

- Mongoose is a convenient, lightweight ODM layer.
- Provides schema validation, index config, virtuals and custom setters and getters.
- Also enables mongoose-auth, a MongoDB storage backend for everyauth. Works with connect and express.

Mongoose Sample Model

```
var JobSchema = new Schema({
    _owner : { type: Schema.ObjectId, ref: 'user' }
  , deploy_exitcode : Number
  , created_timestamp : Date
  , finished_timestamp : Date
  , repo_url : String
  , test_exitcode : Number
  , type : String
});
var Job = mongoose.model('Job', JobSchema);
var job = new Job({
      _owner: user._id
    , created_timestamp: new Date()
    , finished_timestamp: null
    , type: "TEST_ONLY"
    , repo_url: repo_config.url
});
job.save(function callback(err, user) { });
```

Mongoose Sample Model II

```
// Custom getter & setter for transparent encryption/decryption
var RepoConfig = new Schema({
    pubkey: {type: String, get: decrypt, set: encrypt }
  , privkey: {type: String, get: decrypt, set: encrypt }
  , secret: {type: String, get: decrypt, set: encrypt }
});

// Virtual method, RepoConfig.has_prod_deploy_target()
RepoConfig.virtual('has_prod_deploy_target').get(function() {
  return (this.prod_deploy_target !== undefined &&
    this.prod_deploy_target.provider !== undefined &&
    this.prod_deploy_target.account_id !== undefined);
});
```

MongoDB Pattern: Fast Accounting

- You want to keep track of something on a per-user basis
- Not just total over all time, but “today”, “last month”, “3 weeks ago” etc.
- Consider high score tables, services with activity-based billing, activity feeds, etc.
- This needs to be ***very fast***. Both to read and write.

MongoDB Pattern: Fast Accounting

- Keeping a log collection works, but retrieval is slow.
- Faster technique is to use separate properties per time period.
- Add one key for each day/week/month/year
- MongoDB $inc operator is fast & atomic
- For example, a high score table with week, month and total resolution

MongoDB Pattern: Fast Accounting

- 52 weeks per year, include year in key name to disambiguate
- var user_doc = { “scores_weekly”:

  { “2012_01” : 10,

  “2012_02” : 20 }};
- “Score this week”

  var sw = user_doc[“scores_weekly”][“2012_”+weeknum];

MongoDB Pattern: Fast Accounting

- Fast, atomic updates to counts via $inc / $dec
- db.users.update({“_id”:blah}, {“$inc” :
  { “scores_weekly.2012_” + weeknum : 1 }});
- Awesome for scores, activity, billing systems, etc.
- Can still use logging-style collection for more details. Analyze offline in batch mode.
- Great example of pattern hard to do in RDBMS

MongoDB Pattern: Location Aware

- Many mobile apps today need the feature of finding people or things ***near you***
- MongoDB has support for this out-of-the-box
- Geospatial Indexing works well
- For city or country granularity it might suffice to set user location at registration time
- Otherwise on app start

MongoDB Pattern: Location Aware

- MongoDB $near sorts documents by proximity to single point
- Databases exist with bounding-box co-ordinates for places (cities, countries, etc)
- Use $within operator to find all documents in those boxes
- NB today Geospatial Indexing is all point based – no track / route support yet

Fin

- Thanks for checking out my talk
- Reach me at niallo@beyondfog.com or @niallohiggins on Twitter.
- I am building a Hosted Continuous Delivery Platform (Github → Our Platform → Heroku/AWS/Nodejitsu/etc) for Node.JS.
- Would love to talk to anybody using Node.JS/Heroku/AWS/etc about how to make our product awesome
- http://beyondfog.com