NOSQL

INTRODUCTION WITH MONGODB AND RUBY

GEOFF LANE <GEOFF@ZORCHED.NET>

@GEOFFLANE

WHAT IS NOSQL?

- NON-RELATIONAL DATA STORAGE
- USUALLY SCHEMA-FREE
- ACCESS DATA WITHOUT SQL (THUS... NOSQL)

WIDE-COLUMN / TABULAR

- GOOGLE BIGTABLE
- MNESIA (ERLANG)
- HBASE / HADOOP

KEY/VALUE STORES

- [ ] GOOGLE BIGTABLE
- [ ] MEMCACHED
- [ ] REDIS
- [ ] CASSANDRA
- [ ] AMAZON SIMPLEDB
- [ ] MS AZURE DB

DOCUMENT STORE

- ☐ COUCHDB
- ☐ MONGODB
- ☐ APACHE JACKRABBIT

OBJECT DATABASES

- ☐ CACHÉ
- ☐ ZOPE
- ☐ DB4O

WHY NOT NOSQL?

YAGNI
KISS
WHATEVER...

IT'S NOT EITHER/OR...
IT'S BOTH

ACID

- ☐ ATOMICITY (ALL OR NOTHING)
- ☐ CONSISTENCY (ENSURES THAT DB GOES FROM ONE CONSISTENT STATE TO ANOTHER)
- ☐ ISOLATION (TRANSACTION ISOLATION - OTHER TRANSACTIONS DON'T SEE CHANGED DATA)
- ☐ DURABILITY (RECOVER FROM FAILURES)

SQL

- [ ] SQL WORKS
- [ ] IT'S WELL KNOWN
- [ ] A LOT OF FREAKING SMART PEOPLE BUILT AND OPTIMIZED ORACLE, POSTGRESQL, MYSQL, SQL SERVER, ETC

SQL

```
SELECT id,entityId,outcomeType,type,name,shortDesc,longDesc,strategicFramework,enabled
FROM OutcomeTemplates
WHERE o.id IN (
    SELECT atm.moduleId as id FROM ApplicationTemplateModules atm
        WHERE atm.applicationTemplateId=?
)
OR o.id IN (
    SELECT m.moduleId as id FROM EntityModules m
        INNER JOIN Workset w
            ON m.entityId = w.entityId
        INNER JOIN ApplicationTemplate t
            ON w.Id = t.worksetId AND t.Id=?
)
AND o.enabled='Y'

```

MULTI-OBJECT TRANSACTIONS

- EVERY NOW AND THEN YOU NEED TO UPDATE MORE THAN ONE OBJECT AT A TIME RIGHT?

POORLY SUITED FOR SPECIFIC DOMAINS

- THINGS REQUIRING MULTI-OBJECT TRANSACTIONS
  - BANKING
  - ACCOUNTING
- TRADITIONAL BI
  - NIGHTLY-BATCH
  - OLAP

WHY NOSQL?

IT'S WELL SUITED FOR CERTAIN PROBLEMS

- ☐ CONTENT MANAGEMENT
- ☐ DOCUMENT MANAGEMENT
- ☐ LOG STORAGE
- ☐ UNSTRUCTURED DATA
- ☐ CACHING
- ☐ REAL-TIME ANALYTICS

HIGH-VOLUME / SCALABILITY

- ☐ WRITE INTENSIVE APPLICATIONS
- ☐ FIRE AND FORGET WRITES
- ☐ LOOSELY CONSISTENT OK
- ☐ EASY TO REPLICATE AND SHARD (MORE ON THAT LATER)


mongoDB

WHY MONGODB?

- I HAD TO CHOOSE ONE TO START WITH...
  - IT WAS EASY TO INSTALL (IT'S IN PORT AND BREW)
  - IT SEEMS PRETTY POPULAR
  - DOCUMENT-BASED SEEMS MORE GENERALLY USEFUL TO ME (MORE ON THIS...)

MONGODB SPECIFICS

- [ ] DOCUMENT STORE
- [ ] BSON (BINARY-JSON) INTERNAL DOCUMENT FORMAT
- [ ] NATIVE DRIVERS FOR LOTS OF LANGUAGES
- [ ] WRITTEN IN C++
- [ ] USES GOOGLE V8 JAVASCRIPT ENGINE

  (FOR INTERNAL JAVASCRIPT EXECUTION, MAP-REDUCE, AND SHELL)

MONGODB LIMITS

- TRANSACTIONS LIMITED TO A SINGLE DOCUMENT
- 2GB DB SIZE ON 32BIT OS

  (VIRTUALLY UNLIMITED ON 64BIT OS)
- 4MB DOCUMENT SIZE LIMIT

  (WITH GRIDFS WORKAROUND)
- NON INDEXED SORTS HAVE TO FIT IN MEMORY

DATA TYPES

- JSON DATA TYPES: STRING, INTEGER, BOOLEAN, DOUBLE, NULL, ARRAY, AND OBJECT.
- ADDITIONAL DATA TYPES: DATE, OBJECT ID, BINARY DATA, REGULAR EXPRESSION, AND CODE

BASIC CONCEPTS

- ☐ MULTIPLE DATABASES PER SERVER
- ☐ COLLECTIONS
- ☐ DOCUMENTS
  - ☐ EMBEDDED
  - ☐ REFERENCED
- ☐ INDEXES
- ☐ SERVER SIDE JAVASCRIPT FUNCTIONS

  (STORED PROCEDURE-ESQE)

BASIC OPERATIONS

MONGODB WITH RUBY

GET STARTED WITH GEMS

- [ ] GEM INSTALL MONGO

  (PULLS BSON GEM AS WELL)

- [ ] GEM INSTALL BSON_EXT

RUBY DRIVER

```ruby
require 'rubygems'  # not necessary for Ruby 1.9
require 'mongo'

db = Mongo::Connection.new.db("mydb")
coll = db.collection("testCollection")
100.times { |i| coll.insert("i" => i) }
puts coll.count()
coll.find().each { |row| puts row.inspect }
coll.find("i" => 71).each { |row| puts row.inspect }
coll.find("i" => {"$gt" => 50}).each { |row| puts row }
coll.find({"name" => /^a/})


search_string = "123"
coll.find({"name" => /#{search_string}/})

```

MONGODB WITH RAILS

ADVANCED FEATURES

MAP REDUCE

HTTP://WWW.ZORCHED.NET/2010/10/04/MONGODB-MAPREDUCE-FUNCTIONS-FOR-GROUPING/

JAVASCRIPT CONSOLE MAP REDUCE

```
 1 var fmap = function () {
 2     emit(this.metro.city, 1);
 3 }
 4 var fred = function (k, vals) {
 5     var sum = 0;
 6     for (var i in vals) {
 7         sum += vals[i];
 8     }
 9     return sum;
10 }
11 res = db.factories.mapReduce(fmap, fred)
12 db[res.result].find()
13 db[res.result].drop()
14
```

RUBY MAP REDUCE

```ruby
require 'rubygems'
require 'mongo'
include Mongo

db   = Connection.new.db('sample-db')
coll = db.collection('factories')

coll.remove

coll.insert( { :name => "Miller",    :metro => { :city => "Milwaukee", :state => "WI" } } )
coll.insert( { :name => "Lakefront", :metro => { :city: "Milwaukee", :state => "WI" } } )
coll.insert( { :name => "Point",     :metro => { :city => "Steven's Point", :state => "WI" } } )
coll.insert( { :name => "Pabst",     :metro => { :city => "Milwaukee", :state => "WI" } } )
coll.insert( { :name => "Blatz",     :metro => { :city => "Milwaukee", :state => "WI" } } )
coll.insert( { :name => "Coors",     :metro => { :city => "Golden Springs", :state => "CO" } } )

puts "There are #{coll.count()} factories. Here they are:"
coll.find().each { |doc| puts doc.inspect }
map_function = "function () { emit(this.metro.city, this.name); }"
reduce_function = "function (k, vals) { return vals.join(","); }"
coll.map_reduce(map_function, reduce_function).each { |r| puts r.inspect }


```

REPLICATION

SHARDING (HORIZONTAL PARTIONING)

WHAT DID I SKIP?

- LOT'S OF THINGS...
  - INDEXING
  - GRIDFS
  - SERVER-SIDE FUNCTIONS
  - SYSTEM MANAGEMENT STUFF (BACKUPS, ETC)