MongoDB workshop

DevConf12 - Moscow, Russian Federation
Jun 10th, 2012

Derick Rethans - derick@10gen.com - twitter: @derickr

About Me

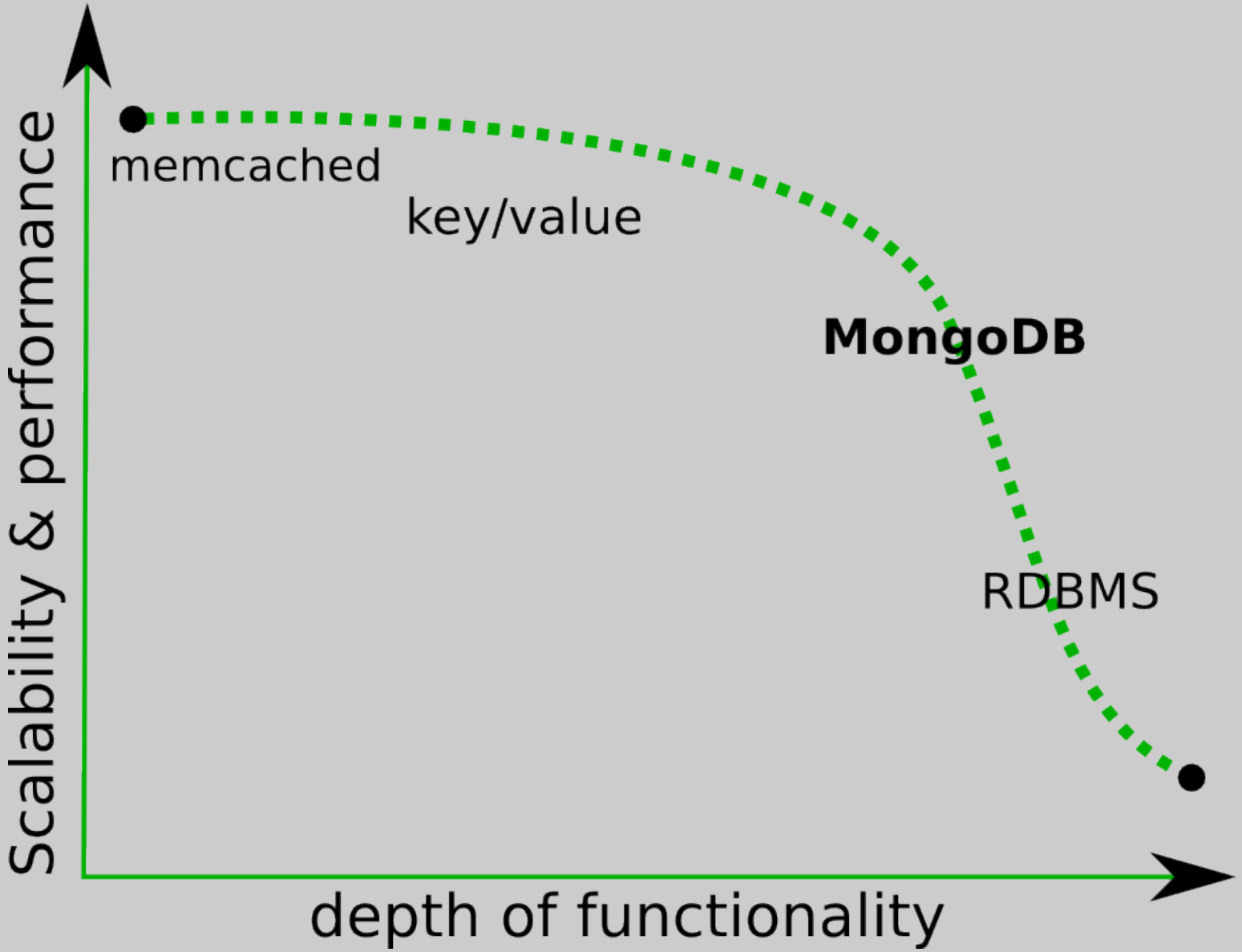
Derick Rethans

- Dutchman living in London
- PHP mongoDB driver maintainer for 10gen (the company behind mongoDB)
- Author of Xdebug
- Author of the mcrypt, input_filter, dbus, translit and date/time extensions

10gen

mongoDB

Database landscape



Terminology

- Document: the data (row)
- Collection: contains documents (table, view)
- Index
- Embedded Document (~join)

Documents

- Stored as BSON (Binary JSON)
- Can have embedded documents
- Have a unique ID (the _id field)
- Are schemaless

Simple document:

```
{
   "_id" : ObjectId("4cb4ab6d7addf98506010001"),
   "handle" : "derickr",
   "name" : "Derick Rethans"
}
```

Document with embedded documents:

```
{
  "_id" : "derickr",
  "name" : "Derick Rethans",
  "talks" : [
    { "title" : "Profiling PHP Applications",
      "url" : "http://derickrethans.nl/talks/profiling-phptour.pdf",
    },
    { "title" : "Xdebug",
      "url" : "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    }
  ]
}
```

The MongoDB PHP extension

- Maintained and supported by 10gen
- Can be installed with pecl: pecl install mongo
- Add extension=mongo.so to php.ini

Connecting to MongoDB

No tables or collections have to do be explicitly created:

```
<?php
$m = new Mongo();
$database = $m->demo;
$collection = $database->testCollection;
?>
```

Different connection strings:

- new Mongo("mongodb://localhost");
- new Mongo("mongodb://localhost:29000");
- new Mongo("mongodb://mongo.example.com");

Inserting a document

```
{
  "_id" : "derickr",
  "name" : "Derick Rethans",
  "talks" : [
    {
      "title" : "Profiling PHP Applications",
      "url" : "http://derickrethans.nl/talks/profiling-phptour.pdf",
    },
    {
      "title" : "Xdebug",
      "url" : "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    }
  ]
}
<?php
$document = array(
  "_id" => "derickr",
  "name" => "Derick Rethans",
  "talks" => array(
    array(
      "title" => "Profiling PHP Applications",
      "url" => "http://derickrethans.nl/talks/profiling-phptour.pdf",
    ),
    array(
      "title" => "Xdebug",
      "url" => "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    )
  )
);

$m = new Mongo();
var_dump( $m->demo->talks->insert( $document ) );
?>
```

mongoDB supports many types

- null
- boolean
- integer (both 32-bit and 64-bit, MongoInt32, MongoInt64)
- double
- string (UTF-8)
- array
- associative array
- MongoRegex
- MongoId
- MongoDate
- MongoCode
- MongoBinData

IDs

- Every document needs a unique ID

```
<?php
$m = new Mongo();
$c = $m->demo->articles;

$c->insert( [ 'name' => 'Derick', '_id' => 'derickr' ] );

$d = array( 'name' => 'Derick' );
$c->insert( $d );
var_dump( $d );
?>
```

Checking for errors

So far, we have not checked for whether inserts worked.

```
<?php
$m = new Mongo;
$c = $m->demo->talks;

$c->insert( array( '_id' => 'derickr' ) );
$c->insert( array( '_id' => 'derickr' ) );
?>
```

"Safe" inserts:

```
<?php
$m = new Mongo;
$c = $m->demo->talks;

try {
    $c->insert(
        array( '_id' => 'derickr' ), // document
        array( 'safe' => true )      // options
    );
} catch ( Exception $e ) {
    echo $e->getMessage(), "\n";
}
?>
```

Checking for errors (2)

Safeness levels:

- Confirm change recorded in memory: array('safe' => true);
- Confirm inclusion in journal: array('j' => true);
- Confirm committed to disk: array('fsync' => true);
- Confirm replication: array('safe' => true, 'w' => 2);

```
<?php
$m = new Mongo();
$c = $m->demo->talks;

$c->insert(
    array( '_id' => 'mongodb' ),       // document
    array( 'safe' => true, 'w' => 2 ) // options
);
?>
```

Querying (1)

```
<?php
$m = new Mongo();
// First "found" item:
$record = $m->demo->talks->findOne();

// Search on the _id field being 'derickr'
$record = $m->demo->talks->findOne( array( '_id' => 'derickr' ) );

// Search on the array element key 'title' in the 'talks' field
$record = $m->demo->talks->findOne(
  array(
    'talks.title' => 'Xdebug'
  )
);
?>
```

Querying (2)

```
<?php
$m = new Mongo();

// Find with 'projection'
$record = $m->demo->talks->findOne(
  array( 'talks.title' => 'Xdebug' ),
  array( 'name' => true, 'talks.url' => true )
);
?>
```

Querying and looping

Finding multiple documents is done with the find(), and returns an iterator in the form of a MongoCursor object.

```
<?php
$m = new Mongo();
$c = $m->demo->talks;

$cursor = $c->find( [ 'talks.title' => [ '$exists' => true ] ] );

foreach ( $cursor as $r )
{
    echo $r['_id'], "\n";
}
?>
```

Cursor methods

```
<?php
$m = new Mongo();
$c = $m->demo->articles;

$cursor = $c->find();
$cursor->sort( array( '_id' => 1 ) )
       ->limit( 3 )
       ->skip( 3 );

// Only now does the query get send
foreach ( $cursor as $r )
{
    echo $r['_id'], "\n";
}
?>
```

Modifying documents

```
<?php
$m = new Mongo;
$c = $m->demo->elephpants;
$c->remove();

$c->insert( [
  '_id' => 'e43',
  'name' => 'Dumbo'
] );

$c->update(
  [ 'name' => 'Dumbo' ], // criteria
  [ '$set' => array [ 'age' => '17' ] ] // modifiers
);

// document is now:
[
  '_id' => 'e43',
  'name' => 'Dumbo',
  'age' => '17',
]
?>
```

Updating documents

Update only updates the first document it finds by default.
You can set an option to get all matching documents to be updated

```
<?php
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();

$c->insert( [ '_id' => 'e42', 'name' => 'Kamubpo', 'age' => 17 ] );
$c->insert( [ '_id' => 'e43', 'name' => 'Denali',  'age' => 17 ] );

$c->update(
    [ 'age' => 17 ],                // criteria
    [ '$inc' => [ 'age' => 1 ] ],   // update spec
    [ 'multiple' => true ]          // options: multiple
);
?>
```

Indexes

```
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();

$c->insert( [ '_id' => 'ele1', 'name' => 'Jumbo' ] );
$c->insert( [ '_id' => 'ele2', 'name' => 'Tantor' ] );
$c->insert( [ '_id' => 'ele3', 'name' => 'Stampy' ] );

var_dump( $c->find( [ 'name' => 'Jumbo' ] )->explain() );
?>
```

Indexes

```
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();

$c->ensureIndex( [ 'name' => 1 ] );

$c->insert( [ '_id' => 'ele1', 'name' => 'Jumbo' ] );
$c->insert( [ '_id' => 'ele2', 'name' => 'Tantor' ] );
$c->insert( [ '_id' => 'ele3', 'name' => 'Stampy' ] );

var_dump( $c->find( [ 'name' => 'Jumbo' ] )->explain() );
?>
```

More about indexes

Compound indexes:

```
$myCol->ensureIndex( [ _id => 1, ts => -1 ] )
```

Indexes can be made on nested fields:

```
$myCol->ensureIndex( [ "article.title" => -1 ] )
```

Indexes can be made on array values:

```
$myCol->insert( [ '_id' => 42, 'values' => [ 'fourty', 'two' ] ] );
$myCol->ensureIndex( [ 'values' => 1 ] );
```

Searching with regexp: ^:

```
$myCol->find( [ 'name' => new MongoRegex( '/^tan/i' ) ] )
```

Commands

- Are run on the database
- Do not return a cursor
- Return one document
- Some commands have helpers in the drivers and shell

Distinct

```
<?php
$m = new Mongo; $c = $m->demo->pubs; $c->drop();

$c->insert( [ 'name' => 'Betsy Smith',   'city' => 'London' ] );
$c->insert( [ 'name' => 'London Tavern', 'city' => 'London' ] );
$c->insert( [ 'name' => 'Lammars',       'city' => 'Manchester' ] );
$c->insert( [ 'name' => 'Weatherspoons', 'city' => 'Coventry' ] );

$r = $m->demo->command( [
    'distinct' => 'pubs',
    'key' => 'city',
    'query' => [ 'name' => [ '$ne' => 'Weatherspoons' ] ]
] );
var_dump( $r['values'] );
?>
```

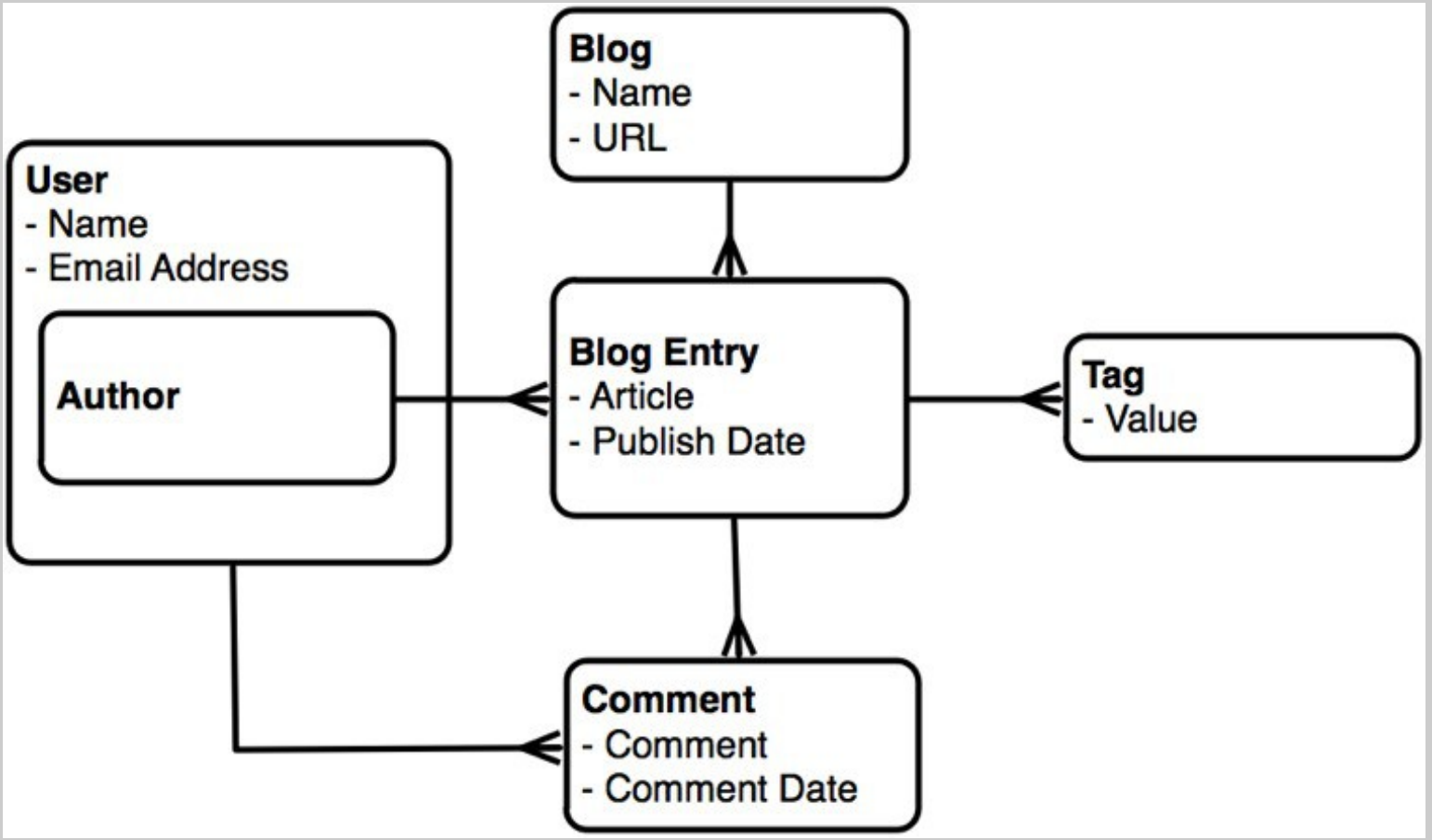
RDBMS: Normalisation

- 1970 E.F.Codd introduces 1st Normal Form (1NF)
- 1971 E.F.Codd introduces 2nd and 3rd Normal Form (2NF, 3NF)
- 1974 Codd & Boyce define Boyce/Codd Normal Form (BCNF)
- 2002 Date, Darween, Lorentzos define 6th Normal Form (6NF)

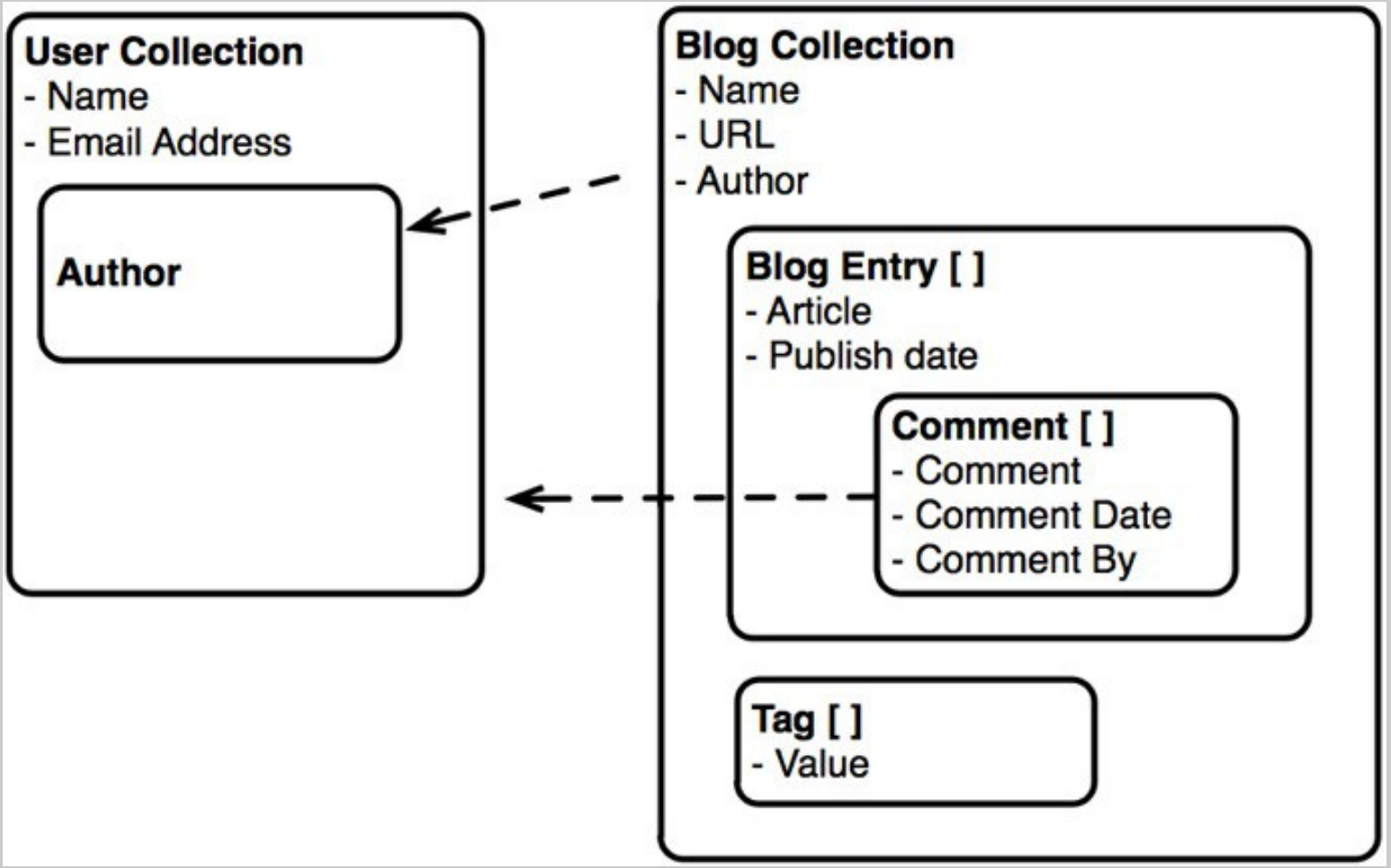
Goals:

- Avoid anomalies when inserting, updating or deleting
- Minimize redesign when extending the schema
- Make the model informative to users
- Avoid bias towards a particular style of query

Blog in a RDBMS



Blog in MongoDB



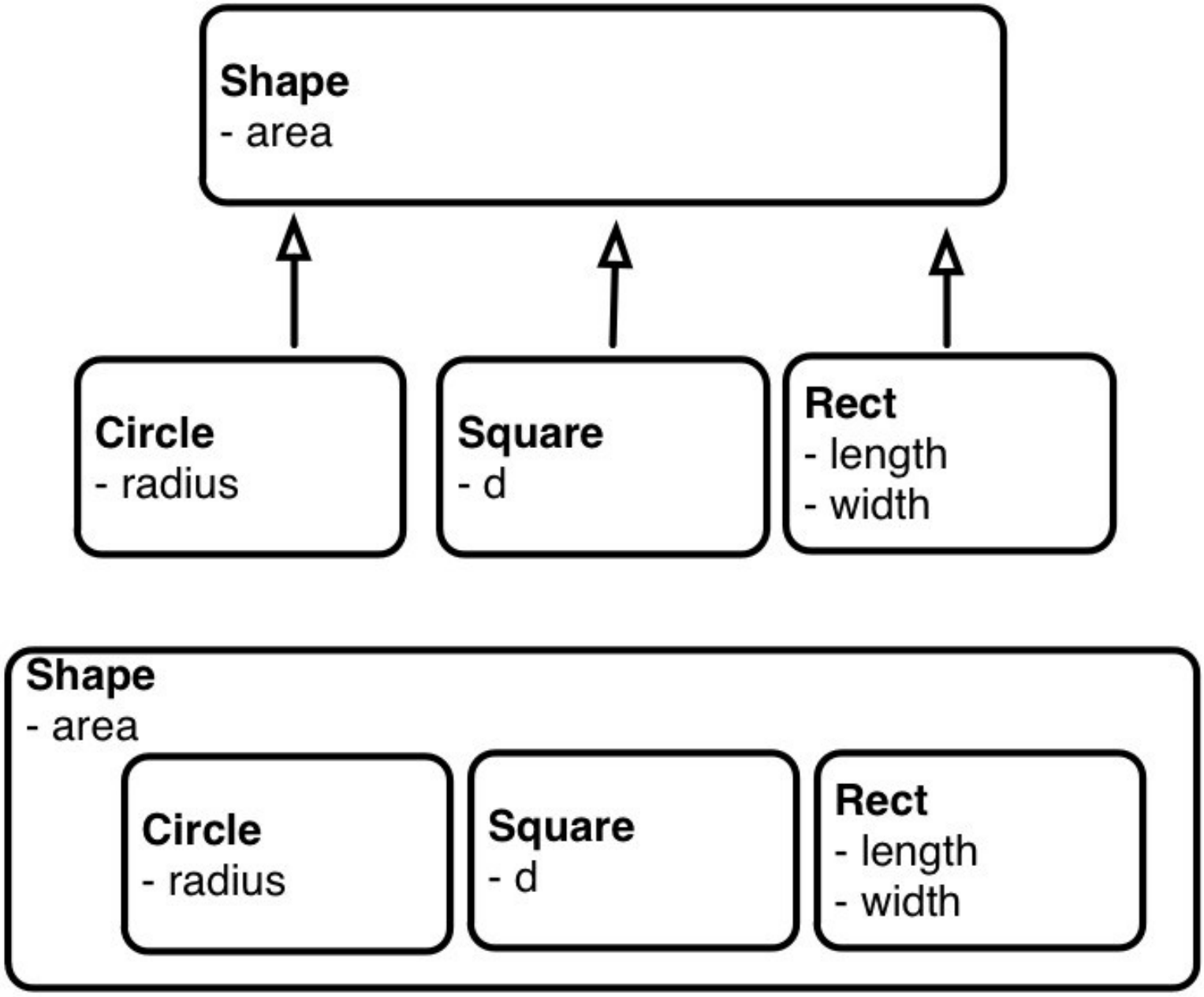
Schema considerations

- Access Patterns?
- Read / Write Ratio
- Types of updates
- Types of queries
- Data life-cycle

Considerations

- No Joins
- Document writes are atomic

Inheritance



Single table inheritance - MongoDB

```
{ _id: "1", type: "circle", area: 3.14, radius: 1}
{ _id: "2", type: "square", area: 4, d: 2}
{ _id: "3", type: "rectangle", area: 10, length: 5, width: 2}
```

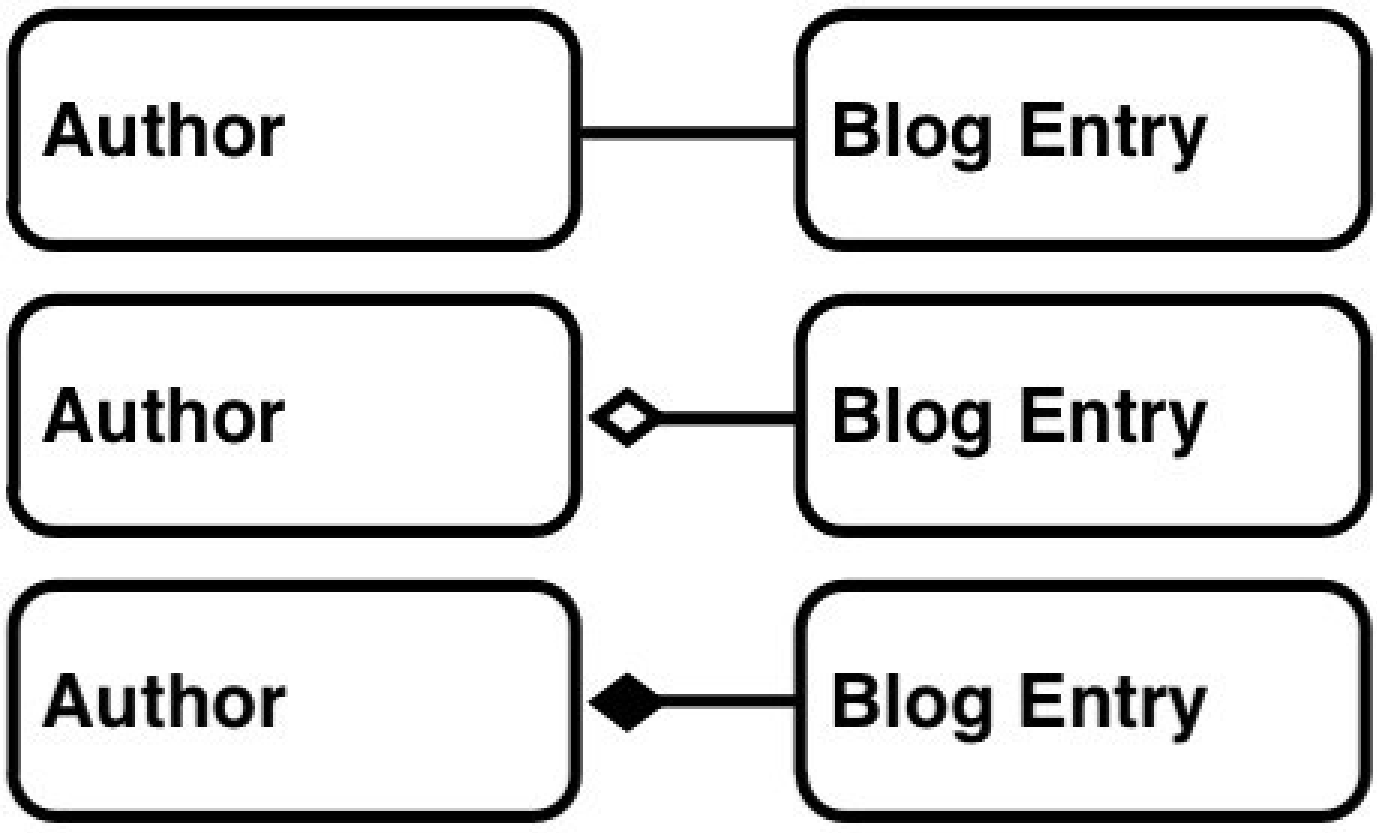
Inheritance

- Simple to query across sub-types
- Indexes on specialized values will be small

One to Many (1:n)

One to Many relationships can specify:

- degree of association between objects
- containment
- life-cycle



One to Many (1:n)

Embedded Array / Array Keys

- some queries harder: e.g find all areas speaking German

```
{
  name: "Germany",
  language: "German",
  areas: [
    { name: "Bavaria", language: "Swabian" }
  ]
},
{
  name: "Switzerland",
  areas: [
    { name: "Deutsche Schweiz",  language: "German"  },
    { name: "Romandy",           language: "French"  },
    { name: "Svizzera italiana", language: "Italian" },
  ]
}
```

One to Many (1:n)

Embedded Tree

- single document
- natural
- hard to query

```
blogs: {
  author: "sambeau",
  date: "84 days ago",
  comments: [
    {
      author: "ElliotH",
      date: "84 days ago",
      text: "It's also missing the necessary credit to OpenStreetMap's
          contributors; we look forward to working with Apple to get that on
          there.",
      replies: [
        {
          author: "sambeau",
          date: "83 days ago",
          text: "I do think that this is a very classy move by the OSM people,
              no ranting blog post or 'Apple stole our stuff', welcoming people
              presents a much better image of the project."
        }
      ]
    }
  ]
}
```

One to Many (1:n)

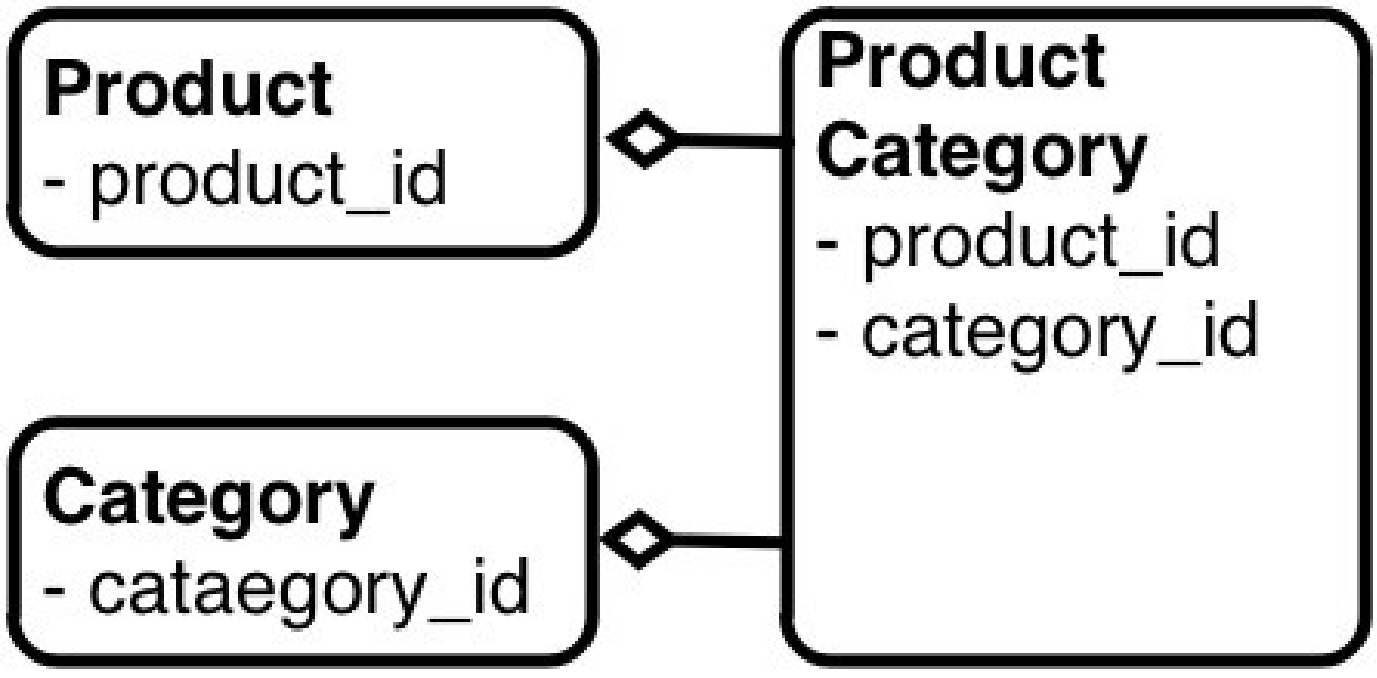
Split out into multiple documents

- most flexble
- more queries

```
{ name: "Germany", language: "German",                    },
{ name: "Bavaria", language: "Swabian", part_of: "Germany" },

// or:
{
  name: "Switzerland",
  parts: [ 'Deutsche Schweiz', 'Romandy', 'Svizzera italiana' ]
},
{ name: "Deutsche Schweiz",  language: "German" },
{ name: "Romandy",          language: "French" },
{ name: "Svizzera italiana", language: "Italian" },
```

Many to Many (n:m)

- products can be in many categories
- category can have many products



Many to Many (n:m)

```
products:
{ _id: 10, name: "Blue elephpant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephpant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys",            product_ids: [ 10, 11 ] }
{ _id: 8, name: "everything pink", product_ids: [ 11 ] }
```

All categories for a given product (pink elephpant):

```
db.categories.find( { product_ids: 11 } );
```

All products for a given category (toys):

```
db.products.find( { category_ids: 4 } );
```

Updates need to be done in two collections

Many to Many (n:m) - alternative 1

```
products:
{ _id: 10, name: "Blue elephpant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephpant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys"           }
{ _id: 8, name: "everything pink"}
```

All categories for a given product (pink elephpant):

```
product = db.products.find( { category_ids: 4 } );
db.categories.find( { _id: { $in: product.category_ids } } );
```

All products for a given category (toys):

```
db.products.find( { category_ids: 4 } );
```

Many to Many (n:m) - alternative 2

```
products:
{ _id: 10, name: "Blue elephpant" }
{ _id: 11, name: "Pink elephpant" }

categories:
{ _id: 4, name: "toys",            product_ids: [ 10, 11 ] }
{ _id: 8, name: "everything pink", product_ids: [ 11 ] }
```

All categories for a given product (pink elephpant):

```
db.categories.find( { product_ids: 11 } );
```

All products for a given category (toys):

```
category = db.categories.find( { category_ids: 4 } );
db.products.find( { _id: { $in: category.product_ids } } );
```

Embedding versus Linking

Embedding

- Simple data structure
- Limited to 16MB
- Larger documents
- How often do you update?
- Will the document grow and grow?

Linking

- More complex data structure
- Unlimited data size
- More, smaller documents
- What are the maintenance needs?

OpenStreetMap

- "Wikipedia for Map Data"
- Licensed under the Creative Commons Attribution-ShareAlike 2.0 licence (CC-BY-SA): Licensed under the Open Database License (ODbL 1.0): You are free to copy, distribute, transmit and adapt our maps and data, as long as you credit OpenStreetMap and its contributors. If you alter or build upon our maps or data, you may distribute the result only under the same licence.
- Rendered map:
- A lot of data is not rendered, but is available.

What is OpenStreetMap?

- A database with geographic data of the whole planet under a free license
- APIs to add, retrieve and query map data
- A social platform for improving the map

The Data

- Nodes (Lat/Lon point) <node id='459517295' lat='50.0100766' lon='8.3162402' user='WoGo' timestamp='2009-08-09T11:45:33Z' uid='152395' version='1' changeset='2083951'>
- Ways (Ordered interconnection of nodes)
- Areas (Closed ways) <way id='76174399' user='Derick Rethans' uid='37137' timestamp='2010-09-06T08:30:14Z' version='1' changeset='5695697'> <nd ref='898861293'/> <nd ref='898861305'/> <nd ref='898861298'/> <nd ref='898861315'/> <nd ref='898861293'/> ... </way>
- Tags (Describe an element) <tag k='addr:housenumber' v='375'/> <tag k='addr:street' v='Kilburn High Road'/> <tag

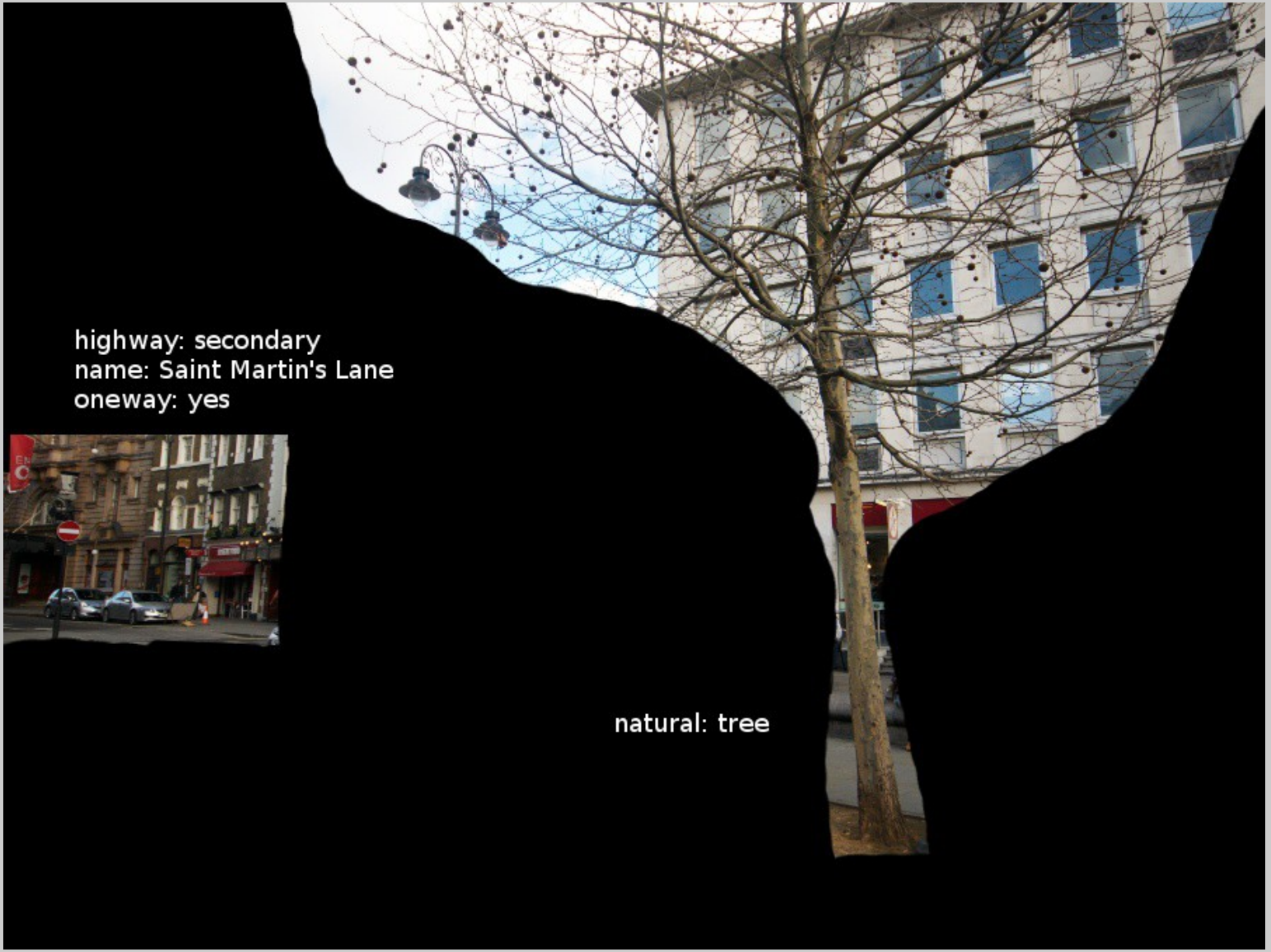
Tagging features (1)

Tagging features (2)

Tagging features (3)

amenity: cafe
name: Pret A Manger
website: http://www.pret.com

Tagging features (4)



Using OpenStreetMap data

Process:

- Use XAPI, API or extracts to fetch data
- Parse XML file with PHP into a DB (MongoDB)
- Query database
- Show data
- Profit (or in our case, we found a restaurant)!

Fetching OSM data

```
wget http://download.geofabrik.de/osm/europe/great_britain/england.osm.bz2
```

(Or pick a smaller country from http://download.geofabrik.de/osm/europe/)

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="Osmosis 0.39">
  <bound box="51.26000,-0.56300,51.68000,0.28000" origin="http://www.openstreetmap.org/api/0.6"/>
  <node id="44" version="3" timestamp="2011-09-08T20:53:43Z" uid="15867" user="mattfromderby" changeset="9249514"
lat="51.5250157" lon="-0.1485302"/>
  <node id="47" version="2" timestamp="2011-09-08T20:53:43Z" uid="15867" user="mattfromderby" changeset="9249514"
lat="51.525182" lon="-0.1479761"/>
  <node id="52" version="3" timestamp="2011-09-08T21:13:19Z" uid="15867" user="mattfromderby" changeset="9249514"
lat="51.5289059" lon="-0.1458559"/>
...
  <node id="1571982070" version="1" timestamp="2011-12-31T20:37:18Z" uid="545369" user="bigfatfrog67" changeset="10257991"
lat="51.5087759" lon="-0.1253258">
    <tag k="name" v="Charing Cross"/>
    <tag k="railway" v="subway_entrance"/>
    <tag k="source:name" v="survey"/>
    <tag k="source:railway" v="survey"/>
  </node>
...
  <way id="74" version="5" timestamp="2010-12-12T22:43:15Z" uid="508" user="Welshie" changeset="6643030">
    <nd ref="196101"/>
    <nd ref="196100"/>
    <nd ref="196331"/>
    <tag k="abutters" v="retail"/>
    <tag k="highway" v="primary"/>
    <tag k="maxspeed" v="30 mph"/>
    <tag k="name" v="Ballards Lane"/>
    <tag k="ref" v="A598"/>
  </way>
...
  <relation id="69" version="13" timestamp="2009-02-09T14:57:24Z" uid="45027" user="PA94" changeset="307506">
    <member type="way" ref="3793834" role="outer"/>
    <member type="way" ref="26454599" role="inner"/>
    <tag k="created_by" v="xybot"/>
    <tag k="type" v="multipolygon"/>
  </relation>
...
</osm>
```

Exercise 1: Connecting and creating indexes

- wget http://derickrethans.nl/files/dump/mongo-exercise1.php.txt
- Open it in your favourite editor and edit.

```
<?php
$file = $argv[1];

// 1 ADD: Set variable $m to the connection, and $d to the collection "poi" in
// the database "demo" on localhost
$m = new Mongo();
$d = $m->demo->poi;

// 2 ADD: Drop the collection
$d->drop();

// 3 ADD: Create an index on "name" and a "2d" index on "loc"
$d->ensureIndex( array( 'loc' => '2d' ) );
$d->ensureIndex( array( 'name' => 1 ) );

var_dump($d->getIndexInfo());
```

- Run the script!

Importing OSM into MongoDB

From http://www.openstreetmap.org/browse/way/4442243:

```
Way:    Brondesbury Road (4442243)
Tags:   highway = secondary
        name = Brondesbury Road
        ref = B451
        source:ref = OS OpenData StreetView
```

Simplest way:

```
{
    '_id': 'w4442243',
    'highway': 'secondary',
    'name' : 'Brondesbury Road',
    'ref' : 'B451',
    'source_ref' : 'OS OpenData StreetView',
}
```

or a little more organised:

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
```

Importing OSM into MongoDB (2)

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
```

Querying all secondary roads:

```
db.poi.find( { 'tags.highway' : 'secondary' } );
```

Find all roads:

```
db.poi.find( { 'tags.highway' : { $exists : true } );
```

Importing OSM into MongoDB (3)

Find all pubs:

```
db.poi.find( { 'tags.amenity' : 'pub' } );
```

Let's look at the "explain" output:

```
db.poi.find( { 'tags.amenity' : 'pub' } ).explain();
{
    "cursor" : "BasicCursor",
    "isMultiKey" : false,
*|e70000|    "n" : 3895,|*
*|e70000|    "nscannedObjects" : 2347913,|*
    "nscanned" : 2347913,
    ...
```

We need an index for that:

```
db.poi.ensureIndex( { "tags.amenity" : 1 } );
```

And the new "explain" output:

```
db.poi.find( { 'tags.amenity' : 'pub' } ).explain();
{
    "cursor" : "BtreeCursor tags.amenity_1",
    "isMultiKey" : false,
    "n" : 3895,
*|00e700|    "nscannedObjects" : 3895,|*
    "nscanned" : 3895,
    ...
```

Importing OSM into MongoDB (4)

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
db.poi.ensureIndex( { "tags.amenity" : 1 } );
db.poi.ensureIndex( { "tags.highway" : 1 } );
db.poi.ensureIndex( { "tags.ref" : 1 } );
db.poi.ensureIndex( { "tags.???" : 1 } );
```

- Lots of indexes make things slow
- Limited to 64 indexes

Think about the data schema!

```
$doc = array(
    'name' => 'A440',
    'tags = [
        'highway' => 'secondary',
        'oneway' => 'yes'
    ]
);
$db->poi->ensureIndex( [ 'name' => 1 ] );
$db->poi->ensureIndex( [ 'tags.highway' => 1 ] );
$db->poi->ensureIndex( [ 'tags.oneway' => 1 ] );
```

or:

```
$doc = array(
    'name' => 'A440',
    'tags' = [
        { 'k' => 'highway', 'v' => 'secondary' },
        { 'k' => 'oneway', 'v' => 'yes' }
    ]
);
$db->poi->ensureIndex( [ 'name' => 1 ] );
$db->poi->ensureIndex( [ 'tags.k' => 1, 'tags.v' => 1 ] );
```

Exercise 2: Add index for tags

- wget http://derickrethans.nl/files/dump/mongo-exercise2.php.txt
- Open it in your favourite editor and edit.

```
<?php
// 1 ADD: Indexes for the tags
$d->ensureIndex( array( 'tags_indexed' => 1 ) );
$d->ensureIndex( array( 'tags_indexed.k' => 1, 'tags_indexed.v' => 1 ) );

var_dump($d->getIndexInfo());
```

- Run the script!

Geospatial Indexes

Helps you with finding POIs in a 2D space

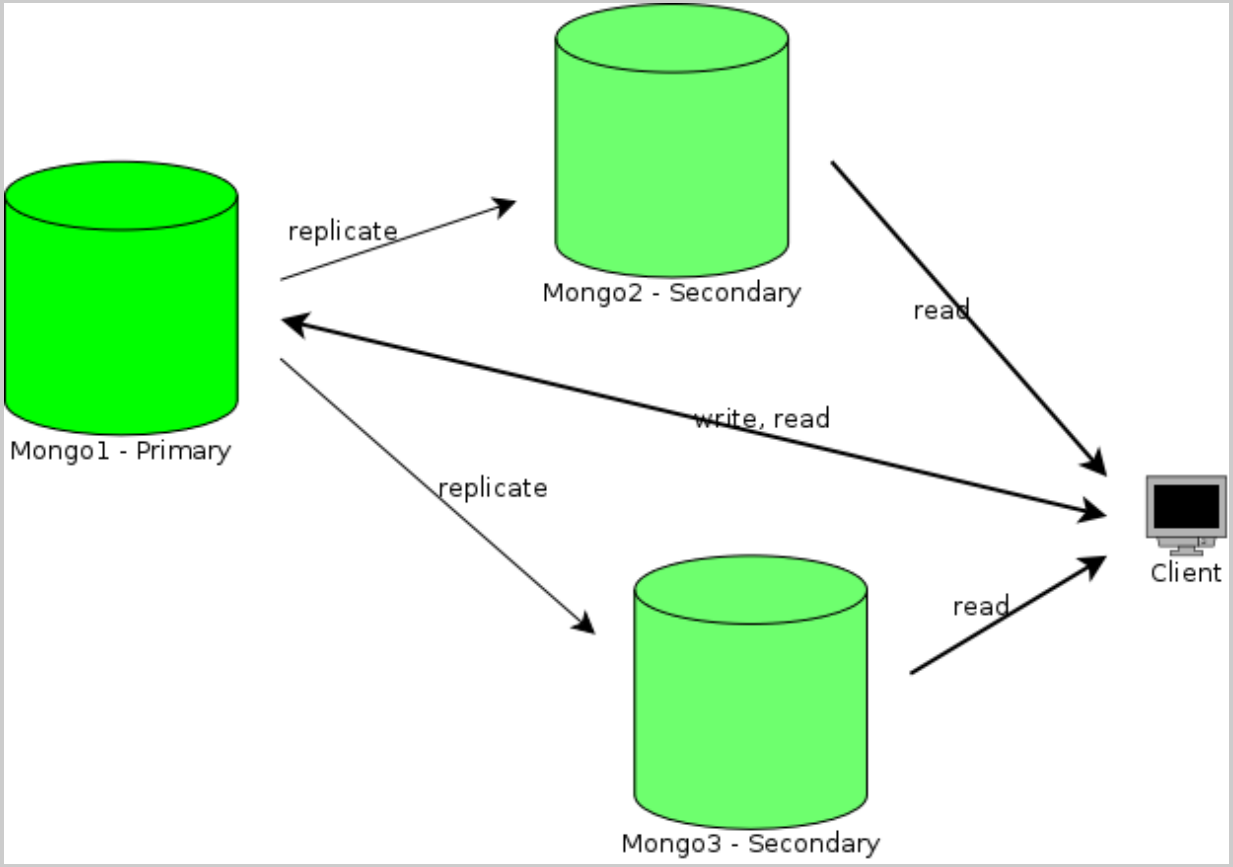
```
<?php
$m = new Mongo; $c = $m->demo->pubs; $c->drop();

$c->ensureIndex( array( 'l' => '2d' ) );
$c->insert([ 'name' => 'Betsy Smith',   'l' => [ -0.193, 51.537 ] ]);
$c->insert([ 'name' => 'London Tavern', 'l' => [ -0.202, 51.545 ] ]);

$closest = $m->demo->command( [
    'geoNear' => 'pubs',
    'near' => [ -0.198, 51.538 ],
    'spherical' => true,
] );

foreach ( $closest['results'] as $res ) {
  printf( "%s: %.2f km\n", $res['obj']['name'], $res['dis'] * 6378 );
}
?>
```

About 2D indexes

A 2d geo index wants two elements. With the first being the longitude (x) and the second one being the latitude (y) (as in GeoJSON).
The following are all equivalent:

```
$myCol->insert( [
    _id: 42, loc: [ 6.43, 52.1233 ]
] );
$myCol->insert( [
    _id: 42, loc: { long: 6.43, lat: 52.1233 }
] );
$myCol->insert( [
    _id: 42, loc: { latitude: 6.43, longitude: 52.1233 }
] );
```

Replication



- Failover/Availability
- Scaling reads
- Primaries, secondaries and arbiters
- Odd number to prevent split brain

Setting up replication

Starting the daemons (2 data, 1 arbiter):

```
mongod -f /etc/mongodb.conf --dbpath=~/repl-slave0 --port 13000 --replSet seta
mongod -f /etc/mongodb.conf --dbpath=~/repl-slave1 --port 13001 --replSet seta
mongod -f /etc/mongodb.conf --dbpath=~/repl-arb --port 13002 --replSet seta --rest
```

Sample /etc/mongodb.conf

```
logpath=/var/log/mongodb/mongodb.log
logappend=true
smallfiles=true
replSet=seta
```

Configure the set:

```
mongo localhost:13000

> db.adminCommand( {
  replSetInitiate: {
    _id: 'seta',
    members: [
      { _id: 0, host: 'localhost:13000' },
      { _id: 1, host: 'localhost:13001' },
    ]
  }
} );

PRIMARY> rs.addArb('localhost:13002');
```

Write concerns

Safeness levels:

- Confirm replication: array('safe' => true, 'w' => 2);

```
<?php
$m = new Mongo( 'mongodb://localhost:13000' );
$c = $m->demo->talks;

$c->insert(
    array( '_id' => 'mongodb' ),       // document
    array( 'safe' => true, 'w' => 2 ) // options
);
?>
```

Read preferences

(in development)

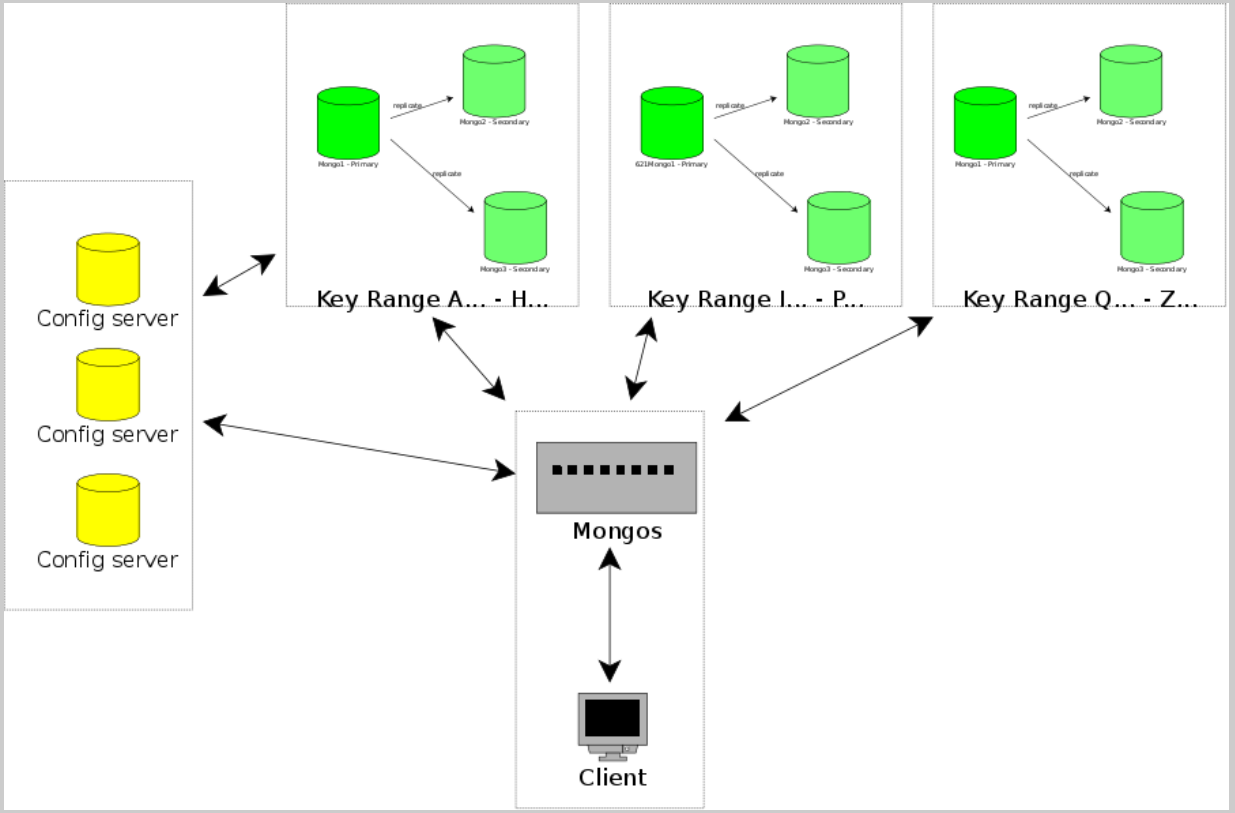
```
<?php
$m = new Mongo( 'mongodb://localhost:13000' );
$c = $m->demo->poi;
$c->setReadPreference( Mongo::READ_NEAREST, array( 'dc' => 'europe1' ) );

$c->find( ... );

// or
$c->find( ... )->setReadPreference( Mongo::SECONDARY );
?>
```

Connection String

```
<?php
$options = array( 'replicaSet' => 'seta' );

$m = new Mongo( 'mongodb://localhost:13000', $options );

$m = new Mongo( 'mongodb://localhost:13000,localhost:13001', $options );

$m = new Mongo( 'mongodb://user:password@localhost:13000,localhost:13001/demo', $options );
?>
```

- Add more than one host for seeding
- Don't add the arbiters to the connection string
- Don't use ->authenticate()

Sharding



- Scaling writes and reads
- Config servers, router (mongos) and replica sets

Resources

- Slides: http://derickrethans.nl/talks/mongo-devconf12
- Contact me: Derick Rethans: @derickr, derick@10gen.com