# GETTING ACQUAINTED WITH

mongoDB

Jeremy Mikola
**@jmikola**

10gen | the MongoDB company

- Develops MongoDB
- Provides support, training, and consulting
- Loves the community
  - Mailing lists, IRC, and Stack Overflow
  - Sponsors conferences and user groups
- Offices: NYC, Palo Alto, Europe, Australia
- Hiring at **10gen.com/careers**

# STARTING OFF

- What sets MongoDB apart?
- What are documents?
- How do I get them into Mongo?
- How can I get them back out?
- Can we do that faster?
- What else can I do?

# TERMINOLOGY

| RDBMS | : | MongoDB |
|---|---|---|
| Database | : | Database |
| Table | : | Collection |
| Row | : | Document |
| Column | : | Field |
| Primary Key | : | `_id` |

# RDBMS: THE GOOD PARTS

- Tried and true
- SQL is a rich query language
- ACID compliance
- Transactions

# RDBMS: THE BAD PARTS

- Modeling complex or polymorphic data
- Schema migrations
- Administration
- Scalability is a trade-off
- Partition tolerance

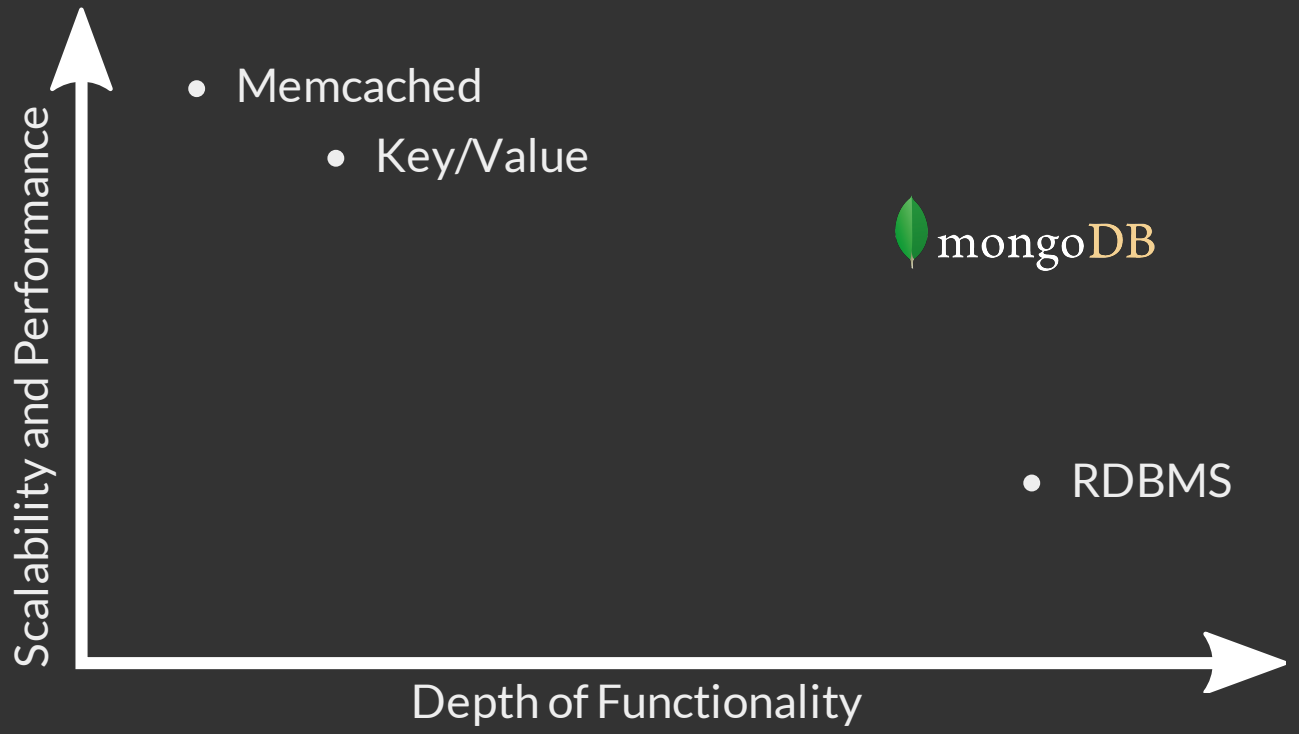
# MONGODB: THE GOOD PARTS

- Document model
- Dynamic schemas
- Scalability and performance
- Features (aggregation, geo, GridFS)

# MONGODB: THE BAD PARTS

- Limited atomicity
- Consistency is a trade-off
- Query language has its limits
- Concurrency (improving)

# DATABASE LANDSCAPE



# WHY MONGODB?

*“ MongoDB has the best features of key/value stores, document databases and relational databases in one.*
*— **John Nunemaker** ”*

# RELATIONAL MODELING

- Articles
  - One author
  - Many comments
  - Many tags
- Tags
  - Many articles
- Authors
  - Many articles
  - Many comments
- Comments
  - One article
  - One author

# RELATIONAL MODELING

| articles | | | |
|---|---|---|---|
| id | author_id | title | body |
| 1 | 2 | Praesent ante dui | Lorem ipsum… |

| articles_to_tags | | |
|---|---|---|
| id | article_id | tag_id |
| 36 | 1 | 7 |
| 37 | 1 | 8 |

| authors | | |
|---|---|---|
| id | name | email |
| 2 | Bob | bob@example.com |
| 3 | John | john@example.com |

| comments | | | |
|---|---|---|---|
| id | article_id | author_id | body |
| 4 | 1 | 3 | Morbi libero erat… |
| 5 | 1 | 2 | Dapibus quis… |
| 6 | 1 | 3 | Fusce fermentum… |

| tags | |
|---|---|
| id | name |
| 7 | luctus |
| 8 | rhoncus |

# THINGS MAY GET OUT OF HAND

# DOCUMENT MODELING

- Articles
  - One author
  - Many comments
    - One author
  - Many tags

# DOCUMENT MODELING

```
{
    _id: 1,
    title: "Praesent ante dui",
    body: "Lorem ipsum…",
    author: { name: "Bob", email: "bob@example.com" },
    comments: [
        {
            body: "Morbi libero erat…",
            author: { name: "John", email: "john@example.com" }
        },
        {
            body: "Dapibus quis…",
            author: { name: "Bob", email: "bob@example.com" }
        },
        {
            body: "Fusce fermentum…",
            author: { name: "John", email: "john@example.com" }
        }
    ],
    tags: [ "luctus", "rhoncus" ]
}
```

# DOCUMENTS ARE BSON

- Binary JSON
- Zero or more key/value pairs
- Values are scalars, arrays, and objects
- Additional data types
  - Binary strings
  - JavaScript Code
  - Dates

# BINARY REPRESENTATION

```
{"hello": "world"}

"\x16\x00\x00\x00\x02hello\x00
 \x06\x00\x00\x00world\x00\x00"
```

```
{"things": ["foo", 5.05, 2012]}

"\x33\x00\x00\x00\x04things\x00
 \x26\x00\x00\x00\x02\x30\x00\x04
 \x00\x00\x00foo\x00\x01\x31\x00
 \x33\x33\x33\x33\x33\x33\x14\x40
 \x12\x32\x00\xDC\x07\x00\x00\x00
 \x00\x00\x00\x00\x00"
```

Curious? **BSONSpec.org**

# DOCUMENTS IN PHP

```php
// Arrays are most common
$a = ['hello' => 'world'];

$b = ['things' => ['foo', 5.05, 2012]];
```

```php
// Objects work, too!
$a = new stdClass();
$a->hello = 'world';

$b = new stdClass();
$b->things = ['foo', 5.05, 2012];
```

# GETTING UP AND RUNNING WITH MONGO

## IN 60 SECONDS

* EXCLUDING DOWNLOAD TIME :)

# MONGODB.ORG/DOWNLOADS

- Compiled binaries
  - OS X, Linux, Windows, Solaris
- Packages
  - MacPorts, Homebrew, Debian, CentOS
- Drivers for over a dozen languages

# INSTALLING

```
$ tar xvzf mongodb-linux-x86_64-2.2.0-rc1.tgz
mongodb-linux-x86_64-2.2.0-rc1/GNU-AGPL-3.0
mongodb-linux-x86_64-2.2.0-rc1/README
mongodb-linux-x86_64-2.2.0-rc1/THIRD-PARTY-NOTICES
mongodb-linux-x86_64-2.2.0-rc1/bin/mongodump
mongodb-linux-x86_64-2.2.0-rc1/bin/mongorestore
mongodb-linux-x86_64-2.2.0-rc1/bin/mongoexport
mongodb-linux-x86_64-2.2.0-rc1/bin/mongoimport
mongodb-linux-x86_64-2.2.0-rc1/bin/mongostat
mongodb-linux-x86_64-2.2.0-rc1/bin/mongotop
mongodb-linux-x86_64-2.2.0-rc1/bin/mongooplog
mongodb-linux-x86_64-2.2.0-rc1/bin/mongofiles
mongodb-linux-x86_64-2.2.0-rc1/bin/bsondump
mongodb-linux-x86_64-2.2.0-rc1/bin/mongoperf
mongodb-linux-x86_64-2.2.0-rc1/bin/mongosniff
mongodb-linux-x86_64-2.2.0-rc1/bin/mongod
mongodb-linux-x86_64-2.2.0-rc1/bin/mongos
mongodb-linux-x86_64-2.2.0-rc1/bin/mongo
```

# STARTING

```
$ mkdir /data/db

$ mongodb-linux-x86_64-2.2.0-rc1/bin/mongod
[initandlisten] MongoDB starting : pid=28285 port=27017 dbpath=/data/db/ 64-bit
[initandlisten] db version v2.2.0-rc1, pdfile version 4.5
[initandlisten] git version: cf117b7c7d0655282deab662ed68e11119f844c7
[initandlisten] build info: Linux ip-10-2-29-40 2.6.21.7-2.ec2.v1.2.fc8xen #1 SM
[initandlisten] options: {}
[initandlisten] journal dir=/data/db/journal
[initandlisten] recover : no journal files present, no recovery needed
[initandlisten] waiting for connections on port 27017
[websvr] admin web console waiting for connections on port 28017
```

# CONNECTING

```
$ mongo
MongoDB shell version: 2.2.0-rc1
connecting to: test
> show dbs
local   (empty)

> db.foo.insert({x: 1})

> db.foo.find()
{ "_id" : ObjectId("50368f0cdea8fae83cf3d097"), "x" : 1 }

> !!"Does this shell use JavaScript?"
true
```

# PHP DRIVER

- **PECL extension**
- Classes for Mongo resources, types
- BSON encode/decode functions
- Persistent connections, logging
- **INI configuration**
- Track development on **GitHub** and **JIRA**

*" The single, best interface of any PHP extension — **Matthew Weier O'Phinney** "*

# PHP DRIVER INSTALLATION

```
$ pecl install mongo
downloading mongo-1.2.12.tgz ...
Starting to download mongo-1.2.12.tgz (90,517 bytes)
.....................done: 90,517 bytes
36 source files, building

...

Build process completed successfully
Installing '/usr/lib/php5/20100525/mongo.so'
install ok: channel://pecl.php.net/mongo-1.2.12
You should add "extension=mongo.so" to php.ini
```

- **Compiled binaries** for Windows
- Included with Zend Server

# LET'S GET COOKING

# CORE CLASSES

- Mongo
- MongoDB
- MongoCollection
- MongoCursor

# TYPE CLASSES

- MongoId
- MongoDate
- MongoBinData
- MongoInt64
- MongoMaxKey
- MongoCode
- MongoRegex
- MongoInt32
- MongoMinKey
- MongoTimestamp

# CONNECTING

```
// localhost:27017
$m = new Mongo();

// example.com:27018
$m = new Mongo('mongodb://example.com:27018');

// authentication
$m = new Mongo('mongodb://user:password@localhost');

// replica set
$m = new Mongo(
    'mongodb://rs1.example.com,rs2.example.com',
    ['replicaSet' => 'myReplSet']
);
```

# THE MONGO CLASS

```
$m = new Mongo();

// Get an array of databases and their sizes
$m->listDBs();

// Get a MongoDB for the "test" database
$m->test;
$m->selectDB('test');

// Get a MongoCollection for "test.users"
$m->test->users;
$m->selectCollection('test', 'users');

// Drop the "test" database
$m->dropDB('test');
```

# THE MONGODB CLASS

```php
$db = $m->test;
$db = new MongoDB($m, 'test');

$db->getCollectionNames(); // Array of collection names
$db->listCollections();    // Array of MongoCollection instances

// Get a MongoCollection for "test.users"
$db->users;
$db->selectCollection('users');

// Create a capped collection limited to 10KiB and 10 documents
$db->createCollection('logs', true, 10240, 10)

$db->dropCollection('users'); // Drop "test.users"
$db->drop();                  // Drop the entire database
```

# THE MONGOCOLLECTION CLASS

```
$c = $db->users;
$c = new MongoCollection($db, 'users');

// Insert a user document for Bob
$c->insert(['username' => 'bob', 'roles' => ['admin']]);

// Retrieve Bob's user
$c->findOne(['username' => 'bob']);

// Find all admins
$c->find(['roles' => 'admin']);
```

# THE MONGOCOLLECTION CLASS

```
$c->count();                      // Count all users
$c->count(['roles' => 'admin']); // Count only admins

$c->remove(['username' => 'john']); // Delete a user
$c->remove();                       // Delete all users

// Drop the entire collection
$c->drop();
```

# THE MONGOCOLLECTION CLASS

```php
// Create a user document for Tom
$user = ['username' => 'tom'];

$c->insert($user); // Insert Tom's document

var_dump($user);
```

```
array(2) {
  ["username"]=>
  string(3) "tom"
  ["_id"]=>
  object(MongoId)#3 (1) {
    ["$id"]=>
    string(24) "503b0772e84df1f87b000001"
  }
}
```

# IDENTIFIERS

*“ You have the right to provide an `_id`.*
*Any `_id` you provide must be unique for its collection.*
*If you do not provide an `_id`, one will be generated for you. ”*

# THE `_id` FIELD

- Immutable
- Unique value within the collection
- Indexed by default
- Custom values allowed
  - Scalars: `"V4C3D5C2"`, `5034`
  - Objects: `{ x: 1, y: "foo" }`
  - **No arrays**

# BSON OBJECTID

| `5033ea5c` | `e84df1` | `110f` | `000001` |
|---|---|---|---|
| Timestamp | Hostname | PID | Sequence |

- 12-byte, binary string
- Easily generated in cluster environments
- Timestamp prefix useful for sorting

# THE MONGOID CLASS

```
$id = new MongoId();

echo (string) $id;         // 5033ea5ce84df1110f000001
echo $id->getTimestamp(); // 1345579612
echo $id->getHostname();  // honeydew
echo $id->getPID();        // 3857
echo $id->getInc();        // 1

// Strings will not match an ObjectId
$c->find(['_id' => '5033ea5ce84df1110f000001']);

// Use MongoId in query criteria
$c->find(['_id' => new MongoId('5033ea5ce84df1110f000001']));
```

# INSERTING

```
$c = $db->users;

$user = $c->findOne(['username' => 'tom']);

$c->insert($user); // No error, but this does nothing

$db->users->lastError(); // Check the last error
```

```
array(
  "err" => "E11000 duplicate key error…",
  "code" => 11000,
  "n" => 0,
  "connectionId" => 16,
  "ok" => 1,
)
```

```
$c->insert($user, ['safe' => true]); // Use safe mode!
```

```
Uncaught exception: MongoCursorException:
  E11000 duplicate key error index: test.users.$_id_
  dup key: { : ObjectId('503b0772e84df1f87b000001') }
```

# WRITE CONCERN

- `{safe: false}` (default)
  - Fire and forget
  - Trade consistency for performance
- `{safe: true}`
  - Ensure primary acknowledges write
  - Issues **getLastError** command
- `{safe: #}` checks multiple servers
  - Waits for replication
  - `{safe: 3}` for primary and two slaves

# SAVING

```
// Modify Tom's user
$user = $c->findOne(['username' => 'tom']);
$user['email'] = 'tom@example.com';
$user['roles'][] = 'moderator';
$c->save($user);
```

- Combines update/upsert semantics
- Is there an `_id`?
  - Yes, update the document
  - No, insert the document

# UPDATING

```
// Change Tom's email address
$c->update(
    ['username' => 'tom'],
    ['$set' => ['email' => 'thomas@example.com']]
);
```

- Criteria and new object
- Overwrite entire documents
- Atomic operations

# UPDATING MULTIPLE DOCUMENTS

```
// Make everyone an admin (probably a bad idea :)
$c->update(
    [],
    ['$addToSet' => ['roles' => 'admin']],
    ['multiple' => true]
);
```

- Modifies one document by default
- `{multiple: true}`

# UPSERTING

```php
// Ensure Sam exists as staff
$c->update(
    ['username' => 'sam'],
    ['$set' => ['role' => 'staff']],
    ['upsert' => true]
);
```

- `update()` with `{upsert: true}`
- No multi-document operation
- Does the criteria match a document?
  - Yes, update the document
  - No, insert the document
- **Insert applies new object to criteria**

# ATOMIC OPERATORS

- Numbers
  - **$inc**
- Anything
  - **$set**
  - **$unset**
  - **$rename**
- Integers
  - **$bit**
- Arrays
  - **$addToSet**
  - **$pop**
  - **$push**, **$pushAll**
  - **$pull**, **$pullAll**

# POSITIONAL UPDATES

```php
// An article with votable comments
$db->articles->insert([
    '_id' => 1,
    'title' => 'Praesent ante dui',
    'comments' => [
        ['body' => 'Dapibus quis…', 'votes' => 2],
        ['body' => 'Fusce fermentum…', 'votes' => 0],
    ],
]);

// Upvote the second comment
$db->articles->update(
    ['comments.body' => 'Fusce fermentum…'],
    ['$inc' => ['comments.$.votes' => 1]]
);
```

# QUERIES

# BASIC QUERYING

```
$c->findOne(); // Find one document as an array
$c->find();    // Find all documents (MongoCursor)

// Query on field values
$c->findOne(['lastName' => 'Smith']);
$c->find(['lastName' => 'Smith']);
```

- Query criteria is BSON
- No grammar to parse

# QUERIES RETURN CURSORS

- Cursors navigate a query result
- Pre-query state
  - No database contact yet
  - `limit()`, `skip()`, `sort()`
  - Set **special flags**
- Post-query state
  - Iteration in-progress or completed
  - Cannot be modified

# THE MONGOCURSOR CLASS

```php
$cursor = $db->users->find();       // Find all users
$cursor->sort(['username' => -1]); // Sort by username, descending
$cursor->limit(10);                // Limit to 10 results
$cursor->skip(5);                  // Skip the first 5

foreach ($cursor as $result) { }       // Iterate through results
$results = iterator_to_array($cursor); // Get them up front

$cursor->count();     // Count before limit/skip
$cursor->count(true); // Count with respect to limit/skip
```

# AD-HOC QUERYING

*“ Arbitrary Business Requirement #42789!*
*We need the usernames for all admins that use Gmail and whose accounts were created within the last year. ”*

# DATA TO QUERY

```
$c = $db->users;

$c->save([
    'username' => 'bob',
    'email' => 'bob@example.com',
    'profile' => [
        'bio' => "I am a data fixture.",
        'createdAt' => new MongoDate(),
    ],
    'roles' => ['moderator', 'admin'],
]);

// Among others…
```

# COMPLEX CRITERIA

```
$lastYear = strtotime('last year');

// Arbitrary Business Requirement #42789!
$db->users->find([
    'email' => new MongoRegex('/gmail\.com$/i');
    'profile.createdAt' => ['$gt' => new MongoDate($lastYear)],
    'roles' => 'admin',
]);
```

- Regular expressions
- Matching values in embedded objects
- Conditional operators
- Matching a value in an array

# FIELD SELECTION

```
// Optimize by only selecting usernames
$db->users->find([
    'email' => new MongoRegex('/gmail\.com$/i');
    'profile.createdAt' => ['$gt' => new MongoDate($lastYear)],
    'roles' => 'admin',
], ['_id' => 0, 'username' => 1]);
```

- Retrieving a subset of fields
- Retrieving a slice of an array

# CONDITIONAL OPERATORS

- Comparison
  - $gt, $gte
  - $lt, $lte
  - $ne
- Logical
  - $and
  - $or, $nor
- Arrays
  - $all
  - $in, $nin
  - $size
  - $elemMatch
- Misc
  - $exists
  - $type
- Numbers
  - $mod
- Meta
  - $not
  - $where

# INDEXING

# INDEXES IN MONGODB

- B-trees
- Multiple indexes per collection
- Any field(s), top-level or embedded
- **Multikey indexing of array values**
- **Sparse**, **unique**, **geospatial**

# MANAGING INDEXES

```php
$c = $db->things;

// Ensure unique values for x
$c->ensureIndex(['x' => 1], ['unique' => true]);

// Check that index creation succeeds
$c->ensureIndex(['x' => 1], ['unique' => true, 'safe' => true]);

// Delegate index creation as a background task
$c->ensureIndex(['x' => 1], ['background' => true]);

$c->getIndexInfo(); // Array describing all indexes

$c->deleteIndex('x'); // Delete an index by name or field(s)
$c->deleteIndexes();  // Delete all indexes on the collection
```

# COMPOUND INDEXES

```php
$c->ensureIndex(['x' => 1, 'y' => 1, 'z' => -1]);

// These queries will use the index
$c->find(['x' => 'foo']);
$c->find(['x' => 'foo', 'y' => ['$gt' => 5]]);
$c->find(['x' => 'foo', 'y' => 4])->sort(['z' => -1]);
$c->find(['x' => "foo", 'y' => 8, 'z' => 'baz']);

// This query will not by default
$c->find(['y' => 6]);
```

- Index multiple fields
- Direction per field (range queries, sorting)
- Usable for constituent field queries
  - Optimal ordering
  - Query hinting

# EXPLAIN YOUR QUERIES

```
$cursor = $c->find(['x' => 'foo']);
$cursor->explain();
```

```
array(
  "cursor" => "BtreeCursor x_1", // Avoid BasicCursor
  "isMultiKey" => false,
  "n" => 0,
  "nscannedObjects" => 3,
  "nscanned" => 3,
  "indexOnly" => false,          // Fastest, if possible
  "nYields" => 0,
  "millis" => 0,
  "indexBounds" => array(…),
)
```

# INDEXING TIPS

- Contain indexes in RAM
- Mind your read/write ratio
- Support multiple queries per index
- Avoid non-indexed queries and table scans
- Caveats
  - Single-key, low-selectivity indexes
  - `$exists`, `$ne`, and `$nin`
  - `$all` and `$in`
- **Indexing advice and FAQ**

# DATABASE COMMANDS

- Queries to `$cmd` collection
- Shell and driver helpers for some
- **MongoDB::command()** for the rest
- **Reference list**
  - Sharding
  - Collections
  - Diagnostic
  - Replication
  - Indexing
  - Administration
  - Aggregation
  - Geospatial

# AGGREGATION

- Aggregation framework
- count(), distinct(), and group()
- MapReduce
- Hadoop adapter

# REPORTING

*“ Arbitrary Business Requirement #27533!*
*We need a report listing all authors that have written an article for each tag. ”*

# DATA TO AGGREGATE

```
$c = $db->articles;

$c->save(['author' => 'jen', 'tags' => ['politics', 'tech']]);
$c->save(['author' => 'sue', 'tags' => ['business']]);
$c->save(['author' => 'tom', 'tags' => ['sports']]);
$c->save([
    'author' => 'bob',
    'tags' => ['business', 'sports', 'tech']
]);
```

# MAPREDUCE

```
$map = '
function() {
    for (var i = 0; i < this.tags.length; i++) {
        emit(this.tags[i], { authors: [this.author] });
    }
}';

$reduce = '
function(key, values) {
    var result = { authors: [] };
    values.forEach(function(value) {
        value.authors.forEach(function(author) {
            if (-1 == result.authors.indexOf(author)) {
                result.authors.push(author);
            }
        });
    });
    return result;
}';
```

# MAPREDUCE

```
$rs = $db->command([
    'mapreduce' => 'articles',
    'map' => new MongoCode($map),
    'reduce' => new MongoCode($reduce),
    'out' => ['inline' => true],
]);

foreach ($rs['results'] as $r) {
    $authors = implode(', ', $r['value']['authors']);
    printf("%s: %s\n", $r['_id'], $authors);
}
```

```
business: sue, bob
politics: jen
sports: tom, bob
tech: jen, bob
```

# AGGREGATION FRAMEWORK

```
$rs = $db->command([
    'aggregate' => 'articles',
    'pipeline' => [
        ['$project' => ['author' => 1, 'tags' => 1]],
        ['$unwind' => '$tags'],
        ['$group' => [
            '_id' => '$tags',
            'authors' => ['$addToSet' => '$author']
        ]],
    ],
]);

foreach ($rs['result'] as $r) {
    $authors = implode(', ', $r['authors']);
    printf("%s: %s\n", $r['_id'], $authors);
}
```

```
business: bob, sue
tech: bob, jen
sports: bob, tom
politics: jen
```

# GRIDFS

# GRIDFS

- Large file storage in MongoDB
- BSON collections
  - `fs.files`: metadata, custom fields
  - `fs.chunks`: binary data (1+ per file)
- PHP classes
  - **MongoGridFS** extends **MongoCollection**
  - **MongoGridFSFile**
  - **MongoGridFSCursor** extends **MongoCursor**

# MONGOGRIDFS

```
// Default construction (fs.files and fs.chunks)
$grid = $db->getGridFS();
$grid = new MongoGridFS($db);

// Custom prefix (images.files and images.chunks)
$grid = $db->getGridFS('images');
$grid = new MongoGridFS($db, 'images');

$grid->chunks; // MongoCollection for chunks
```

# STORING FILES

```php
// Store a file and return its _id
$grid->storeFile('photo.jpg', ['extra' => 'metadata']);
$grid->put('photo.jpg', ['extra' => 'metadata']);

// Store bytes as a file and return its _id
$grid->storeBytes('foo', ['extra' => 'metadata']);

// Store uploads from POST data
$grid->storeUpload('field_name', ['extra' => 'metadata']);
```

# RETRIEVING FILES

```php
// Get a MongoGridFSFile by its _id
$grid->get(new MongoId('503ce546e84df14d54000000'));

// Query for a MongoGridFS file
$grid->findOne(['filename' => 'photo.jpg']);

// Query for multiple MongoGridFS files
$yesterday = new MongoDate(strtotime('yesterday'));
$grid->find(['uploadDate' => ['$gt' => $yesterday]]);
```

# MONGOGRIDFSFILE

```php
$file = $grid->findOne(['filename' => 'photo.jpg']);

$file->getFilename(); // Filename string
$file->getLength();   // File size

$file->getBytes();    // Get all chunks as a byte string
$file->getResource(); // Stream resource (new in 1.3+)

$file->write();               // Write data to stored filename
$file->write('/tmp/foo.jpg'); // Write data to a new file

$file->file; // Files document data
```

```
array(
  "_id" => <object of type MongoId>,
  "meta" => "data",
  "filename" => "/data/photo.jpg",
  "uploadDate" => <object of type MongoDate>,
  "length" => 1511769,
  "chunkSize" => 262144,
  "md5" => "40ad6901b991fc8c6e8d2e1578db4db9",
)
```

# REMOVING FILES

```php
// Delete a file (and related chunks) by its _id
$grid->delete(new MongoId('503ce546e84df14d54000000'));

// Remove files and their related chunks
$grid->remove(['filename' => 'photo.jpg']);

// Drop the files and chunks collections
$grid->drop();
```

# GEOSPATIAL INDEXING

# GEOSPATIAL INDEXING

- Geo-hashed, 2D coordinates
- Configurable bounds and bit precision
- Suggested point format: `[x, y]`
- One geospatial index per collection
- **Multi-location documents**
- **Spherical model support**

# BOUNDS QUERIES

```php
$db->places->ensureIndex(['loc' => '2d']);

// Rectangle bounds
$db->places->find(['loc' => ['$within' => ['$box' => [
    [10, 10], // Lower-left
    [20, 20], // Upper-right
]]]]);

// Circular bounds
$db->places->find(['loc' => ['$within' => ['$center' => [
    [50, 50], // Center coordinate
    15,       // Radius
]]]]);

// Arbitrary polygonal bounds (e.g. triangle)
$db->places->find(['loc' => ['$within' => ['$polygon' => [
    [10, 20],
    [10, 40],
    [30, 40], // Implicitly connected to first point
]]]]);
```

# SORTED PROXIMITY QUERIES

```php
// Find locations near a point, ordered by proximity
$db->places->find(['loc' => ['$near' => [50, 50]]]);

// Find at most 10 museums within 5 units of a point
$db->places->find([
    'loc' => ['$near' => [50, 50], '$maxDistance' => 5],
    'type' => 'museum',
])->limit(10);

// geoNear command includes distance and diagnostic data
$db->command([
    'geoNear' => 'places',
    'near' => [50, 50],
    'maxDistance' => 5,
    'query' => ['type' => 'museum'],
    'num' => 10,
]);
```

# mongoDB

- API caters to you
- Features empower you
- Scales with you

# THANKS!

- Server and drivers
- PHP documentation
- MongoDB documentation

## QUESTIONS?