Mongo DB

Ref http://en.wikipedia.org/wiki/MongoDB

Monday, April 02, 2012
6:55 PM

Keywords?

- BSON http://en.wikipedia.org/wiki/BSON
- No SQL http://en.wikipedia.org/wiki/NoSQL

Main features

- Ad hoc queries
- Indexing
- Replication
- Load balancing (Sharding http://en.wikipedia.org/wiki/Sharding)
- File storage (GridFS)
- Aggregation
- Server -side JS execution
- Capped collections

Chapter 2. Getting started

Saturday, April 14, 2012
10:33 AM

Document

At the heart of MongoDB is the concept of a document : an ordered set of keys with associated values.
The representation of a document differs by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

Ex : In JS {"greeting" : "Hello, world!"}

- Key/value pairs in documents are ordered.
- Values are not just "blobs.". They can be one of several different data types.
- Keys must not contain the character \0 (null). Null char is used to signify the end of a key.
- The . and $ characters have some special prop and used only in certain circumstances.
- Key starting with _ should be considered reserved; although this is not strictly enforced.
- Can't contain duplicate keys.

Collections

A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

Schema-Free

Collections are schema-free. This means that the documents within a single collection can have any number of different "shapes."

Naming

A collection is identified by its name. Collection names can be any UTF-8 string, with a few restrictions :

- The empty string "" is not a valid collection name.
- Collection names may not contain null char.
- You should not create any collections that start with system., a prefix reserved for system collections. For example, the *system.user*
- User-created collections shouldn't not contain the reserved character $ in the name.

Databases

In addition to grouping documents by collection, MongoDB group collections into databases. A single instance of MongoDB can host several databases.

Database naming convention :

- The empty string is not valid name.
- A name can't contain any of these characters : ' ' (single space), ., $, /, \ or \0 (null)
- Names should be all lowercase
- Names are limited to a maximum of 64bytes

Getting and Starting MongoDB

MongoDB Shell

MongoDB comes with a JS shell that allows interaction with a MongoDB instance from the command

line. The shell is very useful for performing administrative functions, inspecting a running instance, or just playing around.

Mongo DB client

Although the ability to execute arbitrary JS is cool, the real power of the shell lies in the fact that it is also a stand-alone MongoDB client. On startup, the shell connects to the test database on a MongoDB server and assigns this db connection to the global variable **db.** This var is the primary access point to MongoDB through the shell.

>use foobar
Switched to db foobar

Basic operations with the Shell

We can use the four basic operations, create, read, update, and delete (CRUD), to manipulate and view data in the shell.

1. **Create**
   The *insert* function adds a document to a collection.
2. Read
   find()
   findOne()
3. Update

Data Types

Covered the basics of what a document is.

Basic data types

"JSON-like"

1. null
2. Boolean
3. 32-bit int
4. 64-bit int
5. 64-bit floating point
6. string
7. symbol
8. object id : *{"x" : ObjectId()}*
9. date
10. regular expression : *{"x" : /foobar/i}*
11. code
12. binary data
13. maximum value
14. Minimum value
15. undefined
16. array
17. embedded document : *{"x" : {"foo" : "bar"}}*

Chapter 3. Creating, updating, and deleting document

Sunday, April 15, 2012
9:45 AM

Insert and saving

```
> db.foo.insert({"bar" : "baz"})
```

Batch insert

Using when insert multiple documents in to a collection

Inserts : Internals and Implications

Document's size does not exceed 4MB

Removing documents

```
> db.users.remove()
```

This will remove all of the documents in the users collection. This doesn't actually remove the collection and any indexes created on it will still exist.

Remove speed

Updating documents

Updates are atomic : if two updates happen at the same time, whichever one reaches the server first will be applied, and then the next one will be applied.

Document Replacement

Using modifiers

Update modifiers are special keys that can be used to specify complex update operations, such as altering, adding, or removing keys, and even manipulating arrays and embedded documents.

Getting started with "$set" modifier

"$set" sets the value of a key. If the key does not yet exist, it will be created.

"$set" can even change the type of the key it modifies.

If the user realizes that he actually doesn't like reading, he can remove the key altogether with "$unset"

U can also use "$set" to reach in and change embedded documents

Incrementing and cdecrementing

"$inc"

```
db.games.update({"game" : "pinball", "user" : "joe"},
... {"$inc" : {"score" : 50}})
```

Array modifiers

"$push"

Add an element to the end of an array if the specified key already exists and creates a new array if does not.

"$ne"

Add an element to the end of an array only if the value is not already present.

```
> db.papers.update({"authors cited" : {"$ne" : "Richie"}},
... {$push : {"authors cited" : "Richie"}})
```

"$addToSet"

Use this to prevent duplicates.

"$pop

Remove elements from either end.

`{$pop : {key : 1}}` : removes an element from the end of the array.

`{$pop : {key :  -1}}` : removes it from the beginning

"$pull"

Used to remove elements of an array that match the given criteria.

```
> db.lists.insert({"todo" : ["dishes", "laundry", "dry cleaning"]})
> db.lists.update({}, {"$pull" : {"todo" : "laundry"}})
```

Positional array modifications

Upserts

An upsert is a special type of update. If no document is found that matches the update criteria, a new document will be created by combining the criteria and update documents. If a matching document is found, it'll be updated normally.

The save shell helper

```
> var x = db.foo.findOne()
> x.num = 42
42
> db.foo.save(x)
```

Updating Multiple Documents (see Chapter 7 for more information)

```
> db.users.update({birthday : "10/13/1978"},
... {$set : {gift : "Happy Birthday!"}}, false, true)
```

This would ad the "gift" key to all user documents with birthdays on October 13, 1978

Returning Updated Documents

You can get some limited information about what was updated by calling **`getLastError`**, but it doesn't actually return the updated document. For that, you'll need the **`findAndModify`** command.

Chapter 4. Querying

Monday, April 16, 2012
2:13 PM

Introduction to find

The **find** method is used to perform queries in MongoDB.
Querying returns a subset of documents in a collection, from no documents at all to entire collection. Which documents get returned is determined by the first argument to find, which a document specifying to the query to be performed.

`> db.c.find()` ==> return everything in the collection c

`> db.users.find({"age" : 27})` ==> return all documents where the value for "age" is 27

`> db.users.find({"username" : "joe", "age" : 27})` ==> return all documents where "age" is 27 and "name" is "joe"

Specifying Which Keys to return

Purpose : reduce both the amount of data send over the wire and the time and memory used to decode documents on the client side.

`> db.users.find({}, {"username" : 1, "email" : 1})` ==> get 2 keys "name" and "email" for all documents ("_id" is returned)

`> db.users.find({}, {"fatal_weakness" : 0})` ==> not return "fatal_weakness" key

`> db.users.find({}, {"username" : 1, "_id" : 0})`==> "_id" is specified to not return.

Limitations

The value of a query document must be a constant as far as the database is concerned (it can't be a normal variable in your own code.)

```
> db.stock.find({"in_stock" : "this.num_sold"}) // doesn't work
```

Query Criteria

Query Conditionals

`"$lt", "$ne", "$eq" "$lte", "$gt", "$gte"` are all comparison operators. They can be combined to look for a range of values.

Example : look for uers who are between the ages of 18 and 30 inclusive

```
> db.users.find({"age" : {"$gte" : 18, "$lte" : 30}})
> db.users.find({"username" : {"$ne" : "joe"}})
```

OR Queries

There are 2 ways to do an OR query in MongoDB.

- "$in", "$nin" can be used to query for a variety of values for a single key.

```
> db.users.find({"user_id" : {"$in" : [12345, "joe"]})
> db.raffle.find({"ticket_no" : {"$nin" : [725, 542, 390]}})
```

- "$or" is more general, it can be used to query for any of the given value across multiple keys.

```
> db.raffle.find({"$or" : [{"ticket_no" : 725}, {"winner" :
true}]})
> db.raffle.find({"$or" : [{"ticket_no" : {"$in" : [725, 542,
390]}},   {"winner" : true}]})
```

$not

"$not" is a meta-conditional : it can be applied on top of any other criteria. As an example, let's consider the modulus operator, "$mod". "$mod" queries whole values, when devided by the first value given, have a remainder of the second value :

```
> db.users.find({"id_num" : {"$mod" : [5, 1]}})
```

And it's not divided

```
> db.users.find({"id_num" : {"$not" : {"$mod" : [5, 1]}}})
```

Rules for Conditionals

In the previous chapter, you'll notice that the $-prefixed keys are in different positions. In the query, "$lt" is in the inner document; in the update, "$inc" is the key for the outer document. This generally holds true : conditionals are an inner document key, and modifiers are always a key in the outer document.

Multiple conditions can be put on a single key.

```
> db.users.find({"age" : {"$lt" : 30, "$gt" : 20}})
```

Any number of conditionals can be used with a single key. Multiple modifiers **cannot** be used on a single key, however.

Type-Specific Queries

null

Regular Expressions

RegEx are useful for flexible string matching. For example, if we want to find all users with the name Joe or joe, we can use a REGEX to do case-insensitive matching :

```
> db.users.find({"name" : /joe/i})
```

Querying Arrays

Querying for elements of an array is simple. An array can mostly be treated as though each element is the value of the overall key.

$all

If you need to match arrays by more than one element, you can use "$all". This allows to match a list of elements. For example, suppose we created a collection with 3 elements :

```
> db.food.insert({"_id" : 1, "fruit" : ["apple", "banana",
"peach"]})
> db.food.insert({"_id" : 2, "fruit" : ["apple", "kumquat",
"orange"]})
> db.food.insert({"_id" : 3, "fruit" : ["cherry", "banana",
"apple"]})
```

Then we can find all documents with both "apple" and "banana" elements by querying with "$all"

```
> db.food.find({fruit : {$all : ["apple", "banana"]}})
      {"_id" : 1, "fruit" : ["apple", "banana", "peach"]}
      {"_id" : 3, "fruit" : ["cherry", "banana", "apple"]}
```

If you want to query for a specific element of an array, you can specify an index using the syntax ***key.index***

```
> db.food.find({"fruit.2" : "peach"})
```

$size

A useful conditional for querying arrays is "$size", which allows you to query for arrays of a given size.

```
> db.food.find({"fruit" : {"$size" : 3}})
```

One common query is to get a range of sizes . "$size" cannot be combined with another $ conditional (in this example, "$gt"), but this query can be accomplished by adding a "size" key to the document. Then, every time you add an element to the array, in-crement the value of "size". If the original update looked like this:

```
> db.food.update({"$push" : {"fruit" : "strawberry"}})
```

it can simply be changed to this:

```
> db.food.update({"$push" : {"fruit" : "strawberry"}, "$inc" :
{"size" : 1}})
```

Incrementing is extremely fast, so any performance penalty is negligible. Storing documents like this allows you to do queries such as this:

```
> db.food.find({"size" : {"$gt" : 3}})
```

The $slice operator

Querying on Embedded Documents

2 ways : querying for the whole document or querying for its individual key/value pairs.

- Querying for an entire embedded works identically to a normal query.
  However, this type of query is also order-sensitive.
- If possible, it's usually a good idea to query for just a specific key or keys of an embedded document.

```
> db.people.find({"name.first" : "Joe", "name.last" : "Schmoe"})
```

$where Queries

Cursors

The database returns results from find using a cursor. The client-side implementations of cursors generally allow you to control a great deal about the eventual output of a query. You can limit the number of results, skip over some number of results, sort results by any combination of keys in any direction, and perform a number of other powerful operations.

Limits, Skips, and Sorts

Advanced Query Options

There are 2 types of queries : *wrapped* and *plain.* A plain query is something like this

```
> var cursor = db.foo.find({"foo" : "bar"})
```

There are a couple options that "wrap" the query. Ex:

```
> var cursor = db.foo.find({"foo" : "bar"}).sort({"x" : 1})
```

$maxscan : integer

Specify the maximum number of documents that should be scanned for the query

$min : document

Start criteria for querying.

$max : document

End criteria for querying.

$hint : document

Tell the server which index to use for the query.

$explain : boolean

Get an explanation of how the query will be executed (indexes used, number of results, how long it takes, etc.), instead of actually doing the query.

$snapshot : boolean

Ensure that the query's results will be a consistent snapshot from the point in time when the query was executed. See the next section for details.

Mongo vs PHP

Monday, April 16, 2012
5:22 PM

1. Install MongoDB PHP driver
   https://github.com/mongodb/mongo-php-driver/downloads
2. Connecting to a database
   ```
   $connection = new Mongo();
   $connection = new Mongo("172.29.2.12:63445");
   ```
3. Selecting a database
   ```
   $db = $connection->selectDB("dbname");
   OR $db = $connection->dbname;
   ```

The basics (CRUD Operations)

We haven't typed any "CREATE DATABASE" commands.
We haven't created any tables or collections.
We haven't defined any schemas.
All we have done is access the database through the PHP interface as provided by the MongoDB driver and MongoDB has done all of this for us.

1. Creating/Selecting a colleciton
   ```
   $collection = $db->addresses;
   OR
   $addresses = $connection->$dbname->addresses;
   ```
2. Create a document
   ```
   $address = array(
        'first_name' => 'Peter',
        'last_name => 'Parker',
        'address' => '175 Fifth Ave',
        'city' => 'New York',
        'state' => 'NY',
        'zip' => '10010'
        );
   $addresses->insert($address);
   ```

   OR we could use **save** method. The save method just like insert method, except that if an **_id** is specified and exists, save will update instead of insert the array. I nearly always use **save** as it lead to much more reusable code in most circumstances.

Why MongoDB ?

Monday, April 16, 2012
10:22 PM

Shell

Friday, April 27, 2012
2:23 PM

Scripting the shell

http://www.mongodb.org/display/DOCS/Scripting+the+shell