Why I think Mongo is to Databases what Rails was to Frameworks

Strong statement, eh? The more I work with Mongo the more I am coming around to this way of thinking. I tell no lie when I say that I now approach Mongo with the same kind of excitement I first felt using Rails. For some, that may be enough, but for others, you probably require more than a feeling to check out a new technology.

Below are 7 Mongo and MongoMapper related features that I have found to be really awesome while working on switching Harmony, a new website management system by my company, Ordered List, to Mongo from MySQL.

Harmony

1. Migrations are Dead

Remember the first time you created and ran a migration in Rails. Can you? Think back to the exuberance of the moment when you realized tempting fate on a production server was a thing of the past. Well I have news for you Walter Cronkite, migrations are so last year.

Yep, you don't migrate when you want to add or remove columns with Mongo. Heck, you don't even add or remove columns. Need a new piece of data? Throw a new key into any model and you can start adding data to it. No need to bring your app to a screeching halt, migrate and then head back to normal land. Just add a key and start collecting data.

2. Single Collection Inheritance Gone Wild

There are times when inheritance is sweet. Let's take Harmony for example. Harmony is all about managing websites. Websites have content. Content does not equal pages. Most website management tools are called content management systems and all that means is that you get a title field and a content field. There, you can now manage content. Wrong!

Pages are made up of content. Each piece of content could be as tiny as a number or as large a massive PDF. Also, different types of pages behave differently. Technically a blog and a page are both pages, but a page has children that are most likely ordered intentionally, whereas a blog has children that are ordered by publish date.

So how did Mongo help us with this? Well, we created a base Item model. Sites have many items. Items have many custom pieces of data. So, we have an Item model that acts as the base for our Page, Blog, Link, BlogPost and such models. Then each of those defines specific keys and behaviors that they do not have in common the other items.

By using inheritance, they all share the same base keys, validations, callbacks and collection. Then for behaviors and keys that are shared by some, but not all, we are creating modules and including them. One such module is SortableItem. This gets included in Page, Blog and Link as those can all be sorted and have previous and next items. The SortableItem module defines a position key and keeps the position order in check when creating and destroying items that include it. Think of it as acts_as_list.

This has been so handy. Steve was building the doc site and said he wished he had a link type, something that shows up in the navigation, but cross links to another section or another site. I was like, so make it! Here it is in all its glory.

```
class Link < Item
  include SortableItem

  key :url, String, :required => true, :allow_blank => false

  def permalink
    Harmony.escape_url(title)
  end
end
```

Yep, barely any code. We inherit from item, include the sortable attributes, define a new key named url (where the link should go to) and make sure the permalink is always set to the title. Nothing to it. This kind of flexibility is huge when you get new feature ideas.

All these completely different documents are stored in the same collection and follow a lot of the same rules but none of them has any more data stored with it than is absolutely needed. No creating a column for any key that could be in any row. Just define the keys that go with specific document types. Sweet!
3. Array Keys

Harmony has sites and users. Users are unique across all of Harmony. One username and password and you can access any specific site or all sites of a particular account. Normally this would require a join table, maybe even some polymorphism. What we decided to do is very simple. Mongo natively understands arrays. Our site model has an array key named authorizations and our Account model has one named memberships. These two array keys store arrays of user ids. We could de-normalize even more and just have a sites array key on user, but we decided not to.

```
class Site
  include MongoMapper::Document
  key :authorizations, Array, :index => true

  # def add_user, remove_user, authorized?
end

class Account
  include MongoMapper::Document
  key :memberships, Array, :index => true

  # def add_user, remove_user, member?
end
```

What is cool about this is that it is still simple to get all the users for a given site.

```
class Site
  def users
    user_ids = [authorizations, memberships].flatten.compact.uniq
    User.all(:id => user_ids, :order => 'last_name, first_name')
  end
end
```

The sweet thing about this is that not only does Mongo know how to store arrays in documents, but you can even index the values and perform efficient queries on arrays.

Eventually, I want to roll array key stuff like this into MongoMapper supported associations, but I just haven't had a chance to abstract them yet. Look for that on the horizon.
4. Hash Keys

As if array keys were not enough, hash keys are just as awesome. Harmony has a really intelligent activity stream. Lets face it, most activity streams out there suck. Take Github's for example. I will pick on them because I know the guys and they are awesome. They are so successful, they can take it. :)

It may be handy that I can see every single user who follows or forks MongoMapper, but personally I would find it way more helpful if their activity stream just put in one entry that was more like this.

"14 users started watching MongoMapper today and another 3 forked it. Oh, and you had 400 pageviews."

Am I right? Maybe I have too many projects, but their feed is overwhelming for me at times. What we did to remedy this in Harmony is make the activity stream intelligent. When actions happen, it checks if the same action has happened recently and just increments a count. What you end up with are things in the activity stream like:

"Mongo is to Databases what Rails was to Frameworks was updated 24 times today by John Nunemaker."

On top of that, we use a hash key named source to store all of the attributes from the original object right in the activity stream collection. This means we do 0, yes 0, extra queries to show each activity. Our activity model looks something like this (obviously this is really pared down):

```
class Activity
  include MongoMapper::Document
  key :source, Hash
  key :action, String
  key :count, Integer, :default => 1
end
```

Then, we define an API in that model to normalize the different attributes that could be there. For example, here is the title method:

```
class Activity
  def title
    source['title'] || source['name'] || source['filename']
  end
end
```

In order to determine if a new action is already in the stream and just needs to be incremented, we can then use Mongo's ability to query inside hashes with something like this:

```
Activity.first({
  'source._id'   => id,
  :action        => 'updated',
  :created_at.gt => Time.zone.now.beginning_of_day.utc
})
```

How fricken sweet is that? Major. Epic.
5. Embedding Custom Objects

What is that you say? Arrays and hashes just aren’t enough for you. Well go fly a kite…or just use an embedded object. When Harmony was powered by MySQL (back a few months ago), we had an Item model and an ItemAttributes (key, value, item_id) model.

Item was the base for all the different types of content and item attributes were how we allowed for custom data on items. This meant every time we queried for an item, we also had to get its attributes. This isn’t a big deal when you know all the items up front as you can eager load some stuff, but we don’t always know all the items that are going to be shown on a page until it happens.

That pretty much killed the effectiveness of eager loading. It also meant that we just always got all of an item’s attributes each time we got an item (and performed the queries to get the info), just in case one of those attributes was being used (title and path are on all items).

With Mongo, however, we just embed custom data right with the item. Anytime we get an item, all the custom data comes with it. This is great as there is never a time where we would get an attribute without the item it is related to. For example, here is part of an item document with some custom data in it:

```
{
  "_id"   =>...,
  "_type" =>"Page",
  "title" =>"Our Writing",
  "path"  =>"/our-writing/",
  "data"  =>[
    {"_id" =>..., "file_upload"=>false, "value"=>"", "key"=>"content"},
    {"_id" =>..., "file_upload"=>true, "value"=>"", "key"=>"pic"}
  ],
}
```

Now anytime we get an item, we already have the data. No need to query for it. This alone will help performance so much in the future, that it alone had the weight to convince us to switch to Mongo, despite being almost 90% done in MySQL.

The great part is embedded objects are just arrays of hashes in Mongo, but MongoMapper automatically turns them into pure ruby objects.

```
class Item
  include MongoMapper::Document

  many :data do
```

```
    def [](key)
      detect { |d| d.key == key.to_s }
    end
  end
end

class Datum
  include MongoMapper::EmbeddedDocument

  key :key, String
  key :value
end
```

Just like that, each piece of custom data gets embedded in the item on save and converted to a Datum object when fetched from the database. The association extension on data even allows for getting data by its key quite easily like so:

```
Item.first.data['foo'] # return datum instance if foo key present
```

6. Incrementing and Decrementing

A decision we made the moment we switched to Mongo was to take advantage of its awesome parts as much as we could. One way we do that is storing published post counts on year, month and day archive items and label items. Anytime a post is published, unpublished, etc. we use Mongo's increment modifier to bump the count up or down. This means that there is no query at all needed to get the number of posts published in a given year, month or day or of a certain label if we already have that document.

We have several callbacks related to a post's publish status that call methods that perform stuff like this under the hood:

```
# ids is array of item ids
conditions = {:_id => {'$in' => ids}}

# amount is either 1 or -1 for increment and decrement
increments = {'$inc' => {:post_count => amount}}

collection.update(conditions, increments, :multi => true)
```

For now, we drop down the ruby driver (collection.update), but I have tickets (inc, the rest) to abstract this out of Harmony and into MongoMapper. Modifiers like this are super handy for us and will be even more handy when we roll out statistics in Harmony as we'll use increments to keep track of pageviews and such.
7. Files, aka GridFS

Man, with all the awesome I've mentioned above, some of you may be tired, but I need you to hang with me for one more topic. Mongo actually has a really cool GridFS specification that is implemented for all the drivers to allow storing files right in the database. I remember when storing files in the database was a horrible idea, but with Mongo this is really neat.

We currently store all theme files and assets right in Mongo. This was handy when in development for passing data around and was nice for building everything in stage before our move to production. When we were ready to move to production, we literally just dumped stage and restored it on production. Just like that all data and files were up and running.

No need for S3 or separate file backup processes. Just store the files in Mongo and serve them right out of there. We then heavily use etags and HTTP caching and intend on doing more in the future to make sure that serving these files stays performant, but that is for another day. :) As of now, it is plenty fast and sooooo convenient.
Conclusion

We have been amazed at how much code we cut out of Harmony with the switch from MySQL to Mongo. We're also really excited about the features mentioned above and how they are going to help us grow our first product, Harmony. I can't imagine building some of the flexibility we've built into Harmony or some of the ideas we have planned for the future with a relational database.

I am truly as excited about the future of Mongo as I once was (and still am) about the future of Rails.

Tags: harmony, mongodb, and mongomapper
66 Comments

1. Luigi Montanez Luigi Montanez

   Dec 18, 2009

   Great list. While points 2-5 touch on this, I think the best thing about MongoDB is that it's just a more natural fit for web applications than an RDBMS ever could be. Mapping your application's objects to documents in Mongo makes you say, "Wow, if only this was the way we always did it". There's no fighting with a competing paradigm.

   I think it's something you have to work with to really grok, so hopefully this post will get more people to try out Mongo.
2. John Nunemaker John Nunemaker

   Dec 18, 2009

   I was just using that as a reason while talking with someone recently. There is a lot less work moving your ruby objects to the database. Thanks for specifically mentioning that.
3. Dave Woodward Dave Woodward

   Dec 18, 2009

   Welcome back to Object Oriented Programming! :)

   This is the awesomeness I've been rambling about while using Smalltalk with an object database all summer.

MongoDB is an object database of sorts (at least it can now transparently serialize objects). This is going to open up a whole new class of applications for Rails (one that you're already building).

I read somebody who had a metaphor for an object's life cycle in the Rails stack. They said it is "frozen" in the database, turned to "liquid" while in a mongrel, and is "steam" if its stuck in memcache. MongoDB makes it so the objects are still liquid even when they're in the database!

Maglev is another technology (not as mature/stable as MongoDB yet) where all your objects are always a liquid! All cool stuff!

P.S. congrats on launching Harmony!

4. Jim Jim

   Dec 18, 2009

   Excellent post, John!

   Thanks for championing the MongoDB movement in the Rails community!

   I'm very thankful that we happened to sit at the same table at lunch this past RailsConf to get that ball rolling. ;-)

5. John Nunemaker John Nunemaker

   Dec 18, 2009

   @Dave Thanks! Very true about liquid vs frozen/steam.

   @Jim You're thankful? I'm thankful! Who knows how long it would have taken me to stumble across Mongo if it hadn't been for that fateful day. ;)

6. Geoffrey Grosenbach Geoffrey Grosenbach

   Dec 18, 2009

   Given how much of Harmony happens in Javascript on the client, have you tried piping JSON straight from the database to the client without an extra serialization/deserialization step in Ruby?

   I wrote an internal app (in Cappuccino.org) that skips the server altogether and sends JSON straight to CouchDB. It was another "aha" moment that opened my eyes to the value of the schema-free database.

   Of course, one needs some authentication in the middle. But the need for a server-based webapp may now be optional in some cases.

7. John Nunemaker John Nunemaker

Dec 18, 2009

@Geoff That is the next step. We don't feel that step is quite there yet, but browsers keep getting better. I totally think something like that is the future.

8. Josh Owens Josh Owens

Dec 18, 2009

John,

What types of issues did you run into when switching to Mongo?
Was it a 100% full conversion, or do you still have some data in mySQL?
Do you have any performance comparisons?
Does NewRelic track db performance with Mongo?

You have me intrigued with this post and now I am thinking I might try to switch an app over.

9. John Nunemaker John Nunemaker

Dec 18, 2009

@Josh 100%. New relic does not support mongo that I know of. We (railsmachine and orderedlist) have some ideas for a scout mongo plugin, but haven't created it yet.

As far as issues, the biggest one is freeing your mind (ala the matrix). Rails/AR are so built on conventions that you get use to just building to those conventions. It takes some time before you start to think creatively about how you want to store your data.

Worrying about tools and such is valid, but those will spring up fast, I believe.

10. Scott Motte Scott Motte

Dec 18, 2009

Great writeup John. Thanks. For the GridFS system did you use the carrierwave gem's implementation or your own custom implementation?

11. John Nunemaker John Nunemaker

Dec 18, 2009

@Scott Note that I didn't actually say we were using GridFS. :) We actually are just storing files directly in a collection. We have an asset model with key of type Binrary.

We are ok with a max size of uploads being 4MB. If you have a file bigger than that we are just going to suggest you put that file somewhere else as most website related files are small images and pdfs.

12.
Elliott Draper Elliott Draper

Dec 18, 2009

Fantastic article John. I had played with MongoDB a few weeks ago on a tiny pet project after reading one of your earlier articles, but this particular write-up has inspired me to dig deeper and work on something a bit more substantial using Mongo, to get a feel for everything that it can really offer. As discussed in the comments above, it certainly feels like a better fit for a lot of web development than traditional RDBMS, and so I can't wait to get stuck in.

I'd also like to help out with MongoMapper if I can along the way, what's the best place to get started with dev there? Just picking some issues on the GitHub issues list and giving them a go, or is there any other areas that need specific attention?

Congratulations on Harmony by the way, that's shaping up to be a great app from the looks of things.
13.
John Nunemaker John Nunemaker

Dec 18, 2009

@Elliot – Glad you found it interesting. The best way to get involved with MM is to start building apps with it and reporting back to me. :) There are still pain points and those kinds of reports from the field help me know where to focus and help shape solutions. The mailing list is a good place for conversation too.
14.
Cameron Cameron

Dec 18, 2009

Great insight. As a beginner to ruby, rails and programming in general, I was excited to read this, as a project I am working on totally needs the flexibility of a document store.

I'm just not sure how to get it working with rails. How easy is it to swap out AR for something like Mongo? I hear Rails3 will be more agnostic, but does that include stuff like Mongo and CouchDB?
15.
John Nunemaker John Nunemaker

Dec 18, 2009

@Cameron – MongoMapper is insanely easy to use with Rails. See this gist for an initializer and example database.yml. You just config.gem 'mongo_mapper' and optionally you can remove active record.
16.
Brian Brian

Dec 18, 2009

Yay, another awesome overly excitable graphic designer cum Rubyist gets excited by some half-assed hack of a database, and all the other 20-something Rubyists get excited. While I am very happy for you that your small scale CRUD apps and silly toy databases help with your productivity, you might want to dial-down the level of "AWESOME" you spew like a fat kid in a candy store.

17. Greg Greg

    Dec 18, 2009

    I have been using mongo and MM for a project the last few months. I was initially very enthusiastic but as my data model has gotten more complex, I struggled to map it to the mongo way of thinking. Mongo does not do joins, so you are encouraged to store things hierarchically. So if I have a site with departments, and departments have products. The department can contain all the products, which might be nice for showing a department page which lists the products – 1 trip to the database. But to show a product page, guess what – you have to load the entire department again. And if products are in multiple departments, you have to make products a top level collection and then you're doing multiple trips to the database to fetch them all for your product list page. I am having doubts now that document-oriented databases and web applications are a good match. Has anyone else struggled with this?
18. Curtis Curtis

    Dec 18, 2009

    @Brian, wow.
19. rick rick

    Dec 18, 2009

    What's the index story like on Mongo? Can you create/drop indexes on the fly, or does it lock the collection?
20. John Nunemaker John Nunemaker

    Dec 18, 2009

    @Brian – Way to keep it constructive. You are my hero!

    @Greg – I would definitely do different collections for Departments and Products. Just store a department_id on Product and you are good to go. Or if a product can be in multiple departments, you could use an array key. :)

    @rick – You'd have to hit the mailing list. I assume lots of data plus a new index on an existing key would lock/slow things up. Good question.
21. John Nunemaker John Nunemaker

    Dec 18, 2009

@rick – I asked for you as I am curious. I'll post back here when I receive word.

22. John Nunemaker John Nunemaker

    Dec 18, 2009

    @rick – From the 10gen crew: "Creating a new index blocks. It doesn't matter as all objects go into the index even if an object doesn't have a given field."

23. Giles Bowkett Giles Bowkett

    Dec 18, 2009

    This was awesome and I can't wait to learn more.

    (Apologies but: when describing the GitHub feed, you used "there" twice where you meant "their." Their activity stream, their feed.)

24. John Nunemaker John Nunemaker

    Dec 18, 2009

    @Giles – Thanks! No apologies needed. I wrote it in a hurry and by the end was too tired to proofread. Someone else just proofread it for me and I fixed several typos. Should be good now. :)

25. Sandeep Sandeep

    Dec 18, 2009

    I have a question – what about Cassandra ?

    Cassandra seems to be more "production ready" than Mongo and offers a lot of the same features (http://www.engineyard.com/blog/2009/cassandra-and-ruby-a-love-affair/)- did you compare the both ?

26. John Nunemaker John Nunemaker

    Dec 18, 2009

    @Sandeep – I've only grazed over cassandra. For whatever reason it didn't really click with me like Mongo did. Also, what are you basing the "more production ready" claim on?

27. towski towski

    Dec 19, 2009

    Sandeep is probably referring to Cassandra being used at Facebook.

Thanks for the great article, I am downloading MongoDB as I type (my skepticism discarded).

28.

Chris Kimm Chris Kimm

Dec 19, 2009

Thanks for the informative post. Are you using MongoDB on a 64-bit system? If not, are you concerned at all by the ~2.5 GB data limit on 32-bit systems?

29.

John Nunemaker John Nunemaker

Dec 19, 2009

@towski – Enjoy! It is a lot of fun.

@Chris – Yep, 64.

30.

Collin Miller Collin Miller

Dec 19, 2009

Working up a schema in MM, all seems quite natural.

Are you using ActiveModel::Validations or do you plan to?

That sort of thing is one of the things that excites me about Rails3.

I'd much, much, much rather do this:

```
class MyModel
  include MongoMapper::Document
  include ActiveModel::Validations
end
```

Than ever, ever, ever have a Ruby community that re-implements validations. (Unless you have something to prove about validations.)

31.

John Nunemaker John Nunemaker

Dec 19, 2009

@collin – Currently uses a fork of validatable. Actually have a validation gem all written in my head. Just need some time to work it out. If it goes well, I'll switch MM to that. If not, I'll either stick with validatable or switch to ActiveModel::Validations.

32.

Jack Dempsey Jack Dempsey

Dec 19, 2009

Nice writeup John. I saw a great presentation by Kyle Banker recently on MongoDB and it reawakened my interest in the project. This post definitely continues that theme. I often find the biggest barrier to entry is getting a simple example up and going. So, I've added an entry to Beet for mongo which should make it very easy for people to get started with it.

You can read about Beet here: http://jackdempsey.github.com/beet/
and see the actual recipe here…might look familiar :-)

http://github.com/jackdempsey/beet/blob/master/lib/beet/recipes/rails/db/mongo.rb

33.
mrb mrb

Dec 19, 2009

Can you elaborate a bit on your solution from serving files from GridFS to the user?

34.
Janko m. Janko m.

Dec 19, 2009

@John Nunemaker in reply to @Greg you said "I would definitely do different collections for Departments and Products. Just store a department_id on Product and you are good to go. Or if a product can be in multiple departments, you could use an array key. :)"

Can I ask about few use-cases for your proposition. I haven't yet found time to dig in KV stores so I am not sure how this goes. If you store department_id in Producs.
- How many queries do you need to get the list of latest 10 products and the names of the departments they are in?
- How many queries do you need to the list of top 5 departmets and random 5 products in them?

thanks, interesting article otherwise, it shed some light on mongo for me.

35.
Chris Chris

Dec 19, 2009

John – Are you planning on open-sourcing SortableItem, or similar modules you're using with MongoMapper? I think the availability of code like that would really help out the MongoMapper ecosystem.

36.
Gregory Gregory

Dec 19, 2009

Hey John,
I like your post but point 6 is kind of tumbling since rdbms also offer this kind of technique.

37.
Kebab Kebab

Dec 19, 2009

I'm with @brian.

38.

Rahsun McAfee Rahsun McAfee

Dec 19, 2009

I'm currently working on a project for a client that was originally completely built on a relational database. However, situations arose where the relational method didn't fit. (You just have to think about your problem)

Mongo made sense and it gave us the opportunity to put it to the test. It made out like a champ! Even when you have to pull multiple documents (we do it in 1 query) and just grab the data out of those documents has proven to be very effective.

Now that we decided to go with this solution we have been able to grab it from off our tool belt multiple times and implement or solve problems pretty quickly. Auto-sharding is going to be the shizzle when it goes in to stable!

Nice write up @john and good work on Mapper!

39.

John Nunemaker John Nunemaker

Dec 19, 2009

@mrb – Not sure what all you mean by elaborate. We just have a controller and route assets to it. The code is very simple right now.

```
class AssetsController &lt; ApplicationController
  before_filter :site_required

  def show
    file = current_site.assets.first(:id =&gt; params[:id], :select =&gt; [:contents, :filename,
:content_type, :updated_at])
    if file.present?
      if stale?(:etag =&gt; file, :last_modified =&gt; file.updated_at.utc, :public =&gt; true)
        send_data file.contents.to_s, :filename =&gt; file.filename, :type =&gt; file.content_type,
:disposition =&gt; 'inline'
      end
    else
      render_not_found
    end
  end
end
```

@Janko – What I was trying to say with that example is if you want to do normal relationships, you can, not that mongo makes them more awesome. Though if you are just going to use the id and name of the department, you could denormalize and store them in a hash key on product. Then whenever a department updated, you could update the names in the products.

@Chris – Yeah, there are some tweaks I need to do to it, but I can post it on RailsTips after that.

@Rahsun – Thanks for the comment. Agreed, I'm curious about the auto-sharding.

40.
Greg Greg

Dec 19, 2009

@John – the idea of storing the product ids in an array inside the department makes sense. You have to search the departments to find out which department a product is in, but indexes could help with that. Will mongo_mapper support has-many relationships in this fashion in the future?

Another issue I'm struggling with is the lack of transactions. One thing our system has to do is export a large amount of incoming data into another system periodically, and I've always used transactions to mark a a batch of records as processed while writing them to a file. That way, if the file cannot be written, it roles the whole operation back. How would you do this without transactions? And how would auto-sharding impact this kind of process? Thanks!

Greg

41.
John Nunemaker John Nunemaker

Dec 19, 2009

@Greg – Yep, I mentioned that in the post. MM will support many relationships with array keys in the future. Hopefully in the near future. :)

As per transactions, I've never really used them. Maybe here and there, but usually it wasn't a huge deal for me so I can't really speak to that.

42.
mrb mrb

Dec 19, 2009

John – I said 'elaborate' because I figured you were doing something more complex than that! I've seen some other solutions but am happy to know you're just using send_data – I guess you're not seeing bad performance issues, and the caching is working pretty well?

43.
Dru Dru

Dec 19, 2009

Great article, raised some interesting points/ideas that I hadn't considered whilst working with mongodb/mongomapper.

Out of interest, I'm wondering how you're handling protected attributes on your models? As far as I'm aware there's no mongomapper equivalent to attr_accessible or the like. Are you using a before filter at the controller level? Or have you got something in the model(s) to handle it.

44. rick rick

Dec 19, 2009

Greg: from what it sounds like, you should stick to a DB if you need ACID transactions. Don't write your billing system with Mongo (though, invoices might be a great fit for Mongo).

45. Gerhard Gerhard

Dec 19, 2009

John, this is amazing stuff, you've done really well with this article. I'm starting to write a new app, I wasn't sure whether to go with CouchDB or MongoDB, but seeing what your choice was (with examples and all), it's a no brainer really. I wasn't too keen on those map/reduce functions anyways. Cheers!

46. Collin Miller Collin Miller

Dec 19, 2009

@John great to hear about the validations :) I might try to use ActiveModel::Validations anyway and see if it works or breaks.

Awesome library :)

47. John Nunemaker John Nunemaker

Dec 19, 2009

@mrb – All requests are under 100ms right now. Who knows what will happen as load rises, but we'll deal with that then. :)

@Dru – Just filtering params hash.

@Collin – My bet is that it will just work. I know someone used it at one point and it worked.

48. Adam McCrea Adam McCrea

Dec 20, 2009

Just FYI – This post doesn't seem to be showing in your RSS feed. Great post, BTW.

49. mario mario

Dec 20, 2009

Just wondering, are you still using Rails? Seems like most of the benefit of Rails and the large number of plugins is lost if ActiveRecord is not used.

50.
John Nunemaker John Nunemaker

Dec 20, 2009

@Adam McCrea – Thanks for notice, fixed.

@mario – We do use Rails for our Harmony application. We were going to use Merb, but since it will be merged into Rails 3 in the nearest future. We decided to go Rails. Also, we use Sinatra for APIs.

F.Y.I. MongoDB is not really durable so I'm a bit worried about user information we store in there. Personally, I wouldn't use it for storing high-value data.

@Kebab – You are an idiot.

51.
Brian Takita Brian Takita

Dec 20, 2009

Hey, Great article!

Can you go more into how migrations are not necessary? Lets say I want to rename a model. How would I rename the associations in the data? Is the normal way to solve this to create a job that goes through the entire data set and update the documents? That seems like a migration. The only thing is it would take longer with the document database, right?

Do you find yourself ever needing to do this? It seems like, unless you get the names perfectly from day 1, you will need to make these sort of changes.

In the sql world, you just do a rename of the column. What approach would you take in this situation?

Thanks,
Brian

52.
Kyle Slattery Kyle Slattery

Dec 20, 2009

John—do you mean you use Sinatra for Harmony's API endpoints? Does that mean you share models between Rails and Sinatra, or do you have Sinatra connect to the DB differently?

53.
Brian Jones Brian Jones

Dec 21, 2009

Excellent post. I started playing with MongoDB last week, and ended up converting my two latest projects over to it immediately. I was shopping around for ActiveRecord-like mappers, and settled on

MongoMapper.

I feel exactly the same way you do about the whole thing. Much like Rails was a web developer's dream, and migrations a godsend, MongoDB is making a lot more sense in the DB realm.

@Brian Takita – That's a good point.

I imagine it would be trivial to rename a collection, and if you rename a column only older data would contain the unused keys. You'd definitely have to write a migration-like script if you really need to change things around, not sure how else you'd get around that, especially with a production system.

I ended up writing a rake job that drops all the collections, and rebuilds the database from db/seeds.rb. In a multi-developer scenario this has worked out the best, since we're not dependent on any data we've thrown into the DB while developing.

54. SMiGL SMiGL

Dec 21, 2009

Nice post. Thanks!

55. moonmaster9000 moonmaster9000

Dec 21, 2009

hi, first, thanks for this post, reading this was a real lightbulb moment for me.

you've talked a lot about what you gained by switching to mongodb, but i'm wondering if you could talk a little bit about what you've given up.

without relationships, i'm wondering how much work you have to do to provide data integrity. for example, suppose you have a blog application, and in it 30 posts are written by "Author X". now you need to change the author's name. do you have to change it in all 30 posts? with a relational database (assuming a post has_one author), then you would simply change the author's database record, and the has_one relationship would take care of the rest.

56. Brian Jones Brian Jones

Dec 21, 2009

@moonmaster9000 – There's two ways you can store data. Either directly inside the object, or by creating a "database reference" which stores the id of the object it points to. The second method solves the case which you came up with.

Using a blog as an example, you would create a blog collection, which contains many comments per blog entry. However, the blog/comment author relationship would then point to a separate author collection, accessed via the dbref. (From what I can tell, MongoMapper :many relationships do dbrefs automatically).

One thing you do give up by using mongodb is transactions, although that's of course dependent on what kind of system you are creating. As the official documentation states, you wouldn't want to use this for a billing system or similar.

57. Cameron Cameron

Dec 21, 2009

When you say MongoDB is not really durable…. can you elaborate on that a bit?

58. Mike H Mike H

Dec 21, 2009

In your Link/Item inheritance example, how are the different classes in the same collection differentiated?

59. Mathias Mathias

Dec 22, 2009

@john @cameron We recently added a feature that you will like if worried about durability. You can now (in master) pass fsync:true in your getlasterror commands and it will block until after an fsync has completed, ensuring that the data is on disk. Obviously using this will significantly limit the number of writes you can do per second (unless you have an SSD), but it can be useful where you can afford the overhead.

We are still working out the driver API but it is likely to be similar to the "safe-mode" flag on update and inserts.

60. Vitalik Vitalik

Dec 22, 2009

You state Mongo is a convenient tool in data migration to Rails programs, but where can I learn more features about database development? Is this process similar to MySQL database implementation?

Thanks, Vitaly

61. José Valim José Valim

Dec 23, 2009

Hey John, congratulations on your work!

I started to watch MongoMapper more closely in the last weeks because there were a bunch of users asking if we can make Devise compatible with MongoMapper (now it's!).

As one of the persons watching and contributing to Rails 3, I would suggest you taking a closer look to ActiveModel. Specially, take a look into the following modules: ActiveModel::Translation, ActiveModel::Validation and ActiveModel::Naming.

Those modules contain almost all the API used by ActionView and ActionController, which would also make MongoMapper easier to use with other plugins which were built on top of ActiveRecord. Besides, it would also add I18n for free (the main reason I haven't switched to DataMapper yet).

Finally, I'm still working/refactoring ActiveModel::Validations on this rails fork, where validations are moved to classes and it should be merged soon. We are aware that Rails Validations should be improved, and the current refactoring already makes that easier.

I propose that instead of working in a new implementation, we could continue working on more improvements to ActiveModel::Validations, so the biggest share of plugins can make use of it as well.

What do you think?

Regards!

62.

Kebab Kebab

Dec 23, 2009

hahaha. Have a nice Xmas. See ya when I'm back in town

63.

Matthew Matthew

Dec 25, 2009

Wonderful post and I share the excitement that you have for NoSQL databases and MongoDB in particular. Right now I am trying to decide between CouchDB, MongoDB, and db4o. Right now I'm leaning towards db4o because of it is a object-oriented DB, unlike Couch and Mongo. But of course I'm not using Rails for this particular application and db4o wouldn't be an option if I was.

But my curiosity with Mongo and Couch has more to do with their performance. Using Greg's example, let's say that this store is a toy store. Since Mongo and Couch like to store data hierarchically, what happens when you need to add a layer ABOVE the top-level document? For example, let's say that this toy store has some success and decides to open a few other stores. So the top level document is not "toy store", it is "toy franchise" and toy stores are sub-arrays of this json document. Then let's say that the toy franchise gets the exclusive rights to carry Jonas Brothers toys and explodes in popularity. Now it is franchising even more and adds a "region" layer to the mix. Is it so easy to turn a document into an hash/array of a newly created document?

64.

Mike Heffner Mike Heffner

Dec 26, 2009

The SortableItem makes sense for top-level documents that must be ordered. However, when creating an ordered list of sub-documents it would appear that creating an Array key would be a better solution than reimplementing acts_as_list. However, AFAICT there is no way to construct an Array key of non-simple types. In my case, I would like to construct an Array of EmbeddedDocument's. Storing the sub-documents as an Array of IDs to them and loading them manually appears relational. Is there a plan to supporting defining the type of Array elements, maybe like:

key :ordered_steps, Array, :as =&gt; Step

Where "Step" is an EmbeddedDocument class.

Associations appear to use arrays, but does MM guarantee their order across stores and reloads?

65. Mike Mike

Dec 29, 2009

Man, I don't understand 90% of this post. It might be great, but I just don't 'get' it. I'm a .NET developer, so maybe that's my problem. But what is this thing that you are talking about?

66. Patrick Patrick

Dec 31, 2009

Great post. In my case though, rails (or any other web framework) is just one of the possible interfaces to the data so I still need to use an RDBMS.

@Mike this is about a method storing/accessing your data from your application – i.e what you'd ordinarily dump into a custom binary data file, or a database. Your choice in storage method dictates to an extent how you can organise the data in there.