

John C. Zablocki

Development Lead, MagazineRadar

NJ .NET User Group

2011-05-12

# **.NET and MongoDB**

**A Code First Introduction to .NET and MongoDB**

---

# Agenda

- NoSQL Overview
- MongoDB Basic Concepts
- MongoDB Shell
- NoRM
- MongoDB C# Driver
- MongoDB Design Considerations
- In Depth: Meringue
- Case Study: RateMySnippet
- Questions?

# NoSQL

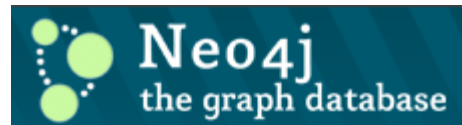
Not Only SQL

---

# What is NoSQL?

- Coined in 1998 by Carlos Strozzi to describe a database that did not expose a SQL interface
- In 2008, Eric Evans reintroduced the term to describe the growing non-RDBMS movement
- Broadly refers to a set of data stores that do not use SQL or a relational data model
- Popularized by large web sites such as Google, Facebook and Digg

# NoSQL Databases



# NoSQL Databases

- NoSQL databases come in a variety of flavors
  - XML (myXMLDB, Tamino, Sedna)
  - Wide Column (Cassandra, Hbase, Big Table)
  - Key/Value (Redis, Memcached with BerkleyDB)
  - Object (db4o, JADE)
  - Graph (neo4j, InfoGrid)
  - Document store (CouchDB, MongoDB)

# Introducing MongoDB

---

# MongoDB - Concepts

- Schema-less documents stored in collections
- Documents are stored as BSON (Binary JSON)
- JavaScript used to query and manipulate documents and collections
- Each document in a collection has a unique BSON ObjectId field named `_id`
- Collections belong to a database

# Installing MongoDB on Windows

- Download the binaries from [mongodb.org](http://mongodb.org)
- Extract to Program Files directory (or wherever)
- Create a directory `c:\data\db`
- Run `mongod.exe` from the command line with the `--install` switch
- See <http://bit.ly/aed1RW> for some gotchas
- To run the daemon without installing, simply run `mongod.exe` without arguments
- Run `mongo.exe` to verify the daemon is running

# MongoDB - Shell

- The MongoDB interactive JavaScript shell (mongo.exe) is a command line utility for working with MongoDB servers
- Allows for CRUD operations on collections
- May be used for basic administration
- Creating indexes
- Cloning databases
- Also useful as a test-bed while building apps

# MongoDB - Shell

```
/*  
  This script file demonstrates the basics of MongoDB from the interactive shell.  
  It's not intended to be a best-practice example for anything!  
*/  
  
/*Connect to a server:port/database  
(defaults are localhost:27017/test):*/  
mongo.exe localhost:27017/AltNetGroup  
  
//Switch database:  
use CodeCamp  
  
//View collections in a database:  
show collections  
  
//create an index on Name field  
db.Posts.ensureIndex({ Name : 1 });  
  
//copy one database to another  
db.copyDatabase("CodeCamp", "AltNetGroup")
```

# MongoDB - Shell

```
//create a document
var post = { Title: "On Installing MongoDB as a Service on Windows" }

//insert a document, if the collection doesn't exist it's created
db.Posts.insert(post);

//verify that the document was created
db.Posts.find();

//write a query to find a post with a valid title
var query = { Title: { $ne: null} }

//use that query to find the post
var post = db.Posts.findOne(query);

//this line will actually set the content after pressing enter
post.Content = "When installing MongoDB as a service on Windows..."
```

# MongoDB - Shell

```
//update the content to include an author using collection update method
db.Posts.update( { Title : "On Installing MongoDB as a Service on Windows"
}, { Author : "John Zablocki" } )
```

```
//check that the post was updated
db.Posts.findOne()
```

```
//where'd my document go?  updates are in place, replacing entire docs!
//need to use the $set operator to update partial documents - start over
//first remove the new document.  Notice remove takes a function argument.
//find and findOne also accept functions as arguments
db.Posts.remove(function (e) { return this.Author == "John Zablocki" })
```

```
//rerun the first statements up to but not including the
db.Posts.update(...
db.Posts.update({ Title: "On Installing MongoDB as a Service on Windows"
},
                { $set: { Author: "John Zablocki" } })
```

```
//verify that the update worked
db.Posts.findOne()
```

# MongoDB - Shell

```
//add two more tags
db.Posts.update(
{ Title: "On Installing MongoDB as a Service on Windows" },
{ $pushAll: { Tags: ["windows", "nosql"] } })

//add another post
db.Posts.insert(
{ Author : "John Zablocki", Title : "On MapReduce in MongoDB",
  Tags: ["mongodb", "nosql"]
})

//verify that last insert worked
db.Posts.findOne(function (e) { return this.Title.indexOf("MapReduce") !=
-1; })
```

# MongoDB - Shell

```
//add a "like" counter to the post. The boolean arguments tell
//update not to insert if the document doesn't exist and to
//update all documents, not just one respectively
db.Posts.update({ Author: "John Zablocki" }, { $set: { Likes: 0}
}, false, true)

//increment the likes counter for the mapreduce article
db.Posts.update({ Title: /mapreduce/i }, { $inc: { Likes: 1} })

//check that the counter was incremented
db.Posts.findOne({ Title: /mapreduce/i }).Likes
```

# MongoDB - Shell

//use MapReduce to get counts of the tags  
create the map and reduce functions

```
var map = function() {  
    if (!this.Tags) { return; }  
    for (var index in this.Tags) {  
        emit(this.Tags[index], 1);  
    }  
};
```

//conceptually, reduce gets called like:

```
//reduce("mvc", [1, 1]);  
//reduce("norm", [1  
var reduce = function(key, vals) {  
    var count = 0;  
    for (var index in vals) {  
        count += vals[index];  
    }  
    return count;  
};
```

```
/*
```

# MongoDB - Shell

run the mapreduce command on the Posts collection  
using the map and reduce functions defined above  
store the results in a collection named Tags

```
*/  
var result = db.runCommand(  
  {  
    mapreduce : "Posts",  
    map : map,  
    reduce : reduce,  
    out : "Tags"  
  });  
  
db.Tags.find()
```

# MongoDB - Shell

```
//first, insert some data
```

```
db["UserActions"].insert({ Username : "jzablocki", Action : "Login"})
db["UserActions"].insert({ Username : "jzablocki", Action : "Login"})
db["UserActions"].insert({ Username : "jzablocki", Action : "Login"})
db["UserActions"].insert({ Username : "jzablocki", Action : "PasswordChange"})
db["UserActions"].insert({ Username : "mfreedman", Action : "PasswordChange"})
db["UserActions"].insert({ Username : "mfreedman", Action : "PasswordChange"})
db["UserActions"].insert({ Username : "mfreedman", Action : "Login"})
```

```
//now run the group by
```

```
db.UserActions.group(
  { key : { Username : true, Action : true },
    cond : null,
    reduce : function(doc, out) { out.count++; },
    initial: { count: 0 }
  });
```

# Something Completely Different



# NoRM

A funny thing happened on the way to Philadelphia...



# MongoDB C# Driver

- 10gen developed and supported
- Consists of two primary components, a BSON serializer and the MongoDB driver
- Support for typed and untyped collections, MapReduce, and all CRUD operations
- Currently lacking a LINQ provider
- Current version (as of 5/5/11) is 1.0.4098.x

# MongoDB C# Driver – The Basics

```
private static MongoDBDatabase _mongoDatabase = null;

static Program() {

    //MongoServer manages access to MongoDBDatabase
    MongoServer mongoServer =
        MongoServer.Create("mongodb://localhost:27017");

    //MongoDatabase used to access MongoClient instances
    _mongoDatabase = mongoServer.GetDatabase("CodeCamp");
}
```

# MongoDB C# Driver - CRUD

```
var artist = new Artist() { Name = "The Decembrists" };

//Inserting a document into a typed collection
_mongoDatabase.GetCollection<Artist>(COLLECTION)
    .Insert(artist);

//Updating (replacing) a document in a typed collection
artist.Name = "The Decemberists";
_mongoDatabase.GetCollection<Artist>(COLLECTION)
    .Save(artist);
//Updating a nested collection
_mongoDatabase.GetCollection<Artist>(COLLECTION).Update(
    Query.EQ("Name", "The Decemberists"),
    Update.PushAll("Albums", "Castaways and Cutouts",
    "Picaresque", "Hazards of Love", "The Crane Wife")
);
```

# MongoDB C# Driver - CRUD

```
//Find all documents in a typed collection
var artists =
_mongoDatabase.GetCollection<Artist>(COLLECTION).FindAll(
);
Console.WriteLine("Artist name: " +
artists.FirstOrDefault().Name);

//Query with a document spec
var artist =
_mongoDatabase.GetCollection<Artist>(COLLECTION).FindOne(
Query.EQ("Name", "The Decemberists"));
Console.WriteLine("Album count: " + artist.Albums.Count);

//Count the documents in a collection
long count =
_mongoDatabase.GetCollection<Artist>(COLLECTION).Count();
Console.WriteLine("Document count: " + count);
```

# MongoDB C# Driver - Queries

```
var artists = _mongoDatabase.GetCollection<Artist>(COLLECTION);

//Find items in typed collection
var artistsStartingWithThe = artists.Find(Query.Matches("Name",
new Regex("the", RegexOptions.IgnoreCase)));
Console.WriteLine("First artist starting with The: " +
artistsStartingWithThe.First().Name);

//Find artists without pulling back nested collections
var artistsWithDecInTheName =
artists.Find(Query.Matches("Name", "Dec")).SetFields("Name");
Console.WriteLine("First artist with dec in name: " +
artistsWithDecInTheName.First().Name);

/////Find artists with a given tag
var artistsWithIndieTag = artists.Find(Query.In("Tags",
"Indie"));
Console.WriteLine("First artist with indie tag: " +
artistsWithIndieTag.First().Name);
```

# MongoDB C# Driver - MapReduce

```
//Add some tags
_mongoDatabase.GetCollection<Artist>(COLLECTION).Update(
    Query.EQ("Name", "The Decemberists"),
    Update.PushAll("Tags", "Folk rock",
        "Indie")
);

var artist = new Artist() {
    Name = "Sunny Day Real Estate",
    Albums = new List<string>() { "How it Feels to
be Something On", "Diary" },
    Tags = new List<string>() { "Indie", "Emo" }
};
_mongoDatabase.GetCollection<Artist>(COLLECTION).Save(artist);
```

# MongoDB C# Driver - MapReduce

```
//Create map and reduce functions
BsonJavaScript map = @"function() {
    if (!this.Tags ) { return; }
    for (index in this.Tags) { emit(this.Tags[index], 1); }
}";

BsonJavaScript reduce = @"function(previous, current) {
    var count = 0;
    for (index in current) { count += current[index]; }
    return count;
}";

var result =
_mongoDatabase.GetCollection<Artist>(COLLECTION).MapReduce(map, reduce,
MapReduceOptions.SetKeepTemp(true).SetOutput("Tags"));

var collection =
_mongoDatabase.GetCollection<Tag>(result.CollectionName);
Console.WriteLine("Tag count: " + collection.Count());
```

# MongoDB C# Driver - GroupBy

```
//add one more artist for good measure
var artists = _mongoDatabase.GetCollection<Artist>(COLLECTION);
artists.Insert(new Artist() { Name = "Blind Pilot", Albums = new
List<string>() { "3 Rounds and a Sound" } });

BsonJavaScript reduce =
    @"function(obj, out) { out.count += obj.Albums.length; }";

var groupBy = _mongoDatabase.GetCollection<Artist>(COLLECTION)
    .Group(Query.Null, GroupBy.Keys("Name"),
        new BsonDocument("count", 1), reduce, null);

foreach (var item in groupBy) {
    Console.WriteLine("{0}: {1} Album(s)",
        item.GetValue(0), item.GetValue(1));
}
```

# In Depth: Meringue

[Sign In](#) or [Register](#)

## Meringue

When life gives you lemons, make meringue...



### Recent Posts

#### [On Defining ALT.NET](#)

I had the opportunity to be part of an open spaces session at the Philly.NET Code Camp this past weekend. Part of the conversation centered around defining what ALT.NET really means or more specifically, what does it mean to be an ALT.NET developer. It's pretty clear that as ubiquitous as the term is, its definition is still largely subjective. Though I'm hardly the authority on this bit of lexicography, I'll take a stab...

#### [Grunge.NET](#)

I'm a huge fan of early 90s Seattle grunge rock. Though most of these acts had clear roots in acts such as Led Zeppelin and Jimmy Hendrix, they were often simply labeled "Alternative Rock." I think the whole ALT.NET label is pretty similar. After all, techniques and technologies such as DDD and ORMs have deep roots in late 90's OOPD patterns. Yet, the ALT.NET label gets applied, probably because most .NET devs grew up in the Microsoft space, where data-centric, procedural patterns were commonplace

#### [ALT.NET is Elitist](#)

This is a charge I hear often in podcasts, blogs and even in the open spaces event. Though I disagree that ALT.NET is fundamentally elitist, there is merit to this notion. In some ways, ALT.NET is a reaction to the often thoughtless ways .NET developers work. I've interviewed hundreds of developers over the past few years and I'm generally shocked at how many .NET devs work in drag and drop shops. I've also found that most .NET devs don't worry about things like open source, patterns or OOPD.

In no way am I trying to put down the community at large nor am I attempting to be elitist myself. The point is that there are a lot of devs, particularly in the .NET space, who don't think out of the toolbox. The ALT.NET movement evolved in part as a reaction to this behavior. ALT.NET developers learned from other platforms and languages and sought to use tools that didn't come from Redmond. ALT.NET developers don't typically like drag and drop. So by virtue of being a reaction against the way people work, I believe a lot of people feel they are the target of some dev angst.

Another reason I think devs feel ALT.NET is elitist has to do with the fact that many popular ALT.NET bloggers are quite opinionated. Let's face it, a lot of proponents of ALT.NET techniques and technologies often speak as if they have discovered nirvana (the metaphysical state, not the grunge band). I mean, even the language of DDD feels a bit elitist if not weird (ubiquitous language, really?). Even worse, are developers who treat ALT.NET as if it were some sort of scripture, blindly applying patterns to problems they don't understand. Being preachy only works in church. When I interviewed those candidates who knew nothing of common architectural patterns or OOP idioms, I took the opportunity to share what I've learned to give them another perspective. But, I hope, I never made them feel stupid for not knowing how to configure Spring.NET's XML files.

#### [What Makes an ALT.NET Developer an ALT.NET Developer?](#)

Contrary to popular belief, there is no ALT.NET stack to know. We have no secret handshake to join our society.

### Tags

[alt.net](#) [mapreduce](#)  
[mercurial](#)  
**[mongodb](#)**  
[norm](#)

[All Tags](#)

# Design Considerations

- Your object graph is your data model
- Don't be afraid to store data redundantly
  - Your graph might be redundant!
- Not everything has to fit in 1 document
- Don't be afraid to store aggregate statistics with a document.

# Object Graph as Data Model

- Generally speaking, most MongoDB drivers will serialize an object graph as a single document
  - The relationships of your classes creates an implied schema!
  - Migrating this schema is not trivial if you are trying to deserialize properties that did not or no longer exist
- Consider use cases carefully to avoid inefficiently structured documents
- Projection queries will be your friend

# Redundant Data is OK

- Optimize documents for quick reads and writes
- Your application layer will have to maintain referential integrity!
- If every time you access a Post document, you need some of an Author document's data, store that data with Post
- Design simple classes for this redundant data for reusability (see AuthorInfo in Meringue)

# Don't Stuff it All In One Document

- Nothaving formal relationships does not mean throwing away relationships
- Consider a user and his or her logged actions
  - The user would likely have a User class/doc with properties for name, email, etc.
  - User actions are generally write heavy and read out of band.
  - Don't clutter user documents - create a separate collection for user actions

# Don't Be Afraid of Extra Data

- The schema-less nature of documents makes it easy to store meta data about that document – particularly aggregate data
- Consider a blog post with a rating feature
  - Each rating would be stored as a nested document of the post
  - Rather than compute vote totals and averages real time, simply add these properties to the document and update on writes

# Case Study: Rate My Snippet

[Sign In or Register](#)

## Rate My Snippet

Search

*Where Geeks Go To Waste Time! (BETA)*

### Tags

algorithms dict factorial  
fibonacci fold  
**helloworld**  
higherorderfunctions  
ienumerable map  
**mongodb** norm  
python2 recursion reduce  
switch  
[All Tags](#)

### Rate this Snippet!

Select a rating below:

● 0 ● 1 ● 2 ● 3 ● 4 ● 5 ● 6 ● 7 ● 8 ● 9

#### Push An Item Onto A Nested Collection With NoRM

```
//add a tag to a blog Post
mongo.Database.GetCollection<Post>("Post").UpdateOne(
  new { Title = "Some Post Title"},
  new { Tags = M.Push("mongodb") }
);
```

*Posted by johnzablocki in C# on 6/26/2010 9:48:46 PM*  
Tagged with: norm mongodb

[Flag Snippet](#)

### Languages

APL Boo **C#** F#  
JavaScript Python  
[All Languages](#)

### Geeks

danielgtaylor  
**johnzablocki**  
RickMinerich  
[All Geeks](#)

Ads by Google

[C# Code Productivity Tool](#)  
Add-in to VS.NET  
For C# and ASP  
Developers.  
Download Now.  
[www.jetbrains.com/resh](#)

[Travel Like a Human](#)  
Save \$20 On All  
Travel Use Coupon  
GTRAV20  
[www.Airbnb.com](#)

[Pharmaceutical Classes](#)  
Take  
Pharmaceutical  
Classes Online!  
Request Free  
Information Today.  
[PharmTechDegrees.com](#)

[Secure Your Snippets](#)  
Code Barrel - A  
hosted, private,  
and secure code  
snippet repository.  
[www.codebarrel.com](#)

# Final Thoughts

---

Eat food. Not too much. Mostly Plants.  
- Michael Pollan

# Final Thoughts

---

Write code. Not too much. Mostly C#.  
- John Zablocki

# Links

- <http://dllHell.net> - my blog
- <http://www.CodeVoyeur.com> - my code
- <http://www.linkedin.com/in/johnzablocki>
- <http://twitter.com/codevoyeur>
- <http://mongodb.org> - Official MongoDB site
- <http://bitbucket.org/johnzablocki/meringue>
- <http://bitbucket.org/johnzablocki/codevoyeur-samples>
- <http://about.me/johnzablocki>

# Questions?

---