Robson Castilho

Código de qualidade

Testando o RavenDB

Comecei há alguns dias a estudar o **RavenDB**, banco de dados NoSQL, feito em .Net.

Neste artigo, farei uma breve introdução sobre ele, mostrando um exemplo do uso do cliente para .Net (.Net Client API) e deixando minhas impressões iniciais sobre essa tecnologia.

UM POUCO DE NOSQL

Não há uma definição formal e única onde podemos encaixar todos os bancos NoSQL existentes.

Podemos dizer que “NoSQL” é mais um movimento, impulsionado pelos esforços da Google (BigTable) e da Amazon (Dynamo) de buscarem uma nova forma de armazenamento de dados. Há pouco mais de uma década, essas empresas concluíram que escalar dados horizontalmente, com a criação de clusters, era uma boa ideia, tanto do ponto de vista técnico quanto do ponto de vista econômico.

De lá para cá, vários bancos surgiram, com várias características específicas, mas podemos dizer que os bancos NoSQL possuem algumas características comuns, que formam a “essência” desse tipo de banco:

– Não utilizam um modelo relacional
– Não possuem esquema
– Funcionam bem em clusters

RAVENDB

O RavenDB é um banco de dados NoSQL, opensource, escrito em .Net. Possui uma série de recursos, como suporte a transações, banco em memória, API RESTful, banco embutido com a aplicação (embedded database), entre tantos outros.

Ele é classificado como um banco NoSQL do tipo “Document Database”, o que significa que ele armazena e recupera documentos, que podem estar em formato XML, JSON, entre outros. De uma forma ingênua, podemos dizer que um documento equivale a uma linha de uma tabela em um modelo relacional.

Um banco do tipo “Document” é orientado a agregações (aggregates, termo que vem de Domain-Driven Design), o que significa, resumidamente, que cada documento é uma coleção de objetos relacionados, tratados como uma unidade.

Vamos clarear as coisas assim que colocarmos a mão na massa.

TEST-DRIVE

O foco do exemplo será o cliente para .Net (**.Net Client API**), embora seja possível utilizar os mesmos recursos do Raven por meio da API HTTP.

1) Vamos começar instalando o RavenDB localmente. Para isso, baixe a última versão estável em http://hibernatingrhinos.com/builds/ravendb-stable e descompacte o zip em, por exemplo, C:\RavenDB.

2) Na raiz da pasta onde você descompactou os arquivos, procure por "**start.cmd**" e execute-o. Isso abrirá o prompt de comando, inicializando o servidor do RavenDB (Raven.Server.exe) e ainda abrirá no browser o **Management Studio**, ferramenta web para gerenciamento do Raven (em endereço similar a [suamaquina][porta]/raven/studio.html).

O prompt de comando mostrará que o servidor foi iniciado com sucesso e ficará exibindo todas as requisições feitas. Mantenha o prompt e o Management Studio abertos, que precisaremos deles.

3) Já no Visual Studio, crie uma Console Application com o nome TestDriveRavenDB.

4) Instale, via NuGet, o pacote **RavenDB Client**. Feito isso, teremos duas referências adicionadas ao projeto: Raven.Abstractions e Raven.Client.Lightweight.

5) Crie uma classe "Cliente" para realizarmos operações básicas de consulta, inclusão, alteração e exclusão:

```
1  public class Cliente
2  {
3      public int Id { get; set; }
4      public string Nome { get; set; }
5      public DateTime DataDeNascimento { get; set; }
6  }
```

6) Agora vamos **incluir** um cliente no banco de dados. Basicamente iremos utilizar dois objetos: Document Store e Document Session. A API do RavenDB segue a mesma ideia de um ORM, como o NHibernate ou EntityFramework: Document Store é responsável por criar sessões (uma SessionFactory, no NHibernate), ou seja, objetos Document Session.

Dessa forma, quem já trabalha com algum ORM não terá dificuldades em entender o código abaixo:

```
class Program
{
    static void Main(string[] args)
    {
        var documentStore = new DocumentStore();
        documentStore.Url = "http://localhost:8081";
        documentStore.Initialize();

        using (var session = documentStore.OpenSession())
        {
            var cliente = new Cliente
            {
                Nome = "Robson",
                DataDeNascimento = Convert.ToDateTime("05/04/19
            };
            session.Store(cliente);
            session.SaveChanges();
        }
        documentStore.Dispose();

        Console.WriteLine("Operação concluída");
        Console.ReadKey();
    }
}
```

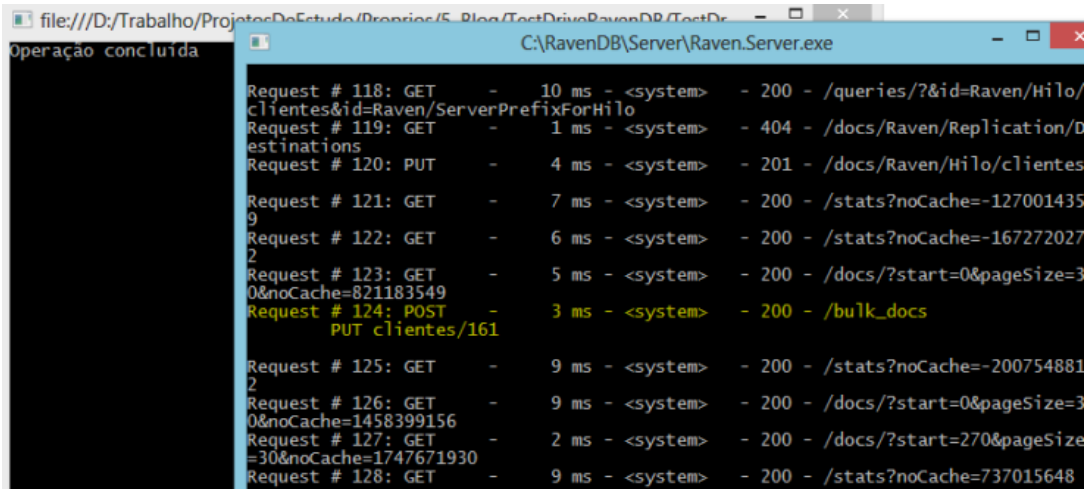
O código acima é bem simples e está contido no método Main da classe Program, default da aplicação Console. Em primeiro lugar, criamos um document store, definindo a URL do servidor (será o endereço onde está rodando o seu Management Studio).

Em seguida, criamos uma sessão através do método OpenSession(), criamos um novo cliente, armazenamos o cliente na sessão (session.Store(cliente)) e enviamos a operação para o banco de dados através do SaveChanges().

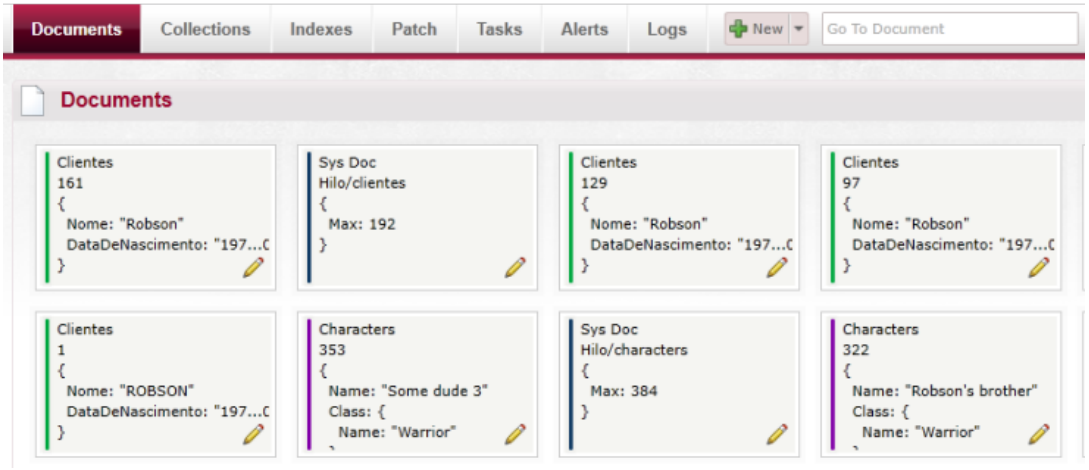
Por fim, damos um Dispose() no document store. Simples, não é?

7) Execute o programa Console e, se tudo der certo, você verá a mensagem “Operação concluída”.

Observe também no prompt onde estão sendo exibidas as requisições do RavenDB que deverá haver uma requisição HTTP com o verbo PUT, como na imagem abaixo (destacado em amarelo):



Vá ao Management Studio e clique no primeiro link do topo, chamado Documents. Você deverá ver na listagem de documentos a nova entrada para o cliente recém-criado:



Vejam pela imagem que um documento RavenDB é armazenado em formato **JSON**. Ainda pelo Management Studio, você pode ver mais detalhes do documento e até editá-lo, clicando no botão do "lápis" ou 2x sobre o documento.

8) Para realizarmos outras operações, é bastante trivial:

```
//...
using (var session = documentStore.OpenSession())
{
    // usamos o método Load para recuperar um documento pelo Ic
    var cliente = session.Load<Cliente>("clientes/161");

    // agora podemos altera-lo...
    cliente.Nome = "ROBSON [ALTERADO]";

    // ...ou exclui-lo
    session.Delete(cliente);

    // sem esquecer do SaveChanges() para confirmar a operação
    session.SaveChanges();
}
//...
```

Ainda podemos fazer queries usando LINQ (namespace Raven.Client.Linq), como estamos habituados:

```
//...
using (var session = documentStore.OpenSession())
{
    var clientes = session.Query<Cliente>().Where(c => c.Nome.S
    foreach(var cliente in clientes)
    {
        // iterando clientes
    }
}
//...
```

9) Experimente fazer alterações, exclusões e outras queries e verificar os resultados no Management Studio e as requisições feitas ao servidor, pelo prompt de comando.

CONCLUSÃO

O RavenDB é um banco de dados NoSQL, opensource, escrito em .Net e que funciona como um serviço REST. Ele possui uma API para uso com o .Net muito simples de usar por ser semelhante ao que já conhecemos em frameworks ORM, como o NH e o EF.

Neste artigo, vimos como é muito fácil colocá-lo para funcionar e fazer os primeiros experimentos com o mesmo. A .Net Client API também dá suporte à diversos outros recursos do RavenDB, como índices, attachments, paginação com LINQ, patching, entre outros.

Outras fontes de estudos:
– Site oficial: http://ravendb.net/docs
– GitHub: https://github.com/ayende/RaccoonBlog e https://github.com/ravendb/ravendb
– Exemplos que vêm com o próprio RavenDB (pasta Samples)

Quer saber mais sobre NoSQL de modo geral? Recomendo o livro “NoSQL Distilled” do Martin Fowler. Ele dá uma visão geral sobre o conceito, os tipos de bancos existentes e suas principais características.

30/05/2013
Todos

json, nosql, ravendb, test-drive

4 comentários em “Testando o RavenDB”

1. **Rafael Hitz**

   disse:
   10/06/2013 às 15:20
   Funcionou legal aqui, parabéns pelo post. Muito bacana esse RavenDB ai. =)

   1. **Robson Castilho**

      disse:
      10/06/2013 às 17:52
      Obrigado, Rafael!
      []s

2. **Rodrigo**

   disse:
   07/10/2013 às 12:22
   Muito bacana…Mas por ele armazenar via “documento”, não há perigo de se corromper? algo assim?
   E quanto a usar para autenticação de usuário, seria uma boa? Nosql parece ser boa a ideia e tal, mas é meio vago de aonde se usar exatamente, será que para criação de um sistema ERP web seria uma boa usar 100% nosql?

   1. **Robson Castilho**

      disse:
      09/10/2013 às 00:00

Olá, Rodrigo
O principal cenário para uso de um banco NoSQL é aquele onde há volumes gigantescos de dados, onde se compensa distribuí-los em clusters, e, assim, escalar melhor. Esse tipo de banco já nasce com esse propósito.
Ainda não estou utilizando em nenhum projeto em produção para dar maiores detalhes sobre problemas encontrados.
[]s e obrigado por comentar.