

عنوان:	شروع به کار با RavenDB
نویسنده:	وحید نصیری
تاریخ:	۱۳:۵۳ ۱۳۹۲/۰۶/۱۴
آدرس:	www.dotnettips.info
گروه‌ها:	NoSQL, RavenDB

پیشنیازهای بحث

- [مروری بر مفاهیم مقدماتی NoSQL](#)
- [رده‌ها و انواع مختلف بانک‌های اطلاعاتی NoSQL](#)
- [چه زمانی بهتر است از بانک‌های اطلاعاتی NoSQL استفاده کرد و چه زمانی خیر؟](#)

لطفا یکبار این پیشنیازها را پیش از شروع به کار مطالعه نمائید؛ چون بسیاری از مفاهیم پایه‌ای و اصطلاحات مرسوم دنیای NoSQL در این سه قسمت بررسی شده‌اند و از تکرار مجدد آن‌ها در اینجا صرفنظر خواهد شد.

RavenDB چیست؟

RavenDB یک بانک اطلاعاتی سورس باز NoSQL سندگرای تهیه شده با دات نت است. ساختار کلی بانک‌های اطلاعاتی NoSQL سندگرا، از لحاظ نحوه ذخیره سازی اطلاعات، با بانک‌های اطلاعاتی رابطه‌ای متداول، کاملاً متفاوت است. در اینگونه بانک‌های اطلاعاتی، رکوردهای اطلاعات، به صورت اشیاء JSON ذخیره می‌شوند. اشیاء JSON یا JavaScript Object Notation بسیار شبیه به anonymous objects سی شارپ هستند. JSON روشی است که توسط آن JavaScript اشیاء خود را معرفی و ذخیره می‌کند. به عنوان رقیبی برای XML مطرح است؛ نسبت به XML اندکی فشرده‌تر بوده و عموماً دارای اسکیمای خاصی نیست و در بسیاری از اوقات تفسیر المان‌های آن به مصرف کننده واگذار می‌شود. در JSON عموماً سه نوع المان پایه مشاهده می‌شوند:

- اشیاء که به صورت {object} تعریف می‌شوند.
- مقادیر "key": "value" که شبیه به نام خواص و مقادیر آن‌ها در دات نت هستند.
- و آرایه‌ها به صورت [array]

همچنین ترکیبی از این سه عنصر یاد شده نیز همواره میسر است. برای مثال، یک key مشخص می‌تواند دارای مقداری حاوی یک آرایه یا شیء نیز باشد.

JSON: JavaScript Object Notation

```
document : {
  key: "Value",
  another_key: {
    name: "embedded object"
  },
  some_date: new Date(),
  some_number: 12
}
```

C# anonymous object

```
var Document = new {
  Key= "Value",
  AnotherKey= new {
    Name = "embedded object"
  },
  SomeDate = DateTime.Now(),
  SomeNumber = 12
};
```

به این ترتیب می‌توان به یک ساختار دلخواه و بدون اسکیمای از هر سند به سند دیگری رسید. اغلب بانک‌های اطلاعاتی سندگرا، اینگونه اسناد را در زمان ذخیره سازی، به یک سری binary tree تبدیل می‌کنند تا تهیه کوئری بر روی آن‌ها بسیار سریع شود. مزیت دیگر استفاده از JSON، سادگی و سرعت بالای Serialize و Deserialize اطلاعات آن برای ارسال به کلاینت‌ها و یا دریافت آن‌ها است؛ به همراه فشرده‌تر بودن آن نسبت به فرمت‌های مشابه دیگر مانند XML.

یک نکته مهم

اگر پیشنهادها بحث را مطالعه کرده باشید، حتماً بارها با این جمله که دنیای NoSQL از تراکنش‌ها پشتیبانی نمی‌کند، برخورد داشته‌اید. این مطلب در مورد RavenDB صادق نیست و این بانک اطلاعاتی NoSQL خاص، از تراکنش‌ها پشتیبانی می‌کند. RavenDB در Document store خود ACID عملکرده و از تراکنش‌ها پشتیبانی می‌کند. اما تهیه ایندکس‌های آن بر مبنای مفهوم عاقبت یک دست شدن عمل می‌کند.

مجوز استفاده از RavenDB

هرچند مجموعه سرور و کلاینت RavenDB سورس باز هستند، اما این مورد به معنای رایگان بودن آن نیست. مجوز استفاده از RavenDB نوع خاصی به نام AGPL است. به این معنا که یا کل کار مشتق شده خود را باید به صورت رایگان و سورس باز ارائه دهید و یا اینکه مجوز استفاده از آن را برای کارهای تجاری بسته خود خریداری نمایید. نسخه استاندارد آن نزدیک به هزار دلار است و [نسخه سازمانی آن](#) نزدیک به 2800 دلار به ازای هر سرور.

شروع به نوشتن اولین برنامه کار با RavenDB

ابتدا یک پروژه کنسول ساده را آغاز کنید. سپس کلاس‌های مدل زیر را به آن اضافه نمایید:

```
using System.Collections.Generic;

namespace RavenDBSample01.Models
{
    public class Question
    {
        public string By { set; get; }
        public string Title { set; get; }
        public string Content { set; get; }

        public List<Comment> Comments { set; get; }
        public List<Answer> Answers { set; get; }

        public Question()
        {
            Comments = new List<Comment>();
            Answers = new List<Answer>();
        }
    }
}

namespace RavenDBSample01.Models
{
    public class Comment
    {
        public string By { set; get; }
        public string Content { set; get; }
    }
}

namespace RavenDBSample01.Models
{
    public class Answer
    {
        public string By { set; get; }
        public string Content { set; get; }
    }
}
```

سپس به کنسول [یاور شل نیوگت](#) در ویژوال استودیو مراجعه کرده و دستورات ذیل را جهت افزوده شدن وابستگی‌های مورد نیاز RavenDB، صادر کنید:

```
PM> Install-Package RavenDB.Client
PM> Install-Package RavenDB.Server
```

به این ترتیب بسته‌های کلاینت (مورد نیاز جهت برنامه نویسی) و سرور RavenDB به پروژه جاری اضافه خواهند شد (نگارش 2.5

در زمان نگارش این مطلب؛ جمعا نزدیک به 75 مگابایت).

اکنون به پوشه packages\RavenDB.Server.2.5.2700\tools Raven.Server.exe را اجرا کنید تا سرور RavenDB شروع به کار کند. این سرور به صورت پیش فرض بر روی پورت 8080 اجرا می‌شود. از این جهت که در RavenDB نیز همانند سایر Document Stores مطرح، امکان دسترسی به اسناد از طریق REST API و URLها وجود دارد. البته لازم به ذکر است که RavenDB در 4 حالت برنامه کنسول (همین سرور فوق)، نصب به عنوان یک سرویس ویندوز NT، هاست شدن در IIS و حالت مدفون شده یا Embedded قابل استفاده است.

خوب؛ همین اندازه برای برپایی اولیه RavenDB کفایت می‌کند.

```
using Raven.Client.Document;
using RavenDBSample01.Models;

namespace RavenDBSample01
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var store = new DocumentStore
                {
                    Url = "http://localhost:8080"
                }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    session.Store(new Question
                    {
                        By = "users/Vahid",
                        Title = "Raven Intro",
                        Content = "Test...."
                    });
                    session.SaveChanges();
                }
            }
        }
    }
}
```

اکنون کدهای برنامه کنسول را به نحو فوق برای ذخیره سازی اولین سند خود، تغییر دهید. کار با ایجاد یک DocumentStore که به آدرس سرور اشاره می‌کند و کار مدیریت اتصالات را برعهده دارد، شروع خواهد شد. اگر نمی‌خواهید URL را درون کدهای برنامه مقدار دهی کنید، می‌توان از فایل کانفیگ برنامه نیز برای این منظور کمک گرفت:

```
<connectionStrings>
  <add name="ravenDB" connectionString="Url=http://localhost:8080"/>
</connectionStrings>
```

در این حالت باید خاصیت ConnectionStringName شیء DocumentStore را مقدار دهی نمود. سپس با ایجاد Session در حقیقت یک Unit of work آغاز می‌شود که درون آن می‌توان انواع و اقسام دستورات را صادر نمود و سپس در پایان کار، با فراخوانی SaveChanges، این اعمال ذخیره می‌گردند. در RavenDB یک سشن باید طول عمری کوتاه داشته باشد و اگر تعداد عملیاتی که در آن صادر کرده‌اید، زیاد است با خطای زیر متوقف خواهید شد:

The maximum number of requests (30) allowed for this session has been reached.

البته این نوع محدودیت‌ها عمدی است تا برنامه نویسی به طراحی بهتری برسد.

در یک برنامه واقعی، ایجاد DocumentStore یکبار در آغاز کار برنامه باید انجام گردد. اما هر سشن یا هر واحد کاری آن، به ازای تراکنش‌های مختلفی که باید صورت گیرند، بر روی این DocumentStore، ایجاد شده و سپس بسته خواهند شد. برای مثال در یک برنامه ASP.NET، در فایل Global.asax در زمان آغاز برنامه، کار ایجاد DocumentStore انجام شده و سپس به ازای هر درخواست رسیده، یک سشن RavenDB ایجاد و در پایان درخواست، این سشن آزاد خواهد شد.

برنامه را اجرا کنید، سپس به کنسول سرور RavenDB که پیشتر آن را اجرا نمودیم مراجعه نمایید تا نمایی از عملیات انجام شده را بتوان مشاهده کرد:

```
Raven is ready to process requests. Build 2700, Version 2.5.0 / 6dce79a Server started in 14,438 ms
Data directory: D:\Prog\RavenDBSample01\packages\RavenDB.Server.2.5.2700\tools\Database\System
HostName: <any> Port: 8080, Storage: Esent
Server Url: http://localhost:8080/
Available commands: cls, reset, gc, q
Request # 1: GET - 514 ms - <system> - 404 - /docs/Raven/Replication/Destinations
Request # 2: GET - 763 ms - <system> - 200 - /queries/?&id=Raven%2FHilo%2Fquestions&id=Raven%2FServerPrefixForHilo
Request # 3: PUT - 185 ms - <system> - 201 - /docs/Raven/Hilo/questions
Request # 4: POST - 103 ms - <system> - 200 - /bulk_docs
PUT questions/1
```

زمانیکه سرور RavenDB در حالت دیباگ در حال اجرا باشد، لاگ کلیه اعمال انجام شده را در کنسول آن می‌توان مشاهده نمود. همانطور که مشاهده می‌کنید، یک کلاینت RavenDB با این بانک اطلاعاتی با پروتکل HTTP و یک REST API ارتباط برقرار می‌کند. برای نمونه، کلاینت در اینجا با اعمال یک HTTP Verb خاص به نام PUT، اطلاعات را درون بانک اطلاعاتی ذخیره کرده است. تبادل اطلاعات نیز با فرمت JSON انجام می‌شود.

عملیات PUT حتما نیاز به یک Id از پیش مشخص دارد و این Id، پیشتر در سطرى که Hilo در آن ذکر شده (یکی از الگوریتم‌های محاسبه Id در RavenDB)، محاسبه گردیده است. برای نمونه در اینجا الگوریتم Hilo مقدار "questions/1" را به عنوان Id محاسبه شده بازگشت داده است.

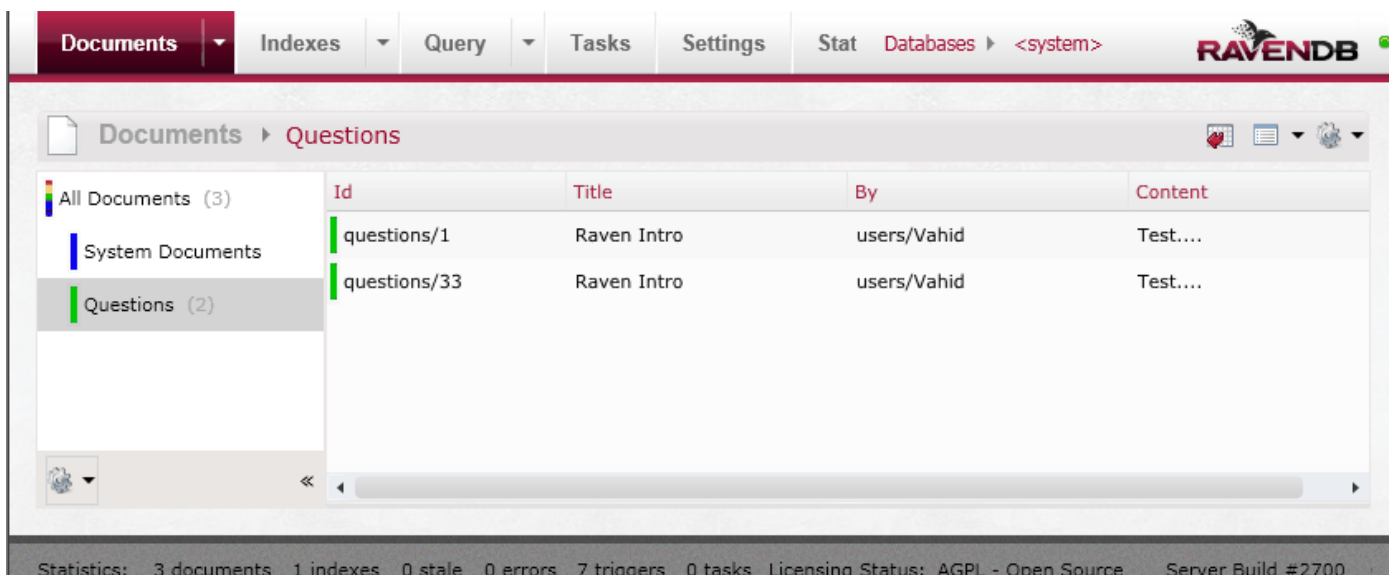
در سطرى که عملیات Post به آدرس bulk_docs سرور ارسال گردیده است، کار ارسال یکباره چندین شیء JSON برای کاهش رفت و برگشت‌ها به سرور انجام می‌شود.

و برای کوئری گرفتن مقدماتی از اطلاعات ثبت شده می‌توان نوشت:

```
using (var session = store.OpenSession())
{
    var question1 = session.Load<Question>("questions/1");
    Console.WriteLine(question1.By);
}
```

نگاهی به بانک اطلاعاتی ایجاد شده

در همین حال که سرور RavenDB در حال اجرا است، مرورگر دلخواه خود را گشوده و سپس آدرس <http://localhost:8080> را وارد نمایید. بلافاصله، کنسول مدیریتی تحت وب این بانک اطلاعاتی که با سیلورلایت نوشته شده است، ظاهر خواهد شد:

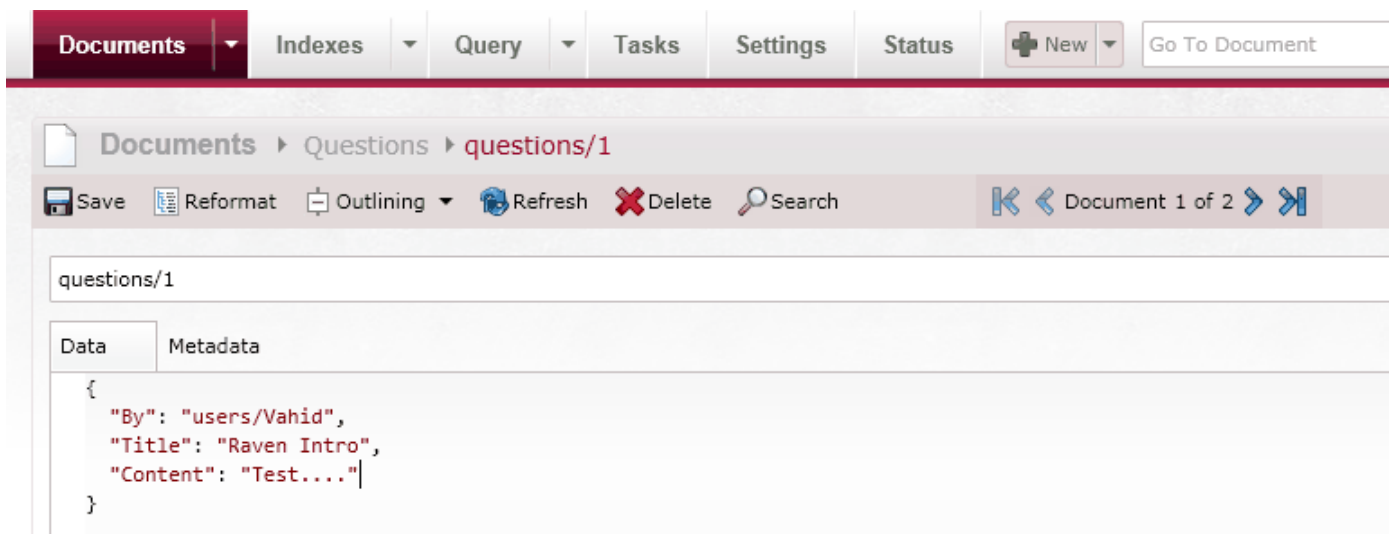


The screenshot shows the RavenDB web interface. The top navigation bar includes 'Documents', 'Indexes', 'Query', 'Tasks', 'Settings', 'Stat', 'Databases', and '<system>'. The 'Documents' tab is selected, and the 'Questions' collection is chosen. A table displays the following documents:

	Id	Title	By	Content
All Documents (3)	questions/1	Raven Intro	users/Vahid	Test....
System Documents	questions/33	Raven Intro	users/Vahid	Test....
Questions (2)				

At the bottom, a status bar shows: Statistics: 3 documents 1 indexes 0 stale 0 errors 7 triggers 0 tasks Licensing Status: AGPL - Open Source Server Build #2700

و اگر بر روی هر سطر اطلاعات دوبار کلیک کنید، به معادل JSON آن نیز خواهید رسید:



The screenshot shows the RavenDB web interface with the 'questions/1' document selected. The 'Data' tab is active, displaying the following JSON:

```
{
  "By": "users/Vahid",
  "Title": "Raven Intro",
  "Content": "Test...."
}
```

The interface also shows a 'Metadata' tab and various action buttons like 'Save', 'Reformat', 'Outlining', 'Refresh', 'Delete', and 'Search'.

اینبار برنامه را به صورت زیر تغییر دهید تا روابط بین کلاس‌ها را نیز پیاده سازی کند:

```
using System;
using Raven.Client.Document;
using RavenDBSample01.Models;

namespace RavenDBSample01
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {

```

```

        var question = new Question
        {
            By = "users/Vahid",
            Title = "Raven Intro",
            Content = "Test...."
        };
        question.Answers.Add(new Answer
        {
            By = "users/Farid",
            Content = "بررسی می‌شود"
        });
        session.Store(question);

        session.SaveChanges();
    }

    using (var session = store.OpenSession())
    {
        var question1 = session.Load<Question>("questions/1");
        Console.WriteLine(question1.By);
    }
}
}
}
}
}

```

در اینجا یک سؤال به همراه پاسخی به آن تعریف شده است. همچنین در مرحله بعد، نحوه کوئری گرفتن مقدماتی از اطلاعات را بر اساس Id سند مرتبط، مشاهده می‌کنید. چون یک Session، الگوی واحد کار را پیاده سازی می‌کند، اگر پس از Load یک سند، خواصی از آن را تغییر دهیم و در پایان Session متد SaveChanges فراخوانی شود، به صورت خودکار این تغییرات به بانک اطلاعاتی نیز اعمال خواهند شد (روش به روز رسانی اطلاعات). این مورد بسیار شبیه است به مباحث پایه ای Change tracking که در بسیاری از ORM‌های معروف تاکنون پیاده سازی شده‌اند. روش حذف اطلاعات نیز به همین ترتیب است. ابتدا سند مورد نظر یافت شده و سپس متد session.Delete بر روی این شیء یافت شده فراخوانی گردیده و در پایان سشن باید SaveChanges جهت نهایی شدن تراکنش فراخوانی گردد.

اگر برنامه فوق را اجرا کرده و به ساختار اطلاعات ذخیره شده نگاهی بیندازیم به شکل زیر خواهیم رسید:

Documents
Indexes
Query
Tasks
Settings

Documents
Questions
questions/65

Save
Reformat
Outlining
Refresh
Delete
Search

questions/65

Data
Metadata

```

{
  "By": "users/Vahid",
  "Title": "Raven Intro",
  "Content": "Test....",
  "Comments": [],
  "Answers": [
    {
      "By": "users/Farid",
      "Content": "بررسی می‌شود"
    }
  ]
}

```

نکته جالبی که در اینجا وجود دارد، عدم نیاز به join نویسی برای دریافت اطلاعات وابسته به یک شیء است. اگر سؤالی وجود دارد، پاسخ‌های به آن و یا سایر نظرات، یکجا داخل همان سؤال ذخیره می‌شوند و به این ترتیب سرعت دسترسی نهایی به اطلاعات بیشتر شده و همچنین قفل گذاری روی سایر اسناد کمتر. این مساله نیز به ذات NoSQL و یا غیر رابطه‌ای RavenDB بر می‌گردد. در بانک‌های اطلاعاتی NoSQL، مفاهیمی مانند کلیدهای خارجی، JOIN بین جداول و امثال آن وجود خارجی ندارند. برای نمونه اگر به کلاس‌های مدل‌های برنامه دقت کرده باشید، خبری از وجود Id در آن‌ها نیست. RavenDB یک Document store است و نه یک Relation store. در اینجا کل درخت تو در توی خواص یک شیء دریافت و به صورت یک سند ذخیره می‌شود. به حاصل این نوع عملیات در دنیای بانک‌های اطلاعاتی رابطه‌ای، Denormalized data هم گفته می‌شود.

البته می‌توان به کلاس‌های تعریف شده خاصیت رشته‌ای Id را نیز اضافه کرد. در این حالت برای مثال در حالت فراخوانی متد Load، این خاصیت رشته‌ای، با Id تولید شده توسط RavenDB مانند "questions/1" مقدار دهی می‌شود. اما از این Id برای تعریف ارجاعات به سؤالات و پاسخ‌های متناظر استفاده نخواهد شد؛ چون تمام آن‌ها جزو یک سند بوده و داخل آن قرار می‌گیرند.

نظرات خوانندگان

نویسنده: رضا ساکت
تاریخ: ۱۹:۲۰ ۱۳۹۲/۰۸/۰۵

با سلام و احترام

بیان کردید "البته لازم به ذکر است که RavenDB در 4 حالت برنامه کنسول (همین سرور فوق)، نصب به عنوان یک سرویس ویندوز NT، هاست شدن در IIS و حالت مدفون شده یا Embedded قابل استفاده است.
" خواهش میکنم راجع به هر مورد توضیح دهید.

نویسنده: وحید نصیری
تاریخ: ۲۰:۱۱ ۱۳۹۲/۰۸/۰۵

در ادامه دوره در مطلب « [بررسی حالت‌های مختلف نصب RavenDB](#) » در این مورد بیشتر بحث شده.

نویسنده: vici
تاریخ: ۱۱:۳۸ ۱۳۹۲/۱۰/۲۹

سلام

آقای نصیری موارد کاربردهای Raven.DB چی هست؟

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۱:۴۴ ۱۳۹۲/۱۰/۲۹

لطفا پیشنیازهای بحث را که در ابتدای مطلب عنوان شده مطالعه کنید تا با مفاهیم اولیه و علت وجودی بانک‌های اطلاعاتی NoSQL آشنا شوید.