عنوان: **ایندکس‌ها در RavenDB**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۱۵ ۱۴:۷
آدرس: www.dotnettips.info
برچسب‌ها: NoSQL, RavenDB

RavenDB یک Document database است و در این نوع بانک‌های اطلاعاتی، اسکیما و ساختار مشخصی وجود ندارد. شاید اینطور به نظر برسد، زمانیکه با دات نت کلاینت RavenDB کار می‌کنیم، یک سری کلاس مشخص دات نتی داشته و این‌ها ساختار اصلی کار را مشخص می‌کنند. اما در عمل RavenDB چیزی از این کلاس‌ها و خواص نمی‌داند و این کلاس‌های دات نتی صرفا کمکی هستند جهت سهولت اعمال Serialization و Deserialization اطلاعات. زمانیکه اطلاعاتی را در RavenDB ذخیره می‌کنیم، هیچ نوع قیدی در مورد ساختار نوع سندی که در حال ذخیره است، اعمال نمی‌شود.
خوب؛ اکنون این سؤال مطرح می‌شود که RavenDB چگونه اطلاعاتی را در این اسناد بدون اسکیما جستجو می‌کند؟ اینجا است که مفهوم و کاربرد ایندکس‌ها مطرح می‌شوند. ما در قسمت قبل که کوئری نویسی مقدماتی را بررسی کردیم، عملا ایندکس خاصی را به صورت دستی جهت انجام جستجوها ایجاد نکردیم؛ از این جهت که خود RavenDB به کمک امکانات dynamic indexing آن، پیشتر اینکار را انجام داده است. برای نمونه به سطر ارسال کوئری به سرور، که در قسمت قبل ارائه شد، دقت کنید. در اینجا ارسال کوئری به indexes/dynamic کاملا مشخص است:

```
Request #   2: GET     - 3,818 ms - <system>   - 200 -
/indexes/dynamic/Questions?&query=Title%3ARaven*&pageSize=128
```

Dynamic Indexes یا ایندکس‌های پویا

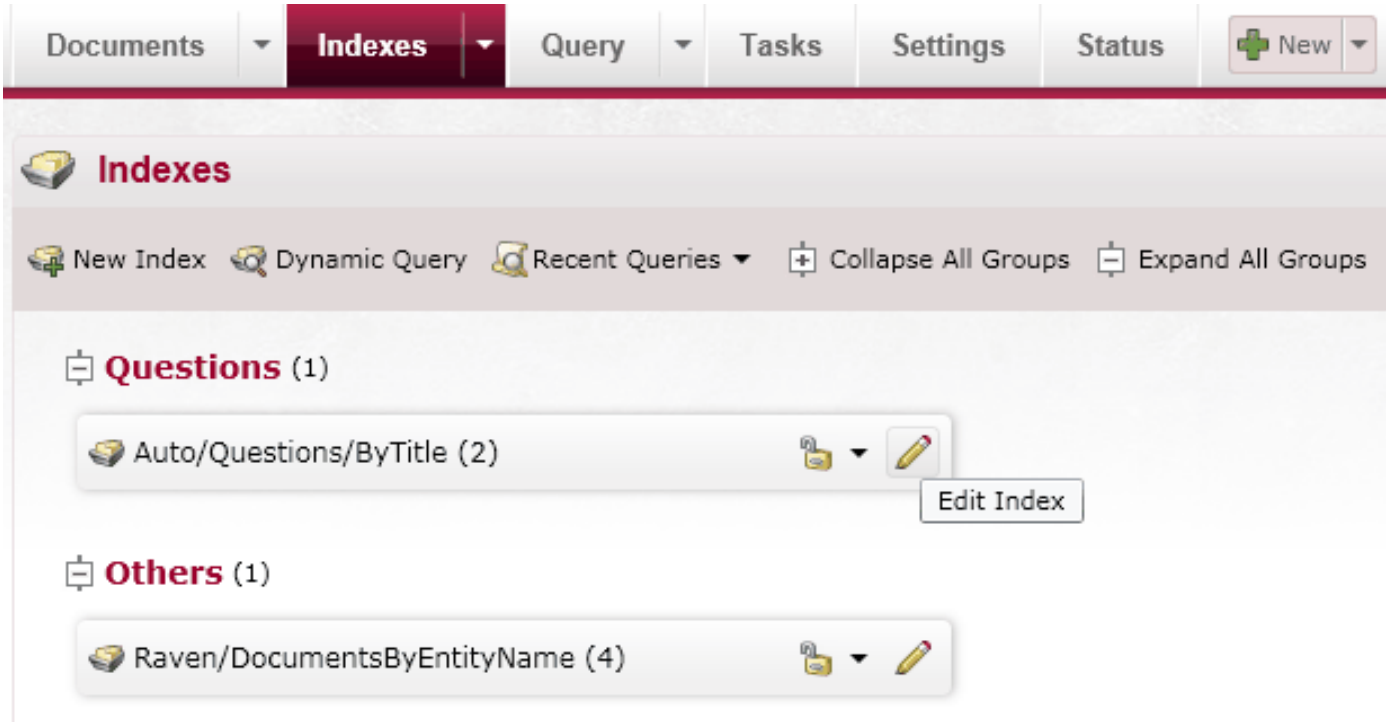
ایندکس‌های پویا زمانی ایجاد خواهند شد که ایندکس صریحی توسط برنامه نویس تعریف نگردد. برای مثال زمانیکه یک کوئری LINQ را صادر می‌کنیم، RavenDB بر این اساس و برای مثال فیلدهای قسمت Where آن، ایندکس پویایی را تولید خواهد کرد. ایجاد ایندکس‌ها در RavenDB از اصل عاقبت یک دست شدن پیروی می‌کنند. یعنی مدتی طول خواهد کشید تا کل اطلاعات بر اساس ایندکس جدیدی که در حال تهیه است، ایندکس شوند. بنابراین تولید ایندکس‌های پویا در زمان اولین بار اجرای کوئری، کوئری اول را اندکی کند جلوه خواهند داد؛ اما کوئری‌های بعدی که بر روی یک ایندکس آماده اجرا می‌شوند، بسیار سریع خواهند بود.

Static indexes یا ایندکس‌های ایستا

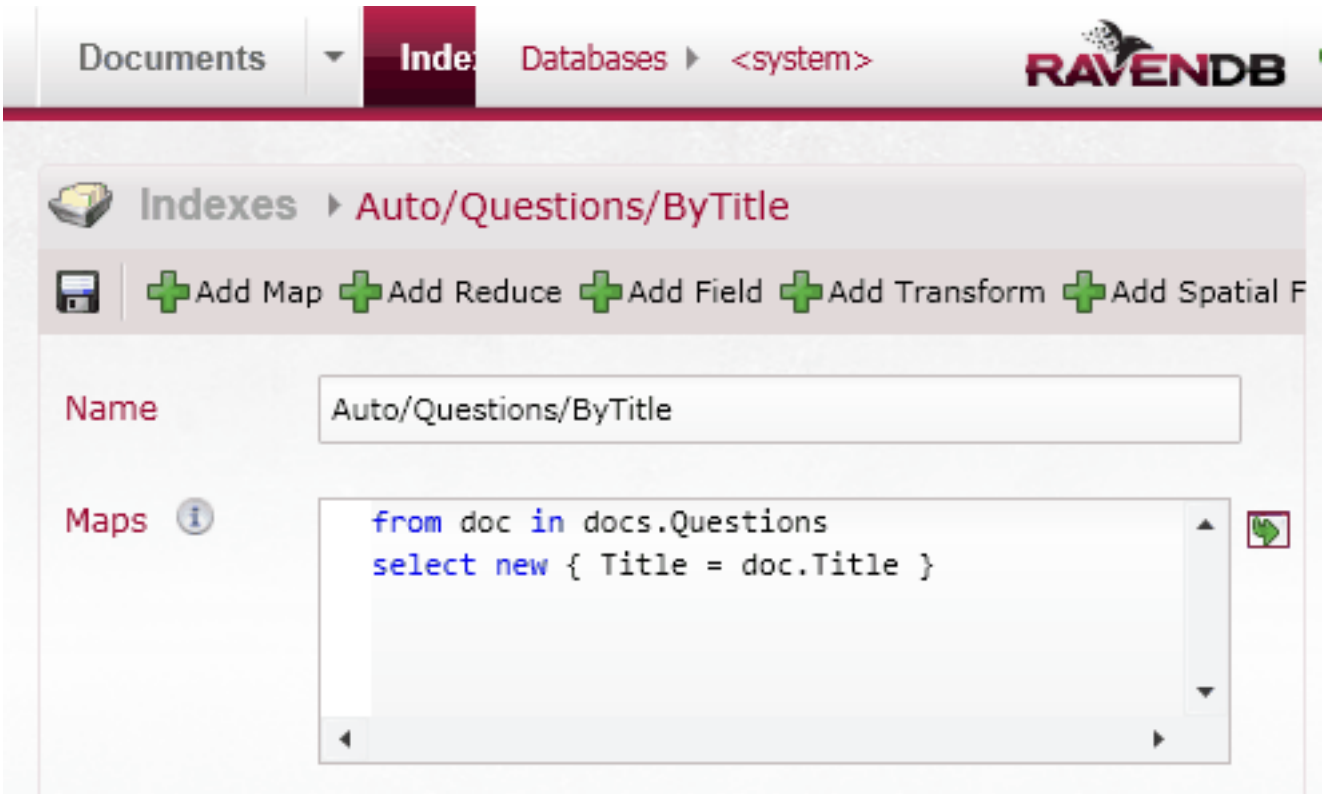
ایندکس‌های پویا به دلیل وقفه ابتدایی که برای تولید آن‌ها وجود خواهد داشت، شاید آنچنان مطلوب به نظر نرسند. اینجا است که مفهوم ایندکس‌های ایستا مطرح می‌شوند. در این حالت ما به RavenDB خواهیم گفت که چه چیزی را ایندکس کند. برای تولید ایندکس‌های ایستا، از مفاهیم Map/Reduce که در پیشنیازهای دوره جاری در مورد آن بحث شد، استفاده می‌گردد. خوشبختانه تهیه Map/Reduceها در RavenDB پیچیده نبوده و کل عملیات آن توسط کوئری‌های LINQ قابل پیاده سازی است.
تهیه ایندکس‌های پویا نیز در تردهای پس‌زمینه انجام می‌شوند. از آنجائیکه RavenDB برای اعمال Read، بهینه سازی شده است، با ارسال یک کوئری به آن، این بانک اطلاعاتی، کلیه اطلاعات آماده را در اختیار شما قرار خواهد داد؛ صرفنظر از اینکه کار تهیه ایندکس تمام شده است یا خیر.

چگونه یک ایندکس ایستا را ایجاد کنیم؟

اگر به کنسول مدیریتی سیلورلایت RavenDB مراجعه کنیم، حاصل کوئری‌های LINQ قسمت قبل را در برگه‌ی ایندکس‌های آن می‌توان مشاهده کرد:



در اینجا بر روی دکمه Edit کلیک نمائید، تا با نحوه تهیه این ایندکس پویا آشنا شویم:



این ایندکس، یک نام داشته به همراه قسمت Map از پروسه Map/Reduce که توسط یک کوئری LINQ تهیه شده است. کاری که در اینجا انجام شده، ایندکس کردن کلیه سؤالات، بر اساس خاصیت عنوان آن‌ها است.

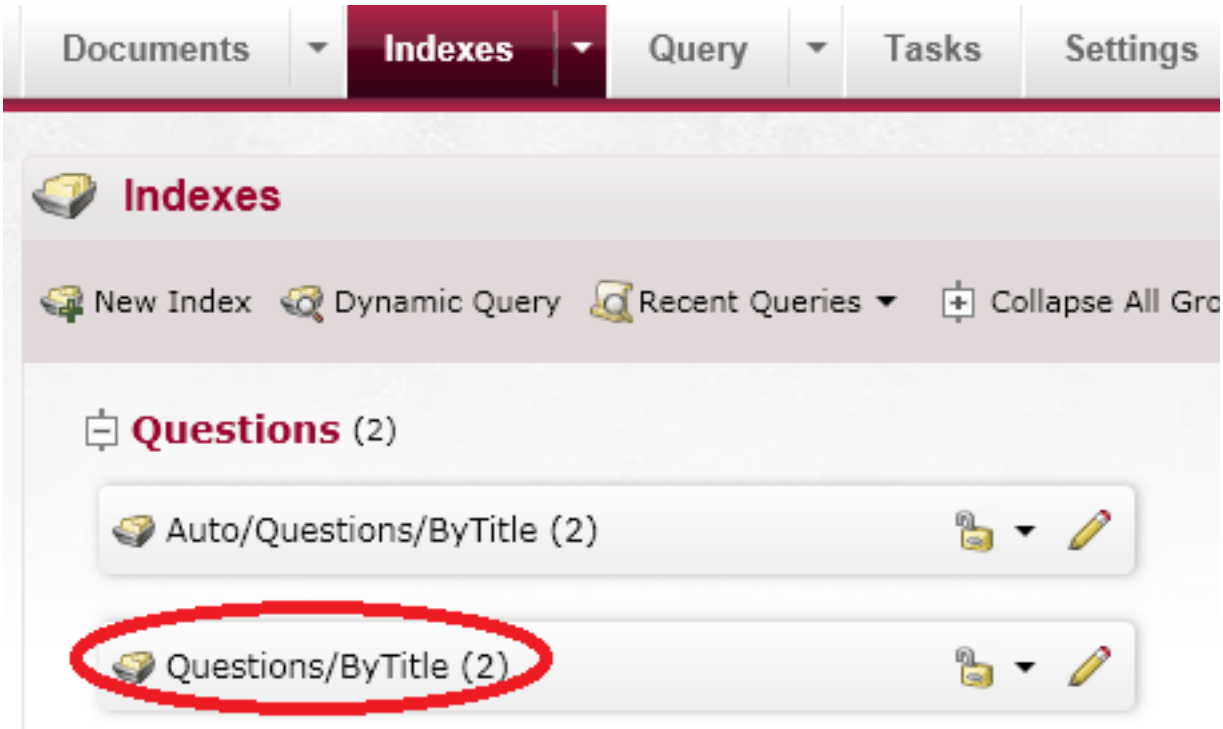
اکنون اگر بخواهیم همین کار را با کدنویسی انجام دهیم، به صورت زیر می‌توان عمل کرد:

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

namespace RavenDBSample01
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                store.DatabaseCommands.PutIndex(
                name: "Questions/ByTitle",
                indexDef: new IndexDefinitionBuilder<Question>
                {
                    Map = questions => questions.Select(question => new { Title = question.Title } )
                });
            }
        }
    }
}
```

کار با شیء DatabaseCommands یک DocumentStore شروع می‌شود. سپس توسط متد PutIndex آن می‌توان یک ایندکس جدید را تعریف کرد. این متد نیاز به نام ایندکس ایجاد شده و همچنین حداقل، متد Map آن‌را دارد. برای این منظور از شیء IndexDefinitionBuilder برای تعریف نحوه جمع آوری اطلاعات ایندکس کمک خواهیم گرفت. در اینجا خاصیت Map آن‌را باید توسط یک کوئری LINQ که فیلدهای مدنظر را بازگشت می‌دهد، مقدار دهی کنیم.

برنامه را اجرا کرده و سپس به کنسول مدیریتی تحت وب RavenDB، قسمت ایندکس‌های آن مراجعه کنید. در اینجا می‌توان ایندکس جدید ایجاد شده را مشاهده کرد:



هرچند همین اعمال را در کنسول مدیریتی نیز می‌توان انجام داد، اما مزیت آن در سمت کدها، دسترسی به intellisense و نوشتن کوئری‌های strongly typed است.

روش استفاده از store.DatabaseCommands.PutIndex اولین روش تولید Index در RavenDB با کدنویسی است. روش دوم، بر اساس ارث بری از کلاس AbstractIndexCreationTask شروع می‌شود و مناسب است برای حالتیکه نمی‌خواهید کدهای تولید ایندکس، با کدهای سایر قسمت‌های برنامه مخلوط شوند:

```
public class QuestionsByTitle : AbstractIndexCreationTask<Question>
    {
        public QuestionsByTitle()
        {
            Map = questions => questions.Select(question => new { Title = question.Title });
        }
    }
```

در اینجا با ایجاد یک کلاس جدید و ارث بری از کلاس AbstractIndexCreationTask کار شروع می‌شود. سپس در سازنده این کلاس، خاصیت Map را مقدار دهی می‌کنیم. مقدار آن نیز یک کوئری LINQ است که کار Select فیلدهای شرکت دهنده در کار تهیه ایندکس را انجام می‌دهد.
اکنون برای معرفی آن به برنامه باید از متد IndexCreation.CreateIndexes استفاده کرد. این متد، نیاز به دریافت اسمبلی محل تعریف کلاس‌های تولید ایندکس را دارد. به این ترتیب تمام کلاس‌های مشتق شده از AbstractIndexCreationTask را یافته و ایندکس‌های متناظری را تولید می‌کند.

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                IndexCreation.CreateIndexes(typeof(QuestionsByTitle).Assembly, store);
            }
```

این روش، قابلیت نگهداری و نظم بهتری دارد.

استفاده از ایندکس‌های ایستای ایجاد شده

تا اینجا موفق شدیم ایندکس‌های ایستای خود را با کد نویسی ایجاد کنیم. در ادامه قصد داریم از این ایندکس‌ها در کوئری‌های خود استفاده نمائیم.

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    var questions = session.Query<Question>(indexName: "QuestionsByTitle")
                                           .Where(x => x.Title.StartsWith("Raven")).Take(128);
                    foreach (var question in questions)
                    {
                        Console.WriteLine(question.Title);
                    }
                }
            }
```

استفاده از ایندکس تعریف شده نیز بسیار ساده می‌باشد. تنها کافی است نام آن‌را به متد Query ارسال نمائیم. اینبار اگر به خروجی کنسول سرور RavenDB دقت کنیم، از ایندکس indexes/QuestionsByTitle بجای ایندکس‌های پویا استفاده کرده است:

```
Request # 147: GET     -    58 ms - <system>   - 200 -
/indexes/QuestionsByTitle?&query=Title%3ARaven*&pageSize=128
```

```
        Query: Title:Raven*
        Time: 7 ms
        Index: QuestionsByTitle
        Results: 2 returned out of 2 total.
```

روش مشخص سازی نام ایندکس با استفاده از رشته‌ها، با هر دو روش store.DatabaseCommands.PutIndex و استفاده از AbstractIndexCreationTask سازگار است. اما اگر ایندکس‌های خود را با ارث بری از AbstractIndexCreationTask ایجاد کرده‌ایم، می‌توان نام کلاس مشتق شده را به صورت یک آرگومان جنریک دوم به متد Query به شکل زیر ارسال کرد تا از مزایای تعریف strongly typed آن نیز بهره‌مند شویم:

```
var questions = session.Query<Question, QuestionsByTitle>()
                                        .Where(x => x.Title.StartsWith("Raven")).Take(128);
```

ایجاد ایندکس‌های پیشرفته با پیاده سازی Map/Reduce

حالتی را در نظر بگیرید که در آن قصد داریم تعداد عنوان‌های سؤالات مانند هم را بیابیم (یا تعداد مطالب گروه‌های مختلف یک وبلاگ را محاسبه کنیم). برای انجام اینکار با سرعت بسیار بالا، می‌توانیم از ایندکس‌هایی با قابلیت محاسباتی در RavenDB استفاده کنیم. کار با ارث بری از کلاس AbstractIndexCreationTask شروع می‌شود. آرگومان جنریک اول آن، نام کلاسی است که در تهیه ایندکس شرکت خواهد داشت و آرگومان دوم (و اختیاری) ذکر شده، نتیجه عملیات Reduce است:

```
public class QuestionsCountByTitleReduceResult
    {
        public string Title { set; get; }
        public int Count { set; get; }
    }

    public class QuestionsCountByTitle : AbstractIndexCreationTask<Question,
QuestionsCountByTitleReduceResult>
    {
        public QuestionsCountByTitle()
        {
            Map = questions => questions.Select(question =>
                                                new
                                                {
                                                    Title = question.Title,
                                                    Count = 1
                                                });
            Reduce = results => results.GroupBy(x => x.Title)
                                       .Select(g =>
                                                new
                                                {
                                                    Title = g.Key,
                                                    Count = g.Sum(x => x.Count)
                                                });
        }
    }
```

در اینجا یک ایندکس پیشرفته را تعریف کرده‌ایم که در آن در قسمت Map، کار ایندکس کردن تک تک عنوان‌ها انجام خواهد شد. به همین جهت مقدار Count در این حالت، عدد یک است. در قسمت Reduce، بر روی نتیجه قسمت Map کوئری LINQ دیگری نوشته شده و تعداد عنوان‌های همانند، با گروه بندی اطلاعات، شمارش گردیده است.
اکنون برای استفاده از این ایندکس، ابتدا توسط متد IndexCreation.CreateIndexes، کار معرفی آن به RavenDB صورت گرفته و سپس متد Query سشن باز شده، دو آرگومان جنریگ را خواهد پذیرفت. اولین آرگومان، همان نتیجه Map/Reduce است و دومین آرگومان نام کلاس ایندکس جدید تعریف شده می‌باشد:

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                IndexCreation.CreateIndexes(typeof(QuestionsCountByTitle).Assembly, store);

                using (var session = store.OpenSession())
                {
```

```
                    var result = session.Query<QuestionsCountByTitleReduceResult,
QuestionsCountByTitle>()
                                        .FirstOrDefault(x => x.Title == "Raven") ?? new
QuestionsCountByTitleReduceResult();
                    Console.WriteLine(result.Count);
                }
            }
```

در کوئری فوق چون عملیات بر روی نتیجه نهایی باید صورت گیرد از FirstOrDefault استفاده شده است. این کوئری در حقیقت بر روی قسمت Reduce پیشتر محاسبه شده، اجرا می‌شود.