عنوان: **کوئری نویسی مقدماتی در RavenDB**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۱۵ ۰:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: NoSQL, RavenDB

با شروع کوئری نویسی مقدماتی در RavenDB، در قسمت اول این مباحث، توسط فراخوانی متد Load یک سشن، آشنا شدید. در ادامه مباحث تکمیلی آن‌را مرور خواهیم کرد.

امکان استفاده از LINQ در RavenDB

RavenDB از LINQ جهت کوئری نویسی پشتیبانی می‌کند. برای استفاده از آن، در ادامه مطلب اول، ابتدا سرور RavenDB را اجرا نموده و سپس برنامه کنسول را به نحو ذیل تغییر دهید:

```
using System;
using System.Linq;
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 questions = session.Query<Question>().Where(x => x.Title.StartsWith("Raven"));
                    foreach (var question in questions)
                    {
                        Console.WriteLine(question.Title);
                    }
                }
            }
        }
    }
}
```

در RavenDB برای دسترسی به امکانات LINQ، کار با متد Query یک سشن آغاز می‌شود و پس از آن، امکان استفاده از متدهای متداول LINQ مانند مثال فوق وجود خواهد داشت. البته بدیهی است مباحثی مانند JOIN و امثال آن در یک بانک اطلاعاتی NoSQL پشتیبانی نمی‌شود. ضمنا باید درنظر داشت که مبحث safe by default در اینجا نیز اعمال می‌شود. برای مثال اگر به کنسول سرور RavenDB که در حال اجرا است مراجعه کنید، یک چنین خروجی را حین اجرای مثال فوق می‌توان مشاهده کرد که در آن pageSize پیش فرضی اعمال شده است:

```
Available commands: cls, reset, gc, q
Request #   1: GET     -   179 ms - <system>   - 404 - /docs/Raven/Replication/Destinations
Request #   2: GET     - 3,818 ms - <system>   - 200 -
/indexes/dynamic/Questions?&query=Title%3ARaven*&pageSize=128
        Query: Title:Raven*
        Time: 3,494 ms
        Index: Auto/Questions/ByTitle
        Results: 2 returned out of 2 total.
```

یعنی در عمل کوئری‌را که اجرا کرده است، شبیه به کوئری ذیل می‌باشد و یک Take پیش فرض بر روی آن اعمال شده است:

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

علت این مساله نیز به تصمیم نویسنده اصلی آن بر می‌گردد؛ ایشان پیش از شروع به تهیه RavenDB، کار تهیه انواع و اقسام

پروفایلرهای مهم ORMهای معروف مانند NHibernate و Entity framework را انجام داده است و در این حین، یکی از مهم‌ترین مشکلاتی را که با آن‌ها در کدهای متداول برنامه نویس‌ها یافته است، unbounded queries است. کوئری‌هایی که حد و مرزی برای بازگشت اطلاعات قائل نمی‌شوند. داشتن این نوع کوئری‌ها با تعداد بالای کاربر، یعنی مصرف بیش از حد RAM بر روی سرور، به همراه بار پردازشی بیش از حد و غیر ضروری. چون عملا حتی اگر 10 هزار رکورد بازگشت داده شوند، عموم برنامه نویس‌ها حداکثر 100 رکورد آن‌را در یک صفحه نمایش می‌دهند و نه تمام رکوردها را.

ارتباط Lucene.NET و RavenDB

کل LINQ API تهیه شده در RavenDB یک محصور کننده امکانات Lucene.NET است. اگر پیشتر با Lucene.NET کار کرده باشید، در خروجی حالت دیباگ کنسول سرور فوق، سطر «Query: Title:Raven*» آشنا به نظر خواهد رسید. دقیقا کوئری LINQ نوشته شده به یک کوئری با Syntax مخصوص Lucene.NET ترجمه شده‌است. برای نمونه اگر علاقمند باشید که مستقیما کوئری‌های خاص لوسین را در RavenDB اجرا کنید، از Syntax ذیل می‌توان استفاده کرد:

```
var questions = session.Advanced.LuceneQuery<Question>().Where("Title:Raven*").ToList();
```

و یا اگر علاقمند به حفظ کردن Syntax خاص لوسین نیستید، یک سری متد الحاقی خاص نیز در اینجا برای LuceneQuery تدارک دیده شده است. برای مثال کوئری رشته‌ای فوق، معادل کوئری strongly typed ذیل است:

```
var questions = session.Advanced.LuceneQuery<Question>().WhereStartsWith(x => x.Title,
"Raven").ToList();
```

استفاده مجدد از کوئری‌ها در RavenDB

در RavenDB، متد Query به صورت immutable تعریف شده است و متد LuceneQuery حالت mutable دارد (ترکیبات آن نیز یک وهله است).
یک مثال:

```
var query = session.Query<User>().Where(x => x.Name.StartsWith("A"));
var ageQuery = query.Where(x => x.Age > 21);
var eyeQuery = query.Where(x => x.EyeColor == "blue");
```

در اینجا از کوئری ابتدایی، در دو کوئری مجزا استفاده مجدد شده است. ترجمه خروجی سه کوئری فوق به نحو زیر است:

```
query - Name:A*
ageQuery - (Name:A*) AND (Age_Range:{Ix21 TO NULL})
eyeQuery - (Name:A*) AND (EyeColor:blue)
```

به این معنا که زمانیکه به eyeQuery رسیدیم، نتیجه ageQuery با آن ترکیب نمی‌شود؛ چون متد Query از نوع immutable است.
در ادامه اگر همین سه کوئری فوق را با فرمت LuceneQuery تهیه کنیم، به عبارات ذیل خواهیم رسید:

```
var luceneQuery = session.Advanced.LuceneQuery<User>().WhereStartsWith(x => x.Name, "A");
var ageLuceneQuery = luceneQuery.WhereGreaterThan(x => x.Age, 21);
var eyeLuceneQuery = luceneQuery.WhereEquals(x => x.EyeColor, "blue");
```

در خروجی‌های این سه کوئری، مورد سوم مهم است:

```
luceneQuery - Name:A*
ageLuceneQuery - Name:A* Age_Range:{Ix21 TO NULL}
eyeLuceneQuery - Name:A* Age_Range:{Ix21 TO NULL} EyeColor:blue
```

همانطور که مشاهده می‌کنید، کوئری سوم، عبارت کوئری دوم را نیز به همراه دارد؛ این مورد دقیقا مفهوم اشیاء mutable یا تک

وهله‌ای است مانند LuceneQuery در اینجا.

And و Or شدن کوئری‌های ترکیبی در RavenDB

در مثال استفاده مجدد از کوئری‌ها، زمانیکه از Where استفاده شد، بین عبارات حاصل AND قرار گرفته است. این مورد را به نحو ذیل می‌توان تنظیم کرد و مثلا به OR تغییر داد:

```
session.Advanced.LuceneQuery<User>().UsingDefaultOperator(QueryOperator.And);
```

صفحه بندی اطلاعات در RavenDB

در ابتدای بحث عنوان شد که کوئری LINQ اجرا شده در RavenDB، یک Take مخفی و پیش فرض تنظیم شده به 128 آیتم را دارد. اکنون سؤال این خواهد بود که چگونه می‌توان اطلاعات را به صورت صفحه بندی شده، بر اساس شماره صفحه خاصی نمایش داد.

```
using System;
using System.Linq;
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())
                {
                    int pageNumber = 0;
                    int resultsPerPage = 2;

                    var questions = session.Query<Question>()
                                           .Where(x => x.Title.StartsWith("Raven"))
                                           .Skip(pageNumber * resultsPerPage)
                                           .Take(resultsPerPage);
                    foreach (var question in questions)
                    {
                        Console.WriteLine(question.Title);
                    }
                }
            }
        }
    }
}
```

برای انجام صفحه بندی در RavenDB، کافی است از متدهای Skip و Take بر اساس محاسباتی که مشاهده می‌کنید، استفاده گردد.

دریافت اطلاعات آماری کوئری اجرا شده

در RavenDB امکان دریافت یک سری اطلاعات آماری از کوئری اجرا شده نیز وجود دارد؛ برای مثال یک کوئری چند ثانیه طول کشیده است، چه تعدادی رکورد را بازگشت داده است و امثال آن. برای پیاده سازی آن، نیاز است از متد الحاقی Statistics به نحو ذیل استفاده کرد:

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

namespace RavenDBSample01
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    int pageNumber = 0;
                    int resultsPerPage = 2;
                    RavenQueryStatistics stats;
                    var questions = session.Query<Question>()
                                           .Statistics(out stats)
                                           .Where(x => x.Title.StartsWith("Raven"))
                                           .Skip(pageNumber * resultsPerPage)
                                           .Take(resultsPerPage);
                    foreach (var question in questions)
                    {
                        Console.WriteLine(question.Title);
                    }

                    Console.WriteLine("TotalResults: {0}", stats.TotalResults);
                }
            }
        }
    }
}
```

متد الحاقی Statistics پس از متد Query که نقطه آغازین نوشتن کوئری‌های LINQ است، فراخوانی شده و یک پارامتر out از نوع RavenQueryStatistics تعریف شده در فضای نام Raven.Client را دریافت می‌کند. پس از پایان کوئری می‌توان از این خروجی جهت نمایش اطلاعات آماری کوئری استفاده کرد.

امکانات ویژه فضای نام Raven.Client.Linq

یک سری متد الحاقی خاص جهت تهیه ساده‌تر کوئری‌های LINQ در فضای نام Raven.Client.Linq قرار دارند که پس از تعریف آن قابل دسترسی خواهند بود:

```
var list = session.Query<Question>().Where(q => q.By.In<string>(arrayOfUsers))).ToArray()
```

برای مثال در اینجا متد الحاقی جدید In را مشاهده می‌کنید که شبیه به کوئری SQL ذیل در دنیای بانک‌های اطلاعاتی رابطه‌ای عمل می‌کند:

```
SELECT * FROM tbl WHERE data IN (1, 2, 3)
```

اتصال به RavenDB با استفاده از برنامه معروف LINQPad

اگر علاقمند باشید که کوئری‌های خود را در محیط برنامه معروف LINQPad نیز آزمایش کنید، درایور مخصوص RavenDB آن‌را از آدرس ذیل می‌توانید دریافت نمائید:

https://github.com/ronnieoverby/RavenDB-Linqpad-Driver