عنوان: **ایجاد ایندکس منحصربفرد در EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۴ ۲۲:۱۳
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در EF Code first برای ایجاد UNIQUE INDEX ویژگی یا تنظیمات Fluent API خاصی درنظر گرفته نشده است و می‌توان از همان روش‌های متداول اجرای مستقیم کوئری SQL بر روی بانک اطلاعاتی جهت ایجاد UNIQUE INDEXها کمک گرفت:

```
public static void CreateUniqueIndex(this DbContext context, string tableName, string fieldName)
{
      context.Database.ExecuteSqlCommand("CREATE UNIQUE INDEX [IX_Unique_" + tableName
+ "_" + fieldName + "] ON [" + tableName + "]([" + fieldName + "] ASC);");
}
```

و این کل کاری است که باید در متد Seed کلاس مشتق شده از DbMigrationsConfiguration انجام شود. مثلا:

```
public class MyDbMigrationsConfiguration : DbMigrationsConfiguration<MyContext>
{
        public BlogDbMigrationsConfiguration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            CreateUniqueIndex(context, "table1", "field1");
            base.Seed(context);
        }
}
```

روش فوق کار می‌کند اما آنچنان مناسب نیست؛ چون به صورت strongly typed تعریف نشده است و اگر نام جداول یا فیلدها را بعدها تغییر دادیم، به مشکل برخواهیم خورد و کامپایلر خطایی را صادر نخواهد کرد؛ چون table1 و field1 در اینجا به صورت رشته تعریف شده‌اند.

برای حل این مشکل و تبدیل کدهای فوق به کدهایی Refactoring friendly، نیاز است نام جدول به صورت خودکار از DbContext دریافت شود. همچنین نام خاصیت یا فیلد نیز به صورت strongly typed قابل تعریف باشد.

کدهای کامل نمونه بهبود یافته را در ذیل مشاهده می‌کنید:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Objects;
using System.Linq;
using System.Linq.Expressions;
using System.Text.RegularExpressions;

namespace General
{
    public static class ContextExtensions
    {
        public static string GetTableName<T>(this DbContext context) where T : class
        {
            ObjectContext objectContext = ((IObjectContextAdapter)context).ObjectContext;
            return objectContext.GetTableName<T>();
        }

        public static string GetTableName<T>(this ObjectContext context) where T : class
        {
            var sql = context.CreateObjectSet<T>().ToTraceString();
            var regex = new Regex("FROM (?<table>.*) AS");
            var match = regex.Match(sql);
            string table = match.Groups["table"].Value;
            return table
                    .Replace("`", string.Empty)
                    .Replace("[", string.Empty)
                    .Replace("]", string.Empty)
                    .Replace("dbo.", string.Empty)
```

```
                    .Trim();
        }

        private static bool hasUniqueIndex(this DbContext context, string tableName, string indexName)
        {
            var sql = "SELECT count(*) FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS where table_name = '"
                      + tableName + "' and CONSTRAINT_NAME = '" + indexName + "'";
            var result = context.Database.SqlQuery<int>(sql).FirstOrDefault();
            return result > 0;
        }

        private static void createUniqueIndex(this DbContext context, string tableName, string
fieldName)
        {
            var indexName = "IX_Unique_" + tableName + "_" + fieldName;
            if (hasUniqueIndex(context, tableName, indexName))
                return;

            var sql = "ALTER TABLE [" + tableName + "] ADD CONSTRAINT [" + indexName + "] UNIQUE ([" +
fieldName + "])";
            context.Database.ExecuteSqlCommand(sql);
        }

        public static void CreateUniqueIndex<TEntity>(this DbContext context, Expression<Func<TEntity,
object>> fieldName) where TEntity : class
        {
            var field = ((MemberExpression)fieldName.Body).Member.Name;
            createUniqueIndex(context, context.GetTableName<TEntity>(), field);
        }
    }
}
```

توضیحات

متد GetTableName ، به کمک SQL تولیدی حین تعریف جدول متناظر با کلاس جاری، نام جدول را با استفاده از عبارات باقاعده جدا کرده و باز می‌گرداند. به این ترتیب به دقیق‌ترین نامی که واقعا جهت تولید جدول مورد استفاده قرار گرفته است خواهیم رسید.
در مرحله بعد آن، همان متد createUniqueIndex ابتدای بحث را ملاحظه می‌کنید. در اینجا جهت حفظ سازگاری بین SQL Server کامل و SQL CE از UNIQUE CONSTRAINT استفاده شده است که همان کار ایجاد ایندکس منحصربفرد را نیز انجام می‌دهد. به علاوه مزیت دیگر آن امکان دسترسی به تعاریف قید اضافه شده توسط view ایی به نام INFORMATION_SCHEMA.TABLE_CONSTRAINTS است که در نگارش‌های مختلف SQL Server به یک نحو تعریف گردیده و قابل دسترسی است. از این view در متد hasUniqueIndex جهت بررسی تکراری نبودن UNIQUE CONSTRAINT در حال تعریف، استفاده می‌شود. اگر این قید پیشتر تعریف شده باشد، دیگر سعی در تعریف مجدد آن نخواهد شد.
متد CreateUniqueIndex تعریف شده در انتهای کلاس فوق، امکان دریافت نام خاصیتی از TEntity را به صورت strongly typed میسر می‌سازد.
اینبار برای تعریف یک قید و یا ایندکس منحصربفرد بر روی خاصیتی مشخص در متد Seed، تنها کافی است بنویسیم:

```
context.CreateUniqueIndex<User>(x=>x.Name);
```

در اینجا دیگر از رشته‌ها خبری نبوده و حاصل نهایی Refactoring friendly است. به علاوه نام جدول را نیز به صورت خودکار از context استخراج می‌کند.

نظرات خوانندگان

نویسنده: مرادی
تاریخ: ۱۳۹۱/۰۶/۱۴ ۲۲:۳۶

دو نکته
اول این که NHibernate پشتیبانی تو کار و کاملی از این آیتم داره
دو این که Metadata Workspace همه اطلاعات لازم Mapping رو به شما می‌داد و کد متد GetTableName از نظر من جیمز باندیه !
موفق باشید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۷ ۲۳:۱۳

یک نکته‌ی تکمیلی

از EF 6.1 به بعد ، دیگر نیازی به این مطلب نیست. تعریف ایندکس به صورت توکار میسر شده‌است .