عنوان: **ایندکس منحصر به فرد با استفاده از Data Annotation در EF Code First**
نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۰۹/۲۲ ۲۳:۲۸
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Data Annotation, Unique, Index

در حال حاضر امکان خاصی برای ایجاد ایندکس منحصر به فرد در EF First Code وجود ندارد, برای این کار راه‌های زیادی وجود دارد مانند پست قبلی آقای نصیری, در این آموزش از Data Annotation و یا همان Attribute هایی که بالای Propertyهای مدل‌ها قرار می‌دهیم, مانند کد زیر :

```
public class User
    {
        public int Id { get; set; }

        [Unique]
        public string Email { get; set; }

        [Unique("MyUniqueIndex",UniqueIndexOrder.ASC)]
        public string Username { get; set; }

        [Unique(UniqueIndexOrder.DESC)]
        public string PersonalCode{ get; set; }

        public string Password { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
```

همانطور که در کد بالا می‌بینید با استفاده از Attribute Unique ایندکس منحصر به فرد آن در دیتابیس ساخته خواهد شد.
ابتدا یک کلاس برای Attribute Unique به صورت زیر ایجاد کنید :

```
using System;

namespace SampleUniqueIndex
{
    [AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple = false)]
    public class UniqueAttribute : Attribute
    {
        public UniqueAttribute(UniqueIndexOrder order = UniqueIndexOrder.ASC) {
            Order = order;
        }
        public UniqueAttribute(string indexName,UniqueIndexOrder order = UniqueIndexOrder.ASC)
        {
            IndexName = indexName;
            Order = order;
        }
        public string IndexName { get; private set; }
        public UniqueIndexOrder Order { get; set; }
    }

    public enum UniqueIndexOrder
    {
        ASC,
        DESC
    }
}
```

در کد بالا یک Enum برای مرتب سازی ایندکس به دو صورت صعودی و نزولی قرار دارد, همانند کد ابتدای آموزش که مشاهده می‌کنید امکان تعریف این Attribute به سه صورت امکان دارد که به صورت زیر می‌باشد:
1. ایجاد Attribute بدون هیچ پارامتری که در این صورت نام ایندکس با استفاده از نام جدول و آن فیلد ساخته خواهد شد : IX_Unique_TableName_FieldName و مرتب ساری آن به صورت صعودی می‌باشد.
2.نامی برای ایندکس انتخاب کنید تا با آن نام در دیتابیس ذخیره شود, در این حالت مرتب سازی آن هم به صورت صعودی می‌باشد.
3. در حالت سوم شما ضمن وارد کردن نام ایندکس مرتب سازی آن را نیز وارد می‌کنید.

بعد از کلاس Attribute حالا نوبت به کلاس اصلی میرسد که به صورت زیر می‌باشد:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Metadata.Edm;
using System.Linq;
using System.Reflection;

namespace SampleUniqueIndex
{
    public static class DbContextExtention
    {
        private static BindingFlags PublicInstance = BindingFlags.Public | BindingFlags.Instance |
BindingFlags.FlattenHierarchy;

        public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            var query = "";
            foreach (var dbSet in GetDbSets(context))
            {
                var entityType = dbSet.PropertyType.GetGenericArguments().First();
                var table = tables[entityType.Name];
                var currentIndexes = GetCurrentUniqueIndexes(context, table.TableName);
                foreach (var uniqueProp in GetUniqueProperties(context, entityType, table))
                {
                    var indexName = string.IsNullOrWhiteSpace(uniqueProp.IndexName) ?
                        "IX_Unique_" + uniqueProp.TableName + "_" + uniqueProp.FieldName :
                        uniqueProp.IndexName;

                    if (!currentIndexes.Contains(indexName))
                    {
                        query += "ALTER TABLE [" + table.TableSchema + "].[" + table.TableName + "] ADD
CONSTRAINT [" + indexName + "] UNIQUE ([" + uniqueProp.FieldName + "] " + uniqueProp.Order + "); ";
                    }
                    else
                    {
                        currentIndexes.Remove(indexName);
                    }
                }
                foreach (var index in currentIndexes)
                {
                    query += "ALTER TABLE [" + table.TableSchema + "].[" + table.TableName + "] DROP
CONSTRAINT " + index + "; ";
                }
            }

            if (query.Length > 0)
                context.Database.ExecuteSqlCommand(query);
        }

        private static List<string> GetCurrentUniqueIndexes(DbContext context, string tableName)
        {
            var sql = "SELECT CONSTRAINT_NAME FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS where
table_name = '"
                      + tableName + "' and CONSTRAINT_TYPE = 'UNIQUE'";
            var result = context.Database.SqlQuery<string>(sql).ToList();
            return result;
        }
        private static IEnumerable<PropertyDescriptor> GetDbSets(DbContext context)
        {
            foreach (PropertyDescriptor prop in TypeDescriptor.GetProperties(context))
            {
                var notMapped = prop.GetType().GetCustomAttributes(typeof(NotMappedAttribute),true);
                if (prop.PropertyType.Name == typeof(DbSet<>).Name && notMapped.Length == 0)
                    yield return prop;
            }
        }
        private static List<UniqueProperty> GetUniqueProperties(DbContext context, Type entity,
TableInfo tableInfo)
        {
            var indexedProperties = new List<UniqueProperty>();
            var properties = entity.GetProperties(PublicInstance);
            var tableName = tableInfo.TableName;
            foreach (var prop in properties)
            {
```

```
                if (!prop.PropertyType.IsValueType && prop.PropertyType != typeof(string)) continue;

                UniqueAttribute[] uniqueAttributes =
(UniqueAttribute[])prop.GetCustomAttributes(typeof(UniqueAttribute), true);
                NotMappedAttribute[] notMappedAttributes =
(NotMappedAttribute[])prop.GetCustomAttributes(typeof(NotMappedAttribute), true);
                if (uniqueAttributes.Length > 0 && notMappedAttributes.Length == 0)
                {
                    var fieldName = GetFieldName(context, entity, prop.Name);
                    if (fieldName != null)
                    {
                        indexedProperties.Add(new UniqueProperty
                        {
                            TableName = tableName,
                            IndexName = uniqueAttributes[0].IndexName,
                            FieldName = fieldName,
                            Order = uniqueAttributes[0].Order.ToString()
                        });
                    }
                }
            }
            return indexedProperties;
        }
        private static Dictionary<string, TableInfo> GetTables(DbContext context)
        {
            var tablesInfo = new Dictionary<string, TableInfo>();
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var tables = metadata.GetItemCollection(DataSpace.SSpace)
              .GetItems<EntityContainer>()
              .Single()
              .BaseEntitySets
              .OfType<EntitySet>()
              .Where(s => !s.MetadataProperties.Contains("Type")
                || s.MetadataProperties["Type"].ToString() == "Tables");
            foreach (var table in tables)
            {
                var tableName = table.MetadataProperties.Contains("Table")
                    && table.MetadataProperties["Table"].Value != null
                  ? table.MetadataProperties["Table"].Value.ToString()
                  : table.Name;
                var tableSchema = table.MetadataProperties["Schema"].Value.ToString();
                tablesInfo.Add(table.Name, new TableInfo
                {
                    EntityName = table.Name,
                    TableName = tableName,
                    TableSchema = tableSchema,
                });
            }

            return tablesInfo;
        }
        public static string GetFieldName(DbContext context, Type entityModel, string propertyName)
        {
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var osMembers = metadata.GetItem<EntityType>(entityModel.FullName,
DataSpace.OSpace).Properties;
            var ssMebers = metadata.GetItem<EntityType>("CodeFirstDatabaseSchema." + entityModel.Name,
DataSpace.SSpace).Properties;

            if (!osMembers.Contains(propertyName)) return null;

            var index = osMembers.IndexOf(osMembers[propertyName]);
            return ssMebers[index].Name;
        }

        internal class UniqueProperty
        {
            public string TableName { get; set; }
            public string FieldName { get; set; }
            public string IndexName { get; set; }
            public string Order { get; set; }
        }
        internal class TableInfo
        {
            public string EntityName { get; set; }
            public string TableName { get; set; }
            public string TableSchema { get; set; }
        }
    }
}
```

در کد بالا با استفاده از Extension Method برای کلاس DbContext یک متد با نام ExecuteUniqueIndexes ایجاد می‌کنیم تا برای ایجاد ایندکس‌ها در دیتابیس از آن استفاده کنیم.
روند اجرای کلاس بالا به صورت زیر می‌باشد:
در ابتدای متد ExecuteUniqueIndexes() :

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            ...
        }
```

با استفاده از متد GetTables() ما تمام جداول ساخته توسط دیتایس توسط DbContext را گرفته:

```
private static Dictionary<string, TableInfo> GetTables(DbContext context)
        {
            var tablesInfo = new Dictionary<string, TableInfo>();
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var tables = metadata.GetItemCollection(DataSpace.SSpace)
              .GetItems<EntityContainer>()
              .Single()
              .BaseEntitySets
              .OfType<EntitySet>()
              .Where(s => !s.MetadataProperties.Contains("Type")
                || s.MetadataProperties["Type"].ToString() == "Tables");
            foreach (var table in tables)
            {
                var tableName = table.MetadataProperties.Contains("Table")
                    && table.MetadataProperties["Table"].Value != null
                  ? table.MetadataProperties["Table"].Value.ToString()
                  : table.Name;
                var tableSchema = table.MetadataProperties["Schema"].Value.ToString();
                tablesInfo.Add(table.Name, new TableInfo
                {
                    EntityName = table.Name,
                    TableName = tableName,
                    TableSchema = tableSchema,
                });
            }

            return tablesInfo;
        }
```

با استفاده از این طریق چنانچه کاربر نام دیگری برای هر جدول در نظر بگیرد مشکلی ایجاد نمی‌شود و همینطور Schema جدول نیز گرفته می‌شود, سه مشخصه نام مدل و نام جدول و Schema جدول در کلاس TableInfo قرار داده می‌شود و در انتها تمام جداول در یک Collection قرار داده میشوند و به عنوان خروجی متد استفاده می‌شوند.
بعد از آنکه نام جداول متناظر با نام مدل آنها را در اختیار داریم نوبت به گرفتن تمام DbSetها در DbContext می‌باشد که با استفاده از متد GetDbSets() :

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            var query = "";
            foreach (var dbSet in GetDbSets(context))
            {
            ....
        }
```

در این متد چنانچه Property دارای Attribute NotMapped باشد در لیست خروجی متد قرار داده **نمی‌شود**.
سپس داخل چرخه DbSetها نوبت به گرفتن ایندکس‌های موجود با استفاده از متد GetCurrentUniqueIndexes() برای این مدل می‌باشد تا از ایجاد دوباره آن جلوگیری شود و البته اگر ایندکس هایی را در مدل تعریف نکرده باشید از دیتابیس حذف شوند.

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
```

```
            ...
            foreach (var dbSet in GetDbSets(context))
            {
                var entityType = dbSet.PropertyType.GetGenericArguments().First();
                var table = tables[entityType.Name];
                var currentIndexes = GetCurrentUniqueIndexes(context, table.TableName);
            }
        }
```

بعد از آن نوبت به گرفتن Propertyهای دارای Attribute Unique می‌باشد که این کار نیز با استفاده از متد GetUniqueProperties() انجام خواهد شد.

در متد GetUniqueProperties() چند شرط بررسی خواهد شد از جمله اینکه Property از نوع Value Type باشد و نه یک کلاس سپس Attribute NotMapped را نداشته باشد و بعد از آن می‌بایست نام متناظر با آن Property را در دیتابیس به دست بیاریم برای این کار از متد GetFieldName() استفاده می‌کنیم:

```
public static string GetFieldName(DbContext context, Type entityModel, string propertyName)
        {
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var osMembers = metadata.GetItem<EntityType>(entityModel.FullName,
DataSpace.OSpace).Properties;
            var ssMebers = metadata.GetItem<EntityType>("CodeFirstDatabaseSchema." + entityModel.Name,
DataSpace.SSpace).Properties;

            if (!osMembers.Contains(propertyName)) return null;

            var index = osMembers.IndexOf(osMembers[propertyName]);
            return ssMebers[index].Name;
        }
```

برای این کار با استفاده از MetadataWorkspace در DbContext دو لیست SSpace و OSpace استفاده می‌کنیم که در ادامه در مورد این گونه لیست ها بیشتر توضیح می‌دهیم , سپس با استفاده از Memberهای این دو لیست و ایندکس‌های متناظر در این دو لیست نام متناظر با Property را در دیتابیس پیدا خواهیم کرد, **البته یک نکته مهم هست چنانچه برای فیلدهای دیتابیس OrderColumn قرار داده باشید دو لیست Memberها از نظر ایندکس متناظر متفاوت خواهند شد پس در نتیجه ایندکس به اشتباه برروی یک فیلد دیگر اعمال خواهد شد.**

لیست‌ها در MetadataWorkspace:

1. CSpace : این لیست شامل آبجکت‌های Conceptual از مدل‌های شما می‌باشد تا برای Mapping دیتابیس با مدل‌های شما مانند مبدلی این بین عمل کند.
2. OSpace : این لیست شامل آبجکت‌های مدل‌های شما می‌باشد.
3. SSpace : این لیست نیز شامل آبجکت‌های مربوط به دیتابیس از مدل‌های شما می‌باشد
4. CSSpace : این لیست شامل تنظیمات Mapping بین دو لیست SSpace و CSpace می‌باشد.
5. OCSpace : این لیست شامل تنظیمات Mapping بین دو لیست OSpace و CSpace می‌باشد.

روند Mapping مدل‌های شما از OSpace شروع شده و به SSpace ختم میشود که سه لیست دیگز شامل تنظیماتی برای این کار می‌باشند.

و حالا در متد اصلی ExecuteUniqueIndexes() ما کوئری مورد نیاز برای ساخت ایندکس‌ها را ساخته ایم.

حال برای استفاده از متدExecuteUniqueIndexes() می‌بایست در متد Seed آن را صدا بزنیم تا کار ساخت ایندکس‌ها شروع شود، مانند کد زیر:

```
protected override void Seed(myDbContext context)
        {
            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method
            //  to avoid creating duplicate seed data. E.g.
            //
            //    context.People.AddOrUpdate(
            //      p => p.FullName,
            //      new Person { FullName = "Andrew Peters" },
            //      new Person { FullName = "Brice Lambson" },
```

```
            //      new Person { FullName = "Rowan Miller" }
            //    );
            //
            context.ExecuteUniqueIndexes();
        }
```

**چند نکته برای ایجاد ایندکس منحصر به فرد وجود دارد که در زیر به آنها اشاره می‌کنیم:**

**1. فیلدهای متنی باید حداکثر تا 350 کاراکتر باشند تا ایندکس اعمال شود.**

**2. همانطور که بالاتر اشاره شد برای فیلدهای دیتابیس OrderColumn اعمال نکنید که علت آن در بالا توضیح داده شد**

**دانلود فایل پروژه:**

Sample_UniqueIndex.zip

## نظرات خوانندگان

نویسنده: rahimi
تاریخ: ۱۳۹۱/۰۹/۲۳ ۱۶:۳۲

سلام ممنون از آموزش‌های خوبتون
می‌خواستم خواهش کنم ازتون مثال هایی که توضیح دادید را فایلشو هم قرار بدید تا بتونیم استفاده کنیم
ممنون

نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۰۹/۲۴ ۹:۲۷

سلام
خواهش می‌کنم
فایل پروژه به انتهای آموزش اضافه شد.

نویسنده: آرش مصیر
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۶:۶

با تشکر از سایت خوبتون من چند ماه پیش به این مشکل بر خورده بودم و در متد Seed مربوط به Context مستقیما اسکریپت ساخت ایندکس رو گذاشته بودم حالا می‌خوام از روشی که گفتید استفاده کنم

نویسنده: اکبر
تاریخ: ۱۳۹۲/۰۷/۱۲ ۲۱:۱۰

با سلام.
وقتی از این اتریبیوت بر روی پراپرتی email استفاده میکنم، و چون مقدار این فیلد الزامی نیست، وقتی کاربر این فیلد را خالی بگذارد خطای زیر را دریافت میکنم.

```
Violation of UNIQUE KEY constraint 'IX_Unique_Members_Email'. Cannot insert duplicate key in object
'dbo.Members'
```

با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۱۲ ۲۱:۲۸

از چه دیتابیسی استفاده می‌کنید؟ اگر SQL Server است که تا قبل از نگارش 2008 آن چنین اجازه‌ای رو به شما نمی‌ده تا یک فیلد منحصربفرد نال پذیر داشته باشید. اگر 2008 به بعد است، باید ایندکس فیلتر شده برای اینکار تعریف کنید. مثلا:

```
create unique nonclustered index idx on dbo.DimCustomer(emailAddress)
where EmailAddress is not null;
```

اطلاعات بیشتر اینجا و اینجا
بر همین مبنا باید قسمت ADD CONSTRAINT متد ExecuteUniqueIndexes را در صورت نیاز بازنویسی کنید.

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 .