عنوان: **کنترل نوع‌های داده با استفاده از EF در SQL Server**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۰۴ ۱۸:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, SQL Server, database design, Performance

ورود سیستم‌های ORM مانند EF تحولی عظیم در در مباحث کار و تغییرات بر روی داده‌ها یا Data Manipulation بود. به طور خلاصه اصلی‌ترین هدف یک ORM، ایجاد فرامین شیء گرا به جای فرامین رابطه‌ای است؛ ولی در این بین نکات دیگری هم مد نظر گرفته شده‌است که یکی از آن‌ها پشتیبانی از چندین دیتابیس هست تا توسعه گران از یک سیستم واحد جهت اتصال به همه‌ی دیتابیس‌ها استفاده کنند و نیازی به دانش اضافه و سیستم جداگانه‌ای برای هر دیتابیس نباشد؛ مانند ADO که در دات نت به چندین دسته نقسیم شده بود و هم اینکه در صورتی که تمایلی به تغییر دیتابیس در آینده داشتید، کدها برای توسعه باز باشند و نیازی به تغییر سیستم نباشد.

ولی اگر کمی بیشتر به دنیای واقعی وارد شویم گاهی اوقات نیاز است که تنها بر روی یک دیتابیس فعالیت کنیم و یک دیتابیس نیاز است تا حد ممکن بهینه طراحی شود تا کارآیی بانک در حال حاضر و به خصوص در آینده تا حدی تضمین شود.
من همیشه در مورد EF یک نظری داشتم و آن اینست که با اینکه یک ORM، یک هدف مهم را در نظر دارد و آن اینست که تا حد ممکن استانداردهایی را که بین تمامی دیتابیس‌ها مشترک است، رعایت کند، ولی باز قابل قبول است اگر بگوییم که کاربران EF انتظار داشته باشند تا اطلاعات بیشتری در مورد sql server در آن نهفته باشد. از یک سو هر دو محصول مایکروسافت هستند و از سوی دیگر مطمئنا توسعه گران محصولات دات نت بیش از هر چیزی به sql server نگاه ویژه‌تری دارند. پس مایکروسافت در کنار حفظ آن ویژگی‌های مشترک، باید به حفظ استانداردهای جدایی برای sql server هم باشد.

تعدادی از برنامه نویسان در هنگام ایجاد Domain Model کم لطفی‌های زیادی را می‌کنند که یکی از آن‌ها عدم کنترل نوع داده‌های خود است. مثلا برای رشته‌ها هیچ محدودیتی را در نظر نمی‌گیرند. شاید در سمت کلاینت اینکار را انجام می‌دهند؛ ولی نکته‌ی مهم در طرف دیتابیس است که چگونه تعریف می‌شود. در این حالت nvarchar(MAX) در نظر گرفته میشود که به معنی اشاره به منطقه دوگیگابایتی از اطلاعات است. در نکات بعدی، قصد داریم این مرحله را یک گام به جلوتر پیش ببریم و آن هم ایجاد نوع داده‌های بهینه‌تر در Sql Server است.

نکته مهم: بدیهی است که تغییرات زیر، ORM شما را تنها به sql server مقید می‌کند که بعدها در صورت تغییر دیتابیس نیاز به حذف موارد زیر را خواهید داشت؛ در غیر اینصورت به مشکل عدم ایجاد دیتابیس برخواهید خورد.

اولین مورد مهم بحث تاریخ و زمان است ؛ وقتی ما یک نوع داده را تنها DateTime در نظر بگیریم، در Sql Server هم همین نوع داده وجود دارد و انتخاب میشود. ولی اگر شما واقعا نیازی به این نوع داده نداشته باشید چطور؟ در حال حاضر من بر روی یک برنامه‌ی کارخانه کار میکنم که بخش کارمندان و گارگران آن سه داده زمانی زیر را شامل می‌شود:

```
public DateTime BirthDate { get; set; }
        public DateTime HireDate { get; set; }
        public DateTime? LeaveDate { get; set; }
```

حال به جدول زیر نگاه کنید که هر نوع داده چه مقدار فضا را به خود اختصاص می‌دهد:

SmallDateTime	4 بایت
DateTime	8 بایت
DateTime2	6 تا 8 بایت
DateTimeOffset	8 تا 10 بایت
Date	3 بایت

SmallDateTime	4 بایت
Time	3 تا 5 بایت

از این جدول چه می‌فهمید؟ با یک نگاه می‌توان فهمید که ساختار بالای من باید 24 بایت گرفته باشد؛ برای ساختاری که هم تاریخ و هم زمان (ساعت) را پشتیبانی می‌کند. ولی با نگاه دقیق‌تر به نام پراپرتی‌ها این نکته روشن می‌شود که ما یک گپ Gap (فضای بیهوده) داریم چون زمان تولد، استخدام و ترک سازمان اصلا نیازی به ساعت ندارند و همان تاریخ کافی است. یعنی نوع Date با حجم کلی 9 بایت؛ که در نتیجه 15 بایت صرفه جویی در یک رکورد صورت خواهد گرفت.

پس کد بالا را به شکل زیر تغییر می‌دهم:

```
[Column(TypeName = "date")]
        public DateTime BirthDate { get; set; }

        [Column(TypeName = "date")]
        public DateTime HireDate { get; set; }

        [Column(TypeName = "date")]
        public DateTime? LeaveDate { get; set; }
```

خصوصیت Column از نسخه 4.5دات نت فریم ورک اضافه شده و در فضای نام System.ComponentModel.DataAnnotations.Schema قرار گرفته است.
نوع‌هایی که در بالا با سایز متغیر هستند، به نسبت دقتی که برای آن تعیین می‌کنید، سایز می‌گیرند. مثل time(0) که 3 بایت از حافظه را می‌گیرد. در صورتی که time معرفی کنید، به جای اینکه از شیء DateTime استفاده کنید، از شی Timespan استفاده کنید، تا در پشت صحنه از نوع داده time استفاده کند. در این حالت حداکثر حافظه یعنی 5 بایت را برخواهد داشت و بهترین حالت ممکن این هست که نیاز خود را بسنجید و خودتان دقت آن را مشخص کنید. دو شکل زیر نحوه‌ی تعریف نوع زمان را مشخص می‌کنند. یکی حالت پیش فرض و دیگری انتخاب دقت:

```
public class Testtypes
    {
           public TimeSpan CloseTime { get; set; }
            public TimeSpan CloseTime2 { get; set; }
    }
   public class TestConfig : EntityTypeConfiguration<Testtypes>
    {
        public TestConfig()
        {
            this.Property(x => x.CloseTime2).HasPrecision(3);
        }
    }
```

در تکه کد بالا همه از نوع time تعریف می‌شوند ولی در خصوصیت شماره یک نهایت استفاده از نوع تایم یعنی time(7) مشخص می‌شود. ولی در خصوصیت بعدی چون در کانفیگ دقت آن را مشخص کرده‌ایم از نوع time(3) تعریف می‌شود.

مورد دوم در مورد داده‌های اعشاری است:

بسیاری از برنامه نویسان تا آنجا که دیده‌ام از نوع float و single و double برای اعداد اعشاری استفاده می‌کنند ولی باید دید که در آن سمت دیتابیس، برای این نوع داده‌ها چه اتفاقی می‌افتد. نوع float در دات نت، با نوع single برابری میکند؛ هر دو یک نام جدا دارند، ولی در واقع یکی هستند. عموما برنامه نویسان به طور کلی بیشتر از همان single استفاده می‌کنند و برای انتساب برای این دو نوع هم حتما باید حرف f را بعد از عدد نوشت:

```
float flExample=23.2f;
```

باید توجه کنید که اگر مثلا float انتخاب کردید، تصور نکنید که همان float در دیتابیس خواهد بود. این دو متفاوت هستند تبدیلات به شکل زیر رخ می‌دهد:

```
//real
```

```
public float FloatData { get; set; }
//real
public Single SingleData { get; set; }
//float
public double DoubleData { get; set; }
```

همه نوع‌های بالا اعداد اعشاری هستند که به صورت تقریبی و به صورت نماد اعشاری ذخیره می‌گردند و برای به دست آوردن مقدار ذخیره شده، هیچ تضمینی نیست همان عددی که وارد شده است بازگردانده شود. اگر تا به حال در برنامه هایتان به چنین مشکلی برنخوردید دلیلش اعداد اعشاری کوچک بوده است. ولی با بزرگتر شدن عدد، این تفاوت به خوبی دیده می‌شود. حالا اگر بخواهیم اعداد اعشاری را به طور دقیق ذخیره کنیم، مجبور به استفاده از نوع decimal هستیم. در دات نت آنچنان محدودیتی بر سر استفاده‌ی از آن نداریم. ولی در سمت سرور داده‌ها بهتر هست برای آن تدابیری اندیشیده شود. هر عدد دسیمال از دقت و مقیاس تشکیل می‌شود. دقت آن تعداد ارقامی است که در عدد وجود دارد و مقیاس آن تعداد ارقام اعشاری است. به عنوان مثال عدد زیر دقتش 7 و مقیاسش 3 است:

```
4235.254
```

در صورتی که عدد اعشاری را به دسیمال نسبت دهیم باید حرف m را بعد از عدد وارد کنیم:

```
decimal d1=4545.112m;
```

برای اعداد صحیح نیازی نیست. برای تعیین نوع دسیمال از fluent api استفاده می‌کنیم:

```
modelBuilder.Entity<Class>().Property(object => object.property).HasPrecision(7, 3);
```

کد زیر برای خصوصیت شماره یک، دقت 18 و مقیاس 2 را در نظر می‌گیرد و دومین خصوصیت طبق آنچه که برایش تعریف کرده ایم دقت 7 و مقیاس 3 است:

```
public class Testtypes
    {
        public Decimal Decimal1 { get; set; }
        public Decimal Decimal2 { get; set; }

     }

 public class TestConfig : EntityTypeConfiguration<Testtypes>
    {
        public TestConfig()
        {
            this.Property(x => x.Decimal2).HasPrecision(7, 3);
        }
    }
```

مورد سوم مبحث رشته هاست :

کدهای زیر را مطالعه فرمایید:

```
[StringLength(25)]
public string FirstName { get; set; }

[StringLength(30)]
[Column(TypeName = "varchar")]
public string EnProductTitle { get; set; }


public string ArticleContent { get; set; }
[Column(TypeName = "varchar(max)")]
public string ArticleContentEn { get; set; }
```

اولین رشته بالا (نام) را به محدوده‌ای از کاراکترها محدود کرده‌ایم. به طور پیش فرض تمامی رشته‌ها به صورت nvarchar در نظر گرفته می‌شوند. بدین ترتیب در رشته نام کوچک nvarchar(25) در نظر گرفته خواهد شد. حال اگر بخواهیم فقط حروف انگلیسی

پشتیبانی شوند، مثلا نام فنی کالا را بخواهید وارد کنید، بهتر هست که نوع آن به طرز صحیحی تعریف شود که در کد بالا با استفاده از خصوصیت Column نوع varchar را معرفی می‌کنم. بدین ترتیب تعریف نهایی نوع به شکل varchar(30) خواهد بود. استفاده از fluentApiها هم در این رابطه به شکل زیر است:

```
this.Property(e => e.EnProductTitle).HasColumnType("VARCHAR").HasMaxLength(30);
```

برای مواردی که محدوده‌ای تعریف نشود nvarchar(MAX) در نظر گرفته میشود مانند پراپرتی ArticleContent بالا. ولی اگر قصد دارید فقط حروف اسکی پشتیبانی گردند، مثلا متن انگلیسی مقاله را نیز نگه می‌دارید بهتر هست که نوع آن بهینه‌ترین حالت در نظر گرفته شود که برای پراپرتی ArticleContentEn نوع varchar(MAX) تعریف کرده‌ایم.
همانطور که گفتیم پیش فرض رشته‌ها nvarchar است، در صورتی که دوست دارید این پیش فرض را تغییر دهید روش زیر را دنبال کنید:

```
modelBuilder.Properties<string>().Configure(c => c.HasColumnType("varchar"));

//========== یا

modelBuilder.Properties<string>().Configure(c => c.IsUnicode(false));
```

جهت تکمیل بحث نیز هر کدام از متغیرهای عددی در سی شارپ معادل نوع‌های زیر در Sql Server هستند:

```
//tinyInt
public byte Age { get; set; }

//smallInt
public Int16 OldInt { get; set; }

//int
public int Int32 { get; set; }

//Bigint
public Int64 HighNumbers { get; set; }
```

نظرات خوانندگان

نویسنده: مرتضی رییسی
تاریخ: ۱۳۹۴/۰۵/۰۴ ۲۰:۴۴

سلام. ممنون.
میشه بفرمائید برای مقادیر مالی به ریال و تومان بهترین نوع داده ای چیست؟
من اینچنین استفاده میکنم:

```
public virtual decimal CostPrice { set; get; }
```

و در کانفیگ:

```
this.Property(x => x.CostPrice)
    .HasColumnType("money")
    .IsRequired();
```

همین نوع و همین اندازه تنظیم کافیه؟ آیا تنظیم بیشتری نیاز دارد؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۰۴ ۲۲:۲

توصیه می‌کنم برای واحد پولی ایران از این جنس استفاده نکنید. از همان واحدهای عددی دقیق استفاده کنید.
اگر واحد مالی ما مانند کشورهای خارجی به صورت اعشار بیان میشد بله بهینه بود ولی در حال حاضر خیر.
من خودم تا به الان از Int استفاده کردم و می‌توان برای واحدهای بزرگتر BigInt را مورد استفاده قرار داد. هر چند int تا میلیارد را به خوبی پشتیبانی می‌کند