عنوان: **نوشتن پرس و جو در Entity Framework با استفاده از LINQ To Entity قسمت اول**
نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۴/۰۷ ۱۳:۲۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ to Entity

موجودیت‌های زیر را در نظر بگیرید:

```
public class Customer
{
    public Customer()
    {
        Orders = new ObservableCollection<Order>();
    }
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Family { get; set; }

    public string FullName
    {
        get
        {
            return Name + " " + Family;
        }
    }

    public virtual IList<Order> Orders { get; set; }
}
```

```
public class Product
{
    public Product()
    {
    }

    public Guid Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }
}
public class OrderDetail
{
    public Guid Id { get; set; }
    public Guid ProductId { get; set; }
    public int Count { get; set; }
    public Guid OrderId { get; set; }
    public int Price { get; set; }

    public virtual Order Order { get; set; }
    public virtual Product Product { get; set; }

    public string ProductName
    {
        get
        {
            return Product != null ? Product.Name : string.Empty;
        }
    }
}
```

```
public class Order
{
    public Order()
    {
        OrderDetail = new ObservableCollection<OrderDetail>();
    }
    public Guid Id { get; set; }
    public DateTime Date { get; set; }

    public Guid CustomerId { get; set; }
    public virtual Customer Customer { get; set; }
    public virtual IList<OrderDetail> OrderDetail { get; set; }

    public string CustomerFullName
    {
        get
```

```
        {
            return Customer == null ? string.Empty : Customer.FullName;
        }
    }

    public int TotalPrice
    {
        get
        {
            if (OrderDetail == null)
                return 0;

            return
                OrderDetail.Where(orderdetail => orderdetail.Product != null)
                .Sum(orderdetail => orderdetail.Price*orderdetail.Count);
        }
    }
}
```

و نگاشت موجودیت ها:

```
public class CustomerConfiguration : EntityTypeConfiguration<Customer>
{
    public CustomerConfiguration()
    {
        HasKey(c => c.Id);
        Property(c => c.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class ProductConfiguration : EntityTypeConfiguration<Product>
{
    public ProductConfiguration()
    {
        HasKey(p => p.Id);
        Property(p => p.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class OrderDetailConfiguration : EntityTypeConfiguration<OrderDetail>
{
    public OrderDetailConfiguration()
    {
        HasKey(od => od.Id);
        Property(od => od.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class OrderConfiguration: EntityTypeConfiguration<Order>
{
    public OrderConfiguration()
    {
        HasKey(o => o.Id);
        Property(o => o.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
```

و برای معرفی موجودیت‌ها به Entity Framwork کلاس StoreDbContext را به صورت زیر تعریف می‌کنیم:

```
public class StoreDbContext : DbContext
{
    public StoreDbContext()
        : base("name=StoreDb")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Configurations.Add(new CustomerConfiguration());
        modelBuilder.Configurations.Add(new OrderConfiguration());
        modelBuilder.Configurations.Add(new OrderDetailConfiguration());
        modelBuilder.Configurations.Add(new ProductConfiguration());
    }
```

```
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
}
```

جهت مقدار دهی اولیه به database تستی یک DataBaseInitializer به صورت زیر تعریف می‌کنیم:

```
public class MyTestDb : DropCreateDatabaseAlways<StoreDbContext>
{
    protected override void Seed(StoreDbContext context)
    {
        var customer1 = new Customer { Name = "Vahid", Family = "Nasiri" };
        var customer2 = new Customer { Name = "Mohsen", Family = "Jamshidi" };
        var customer3 = new Customer { Name = "Mohsen", Family = "Akbari" };

        var product1 = new Product {Name = "CPU", Price = 350000};
        var product2 = new Product {Name = "Monitor", Price = 500000};
        var product3 = new Product {Name = "Keyboard", Price = 30000};
        var product4 = new Product {Name = "Mouse", Price = 20000};
        var product5 = new Product {Name = "Power", Price = 70000};
        var product6 = new Product {Name = "Hard", Price = 250000};

        var order1 = new Order
        {
            Customer = customer1, Date = new DateTime(2013, 1, 1),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 1, Price = product1.Price},
                new OrderDetail {Product = product2, Count = 1, Price = product2.Price},
                new OrderDetail {Product = product3, Count = 1, Price = product3.Price},
            }
        };

        var order2 = new Order
        {
            Customer = customer1,
            Date = new DateTime(2013, 1, 5),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 2, Price = product1.Price},
                new OrderDetail {Product = product3, Count = 4, Price = product3.Price},
            }
        };

        var order3 = new Order
        {
            Customer = customer1,
            Date = new DateTime(2013, 1, 9),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 4, Price = product1.Price},
                new OrderDetail {Product = product3, Count = 5, Price = product3.Price},
                new OrderDetail {Product = product5, Count = 6, Price = product5.Price},
            }
        };

        var order4 = new Order
        {
            Customer = customer2,
            Date = new DateTime(2013, 1, 9),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product4, Count = 1, Price = product4.Price},
                new OrderDetail {Product = product3, Count = 1, Price = product3.Price},
                new OrderDetail {Product = product6, Count = 1, Price = product6.Price},
            }
        };

        var order5 = new Order
        {
            Customer = customer2,
            Date = new DateTime(2013, 1, 12),
            OrderDetail = new List<OrderDetail>
            {
```

```
                new OrderDetail {Product = product4, Count = 1, Price = product4.Price},
                new OrderDetail {Product = product5, Count = 2, Price = product5.Price},
                new OrderDetail {Product = product6, Count = 5, Price = product6.Price},
            }
        };

        context.Customers.Add(customer3);

        context.Orders.Add(order1);
        context.Orders.Add(order2);
        context.Orders.Add(order3);
        context.Orders.Add(order4);
        context.Orders.Add(order5);

        context.SaveChanges();
    }
```

و در ابتدای برنامه کد زیر را جهت مقداردهی اولیه به Database مان قرار می‌دهیم:

```
Database.SetInitializer(new MyTestDb());
```

در انتها ConnectionString را در App.Config به صورت زیر تعریف می‌کنیم:

```
<connectionStrings>
    <add name="StoreDb" connectionString="Data Source=.\SQLEXPRESS;
Initial Catalog=StoreDBTest;Integrated Security = true" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

بسیار خوب، حالا همه چیز محیاست برای اجرای اولین پرس و جو:

```
using (var context = new StoreDbContext())
{
    var query = context.Customers;

    foreach (var customer in query)
    {
        Console.WriteLine("Customer Name: {0}, Customer Family: {1}",
                                    customer.Name, customer.Family);
    }
}
```

پرس و جوی تعریف شده لیست تمام Customerها را باز می‌گرداند. query فقط یک "عبارت" پرس و جو هست و زمانی اجرا می‌شود که از آن درخواست نتیجه شود. در مثال بالا این درخواست در اجرای حلقه foreach اتفاق می‌افتد و درست در این لحظه است که دستور SQL ساخته شده و به Database فرستاده می‌شود. EF در این حالت تمام داده‌ها را در یک لحظه باز نمی‌گرداند بلکه این ارتباط فعال است تا حلقه به پایان برسد و تمام داده‌ها از database واکشی شود. خروجی به صورت زیر خواهد بود:

```
Customer Name: Vahid, Customer Family: Nasiri
Customer Name: Mohsen, Customer Family: Jamshidi
Customer Name: Mohsen, Customer Family: Akbari
```

نکته : با هر بار درخواست نتیجه از query ، پرس و جوی مربوطه دوباره به database فرستاده می‌شود که ممکن است مطلوب ما نباشد و باعث افت سرعت شود. برای جلوگیری از تکرار این عمل کافیست با استفاده از متد ToList پرس و جو را در لحظه تعریف به اجرا در آوریم

```
var customers = context.Customers.ToList();
```

خط بالا دیگر یک عبارت پرس و جو نخواهد بود بلکه لیست تمام Customer هاست که به یکباره از database بازگشت داده شده است. در ادامه هرجا که از customers استفاده کنیم دیگر پرس و جویی به database فرستاده نخواهد شد.

پرس و جوی زیر مشتریهایی که نام آنها Mohsen هست را باز می‌گرداند:

```
private static void Query3()
{
    using (var context = new StoreDbContext())
    {
        var methodSyntaxquery = context.Customers
                    .Where(c => c.Name == "Mohsen");
        var sqlSyntaxquery = from c in context.Customers
                             where c.Name == "Mohsen"
                             select c;

        foreach (var customer in methodSyntaxquery)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}",
                                    customer.Name, customer.Family);
        }
    }

    // Output:
    // Customer Name: Mohsen, Customer Family: Jamshidi
    // Customer Name: Mohsen, Customer Family: Akbari
}
```

همانطور که مشاهده می‌کنید پرس و جو به دو روش Method Syntax و Sql Syntax نوشته شده است.

روش Method Syntax روشی است که از متدهای الحاقی (Extention Method) و عبارت‌های لامبدا (Lambda Expersion) برای نوشتن پرس و جو استفاده می‌شود. اما #C روش Sql Syntax را که همانند دستورات SQL هست، نیز فراهم کرده است تا کسانیکه آشنایی با این روش دارند، از این روش استفاده کنند. در نهایت این روش به Method Syntax تبدیل خواهد شد بنابراین پیشنهاد می‌شود که از همین روش استفاده شود تا با دست و پنجه نرم کردن با این روش، از مزایای آن در بخشهای دیگر کدنویسی استفاده شود.

اگر به نوع Customers که در DbContext تعریف شده است، دقت کرده باشید، خواهید دید که DbSet می‌باشد. DbSet کلاس و اینترفیس‌های متفاوتی را پیاده سازی کرده است که در ادامه با آنها آشنا خواهیم شد:

IQueryable<TEntity>, IEnumerable<TEntity>, IQueryable, IEnumerable: که امکان استفاده از متدهای نام آشنای LINQ را برای ما فراهم می‌کند. البته فراموش نشود که EF از Provider ای با نام LINQ To Entity برای تفسیر پرس و جوی ما و ساخت دستور SQL متناظر آن استفاده می‌کند. بنابراین تمامی متدهایی که در LINQ To Object استفاده می‌شوند در اینجا قابل استفاده نیستند. بطور مثال اگر در پرس و جو از LastOrDefault روی Customer استفاده شود در زمان اجرا با خطای زیر مواجه خواهیم شد و در نتیجه در استفاده از این متدها به این مسئله باید دقت شود.

LINQ to Entities does not recognize the method 'Store.Model.Customer

LastOrDefault[Customer](System.Linq.IQueryable`1[Store.Model.Customer], System.Linq.Expressions.Expression`1[System.Func`2[Store.Model.Customer,System.Boolean]])' method, and this method cannot be translated into a store expression.

<DbSet<TEntity>: که دارای متدهای Add, Attach, Create, Find, Remove, Local می‌باشد و برای بحث ما Find و Local جهت ساخت پرس و جو استفاده می‌شوند که در ادامه توضیح داده خواهند شد.

<DbQuery<TEntity>: که دارای متدهای AsNoTracking و Include می‌باشد و در ادامه توضیح داده خواهند شد.

متد Find: این متد کلید اصلی را به عنوان ورودی گرفته و برای بازگرداندن نتیجه مراحل زیر را طی می‌کند:

داده‌های موجود در حافظه را بررسی می‌کند یعنی آنهایی که Load و یا Attach شده اند.

داده هایی که به DbContext اضافه (Add) ولی هنوز در database درج نشده اند.

داده هایی که در database هستند ولی هنوز Load نشده اند.

Find در صورت پیدا نکردن Exception ای صادر نمی‌کند بلکه مقدار null را بر می‌گرداند.

```
private static void Query4()
{
    using (var context = new StoreDbContext())
    {
        var customer = context.Customers.Find(new Guid("2ee2fd32-e0e9-4955-bace-1995839d4367"));

        if (customer == null)
            Console.WriteLine("Customer not found");
        else
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name,
customer.Family);
    }
}
```

با توجه به اینکه Idها توسط Database ساخته می‌شوند. شما باید از Id دیگری که موجود می‌باشد، استفاده کنید تا نتیجه ای برگشت داده شود.
نکته: در صورتیکه کلید اصلی شما از دو یا چند فیلد تشکیل شده بود. می‌بایست این دو یا چند مقدار را به عنوان پارامتر به Find بفرستید.

متد Single: گاهی نیاز هست که داده‌ای پرس و جو شود اما نه با کلید اصلی بلکه با شرط دیگری، در این حالت از Single استفاده می‌شود. این متد یک مقدار را باز می‌گرداند و در صورتی که صفر یا بیش از یک مقدار در شرط صدق کند exception صادر می‌کند. متد SingleOrDefault رفتاری مشابه دارد اما اگر مقداری در شرط صدق نکند مقدار پیش فرض را باز می‌گرداند.
نکته: مقدار پیش فرض بستگی به نوع خروجی دارد که اگر object باشد مقدار null و اگر بطور مثال نوع عددی باشد، صفر می‌باشد.

```
private static void Query5()
{
    using (var context = new StoreDbContext())
    {
        try
        {
            var customer1 = context.Customers.Single(c => c.Name == "Unkown");  // Exception: Sequence
contains no elements
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
```

```
        }

        try
        {
            var customer2 = context.Customers.Single(c => c.Name == "Mohsen");  // Exception: Sequence
contains more than one element
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        var customer3 = context.Customers.SingleOrDefault(c => c.Name == "Unkown");  // customer3 ==
null

        var customer4 = context.Customers.Single(c => c.Name == "Vahid");  // customer4 != null
    }
}
```

متد First: در صورتیکه به اولین نتیجه پرس و جو نیاز هست می‌توان از First استفاده کرد. اگر پرس و جو نتیجه در بر نداشته باشد یعنی null باشد exception صادر خواهد شد اما اگر FirstOrDefault استفاده شود مقدار پیش فرض برگردانده خواهد شد.

```
private static void Query6()
{
    using (var context = new StoreDbContext())
    {
        try
        {
            var customer1 = context.Customers.First(c => c.Name == "Unkown");  // Exception: Sequence
contains no elements
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        var customer2 = context.Customers.FirstOrDefault(c => c.Name == "Unknown");  // customer2 ==
null
        var customer3 = context.Customers.First(c => c.Name == "Mohsen");
    }
}
```

نظرات خوانندگان

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۵

خیلی ممنون مهندس، فقط اینکه در داخل سازنده StoreDbContext چرا به این شکل عمل کرده اید:

```
public StoreDbContext()
: base("name=StoreDb")
{
}
```

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۳۳

StoreDb نام مدخل Connection String ای هست که در AppConfig ایجاد شده

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۴۶

ممنون از پاسخگوییتون ، در واقع اگر اینکارو نمیکردید باید در AppConfig نام Connection را StoredDbContext میگذاشتیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۲:۲۶

چندین روش برای تعریف رشته اتصالی در EF وجود دارد. بیشتر در اینجا

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۴:۳۰

بله البته به همراه namespace اگه اشتباه نکنم