به نام کسی که آفرید از عدم به انسان عطا کرد فکر و قلم

Linq در C#

ارائه دهنده : مصطفی ذاکری نسب

استاد : مهندس محمد سلیمی

امروزه دیگر نمیتوان منکر کاربرد نرمافزارهای کامپیوتری در زندگی روزمره بشر گردید. همچنین در اکثر برنامههایی که امروزه بر روی کامپیوترها به کار میروند، استفاده از بانکهای اطلاعاتی انکارناپذیر است. به همین دلیل از اوایل دهه 80 میلادی تلاشهای فراوانی برای بهبود نرمافزاریهای بانک اطلاعاتی گردید و منجر به ظهور مدلهای جدید و متعددی برای ایجاد پایگاههای داده گردید. یکی از این مدلها که هم اکنون در بسیاری از نرمافزارها پیاده سازی شده است مدل RDBMS را نام برد. در حال حاضر، برای ایجاد و Oracle و همچنین SQL Server میباشد. به عنوان نمونه میتوان توسعه برنامههای مبتنی بر بانکهای اطلاعاتی، این دو نرمافزار بیشتر از مدلهای مشابه مورد استفاده قرار میگیرند.

پس از انتخاب نرمافزار ارائه دهنده خدمات بانک اطلاعاتی، نوبت به انتخاب زبان برنامه نویسی میرسد. در اینجا نیز ذکر این مورد کافیست که اگر نگاهی که گذشته داشته باشیم متوجه میشویم که در بیست سال گذشته زبانهای با قابلیت برنامه نویسی شئگرا به عنوان اولین و بهترین زبان برای توسعه نرمافزارها در نظر گرفته میشوند. در میان زبانهای با قابلیت برنامه نویسی شئگرا زبان C# و Java از امکانات بهتری برخوردارند.

با نگاهی به دو نرمافزار انتخاب شده در بالا متوجه به تضادی میشویم که از گذشته تا کنون گریبانگیر برنامهنویسان متعددی بوده است. با توجه به اینکه SQL Server یک بانک اطلاعاتی رابطهای است پس نمیتوان با آن به صورت شئ-گرا در برنامهها ارتباط برقرار کرد. البته مایکروسافت برای رفع این مشکل تا قبل از سال 2005 فعالیتهایی نظیر ارائه DataSet و ADO.NET انجام داده است. اما با این حال هنوز نمیتوان به صورت کاملاً شئگرا با این مدل از بانکهای اطلاعاتی ارتباط برقرار کرد.

در این میان شرکتهای مطرح نرمافزاری دنیا مانند Mircrosoft نیز بیکار نماندهاند و بر روی پروژههایی برای رفع مشکل یاد شده در بالا فعالیت نمودهاند. مهمترین و بهترین پروژهای که در این مورد بر روی آن کار شده است پروژه LINQ میباشد. با استفاده از این تکنولوژی دیگر نیاز به کار بردن روشهای متفاوت برای بازیابی اطلاعات از منابع دادهای گوناگون مانند بانکهای اطلاعاتی رابطهای، اسناد XML و حتی اشیاء درون حافظه نیست. در این روش از یک گرامر مشابه برای بازیابی اطلاعات از هر نوع منبع دادهای استفاده میشود؛ و البته گرامری که این روش از آن استفاده میکند به صورت شئگرا میباشد.

در ادامه نگاهی کلی بر فرم کلی این روش خواهیم انداخت و سپس به توضیح و بررسی قسمتهای مختلف پروژه می-پردازیم.

برای درک کلی گرامر این روش مثال زیر را در نظر بگیرید:

```
int[ ] i= {1,2,3,4,5,6,7,8,9,10,11};

var query = from p in i

        where p>=5

        select p;

objectdumper.write(query);
```

اگر کدهای بالا را کامپایل و اجرا کنید کلیه اعدادی که در آرایه i از عدد 5 بیشتر میباشند انتخاب و در متتغیر query ذخیره میشوند. این مدل استفاده از LINQ به مدل عملگری مشهور است. کامپایلر C# پس از برخورد با این عبارت آن را به فراخوانی متدهایی تبدیل میکند. پس از تبدیل، عبارت پرس و جوی بالا به صورت زیر در خواهد آمد.

var query= i.where(p=> p>5);

خروجی هر دو مدل معادل یکدیگر میباشند. متغیر p که در هر دو حالت از آن استفاده شده است برای حرکت در مجموعه مورد نظر، به کار میرود و در هر بار به یک عنصر در این مجموعه اشاره میکند.

در مورد کلاس ObjectDumper باید به عرض برسانم که این ابزار یک کلاس ساده با یک متد به نام Write() است و از آن برای تهیه خروجی قالبدار استفاده میشود. این بدان معناست که دیگر برای چاپ اطلاعات لازم نیست با استفاده از ساختارهای تکرار در مجموعه مورد نظر حرکت کنید و با استفاده از متد Write()و یا WriteLine() که در کلاس Console قرار دارند اطلاعات را چاپ کنید.

ObjectDumper هنگام نصب .NET بر روی کامپیوتر شما کپی میشود. بنابراین برای استفاده از آن کافیست فایل Objectdumper.cs را جستجو کرده و آن را به پروژه خود اضافه کنید.

اما اگر به جای استفاده از LINQ، بخواهیم با استفاده از روشهای متداول قبلی این کار را انجام دهیم باید از دستوراتی مانند زیر استفاده نمود.

```
int[] i= {1,2,3,4,5,6,7,8,9,10,11};

for (int j = 0; j < i.Length ; j++)

{

    if (i[j]>=5)

    {

        Console.WriteLine(i[j]);

    }

}
```

همانطور که مشاهده میکنید استفاده از LINQ برای بازیابی اطلاعات بسیار سادهتر و قابل فهمتر میباشد. البته مزیت دیگر این تکنولوژی به گرامر آن برمیگردد. گرامر LINQ بسیار شبیه به گرامر SQL است؛ با توجه به اینکه کمتر برنامه-نویسی پیدا میشود که با گرامر SQL آشنایی نداشته باشد، لذا استفاده از این گرامر بهترین انتخاب برای این تکنولوژی به حساب میرود.

در ادامه قصد دارم که با بررسی مثالی پیچیدهتر وارد جزئیات LINQ شوم. برای این منظور ابتدا یک کلاس به نام Person که دارای 4 خصوصیت می باشد به صورت زیر تعریف میکنم.

```
Class Person

{

}
```

تکه کد زیر را در در نظر بگیرید:

```
List<Person> people = new List<Person>

{

  new Person{ID=1, IDRole=1, LastName="Andy", FirstName="Brad"},

  new Person{ID=2, IDRole=2, LastName="Gray", FirstName="Tom" },

  new Person{ID=3, IDRole=2, LastName="Gran", FirstName="Mary"},

  new Person{ID=4, IDRole=3, LastName="Cops", FirstName="Gary"}

};

var query = from p in people

            where p.FirstName.Length  ==  4

            select new

        {

           p.FirstName, p.LastName

        };

ObjectDumper.Write(query);
```

در این برنامه چند نکته جالب توجه وجود دارد که از خصوصیت جدید C# 3.0 می باشد. در ابتدای کار ممکن است تعجب کنید که چرا برای شئ people نوعی در نظر گرفته نشده است و از کلمه کلیدی var استفاده شده است.

آیا var یک نوع جدید در C# می باشد؟ در جواب باید گفت که نه؛ در C# 3.0 میتوان متغیرهای محلی با نوعهای ضمنی ایجاد کرد. این بدان معناست که دیگر الزامی نیست که برای متغیرها حتماً یک نوع عرفی تعیین کنید و با گذاشتن کلمه کلیدی var قبل از تعریف متغیر به کامپایلر اعلام میکنید که نوع این متغیر را براساس عبارتی که برای مقداردهی اولیه آن در نظر گرفته شده است تعیین کند.

به عنوان مثال هر دو دستور زیر معادل یکدیگر می باشند:

```
var n = 5;

int n = 5;
```

در استفاده از کلمه کلیدی var باید توجه داشته باشید که متغیری که با این نوع اعلام میشود باید حتماً مقداردهی اولیه گردد؛ در غیر اینصورت برنامه با خطای کامپایلری روبرو میشود.

استفاده از کلمه کلیدی var برای تعیین نوع خروجی پرس و جوهای پیچیده‌ای که در ادامه ممکن است با آنها روبرو شویم بسیار مفید است و برنامه نویس را از قید تعیین یک نوع تعریف شده برای خروجی پرس و جوها آزاد میسازد.

به توضیح ادامه برنامه بالا برمیگردیم. در ابتدا یک شئ از نوع T>List< به نام people ایجاد شده است. در تعریف این شئ یکی دیگر از خصوصیات C# 3.0 مشاهده میشود. این خصوصیت تحت عنوان مقداردهنده اولیه به اشیاء شناخته می شود.

با استفاده از مقداردهنده اولیه به اشیاء، همانند مقداردهنده اولیه به آرایه‌ها، میتوانید بدون فراخوانی سازنده کلاسی که از روی آن نمونه‌سازی انجام میگیرد (و یا فراخوانی متدهایی که برای درج در نظر گرفته شده اند مانند Add() در کلاس T>List<) با استفاده از یک خط کد یک شئ از روی کلاس ایجاد و مقداردهی کرد. برای درک بیشتر به مثال زیر توجه کنید:

```
Person p1= new Person{ID = 1, FirstName="A", LastName="A"};

Person p2 = new Person();

p2.ID = 1;
```

مشاهده میکنید که با استفاده از این خصوصیت کار ایجاد اشیاء چقدر آسانتر میشود. در ادامه با استفاده از یک پرس و جو ساده نام و نام خانوادگی افرادی که طول فیلد نام آنها برابر 4 باشد بازیابی و نشان داده میشود.

همانطورکه مشاهده میکنید دستوری که برای این منظور در نظر گرفته شده است بسیار شبیه به دستور قبلی است.

امیدوارم که تا اینجای کار متوجه اصول کلی کار با LINQ شده باشید. در LINQ عملگرهای پرس و جو بسیاری وجود دارد؛ اما به دلیل کمبود وقت از تشریح آنها صرف نظر میکنم. البته در انتهای مطلب برای مطالعه بیشتر منابعی را که مورد استفاده قرار دادهام ذکر خواهم کرد.

تا بدین جا نحوه استفاده از LINQ برای بازیابی اطلاعات از اشیاء درون حافظه مورد بررسی قرار گرفت. در ادامه نگاهی بر نحوه ارتباط این تکنولوژی با پایگاه داده Server SQL خواهیم انداخت

ارتباط LINQ با SQL

برای بازیابی اطلاعات از پایگاه داده SQL روشی مشابه روش بالا مورد استفاده قرار میگیرد. تنها تفاوت موجود به تعیین منبع داده در قسمت from از عبارت پرس و جوی LINQ مربوط میگردد. در اینجا باید برای بازیابی اطلاعات از جداول داده و استفاده در برنامه کلاسهایی را ایجاد کرد.

اولین کار اضافه کردن فضانام مربوط به استفاده از LINQ to SQL<> می باشد. برای این کار باید فضانام System.Data.Linq را به پروژه اضافه کنید. البته باید اسمبلی این فضانام را به Reference های برنامه اضافه نمود. اسمبلی حاوی این فضانام به صورت پیش فرض در لیست موجود در پنجره Add Reference به نام System.Data.Linq.dll وجود دارد.

پس از این کار باید به ازاء هر جدولی که قصد داریم در برنامه از آن استفاده کنیم، یک کلاس ایجاد کرد. البته تعریف این کلاس قدری با تعریف کلاسهای معمولی در C # تفاوت دارد.

برای شروع ایجاد کلاس ابتدا جدول زیر را در نظر بگیرید. در این جدول 3 فیلد وجود دارد.

در کلاسی که قرار است با این جدول ارتباط داده شود باید به ازاء هر فیلد در جدول یک فیلد خصوصی در کلاس ایجاد کنید؛ و البته برای دسترسی به اطلاعات این فیلد خصوصی باید یک خصوصیت عمومی نیز تعریف شود.

تفاوتی که در بالا به آن اشاره شد، به استفاده از ویژگیهای جدید C # در تعریف کلاس برمیگردد. اولین ویژگی، ویژگی Table است. هدف استفاده از این ویژگی این است که به کامپایلر اعلام شود این کلاس قرار است با کدام جدول در پایگاه داده ارتباط داده شود

```
[Table(Name = "Person")]

public class Person

{

}
```

خصوصیت Name از این ویژگی در واقع بیانگر نام جدول است. اگر از این خصوصیت استفاده نکنید، کامپایلر به صورت خودکار نام کلاس را به عنوان نام جدول در نظر می گیرد.

```
[Table(Name = "Person")]

public class Person

{

}
```

در ادامه باید فیلدهای خصوصی مناسبی برای نگهداری اطلاعات فیلدهای موجود در جدول ایجاد شود. ایجاد این فیلدها مانند ایجاد فیلدهای عادی است.

```
[Table(Name = "Person")]

public class Person

{

    private int _ID;
```

}

حال باید برای دسترسی به اطلاعات این فیلدها و همچنین ارتباط آنها با ستون های موجود در جدول، خصوصیتهای مناسبی تعریف کرد. البته در تعریف خصوصیتها باید از ویژگی Column نیز استفاده کرد.

```
[Column(Name = "ID", Storage = "_ID", DbType = "int NOT NULL IDENTITY", IsPrimaryKey = true, IsDbGenerated = true)]
  public int ID
  {
    get { return _ID; }
    set { _ID = value; }
  }
```

از این خصوصیت برای دسترسی به اطلاعات فیلد _ID استفاده میشود. البته با استفاده از ویژگی Column ستونی در جدول که قرار است اطلاعات آن در فیلد _ID قرار گیرد مشخص میشود. در اینجا ستون مورد نظر ID است.

ویژگی Column دارای خصوصیات متعددی است. در ادامه تعدادی از آنها را که پر کاربرد هستند توضیح میدهم.

Name: از این خصوصیت برای تعیین نام ستون مورد نظر در جدول دادهها استفاده میشود.

Storage: با استفاده از این خصوصیت نام فیلدی که قرار است اطلاعات ستون مورد نظر در آن قرار گیرد مشخص می شود.

DbType: از این خصوصیت برای تعیین نوعی که در جدول برای ستون مورد نظر در نظر گرفته شده است، استفاده می-کنیم. با به کارگیری این خصوصیت، کامپایلر قادر به چک کردن نوع مقادیری که انتساب داده میشوند در زمان کامپایل میباشد.

IsPrimaryKey: اگر ستونی که قرار است آن را Bind کنیم در جدول به عنوان کلید اصلی در نظر گرفته شده باشد، با استفاده از این خصوصیت باید این مورد را به کامپایلر و LINQ اعلام کرد.

IsDbGenerated: اگر در تعریف ستون مورد نظر در پایگاه داده اعلام شده باشد که مقدار این فیلد در توسط خود پایگاه داده تولید میشود، باید از این خصوصیت به همراه مقدار true استفاده کرد.

در زیر کد کاملی که برای کلاس Person باید نوشته شود را مشاهده می کنید.

```
[Table(Name = "Person")]
public class Person
{
  private int _ID;
  private string _lastName;
```

```
    private string _firstName;

    [Column(Name = "ID", Storage = "_ID", DbType = "int NOT NULL IDENTITY", IsPrimaryKey = true, IsDbGenerated = true)]
    public int ID
    {
        get { return _ID; }
        set { _ID = value; }
    }

    [Column(Name = "LastName", Storage = "_lastName", DbType = "nvarchar NOT NULL")]
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }

    [Column(Name = "FirstName", Storage = "_firstName", DbType = "nvarchar NOT NULL")]
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
}
```

تا بدینجا کلیه عملیات لازم برای ذخیره اطلاعات جدول Person در اشیاء موجود در برنامه انجام شده است. اکنون باید برای برقراری ارتباط با پایگاه داده تدبیری اندیشید برای اینکار دیگر لازم نیست مانند قبل از اشیاء SqlConnection استفاده کرد. LINQ برای برطرف کردن این مورد کلاسی به نام DataContext معرفی کرده است. برای استفاده از ویژگیهای این کلاس میبایست ابتدا کلاسی ایجاد کنید و از کلاس DataContext ارث بری کنید

```
public partial class PeopleDataContext : DataContext

{

}
```

حال باید در سازنده کلاس PeopleDataContext سازنده کلاس پدر (DataContext) را فراخوانی کرد. در واقع DataContext یک کانال دوطرفه برای ارتباط با پایگاه داده میباشد. یعنی از یک طرف دستورات LINQ را گرفته و آن را به دستورات معادل در SQL تبدیل می کند و آن را به سمت سرور پایگاه داده منتقل میکند. و پس از اجرای دستورات در سرور نتایج بدست آمده را دریافت و در اشیاء مربوط در برنامه قرار می دهد.

بنابراین برای ارتباط به سرور پایگاه داده در کلاس DataContext سازنده های مختلفی در نظر گرفته شده است. ساده ترین سازنده استفاده از یک رشته اتصال است که در مدلهای قبلی اتصال به پایگاه داده نیز مورد استفاده قرار می گرفت.

برای این منظور باید کلاس PeopleDataContext را به صورت زیر تغییر داد:

```
public partial class PeopleDataContext : DataContext

{

   public PeopleDataContext(String connString) :

      base(connString) { }

}
```

بنابراین باید در هنگام نمونه سازی از این کلاس، باید رشته اتصال مورد نظر را به سازنده این کلاس ارسال کرد.
کمی صبر کنید؛ هنوز کار تمام نشده است. باید برای نگهداری اطلاعاتی که قرار است از جدول Person بازیابی شوند باید شئ ای از نوع T>Table < را در کلاسی که از DataContext ارث بری میکند اضافه کنیم. برای این منظور باید فیلد زیر را به کلاس PeopleDataContext اضافه کنیم.

```
public Table<Person> People;
```

نوع T>Table< یک کلاس جامع می باشد که توسط LINQ ارائه شده است.

اکنون همه چیز برای انجام پرس و جو ار پایگاه داده آماده است. فرض کنید که میخواهیم اطلاعات افرادی که شماره شناسایی (ID) از 2 بیشتر است را بازیابی کنیم و در یک کنترل DataGridView نشان دهیم. برای این کار باید از کدهایی زیر استفاده کنیم.

```
private void btnLoad_Click(object sender, EventArgs e)

{

   String connString = @" Data Source=.;Initial Catalog=People; Integrated Security=True";

   PeopleDataContext people = new PeopleDataContext(connString);
```

```
    var query = from p in people.People
                where p.ID > 2
                select p;
    dataGridView1.DataSource = query;
}
```

کدهای فوق در رویداد کلیک از کنترل button نوشته شده است.

همانطورکه مشاهده میکنید نحوه پرس و جو دقیقاً مانند روشی است که برای پرس و جو از اشیاء درون حافظه به کار می-رود؛ فقط باید منبع دادهای که برای پرس و جو در نظرگرفته شده است را مطابق با نیاز تنظیم کنیم. در انتها نیز برای نشان داده اطلاعات بازیابی شده کافی است به خصوصیت DataSource از کنترل DataGridView را به متغیری که برای ارجاع به خروجی پرس و جو در نظر گرفته شده است مرتبط کنیم.

کلاس DataContext خصوصیات پیشرفته دیگری نیز دارد که در ادامه تعدادی از آنها و همچنین نحوه درج و حذف اطلاعات را نیز بررسی میکنیم.

یکی از خصوصیات جالب کلاس DataContext خصوصیت Log میباشد. از این خصوصیت برای بدست آوردن دستوری که DataContext برای ارسال به پایگاه داده ایجاد می کند، استفاده می شود. به عنوان مثال تکه برنامه زیر را در نظر بگیرید، با استفاده از خصوصیت Log دستور Selectی که برای بازیابی اطلاعات ایجاد شده است، را مشاهده می کنیم.

```
String connString = @"Server=.;database=People;integrated security=sspi";
PeopleDataContext people = new PeopleDataContext(connString);
people.Log = Console.Out;
var query =
from p in people.People
where p.ID == 1
select new
{
    p.LastName, p.FirstName
};
foreach (var row in query)
```

```
{
   Console.WriteLine("Name: {0}, {1}",row.LastName, row.FirstName);
}
```

خروجی این برنامه که در حالت Console نوشته شده است، به صورت زیر میباشد.

```
Asdadasdasd
```

مشاهده میشود که دستور مورد نظر ما با گرامر SQL توسط DataContext ایجاد شده است.

درج رکورد با استفاده از LINQ

برای درج یک رکورد جدید با استفاده از LINQ فقط کافیست که یک شئ جدید از نوع کلاسی که به جدول مورد نظر شما Bind شده است، ایجاد کنید و سپس آن را به فیلدی که از نوع <T>Table در کلاس اصلی برنامه (کلاسی که از DataContext ارث بری میکند) اضافه کنید. با انجام این کار اطلاعات به صورت موقت در جدول موجود در برنامه درج میشود. برای درج کامل ونهایی اطلاعات اضافه شده میبایست متد () SubmitChanges که در کلاس DataContext قرار دارد را فراخوانی کنید.

به عنوان مثال کدهای زیر یک رکورد جدید به جدول Person اضافه میکند.

```
Person person = new Person();
person.FirstName = txtName.Text;
person.LastName = txtFamily.Text;
people.People.Add(person);
people.SubmitChanges();
```

توجه داشته باشید که در رکورد جدید مقداری برای فیلد ID در نظر گرفته نشده است؛ زیرا این فیلد باید توسط پایگاه داده مقداردهی گردد. در اینجا یکی دیگر از مزیتهای LINQ به چشم میخورد. پس از اینکه متد () SubmitChanges فراخوانی گردید اطلاعاتی که توسط پایگاه داده تولید میشود به شئ موجود در برنامه برگشت داده میشود. بنابراین برای اعلام شماره شناسایی افراد لازم نیست که با دیگر یک پرس و جو بر روی جدول اعمال شود. زیرا در همان رکوردی که اطلاعات فرد قرار دارد شماره شناسایی وی نیز قرار گرفته است.

البته اگر از خصوصیت Log در کلاس DataContext برای بدست آوردن دستوری که برای پایگاه داده ارسال شده است، استفاده کنید، مشاهده خواهیم کرد که پس از دستور Insert یک دستور Select استفاده شده است و در آن با استفاده از تابع Scope_Identity() آخرین مقداری که به فیلدی که به خصوصیت Identity آن true است برگشت داده میشود.

ویرایش رکوردها

برای ویرایش رکوردها ابتدا باید رکورد مورد نظر را بازیابی کنیم و سپس اطلاعات موجود در آن را تغییر دهیم و سپس با استفاده از فراخوانی متد SubmitChanges() تغییرات را به پایگاه داده ارسال کرد. در اینجا یکی دیگر از مزیتهای LINQ مورد استفاده قرار میگیرد. در LINQ سرویسی به نام سرویس پیگیری تغییرات وجود دارد؛ با استفاده از این سرویس تغییرات اعمال شده بر روی دادهها پیگری میشود. اگر متد SubmitChanges() فراخوانی شود و این در حال باشد که در اطلاعات موجود تغییری داده نشده باشد، LINQ از ایجاد دستور Update و ارسال برای پایگاه داده خودداری میکنید.

در تکه برنامه زیر ابتدا رکوردی بازیابی شده و سپس مقدار خصوصیات آن تغییر میکند. در انتها با فراخوانی متد SubmitChanges() تغییرات به پایگاه داده اعمال میگردد.

```
public String connString = @"Server=.;database=People;integrated security=sspi";

PeopleDataContext people = new PeopleDataContext(connString);

 var person = people.People.Single(p => p.ID == 1);

person. FirstName="A";

person. LastName="A";

people.SubmitChanges();
```

با اجرای برنامه مقادیر فیلدهای نام و نام خانوادگی فردی که شماره شناسایی وی برابر 1 میباشد به A تغییر پیدا کرده و با فراخوانی متد SubmiyChanges() دستور مناسبی تولید شده و برای پایگاه داده ارسال میگردد. حال اگر دوباره برنامه را اجرا کنیم با توجه به اینکه تغییری در اطلاعات حاصل نمیشود بنابراین دستوری برای ارسال به پایگاه داده ایجاد نمی-شود.

حذف رکوردها

حذف رکورد از دو عمل قبل کاملاً آسانتر است. برای اینکار کافی است که اطلاعات را از جدول بازیابی کنید و سپس با استفاده از متد Remove() شئ مربوطه را حذف کنید. در زیر قطعه کدی برای این منظور نوشته شده است.

```
var person = people.People.Where (p => p.IDRole == 1);

people.People.Remove(person);

people.SubmitChanges();
```

در این برنامه اطلاعات فردی که شماره شناسایی وی برابر با ۱ میباشد از پایگاه داده حذف میشود.

امکانات اضافی LINQ

ممکن است بسیاری از برنامه نویسان فکر کنند که ایجاد کلاسهای موجودیت برای برنامههای بزرگ که بانک اطلاعاتی بسیار بزرگی دارند، مقرون به صرفه نباشد و استفاده از مدلهای قبلی بسیار مفیدتر باشد.

البته این تفکر شاید تا حدودی درست باشد. اما با تاملی دوباره و عمیق درباره استفاده شئگرایانه از بانکهای اطلاعاتی ارتباطی میتواند بر روی این مشکل سرپوش بگذارد. البته جای نگرانی نیز وجود ندارد؛ زیرا سازندگان این اثر نیز به فکر این مشکل بودهاند و برای حل آن ابزارهایی نیز ارائه کردهاند.

یکی از این ابزارها که بسیار پرکاربرد نیز میباشد، ابزار SQLMetal میباشد. با استفاده از این ابزار میتوانید کلاس-های موجودیت برای تمامی جداول موجود در پایگاه داده را ایجاد کنید. برای این منظور ابتدا باید خط فرمان که توسط Visual Studio 2008 ارائه شده است را اجرا کنید و از این ابزار استفاده کنید.

دستور زیر را در نظر بگیرید:

```
sqlmetal /server:. /database:People /pluralize /code:People.cs
```

با استفاده از این دستور کلاسهای موجودیت که در بانک اطلاعاتی People وجود دارد در فایل People.cs ایجاد میشود و به راحتی میتوانید با اضافه کردن این فایل به پروژه از کلاسهای آن استفاده کنید.

این ابزار پارامترهای متعددی دارد که در ادامه تعدادی از پر کاربردترین آنها را شرح میدهیم:

Server: از این پارامتر برای تعیین سرور پایگاه داده استفاده میشود. در این مثال با توجه به اینکه پایگاه داده در کامپیوتر محلی قرار دارد از علامت نقطه (.) استفاده شده است.

Database: از این پارامتر برای تعیین نام پایگاه دادهای که قرار است براساس آن کلاسهای موجودیت ایجاد شود، استفاده میشود.

Pluralize: از این پارامتر برای تعیین گرامر انگلیسی در ایجاد نام کلاسها و خصوصیات استفاده میشود.

Code: از این پارامتر برای تعیین محلی که باید کلاسهای موجودیت در آن قرار گیرند استفاده میشود. البته براساس پسوند فایل زبان C# و یا VB برای ایجاد کلاسها استفاده میشود. برای این کار میتوان از پارامتر Language نیز استفاده کرد.

این ابزار پارامترهای دیگری نیز دارد. برای درک عملکرد آن پارامترها میتوانید به منابعی که ذکر میشود مراجعه کنید.

منابع مورد استفاده :

- کتاب "LINQ در C# 2008" تالیف گروه واژه
- Ebook های موجود در سایت PersiaDevelopers

نکته: برای تست کدهای این مقاله از Visual Studio 2008 Beta 2 استفاده شده است. در ویرایش نهایی این نرمافزار تغییرات جزئی در LINQ to SQL اعمال شده است. این تغییرات در حال بررسی میباشند و در مقالههای بعدی ممکن است برای شما عزیزان قرار داده شود.