بسـم الله الرحمن الرحيم

Multilayer database application

سنقوم في هذا الدرس بعمل تطبيق بسيط، متعدد الطبقات، مع قاعدة البيانات، كما تحدثنا في المقالة السابقة بعنوان [هام في تصميم تطبيقات قواعد البيانات] رابط المقال

عن أهمية فصل الطبقات في العمل. لذلك سنكمل بشكل مبسط ما بدأناه في الدرس السابق، ولاحقاً بإذن الله سنتوسع بشكل أكبر لنشمل أكثر عدد من المفاهيم، التي من الممكن أن تواجهنا في تطبيقاتنا.

لنبدأ.

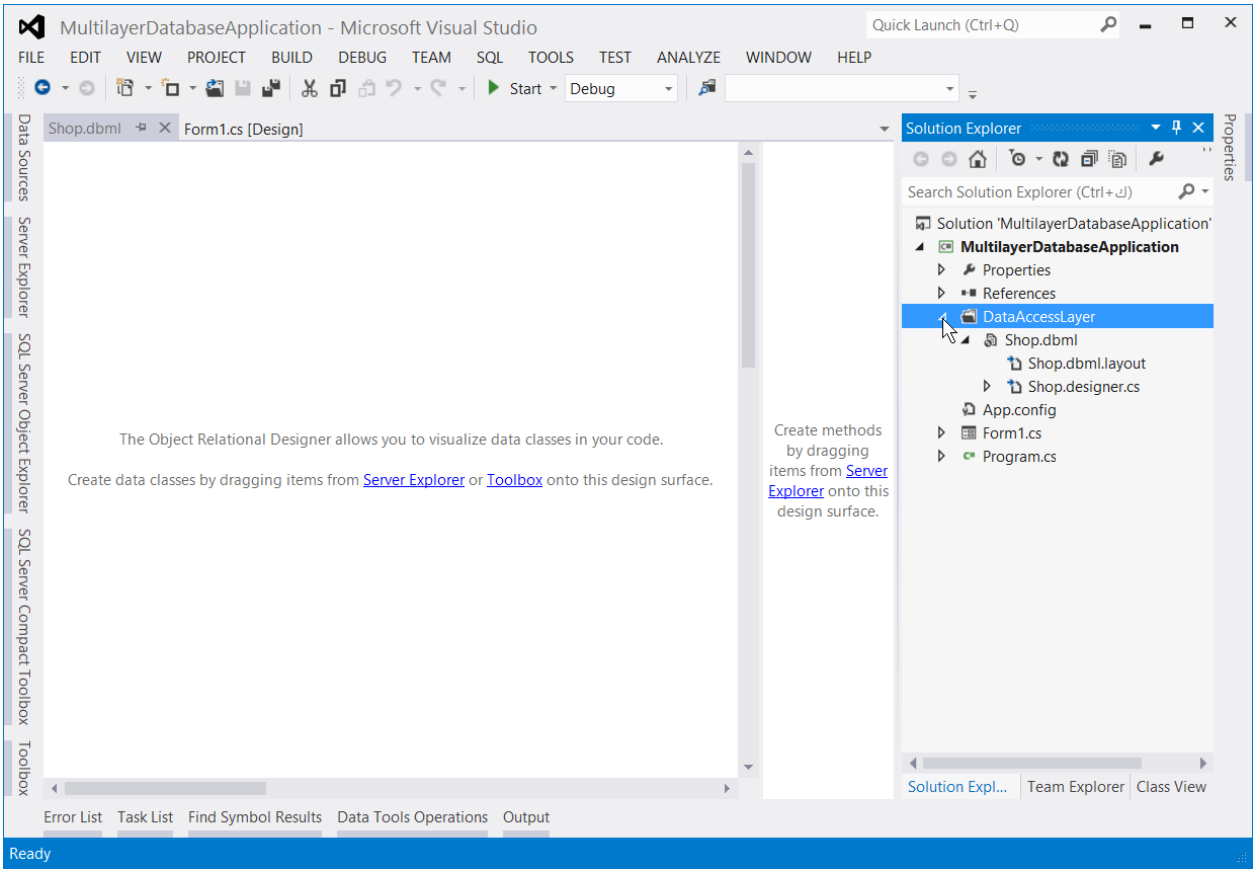
سوف نعتمد في هذا الدرس على قاعدة بيانات صغيرة إسمها Shop قمنا بعملها في درس سابق، يمكنك إنشاءها بسرعة من خلال عمل قاعدة بيانات جديدة بإسم Shop وإضافة جدول بإسم Category يحوي على حقل رئيسي Id وحقل Name من نوع nvarchar. أو يمكنك مراجعتها بسرعة من المقال بعنوان [Entity Framework with WPF | Lesson 01] رابط المقال.

وسوف نعتمد في صناعة طبقة الوصول للبيانات على تقنية Linq to Sql Classes المقدمة من ميكروسوفت.

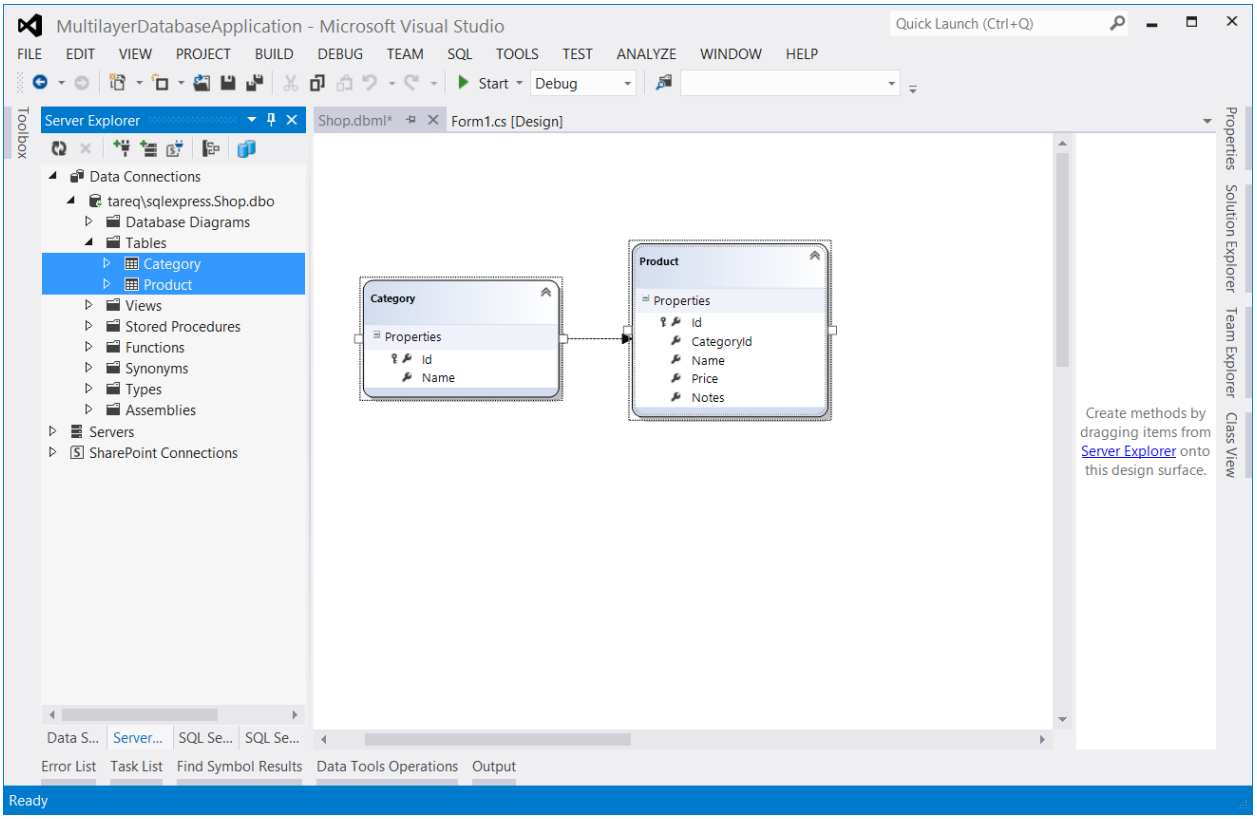
1. صناعة طبقة الوصول للبيانات DataAccessLayer

نقوم بفتح Visual Studio ونقوم بعمل مشروع جديد من نوع Windows Forms Application أو WPF، لا يهم فالعمل منفصل الطبقات كما تحدثنا. نسمي المشروع MultilayerDatabaseApplication

بعد أن أنشأنا المشروع نقوم الأن بإضافة مجلد جديد للمشروع ونسميه DataAccessLayer، ثم نضغط بالزر اليمين عليه ونضيف عنصر جديد Add New Item، ونختار LINQ To SQL Classes ونقوم بتسميته Shop، بعدها سيقوم Visual Studio بعرض المصمم Designer الخاص بها. حالياً سيكون لديك الشكل التالي:



الأن نحن بحاجة للربط مع قاعدة البيانات، بعد أن نكون قد جهزنا قاعدة البيانات، نقوم بإضافة إتصال معها من خلال Server Explorer، وبعد أن يظهر الإتصال، نقوم بفتح مجلد قاعدة البيانات، ثم نفتح مجلد Tables ونقوم بسحب الجدولين Product, Category إلى المساحة البيضاء الخاصة بالمصمم.



هكذا نكون قد انتهينا من صناعة طبقة الوصول للبيانات، ويجدر الإشارة هنا إلى أهمية التقنيات الحديثة في صناعة هكذا طبقات فهي تقوم تلقائيا بصناعة الكلاسات الخاصة بالجدوال، وتنظم العلاقات بين الجداول، فالعلاقة بين الجدولين ستتحول لعمليات ضمن كل كلاس، فمثلا نعلم أن كل Product تابع ل Category معين وبالتالي سنجد في كلاس Product تعريف ل Object من نوع Category. بالطبع تكمن فائدة عظمى هنا، مثلاً توفر علينا الكثير من علميات Join ضمن الQueries وبالطبع لا يخفى على أحد ما أقصد.

2. صناعة طبقة الضوابط والقواعد Business Logic Layer

في هذه الطبقة سنعرّف القواعد والضوابط للعمل، وهذه الطبقة ستربط طبقة الوصول للبيانات بطبقة العرض، لذلك تعتبر أهميه هذه الطبقة بكونها تعرّف سلوك البيانات والضوابط عليها. وترسلها للعرض.

نقوم بإضافة مجلد جديد للمشروع ونسميه BusinessLogicLayer، ثم نقوم بإضافة كلاس جديد إلى هذا المجلد ونسميه ShopBusinessLogic.

نقوم الأن بتعريف Object من نوع طبقة الوصول للبيانات والذي سيكون إسمه ShopDataContext، وكون طبقة الوصول للبيانات موجود ضمن مجلد أخر بالتالي تعتبر ضمن namespace مختلف، يجب علينا إضافته للتمكن من تعريف Object منها، نقوم بإضافته كالتالي:

```
using MultilayerDatabaseApplication.DataAccessLayer;
```

وسيكون لدينا شكل الكلاس كالتالي:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MultilayerDatabaseApplication.DataAccessLayer;

namespace MultilayerDatabaseApplication.BusinessLogicLayer
{
    class ShopBusinessLayer
    {
        ShopDataContext db;

        public ShopBusinessLayer()
        {
            db = new ShopDataContext();
        }
    }
}
```

الأن أصبح بإمكاننا التكلم مع طبقة الوصول للبيانات.

لذلك نقوم بتعريف العمليات على قاعدة البيانات هنا بشكل منضبط، سنقوم بتعريف الدالة الخاصة بجلب بيانات جدول Category من قاعدة البيانات.

```
public List<Category> GetCategories()
{
     var query = from category in db.Categories select category;
     return query.ToList();
}
```

إذا أردنا أي شرط معين على جلب البيانات نقوم بتعريفه ضمن هذه الميثود، ضمن الكلاس ShopBusinessLogic.
وبالتالي فصلنا قواعدنا عن مفهوم التحدث مع قاعدة البيانات.

الآن سنقوم بكتابة الدالة الخاصة بإضافة Category جديد على قاعدة البيانات، وسنقوم بالتحقق من شرطين:

1. أن لا يكون إسم الـ Category فارغ.

2. أن لا يكون إسم الـ Category موجود مسبقاً في قاعدة البيانات.

الشرط الأول بسيط، أما الشرط الثاني بحاجة للتأكد منه، لذلك نحن بحاجة لدالة جديدة تقوم بالتأكد من وجود Category أو لا، تظهر معنى ضوابط وقواعد العمل في هذه الدالة نوعا ما، لذلك سيكون شكلها على الشكل التالي:

```
private bool IsCategoryExist(string name)
{
    var query = from c in db.Categories where c.Name == name select c;
    return (query.Count() > 0);
}
```

قمنا بعمل query على قاعدة البيانات، إذا كانت النتيجة فارغة فإن عدد البيانات المرجع يساوي صفر، بالتالي الدالة سترجع False إي إن الـ Category غير موجود.

الآن أصبح بإمكاننا تطبيق دالة إضافة Category جديد إلى قاعدة البيانات والتي سيكون شكلها على الشكل التالي:

```
public void AddCategory(Category category)
{
   if (category.Name == "")
         throw new Exception("Please enter the category name");
   if (IsCategoryExist(category.Name))
         throw new Exception("Category is already exist! Please try another name");
```

```
    db.Categories.InsertOnSubmit(category);
    db.SubmitChanges();
}
```

قمنا هنا بالتحقق من بعض الضوابط، في حال وجود خلل معين ستقوم الدالة برمي خطأ throw exception، قمنا بتزويده برسالة معينة يفهمها المستخدم، وفي حال كانت البيانات صحيحة والضوابط موافقة للبيانات، نقوم بإستدعاء الدالة الخاصة بإضافة Category إلى قاعدة البيانات، ثم نقوم بحفظ العمل.

طبعاً طريقة الإضافة والحفظ، تعتمد على طبقة الوصول للبيانات، لذلك لو قمنا بتغيير تلك الطبقة إلى أي طبقة أخرى، سيكون التغيير في BusinessLogic يسير، بما يوافق أسماء الدوال الجديدة بالطبقة الجديدة، وتبقى الضوابط وآليات التحقق نفسها.

يمكننا الآن إضافة أي دالة إلى هذا الكلاس وعمل التحقق المناسب بنفس الآلية، أي فصله عن طبقة العرض فمثلاً نلاحظ أن طريقة رمي الخطأ لبرنامج المستخدم هذه، نستطيع معالجتها في أي مشروع من نوع أخر مثل Windows Forms Application, WPF, ASP.NET, Silverligh حيث أن لكل مشروع آلية عرض معينة، وما يهمنا هو إلتقاط الخطأ فقط من هذه الطبقة، وهذا ما عملناه بالفعل.

3. صناعة طبقة العرض Presentation Layer

كوننا قمنا بعمل المشروع من نوع Windows Forms Application بالتالي طبقة العرض، جاهزة للعمل ولسنا بحاجة لأي إضافة أخرى.

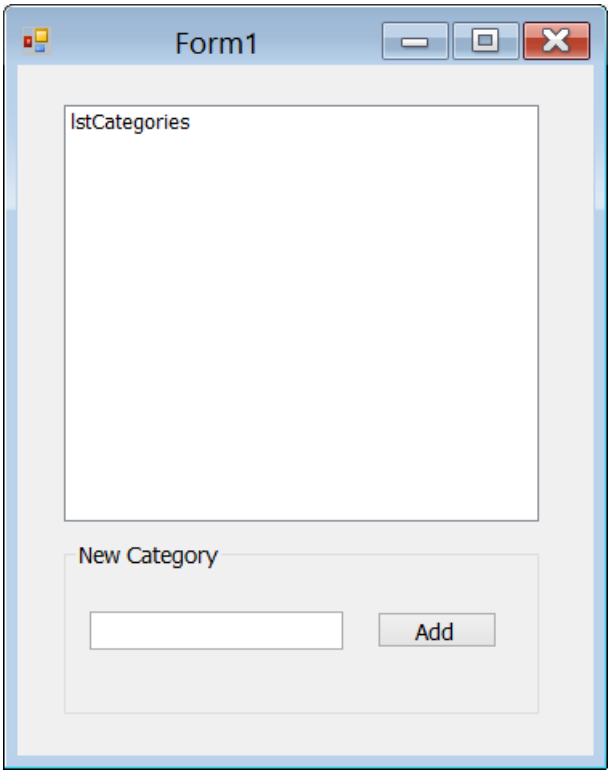
لكن يجدر الإشارة هنا إلى أننا قمنا بصناعة الطبقات كل طبقة في مجلد مختلف، ولكن لإمكانية إعادة استخدام تلك الطبقات في مشاريع أخرى لاحقة، الأفضل أن نقوم بصناعة كل طبقة ضمن مشروع خاص بها من نوع Class Library مثلاً، ونقوم بعدها بإضافة هذه المشاريع إلى مشروع العرض النهائي عن طريق Add Reference.

الآن نذهب إلى Form1.cs ونقوم بإضافة بعض العناصر إلى النافذة، كما في الشكل التالي مع مراعاة التسميات التالية:

1. ListBox نقوم بتسميتها lstCategories ونقوم بوضع خاصية DisplayMember إلى Name لنتمكن من عرض إسم الـ Category ضمنها، وبدونها ستعرض إسم الكلاس كامل.

2. TextBox نقوم بتسميته إلى txtName حيث سنقوم بإدخال إسم الـ Category الجديد ضمنه.

3. Button نقوم بتسميته إلى btnAdd ونضع خاصية Text إلى Add فيكون لدينا الشكل التالي:



الأن نقوم بالنقر مرتين على هذه النافذه لفتح الحدث Form_Load ضمن ملف Form.cs

ضمن هذا الكلاس سنتحدث مع object من نوع الطبقة الثانية BusinessLogicLayer كونه على دراية بضوابط العمل وقادر على التحدث مع قاعدة البيانات.

لذلك نقوم بتعريف Object من نوع ShopBusinessLogic، ومن ثم نقوم بربط عناصر ListBox مع مجموعة الـ Categories القادمة من قاعدة البيانات، فيكون لدينا شكل الكلاس Form1.cs كالتالي:

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MultilayerDatabaseApplication.BusinessLogicLayer;

namespace MultilayerDatabaseApplication
{
    public partial class Form1 : Form
    {
        ShopBusinessLogic logic;

        public Form1()
        {
            InitializeComponent();
            logic = new ShopBusinessLogic();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            lstCategories.DataSource = logic.GetCategories();
        }
    }
```

```
}
```

ولا ننسا أنا قمنا بإضافة فضاء الأسماء الخاص بالطبقة الثانية، كونها بمجلد مختلف كما حصل معنا بالطبقة الأولى، ونضيف فضاء أسماء الطبقة الأولى لكي نتمكن من التعامل مع Classes المولدة ضمنها.

الآن أصبح لدينا ألية جلب البيانات من قاعدة البيانات بكل سهولة بواسطة الطبقة الثانية.

ماذا عن إضافة Category جديد لقاعدة البيانات؟ نعم. نقوم بالنقر المزدوج على الزر btnAdd لفتح الحدث الخاص به Button_Click. سنقوم بكتابة كود إضافة Category جديد بداخله. ونحتاج لكتابته ضمن try catch، لإلتقاط أي خطأ ممكن حدوثه وعرضه للمستخدم. فيكون لدينا حدث هذا الزر كالتالي:

```
private void btnAdd_Click(object sender, EventArgs e)
{
    try
    {
        Category category = new Category() { Name = txtName.Text };
        logic.AddCategory(category);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

هكذا نكون قد صنعنا ثلاث طبقات منفصلة في العمل، وبإمكاننا الأن إضافة التعديل والإضافة على المشروع بكل سهولة بسبب فصل الطبقات عن بعضها.

...

لكم كل الشكر والتحية

م. طارق جهاد الجباوي

tareqjehad@yahoo.com

2013/11/29