C# And SQL Server

ذخیره، ویرایش و حذف در SQLServer توسط C# و LINQ

ایجاد پروژه Setup

توسط: مهندس یوسف امیری

سطح: متوسط

ویرایش: اردیبهشت **1389**

در این مقاله، با مفاهیم زیر آشنا می شوید:

آموزش نحوه ارتباط با SQLServer در سی شارپ
ذخیره، تغییر و حذف رکوردهای جدول توسط C#
استفاده از پروسیجر های ذخیره شده در C#
روش ارسال پارامتر به Stored Procedures
استفاده از LINQ برای ذخیره و بازیابی داده های جدول SQL Server
نحوه ارسال متغیر و فیلد های جدول و گراید از فرمی به فرم دیگر
ارتباط گراید (DataGridView) با جدول توسط LINQ
به روز رسانی گراید پس از تغییر در جدول بانک اطلاعاتی
مراحل ایجاد یک پروژه دیتابیس توسط C#
ایجاد پروژه Setup در ویژوال استودیو

معرفـی

در این PDF آموزشی قصد دارم برای دانشجویانی که می خواهند توسط Visual Studio 2008 و بالاتر و با استفاده از زبان C#، با SQL Server ارتباط برقرار کرده و در آن به ذخیره و بازیابی اطلاعات بپردازند، از ابتدا و بصورت گام به گام، مراحل را توضیح دهم. هدف از این بحث توانایی ایجاد اولین پروژه Databaseی است. اما در اینجا قصد ندارم از ADO.NET و کدنویسی های خسته کننده آن برای ارتباط با بانک اطلاعاتی استفاده کنم. بلکه از طریق LINQ همین کار را بصورت کاملا شیء گرا انجام خواهیم داد.

همانطور که می دانید از لحاظ دسترسی به بانک اطلاعاتی، دو نوع راه حل وجود دارد:

1. ایستگاه کاری
2. سرور – کلاینت

در حالت اول که موضوع این آموزش است، بصورت یک فایل با بانک اطلاعاتی SQL Server برخورد می شود و اغلب برای پروژه های دانشجویی کافیست.

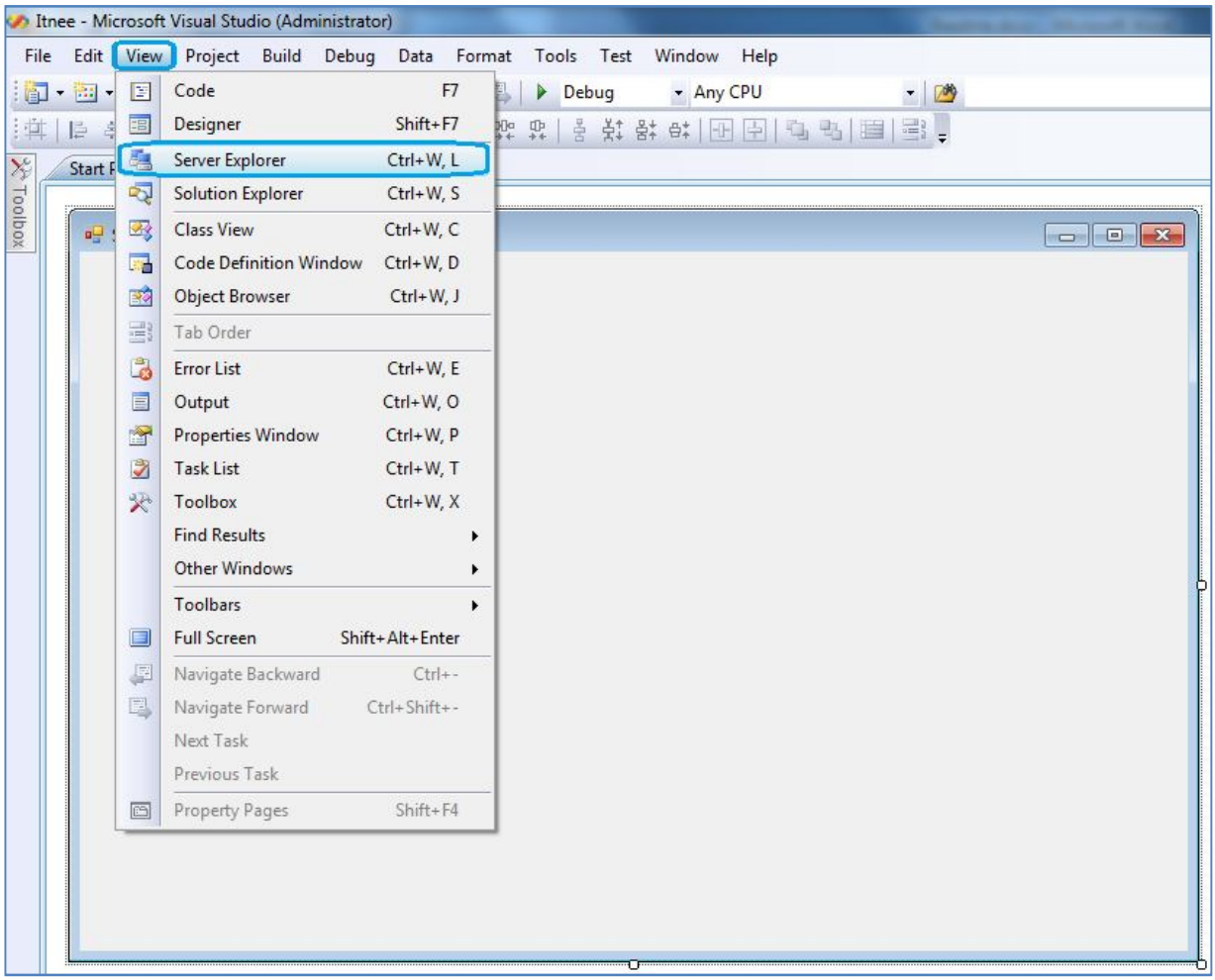
در حالت دوم، SQL Server را روی یک سیستم Server نصب کرده و فایلهای بانک اطلاعاتی را به آن متصل (Attach) می کنیم یا می سازیم. سپس در قسمت Security نرم افزار مدیریت SQL Server یا همان SQL Server Management Studio کاربران و سطوح دسترسی آنها به بانک اطلاعاتی را تعریف می کنیم. سپس در کامپیوتر کلاینت شروع به برنامه نویسی کرده و هر زمان که لازم شد توسط رابط ویژوال استودیو با سرور و بانک اطلاعاتی، توسط کاربر تعریف شده، ارتباط برقرار می کنیم این ارتباط توسط یک رشته کد به نام Connection String انجام می گیرد که خود ویژوال استودیو آنرا می سازد. سپس برای برنامه خود Setup می سازیم (توسط Visual Studio) و آنرا در کلاینتهای دیگر نیز نصب می کنیم. در واقع نمونه های برنامه، با سرور ارتباط برقرار کرده و بطور مشترک نسبت به انجام چهار عمل اصلی روی داده ها (ایجاد، ویرایش، حذف و جستجو) اقدام می کنند.

رابط بین زبان برنامه نویسی و بانک اطلاعاتی

برای ارتباط با بانک اطلاعاتی نیاز به یک رابط داریم که جدولها، پروسیجرهای ذخیره شده و سایر اشیاء بانک اطلاعاتی را در خود نگهداری و به روز رسانی کند. کار با رابطهای قدیمی مانند DAO، ADO و ADO.NET بسیار ناخوشایند و اغلب همراه با خطا و پیچیده بود و همان نقطه ای بود که تازه کارها معمولا پس از رسیدن به آن، دیگر ادامه نمی دادند! اما مایکروسافت سرانجام در NET 3.0. نگاه شیء گرای خود را با توسعه LINQ، به تکامل تحسین برانگیزی رساند و این بخش از کار، اکنون جزو لذت بخش ترین مراحل توسعه نرم افزارهای مبتنی بر دیتابیس است.

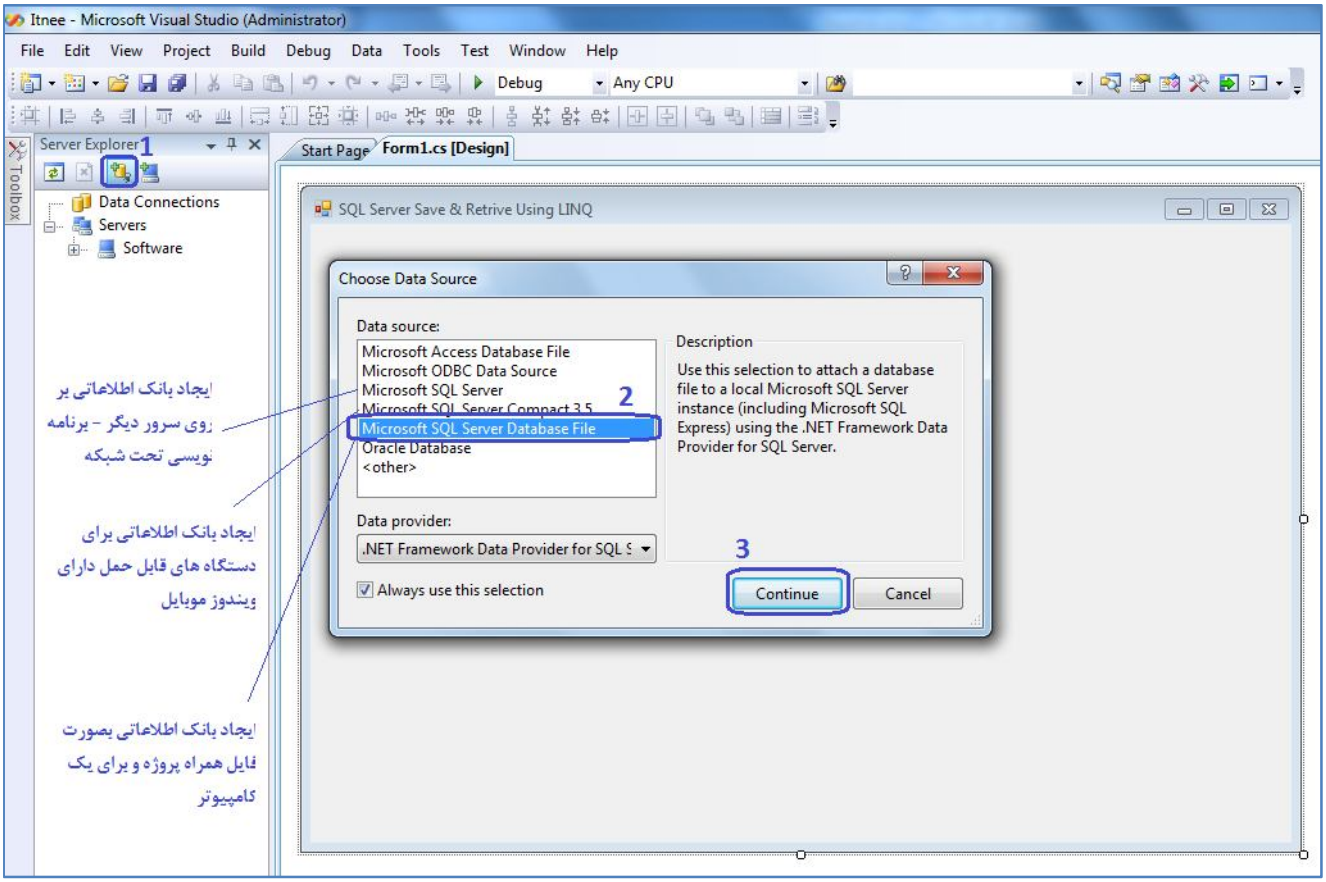
پروژه جدیدی در Visual Studio ایجاد کنید.

پانل Server Explorer را برای ایجاد بانک اطلاعاتی فعال کنید. (شکل 1)



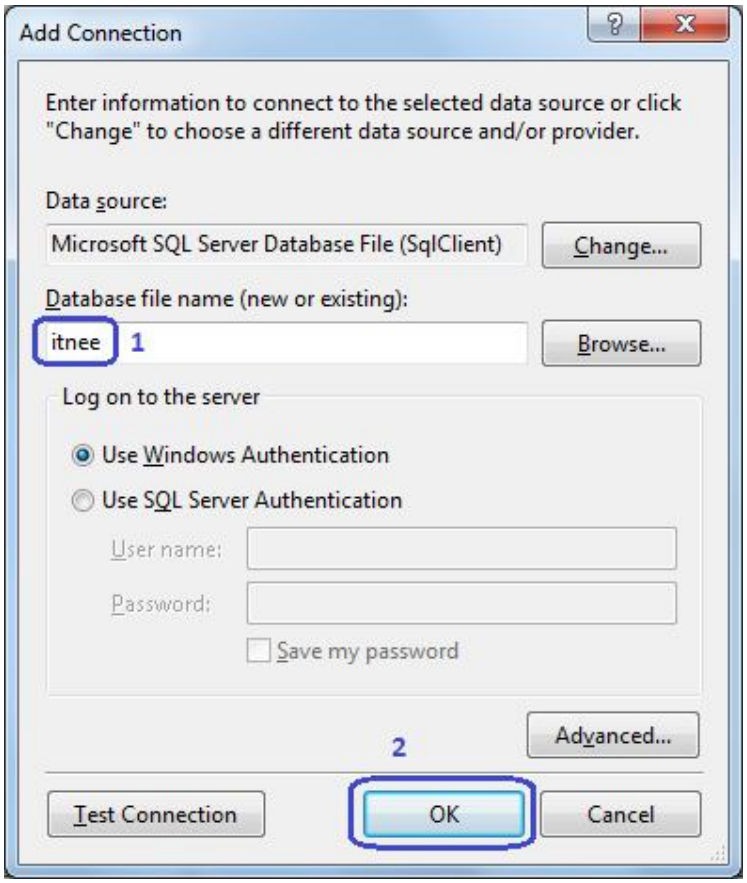
شکل 1 - فعال کردن پانل **Server Explorer** برای ایجاد و مدیریت بانک اطلاعاتی

برای ایجاد بانک اطلاعاتی و Connection String، آیکن Connect to Database را کلیک کنید (شکل 2)

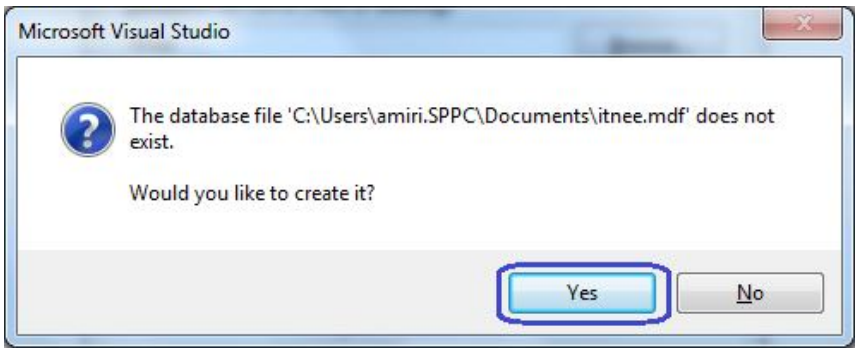


شکل 2- انتخاب نوع بانک اطلاعاتی

مراحل ایجاد بانک اطلاعاتی را مانند شکلهای3 و 4 انجام دهید :

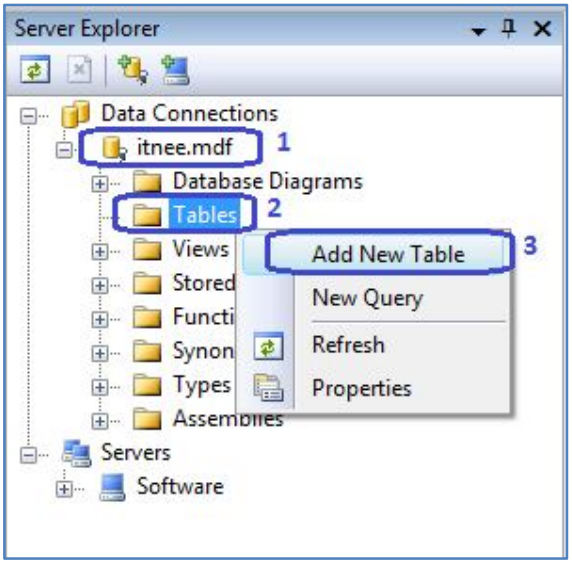


شکل 3 - انتخاب نام بانک اطلاعاتی

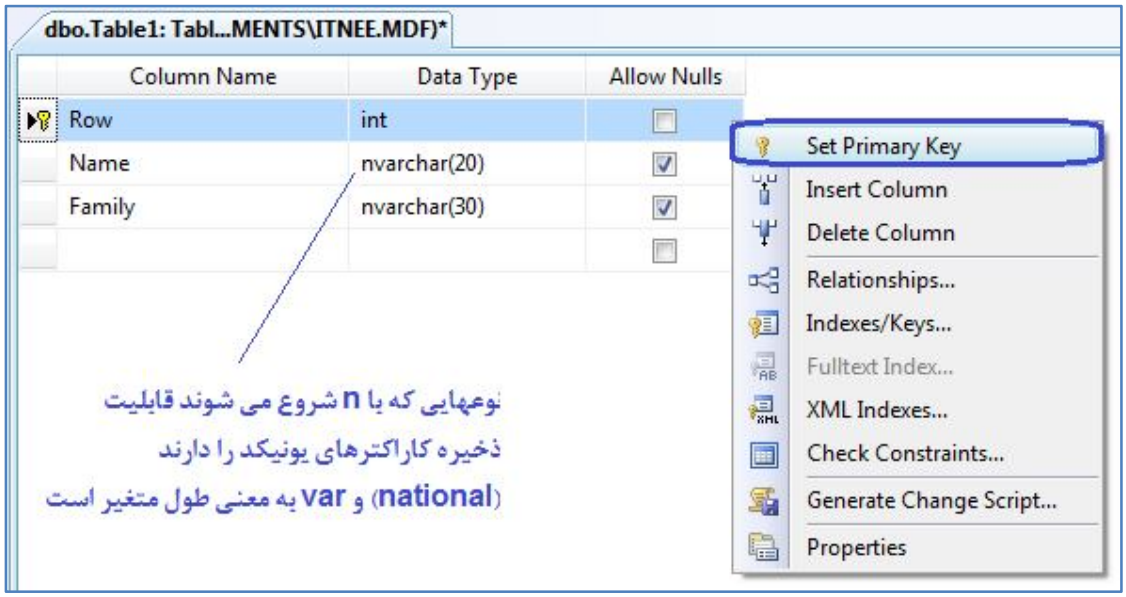


شکل 4 - ایجاد بانک اطلاعاتی جدید

پس از این مرحله، بانک اطلاعاتی ایجاد شده و به پانل Server Explorer اضافه می شود (شکل 5) آنرا از ساختار درختی باز کرده و با کلیک راست روی Table، اولین جدول را ایجاد می کنیم. در هنگام وارد کردن نام فیلدها، با کلیک راست روی اولین فیلد، آنرا به عنوان کلید انتخاب می کنیم. (شکل 6)

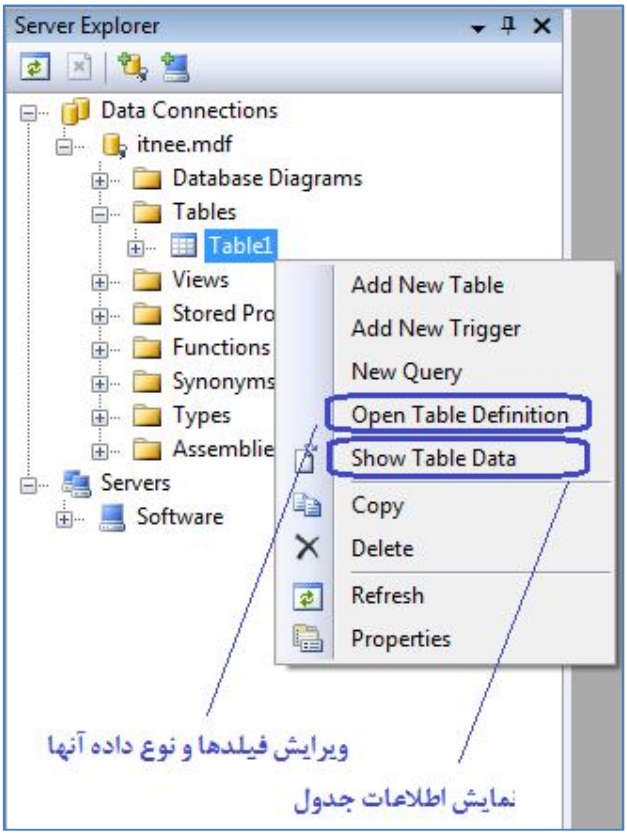


شکل 5 - باز کردن بانک اطلاعاتی و ایجاد جدول در آن



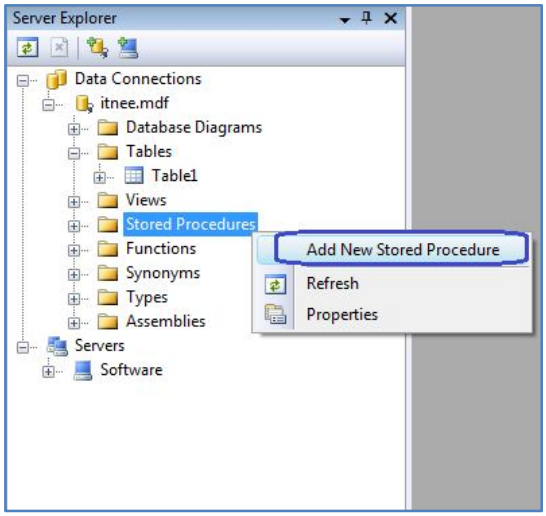
شکل 6 - جزئیات ایجاد جدول

جدول را با همان نام پیش فرض ذخیره می کنیم. برای ویرایش جدول یا دیدن اطلاعات آن، روی نام جدول کلیک راست کرده و گزینه مربوطه را انتخاب می کنیم. (شکل 7)



شکل 7 - انجام تغییرات روی ساختار جدول و یا نمایش داده های آن

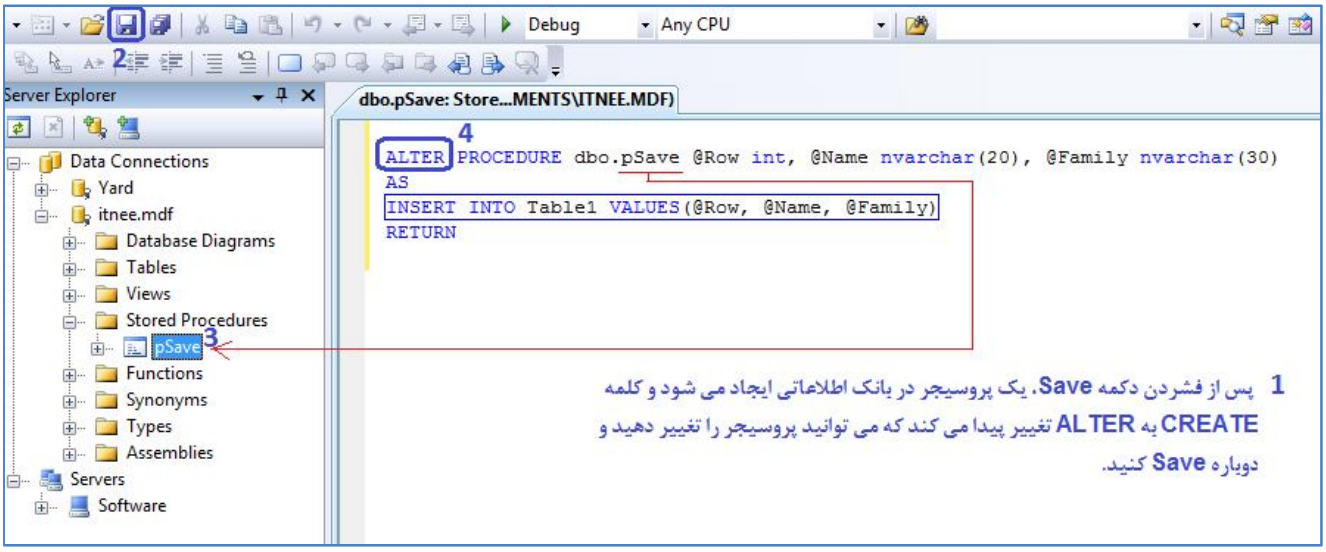
حال نوبت به ذخیره، ویرایش و حذف در جدول می رسد. ما این اعمال را توسط پروسیجرهای ذخیره شده انجام می دهیم. پروسیجرهای ذخیره شده بهترین حالت را از نظر سرعت تراکنش، کد نویسی چند لایه، فراهم کردن امکان تغییرات سریع در برنامه و ... ایجاد می کنند. برای این کار، همانند ایجاد یک جدول، این بار روی Stored Procedures کلیک راست می کنیم. (شکل 8)



شکل 8 - ایجاد پروسیجر ذخیره شده برای انجام اعمال ذخیره، ویرایش و حذف از جدول بانک اطلاعاتی

برای نوشتن پروسیجر ذخیره، همه کدهای پیش فرض را حذف کرده و کد زیر را جایگزین آن کنید. سپس دکمه Save را بزنید. با این کار یک پروسیجر به نام pSave ایجاد می شود. (شکل 9)

```
CREATE PROCEDURE dbo.pSave @Row int, @Name nvarchar(20), @Family nvarchar(30)
AS
INSERT INTO Table1 VALUES(@Row, @Name, @Family)
RETURN
```



شکل 9 - ایجاد پروسیجرهای ذخیره شده

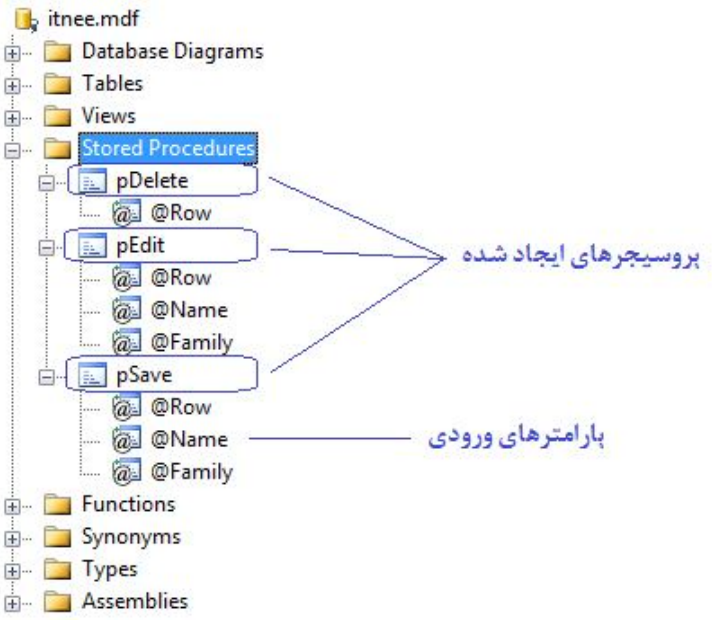
برای نوشتن پروسیجر ویرایش، مانند مرحله قبل یک پروسیجر جدید ایجاد کنید و کد زیر را جایگزین آن کنید. سپس دکمه Save را بزنید. با این کار یک پروسیجر به نام pEdit ایجاد می شود.

```
CREATE PROCEDURE dbo.pEdit @Row int, @Name nvarchar(20), @Family nvarchar(30)
AS
UPDATE Table1 SET Name=@Name, Family=@Family
WHERE Row=@Row
RETURN
```

همانطور که مشاهده می کنید پارامترهای ورودی پروسیجر ذخیره و ویرایش با هم یکسان هستند (که البته می توانید در پروسیجر ویرایش فقط فیلدهایی را به عنوان پارامتر ورودی تعیین کنید که تغییرات دارند) . در خط 4 کد ویرایش، از شرط استفاده شده است. در اینجا *فیلد کلید* اولیه را می نویسیم

برای نوشتن پروسیجر حذف فقط به *فیلد کلید* اولیه نیاز داریم. پروسیجر جدیدی ایجاد کنید و کد زیر را در آن بنویسید و دکمه Save را بزنید:

```
CREATE PROCEDURE dbo.pDelete @Row int
AS
DELETE FROM Table1
WHERE Row=@Row
RETURN
```



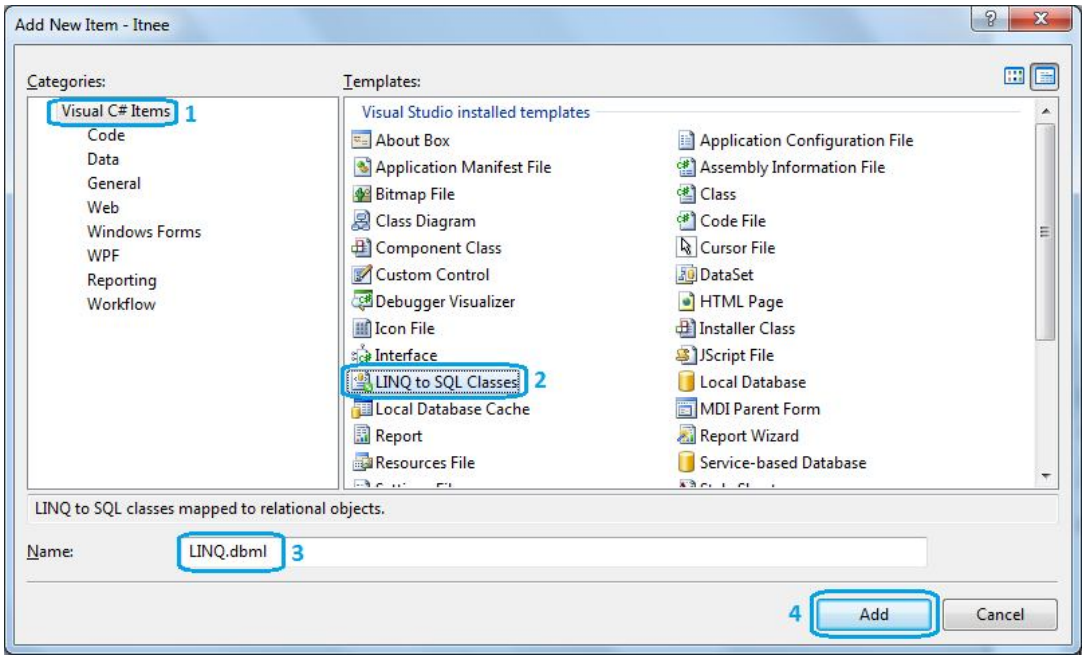
شکل 10 - ایجاد پروسیجرهای ذخیره شده

ایجاد LINQ To SQL

برای ایجاد LINQ to SQL به منوی زیر بروید:

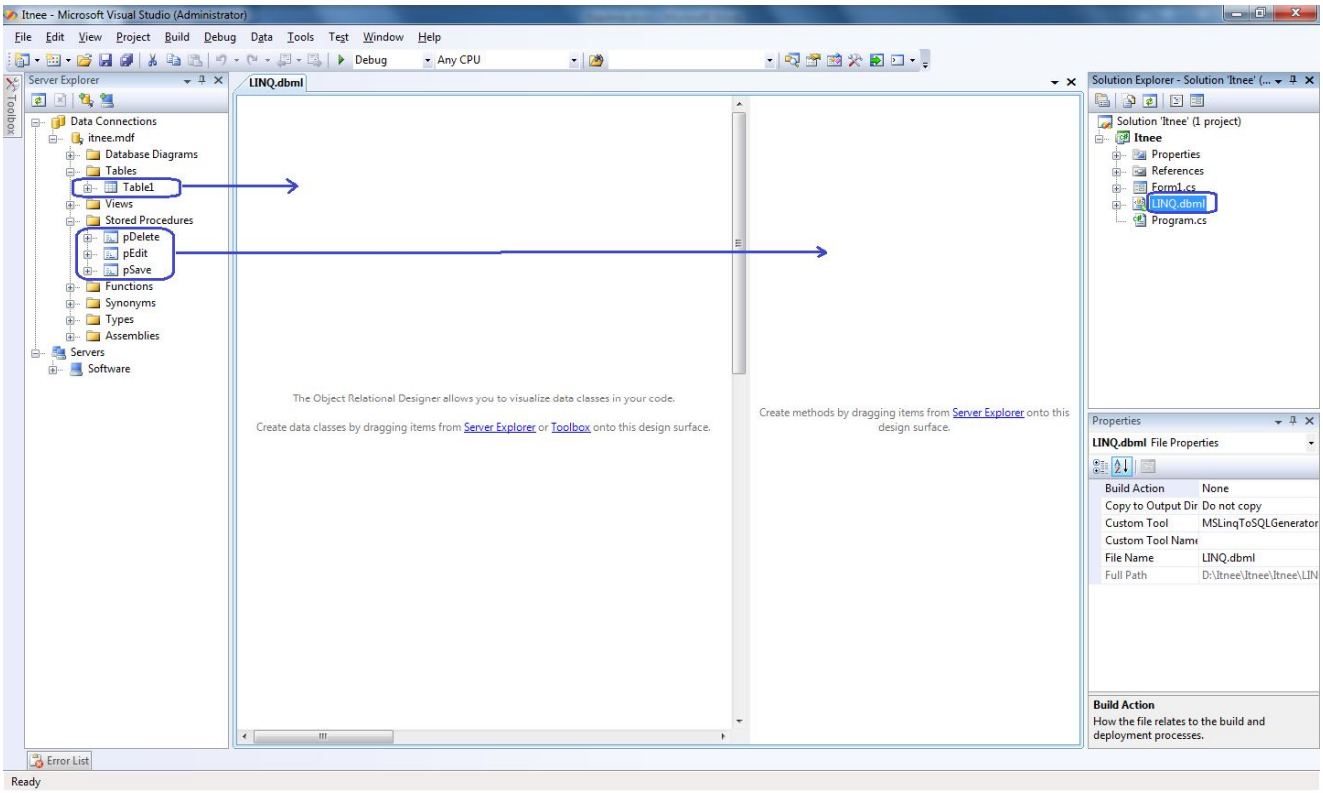
```
Project > Add New Item…
```

سپس گزینه LINQ to SQL Classes را انتخاب کنید و مطابق شکل 11 نام LINQ را به آن بدهید.

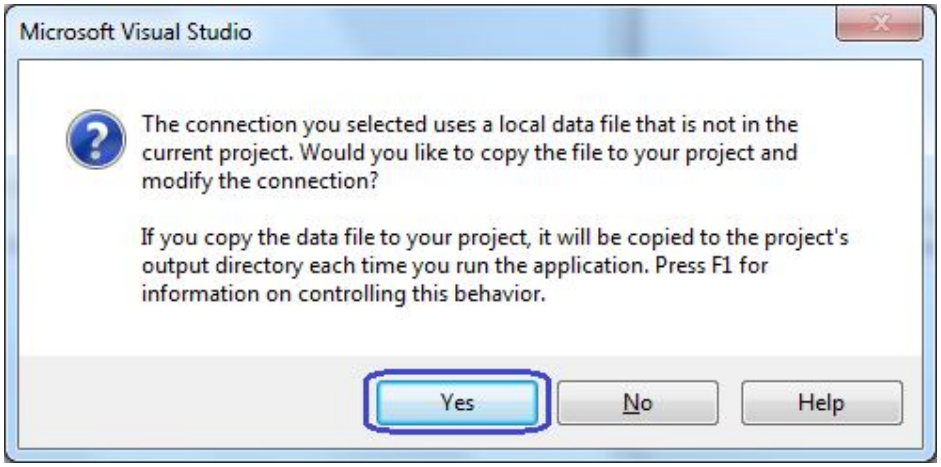


شکل 11 - ایجاد LINQ

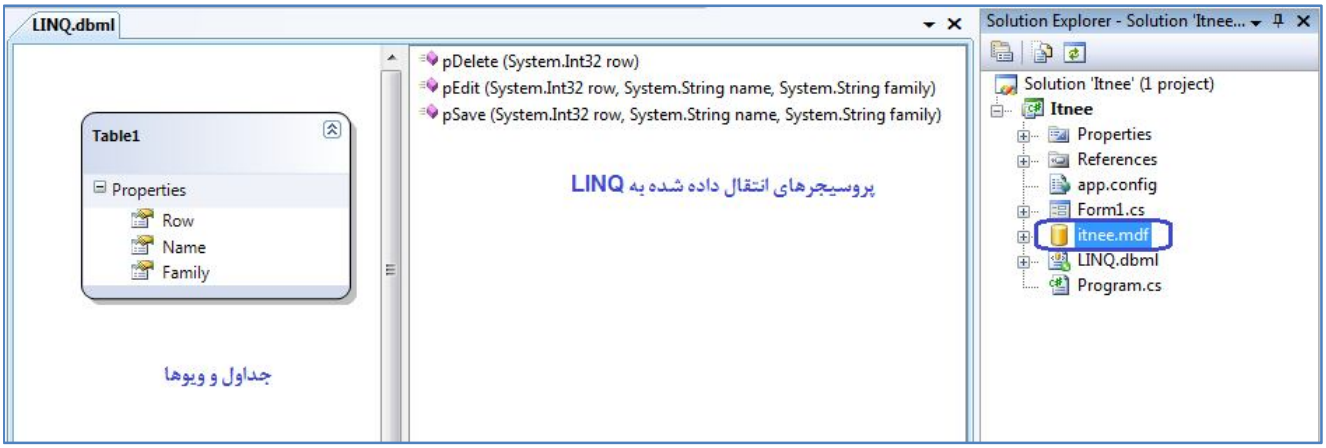
پس از ایجاد محیط LINQ، چهار شیء ایجاد شده در بانک اطلاعاتی (Table1, pSave, pEdit, pDelete) را به پانلهای آن درگ کنید. پانل سمت چپ برای جداول و ویوها و پانل سمت راست برای پروسیجرهای ذخیره شده. (شکل 12) ضمن انتقال اشیاء به محیط LINQ، ویژوال استودیو بانک اطلاعاتی را به مسیر پروژه کپی کرده و Connection String لازم را نیز می سازد. (شکل 13)



شکل 12 - محیط LINQ در حالت اولیه



شکل 13 - تاییدیه انتقال بانک اطلاعاتی به مسیر پروژه و ایجاد Connection String برای ارتباط با آن

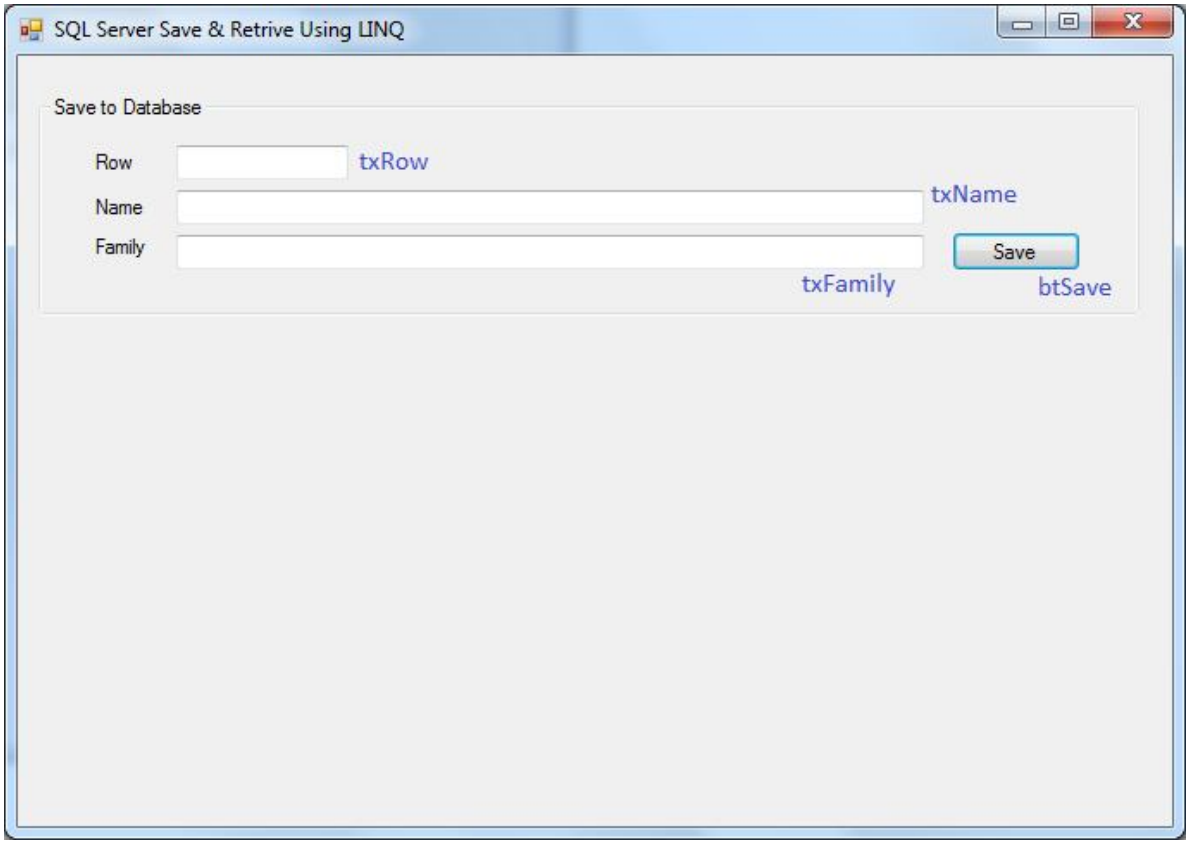


شکل 14 - اضافه شدن بانک اطلاعاتی به پروژه، هنگام اضافه کردن اشیاء آن به محیط LINQ

نکته: پس از هر تغییر در جداول، ویوها و یا پروسیجرهای ذخیره شده در بانک اطلاعاتی، باید LINQ را نیز آپدیت کرد. برای این کار کافیست آن شیء را از LINQ حذف کرده و دوباره از بانک اطلاعاتی به آن درگ کنید و فایل LINQ.dbml را مجددا ذخیره کنید.

ذخیره در بانک اطلاعاتی

فرم را مطابق شکل 15 طراحی و نام گذاری کنید:



شکل 15 - طراحی بخش Save روی فرم

دوبار روی دکمه Save کلیک کنید و کد زیر را برای آن بنویسید:

```
var db = new LINQDataContext();
db.pSave(Convert.ToInt32(txRow.Text), txName.Text, txFamily.Text);
```

در واقع با این دو خط کد، شما موفق به ذخیره یک رکورد در بانک اطلاعاتی شده اید!

خط اول یک متغیر از شیء LINQ ایجاد می کند و خط دوم پروسیجر pSave را از LINQ فراخوانی کرده و مقادیر کنترلهای روی فرم را به آن ارسال می کند تا در جدول ذخیره شود.

البته فرض بر این است که شما چک می کنید، کاربر همه مقادیر روی فرم را به درستی وارد کرده است، و برای ردیف (txRow) که کلید جدول است، یک مقدار عددی غیر تکراری وارد شده است. و پس از ذخیره در جدول همه TextBoxها را خالی می کنید.

مقایسه با ADO.NET و مقدار کاری که آنجا لازم بود!

ADO.NET

```
//using System.Data;
//using System.Data.SqlClient;

Cn = new SqlConnection(@"Server=.\SQLEXPRESS;AttachDbFilename=" + Application.StartupPath +
@"\Itnee.mdf;Integrated Security=True;User Instance=True");
Cn.Open();

SqlCommand Cm = new SqlCommand("", Cn);
Cm.CommandText = "INSERT INTO Table1(Row, Name, Family) VALUES(@Row,@Name,@Family)";
Cm.Parameters.AddWithValue("@Row",lbCode.Text);
Cm.Parameters.AddWithValue("@Name", txName.Text);
Cm.Parameters.AddWithValue("@Family", txFamily.Text);
Cm.ExecuteNonQuery();
```

نمایش محتویات یک جدول از بانک اطلاعاتی

برای ملموس بودن عمل ذخیره، بهتر است به جای پیامهای ثبت موفقیت آمیز و اعمالی از این قبیل، نتیجه را بلافاصله در یک گراید، به روز رسانی کنید. برای این کار از Toolbox و زیر مجموعه Data، روی کنترل DataGridView دوبار کلیک کنید. نام آنرا gvTable1 قرار دهید و مطابق شکل 16 خصوصیات دیگر آنرا تنظیم کنید. سپس کد زیر را به انتهای دستورات دکمه Save اضافه کنید:

```
gvTable1.DataSource = db.Table1s;
```

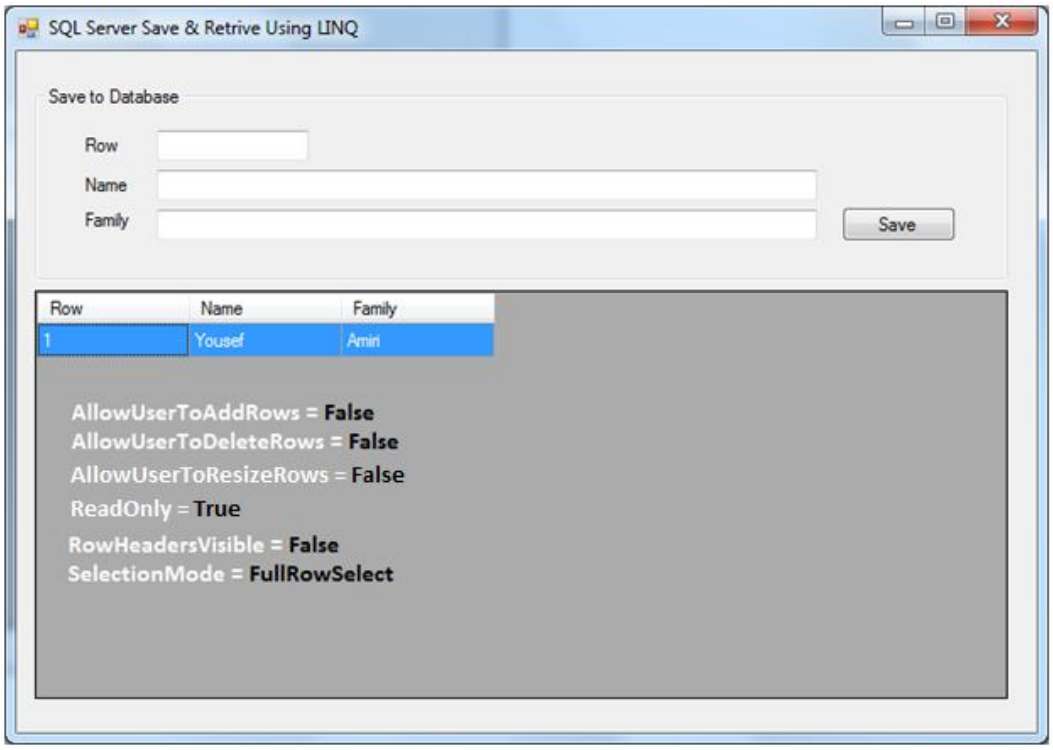
این دستور سبب می شود با هر بار ذخیره رکورد جدید در بانک اطلاعاتی، دوباره جدول Table1 به گراید بارگذاری شود. (گراید Refresh شود). بهتر است در هنگام Load شدن فرم نیز همین کد را قرار دهیم تا محتویات جدول نمایش داده شود. تا این مرحله، Form1 شامل دو رویداد زیر خواهد بود:

```
private void Form1_Load(object sender, EventArgs e)
{
      var db = new LINQDataContext();
      gvTable1.DataSource = db.Table1s;
}
```

```
private void btSave_Click(object sender, EventArgs e)
{
      var db = new LINQDataContext();
      db.pSave(Convert.ToInt32(txRow.Text), txName.Text, txFamily.Text);
      gvTable1.DataSource = db.Table1s;
}
```

نکته مهم :

در این مرحله، برنامه را اجرا کنید و چند رکورد به آن اضافه کنید، چنانچه برنامه را `Close` کرده و دوباره باز کنید می بینید هیچ کدام از رکوردهای ذخیره شده وجود ندارد! نگران نباشید، شما در حالت `Debug` هستید، در محیط ویندوز، به مسیر پروژه بروید و از پوشه `bin > Debug` فایل `Itnee.exe` را اجرا کنید، و نتیجه را ببینید، تا زمان `Setup` سازی و ایجاد پروژه کامل صبر کنید.



شکل 16 - اضافه کردن گراید به فرم برای نمایش اطلاعات جدول

ویرایش در بانک اطلاعاتی

برای اضافه کردن قابلیت ویرایش به برنامه، راه حلهای مختلفی وجود دارد. استفاده از DataSet و درگ کردن جدول آن بر روی فرم، در این حالت ویژوال استودیو همه اشیاء لازم را می سازد و به فرم اضافه می کند. کاربر در همان گراید امکان ویرایش یا حذف را هم خواهد داشت. راه حل ساده ایست اما تغییرات در بانک اطلاعاتی و یا گسترش برنامه، معمولا مشکلات گسترده ای در آینده بوجود خواهد آورد. چون باید ببینید ویژوال استویو چکار کرده است که بتوانید آنرا تغییر دهید! از طرفی قرار شد فقط از LINQ استفاده کنیم.

روش الف: با دوبار کلیک کاربر روی یک ردیف از گراید، کلیه اطلاعات آن ردیف به TextBox های روی فرم منتقل شود. دقت کنید فیلد کلید به حالت ReadOnly در فرم بارگذاری شود (txRow را به حالت ReadOnly در آورید. تا تغییر نکند.) روی فرم، علاوه بر دکمه Save، دو دکمه Update و Cancel نیز قرار دهید. در حالت ویرایش دکمه Save را غیرفعال و این دو دکمه را فعال سازید. کد دکمه Update همانند دکمه Save خواهد بود با این تفاوت که از pEdit استفاده می کنیم.

روش ب: با دوبار کلیک کاربر روی یک ردیف از گراید، اطلاعات آن ردیف به TextBox های روی فرم دیگری منتقل شود و آنجا عمل ویرایش را انجام دهیم. به دلیل جنبه آموزشی داشتن این پروژه، از روش ب استفاده می کنیم. فرمی با مشخصات زیر و با نام frEdit بسازید:

frEdit		txRow		txName		txFamily	
FormBorderStyle	FixedSingle	ReadOnly	True	Modifiers	Public	Modifiers	Public
MaximizeBox	False	Modifiers	Public				
MinimizeBox	False						
StartPosition	CenterParent						

شکل 17 - فرم ویرایش

در فرم اصلی، برای رویداد DoubleClick گراید، کد زیر را بنویسید (فرم ویرایش را فراخوانی می کند و سلولهای گراید را به کنترلهای آن می فرستد. به یاد داشته باشید که هر سه TextBox فرم frEdit خصوصیت Modifiers=Public دارند تا از فرم اصلی بتوان آنها را دید.)

```
private void gvTable1_DoubleClick(object sender, EventArgs e)
{
    if (gvTable1.RowCount > 0)
    {
        var f = new frEdit();
        f.txRow.Text = gvTable1.CurrentRow.Cells["Row"].Value.ToString();
        f.txName.Text = gvTable1.CurrentRow.Cells["Name"].Value.ToString();
        f.txFamily.Text = gvTable1.CurrentRow.Cells["Family"].Value.ToString();
        f.ShowDialog();

        var db = new LINQDataContext();
        gvTable1.DataSource = db.Table1s;   //Refresh Grid After Edit
    }
}
```

در فرم frEdit کد زیر را برای دکمه های btUpdate و btCancel بنویسید:

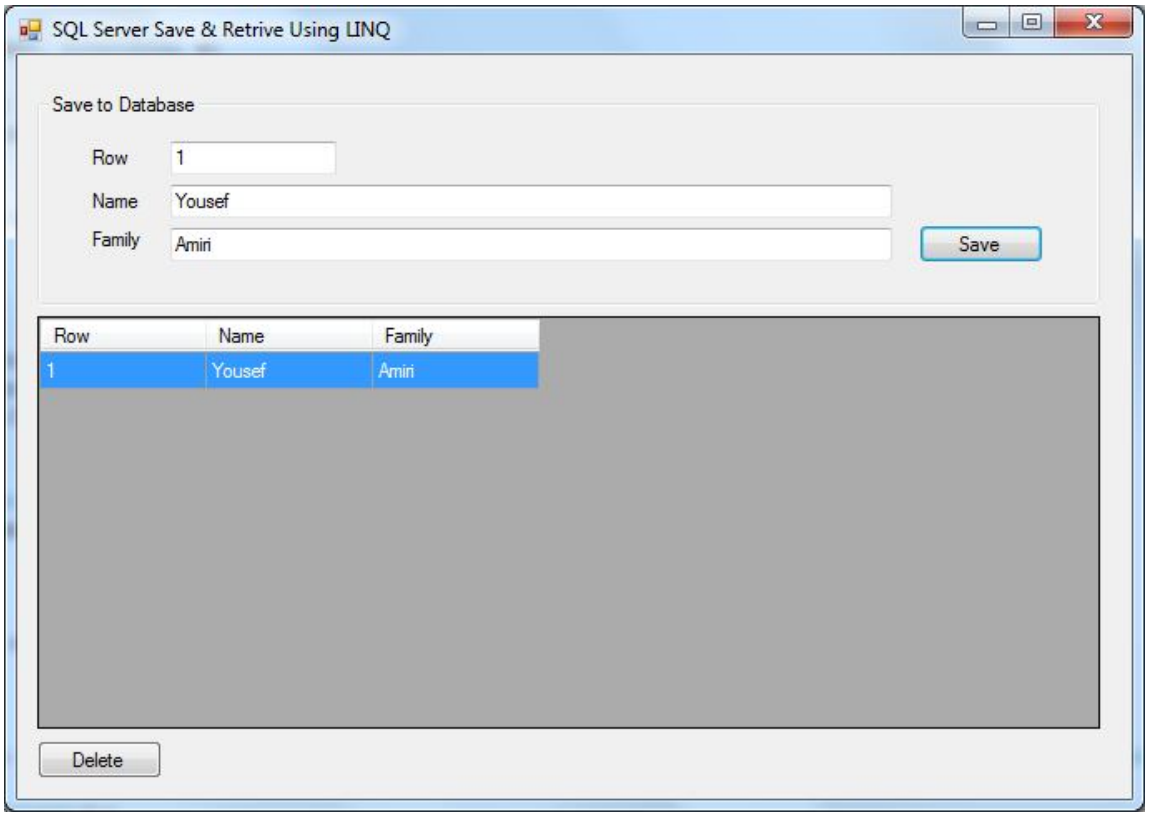
```
private void btUpdate_Click(object sender, EventArgs e)
{
    var db = new LINQDataContext();
    db.pEdit(Convert.ToInt32(txRow.Text), txName.Text, txFamily.Text);
    Close();
}

private void btCancel_Click(object sender, EventArgs e)
{
    Close();
}
```

حذف از بانک اطلاعاتی

حذف از بانک اطلاعاتی، ساده ترین مرحله است. در این حالت یک دکمه زیر گراید و با نام btDelete قرار دهید. سپس کد زیر را برای آن بنویسید:

```
private void btDelete_Click(object sender, EventArgs e)
{
    if (gvTable1.RowCount <= 0) return;
    if (MessageBox.Show("Do you want to delete?", "Delete", MessageBoxButtons.YesNo,
                        MessageBoxIcon.Question) == DialogResult.Yes)
    {
        var db = new LINQDataContext();
        db.pDelete(Convert.ToInt32(gvTable1.CurrentRow.Cells["Row"].Value));
        gvTable1.DataSource = db.Table1s;   //Refresh Grid After Delete
    }
}
```

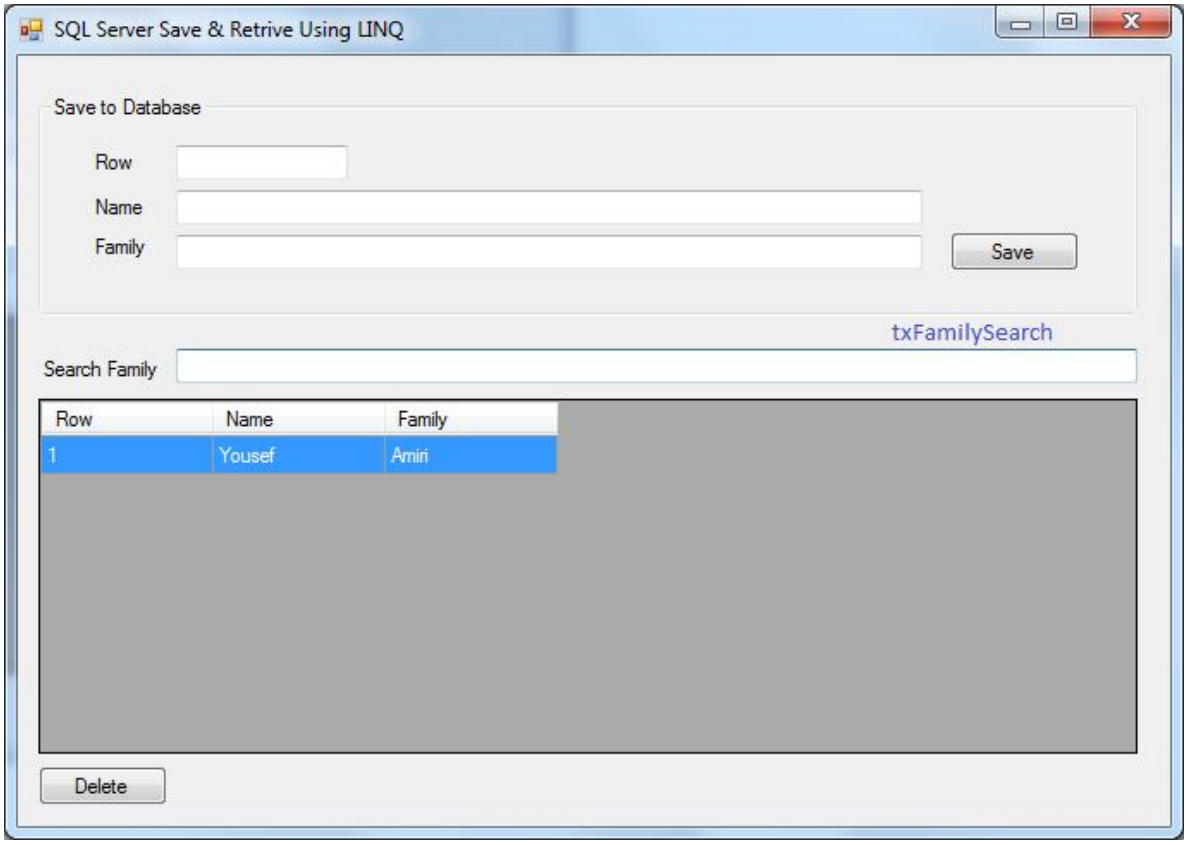


شکل 18 - اضافه کردن دکمه حذف به فرم

همانطور که دیدید LINQ قدرت و سرعت زیادی به توسعه دهندگان می دهد و هزینه نگهداری و ایجاد ویرایش های بعدی برنامه را کاهش می دهد. در این پروژه ما از هیچکدام از اشیاء ADO.NET استفاده نکردیم و با کد نویسی خیلی کمی موفق به انجام اعمال اصلی روی دیتابیس شدیم. همچنین توسعه Application و Database را در لایه های کاملا مجزا و کنترل شده انجام دادیم. (کدهای C# و SQL را ترکیب نکردیم) این کار برای پروژه های بزرگ، بسیار حائز اهمیت است.

جستجو در جدول

فراهم کردن جستجوی اطلاعات جدول، حلقه کد نویسی بانک اطلاعاتی را کامل می کند. برای آشنایی بیشتر با LINQ، این مرحله را توسط پروسیجر ذخیره شده نمی نویسیم (هرچند نوشتن پروسیجری با نام pSearch که پارامتر ورودی آن، فیلد مورد جستجو یا بخشی از آن باشد یک راه حل خوب است). فرم را مطابق شکل 19 تغییر دهید:



شکل 19 - اضافه کردن بخش جستجو به فرم

برای رویداد TextChanged کنترل txFamilySearch کد زیر را بنویسید:

```
private void txFamilySearch_TextChanged(object sender, EventArgs e)
{
       var db = new LINQDataContext();
       if (txFamilySearch.Text == "")
           gvTable1.DataSource = db.Table1s;   //Load All Records to Grid
       else
           gvTable1.DataSource = db.Table1s.Where(c => c.Family.Substring(0,
                   txFamilySearch.Text.Length) == txFamilySearch.Text).Select(c => c);
}
```

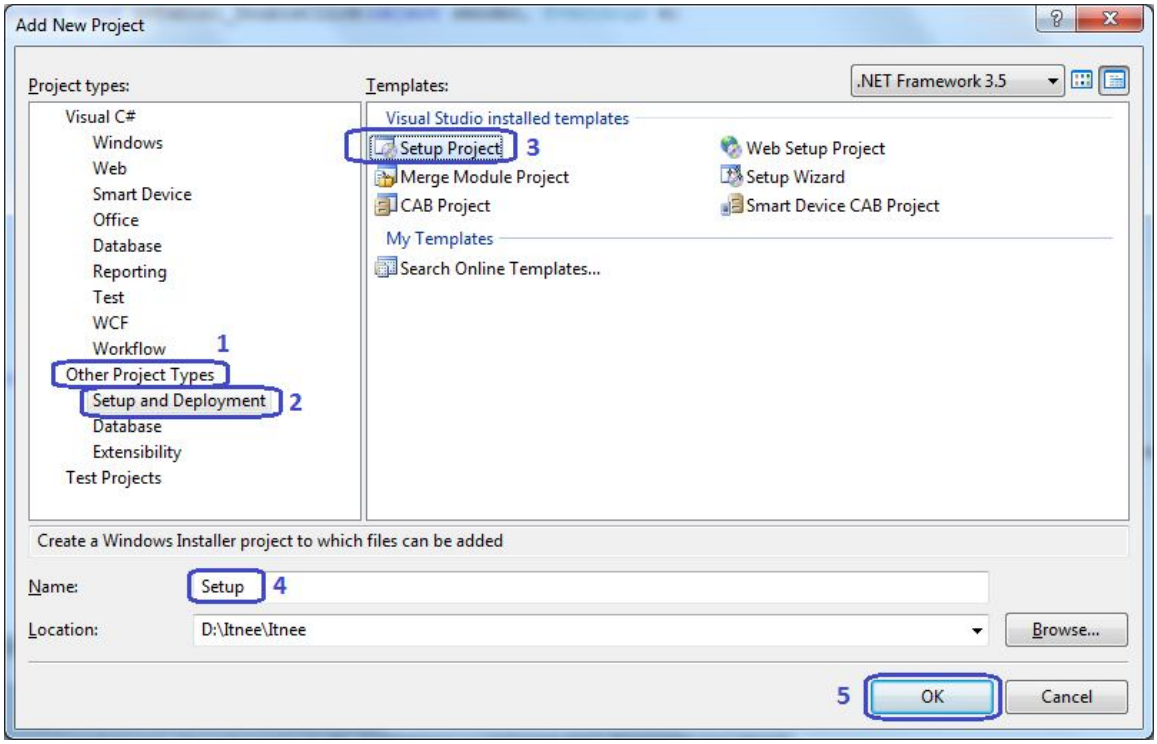
در اینجا ما شرطی روی جدول وضع کرده ایم که توسط عبارات لامبدای LINQ نوشته شده است. c متغیری است که به جدول اشاره می کند و فیلد Family آنرا فراخوانی کرده ایم. عبارت شرطی به این معنی است:

هر رکوردی که بخش سمت چپ (به طول txFamilySearch) فیلد Family آن مساوی عبارت مورد جستجو بود را انتخاب کن.

ایجاد Setup برای پروژه

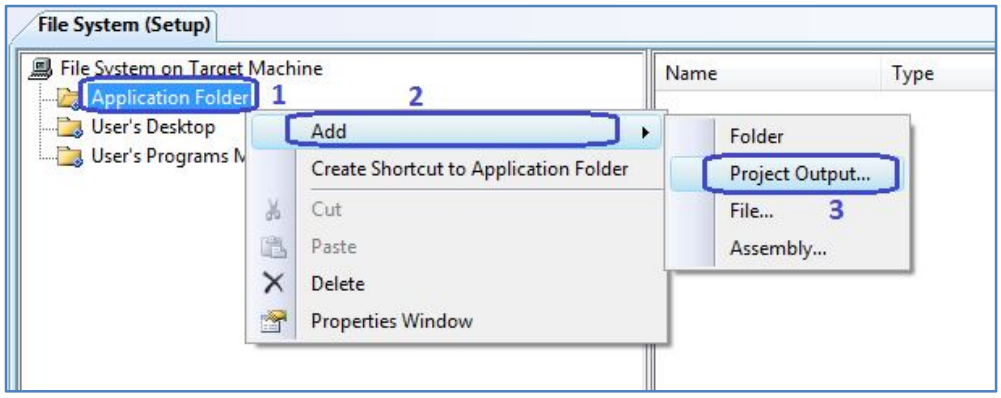
در ویژوال استودیو به مسیر زیر بروید

File > Add > New Project ...



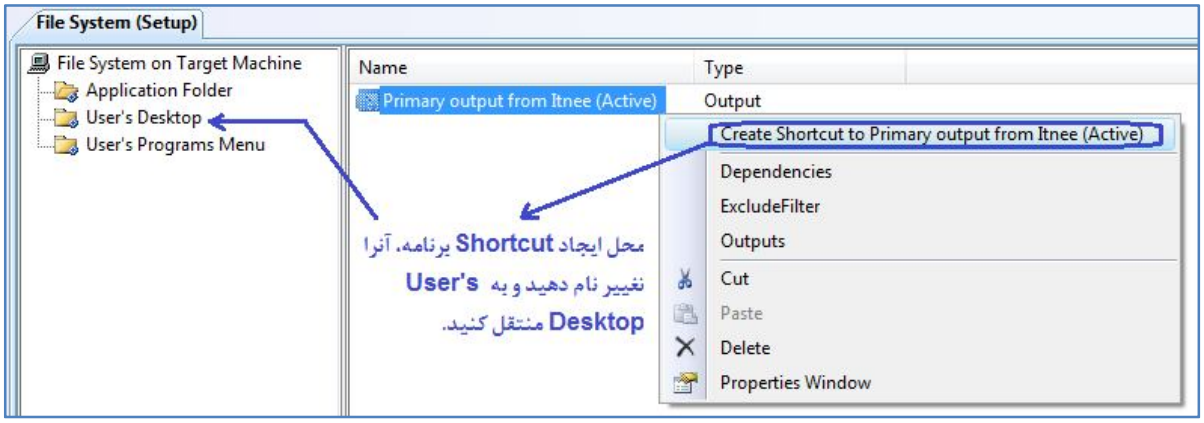
شکل 20 - اضافه کردن پروژه Setup به سلوشن Itnee

برروی Application Folder کلیک راست کنید و مطابق شکل21، خروجی برنامه را به پروژه Setup منتقل کنید. در پنجره ای که پس از آن باز می شود، دکمه OK را بزنید.



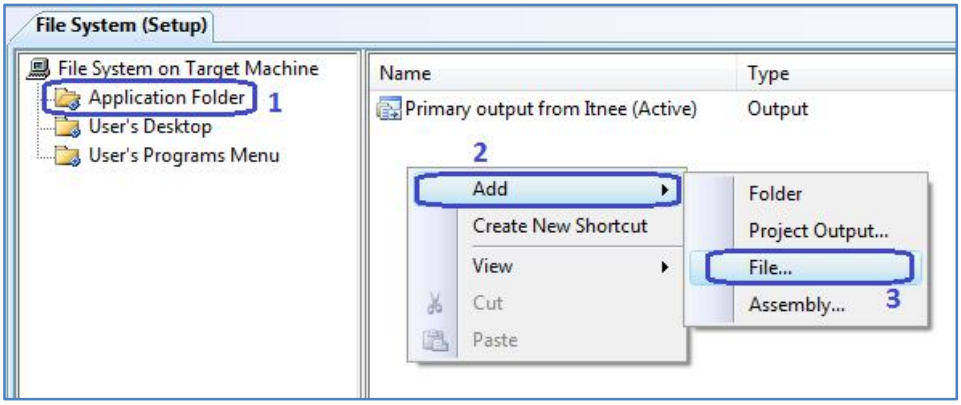
شکل 21 – انتقال فایهای پروژه به Setup

حال نوبت به ایجاد Shortcut فایل اجرایی در دسکتاپ و یا منوی استارت سیستم مقصد است. برای این کار مطابق شکل 22، روی خروجی برنامه کلیک راست کنید و یک شورتکات بسازید:



شکل 22 - ایجاد Shortcut روی دسکتاپ کاربر

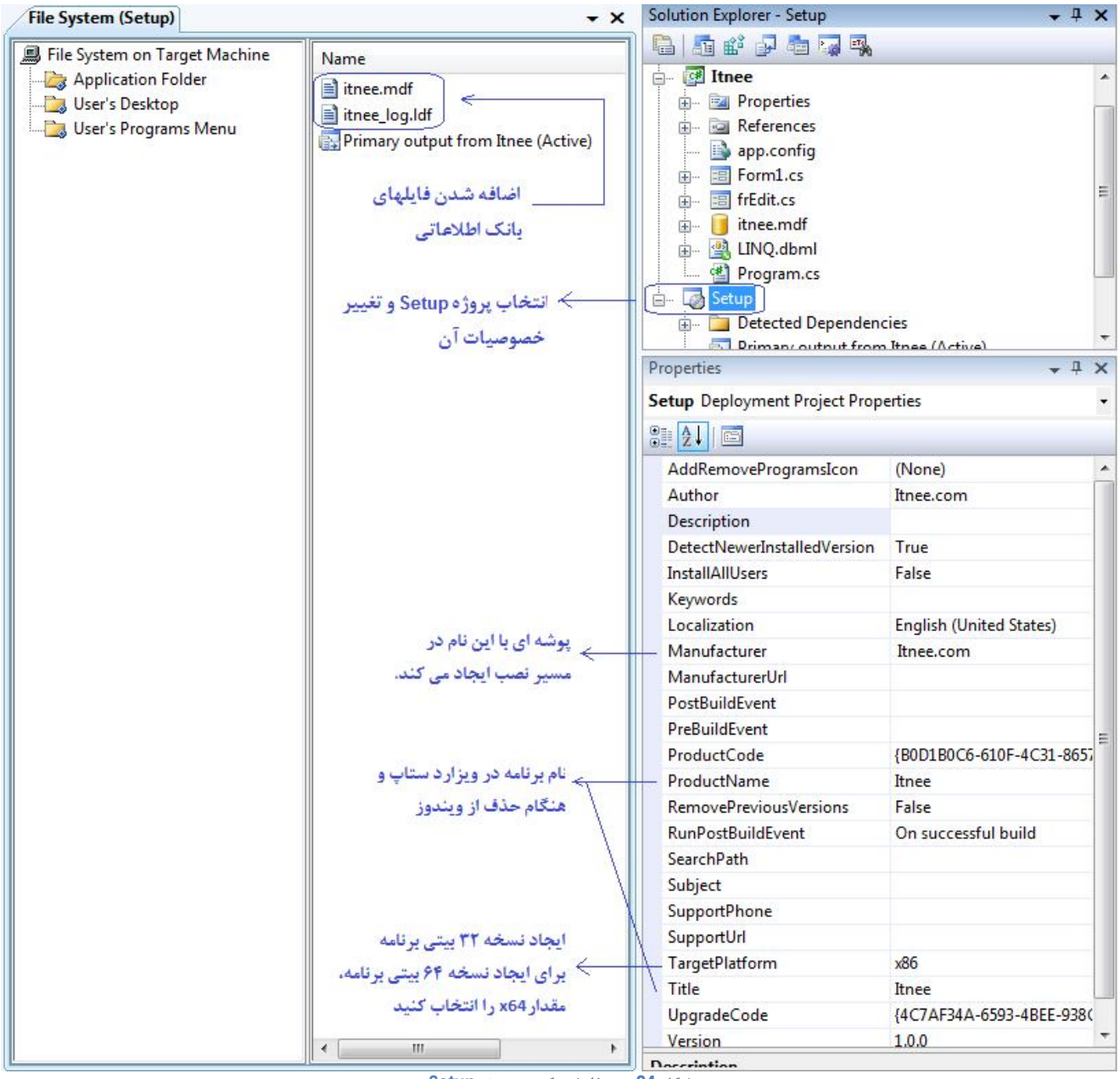
هنوز یک قسمت مهم باقی مانده است. پروژه ما دارای یک بانک اطلاعاتی است که باید به Setup اضافه شود. برای این کار، دوباره در قسمت Application Folder کلیک راست کنید (شکل 23)



شکل 23 - اضافه کردن فایل بانک اطلاعاتی به Setup

به مسیر پروژه بروید و 2 فایل itnee.mdf و itnee_log.ldf را انتخاب و اضافه کنید.

پس از اضافه کردن فایلها، خصوصیات پروژه Setup را سفارشی می کنیم. تا نام پروژه هنگام Setup و Uninstall، شرکت سازنده، نسخه نرم افزار و ... مقادیر درستی داشته باشند. برای این کار در پانل Solution Explorer پروژه Setup را انتخاب کنید و مطابق شکل 24 مقادیر دلخواه را وارد کنید.

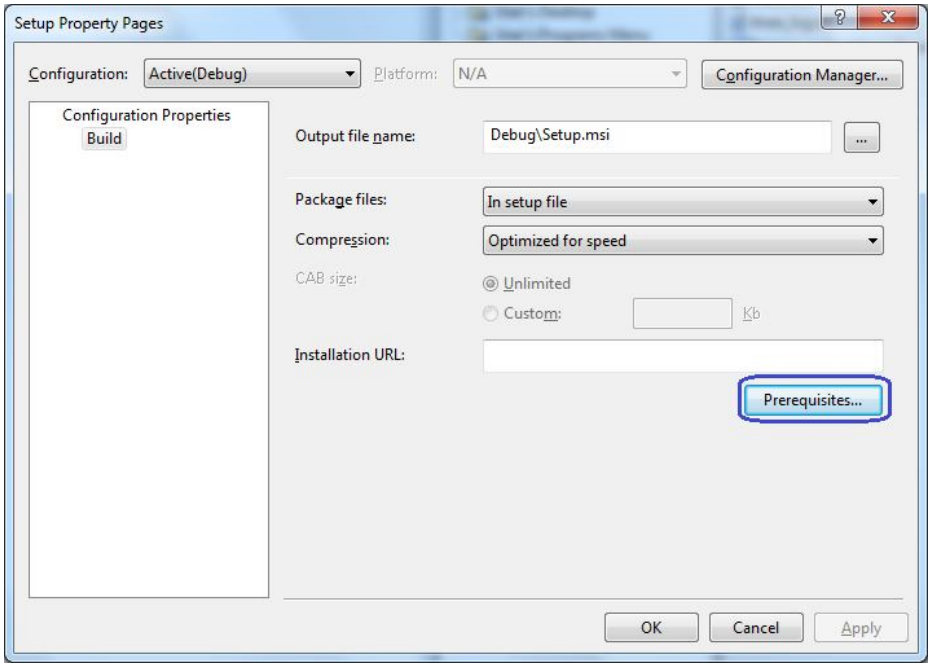


شکل 24 - سفارشی کردن پروژه Setup

آخرین مرحله از Setup سازی

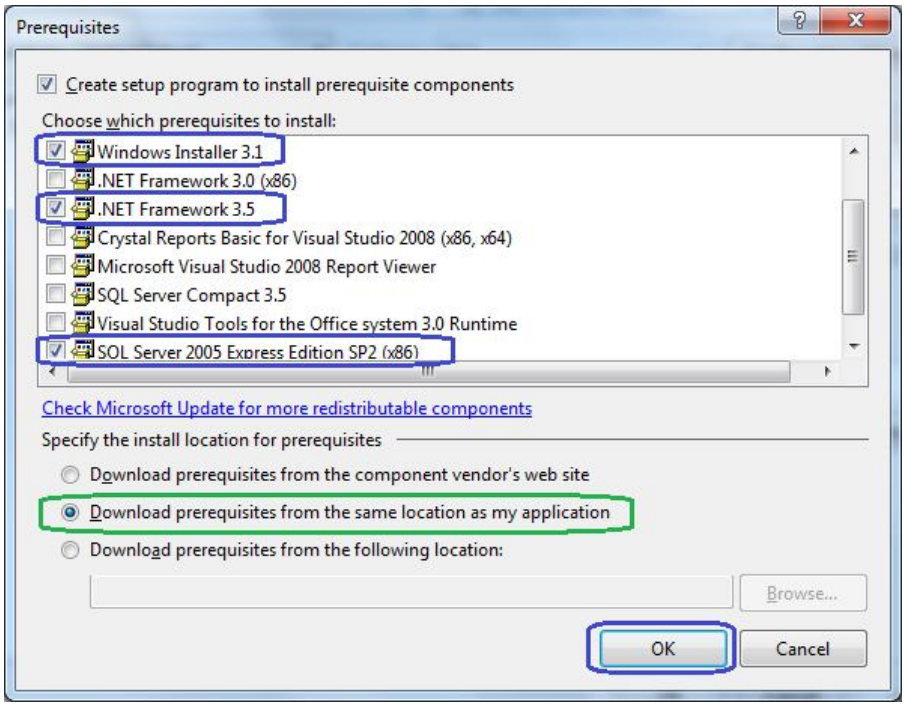
تا این مرحله، در واقع Setup پروژه ما ایجاد شده است و کافیست به مسیر پروژه > پوشه Setup > پوشه Debug برویم و دو فایل Setup ایجاد شده را به سیستمهایی که می خواهیم برنامه را در آنها نصب کنیم، ببریم و Setup.exe را اجرا کنیم. اما پروژه ما برای اجرا نیازمند Net Framework 3.5. و همچنین SQLExpress است. گرچه در ویندوزهای Vista و Windows7 محیط NET. با سیستم عامل عرضه می شود. اما در ویندوز XP لازم است ابتدا NET Framework 3.5. نصب شود. Setup ویژوال استودیو قادر است قبل از نصب برنامه، از نصب بودن پیش نیازهای آن روی سیستم عامل اطمینان حاصل کند و سپس نسبت به نصب برنامه، اقدام کند. در این مرحله، پیش نیازهای نصب برنامه را به پروژه Setup اضافه می کنیم تا در صورت نیاز، آنها نیز نصب شوند.

روی پروژه Setup در Solution Explorer کلیک راست کنید و Properties را انتخاب کنید تا پنجره Setup Property Pages باز شود. (شکل 25) سپس دکمه ...Prerequisites را بزنید



شکل 25 - اضافه کردن پیش نیازهای Setup برای نصب برنامه

مطابق شکل 26، Prerequisites را تنظیم کنید.



شکل 26 - انتخاب پیش نیازهای نصب برنامه و اضافه کردن آنها به پوشه Setup

انتخاب گزینه Download prerequisites from the same location as my application باعث کپی شدن پیش نیازها به پوشه Setup می شود که حجم پروژه شما را بالا می برد. اما Setup همیشه کار خواهد کرد!

حال پروژه Setup را Build کنید و به مسیر آن بروید. کل محتویات پوشه Debug را به سیستم مقصد ببرید و فایل Setup.exe را با خیال راحت اجرا کنید!

■■■