

Identity یکی از Attribute‌هایی که در SQL Server به ازای Column‌های عددی می‌توان در نظر گرفت. به طور خیلی ساده هنگامی که این Attribute به ازای یک فیلد عددی تنظیم گردد. چنانچه رکوردی در جدول مربوط به Identity درج شود فیلد Identity مقداری را به طور اتوماتیک دریافت خواهد نمود.

نحوه دریافت مقدار به ازای فیلد Identity با توجه به آخرین مقدار آن و گام افزایش است که در هنگام ایجاد identity تعریف می‌گردد.

برای ایجاد یک فیلد از نوع Identity می‌توانید زمانی که جدول خود را ایجاد می‌کنید این Attribute را به فیلد مورد نظر خود تخصیص دهید.

**مثال 1:** این مثال نحوه ایجاد یک فیلد از نوع Identity را نمایش می‌دهد.

```
USE tempdb
GO
CREATE TABLE Customers1
(
    ID INT IDENTITY, -- ID INT IDENTITY(1,1)
    Name NVARCHAR(100),
    [Address] NVARCHAR(200)
)
GO
```

همانطور که در مثال 1 مشاهده می‌کنید فیلد ID از نوع Identity تعریف شده است. در این حالت (ID int IDENTITY) مقدار شروع و گام افزایش به ازای این فیلد 1 در نظر گرفته خواهد شد. در این صورت اگر چند رکورد زیر را به ازای این جدول درج کنید. مقدار Identity به صورت زیر خواهد بود.

```
INSERT INTO Customers1 (Name,[Address]) VALUES
(N'میانه',N'مسعود'),
(N'میانه',N'فرید'),
(N'میانه',N'احمد')
GO
SELECT * FROM Customers1
```

Results		Messages	
	ID	Name	Address
1	1	مسعود	میانه
2	2	فرید	میانه
3	3	احمد	میانه

**مثال 2:** این مثال نحوه ایجاد یک فیلد از نوع Identity به همراه مقدار شروع و گام افزایش را مشخص می‌کند.

```
USE tempdb
GO
CREATE TABLE Customers2
(
    ID INT IDENTITY(100,2),
    Name NVARCHAR(100),

```

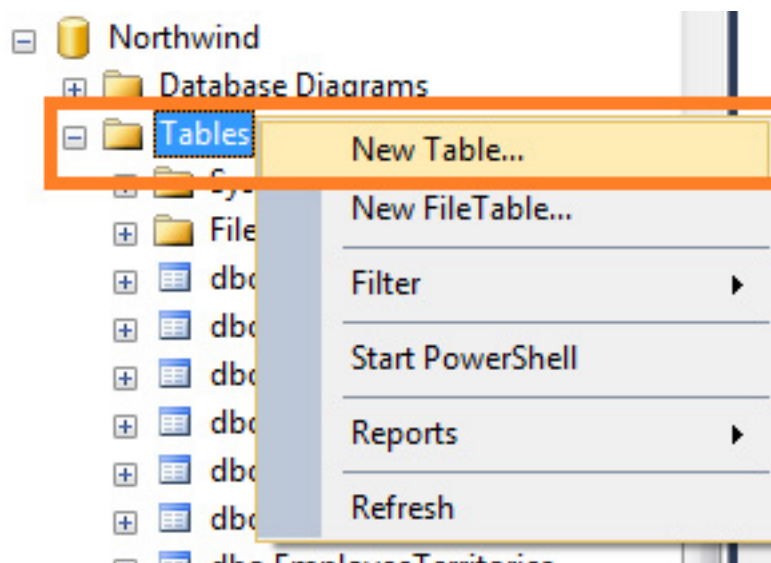
```
[Address] NVARCHAR(200)
)
GO
```

همانطور که در مثال 2 مشاهده می‌کنید فیلد ID از نوع Identity تعریف شده است و مقدار شروع آن از 100 و همچنین گام افزایش 2 در نظر گرفته شده است. در این صورت اگر چند رکورد زیر را به ازای این جدول درج کنید. مقدار Identity به صورت زیر خواهد بود.

```
INSERT INTO Customers2 (Name,[Address]) VALUES
(N'مسعود',N'میانه'),
(N'فرید',N'میانه'),
(N'احمد',N'میانه')
GO
SELECT * FROM Customers2
```

	ID	Name	Address
1	100	مسعود	میانه
2	102	فرید	میانه
3	104	احمد	میانه

**مثال 3:** این مثال نحوه تنظیم یک فیلد به صورت Identity را در محیط SSMS (SQL Server Management Studio) آموزش می‌دهد. 1- برای شروع کار همانند تصویر زیر بر روی قسمت Table کلیک راست کنید و گزینه New Table... را انتخاب کنید.



2- پس از نمایش پنجره زیر فیلدی را که می‌خواهید از نوع Identity باشد را انتخاب کرده و در قسمت Column Properties

خصیصه Is Identity را برابر Yes قرار دهید تا فیلد مورد نظر شما از نوع Identity در نظر گرفته شود. لازم به ذکر است که Identity Seed مقدار شروع و Identity Increment گام افزایش را مشخص می‌نماید.

TAHERI-PC.Northwind - dbo.Table\_1\* × Identity.sql - (loca... \Administrator (51))

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Name	nvarchar(100)	<input checked="" type="checkbox"/>
Address	nvarchar(100)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Column Properties

Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
<b>Identity Specification</b>	<b>Yes</b>
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No

## نظرات خوانندگان

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۰۳/۲۱ ۱۷:۱

آیا فیلد Identity به خودی خود زمانیکه primary key و unique نباشه (مانند مثال‌های بالا) ارزش و کاربردی داره؟

نویسنده: فرید طاهری  
تاریخ: ۱۳۹۲/۰۳/۲۲ ۹:۲۳

این موضوع بستگی به سناریو شما داره  
اما معمولا در بیشتر مواقع Identity را به شکل Unique در نظر می‌گیرند ذکر این نکته هم ضروری است که  
1- در SQL Server معمولا Primary Key بوسیله یک Unique Clustered Index هندل می‌شود  
(هر چند می‌شود اون رو به صورت یک Unique Non Clustered Index در نظر گرفت)  
2- Clustered Index ترتیب و چینش فیزیکی رکوردها را مشخص می‌کند یعنی اگر Identity به عنوان کلاستر ایندکس باشد  
چینش و ترتیب فیزیکی رکوردها بر اساس Identity خواهد بود (سطح leaf Level مربوط به ایندکس که در کلاستر ایندکس  
همان Data Level است)

همانگونه که می‌دانید مقدار Identity پس از درج به آن تخصیص می‌یابد چنانچه بخواهید به این مقدار دسترسی پیدا کنید چندین روش به ازای اینکار وجود دارد که ما در این مقاله سه روش معمول را بررسی خواهیم نمود.

1- استفاده از متغیر سیستمی @@Identity

2- استفاده از تابع Scope\_Identity ()

3- استفاده از تابع Ident\_Current

هر سه این توابع مقدار Identity ایجاد شده برای جداول را نمایش می‌دهند. اما تفاوت هایی باهم دارند که در ادامه مقاله این تفاوت‌ها بررسی شده است.

**1- متغیر سیستمی @@Identity :** این متغیر سیستمی حاوی آخرین Identity ایجاد شده به ازای Session جاری شما است. لازم به ذکر است اگر به واسطه Insert شما، Identity دیگری در یک حوزه دیگر (مانند یک Trigger) ایجاد شود مقدار موجود در این متغیر حاوی آخرین Identity ایجاد شده است. (یعنی Identity ایجاد شده توسط آن تریگر و نه خود جدول). لازم به ذکر است این موضوع به طور کامل در ادامه مقاله شرح داده شده است.

**2- استفاده از تابع Scope\_Identity () :** با استفاده از این تابع می‌توانیم آخرین Identify ایجاد شده به ازای Session جاری را بدست آوریم. لازم به ذکر است مقادیر Identity ایجاد شده توسط سایر حوزه‌ها تاثیر در مقدار بازگشتی توسط این تابع ندارد. در ادامه مقاله این موضوع به طور کامل بررسی شده است.

**3- استفاده از تابع ident\_Current :** این تابع آخرین مقدار Identity موجود در یک جدول را نمایش می‌دهد. ذکر این نکته ضروری است که Identity ایجاد شده توسط سایر Session‌ها هم روی خروجی این تابع تاثیرگذار است. چون این تابع آخرین Identity موجود در جدول را به شما نمایش می‌دهد و نه Identity ایجاد شده به ازای یک Session را.

برای بدست آوردن یک Identity کافی است که پس از درج رکورد در جدول مورد نظر متغیر سیستمی @@Identity و یا توابع Scope\_Identity و یا Ident\_Current را همانند مثال زیر Select کنید.

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees', N'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees
(
    ID int IDENTITY,
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
INSERT INTO Employees (FirstName,LastName) VALUES (N'طاهری',N'مسعود')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

خروجی دستورات بالا پس از درج رکورد مورد نظر به صورت زیر است.

Results		Messages	
@@IDENTITY			
1	1		
SCOPE_IDENTITY()			
1	1		
IDENT_CURRENT('Employees')			
1	1		

اما ممکن است از خودتان این سوال را بپرسید که آیا این توابع در سطح شبکه آخرین مقدار Identity درج شده توسط سایر Sessionها را نمایش می‌دهند و یا Session جاری را؟ (منظور Sessionی که درخواست مقدار موجود در identity را نموده است).

برای دریافت پاسخ این سوال مطابق مراحل اسکریپت‌های زیر را اجرا نمایید.

#### 1- ایجاد جدول Employees1

```
USE TEMPDB GO
IF OBJECT_ID('Employees1', 'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees1
(
    ID int IDENTITY(1,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```

همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 1 و گام افزایش هم برابر 1 در نظر گرفته شده است ((Identity(1,1)).

2- در Session جدید دستورات زیر را اجرا نمایید. (درج رکورد جدید در جدول Employees1 و واکنش مقدار Identity)

USE tempdb

```
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'فرید')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

SPID مربوط به  
Connection جاری

SQLQuery2.sql - (lo...Administrator (55))\* X SQLQuery1.sql - (lo...Administrator (53))\*

```
USE tempdb
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'فرید')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

100 %

Results Messages

@@IDENTITY	
1	1

SCOPE_IDENTITY()	
1	1

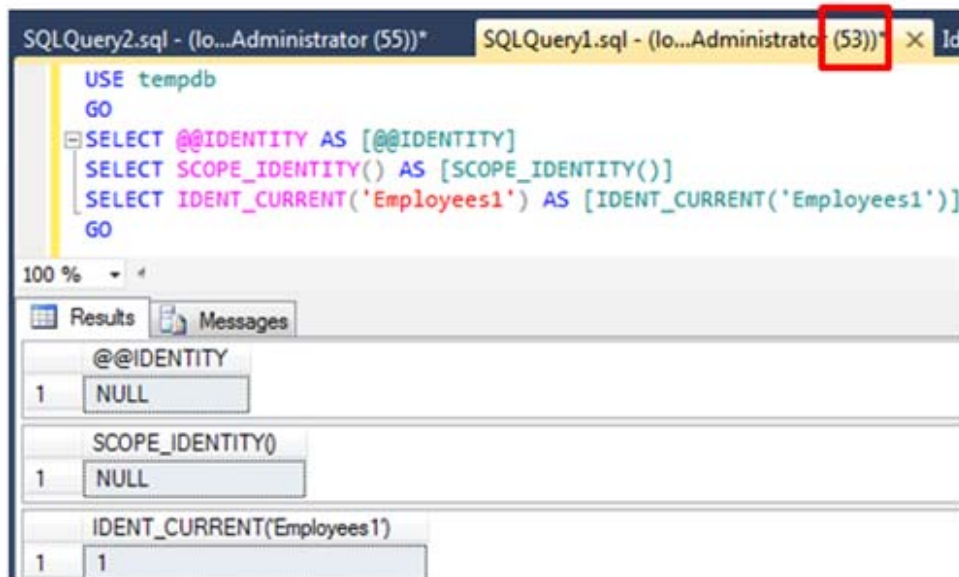
IDENT_CURRENT('Employees1')	
1	1

همانگونه که ملاحظه می‌کنید @@Identity , Scope\_Identity() و Ident\_Current هر سه مقدار Identity (عدد 1) ایجاد شده بوسیله دستور Insert را به شما نمایش می‌دهند.

1- و در انتها در یک Session دیگر دستورات زیر را اجرا نمایید. (واکشی مقدار Identity)

```
USE tempdb
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
GO
```

SPID مربوط به  
Connection جاری



همانطور که مشاهده می‌کنید در این Session ما از SQL خواسته‌ایم آخرین مقدار Identity را به ما نشان داده شود. باید به این نکته توجه کنید با توجه به اینکه در این Session عملیات درجی هنوز انجام نگرفته است که ما Identity ایجاد شده را مشاهده نماییم. بنابراین صرفاً تابع Idet\_Current مقدار Identity موجود در جدول را به ما نمایش می‌دهد.

پس می‌توان به این نکته رسید که

**@@Identity و Scope\_Identity** ایجاد به ازای Session جاری را نمایش داده و به مقادیر تولید شده توسط سایر Session های دیگر دسترسی ندارد.

**Ident\_Current** : آخرین Identity موجود در جدول را به شما نمایش می‌دهد. بنابراین باید این نکته را در نظر داشته باشید که Identity ها ایجاد شده توسط سایر Session ها روی مقدار بازگشتی این تابع تأثیرگذار است.

اما یکی دیگر از مباحث مهم درباره Identity تأثیر Scope بر مقدار Identity است (یعنی چه!). برای اینکه با مفهوم این موضوع آشنا شوید اسکریپت‌های مربوط به مثال زیر را بدقت اجرا کنید.

#### 1- ایجاد جدول Employees1

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees1', N'U') IS NOT NULL
    DROP TABLE Employees1;
GO
CREATE TABLE Employees1
(
    ID int IDENTITY(1,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```



همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 1 و گام افزایش هم برابر 1 در نظر گرفته شده است)  
(Identity(1,1) .

## 2- ایجاد جدول Employees2

```
USE TEMPDB
GO
IF OBJECT_ID(N'Employees2', N'U') IS NOT NULL
    DROP TABLE Employees2;
GO
CREATE TABLE Employees2
(
    ID int IDENTITY(100,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50)
)
GO
```

همانطور که مشاهده می‌کنید مقدار شروع برای Identity برابر 100 و گام افزایش هم برابر 1 در نظر گرفته شده است)  
(Identity(100,1) .

## 3- ایجاد یک Trigger به ازای جدول Employees1

```
USE tempdb
GO
CREATE TRIGGER Employees1_Insert ON Employees1 FOR INSERT
AS
BEGIN
    INSERT Employees2(FirstName,LastName)
    SELECT FirstName,LastName FROM INSERTED
END;
GO
```

Trigger ایجاد شده به ازای جدول Employees1 به ازای عملیات Insert اجرا می‌شود. همچنین مقادیر درج شده در جدول Employees1 بوسیله جدول Inserted در دسترس است. لازم به ذکر است جدول Inserted یک جدول موقت بوده که توسط Trigger ایجاد شده و داخل خود آن معتبر است.

هدف ما از ایجاد این Trigger تهیه یک کپی از رکوردهایی که در جدول Employees1 درج می‌شوند است. این کپی قرار است با استفاده از دستور Insert...Select در جدول Employees2 ایجاد گردد.

## 4- درج یک رکورد در جدول Employees1 و واکنشی مقدار Identity

```
USE tempdb
GO
INSERT INTO Employees1(FirstName,LastName) VALUES (N'طاهری',N'مسعود')
GO
SELECT @@IDENTITY AS [@@IDENTITY]
SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY()]
SELECT IDENT_CURRENT('Employees1') AS [IDENT_CURRENT('Employees1')]
SELECT IDENT_CURRENT('Employees2') AS [IDENT_CURRENT('Employees2')]
GO
```

Results		Messages	
		@@IDENTITY	
1	100		
		SCOPE_IDENTITY()	
1	1		
		IDENT_CURRENT('Employees1')	
1	1		
		IDENT_CURRENT('Employees2')	
1	100		

#### مقادیر استخراج شده به ازای Identity به شرح زیر است

**1- Identity@@ :** پس از درج رکورد در جدول Employees1 متغیر سیستمی @@Identity مقدار 100 را نمایش داده است دلیل این موضوع بر می‌گردد به Trigger موجود در جدول Employees1.

با توجه به اینکه جدول Employees1 دارای یک فیلد Identity بوده است هنگام درج رکورد در جدول مقدار @@Identity=1 است اما چون این جدول دارای Trigger ی است که این Trigger خود با جدولی دیگری درگیر است که دارای Identity است مقدار متغیر @@identity خواهد شد.

**2- Scope\_Identity() :** مقدار نمایش داده شده توسط تابع Scope\_Identity() برابر با مقدار Identity تخصیص (عدد 1) داده شده به ازای رکورد شما می‌باشد که این موضوع در اغلب موارد مد نظر برنامه‌نویسان می‌باشد.

**3- Ident\_Current('Employees1') :** مقدار نمایش شده توسط تابع Ident\_Current آخرین مقدار Identity (عدد 1) موجود در جدول Employees1 است.

**4- Ident\_Current('Employees2') :** مقدار نمایش شده توسط تابع Ident\_Current آخرین مقدار Identity (عدد 100) موجود در جدول Employees2 است.

#### چند نکته مهم

1- مقدار بازگردانده شده توسط تابع Ident\_Current آخرین مقدار Identity موجود در جدول مورد نظر شما بوده است و عملیات درج سایر کاربران در این مقدار تاثیر گذار است.

2- برای بدست آوردن مقدار Identity درست‌تر است از تابع Scope\_Identity() استفاده نماییم. معمولاً در بیشتر مواقع مقدار بازگردانده شده توسط این تابع مد نظر برنامه‌نویسان است.

3- EntityFramework و Nhibernate هم برای بدست آوردن Identity از تابع Scope\_Identity استفاده می‌کند.

## نظرات خوانندگان

نویسنده: کاربر

تاریخ: ۴:۵۰ ۱۳۹۲/۰۴/۳۱

بین دوست من مطلبتون رو خوندم هم اینو و هم قبلی رو، ازش خوشم اومد اما چیزی راجب درج صریح یا بروز رسانی مقادیر Identity ننوشته بودین. یا اینکه همیشه در یک جدول دو identity property داشت.

من بلام با set identity\_insert table\_name on/off کاری کنم که خودم دستی مقداری را برای خصیصه identity لحاظ کنم. ولی متاسفانه نتونستم مقدار یک ستون با خصیصه Identity رو بروز رسانی (یا همون update) کنم. لطفا بهم بگید که اصلا این کار ممکنه یا من بلد نیستم. البته براساس query زیر بمن SQL Server گفته که همیشه این ستون را update کرد که ظاهرا هم همین طور(ستون id همانطور که در پیام آمده از نوع identity هست)

```
update t
set id = new_id
from (select id, row_number() over(order by id) new_id from #temp)t
--Cannot update identity column 'id'.
```

اصلا اجازه بدین یه جور دیگه سوال رو مطرح کنم من نیاز دارم تمام مقادیر identity رو بروز رسانی کنم تا کاملا پشت سر هم و متوالی بشن این کار را میتونم با یک تابع row\_number و یک derived table انجام بدم (اگر بذارن!) همانطور که قبلا نشان دادم، یا با روش زیر این کار را بکنم که البته اجرا نمیشه به این دلیل که در یک جدول همیشه دو identity property داشت. با فرض اجرا شدن دستور select into باز هم در دستور update با مشکل بر می خوردیم (چون همیشه ستون id را بروز رسانی کرد)

```
select id, identity(int, 1,1) new_id
into #temptable
from #temp
order by id asc

/*
cannot add identity column, using the SELECT INTO statement, to table '#temptable',
which already has column 'id' that inherits the identity property.
*/
update t
set id = new_id
from #temp t
join #temptable d
on t.id = d.id;
```

البته یک راهی برای حل این مساله هست اونم اینه که ابتدا بیاییم تمام داده ها جدول را در جدول دیگه ای درج کنیم سپس تمام داده های جدول را حذف کنیم سپس داده های حذف شده را با id جدید و مرتب شده در جدول اول درج کنیم. به این شکل

```
declare @t table(id int)

insert into @t
select id from #temp

delete from #temp

set identity_insert #temp on
insert #temp (id)
select row_number() over(order by id) from @t
set identity_insert #temp off
```

اما مشکلی که وجود داره اینه که اگر جدول ما parent باشه با مشکل واجه میشیم تمام سطرهای جداول child یتیم میشن.

من قصد ندارم صورت مساله نقد و بررسی بشه و اصولی بودن یا صحیح بودنش مورد ارزیابی قرار بگیره فقط برام این یک سوال شده.

مساله عمومی که راجب این ستون وجود داره استفاده کردن از Gap های حاصل شده در این ستون برای درج های بعدی است. که query آن نیز بسیار ساده و در دسترس است. آیا شما میدانید که چگونه این مشکل با sequence ای که در نسخه 2012 معرفی شده است حل می شود؟

نویسنده: وحید نصیری  
تاریخ: ۱۴:۵۵ ۱۳۹۲/۰۴/۳۱

- خیر. چندین نوع استراتژی برای تعیین PK وجود دارند که یکی از آن ها فیلدهای Identity است و این تنها روش و الزاما بهترین روش نیست.  
- مثلا زمانیکه با ORM ها کار می کنید استفاده از فیلدهای Identity در حین ثبت تعداد بالایی از رکوردها مشکل ساز می شوند. چون این فیلدها تحت کنترل دیتابیس هستند و نه برنامه، ORM نیاز دارد پس از هربار Insert یکبار آخرین Id را از بانک اطلاعاتی واکنشی کند. همین مساله یعنی افت سرعت در تعداد بالای Insert ها (چون یکبار کوئری Insert باید ارسال شود و یکبار هم یک Select اضافی دوم برای دریافت Id تولیدی توسط دیتابیس).  
- روش دوم تعیین PK استفاده از نوع Guid است. در این حالت، هم مشکل حذف رکوردها و خالی شدن یک شماره را در این بین ندارید و هم چون عموما تحت کنترل برنامه است، سرعت کار کردن با آن بالاتر است. فقط تنها مشکل آن زیبا نبودنش است در مقایسه با یک عدد ساده فیلدهای Identity.

در مورد فیلدهای Identity، [تغییر شماره Id](#) به صلاح نیست چون:  
الف) همانطور که عنوان کردید روابط بین جداول را به هم خواهد ریخت.  
ب) در یک وب سایت و یا هر برنامه ای، کلا آدرس ها و ارجاعات قدیمی را از بین می برد. مثلا فرض کنید شماره این مطلب 1381 است و شما آن را یادداشت کرده اید. در روزی بعد، برنامه نویس شماره Id ها را کلا ریست کرده. در نتیجه یک هفته بعد شما به شماره 1381 ایی خواهید رسید که تطابقی با مطلب مدنظر شما ندارد (حالا فرض کنید که این عدد شماره پرونده یک شخص بوده یا شماره کاربری او و نتایج و خسارات حاصل را در نظر بگیرید).  
ج) این خوب است که در بین اطلاعات یک ردیف خالی وجود دارد. چون بر این اساس می توان بررسی کرد که آیا واقعا رکوردی حذف شده یا خیر. گاهی از اوقات کاربران ادعا می کنند که اطلاعات ارسالی آن ها نیست در حالیکه نبود این رکوردها به دلیل حذف بوده و نه عدم ثبت آن ها. با بررسی این Id ها می شود با کاربران در این مورد بحث کرد و پاسخ مناسبی را ارائه داد.  
و اگر شماره ای که به کاربر نمایش می دهید فقط یک شماره ردیف است (و از این لحاظ می خواهید که حتما پشت سرهم باشد)، بهتر است یک View جدید ایجاد کنید تا این Id خود افزاینده را تولید کند (بدون استفاده از pk جدول).

پ.ن.

هدف من از این توضیحات صرفا عنوان این بود که به PK به شکل یک فیلد *read only* نگاه کنید. این دقیقا برخوردی است که Entity framework با این مفهوم دارد و صحیح است و اصولی. اگر در یک کشور هر روزه عده ای به رحمت ایزدی می روند به این معنا نیست که سازمان ثبت احوال باید شماره شناسنامه ها را هر ماه ریست کند!

نویسنده: فرید طاهری  
تاریخ: ۲۰:۰۰ ۱۳۹۲/۰۴/۳۱

با تشکر از آقای نصیری و پاسخ مناسبی که ارائه کرده اند  
در مورد استفاده از GUID به جای identity باید به یک نکته هم اشاره کنم که در بیشتر مواقع اگر مقدار GUID ی که به ازای یک فیلد UNIQUEIDENTIFIER تنظیم می کنید به صورت SEQUENTIAL نباشد باعث Fragment شدن ایندکس خواهد شد.  
برای مقایسه بهتر بین Fragmentation ایندکس مربوط به Identity و GUID به مثال زیر دقت کنید. هر دو مثال فیلد ID خود را به شکل Clustered Index دارند بعد از درج تعدادی رکورد مساوی در دو جدول Fragmentation مربوط به جدولی که دارای GUID است به شدت بالا است که این موضوع باعث کاهش کارایی خواهد شد

```

IF OBJECT_ID('TABLE_GUID')>0
DROP TABLE TABLE_GUID
GO
CREATE TABLE TABLE_GUID
(
ID UNIQUEIDENTIFIER PRIMARY KEY,
FirstName NVARCHAR(1000),
LastName NVARCHAR(1000)
)
GO
IF OBJECT_ID('TABLE_IDENTITY')>0
DROP TABLE TABLE_IDENTITY
GO
CREATE TABLE TABLE_IDENTITY
(
ID INT IDENTITY PRIMARY KEY,
FirstName NVARCHAR(1000),
LastName NVARCHAR(1000)
)
GO
INSERT INTO TABLE_GUID(ID,FirstName,LastName) VALUES
(NEWID(),REPLICATE('FARID*',100),REPLICATE('Taheri*',100))
GO 10000

INSERT INTO TABLE_IDENTITY(FirstName,LastName) VALUES
(REPLICATE('FARID*',100),REPLICATE('Taheri*',100))
GO 10000

--Fragmentation بررسی وضعیت
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(),OBJECT_ID('TABLE_GUID'),NULL,NULL,'DETAILED')
DBCC SHOWCONTIG(TABLE_GUID)
GO
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(),OBJECT_ID('TABLE_IDENTITY'),NULL,NULL,'DETAILED')
DBCC SHOWCONTIG(TABLE_IDENTITY)
GO

```

خوب برای اینکه Fragmentation این نوع جداول را رفع کنید چند راه داریم

- 1- تولید GUID به صورت Sequential (لازم می‌دانم اشاره کنم این قابلیت در SQL Server وجود دارد ولی مقدار تولید شده باید به شکل یک Default Constraint باشد که این موضوع نیازمند این است که شما اگر در سورس به این GUID نیاز پیدا کنید مجبور به زدن Select و... شوید. اگر بخواهید در سورس این کار را انجام دهید باید از Extention‌هایی که برای اینکار وجود دارند استفاده کنید فکر کنم Nhibernate این حالت رو پشتیبانی کنه در مورد EF دقیقا اطلاع ندارم باید اهل فن نظر بدن)
- 2- تنظیم مقدار Fillfactor به ازای ایندکس
- 3-Rebuild و یا Reorganize دوره ای ایندکس

نویسنده: برنامه نویس

تاریخ: ۱۳۹۳/۰۷/۰۴ ۱۹:۱

باسلام؛ اگر همزمان 10 کاربر در شبکه بخوان درج کنن و هر بار identity درست بعد از درج نشان داده شود تابع Scope\_Identity درست عمل میکند؟ تداخل به وجود نمی‌آید؟

نویسنده: محسن خان

تاریخ: ۱۳۹۳/۰۷/۰۵ ۱۳:۳۲

در مطلب [بررسی Locks و Transactions در SQL Server](#) پاسخ خود را خواهید یافت.

نویسنده: مزروعی

تاریخ: ۱۳۹۴/۰۵/۲۹ ۹:۰۶

سلام؛ اگر در یک جدول اطلاعاتی 10 رکورد درج شود و کل آن اطلاعات را پاک کنیم و دوباره بخواهیم اطلاعات درج کنیم شماره آدی از 11 شروع می‌شود نه از 1. حال پس از حذف 10 رکورد اگر دستور زیر را اجرا کنیم

```
Ident_Current('tabla_name')
```

مقدار یک برمی گرداند در حالی که باید مقدار 10 را بیاورد برای حل این مشکل راه حلی دارید؟

در این مقاله مهاجرت داده‌های سیستم عضویت، نقش‌ها و پروفایل‌های کاربران که توسط Universal Providers ساخته شده اند به مدل ASP.NET Identity را بررسی می‌کنیم. رویکردی که در این مقاله استفاده شده و قدم‌های لازمی که توضیح داده شده اند، برای اپلیکیشنی که با SQL Membership کار می‌کند هم می‌تواند کارساز باشند. با انتشار Visual Studio 2013، تیم ASP.NET سیستم جدیدی با نام ASP.NET Identity معرفی کردند. می‌توانید [در این لینک](#) بیشتر درباره این انتشار بخوانید.

در ادامه مقاله قبلی تحت عنوان [مهاجرت از SQL Membership به ASP.NET Identity](#)، در این پست به مهاجرت داده‌های یک اپلیکیشن که از مدل Providers برای مدیریت اطلاعات کاربران، نقش‌ها و پروفایل‌ها استفاده می‌کند به مدل جدید ASP.NET Identity می‌پردازیم. تمرکز این مقاله اساساً روی مهاجرت داده‌های پروفایل کاربران خواهد بود، تا بتوان به سرعت از آنها در اپلیکیشن استفاده کرد. مهاجرت داده‌های عضویت و نقش‌ها، شبیه پروسه مهاجرت SQL Membership است. رویکردی که در ادامه برای مهاجرت داده پروفایل‌ها دنبال شده است، می‌تواند برای اپلیکیشنی با SQL Membership نیز استفاده شود. بعنوان یک مثال، با اپلیکیشن وبی شروع می‌کنیم که توسط Visual Studio 2012 ساخته شده و از مدل Providers استفاده می‌کند. پس از آن یک سری کد برای مدیریت پروفایل‌ها، ثبت نام کاربران، افزودن اطلاعات پروفایل به کاربران و مهاجرت الگوی دیتابیس می‌نویسیم و نهایتاً اپلیکیشن را بروز رسانی می‌کنیم تا برای استفاده از سیستم Identity برای مدیریت کاربران و نقش‌ها آماده باشد. و بعنوان یک تست، کاربرانی که قبلاً توسط Universal Providers ساخته شده اند باید بتوانند به سایت وارد شوند، و کاربران جدید هم باید قادر به ثبت نام در سایت باشند. سوره کد کامل این مثال را می‌توانید از [این لینک](#) دریافت کنید.

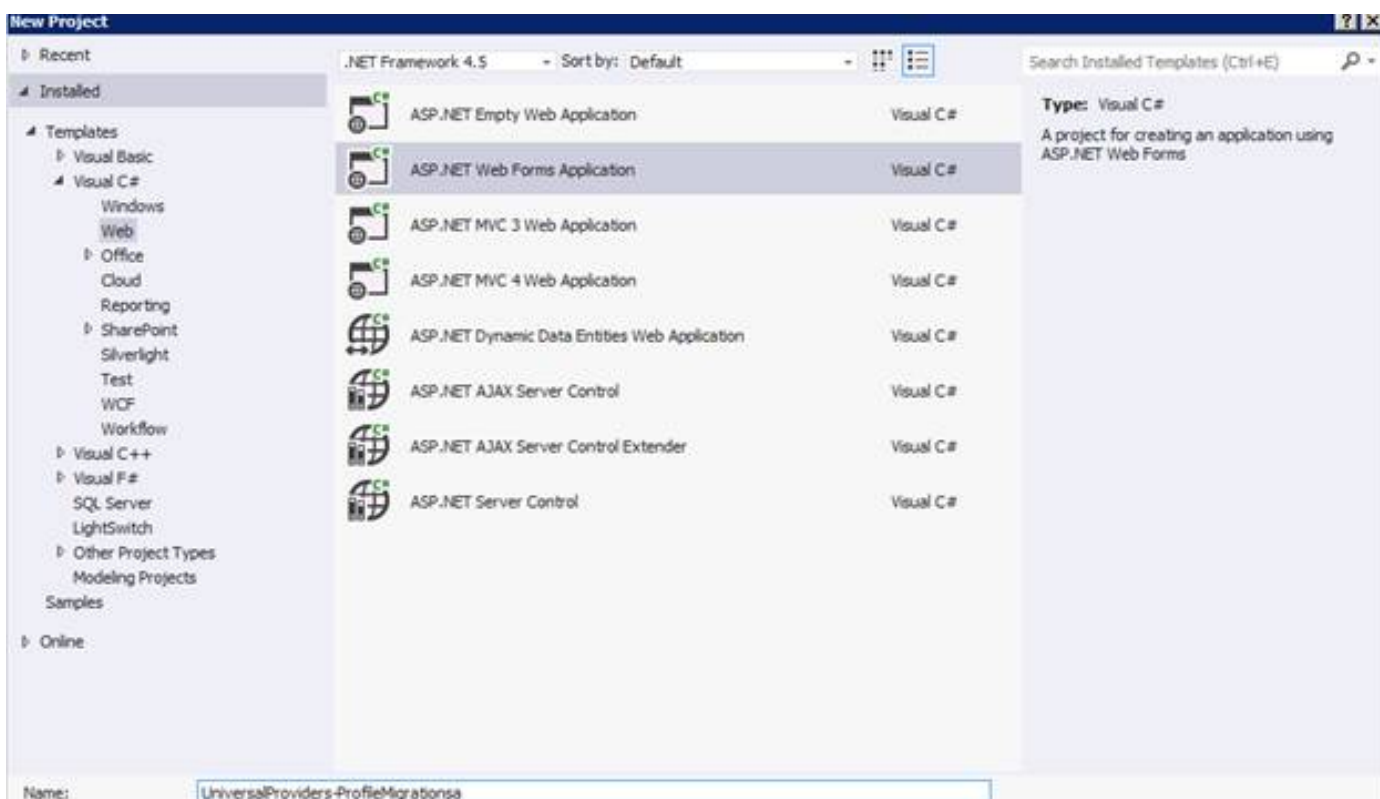
### خلاصه مهاجرت داده پروفایل‌ها

قبل از آنکه با مهاجرت‌ها شروع کنیم، بگذارید تا نگاهی به تجربه مان از ذخیره اطلاعات پروفایل‌ها در مدل Providers ببینیم. اطلاعات پروفایل کاربران یک اپلیکیشن به طرق مختلفی می‌تواند ذخیره شود. یکی از رایج‌ترین این راه‌ها، استفاده از تامین کننده‌های پیش فرضی است که به همراه Universal Providers منتشر شدند. بدین منظور انجام مراحل زیر لازم است کلاس جدیدی بسازید که دارای خواصی برای ذخیره اطلاعات پروفایل است. کلاس جدیدی بسازید که از 'ProfileBase' ارث بری می‌کند و متدهای لازم برای دریافت پروفایل کاربران را پیاده سازی می‌کند. استفاده از تامین کننده‌های پیش فرض را، در فایل web.config فعال کنید. و کلاسی که در مرحله 2 ساختید را بعنوان کلاس پیش فرض برای خواندن اطلاعات پروفایل معرفی کنید.

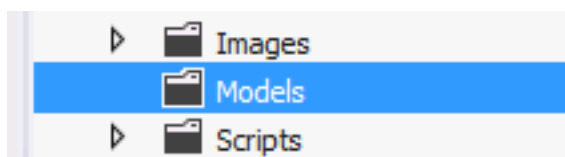
اطلاعات پروفایل‌ها بصورت binary و serialized xml در جدول 'Profiles' ذخیره می‌شوند.

پس از آنکه به سیستم ASP.NET Identity مهاجرت کردیم، اطلاعات پروفایل deserialized شده و در قالب خواص کلاس User ذخیره می‌شوند. هر خاصیت، بعداً می‌تواند به یک ستون در دیتابیس متصل شود. مزیت بدست آمده این است که مستقیماً از کلاس User به اطلاعات پروفایل دسترسی داریم. ناگفته نماند که دیگر داده‌ها serialize/deserialize هم نمی‌شوند.

**شروع به کار** در Visual Studio 2012 پروژه جدیدی از نوع ASP.NET 4.5 Web Forms application بسازید. مثال جاری از یک قالب Web Forms استفاده می‌کند، اما می‌توانید از یک قالب MVC هم استفاده کنید.



پوشه جدیدی با نام 'Models' بسازید تا اطلاعات پروفایل را در آن قرار دهیم.



بعنوان یک مثال، بگذارید تا تاریخ تولد کاربر، شهر سکونت، قد و وزن او را در پروفایلش ذخیره کنیم. قد و وزن بصورت یک کلاس سفارشی (custom class) بنام 'PersonalStats' ذخیره می‌شوند. برای ذخیره و بازیابی پروفایل ها، به کلاسی احتیاج داریم که 'ProfileBase' را ارث بری می‌کند. پس کلاس جدیدی با نام 'AppProfile' بسازید.

```
public class ProfileInfo
{
    public ProfileInfo()
    {
        UserStats = new PersonalStats();
    }
    public DateTime? DateOfBirth { get; set; }
    public PersonalStats UserStats { get; set; }
    public string City { get; set; }
}

public class PersonalStats
{
    public int? Weight { get; set; }
    public int? Height { get; set; }
}

public class AppProfile : ProfileBase
{
    public ProfileInfo ProfileInfo
```



```
{
    get { return (ProfileInfo)GetProperty("ProfileInfo"); }
}
public static AppProfile GetProfile()
{
    return (AppProfile)HttpContext.Current.Profile;
}
public static AppProfile GetProfile(string userName)
{
    return (AppProfile)Create(userName);
}
}
```

پروفایل را در فایل web.config خود فعال کنید. نام کلاسی را که در مرحله قبل ساختید، بعنوان کلاس پیش فرض برای ذخیره و بازیابی پروفایلها معرفی کنید.

```
<profile defaultProvider="DefaultProfileProvider" enabled="true"
    inherits="UniversalProviders_ProfileMigrations.Models.AppProfile">
    <providers>
        .....
    </providers>
</profile>
```

برای دریافت اطلاعات پروفایل از کاربر، فرم وب جدیدی در پوشه Account بسازید و آنرا 'AddProfileData.aspx' نامگذاری کنید.

```
<h2> Add Profile Data for <%= User.Identity.Name %></h2>
<asp:Label Text="" ID="Result" runat="server" />
<div>
    Date of Birth:
    <asp:TextBox runat="server" ID="DateOfBirth"/>
</div>
<div>
    Weight:
    <asp:TextBox runat="server" ID="Weight"/>
</div>
<div>
    Height:
    <asp:TextBox runat="server" ID="Height"/>
</div>
<div>
    City:
    <asp:TextBox runat="server" ID="City"/>
</div>
<div>
    <asp:Button Text="Add Profile" ID="Add" OnClick="Add_Click" runat="server" />
</div>
```

کد زیر را هم به فایل code-behind اضافه کنید.

```
protected void Add_Click(object sender, EventArgs e)
{
    AppProfile profile = AppProfile.GetProfile(User.Identity.Name);
    profile.ProfileInfo.DateOfBirth = DateTime.Parse(DateOfBirth.Text);
    profile.ProfileInfo.UserStats.Weight = Int32.Parse(Weight.Text);
    profile.ProfileInfo.UserStats.Height = Int32.Parse(Height.Text);
    profile.ProfileInfo.City = City.Text;
    profile.Save();
}
```

دقت کنید که فضای نامی که کلاس AppProfile در آن قرار دارد را وارد کرده باشید.

اپلیکیشن را اجرا کنید و کاربر جدیدی با نام 'olduser' بسازید. به صفحه جدید 'AddProfileData' بروید و اطلاعات پروفایل کاربر را وارد کنید.

your logo here

## Add Profile Data for

Date of Birth:

Weight:

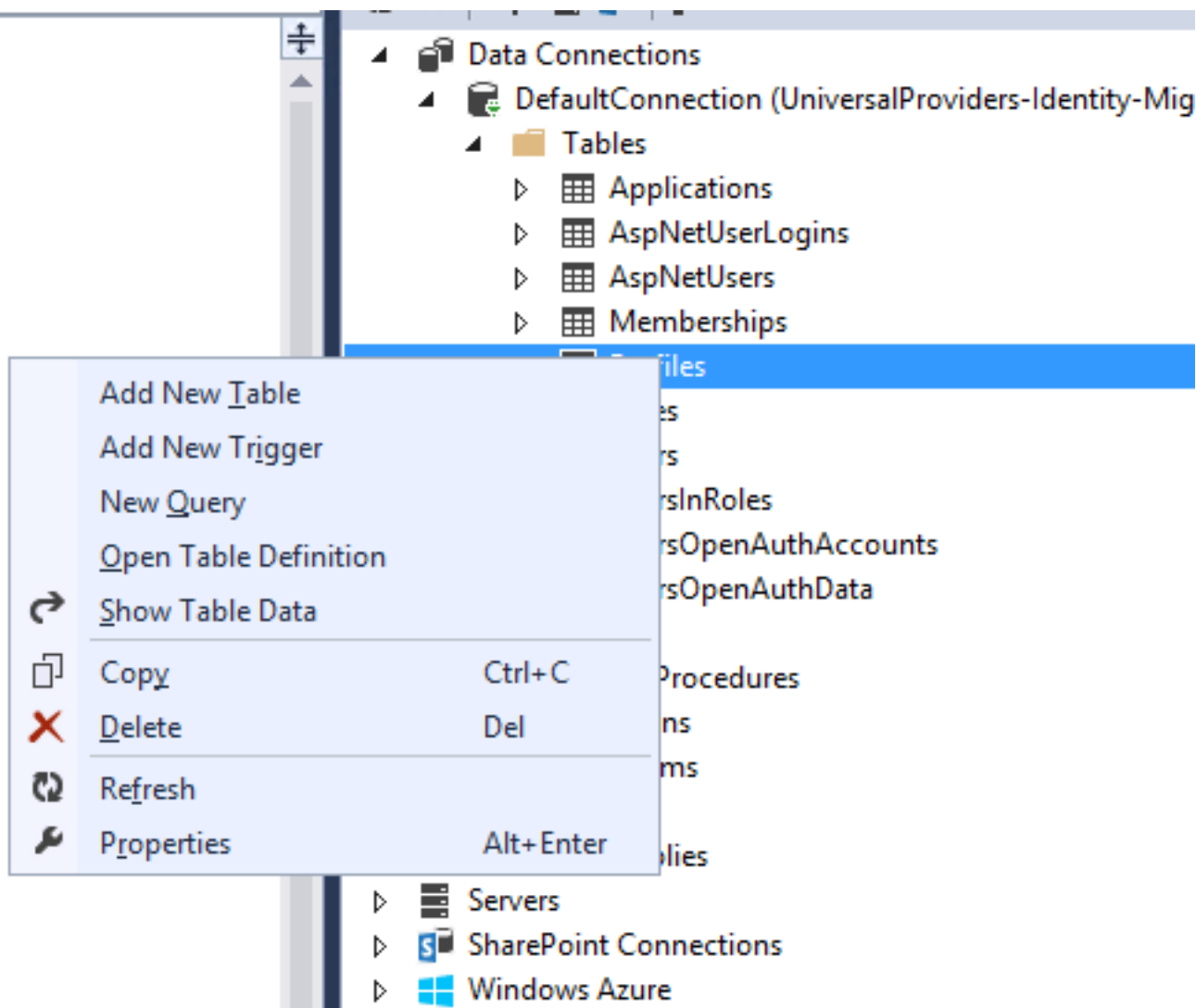
Height:

City:

**Add Profile**

© 2013 - My ASP.NET Application

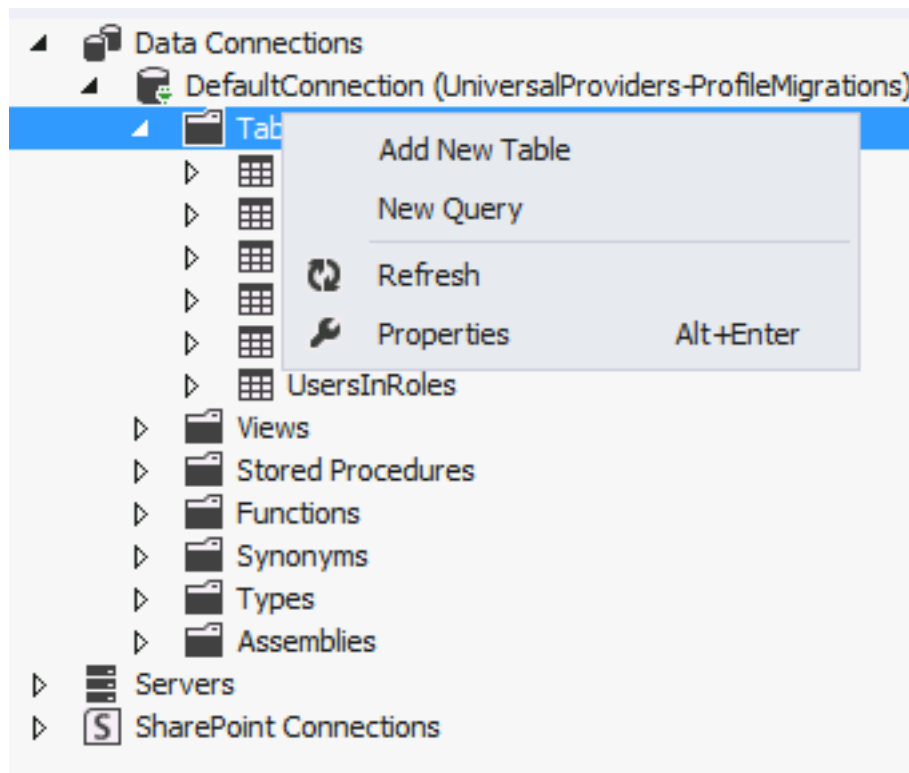
با استفاده از پنجره Server Explorer می‌توانید تایید کنید که اطلاعات پروفایل با فرمت xml در جدول 'Profiles' ذخیره می‌شوند.



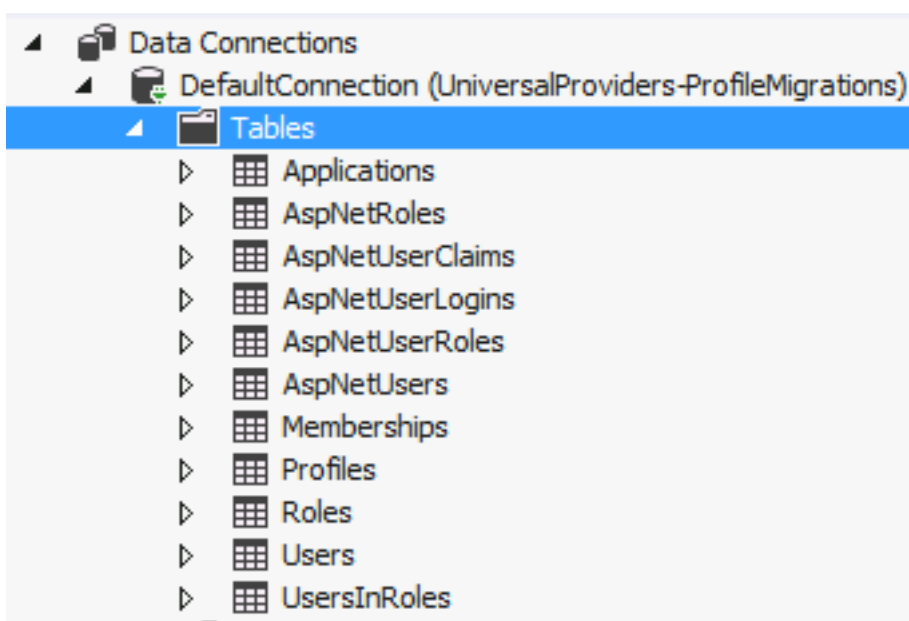
dbo.Profiles [Data]   AddProfileData.aspx.cs   AddProfileData.aspx   Web.config   Defa					
Max Rows: 1000					
	UserId	PropertyNames	PropertyValueS...	PropertyValueB...	LastUpdatedDate
▶	9662-b1889a5d0f30	ProfileInfo:0:326:	<?xml version="..."	<Binary data>	11/26/2013 7:5...
*	NULL	NULL	NULL	NULL	NULL

### مهاجرت الگوی دیتابیس

برای اینکه دیتابیس فعلی بتواند با سیستم ASP.NET Identity کار کند، باید الگوی ASP.NET Identity را بروز رسانی کنیم تا فیلدهای جدیدی که اضافه کردیم را هم در نظر بگیرد. این کار می‌تواند توسط اسکریپت‌های SQL انجام شود، باید جداول جدیدی بسازیم و اطلاعات موجود را به آنها انتقال دهیم. در پنجره 'Server Explorer' گره 'DefaultConnection' را باز کنید تا جداول لیست شوند. روی Tables کلیک راست کنید و 'New Query' را انتخاب کنید.



اسکرپت مورد نیاز را از آدرس <https://raw.githubusercontent.com/suhasj/UniversalProviders-Identity-Migrations/master/Migration.txt> دریافت کرده و آن را اجرا کنید. اگر اتصال خود به دیتابیس را تازه کنید خواهید دید که جداول جدیدی اضافه شده اند. می‌توانید داده‌های این جداول را بررسی کنید تا ببینید چگونه اطلاعات منتقل شده اند.



مهاجرت اپلیکیشن برای استفاده از ASP.NET Identity  
پکیج‌های مورد نیاز برای ASP.NET Identity را نصب کنید:

```
Microsoft.AspNet.Identity.EntityFramework
Microsoft.AspNet.Identity.Owin
Microsoft.Owin.Host.SystemWeb
Microsoft.Owin.Security.Facebook
Microsoft.Owin.Security.Google
Microsoft.Owin.Security.MicrosoftAccount
Microsoft.Owin.Security.Twitter
```

اطلاعات بیشتری درباره مدیریت پکیج‌های NuGet [از اینجا](#) قابل دسترسی هستند.

برای اینکه بتوانیم از الگوی جاری دیتابیس استفاده کنیم، ابتدا باید مدل‌های لازم ASP.NET Identity را تعریف کنیم تا موجودیت‌های دیتابیس را Map کنیم. طبق قرارداد سیستم Identity کلاس‌های مدل یا باید اینترفیس‌های تعریف شده در Identity.Core dll را پیاده سازی کنند، یا می‌توانند پیاده سازی‌های پیش فرضی را که در

Microsoft.AspNet.Identity.EntityFramework وجود دارند گسترش دهند. ما برای نقش‌ها، اطلاعات ورود کاربران و claimها از پیاده سازی‌های پیش فرض استفاده خواهیم کرد. نیاز به استفاده از یک کلاس سفارشی User داریم. پوشه جدیدی در پروژه با نام 'IdentityModels' بسازید. کلاسی با نام 'User' در این پوشه بسازید و کد آن را با لیست زیر تطابق دهید.

```
using Microsoft.AspNet.Identity.EntityFramework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using UniversalProviders_ProfileMigrations.Models;

namespace UniversalProviders_Identity_Migrations
{
    public class User : IdentityUser
    {
        public User()
        {
            CreateDate = DateTime.UtcNow;
            IsApproved = false;
            LastLoginDate = DateTime.UtcNow;
            LastActivityDate = DateTime.UtcNow;
            LastPasswordChangedDate = DateTime.UtcNow;
            Profile = new ProfileInfo();
        }

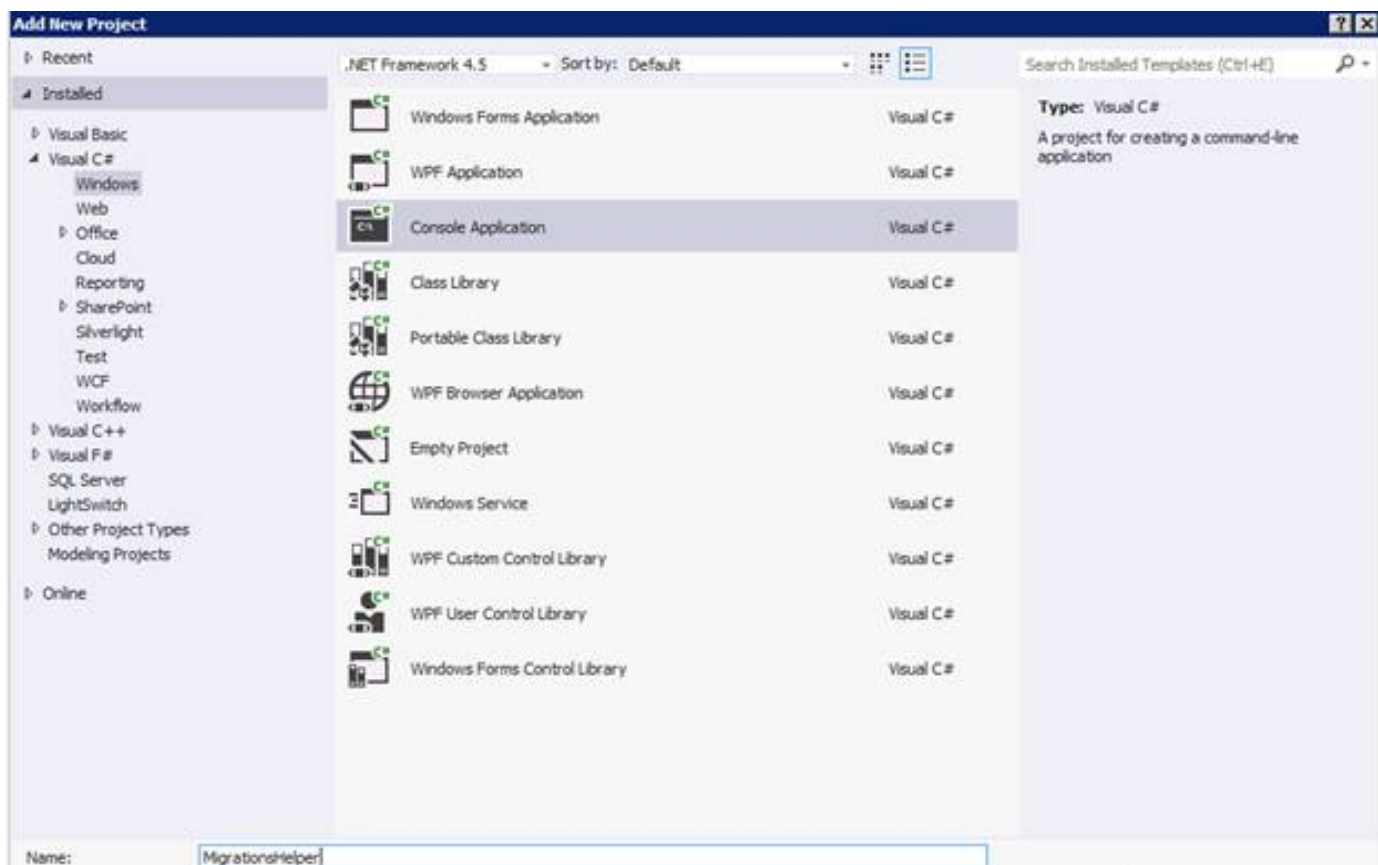
        public System.Guid ApplicationId { get; set; }
        public bool IsAnonymous { get; set; }
        public System.DateTime? LastActivityDate { get; set; }
        public string Email { get; set; }
        public string PasswordQuestion { get; set; }
        public string PasswordAnswer { get; set; }
        public bool IsApproved { get; set; }
        public bool IsLockedOut { get; set; }
        public System.DateTime? CreateDate { get; set; }
        public System.DateTime? LastLoginDate { get; set; }
        public System.DateTime? LastPasswordChangedDate { get; set; }
        public System.DateTime? LastLockoutDate { get; set; }
        public int FailedPasswordAttemptCount { get; set; }
        public System.DateTime? FailedPasswordAttemptWindowStart { get; set; }
        public int FailedPasswordAnswerAttemptCount { get; set; }
        public System.DateTime? FailedPasswordAnswerAttemptWindowStart { get; set; }
        public string Comment { get; set; }
        public ProfileInfo Profile { get; set; }
    }
}
```

دقت کنید که 'ProfileInfo' حالا بعنوان یک خاصیت روی کلاس User تعریف شده است. بنابراین می‌توانیم مستقیماً از کلاس کاربر با اطلاعات پروفایل کار کنیم.

محتویات پوشه‌های IdentityAccount و IdentityModels را از آدرس <https://github.com/suhasj/UniversalProviders-IdentityMigrations/tree/master/UniversalProviders-Identity-Migrations> دریافت و کپی کنید. این فایل‌ها مابقی مدل‌ها، و صفحاتی برای مدیریت کاربران و نقش‌ها در سیستم جدید ASP.NET Identity هستند.

## انتقال داده پروفایل‌ها به جداول جدید

همانطور که گفته شد ابتدا باید داده‌های پروفایل را deserialize کرده و از فرمت xml خارج کنیم، سپس آنها را در ستون‌های جدولAspNetUsers ذخیره کنیم. ستون‌های جدید در مرحله قبل به دیتابیس اضافه شدند، پس تنها کاری که باقی مانده پر کردن این ستون‌ها با داده‌های ضروری است. بدین منظور ما از یک اپلیکیشن کنسول استفاده می‌کنیم که تنها یک بار اجرا خواهد شد، و ستون‌های جدید را با داده‌های لازم پر می‌کند. در solution جاری یک پروژه اپلیکیشن کنسول بسازید.



آخرین نسخه پکیج Entity Framework را نصب کنید. همچنین یک رفرنس به اپلیکیشن وب پروژه بدهید (کلیک راست روی پروژه و گزینه 'Add Reference').

کد زیر را در کلاس Program.cs وارد کنید. این قطعه کد پروفایل تک تک کاربران را می‌خواند و در قالب 'ProfileInfo' آنها را serialize می‌کند و در دیتابیس ذخیره می‌کند.

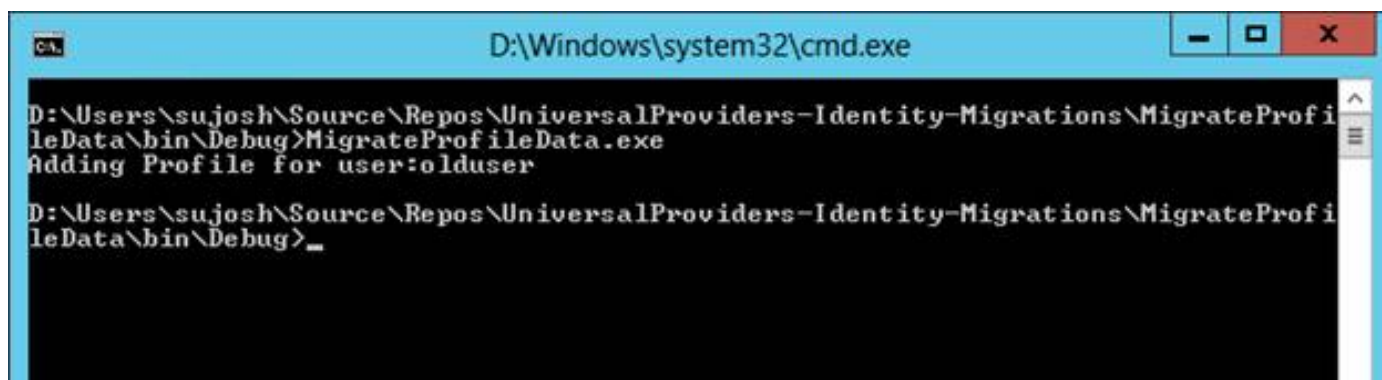
```
public class Program
{
    var dbContext = new ApplicationDbContext();
    foreach (var profile in dbContext.Profiles)
    {
        var stringId = profile.UserId.ToString();
        var user = dbContext.Users.Where(x => x.Id == stringId).FirstOrDefault();
        Console.WriteLine("Adding Profile for user:" + user.UserName);
        var serializer = new XmlSerializer(typeof(ProfileInfo));
        var stringReader = new StringReader(profile.PropertyValueStrings);
        var profileData = serializer.Deserialize(stringReader) as ProfileInfo;
        if (profileData == null)
        {
            Console.WriteLine("Profile data deserialization error for user:" + user.UserName);
        }
        else
    }
}
```

```
{
    {
        user.Profile = profileData;
    }
}
dbContext.SaveChanges();
}
```

برخی از مدل‌های استفاده شده در پوشه 'IdentityModels' تعریف شده اند که در پروژه اپلیکیشن وبمان قرار دارند، بنابراین افزودن فضاهای نام مورد نیاز فراموش نشود.

کد بالا روی دیتابیس‌ای که در پوشه App\_Data وجود دارد کار می‌کند، این دیتابیس در مراحل قبلی در اپلیکیشن وب پروژه ایجاد شد. برای اینکه این دیتابیس را رفرنس کنیم باید رشته اتصال فایل app.config اپلیکیشن کنسول را بروز رسانی کنید. از همان رشته اتصال web.config در اپلیکیشن وب پروژه استفاده کنید. همچنین آدرس فیزیکی کامل را در خاصیت 'AttachDbFilename' وارد کنید.

یک Command Prompt باز کنید و به پوشه bin اپلیکیشن کنسول بالا بروید. فایل اجرایی را اجرا کنید و نتیجه را مانند تصویر زیر بررسی کنید.



```
D:\Windows\system32\cmd.exe

D:\Users\sujosh\Source\Repos\UniversalProviders-Identity-Migrations\MigrateProfileData\bin\Debug>MigrateProfileData.exe
Adding Profile for user:olduser

D:\Users\sujosh\Source\Repos\UniversalProviders-Identity-Migrations\MigrateProfileData\bin\Debug>
```

در پنجره Server Explorer جدول 'AspNetUsers' را باز کنید. حال ستون‌های این جدول باید خواص کلاس مدل را منعکس کنند.

کارایی سیستم را تایید کنید

با استفاده از صفحات جدیدی که برای کار با ASP.NET Identity پیاده سازی شده اند سیستم را تست کنید. با کاربران قدیمی که در دیتابیس قبلی وجود دارند وارد شوید. کاربران باید با همان اطلاعات پیشین بتوانند وارد سیستم شوند. مابقی قابلیت‌ها را هم بررسی کنید. مثلاً افزودن OAuth، ثبت کاربر جدید، تغییر کلمه عبور، افزودن نقش‌ها، تخصیص کاربران به نقش‌ها و غیره. داده‌های پروفایل کاربران قدیمی و جدید همگی باید در جدول کاربران ذخیره شده و بازیابی شوند. جدول قبلی دیگر نباید رفرنس شود.

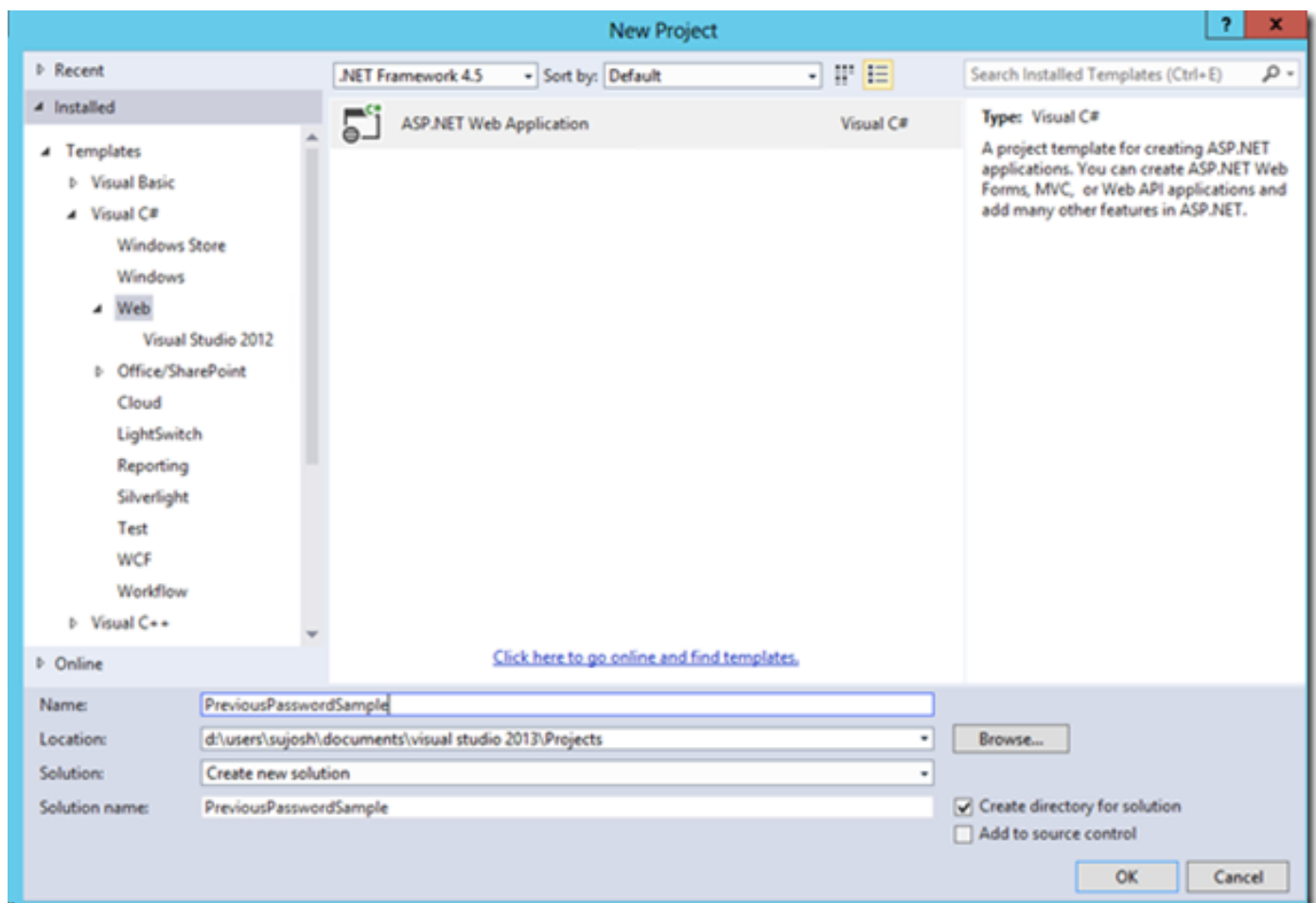
برای فراهم کردن یک تجربه کاربری ایمن‌تر و بهتر، ممکن است بخواهید پیچیدگی password policy را سفارشی سازی کنید. مثلاً ممکن است بخواهید حداقل تعداد کاراکترها را تنظیم کنید، استفاده از چند حروف ویژه را اجباری کنید، جلوگیری از استفاده نام کاربر در کلمه عبور و غیره. برای اطلاعات بیشتر درباره سیاست‌های کلمه عبور به [این لینک](#) مراجعه کنید. بصورت پیش فرض ASP.NET Identity کاربران را وادار می‌کند تا کلمه‌های عبوری بطول حداقل 6 کاراکتر وارد نمایند. در ادامه نحوه افزودن چند خط مشی دیگر را هم بررسی می‌کنیم.

با استفاده از ویژوال استودیو 2013 پروژه جدیدی خواهیم ساخت تا از ASP.NET Identity استفاده کند. مواردی که درباره کلمه‌های عبور می‌خواهیم اعمال کنیم در زیر لیست شده اند.  
تنظیمات پیش فرض باید تغییر کنند تا کلمات عبور حداقل 10 کاراکتر باشند  
کلمه عبور حداقل یک عدد و یک کاراکتر ویژه باید داشته باشد  
امکان استفاده از 5 کلمه عبور اخیری که ثبت شده وجود ندارد

در آخر اپلیکیشن را اجرا می‌کنیم و عملکرد این قوانین جدید را بررسی خواهیم کرد.

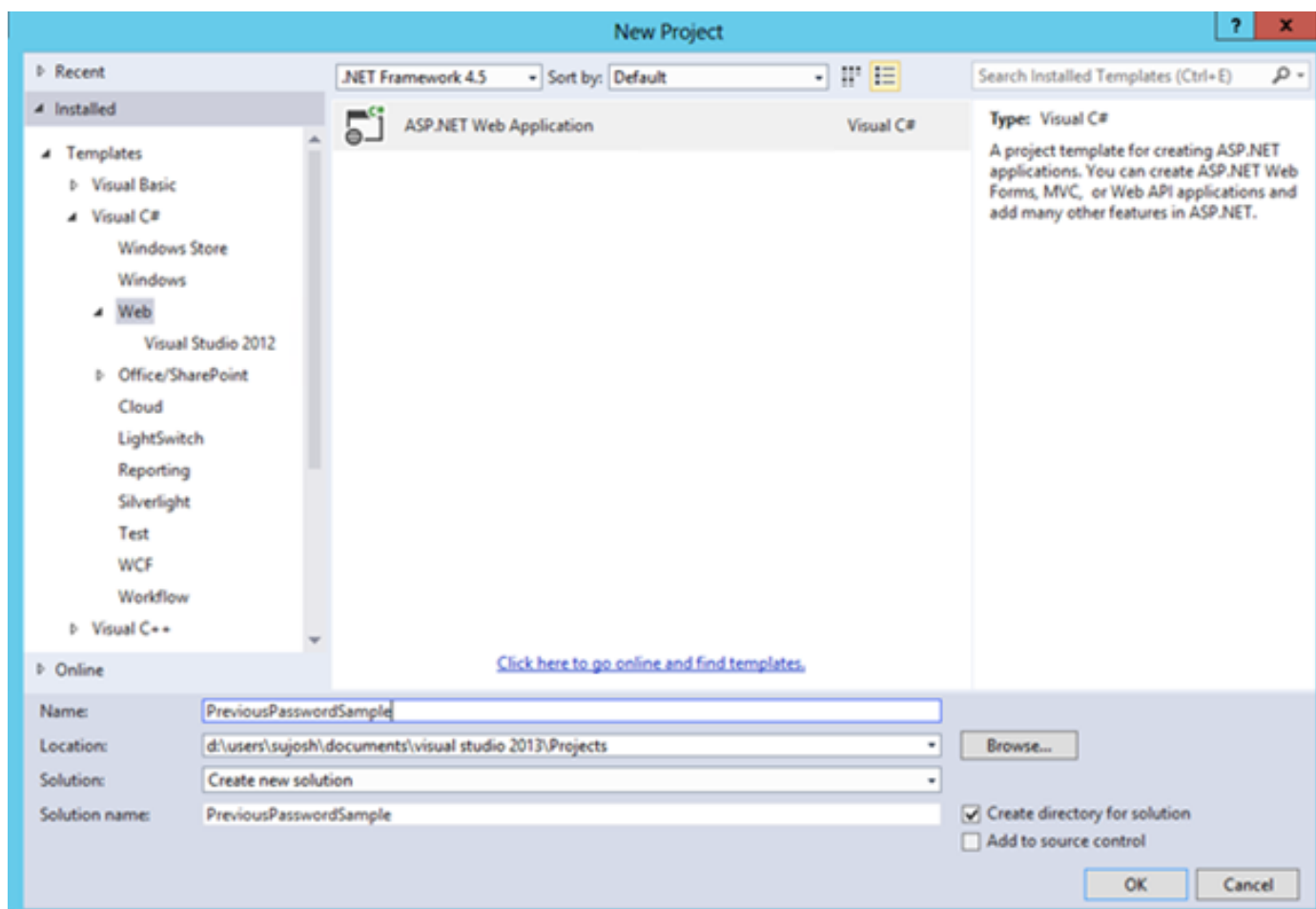
### ایجاد اپلیکیشن جدید

در Visual Studio 2013 اپلیکیشن جدیدی از نوع ASP.NET MVC 4.5 بسازید.





در پنجره Solution Explorer روی نام پروژه کلیک راست کنید و گزینه Manage NuGet Packages را انتخاب کنید. به قسمت **Update** بروید و تمام انتشارات جدید را در صورت وجود نصب کنید.



بگذارید تا به روند کلی ایجاد کاربران جدید در اپلیکیشن نگاهی بیاندازیم. این به ما در شناسایی نیازهای جدیدمان کمک می‌کند. پوشه Controllers حاوی متدهایی برای مدیریت کاربران است. کنترلر Account از کلاس UserManager استفاده می‌کند که در فریم ورک Identity تعریف شده است. این کلاس به نوبه خود از کلاس دیگری بنام UserStore استفاده می‌کند که برای دسترسی و مدیریت داده‌های کاربران استفاده می‌شود. در مثال ما این کلاس از Entity Framework استفاده می‌کند که پیاده سازی پیش فرض است. متد Register POST یک کاربر جدید می‌سازد. متد CreateAsync به طبع متد 'ValidateAsync' را روی خاصیت PasswordValidator فراخوانی می‌کند تا کلمه عبور دریافتی اعتبارسنجی شود.

```
var user = new ApplicationUser() { UserName = model.UserName };
var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)
{
    await SignInAsync(user, isPersistent: false);
    return RedirectToAction("Index", "Home");
}
```

در صورت موفقیت آمیز بودن عملیات ایجاد حساب کاربری، کاربر به سایت وارد می‌شود.

**قانون 1: کلمه‌های عبور باید حداقل 10 کاراکتر باشند**

بصورت پیش فرض خاصیت PasswordValidator در کلاس UserManager به کلاس MinimumLengthValidator تنظیم شده است، که اطمینان حاصل می‌کند کلمه عبور حداقل 6 کاراکتر باشد. هنگام و هله سازی UserManager می‌توانید این مقدار را تغییر دهید. مقدار حداقل کاراکترهای کلمه عبور به دو شکل می‌تواند تعریف شود. راه اول، تغییر کنترلر Account است. در متد سازنده این کنترلر کلاس UserManager و هله سازی می‌شود، همینجا می‌توانید این تغییر را اعمال کنید. راه دوم، ساختن کلاس جدیدی است که از UserManager ارث بری می‌کند. سپس می‌توان این کلاس را در سطح global تعریف کرد. در پوشه IdentityExtensions کلاس جدیدی با نام ApplicationUserManager بسازید.

```
public class ApplicationUserManager : UserManager<ApplicationUser>
{
    public ApplicationUserManager(): base(new UserStore<ApplicationUser>(new ApplicationDbContext()))
    {
        PasswordValidator = new MinimumLengthValidator (10);
    }
}
```

کلاس UserManager یک نمونه از کلاس IUserStore را دریافت می‌کند که پیاده سازی API های مدیریت کاربران است. از آنجا که کلاس UserStore مبتنی بر Entity Framework است، باید آبجکت DbContext را هم پاس دهیم. این کد در واقع همان کدی است که در متد سازنده کنترلر Account وجود دارد. یک مزیت دیگر این روش این است که می‌توانیم متدهای UserManager را بازنویسی (overwrite) کنیم. برای پیاده سازی نیازمندی‌های بعدی دقیقاً همین کار را خواهیم کرد.

حال باید کلاس ApplicationUserManager را در کنترلر Account استفاده کنیم. متد سازنده و خاصیت UserManager را مانند زیر تغییر دهید.

```
public AccountController() : this(new ApplicationUserManager())
{
}

public AccountController(ApplicationUserManager userManager)
{
    UserManager = userManager;
}

public ApplicationUserManager UserManager { get; private set; }
```

حالا داریم از کلاس سفارشی جدیدمان استفاده می‌کنیم. این به ما اجازه می‌دهد مراحل بعدی سفارشی سازی را انجام دهیم، بدون آنکه کدهای موجود در کنترلر از کار بیافتند. اپلیکیشن را اجرا کنید و سعی کنید کاربر محلی جدیدی ثبت نمایید. اگر کلمه عبور وارد شده کمتر از 10 کاراکتر باشد پیغام خطای زیر را دریافت می‌کنید.

# Register.

Create a new account.

- Passwords must be at least 10 characters.

User name

Password

Confirm password

## قانون 2: کلمه‌های عبور باید حداقل یک عدد و یک کاراکتر ویژه داشته باشند

چیزی که در این مرحله نیاز داریم کلاس جدیدی است که اینترفیس `IIdentityValidator` را پیاده سازی می‌کند. چیزی که ما می‌خواهیم اعتبارسنجی کنیم، وجود اعداد و کاراکترهای ویژه در کلمه عبور است، همچنین طول مجاز هم بررسی می‌شود. نهایتاً این قوانین اعتبارسنجی در متد `'ValidateAsync'` بکار گرفته خواهند شد. در پوشه `IdentityExtensions` کلاس جدیدی بنام `CustomPasswordValidator` بسازید و اینترفیس مذکور را پیاده سازی کنید. از آنجا که نوع کلمه عبور رشته (`string`) است از `IIdentityValidator<string>` استفاده می‌کنیم.

```
public class CustomPasswordValidator : IIdentityValidator<string>
{
    public int RequiredLength { get; set; }

    public CustomPasswordValidator(int length)
    {
        RequiredLength = length;
    }

    public Task<IdentityResult> ValidateAsync(string item)
    {
        if (String.IsNullOrEmpty(item) || item.Length < RequiredLength)
        {
            return Task.FromResult(IdentityResult.Failed(String.Format("Password should be of length {0}", RequiredLength)));
        }

        string pattern = @"^(?=.*[0-9])(?=.*[!@#$$%^&*])[0-9a-zA-Z!@#$$%^&*0-9]{10,}$";

        if (!Regex.IsMatch(item, pattern))
        {
            return Task.FromResult(IdentityResult.Failed("Password should have one numeral and one special character"));
        }
    }
}
```

```
return Task.FromResult(IdentityResult.Success);
}
```

در متد **ValidateAsync** بررسی می‌کنیم که طول کلمه عبور معتبر و مجاز است یا خیر. سپس با استفاده از یک RegEx وجود کاراکترهای ویژه و اعداد را بررسی می‌کنیم. دقت کنید که regex استفاده شده تست نشده و تنها بعنوان یک مثال باید در نظر گرفته شود.

قدم بعدی تعریف این اعتبارسنج سفارشی در کلاس UserManager است. باید مقدار خاصیت PasswordValidator را به این کلاس تنظیم کنیم. به کلاس ApplicationUserManager که پیشتر ساختید بروید و مقدار خاصیت PasswordValidator را به CustomPasswordValidator تغییر دهید.

```
public class ApplicationUserManager : UserManager<ApplicationUser>
{
    public ApplicationUserManager() : base(new UserStore<ApplicationUser>(new ApplicationDbContext()))
    {
        PasswordValidator = new CustomPasswordValidator(10);
    }
}
```

هیچ تغییر دیگری در کلاس AccountController لازم نیست. حال سعی کنید کاربر جدید دیگری بسازید، اما اینبار کلمه عبوری وارد کنید که خطای اعتبارسنجی تولید کند. پیغام خطایی مشابه تصویر زیر باید دریافت کنید.

# Register.

## Create a new account.

- Password should have one numeral and one special character

User name

Password

Confirm password

**قانون 3: امکان استفاده از 5 کلمه عبور اخیر ثبت شده وجود ندارد**

هنگامی که کاربران سیستم، کلمه عبور خود را بازنشانی (reset) می کنند یا تغییر می دهند، می توانیم بررسی کنیم که آیا مجدداً از یک کلمه عبور پیشین استفاده کرده اند یا خیر. این بررسی بصورت پیش فرض انجام نمی شود، چرا که سیستم Identity تاریخچه کلمه های عبور کاربران را ذخیره نمی کند. می توانیم در اپلیکیشن خود جدول جدیدی بسازیم و تاریخچه کلمات عبور کاربران را در آن ذخیره کنیم. هر بار که کاربر سعی در بازنشانی یا تغییر کلمه عبور خود دارد، مقدار Hash شده را در جدول تاریخچه بررسی می کنیم.

فایل IdentityModels.cs را باز کنید. مانند لیست زیر، کلاس جدیدی بنام 'PreviousPassword' بسازید.

```
public class PreviousPassword
{
    public PreviousPassword()
    {
        CreateDate = DateTimeOffset.Now;
    }

    [Key, Column(Order = 0)]
    public string PasswordHash { get; set; }
    public DateTimeOffset CreateDate { get; set; }

    [Key, Column(Order = 1)]
    public string UserId { get; set; }
    public virtual ApplicationUser User { get; set; }
}
```

در این کلاس، فیلد 'Password' مقدار Hash شده کلمه عبور را نگاه میدارد و توسط فیلد 'UserId' رفرنس می شود. فیلد 'CreateDate' یک مقدار timestamp ذخیره می کند که تاریخ ثبت کلمه عبور را مشخص می نماید. توسط این فیلد می توانیم تاریخچه کلمات عبور را فیلتر کنیم و مثلاً 5 رکورد آخر را بگیریم.

Entity Framework Code First جدول 'PreviousPasswords' را می سازد و با استفاده از فیلدهای 'Password' و 'UserId' کلید اصلی (composite primary key) را ایجاد می کند. برای اطلاعات بیشتر درباره قراردادهای EF Code First به [این لینک](#) مراجعه کنید. خاصیت جدیدی به کلاس ApplicationUser اضافه کنید تا لیست آخرین کلمات عبور استفاده شده را نگهداری کند.

```
public class ApplicationUser : IdentityUser
{
    public ApplicationUser() : base()
    {
        PreviousUserPasswords = new List<PreviousPassword>();
    }

    public virtual IList<PreviousPassword> PreviousUserPasswords { get; set; }
}
```

همانطور که پیشتر گفته شد، کلاس UserStore پیاده سازی API های لازم برای مدیریت کاربران را در بر می گیرد. هنگامی که کاربر برای نخستین بار در سایت ثبت می شود باید مقدار Hash کلمه عبورش را در جدول تاریخچه کلمات عبور ذخیره کنیم. از آنجا که UserStore بصورت پیش فرض متدی برای چنین عملیاتی معرفی نمی کند، باید یک override تعریف کنیم تا این مراحل را انجام دهیم. پس ابتدا باید کلاس سفارشی جدیدی بسازیم که از UserStore ارث بری کرده و آن را توسعه می دهد. سپس از این کلاس سفارشی در ApplicationUserManager بعنوان پیاده سازی پیش فرض UserStore استفاده می کنیم. پس کلاس جدیدی در پوشه IdentityExtensions ایجاد کنید.

```
public class ApplicationUserStore : UserStore<ApplicationUser>
{
    public ApplicationUserStore(DbContext context) : base(context) { }
```

```

public override async Task CreateAsync(ApplicationUser user)
{
    await base.CreateAsync(user);
    await AddToPreviousPasswordsAsync(user, user.PasswordHash);
}

public Task AddToPreviousPasswordsAsync(ApplicationUser user, string password)
{
    user.PreviousUserPasswords.Add(new PreviousPassword() { UserId = user.Id, PasswordHash = password });
    return UpdateAsync(user);
}
}

```

متد 'AddToPreviousPasswordsAsync' کلمه عبور را در جدول 'PreviousPasswords' ذخیره می‌کند. هرگاه کاربر سعی در بازنشانی یا تغییر کلمه عبورش دارد باید این متد را فراخوانی کنیم. API‌های لازم برای این کار در کلاس UserManager تعریف شده‌اند. باید این متدها را override کنیم و فراخوانی متد مذکور را پیاده کنیم. برای این کار کلاس ApplicationUserManager را باز کنید و متدهای ChangePassword و ResetPassword را بازنویسی کنید.

```

public class ApplicationUserManager : UserManager<ApplicationUser>
{
    private const int PASSWORD_HISTORY_LIMIT = 5;

    public ApplicationUserManager() : base(new ApplicationUserStore(new ApplicationDbContext()))
    {
        PasswordValidator = new CustomPasswordValidator(10);
    }

    public override async Task<IdentityResult> ChangePasswordAsync(string userId, string currentPassword, string newPassword)
    {
        if (await IsPreviousPassword(userId, newPassword))
        {
            return await Task.FromResult(IdentityResult.Failed("Cannot reuse old password"));
        }

        var result = await base.ChangePasswordAsync(userId, currentPassword, newPassword);

        if (result.Succeeded)
        {
            var store = Store as ApplicationUserStore;
            await store.AddToPreviousPasswordsAsync(await FindByIdAsync(userId), PasswordHasher.HashPassword(newPassword));
        }

        return result;
    }

    public override async Task<IdentityResult> ResetPasswordAsync(string userId, string token, string newPassword)
    {
        if (await IsPreviousPassword(userId, newPassword))
        {
            return await Task.FromResult(IdentityResult.Failed("Cannot reuse old password"));
        }

        var result = await base.ResetPasswordAsync(userId, token, newPassword);

        if (result.Succeeded)
        {
            var store = Store as ApplicationUserStore;
            await store.AddToPreviousPasswordsAsync(await FindByIdAsync(userId), PasswordHasher.HashPassword(newPassword));
        }

        return result;
    }

    private async Task<bool> IsPreviousPassword(string userId, string newPassword)
    {
        var user = await FindByIdAsync(userId);
    }
}

```

```

        if (user.PreviousUserPasswords.OrderByDescending(x => x.CreateDate).
            Select(x => x.PasswordHash).Take(PASSWORD_HISTORY_LIMIT)
            .Where(x => PasswordHasher.VerifyHashedPassword(x, newPassword) !=
PasswordVerificationResult.Failed).Any())
        {
            return true;
        }
        return false;
    }
}

```

فیلد 'PASSWORD\_HISTORY\_LIMIT' برای دریافت X رکورد از جدول تاریخچه کلمه عبور استفاده می‌شود. همانطور که می‌بینید از متد سازنده کلاس ApplicationUserStore برای گرفتن متد جدیدمان استفاده کرده ایم. هرگاه کاربری سعی می‌کند کلمه عبورش را بازنشانی کند یا تغییر دهد، کلمه عبورش را با 5 کلمه عبور قبلی استفاده شده مقایسه می‌کنیم و بر این اساس مقدار true/false بر می‌گردانیم.

کاربر جدیدی بسازید و به صفحه **Manage** بروید. حال سعی کنید کلمه عبور را تغییر دهید و از کلمه عبور فعلی برای مقدار جدید استفاده کنید تا خطای اعتبارسنجی تولید شود. پیامی مانند تصویر زیر باید دریافت کنید.

The screenshot shows the 'Manage Account' page. At the top, it says 'You're logged in as foo.' Below that is the 'Change Password Form'. A red error message states: '• Cannot reuse old password'. The form contains three input fields: 'Current password', 'New password', and 'Confirm new password'. Below these fields is a 'Change password' button. At the bottom of the form, there is a link that says 'Use another service to log in.'

سورس کد این مثال را می‌توانید از [این لینک](#) دریافت کنید. نام پروژه Identity-PasswordPolicy است، و زیر قسمت Samples/Identity قرار دارد.