Práctica 1: Code First

Contenido

1. Crear la aplicación

Para mantener la simplicidad, vamos a generar una aplicación de consola básica que use Code First para el acceso a los datos.

- Abra Visual Studio
- **Archivo -> Nuevo -> Proyecto**
- Seleccione **Windows** en el menú de la izquierda y **aplicación de consola**
- Escriba **CodeFirstNewDatabaseSample** como nombre
- Seleccione **Aceptar**

2. Crear el modelo

Vamos a definir un modelo muy sencillo con clases. Solo vamos a definirlas en el archivo Program.cs pero en una aplicación real las clases se dividirían en archivos independientes y, posiblemente, también en un proyecto independiente.

Debajo de la definición de clase Program, en Program.cs, agregue las dos clases siguientes.

```
public class Blog
{
   public int BlogId { get; set; }
   public string Name { get; set; }
```

```
    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public virtual Blog Blog { get; set; }
}
```

Observará que vamos a hacer las dos propiedades de navegación (Blog.Posts y Post.Blog) virtuales. Esto habilita la característica de carga diferida de Entity Framework. La carga diferida significa que el contenido de estas propiedades se cargará automáticamente desde la base de datos al intentar tener acceso.

3. Crear un contexto

Ahora es el momento de definir un contexto derivado que representa una sesión con la base de datos, lo que nos permite consultar y guardar los datos. Definimos un contexto que deriva de System.Data.Entity.DbContext y expone un DbSet<TEntity> con tipo para cada clase en nuestro modelo.

Ahora vamos a comenzar a usar tipos de Entity Framework de modo que necesitamos agregar el paquete de EntityFramework NuGet.

- **Proyecto –> Administrar paquetes de NuGet**
  Nota: si no dispone de la opción **Administrar paquetes de NuGet**, debe instalar la versión más reciente de NuGet
- Seleccione la pestaña **En línea**
- Seleccione el paquete **EntityFramework**
- Haga clic en **Instalar**

Agregue una instrucción using para System.Data.Entity al principio de Program.cs.

```
using System.Data.Entity;
```

Debajo de la clase Post en Program.cs, agregue el siguiente contexto derivado.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
```

```
    public DbSet<Post> Posts { get; set; }
}
```

Esta es una lista completa de lo que ahora debe contener Program.cs.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;

namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}
```

Ese es todo el código que necesitamos para empezar a almacenar y recuperar datos. Obviamente, hay mucho más detrás y le echaremos un vistazo en un momento aunque primero vamos a verlo en acción.

4. Leer y escribir datos

Implemente el método Main en Program.cs como se muestra a continuación. Este código crea una nueva instancia de nuestro contexto y la usa para insertar un nuevo blog. Entonces, usa una consulta LINQ para recuperar todos los blogs de la base de datos ordenados alfabéticamente por el título.

```
class Program
{
   static void Main(string[] args)
   {
      using (var db = new BloggingContext())
      {
         // Create and save a new Blog
         Console.Write("Enter a name for a new Blog: ");
         var name = Console.ReadLine();

         var blog = new Blog { Name = name };
         db.Blogs.Add(blog);
         db.SaveChanges();

         // Display all Blogs from the database
         var query = from b in db.Blogs
                     orderby b.Name
                     select b;

         Console.WriteLine("All blogs in the database:");
         foreach (var item in query)
         {
            Console.WriteLine(item.Name);
         }

         Console.WriteLine("Press any key to exit...");
         Console.ReadKey();
      }
   }
}
```

Ahora puede ejecutar la aplicación y probarla.

```
Escriba un nombre para el nuevo blog: Blog de ADO.NET
Todos los blogs de la base de datos:
Blog de ADO.NET
Presione cualquier tecla para salir.
```

¿Dónde están mis datos?

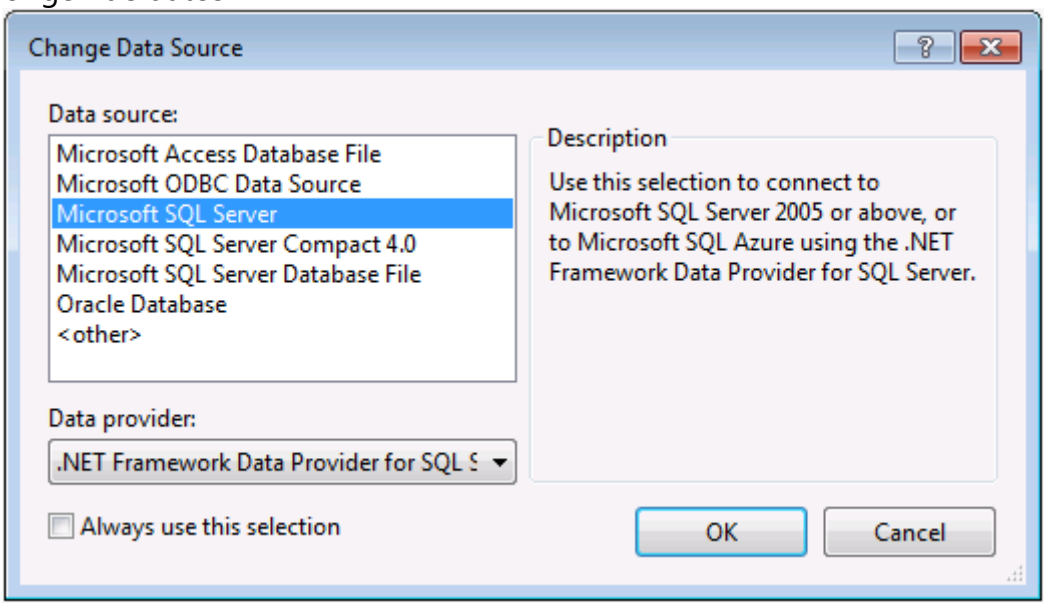
Por convención, DbContext ha creado una base de datos en su lugar.

- Si una instancia local de SQL Express está disponible (se instala de forma predeterminada con Visual Studio 2010), Code First ha creado la base de datos en esa instancia
- Si SQL Express no está disponible, Code First lo intentará y usará LocalDb (se instala de forma predeterminada con Visual Studio 2012)
- Esta base de datos se denomina después del nombre completo del contexto derivado, que en nuestro caso es **CodeFirstNewDatabaseSample.BloggingContext**

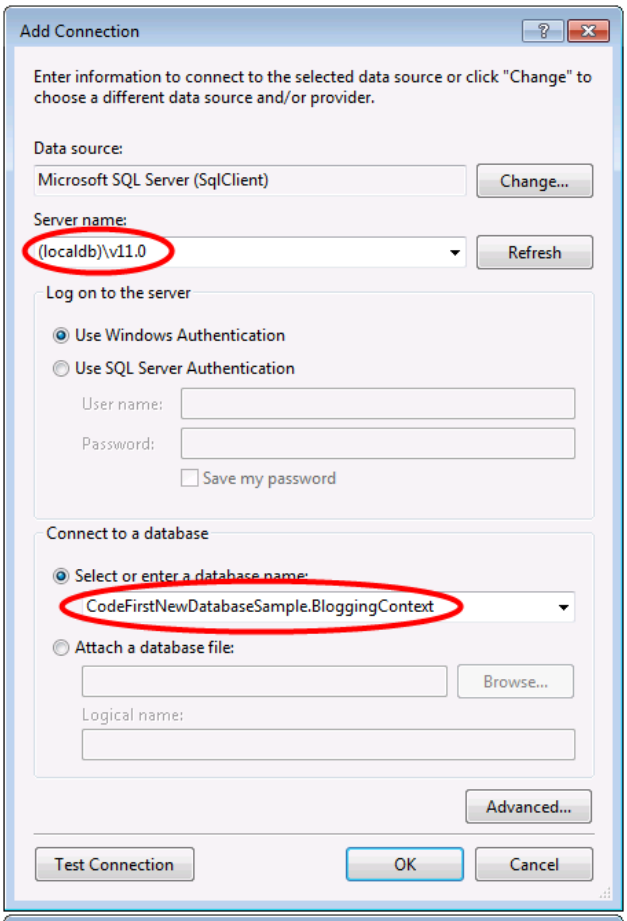
Estas son solo las convenciones predeterminadas y hay varias maneras de cambiar la base de datos que Code First usa; también hay más información disponible en el tema **Cómo detecta DbContext el modelo y la conexión de base de datos** .

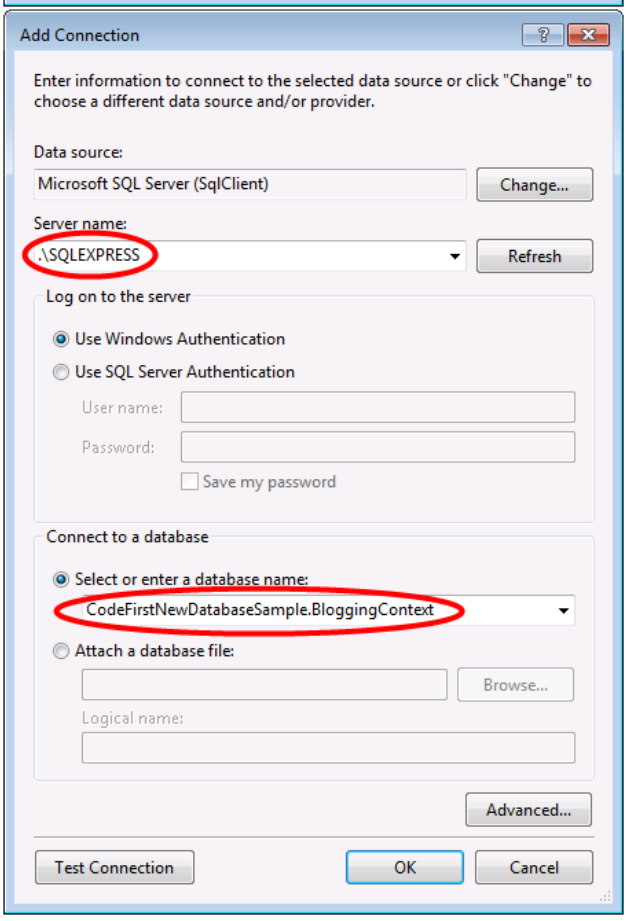
Puede conectarse a esta base de datos mediante el Explorador de servidores en Visual Studio

- **Ver -> Explorador de servidores**
- Haga clic con el botón secundario en **Conexiones de datos** y seleccione **Agregar conexión**
- Si no se ha conectado a una base de datos desde el Explorador de servidores antes, tendrá que seleccionar Microsoft SQL Server como origen de datos

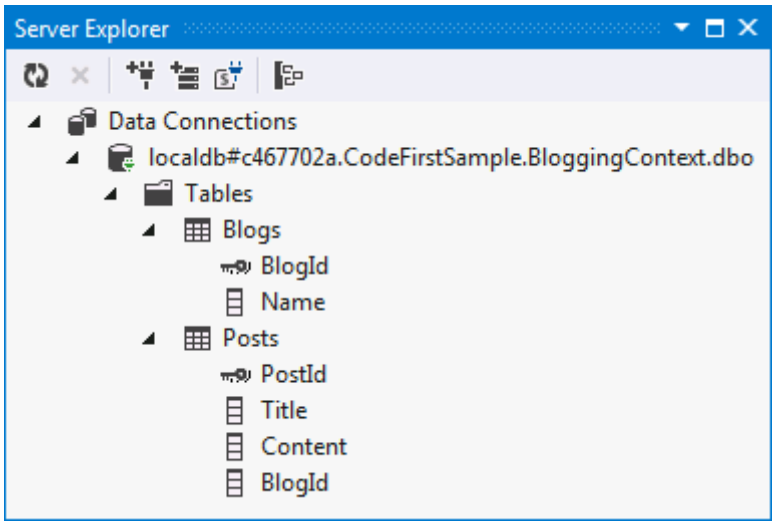


- Conéctese a LocalDb (**(localdb)\v11.0**) o a SQL Express (**.\SQLEXPRESS**), según la que haya instalado


Add Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
Server name:
(localdb)\v11.0
Refresh
Log on to the server
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Save my password
Connect to a database
Select or enter a database name:
CodeFirstNewDatabaseSample.BloggingContext
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel


Add Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
Server name:
.\SQLEXPRESS
Refresh
Log on to the server
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Save my password
Connect to a database
Select or enter a database name:
CodeFirstNewDatabaseSample.BloggingContext
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel

Ahora podemos inspeccionar el esquema que Code First creó.



DbContext determinó las clases que se incluyen en el modelo examinando las propiedades de DbSet que definimos. Después, usa el conjunto predeterminado de convenciones de Code First para determinar los nombres de tabla y columna, determinar los tipos de datos, buscar las claves principales, etc. Más adelante en este tutorial examinaremos cómo puede invalidar estas convenciones.

5. Tratar los cambios en el modelo

Es el momento de realizar algunos cambios en el modelo; cuando realicemos dichos cambios, también necesitaremos actualizar el esquema de la base de datos. Para ello, vamos a usar una característica denominada migraciones de Code First, o migraciones para abreviar.

Las migraciones nos permiten tener un conjunto ordenado de pasos que describen cómo actualizar (y degradar) nuestro esquema de la base de datos. Cada uno de estos pasos, conocido como migración, contiene código que describe los cambios que se aplicarán.

El primer paso es habilitar Migraciones de Code First en nuestro BloggingContext.

- **Herramientas -> Administrador de paquetes de la biblioteca -> Consola del Administrador de paquetes**
- Ejecute el comando **Habilitar-migraciones** en la Consola del Administrador de paquetes
- Una carpeta de migraciones se ha agregado a nuestro proyecto y contiene dos elementos:

- **Configuration.cs**: este archivo contiene las opciones que las migraciones usarán para migrar BloggingContext. No necesitamos cambiar nada para este tutorial pero aquí es donde puede especificar los datos de inicialización, registrar los proveedores para otras bases de datos, cambiar el espacio de nombres en que las migraciones se generan, etc.
- **<timestamp>_InitialCreate.cs**: se trata de la primera migración, representa los cambios que ya se han aplicado a la base de datos para llevarla de una base de datos vacía a una que incluye las tablas de blogs y posts. Aunque permitiéramos que Code First creara automáticamente estas tablas en nuestro lugar, ahora que hemos optado por las migraciones, se han convertido en una migración. Code First también ha registrado en nuestra base de datos local que esta migración ya se ha aplicado. La marca de tiempo en el nombre de archivo se usa para la clasificación.

Ahora vamos a realizar un cambio en nuestro modelo, agregando una propiedad Url a la clase Blog:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public virtual List<Post> Posts { get; set; }
}
```

- Ejecute el comando **Add-Migration AddUrl** en la Consola del Administrador de paquetes.

  El comando Add-Migration comprueba si hay cambios desde la última migración y aplica scaffold a una nueva migración con los cambios encontrados. Podemos dar un nombre a las migraciones; en este caso, le denominamos "AddUrl".

  El código al que se ha aplicado scaffold indica que tenemos que agregar una columna Url, que puede contener datos de cadena, a la tabla dbo.Blogs. Si es necesario, podríamos modificar dicho código pero no se requiere en este caso.

```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;
```

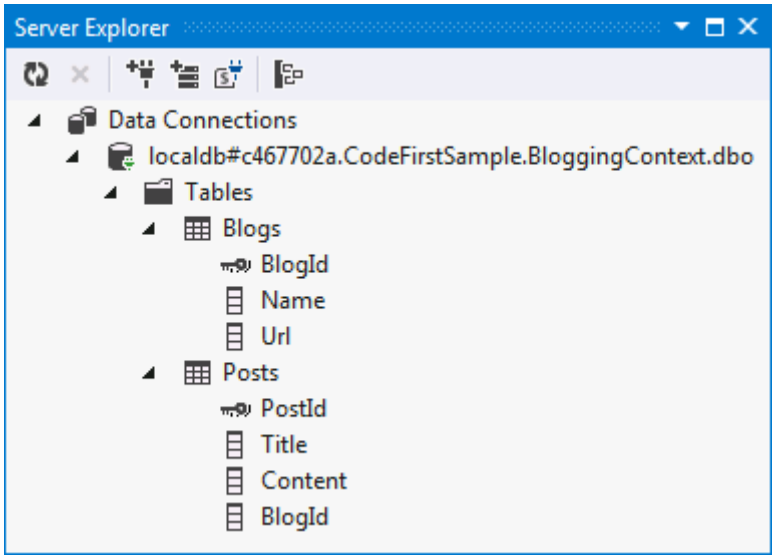
```
    public partial class AddUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

- Ejecute el comando **Update-Database** en la Consola del Administrador de paquetes. Este comando aplicará las migraciones pendientes a la base de datos. Nuestra migración InitialCreate ya se ha aplicado, de modo que las migraciones solo se aplicarán a nuestra nueva migración AddUrl.

  Sugerencia: puede usar el modificador **– Verbose** al llamar a Update-Database para ver el código SQL que se ejecuta en la base de datos.

La nueva columna Url se agrega ahora a la tabla de blogs de la base de datos:



6. Anotaciones de datos

Hasta ahora solo hemos dejado que EF detecte el modelo usando sus convenciones predeterminadas pero habrá ocasiones en que nuestras clases no sigan las convenciones y necesitemos realizar una configuración adicional. Hay

dos opciones para ello; examinaremos las anotaciones de datos en esta sección y luego la API fluida en la sección siguiente.

- Vamos a agregar una clase User a nuestro modelo

```
public class User
{
   public string Username { get; set; }
   public string DisplayName { get; set; }
}
```

- También tenemos que agregar un conjunto a nuestro contexto derivado

```
public class BloggingContext : DbContext
{
   public DbSet<Blog> Blogs { get; set; }
   public DbSet<Post> Posts { get; set; }
   public DbSet<User> Users { get; set; }
}
```

- Si intentáramos agregar una migración, obtendríamos un error que indicaría que "*El usuario EntityType no tiene ninguna clave definida. Defina la clave para este EntityType.*" dado que EF no tiene ninguna manera de saber que el nombre de usuario debe ser la clave principal del usuario.
- Ahora vamos a usar anotaciones de datos de modo que necesitamos agregar una instrucción using al principio de Program.cs

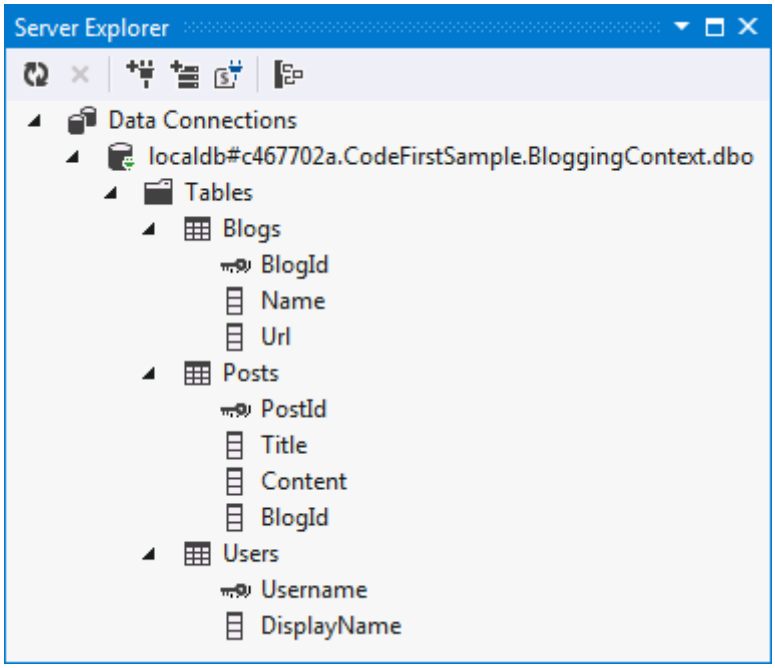
```
using System.ComponentModel.DataAnnotations;
```

- Ahora anote la propiedad Username para identificar que es la clave principal

```
public class User
{
   [Key]
   public string Username { get; set; }
   public string DisplayName { get; set; }
}
```

- Use el comando **Add-Migration AddUser** para scaffold una migración para aplicar estos cambios a la base de datos
- Ejecute el comando **Update-Database** para aplicar la nueva migración a la base de datos

La nueva tabla se agrega ahora a la base de datos:



La lista completa de anotaciones que EF admite es:

- KeyAttribute
- StringLengthAttribute
- MaxLengthAttribute
- ConcurrencyCheckAttribute
- RequiredAttribute
- TimestampAttribute
- ComplexTypeAttribute
- ColumnAttribute
- TableAttribute
- InversePropertyAttribute
- ForeignKeyAttribute
- DatabaseGeneratedAttribute
- NotMappedAttribute

7. API fluida

En la sección anterior examinamos el uso de anotaciones de datos para complementar o invalidar qué se detecta por convención. Otra forma de configurar el modelo es a través de la API fluida de Code First.

La mayor parte de la configuración del modelo se puede hacer con anotaciones de datos simples. La API fluida es una forma más avanzada de especificar la configuración del modelo que abarca todo lo que las anotaciones de datos

pueden hacer además de otras configuraciones avanzadas que no son posibles con las anotaciones de datos. Las anotaciones de datos y la API fluida se pueden usar juntas.

Para tener acceso a la API fluida, invalide el método OnModelCreating en DbContext. Supongamos que desea cambiar el nombre de la columna en que User.DisplayName está almacenada por display_name.

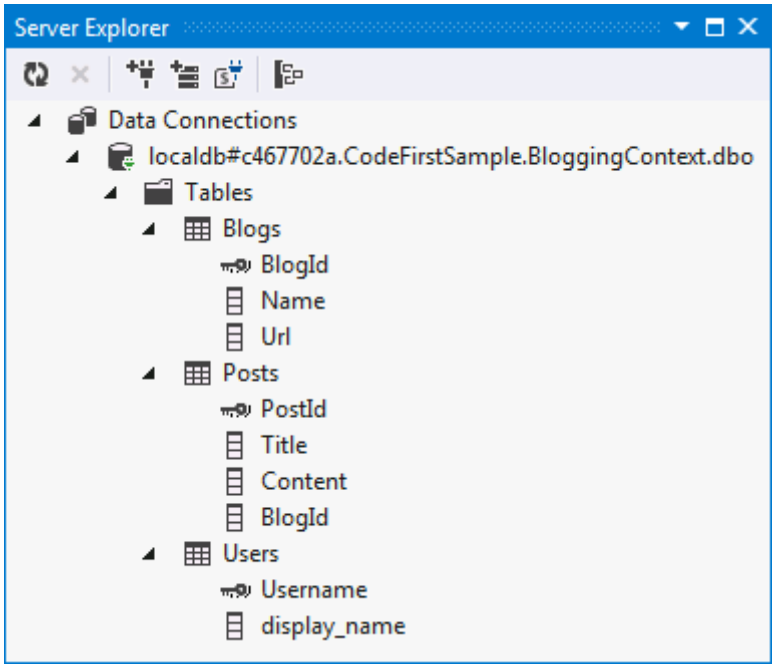
- Invalide el método OnModelCreating en BloggingContext con el siguiente código

```
public class BloggingContext : DbContext
{
   public DbSet<Blog> Blogs { get; set; }
   public DbSet<Post> Posts { get; set; }
   public DbSet<User> Users { get; set; }

   protected override void OnModelCreating(DbModelBuilder modelBuilder)
   {
      modelBuilder.Entity<User>()
         .Property(u => u.DisplayName)
         .HasColumnName("display_name");
   }
}
```

- Use el comando **Add-Migration ChangeDisplayName** para aplicar scaffold a una migración y aplicar estos cambios a la base de datos.
- Ejecute el comando **Update-Database** para aplicar la nueva migración a la base de datos.

El nombre de la columna DisplayName se cambia por display_name:



Resumen

En este tutorial, examinamos el desarrollo de Code First con una nueva base de datos. Definimos un modelo con clases y después usamos ese modelo para crear una base de datos y almacenar y recuperar los datos. Una vez que se creó la base de datos, usamos las Migraciones de Code First para cambiar el esquema a medida que evolucionó nuestro modelo. También vimos cómo configurar un modelo mediante anotaciones de datos y la API fluida.