Code First de Entity Framework le permite usar sus propias clases de dominio para representar el modelo en el que se basa EF para realizar las consultas, el seguimiento de los cambios y las funciones de actualización. Code First usa un modelo de programación conocido como convención sobre la configuración. Lo que significa esto es que Code First supondrá primero que las clases siguen las convenciones que EF usa. En ese caso, EF podrá obtener los detalles que necesita para realizar su trabajo. Sin embargo, si las clases no siguen dichas convenciones, tiene la capacidad de agregar configuraciones a las clases para proporcionar a EF la información que necesita.

Code First le ofrece dos maneras de agregar estas configuraciones a las clases. Una es usar atributos simples denominados DataAnnotations y la otra es emplear la API fluida de Code First, que le proporciona una manera de describir las configuraciones obligatorias, en el código.

Este artículo se centrará en el uso de DataAnnotations (en el espacio de nombres System.ComponentModel.DataAnnotations) para configurar las clases, haciendo hincapié en las configuraciones que suelen requerirse. Diversas aplicaciones .NET, como ASP.NET MVC, también entienden DataAnnotations y esto les permite usar las mismas anotaciones para las validaciones en el lado cliente.

Demostraré las DataAnnotations de Code First con un sencillo par de clases: Blog y Post.

```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set;}
    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

Tal cual están, las clases Blog y Post siguen convenientemente la primera convención de Code First y no requieren ninguna modificación para que EF trabaje con ellas. Pero también puede usar las anotaciones para proporcionar más información a EF sobre las clases y la base de datos a la que se asignan.

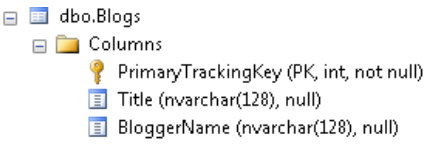
Clave

Entity Framework se basa en cada entidad que tiene un valor de clave que se usa para seguir las entidades. Una de las convenciones de las que depende Code First es el modo en que supone que la propiedad sea la clave de cada una de clases Code First. Esa convención implica buscar una propiedad denominada "Id" u otra que combine el nombre de clase y el "Id" (identificador), como "BlogId". La propiedad se asignará a una columna de clave principal en la base de datos.

Ambas clases, Blog y Post, siguen esta convención. Pero ¿qué ocurriría si no lo hicieran? ¿Qué sucedería si Blog usara el nombre *PrimaryTrackingKe* en lugar de *foo*? Si Code First no encuentra una propiedad que coincida con esta convención, se producirá una excepción debido al requisito de Entity Framework respecto a que debe tener una propiedad de clave. Puede usar la anotación de clave para especificar qué propiedad debe usarse como EntityKey.

```
public class Blog
{
    [Key]
    public int PrimaryTrackingKey { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set;}
    public virtual ICollection<Post> Posts { get; set; }
}
```

Si usa la característica de generación de base de datos de Code First, la tabla Blog tendrá una columna de clave principal denominada PrimaryTrackingKey que también se define como Identity de forma predeterminada.



Required

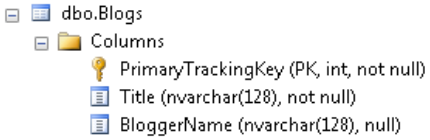
La anotación Required indica a EF que una propiedad determinada es necesaria.

Agregar Required a la propiedad Title hará que EF (y MVC) se aseguren de que la propiedad tiene datos.

```
[Required]
public string Title { get; set; }
```

Sin cambios adicionales de código o de marcas en la aplicación, una aplicación MVC realizará la validación del lado cliente, incluso generando dinámicamente un mensaje con los nombres de propiedad y de anotación.

El atributo Required también afectará a la base de datos generada al hacer que la propiedad asignada no admita valores NULL. Observe que el campo Title ha cambiado a "not null".



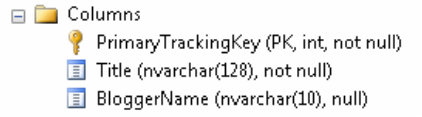
MaxLength y MinLength

Los atributos MaxLength y MinLength permiten especificar validaciones de propiedades adicionales, igual que Required.

Este es el nombre de blogger (BloggerName) con requisitos de longitud. El ejemplo también muestra cómo combinar atributos.

```
[MaxLength(10),MinLength(5)]
public string BloggerName { get; set; }
```

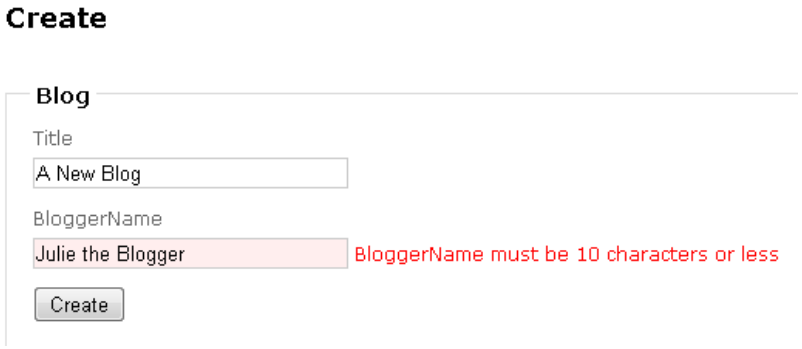
La anotación MaxLength afectará a la base de datos estableciendo la longitud de la propiedad en 10.



La anotación del lado cliente de MVC 3 y la anotación de servidor de EF 4.1 cumplen ambas esta validación y, de nuevo, vuelven a generar dinámicamente un mensaje de error: "El campo BloggerName debe ser un tipo de cadena o de matriz con una longitud máxima de '10'". Este mensaje es un poco largo. Varias anotaciones le permiten especificar un mensaje de error con el atributo ErrorMessage.

```
[MaxLength(10, ErrorMessage="BloggerName debe tener 10 caracteres o menos"),MinLength(5)]
public string BloggerName { get; set; }
```

También puede especificar ErrorMessage en la anotación Required.



NotMapped

La convención de Code First dicta que cada propiedad que sea de un tipo de datos admitido se represente en la base de datos. Pero esto no ocurre siempre en las aplicaciones. Por ejemplo, podría tener una propiedad de la clase Blog que cree un código según los campos Title y BloggerName. Esa la propiedad puede crearse dinámicamente y no necesita almacenarse. Puede marcar una propiedad que no esté asignada a la base de datos con la anotación NotMapped como esta propiedad BlogCode.

```
[NotMapped]
public string BlogCode
{
    get
    {
        return Title.Substring(0, 1) + ":" + BloggerName.Substring(0, 1);
    }
}
```

ComplexType

No es infrecuente describir las entidades de dominio mediante un conjunto de clases y después etiquetar esas clases para describir una entidad completa. Por ejemplo, puede agregar una clase denominada BlogDetails al modelo.

```
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Observe que BlogDetails no tiene ningún tipo de propiedad de clave. En diseño controlado por dominios, se hace referencia a BlogDetails como un objeto de valor. Entity Framework hace referencia a los objetos de valor como tipos complejos. Los tipos complejos no se pueden seguir solos.

Sin embargo, dado que se trata de una propiedad de la clase Blog, BlogDetails será seguida como parte de un objeto Blog. Para que Code First reconozca esto, debe marcar la clase BlogDetails como ComplexType.

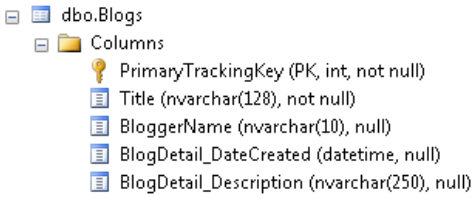
```
[ComplexType]
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Ahora puede agregar una propiedad de la clase Blog para representar el BlogDetails para ese blog.

```
public BlogDetails BlogDetail { get; set; }
```

En la base de datos, la tabla Blog contendrá todas las propiedades del blog incluidas las contenidas en la propiedad BlogDetail. De forma predeterminada, cada una va precedida del nombre del tipo complejo, BlogDetail.



Otra nota interesante es que, aunque la propiedad DateCreated se definió como un DateTime que no admite valores NULL en la clase, el campo de base de datos correspondiente sí los admite. Debe usar la anotación Required si desea afectar al esquema de la base de datos.

ConcurrencyCheck

La anotación ConcurrencyCheck le permite marcar una o varias propiedades que se van a usar para la comprobación de la simultaneidad de la base de datos cuando un usuario modifica o elimina una entidad. Si ha estado trabajando con EF Designer, esto se alinea con configurar el valor ConcurrencyMode de una propiedad en Fixed.

Veamos cómo funciona ConcurrencyCheck agregándola a la propiedad BloggerName.

```
[ConcurrencyCheck, MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"),MinLength(5)]
public string BloggerName { get; set; }
```

Cuando se llama a SaveChanges, debido a la anotación ConcurrencyCheck en el campo BloggerName, el valor original de esa propiedad se usará en la actualización. El comando intentará buscar la fila correcta filtrando no solo por el valor de clave sino también por el valor original de BloggerName. Estas son las partes básicas del comando UPDATE enviado a la base de datos, donde puede ver que el comando que actualizará la fila que tenga un PrimaryTrackingKey es 1 y el BloggerName "Julie", que era el valor original cuando el blog se recuperó desde la base de datos.

```
where (([PrimaryTrackingKey] = @4) and ([BloggerName] = @5))
@4=1,@5=N'Julie'
```

Si alguien ha cambiado el nombre del creador del blog mientras tanto, esta actualización dará error y aparecerá una DbUpdateConcurrencyException que tendrá que controlar.

TimeStamp

Es más frecuente usar los campos rowversion o timestamp para la comprobación de la simultaneidad. Pero, en lugar de usar la anotación ConcurrencyCheck, puede usar la anotación más específica TimeStamp siempre que el tipo de propiedad sea una matriz de bytes. Code First tratará las propiedades de Timestamp igual que las propiedades ConcurrencyCheck, pero también se asegurará de que el campo de base de datos que Code First genera no admite valores NULL. Solo puede tener una propiedad timestamp en una clase determinada.

Agregar la siguiente propiedad a la clase Blog:

```
[Timestamp]
public Byte[] TimeStamp { get; set; }
```

provoca que Code First cree una columna timestamp que no admite valores NULL en la tabla de base de datos.

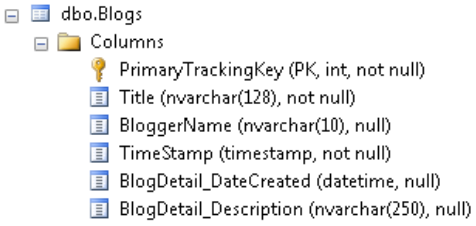


Tabla y columna

Si deja que Code First cree la base de datos, puede que desee cambiar el nombre de las tablas y las columnas que está creando. También puede usar Code First con una base de datos existente. Pero no siempre ocurre que los nombres de las clases y las propiedades del dominio coincidan con los nombres de tablas y columnas de la base de datos.

Mi clase se denomina Blog y, por convención, Code First presupone que se asignará a una tabla denominada Blogs. Si no es así, puede especificar el nombre de la tabla con el atributo Table. Aquí, por ejemplo, la anotación especifica que el nombre de tabla es InternalBlogs.

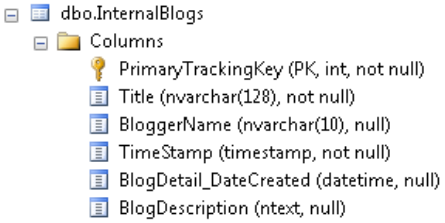
```
[Table("InternalBlogs")]
public class Blog
```

La anotación Column es más adecuada para especificar los atributos de una columna asignada. Puede estipular un nombre, tipo de datos o incluso el orden en que una columna aparece en la tabla. A continuación se muestra un ejemplo del atributo Column.

```
[Column("BlogDescription", TypeName="ntext")]
public String Description {get;set;}
```

No confunda el atributo TypeName de la columna con el DataType DataAnnotation. DataType es una anotación usada para la interfaz de usuario que Code First pasa por alto.

Esta es la tabla una vez regenerada. El nombre de la tabla ha cambiado a InternalBlogs y la columna Description del tipo complejo es ahora BlogDescription. Dado que el nombre se especificó en la anotación, Code First no usará la convención de iniciar el nombre de la columna con el nombre del tipo complejo.



DatabaseGenerated

Una de las características principales de una base de datos es la capacidad de tener propiedades calculadas. Si va a asignar clases de Code First a tablas que contienen columnas calculadas, no deseará que Entity Framework intente actualizar dichas columnas. Pero sí deseará que EF devuelva esos valores de la base de datos después de insertar o actualizar los datos. Puede usar la anotación DatabaseGenerated para marcar aquellas propiedades en la clase junto con la enumeración Computed. Otras enumeraciones son None e Identity.

```
[DatabaseGenerated(DatabaseGenerationOption.Computed)]
public DateTime DateCreated { get; set; }
```

Puede usar la base de datos generada en columnas bytes o timestamp cuando Code First genera la base de datos, en caso contrario, solo debe usar esto al señalar a bases de datos existentes ya que el código primero no podrá determinar la fórmula de la columna calculada.

Antes pudo leer que, de forma predeterminada, una propiedad de clave que es un entero se convertirá en una clave de identidad en la base de datos. Eso sería igual que establecer DatabaseGenerated como DatabaseGenerationOption.Identity. Si no desea usar una clave de identidad, puede establecer el valor en DatabaseGenerationOption.None.

Atributos de relación: InverseProperty y ForeignKey

Nota: *esta página proporciona información sobre la configuración de relaciones en el modelo Code First con anotaciones de datos. Para obtener información general sobre las relaciones en EF y cómo obtener acceso y manipular los datos usando relaciones, vea Propiedades de navegación y relaciones.*

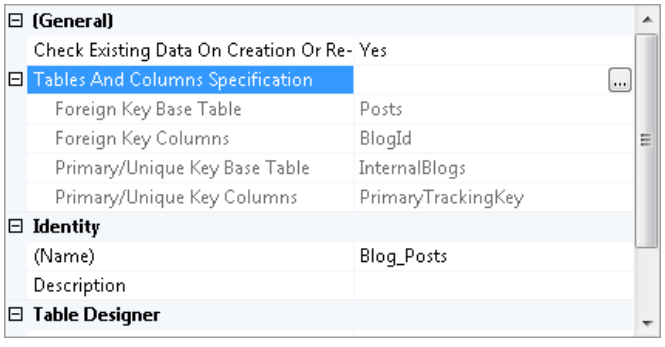
La convención de Code First cuidará de la mayor parte de las relaciones del modelo, pero hay algunos casos en que necesita ayuda.

Cambiar el nombre de la propiedad clave de la clase Blog suponía un problema con su relación con Post.

Al generar la base de datos, Code First ve la propiedad BlogId en la clase Post y la reconoce, por la convención que coincide un nombre de clase además de "Id." como clave externa para la clase Blog. Pero no hay ninguna propiedad BlogId en la clase de blog. La solución para esto es crear una propiedad de navegación en Post y usar Foreign DataAnnotation para ayudar a Code First a entender cómo crear la relación entre las dos clases, mediante la propiedad Post.BlogId, además de cómo especificar restricciones en la base de datos.

```
public class Post
{
        public int Id { get; set; }
        public string Title { get; set; }
        public DateTime DateCreated { get; set; }
        public string Content { get; set; }
        public int BlogId { get; set; }
        [ForeignKey("BlogId")]
        public Blog Blog { get; set; }
        public ICollection<Comment> Comments { get; set; }
}
```

La restricción de la base de datos muestra crea una relación entre InternalBlogs.PrimaryTrackingKey y Posts.BlogId.



Se usa InverseProperty si tiene varias relaciones entre clases.

En la clase Post, puede que desee realizar un seguimiento de quién escribió una entrada en un blog así como quién la editó. A continuación se muestran dos

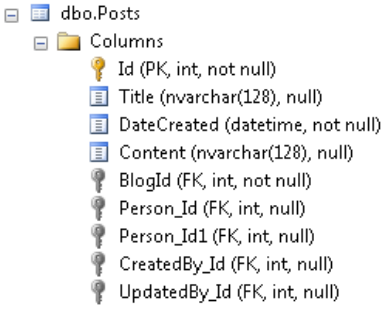
nuevas propiedades de navegación para la clase Post.

```
public Person CreatedBy { get; set; }
public Person UpdatedBy { get; set; }
```

También necesitará agregar la clase Person a la que estas propiedades hacen referencia. La clase Person tiene propiedades de navegación de vuelta al Post, una para todas las entradas escritas por el usuario y otra para las actualizadas por esa persona.

```
public class Person
{
        public int Id { get; set; }
        public string Name { get; set; }
        public List<Post> PostsWritten { get; set; }
        public List<Post> PostsUpdated { get; set; }
}
```

Code First no puede coincidir con las propiedades de las dos clases solas. La tabla de base de datos para Posts debe tener una clave externa para la persona CreatedBy y otra para la persona UpdatedBy pero Code First creará cuatro propiedades de clave externa: Person_Id, Person_Id1, CreatedBy_Id y UpdatedBy_Id.

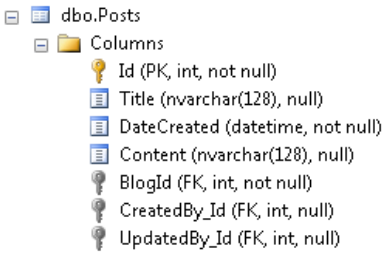


Para solucionar estos problemas, puede usar la anotación InverseProperty para especificar la alineación de las propiedades.

```
[InverseProperty("CreatedBy")]
public List<Post> PostsWritten { get; set; }

[InverseProperty("UpdatedBy")]
public List<Post> PostsUpdated { get; set; }
```

Dado que la propiedad PostsWritten de Person sabe que esto hace referencia al tipo Post, generará la relación con Post.CreatedBy. De igual forma, PostsUpdated se conectará a Post.UpdatedBy. Y Code First no creará claves externas adicionales.



Resumen

DataAnnotations no solo le permite describir la validación del lado cliente y servidor en las clases de Code First, sino que también le permite mejorar e incluso corregir las suposiciones que Code First realizará sobre las clases basándose en sus convenciones. Con DataAnnotations no solo puede controlar la generación del esquema de la base de datos sino que también puede asignar las clases de Code First a una base de datos existente.

Aunque son muy flexibles, tenga en cuenta que DataAnnotations solo proporciona los cambios de configuración más necesarios que puede realizar en las clases de Code First. Para configurar las clases para algunos de los casos extremos, debe buscar un mecanismo alternativo de configuración, la API fluida de Code First.