Práctica 2: Code First BBDD existente

Contenido

Practica 1: Code First BBDD existente 1

1. Create an Existing Database ..... **¡Error! Marcador no definido.**
2. Create the Application ..... **¡Error! Marcador no definido.**
3. Reverse Engineer Model 3

Configuration file 5

Derived Context 6

Model classes ..... **¡Error! Marcador no definido.**

4. Reading & Writing Data ..... **¡Error! Marcador no definido.**

Customizing the Scaffolded Code 8

What if My Database Changes? 8

Using Code First Migrations with an Existing Database 8

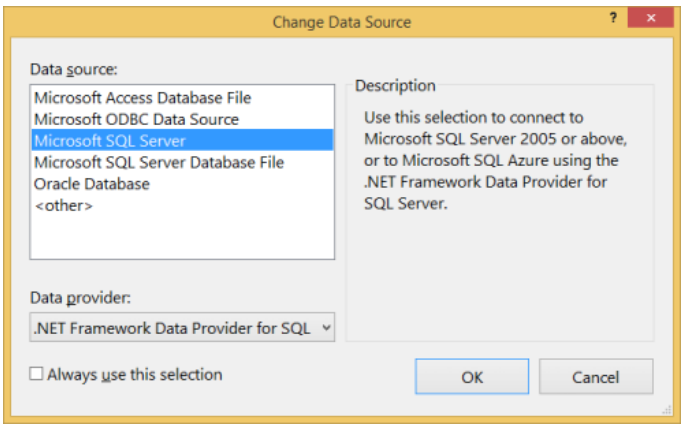
Resumen 8

1. Crear la Base de Datos

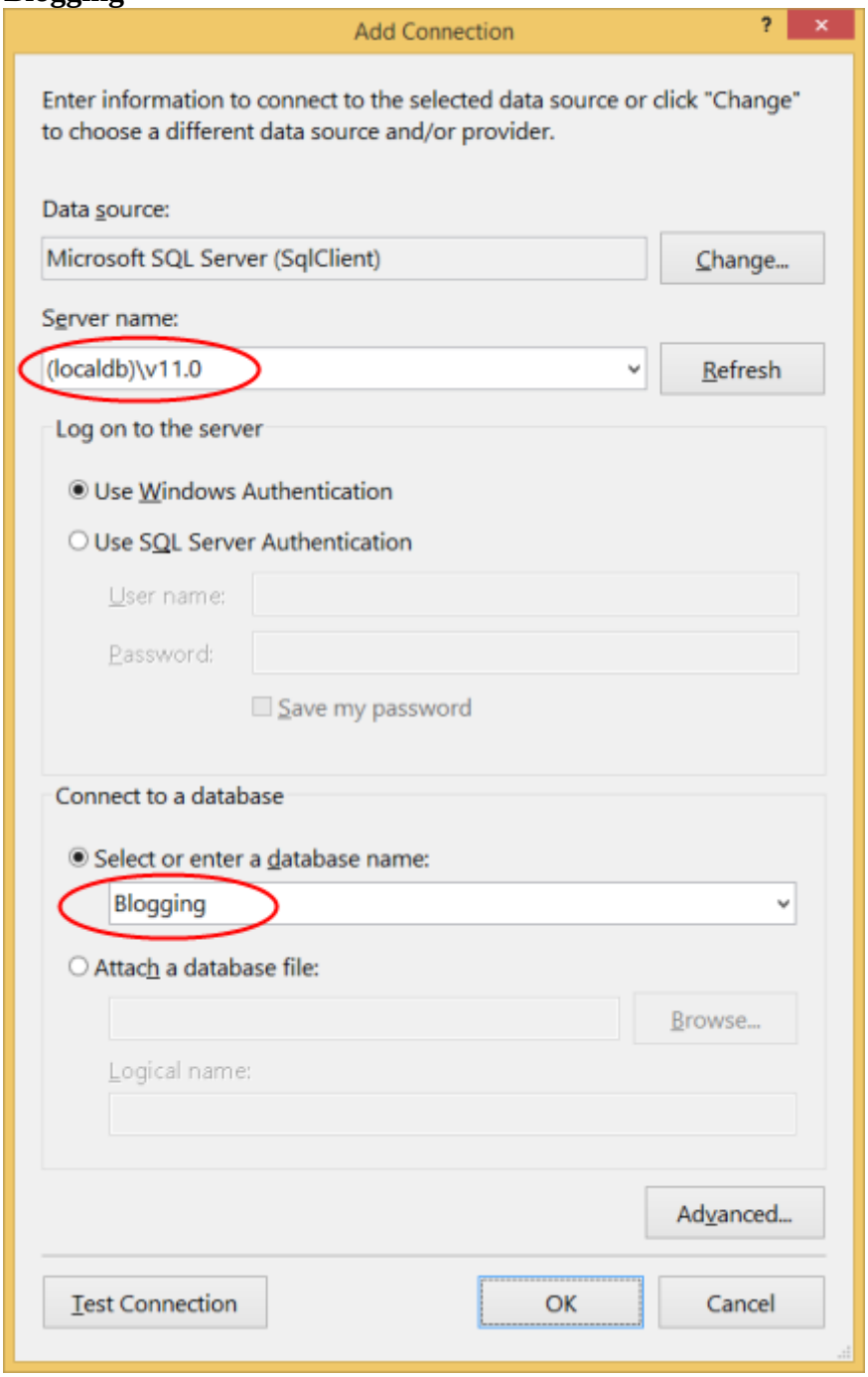
Normalmente la base de datos esta creada pero para esta práctica la crearemos ahora.

Generar la BBDD.

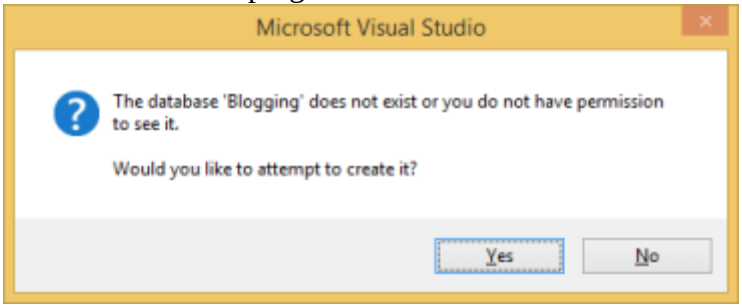
- Abrir Visual Studio
- **View -> Server Explorer**
- Botón derecho **Data Connections -> Add Connection…**
- Si no tiene conectado a una base de datos desde explorador de servidores antes de que usted tendrá que seleccionar Microsoft SQL Server como origen de datos
-



- Conectar con la instancia LocalDb (**(localdb)\v11.0**), y darle el nombre **Blogging**



- Seleccionar **OK** si pregunta si desea crear una nueva BBDD seleccionar **Yes**



- LA nueva BBDD aparecerá en Server Explorer, darle botón derecho y seleccionar **New Query**

- Copiar el Código SQL siguiente en la nueva consulta y Ejecutarlo

```
CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url]  NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId])
REFERENCES [dbo].[Blogs] ([BlogId]) ON DELETE CASCADE
);

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('The Visual Studio Blog', 'http://blogs.msdn.com/visualstudio/
')

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('.NET Framework Blog', 'http://blogs.msdn.com/dotnet/')
```

2. Crear la Aplicación

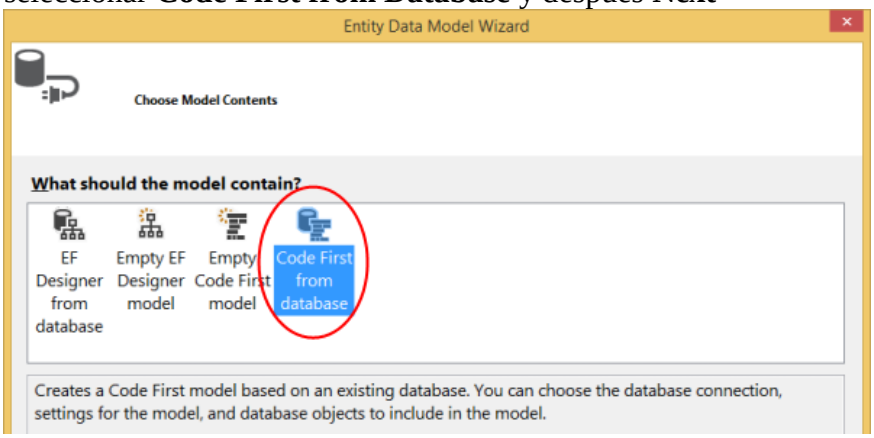
- Abrir Visual Studio
- **File -> New -> Project…**
- Seleccionar **Windows** en el menú de la izquierda y seleccionar la plantilla **Console Application**
- Escribir **CodeFirstExistingDatabaseSample** en el nombre de la aplicación
- Seleccionar **OK**

3. Modelo de ingeniería Inversa

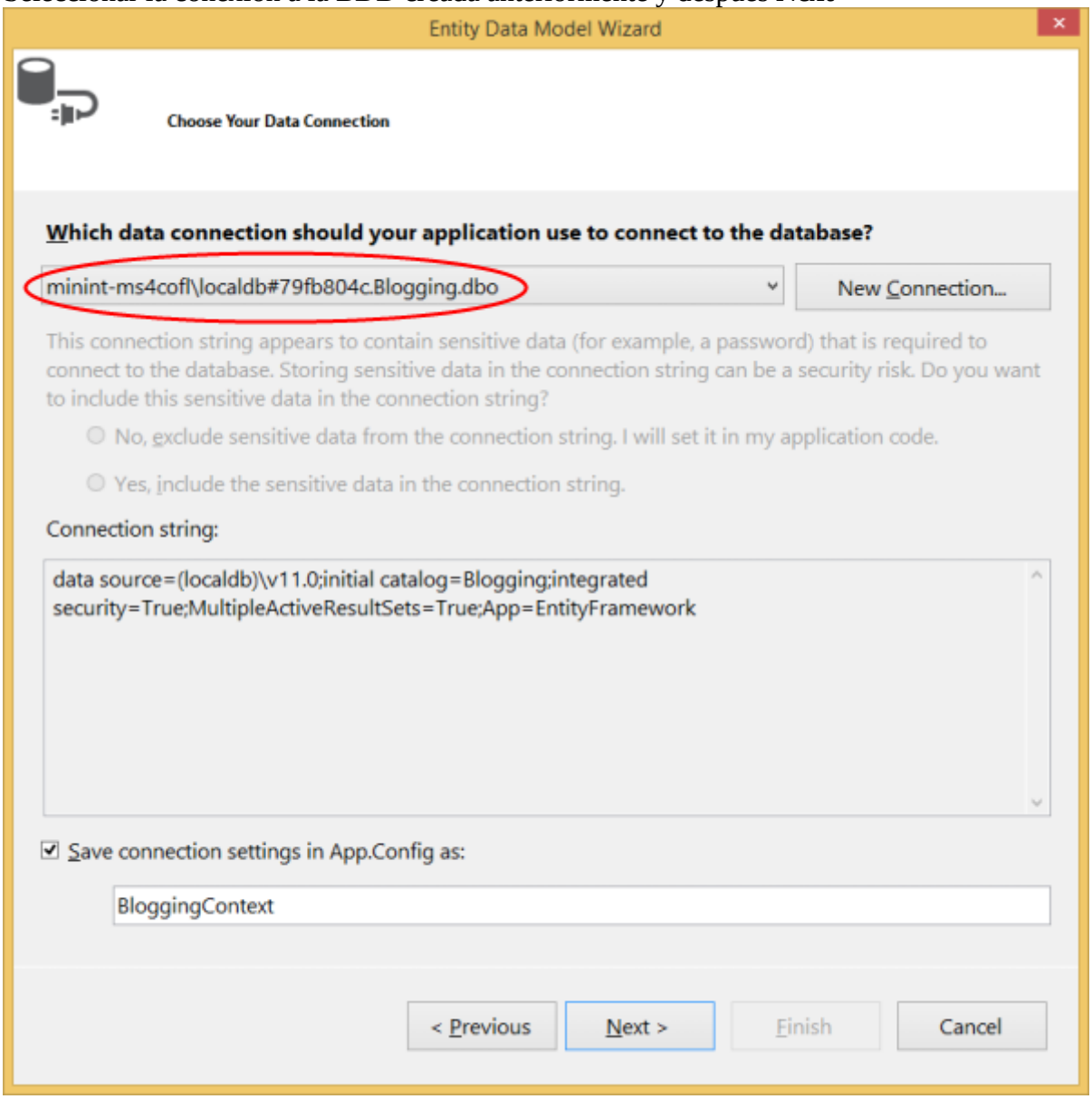
Se van a usar las herramientas Entity Framework Tools para Visual Studio para generar el codigo inicial para mapear la base de datos. Estas herramientas generan código que se puede escribir a mano si se desea.

- **Project -> Add New Item…**
- Seleccionar **Data** en el menú de la izquierda y después elegir la plantilla **ADO.NET Entity Data Model**
- Escribir **BloggingContext** en el nombre y seleccionar **OK**
- Se inicia el asistente de **Entity Data Model**

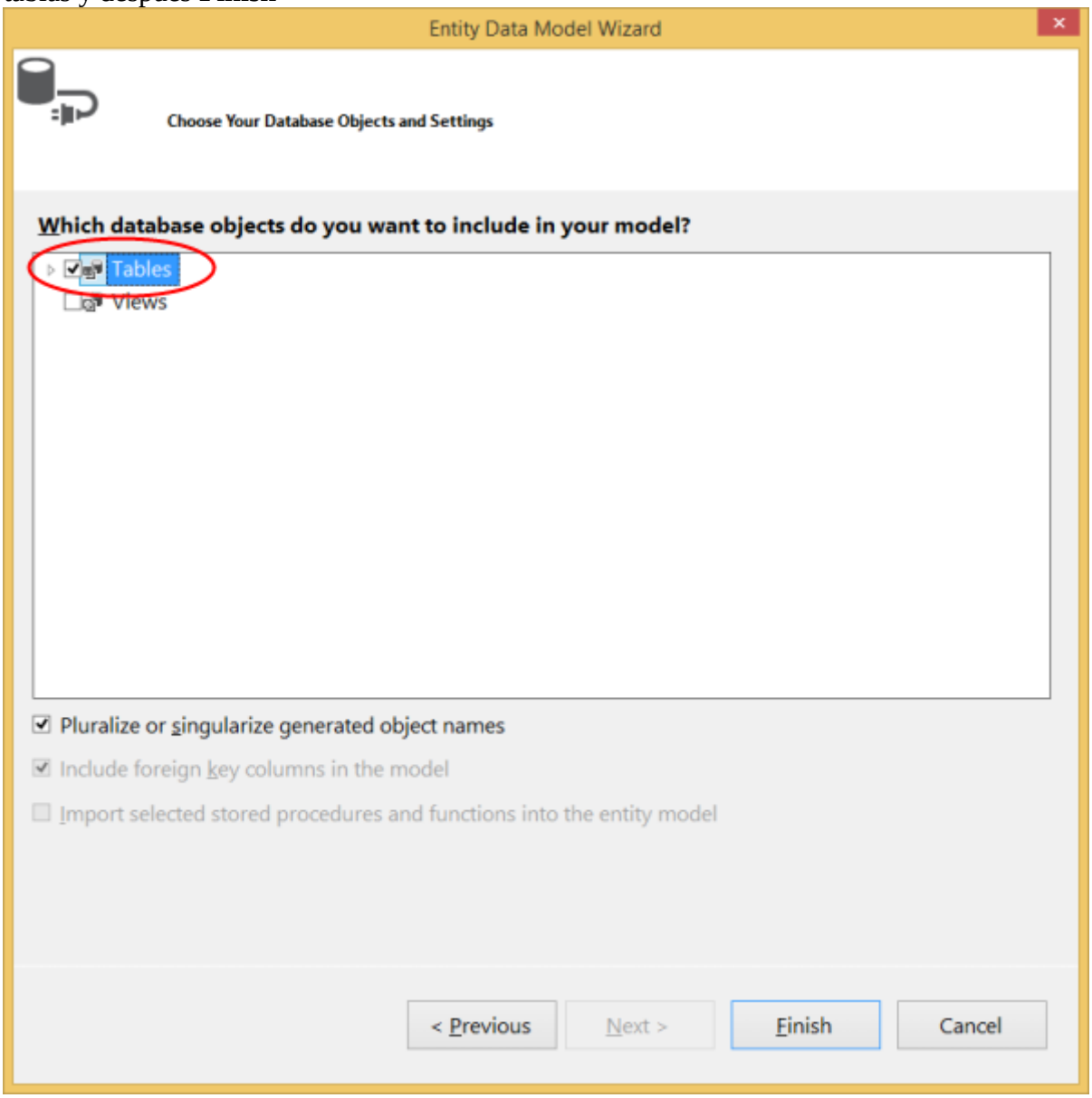
- seleccionar **Code First from Database** y después **Next**



- Seleccionar la conexión a la BDD creada anteriormente y después **Next**



- Seleccionar la casilla de verificación de las tablas **Tables** para importar todas las tablas y después **Finish**



El proceso de ingeniería inversa completa un número de elementos que deben añadirse al proyecto. Veamos cuales son,.

Configuration file

Se añade un fichero App.config al proyecto, este fichero contiene la cadena de conexión a la BBDD.

```
<connectionStrings>
  <add
    name="BloggingContext"
    connectionString="data source=(localdb)\v11.0;initial catalog=Blog
ging;integrated security=True;MultipleActiveResultSets=True;App=Entity
Framework"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

You'll notice some other settings in the configuration file too, these are default EF settings that tell Code First where to create databases. Since we are mapping to an existing database these setting will be ignored in our application.

Contexto derivado

Se añade la clase **BloggingContext** al proyecto. El contexto representa una sesión a la BBDD, que permite recuperar y guardar datos.

EL contexto contiene una propiedad del tipo **DbSet<TEntity>** para cada tipo en el modelo. Debe notarse que el constructor por defecto llama a la clase base pasándole como parámetro el nombre de la conexión. Esto le comunica a CodeFirst que conexión debe cargar del fichero de configuración para este contexto,

```
public partial class BloggingContext : DbContext
    {
        public BloggingContext()
            : base("name=BloggingContext")
        {
        }

        public virtual DbSet<Blog> Blogs { get; set; }
        public virtual DbSet<Post> Posts { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBu
ilder)
        {
        }
    }
```

*You should always use the **name=** syntax when you are using a connection string in the config file. This ensures that if the connection string is not present then Entity Framework will throw rather than creating a new database by convention.*

Clases del modelo

Finalmente se han añadido las clases **Blog** y **Post** al proyecto. Son las clases del dominio que forman nuestro modelo. Podrá ver las anotaciones de datos aplicado a las clases para especificar la configuración donde las convenciones de Code First no se alinean con la estructura de la base de datos existente. Por ejemplo, se ve que la anotación **StringLength** en **Blog.Name** y **Blog.Url** tiene una longitud máxima de **200** en la BBDD (the Code First default is to use the maximun length supported by the database provider - **nvarchar(max)** in SQL Server).

```
public partial class Blog
{
    public Blog()
    {
        Posts = new HashSet<Post>();
    }

    public int BlogId { get; set; }

    [StringLength(200)]
    public string Name { get; set; }
```

```
    [StringLength(200)]
    public string Url { get; set; }

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

4. Leer y escribir datos

Ahora que tenemos un modelo es el momento de usarlo para acceder a algunos datos. Implementando el método Main en Program.cs como se muestra a continuación. Este código crea una nueva instancia de nuestro contexto y luego lo utiliza para insertar un nuevo Blog. Entonces usa una consulta LINQ para recuperar todos los Blogs de la base de datos ordenado alfabéticamente por 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();
        }
    }
}
```

Ejecutar la aplicación y ver el resultado.

```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
.NET Framework Blog
ADO.NET Blog
The Visual Studio Blog
Press any key to exit...
```

Customizing the Scaffolded Code

For information on customizing the code that is generated by the wizard, see Customizing Code First to an Existing Database.

What if My Database Changes?

The Code First to Database wizard is designed to generate a starting point set of classes that you can then tweak and modify. If your database schema changes you can either manually edit the classes or perform another reverse engineer to overwrite the classes.

Using Code First Migrations with an Existing Database

If you want to use Code First Migrations with your existing database, see Code First Migrations with an existing database.

Resumen

En este tutorial analizamos código primer desarrollo de una base de datos existente. Utilizamos las herramientas de marco de entidad de Visual Studio para invertir un conjunto de clases que asignan a la base de datos y podría ser utilizado para almacenar y recuperar datos.