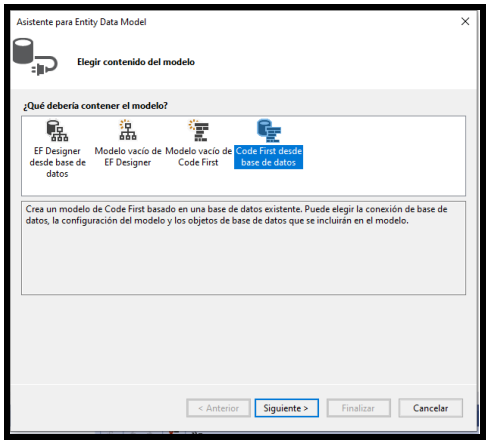
El flujo de trabajo **Code First** (***único flujo o modo de trabajo soportado por EF Core***) permite definir el modelo mediante clases de C# sin emplear complejos y pesados archivos EDMX para mapear objetos a la base de datos, por lo que todo es más ágil. Opcionalmente, se puede realizar una configuración adicional de los atributos mediante anotaciones de datos *(DataAnnotations)* o mediante una API fluida *(Fluent API)*.

Escenario 1: La base de datos existe

A continuación se muestra cómo usar Entity Framework Tools de Visual Studio para realizar ingeniería inversa a partir de conjunto de clases que se asignan a la base de datos y que se pueden usar para almacenar y recuperar datos.

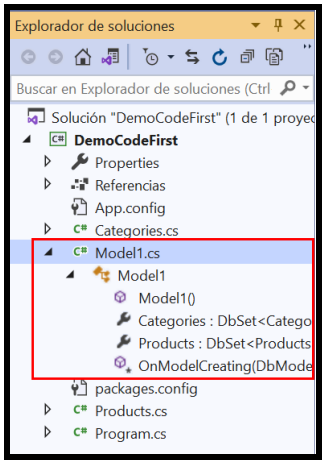
Paso 1 - Crear el modelo de datos

Desde explorador de soluciones / Agregar → Nuevo elemento / seleccionar el modelo de datos de entidad de ADO.NET. Siguiendo el asistente podremos elegir la conexión de base de datos, la configuración del modelo y los objetos de base de datos que se incluirán en el modelo.



Al finalizarlo, en el explorador de soluciones encontraremos:

La clase de contexto (DbContex)




```
1 referencia
public partial class Model1 : DbContext
{
    0 referencias
    public Model1()
        : base("name=Model1")
    {
    }

    0 referencias
    public virtual DbSet<Categories> Categories { get; set; }
    0 referencias
    public virtual DbSet<Products> Products { get; set; }

    0 referencias
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Products>()
            .Property(e => e.UnitPrice)
            .HasPrecision(19, 4);
    }
}
```

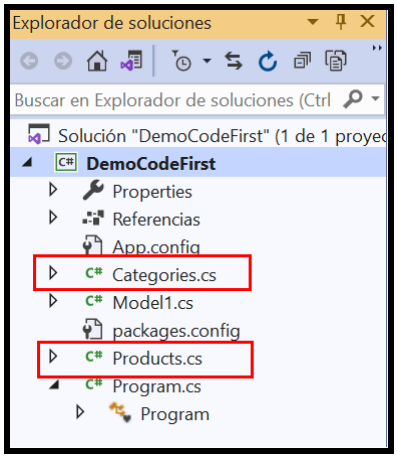
El contexto representa una sesión con la base de datos, lo que nos permite consultar y guardar datos. Observa que:

- El contexto expone una propiedad **DbSet** para cada tipo del modelo. Ej. *DbSet<Categories> Categories*
- El constructor predeterminado llama a un constructor base mediante la sintaxis “name= Model1”. Esto indica a Code First que la cadena de conexión que se va a usar para este contexto se debe cargar desde el archivo de configuración **app Config**, donde encontramos:

```
<connectionStrings>
  <add
    name="Model1"
    connectionString="data source=(localdb)\MSSQLLocalDB;initial catalog=Northwind;integrated
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

La clases del modelo

Son las clases de dominio de nuestro modelo




```
3 referencias
public partial class Categories
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
    0 referencias
    public Categories()
    {
        Products = new HashSet<Products>();
    }

    [Key]
    0 referencias
    public int CategoryID { get; set; }

    [Required]
    [StringLength(15)]
    0 referencias
    public string CategoryName { get; set; }

    [Column(TypeName = "ntext")]
    0 referencias
    public string Description { get; set; }

    [Column(TypeName = "image")]
    0 referencias
    public byte[] Picture { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
    1 referencia
    public virtual ICollection<Products> Products { get; set; }
}
```

Uso del modelo creado - Ejemplo

El siguiente código crea una nueva instancia de nuestro contexto y la usa para insertar una nueva categoría. A continuación, mediante una consulta LINQ recupera todas las categorías de la base de datos ordenadas alfabéticamente por nombre.

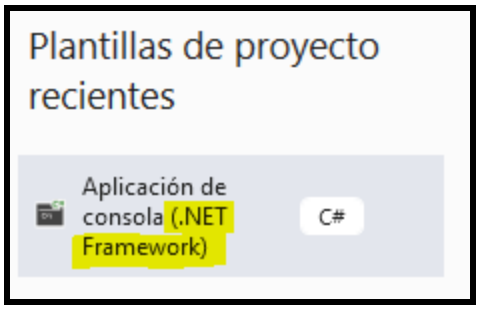
```
using (var db = new Model1())
        {
            // Insertar una nueva categoría
            Console.Write("Introduce un nombre para una nueva categoría: ");
            var name = Console.ReadLine();
            var cat = new Categories { CategoryName = name };
            db.Categories.Add(cat);
            db.SaveChanges();
```

```
        // Muestra todas las categorías de la BD
        var query = from b in db.Categories
                    orderby b.CategoryName
                    select b;

        Console.WriteLine("Todas las categorías de la database:");
        foreach (var item in query)
        {
            Console.WriteLine(item.CategoryName);
        }

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

fuente: Code First a una base de datos existente

Escenario 2: La base de datos No existe

Este escenario incluye como destino una base de datos que no existe y que Code First creará *(también podría ser una base de datos vacía a la que Code First agrega nuevas tablas)*. Code First nos permite definir el modelo mediante clases de C#. Opcionalmente, igual que en el escenario 1 se puede realizar una configuración adicional mediante atributos en las clases y propiedades o mediante una API fluida.



Paso 1 - Creación del modelo

En el ejemplo se define el modelo en el archivo Program.cs *(ojo!, en una aplicación real dividiremos las clases en archivos independientes, incluso potencialmente en un proyecto independiente; ahora por centrarnos en el tema lo hacemos en un mismo archivo)*

```
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; }
}
```

Observa cómo las dos propiedades de navegación (Blog.Posts y Post.Blog) se han definido como ***virtuales***, lo que habilita la característica de "carga diferida" de Entity Framework.

*Nota: Entity Framework admite tres maneras de cargar datos relacionados: carga diligente, carga diferida y carga explícita. **La carga diferida significa que el contenido de estas propiedades se cargará automáticamente desde la base de datos al intentar acceder a ellas.** Es decir, cuando se usa la clase de entidad Blog arriba definida, las entradas relacionadas (los posts) se cargarán la primera vez que se tenga acceso a la propiedad de navegación Posts.*

Paso 2 - Crear el contexto

Definir (a continuación de la clase Post) un contexto que deriva de System.Data.Entity.DbContext y expone un DbSet con tipo para cada clase de nuestro modelo:

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

Previamente necesitas:

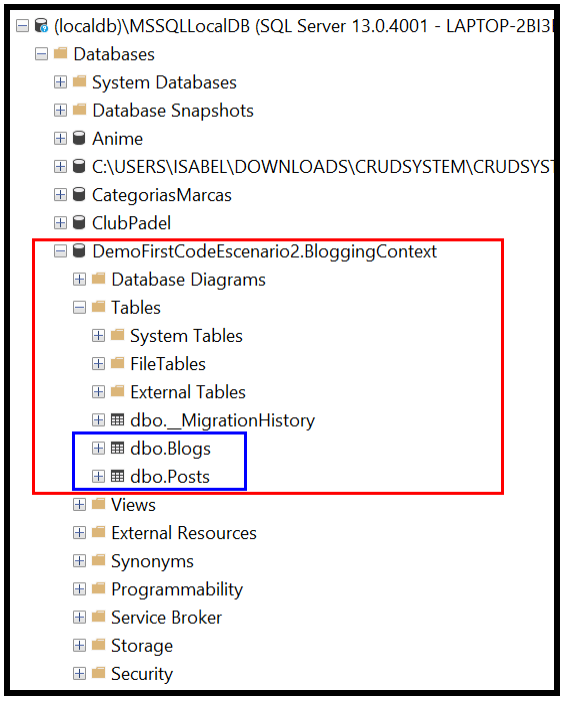
1. Desde el Administrador de paquetes NuGet agregar paquete EntityFramework paquetes.
2. importar el espacio de nombres using System.Data.Entity

Uso del modelo creado - Ejemplo

El siguiente código crea una nueva instancia de nuestro contexto y la usa para insertar un nuevo blog. A continuación, usa una consulta LINQ para recuperar todos los blogs de la base de datos ordenados alfabéticamente por Título.

```
class Program
{
   static void Main(string[] args)
   {
      using (var db = new BloggingContext())
      {
         // Insertar un nuevo 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();


         // Mostrar todos los Blogs
         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("Pulsa una tecla para finalizar...");
         Console.ReadKey();
      }
   }
}
```

Si ejecutas este código y consultas tu servidor de BD puedes observar que se ha creado la Base de datos acorde al modelo definido en C#



En resumen: se ha definido un modelo mediante clases y, a continuación, se ha utilizado ese modelo para crear una base de datos y almacenar y recuperar datos.

Una vez creada la base de datos, *Migraciones* de Code First permite modificar el esquema a medida que evoluciona el modelo *(este apartado queda fuera del objetivo del tema; en los enlaces propuestos se detalla el procedimiento para hacerlo, así como lo que debes saber para realizar una configuración adicional del modelo mediante anotaciones de datos y la API Fluent).*

fuente: *(Code First a una nueva base de datos)*