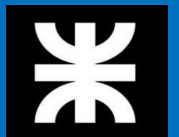


# Unidad 4

Acceso a Datos

## Capitulo 2

ORM Entity Framework



# Contenidos

- Introducción
- Componentes
  - DbContext y DbSet<T>
- Entity State y Change Tracking
- Mapeo
  - Data Annotations
  - Fluent API
  - Conventions
- Insertar, Consultar, Eliminar y Actualizar
- Otras funcionalidades
  - Migrations
  - Lazy, Eager y Explicit Loading

## Modelo de Objetos <> Modelo Relacional

### Modelo de Objetos

Objects

Behavior

Properties

Inheritance

### Modelo Relacional

Tables

Views

Stored Procedures

Foreign Key Relationships

Construir el puente entre estos dos esquemas utilizando DataSets y/o DataReaders genera un costo elevado en terminos de esfuerzo de desarrollo.

# Introducción

- Cada aplicación trabaja en parte con Objetos y en parte con Datos Relacionales.
- Se necesitan conocer 2 lenguajes totalmente diferentes (C# y SQL)
  - Diferentes sintaxis
  - Diferentes tipos de datos
  - Diferentes herramientas de trabajo
  - Diferentes paradigma: Objetos vs Datos Relacionales
- Además se necesita aprender la API que une estos dos mundos: ADO.NET
  - Potente, pero frágil y consumidor importante de tiempo.

# Introducción

- Entity Framework aparecio con el Service Pack 1 de la version 3.5 de .NET
- *System.Data.Entity.dll*
- En lugar de operar sobre filas y columnas se opera sobre objetos denominados *entities*.
- Estas entidades tienen soporte para LINQ
- Soporta dos formas de modelar las entidades:
  - Usando Code First
  - Usando el EF Designer
- Solo vamos a cubrir Code First, el Designer presenta varios problemas y ya no esta incluido en .NET Core

# Un ejemplo rapido

- Agregar un Usuario

```
class Program
{
    static void Main(string[] args)
    {
        // Connection string automatically read from config file.
        using (AcademiaEntities context = new AcademiaEntities())
        {
            // Add a new record to Users table, using our model.
            context.Usuarios.Add(new Usuario() { NombreUsuario = "Admin",
                                                Clave = "Password",
                                                Habilitado = true });

            context.SaveChanges();
        }
    }
}
```

# Componentes - DbContext

Representa una combinación de los patrones Unit of Work y Repository los cuales puede ser usados para consultar datos y acumular cambios sobre los objetos que luego puedan ser guardados en un solo Unit of Work.

Miembro	Descripción
DbContext	Constructor. Toma de parametro el Connection String a la base de datos.
Entry Entry<TEntity>	Obtiene el objeto System.Data.Entity.Infrastructure.DbEntityEntry permitiendo acceso a información sobre el mismo y tambien a la posibilidad de realizar acciones sobre la entidad.
SaveChanges SaveChangesAsync	Graba los cambios realizados en el context actual a la base de datos y devuelve el número de entidades afectadas.
Database	Acceso a la Base de Datos. Creación, Eliminación, Validación de Existencia. Ejecución de Stored Procedures, sentencias SQL y funcionalidades transaccionales.

Mas info sobre DbContext en EF Core

<https://learn.microsoft.com/es-es/dotnet/api/microsoft.entityframeworkcore.dbcontext>

# Ejemplo DbContext

- AcademiaEntities derivada de DbContext

```
public class AcademiaEntities : DbContext
{
    public AcademiaEntities() : base("name=AcademiaConnection")
    {
    }
}
```

# Componentes – DbSet<T>

Para agregar tablas al context se usa el objeto DbSet<T> para cada tabla.

Miembro	Descripción
Add AddRange	Permite ingresar un nuevo objeto a la colección. El objeto queda marcado como Added y debido a esto va a ser insertado en la Base de Datos cuando se llame al SaveChanges.
Attach	Asocia un objeto al DbContext. Este es un uso muy comun para entornos desconectados como ASP.NET/MVC.
Find FindAsync	Encuentra una fila buscando por la primary key y devuelve un objeto representando esa fila.
Remove RemoveRange	Marca un objeto (o rango de objetos) para eliminación.
FromSql	Escribir un SQL puro que devuelva entidades de este set.

Mas info sobre DbSet<T> en EF Core

<https://learn.microsoft.com/es-es/dotnet/api/microsoft.entityframeworkcore.dbset-1>

# Entity State y Change Tracking

Posibles valores que puede tomar el estado de una entidad.

Value	Descripción
Detached	El objeto existe pero su estado no esta siendo trackeado. Una entidad queda en este estado apenas es creado y antes de ser agregado al context.
Unchanged	El objeto no fue modificado desde que fue attachado al context o desde la ultima vez que se llamo al SaveChanges()
Added	El objeto es nuevo y fue agregado al context pero el metodo SaveChanges() todavia no fue llamado.
Deleted	El objeto fue eliminado del context pero todavia no fue eliminado de la base de datos.
Modified	Una de las propiedades escalares del objeto fue modificada y el metodo SaveChanges() todavia no fue llamado.

```
EntityState state = context.Entry(entity).State;
```

# Data Annotations

Se utilizan para configurar las entidades y mapear tablas y columnas. Estas son algunas de las mas comunes:

Data Annotation	Descripción
Key	Define la primary key del modelo. No necesariamente tiene que ser un key de una sola columna, soporta claves multiples.
Required	La propiedad no permite nulos.
ForeignKey	La propiedad se usa como ForeignKey para navegación.
StringLength	Minimo y maximo permitidos para la propiedad.
NotMapped	Propiedad que no tiene relación con la Base de Datos.
ConcurrencyCheck	Marca una propiedad para ser utilizada en controles de concurrencia para updates, inserts o deletes.
Table Column	Permiten mapear las propiedades con columnas de la base de datos que tengan nombres distintos a las propiedades.
DatabaseGenerated	Especifica si la propiedad esta generada por la base de datos, por ejemplo un Identity.
Index	Especifica que una columna tienen que tener un indice creado.

# Fluent API - Ejemplo

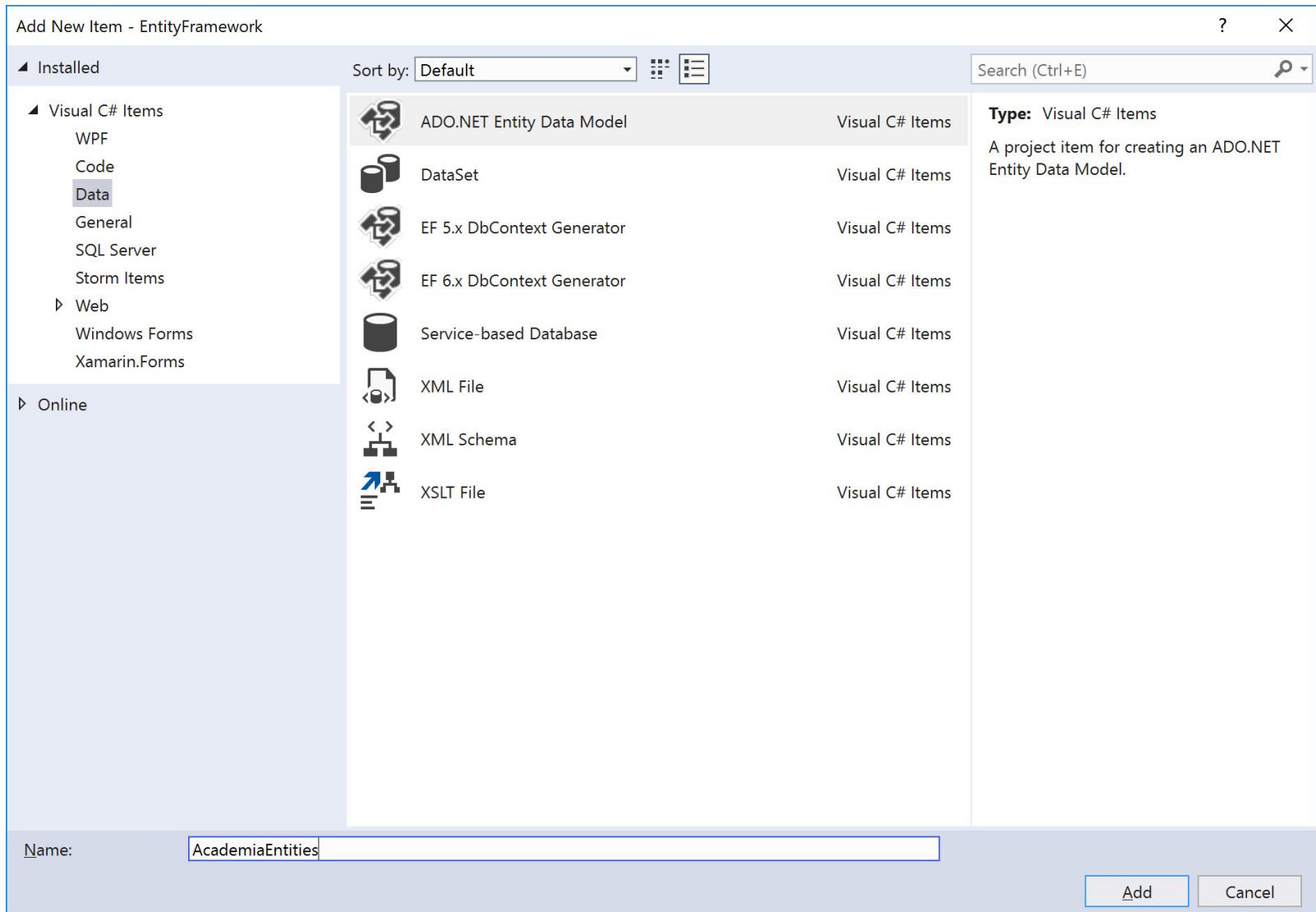
```
using Microsoft.EntityFrameworkCore;
namespace EFModeling.EntityProperties.FluentAPI.Required;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    #region Required
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
    #endregion
}


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


# Generando un EDM



# Generando un EDM


Entity Data Model Wizard








**Choose Model Contents**

**What should the model contain?**

  
EF Designer  
from  
database

  
Empty EF  
Designer  
model

  
Empty Code  
First model

  
Code First  
from  
database

Creates a Code First model based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model.

< Previous

Next >

Finish

Cancel

# Generando un EDM

- El Wizard genera las Entities, DbSet y configuraciones.
- El Wizard usa Data Annotations pero todos los mapeos tambien se podrian hacer con Fluent API.
- Todo lo que se genero se podria haber hecho sin Wizard de manera sencilla y transparente.

- Insertar un Usuario

```
using (var context = new AcademiaEntities())
{
    var usuario = new Usuario()
    {
        NombreUsuario = "Admin",
        Clave = "Password",
        Habilitado = true
    };

    context.Usuarios.Add(usuario);
    context.SaveChanges();
}
```

- Consultar Usuarios

```
using (var context = new AcademiaEntities())
{
    foreach (var usuario in context.Usuarios)
    {
        System.Console.WriteLine(usuario.NombreUsuario);
    }
}
```

- Consultar Usuarios que incluyen la letra “r”

```
using (var context = new AcademiaEntities())
{
    foreach (var usuario in context.Usuarios.Where(u=> u.NombreUsuario.Contains("r")))
    {
        System.Console.WriteLine(usuario.NombreUsuario);
    }
}
```

- Eliminar un Usuario

```
using (var context = new AcademiaEntities())
{
    Usuario usuarioAEliminar = context.Usuarios.Find(idUsuario);

    if (usuarioAEliminar != null)
    {
        context.Usuarios.Remove(usuarioAEliminar);
        context.SaveChanges();
    }
}
```

- Actualizar un Usuario

```
using (var context = new AcademiaEntities())
{
    Usuario usuarioAAActualizar = context.Usuarios.Find(idUsuario);

    if (usuarioAAActualizar != null)
    {
        usuarioAAActualizar.clave = "claveModificada";

        context.SaveChanges();
    }
}
```

# Otras funcionalidades

- Creación y actualización de Base de Datos con Migrations
  - <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/>
- Lazy, Eager y Explicit Loading
- Mas info sobre EF Core:  
<https://learn.microsoft.com/en-us/ef/core/>