# Desarrollador de Aplicaciones Web
## Programación Web III


**Departamento de Ingeniería e Investigaciones Tecnológicas**

# Entity Framework

**Ing. Matias Paz Wasiuchnik**
**Ing. Pablo Nicolás Sanchez**
**Ing. Mariano Juiz**

# Agenda

- Introducción (Ado.Net)
- Arquitectura (EF6)
- Enfoques.
- Creación de un EDM (EF6)
- Ciclo de vida.
- EF Core
- Operaciones CRUD con una entidad.

.

# Introducción: ADO.NET

**ADO.NET** proporciona acceso coherente a orígenes de datos como SQL Server y XML, así como a orígenes de datos expuestos mediante OLE DB y ODBC. Las aplicaciones de consumidor que comparten datos pueden utilizar ADO.NET para conectar a estos orígenes de datos y recuperar, controlar y actualizar los datos contenidos.

**ADO.NET** separa el acceso a datos de la manipulación de datos y crea componentes discretos que se pueden utilizar por separado o conjuntamente. ADO.NET incluye proveedores de datos .NET Framework para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Los resultados se procesan directamente o se colocan en un objeto DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, combinados con datos de varios orígenes, o de pasarlos entre niveles. El objeto `DataSet` de ADO.NET también puede utilizarse independientemente de un proveedor de datos .NET Framework para administrar datos que son locales de la aplicación o que proceden de un origen XML.

- Proveedores de datos ADO.NET:
  - SqlClient `System.Data.SqlClient( )`
  - OleDb `System.Data.OleDb( )`
  - Odbc `System.Data.Odbc( )`
  - OracleClient `System.Data.OracleClient( )`

https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/ado-net-code-examples

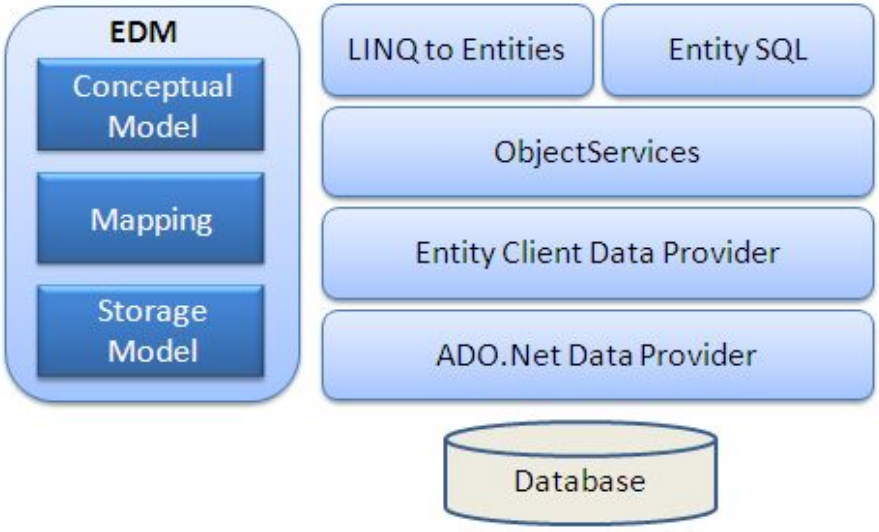
# Introducción

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia.

Un ORM aporta ayuda en la resolución del denominado **desajuste de impedancia** (o impedance mismatch en inglés).
El desajuste de impedancia refiere a las **diferencias existentes** entre los **modelos relacionales** (base de datos) y los **modelos conceptuales** implementados en lenguajes de programación orientados a objetos.

ADO.NET Entity Framework es un ORM que permite a los desarrolladores crear aplicaciones de acceso a datos programando con un modelo de aplicaciones conceptuales en lugar de programar directamente con un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y el mantenimiento necesarios para las aplicaciones orientadas a datos.

# Arquitectura ADO.NET EF




**EDM:** EDM (Entity Data Model) es la definición del mapeo que creamos entre la base de datos y nuestro modelo conceptual o de entidades.

En EF 6, Un modelo se guarda en un fichero con extensión *edmx,* que a través del lenguaje XML declara 3 secciones (que aunque puede estar en ficheros distintos, la norma es que estén en un solo fichero):

# Arquitectura (Continuación)

## LINQ To Entities

Es un lenguaje de consulta usado para escribir consultas contra el modelo de objetos. Devuelve las entidades definidas en el modelo conceptual

## Entity SQL

Es un lenguaje de consulta como LINQ tambien, es decir otra forma de recuperar objetos desde el modelo conceptual. Las consultas deben estar contenidas dentro de una variable string.

## Object Service

Es el responsable de llevar a cabo la **Materialización**, la cual consiste en el proceso de convertir los datos resultantes de la base de datos devueltos por "Entity client data provider" (proxima capa) en una estructura objeto entidad

## Entity Client Data Provider:

Su responsabilidad es convertir las consultas L2E o Entity SQL y sentencias en una consulta o sentencia SQL de base de datos. Se comunica directamente ADO.Net, el cual dialoga con una base de datos.

# Enfoques en Entity Framework

## Existen 3 formas o enfoques distintos para utilizar EF:

- **Data Base First (Este será el enfoque que seguiremos)**

  A partir de un modelo relacional (Base de Datos) genero el EDM. Se trabaja directamente con **DbContext**

  **EF 6: Con wizard y modelo visual EDMX.**

  **EF CORE: Soportado pero sin modelo visual EDMX.**

- **Model First (Solo EF 6)**

  A partir del modelo conceptual (EDMX visual) se genera el modelo relacional. Se trabaja directamente con **DbContext**.

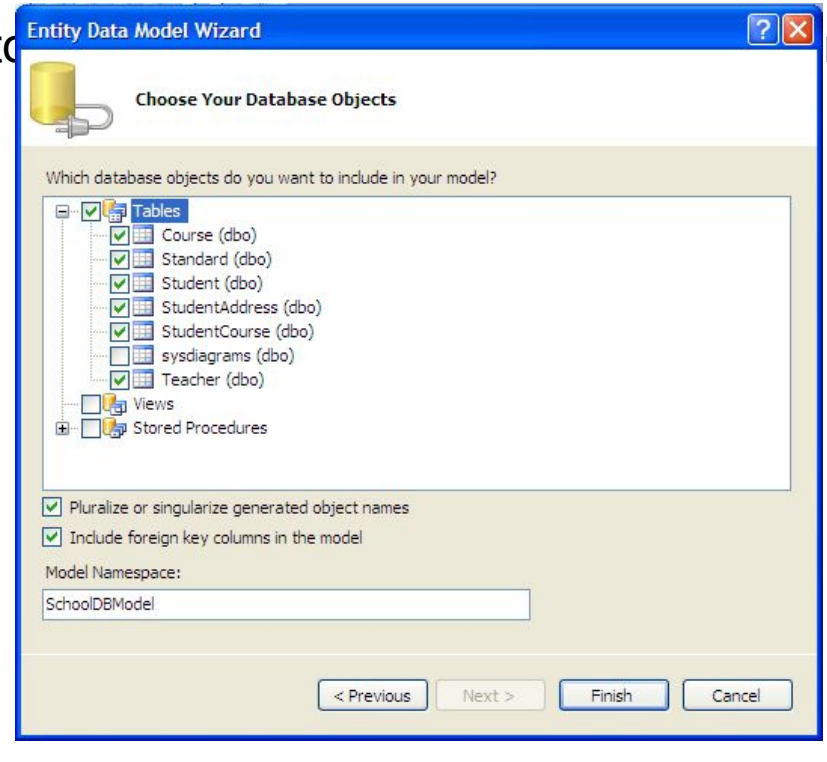
- **Code First (EF 6 y EF CORE):**

  Se parte de un modelo puro de entidades en programación orientada a objetos (clases POCO) y a partir de ahí se genera el modelo de base de datos relacional. Incorpora dos nuevos tipos: **DbContext**

# Enfoque DataBase First (EF 6)

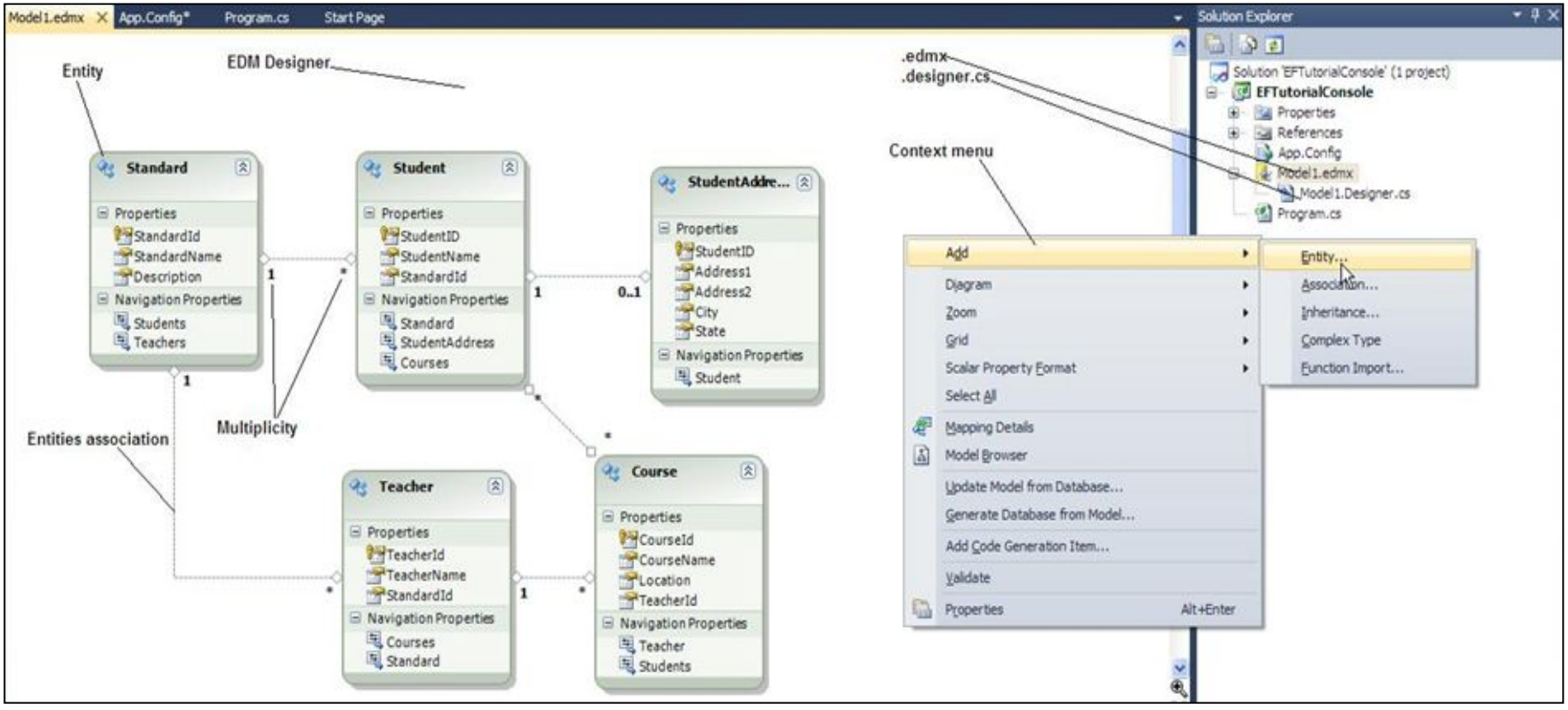
## Pasos para crear el EDMX:

1. Crear un proyecto Sitio Web o Aplicación Web o Consola
2. Botón derecho sobre el proyecto, elegir “Agregar Nuevo Item” y escoger el template ADO.NET Entity Data Model
3. En el Data Model wizard, elegir la opción **Generar desde la Base de Datos.**
4. Elegir una conexión de base de datos.
5. Elegir los objet mapear



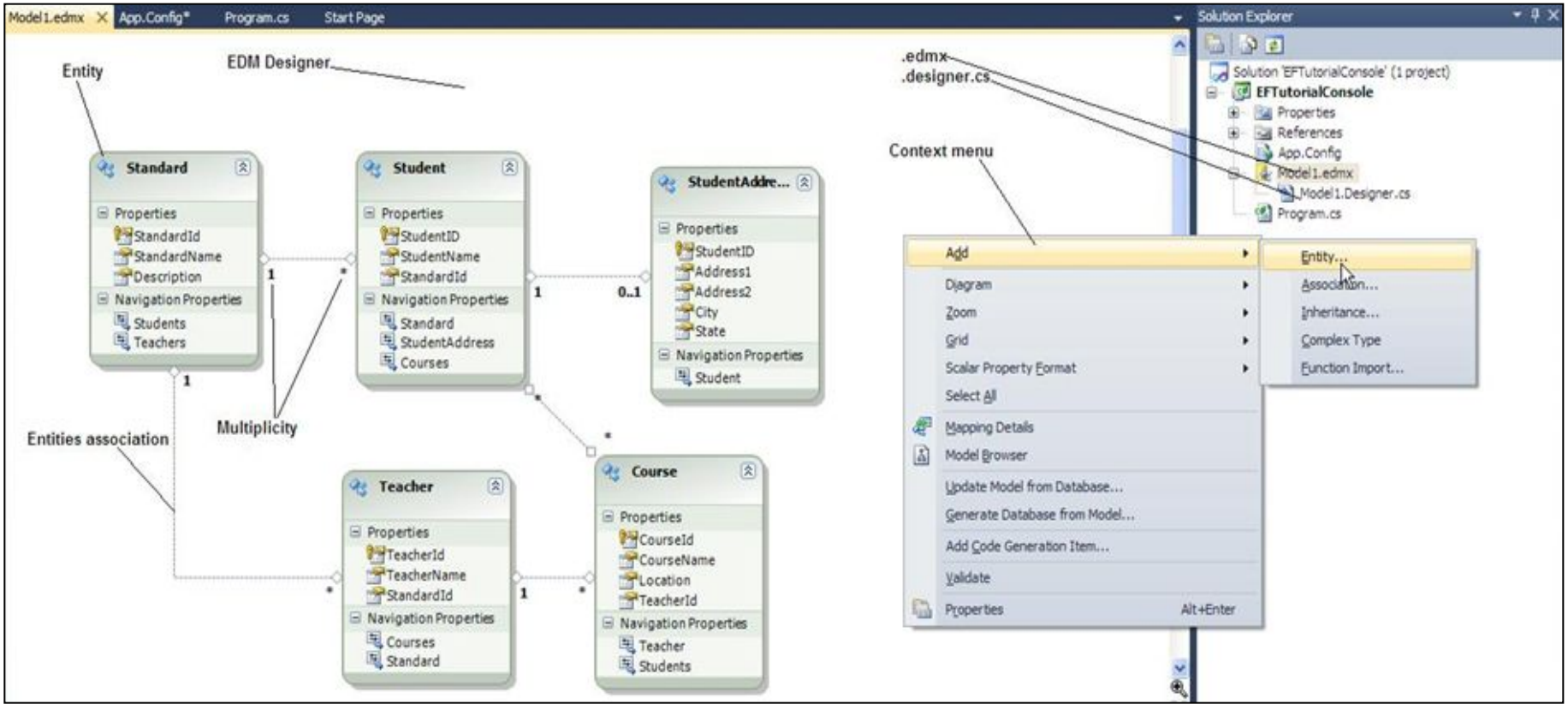
# Enfoque DataBase First (EF 6)

6. Finalizado 5, se genera el EDMX:



# Enfoque DataBase First (EF 6)

6. Finalizado 5, se genera el EDMX:



# Enfoque Code First (EF 6 o CORE)

1. Crear la clase para el Modelo

```
namespace EFEjemplo.Modelo {
   public class Persona
   {
      public int Id { get; set; }
      public string Apellido { get; set; }
      public string Nombre { get; set; }
   }
}
```

2. Crear la clase para Administrar la relación entre la Base de Datos y el DBContext

```
using System.Data.Entity;
namespace EFEjemplo.Modelo
{
public class MiContexto : DbContext
   {
         public MiContexto() : base("name=CONEXION_WEBCONFIG") {}
         public DbSet<Persona> Personas{ get; set; }
   }
}
```

# Enfoque Code First (EF 6 o CORE)

## 3. Crear el controlador para llamar a la clase administradora

```
using System.Linq;
using System.Web.Mvc;
using EFEjemplo.Modelo;
namespace EFEjemplo.Controladores
{
    public class PersonaController : Controller
    {
        MiContexto contexto;
        public PersonaController()
        {
            contexto = new MiContexto();
        }
        [HttpPost]
        public ActionResult Create(Persona persona)
        {
            contexto.Personas.Add(persona);
            contexto.SaveChanges();
            return RedirectToAction("Index");
        }
}
```

# Ciclo de vida

En EF cada entidad tiene un estado (EntityState) que irá cambiando dependiendo de la operación que realice el contexto (DbContext).

El estado de la entidad es un tipo enum (System.Data.EntityState), que declara los siguientes valores:

- **Added:** Entidad agregada al contexto (Add).

- **Deleted:** Indica que la entidad está eliminada del contexto (Delete)

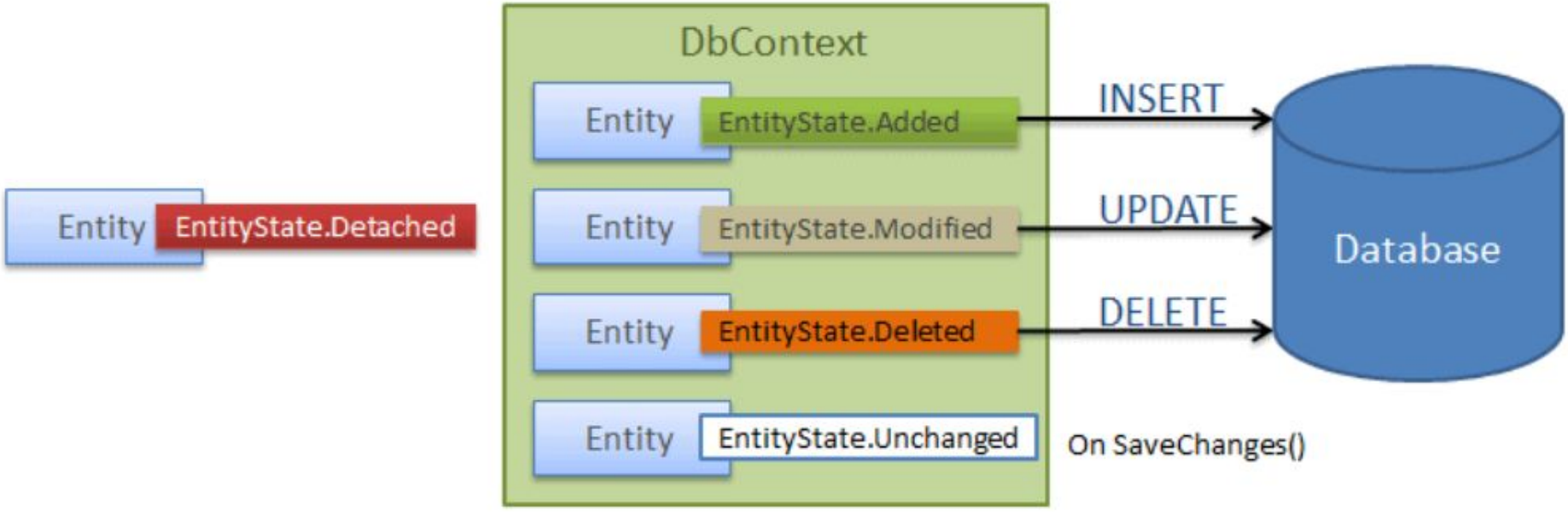
- **Modified:** Cada vez que se modifica el valor de alguna propiedad de una entidad recuperada del contexto.

- **Unchanged**: Indica que la entidad original recuperada desde el contexto no ha sido modificada (escenario conectado)

- **Detached:** Indica que la entidad no esta ligada o no pertenece al contexto (Detach)

El contexto no solamente referencia a todos los objetos recuperados desde la base de datos, sino que mantiene además todo las modificaciones vinculadas a las propiedades de cada entidad. Esta característica es conocida como **Change Tracking.**

# Ciclo de vida




EF compila y ejecuta los comandos INSERT, UPDATE y DELETE según el estado de una entidad cuando se llama al método **context.SaveChanges().**

Ejecuta el comando INSERT para las entidades con estado **Added**, el comando UPDATE para las entidades con estado **Modified** y el comando DELETE para las entidades en estado **Deleted**.

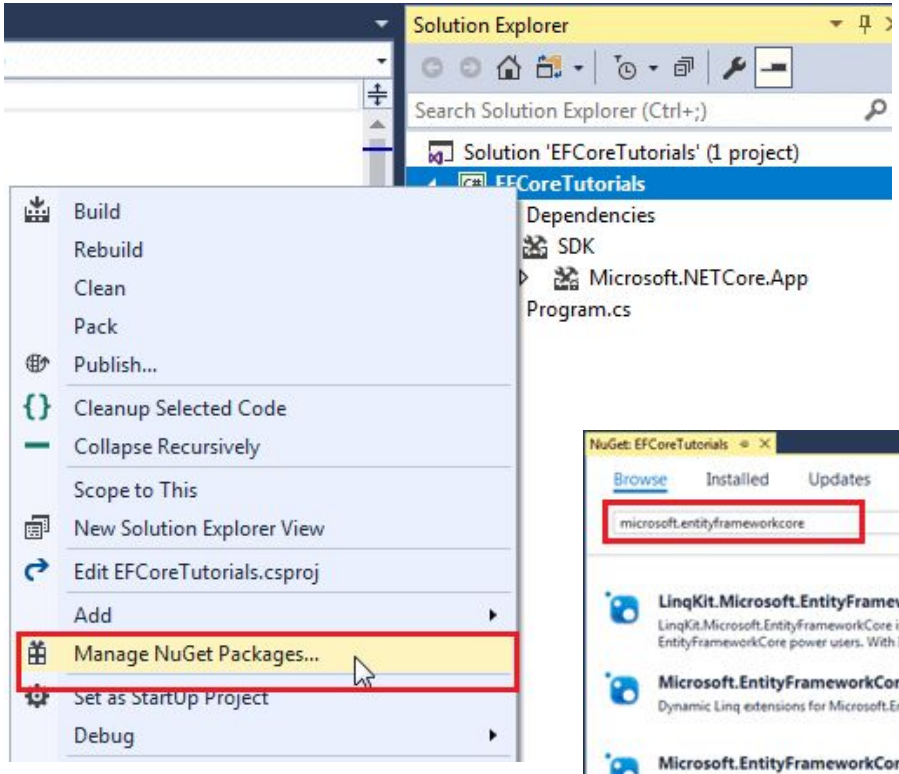
El contexto no trackea entidades en estado "Detached".

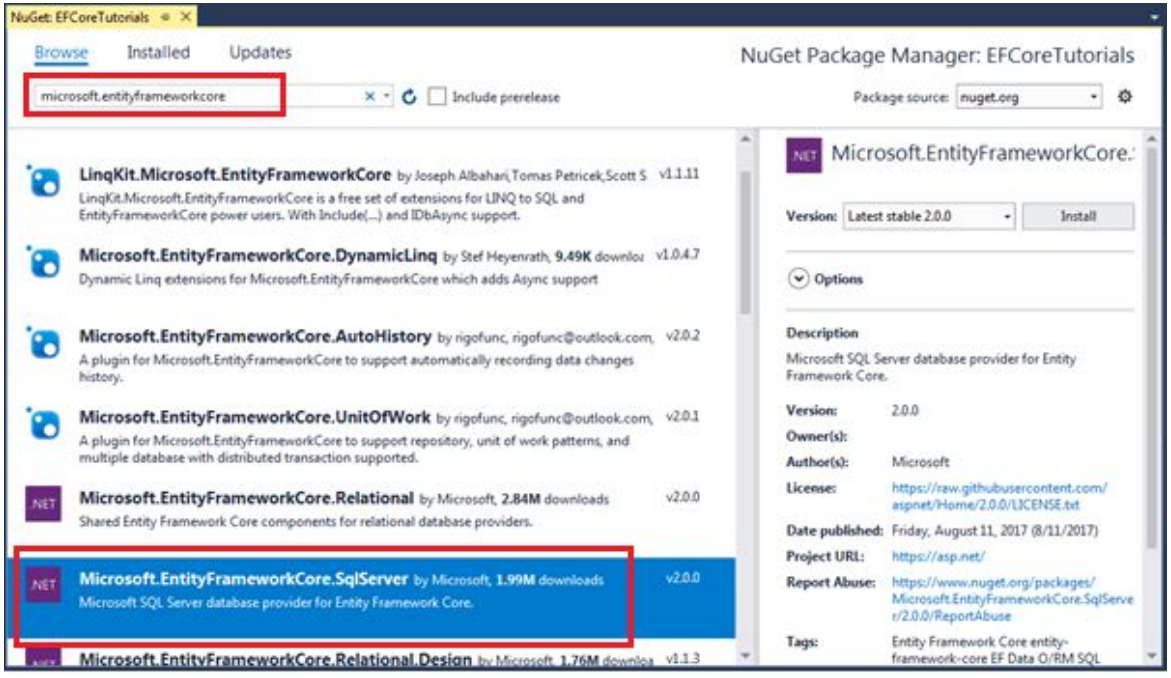
# Prerequisitos EF CORE

- .NET Core 3.1 o 5 - Net 5
- Visual Studio 2019 (v16.8 o +)
- EF Core DB provider
- EF Core tools
- SQL Server Express or SQL Server 2008 o superior

# EF CORE (EF CORE DB Provider)

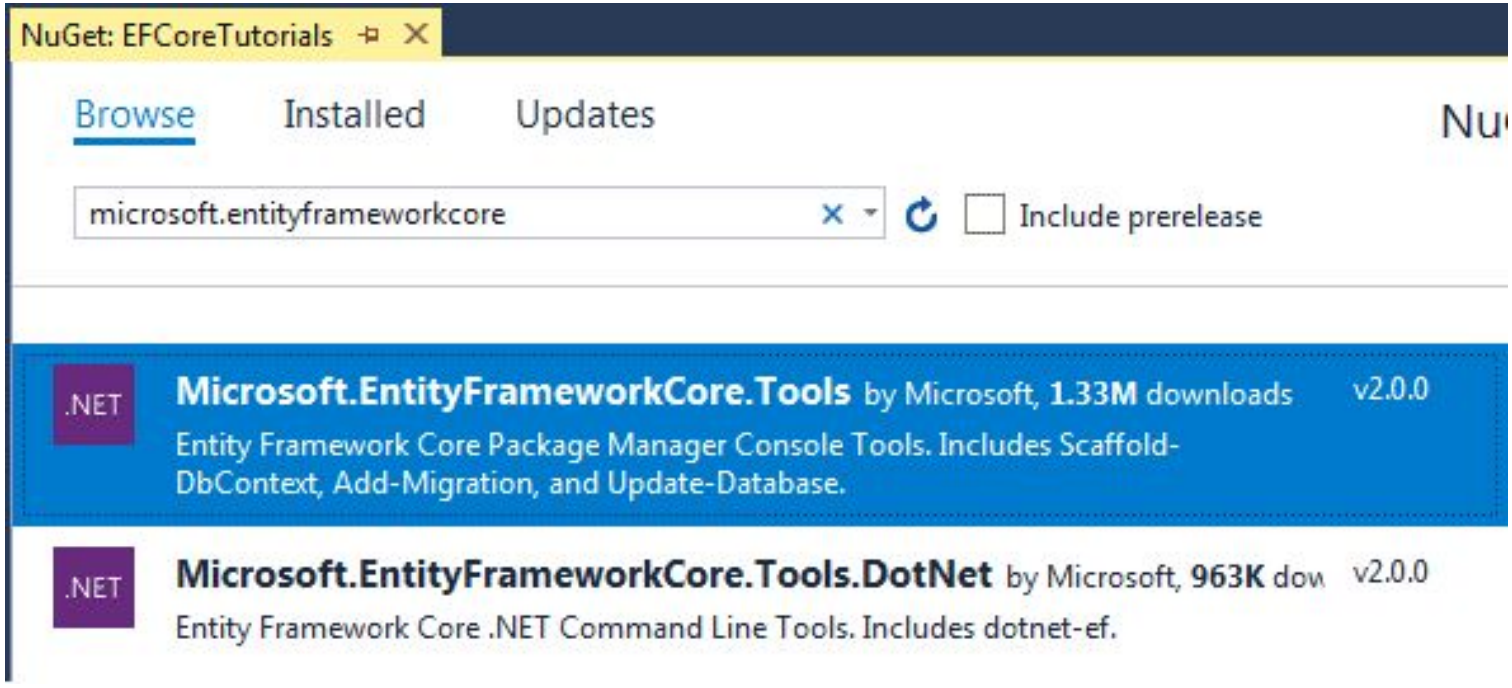
**Instalar el provider de entity framework CORE para SQLServer:**



# Prerequisitos EF CORE (EF CORE TOOLS)

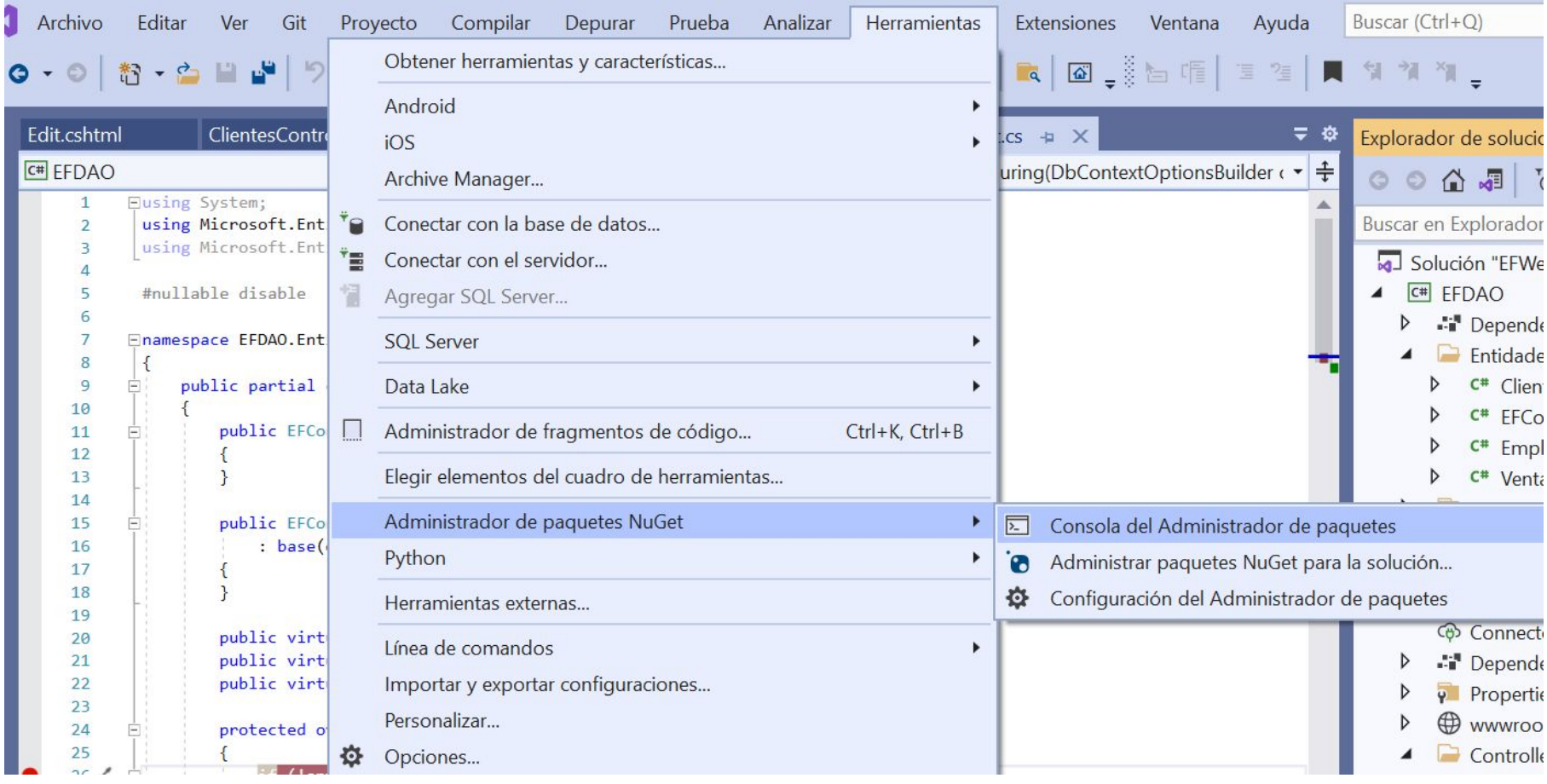
**Instalar e entity framework Core Tools**




**Esto nos permitirá ejecutar: Scaffold-DbContext**

# Prerequisitos EF CORE DataBaseFirst

## 1) En Visual Studio, Herramientas-> Administrador de Paquetes Nuget -> Consola de Administración de Paquetes:



# Prerequisitos EF CORE DataBaseFirst

## 2) Ejecutar: Scaffold-DbContext Command:

**PM> Scaffold-DbContext "Server=(localdb)\MSSQLLocalDB;Database=NombreBaseDatos;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir CarpetaDestino**

### Scaffold-DbContext Command:

scaffold-DbContext [-Connection] [-Provider] [-OutputDir] [-Context] [-Schemas>] [-Tables>]  [-DataAnnotations] [-Force] [-Project] [-StartupProject] [<CommonParameters>]

# EF CORE 5 DataBase First

## 3) Agregar la inyección de dependencia del Contexto en Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<EFCoreContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("EFCoreContext")));
}
```

**EFCoreContext: Es el nombre del contexto (dbContext) de EF.**

## NOTA: Ubicar el connection string en appsettings.json:

```
{
  "ConnectionStrings": {
    "EFCoreContext":
"Server=(localdb)\\MSSQLLocalDB;Database=EFCore;Trusted_Connection=True;"
  }
}
```

# Operaciones CRUD (SaveChanges)

## Crear nuevos elementos (Create)

```
protected void Agregar()
{
        EFPW3 ctx = new EFPW3();

        //1. Nuevo Objeto

        Cliente cli = new Cliente();
        cli.Nombre = "Chevrolet S.A";
        ctx.Clientes.Add(cli);

        //2. Nuevo Objeto

        cli = new Cliente();
        cli.Nombre = "Radio Planeta";
        ctx.Clientes.Add(cli);

        ctx.SaveChanges(); //se persisten los datos en la BD, se
crearon dos clientes
  }
```

## Actualizar elementos (Update)

```
protected void Actualizar()
 {
        using ( var ctx = new EFPW3())
        {
           Empleado emp = (from em in ctx.Empleados
                    select em).First();

           emp.Nombre = "Modificado";
    //Cuando se ejecuta la línea anterior, el entityState pasa a “Modified”

            ctx.SaveChanges();
        }
}
```

## Eliminar elementos (Delete)

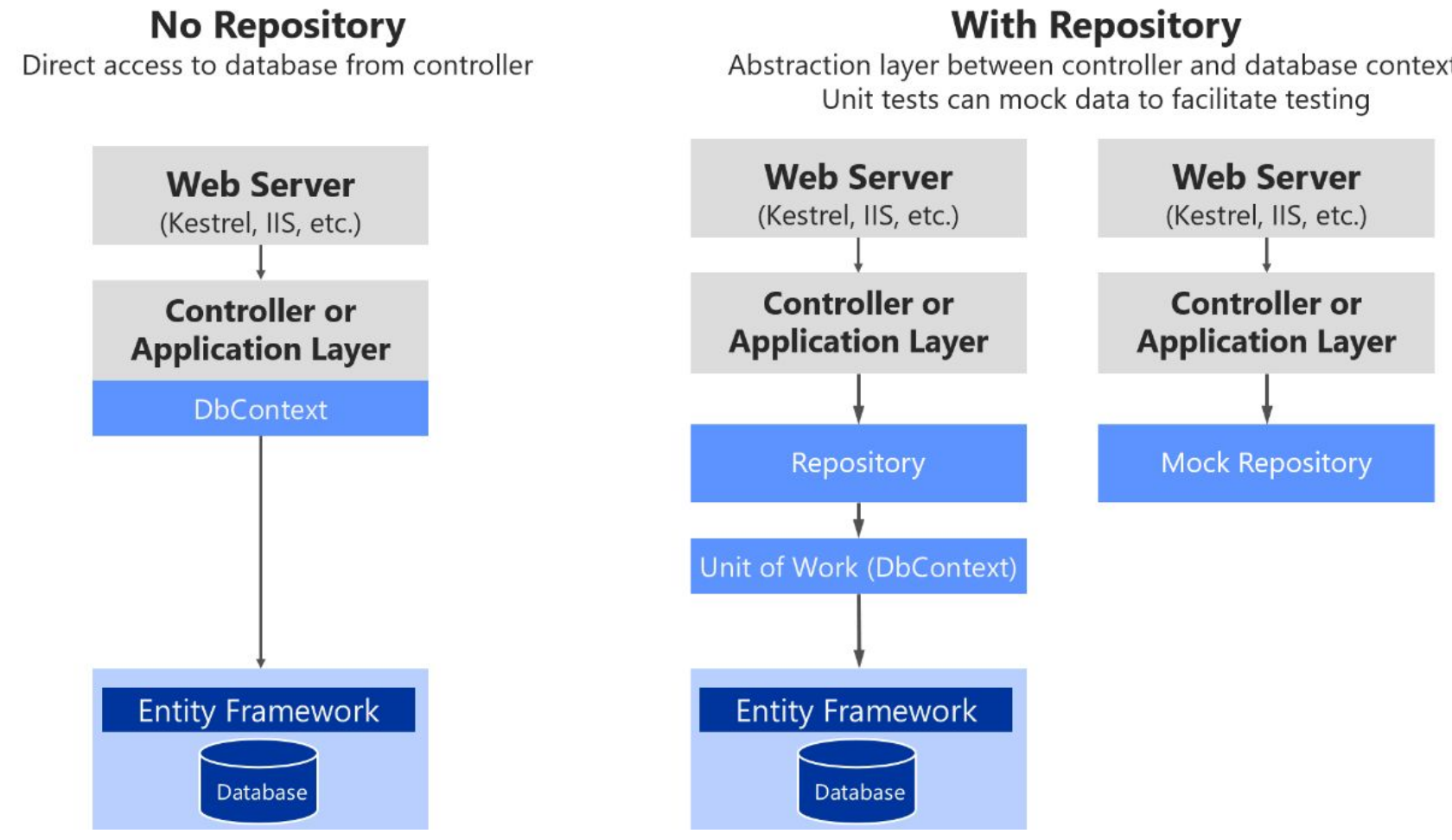
```
protected void Eliminar()
{
        EFPW3 ctx = new EFPW3();

        var cliente = context.Clientes.First();

        ctx.Clientes.Remove(cliente);

        ctx.SaveChanges();
}
```

# Prerequisitos EF CORE DataBaseFirst

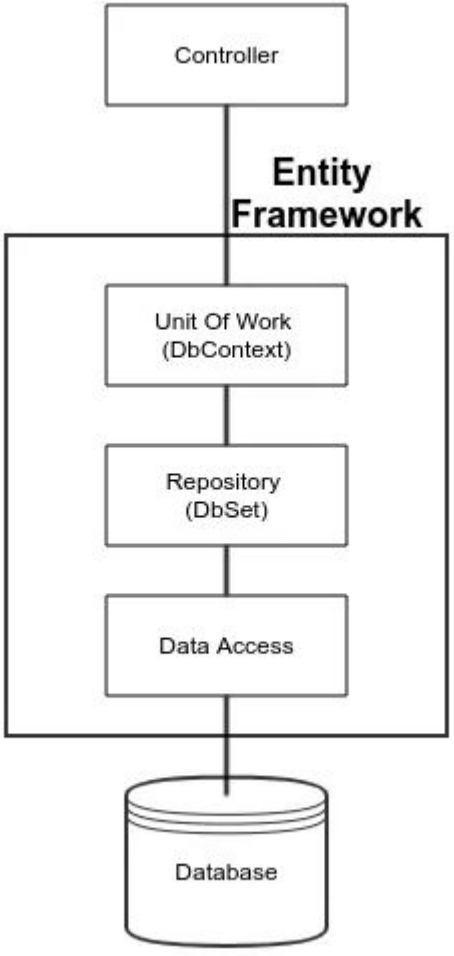



https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core

# Repository vs Unit of Work

“A Unit of Work keeps track of everything you do during a business transaction that can affect the database. When you’re done, it figures out everything that needs to be done to alter the database as a result of your work.”

Martin Fowler



# Desarrollador de Aplicaciones Web
## Programación Web III


**Departamento de Ingeniería e Investigaciones Tecnológicas**

# Muchas gracias

**Ing. Matias Paz Wasiuchnik**
**Ing. Pablo Nicolás Sanchez**
**Ing. Mariano Juiz**