



# **ACCESO A BASES DE DATOS CON ENTITY FRAMEWORK**

Universidad Tecnológica Nacional  
Facultad Regional Concepción del Uruguay

Ingeniería en Sistemas de Información  
Taller de Programación

# Problema

- “En la programación Orientada a Objetos, las tareas de gestión de datos son implementadas generalmente por la manipulación de objetos, los cuales son casi siempre valores no escalares. Para ilustrarlo, considere el ejemplo de una entrada en una libreta de direcciones, que representa a una sola persona con cero o más números telefónicos y cero o más direcciones. En una implementación orientada a objetos, esto puede ser modelado por un ‘objeto persona’ con ‘campos’ que almacenan los datos de dicha entrada: el nombre de la persona, una lista de números telefónicos y una lista de direcciones. La lista de números telefónicos estaría compuesta por ‘objetos de números telefónicos’ y así sucesivamente. La entrada de la libreta de direcciones es tratada como un valor único por el lenguaje de programación (puede ser referenciada por una sola variable, por ejemplo). Se pueden asociar varios métodos al objeto, como uno que devuelva el número telefónico preferido, la dirección de su casa, entre otros datos.”

# Problema

- “Sin embargo, muchos productos populares de Base de Datos, como los Sistemas de Gestión de Bases de Datos SQL, solamente pueden almacenar y manipular valores escalares como enteros y cadenas, organizados en tablas normalizadas. El programador debe convertir los valores de los objetos en grupos de valores simples para almacenarlos en la Base de Datos (y volverlos a convertir luego de recuperarlos de la Base de Datos), o usar sólo valores escalares simples en el programa. El mapeo Objeto-Relacional es utilizado para implementar la primera aproximación.”
- “El núcleo del problema reside en traducir estos objetos a formas que puedan ser almacenadas en la Base de Datos para recuperarlas fácilmente, mientras se preservan las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes.”

*Fuente: Wikipedia ([https://es.wikipedia.org/wiki/Mapeo\\_objeto-relacional](https://es.wikipedia.org/wiki/Mapeo_objeto-relacional))*

# Problemas de los sistemas tradicionales

- Los modelos de datos convencionales, especialmente el relacional, son muy simples como para modelar entidades compuestas.
- Los sistemas convencionales sólo soportan un conjunto limitado de tipos.
- Los modelos de datos convencionales no incluyen conceptos semánticos útiles como generalización, agregación o composición.

# ¿Cómo solucionar las deficiencias?

1. Extender el modelo relacional.
2. Bases de Datos Orientadas a Objetos.
3. Mapeo de Objetos a Bases de Datos Relacionales.



# Puntos a favor de las alternativas 1 y 3

- Existe una base matemática para el lenguaje de consulta (álgebra relacional y cálculo de tuplas).
- Mucha experiencia adquirida.
- Gran base de clientes/usuarios instalada.
- Un estándar en la industria: SQL.

# Puntos a favor de la alternativa 2

- El modelo de datos Orientado a Objetos ya incluye conceptos como herencia, encapsulamiento, polimorfismo, agregación, generalización.
- Los lenguajes de “programación” O.O. pueden ser extendidos para obtener un lenguaje de “programación” y “BBDD” unificado, como consecuencia se obtiene homogeneidad.

# ¿Qué es un ORM?

- Un ORM es una herramienta para el almacenamiento de datos de objetos de dominio en una Base de Datos relacional como MS SQL Server, de forma automatizada , sin mucha programación.
- Un ORM incluye tres partes principales:
  - Objetos de clase de dominio.
  - Los objetos de Bases de Datos relacionales.
  - Información de mapeo sobre la forma de que objetos de dominio mapean con objetos de Bases de Datos relacionales (tablas, vistas y Stored Procedures) .
- Un ORM nos permite mantener nuestro diseño de Base de Datos separada de nuestro diseño de clases de dominio. Esto hace que la aplicación mantenible y extensible. También automatiza operaciones CRUD estándar (Crear, Leer, Actualizar y Eliminar) para que el desarrollador no tenga que escribirlo manualmente.

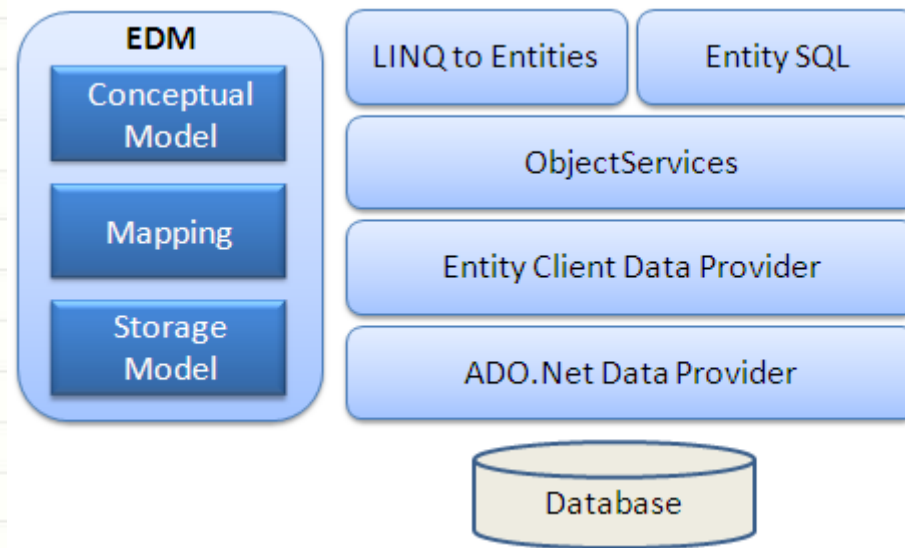




# Entity Framework

- Entity Framework (EF) es un ORM que permite a los desarrolladores de .NET trabajar con datos relacionales usando objetos específicos del dominio. Elimina la necesidad de la mayor parte del código de acceso a datos que los desarrolladores suelen tener que escribir.
- Entity Framework es útil en tres escenarios:
  - Si se tiene una Base de Datos existente o desea diseñar su Base de Datos antes de otras partes de la aplicación (DataBase First).
  - Si se quiere centrarse en sus clases de dominio y luego crear la Base de Datos a partir de las clases de dominio (Code First).
  - Si se desea diseñar el esquema de Base de Datos en el diseñador visual y luego crear la misma y las clases (Model First).

# Arquitectura de Entity Framework

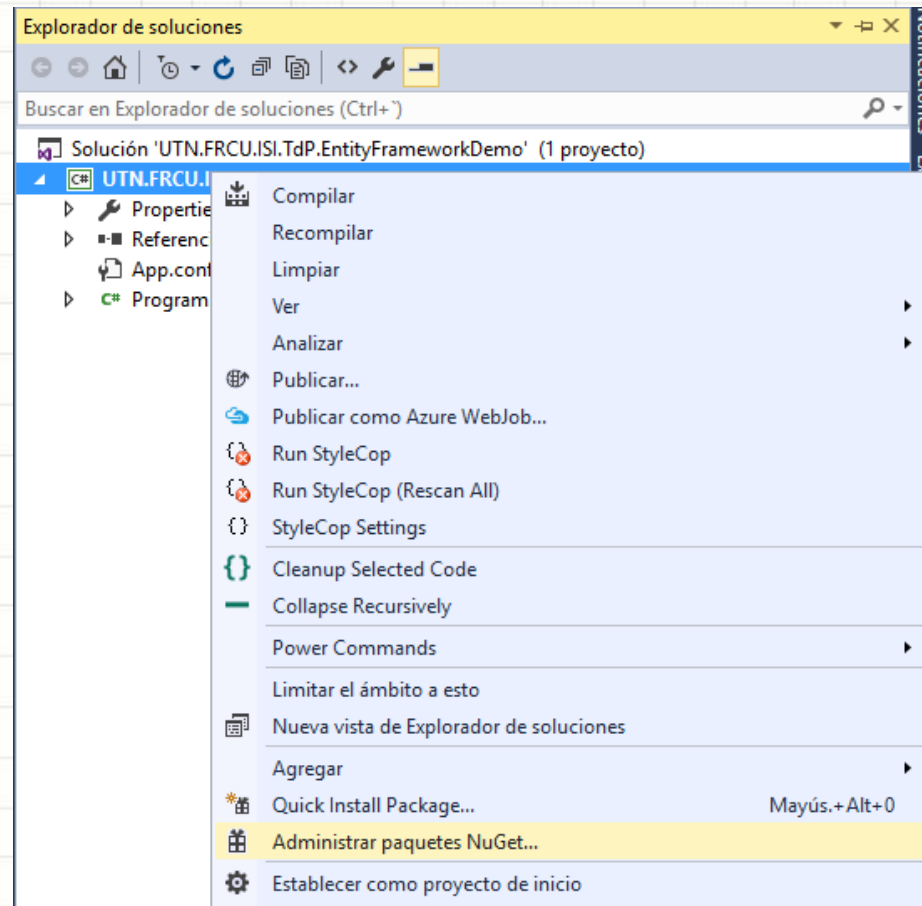


# Uso

- Para entender cabalmente de que estamos hablando, lo mejor es mostrar un ejemplo.
- Para ello necesitaremos:
  - Visual Studio Community.
  - Un gestor de BBDD (SQL Server Express o LocalDB)
  - Conexión a Internet.

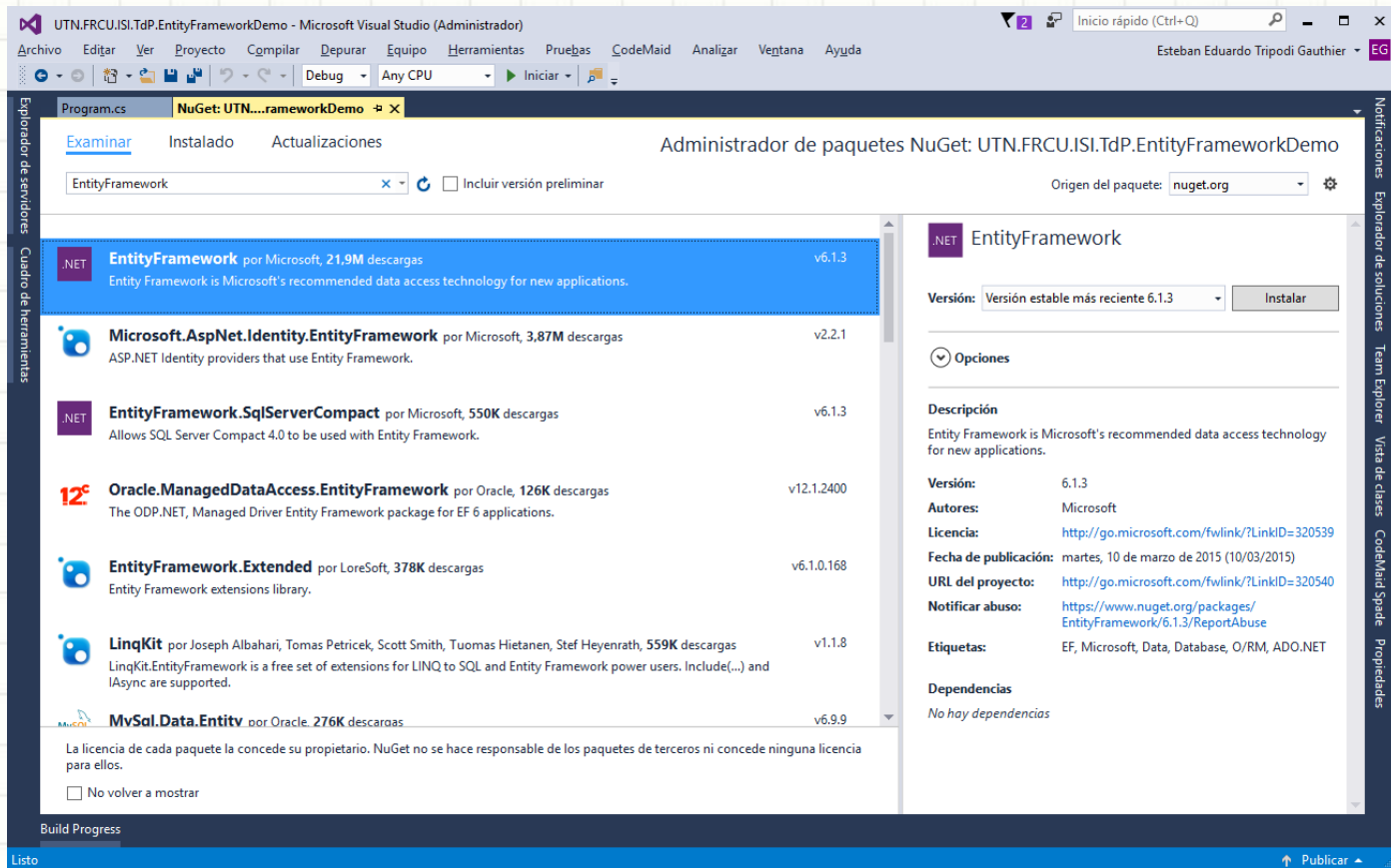
# Uso

- Buscamos Entity Framework en NuGet:



# Uso

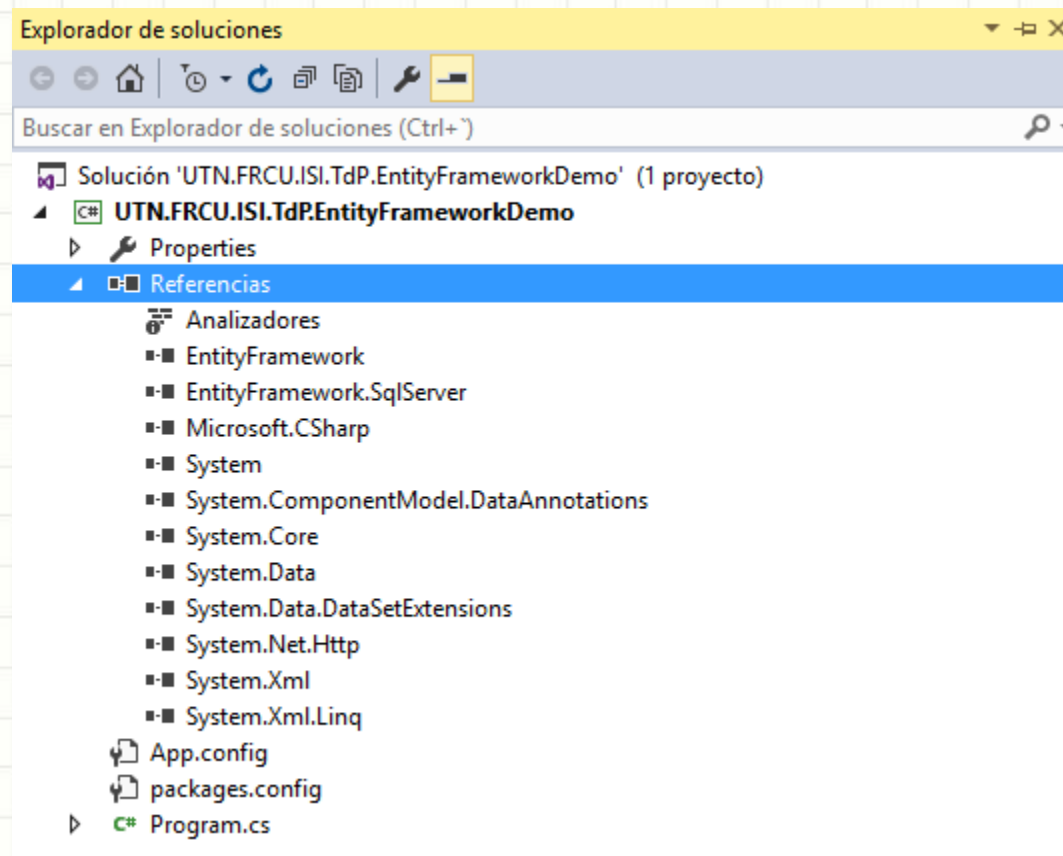
- Agregamos Entity Framework a nuestra solución:





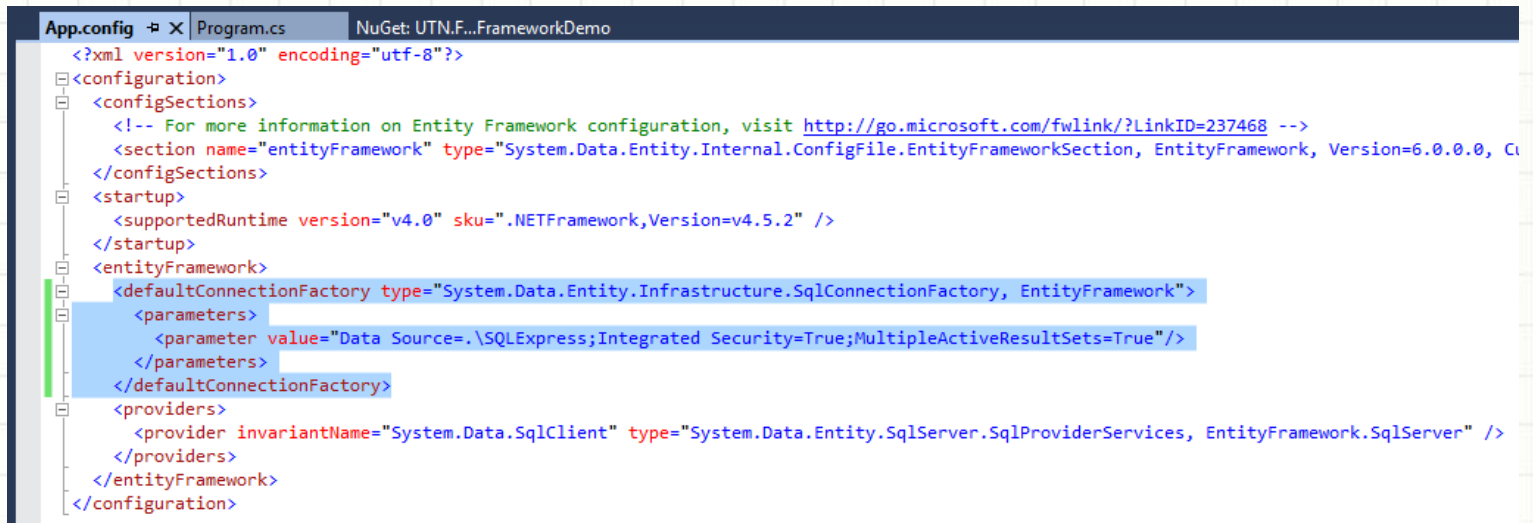
# Uso

- Las referencias agregadas pueden verse en el proyecto:



# Uso

- Se debe configurar la cadena de conexión en el archivo App.config:



The screenshot shows a code editor with the following XML configuration in the App.config file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, C
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="Data Source=.\SQLEXPRESS;Integrated Security=True;MultipleActiveResultSets=True"/>
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
</configuration>
```

# Uso

- Se deben agregar las clases de dominio:

```
using System;
using System.Collections.Generic;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class Persona
    {
        public int PersonaId { get; set; }

        public String Nombre { get; set; }

        public String Apellido { get; set; }

        public IList<Telefono> Telefonos { get; set; }
    }
}
```

```
using System;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class Telefono
    {
        public int TelefonoId { get; set; }

        public String Numero { get; set; }

        public String Tipo { get; set; }
    }
}
```

# Uso

- Ahora es el momento de definir un contexto que representa una sesión con la Base de Datos, lo que nos permite consultar y guardar los datos. Definimos una clase contexto que hereda de *System.Data.Entity.DbContext* y expone un *DbSet<TEntity>* con tipo para cada clase en nuestro modelo.

```
using System.Data.Entity;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class AgendaContext : DbContext
    {
        public DbSet<Persona> Personas { get; set; }
        public DbSet<Telefono> Telefonos { get; set; }
    }
}
```

# Uso

- Se genera el código de persistencia y acceso:

```
using System;
using System.Collections.Generic;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (AgendaContext bContexto = new AgendaContext())
                {
                    Persona bPersona = new Persona
                    {
                        PersonaId = 1,
                        Nombre = "Juan",
                        Apellido = "Pérez",
                        Telefonos = new List<Telefono>
                        {
                            new Telefono
                            {
                                TelefonoId = 1,
                                Numero = "1234567890",
                                Tipo = "Fijo"
                            },
                            new Telefono
                            {
                                TelefonoId = 2,
                                Numero = "0987654321",
                                Tipo = "Móvil"
                            }
                        }
                    };

                    bContexto.Personas.Add(bPersona);
                    bContexto.SaveChanges();

                    foreach (Persona bItem in bContexto.Personas)
                    {
                        Console.WriteLine("Persona recuperada:\nId: {0}, Nombre: {1} - Apellido: {2}",
                            bItem.PersonaId,
                            bItem.Nombre,
                            bItem.Apellido);
                    }

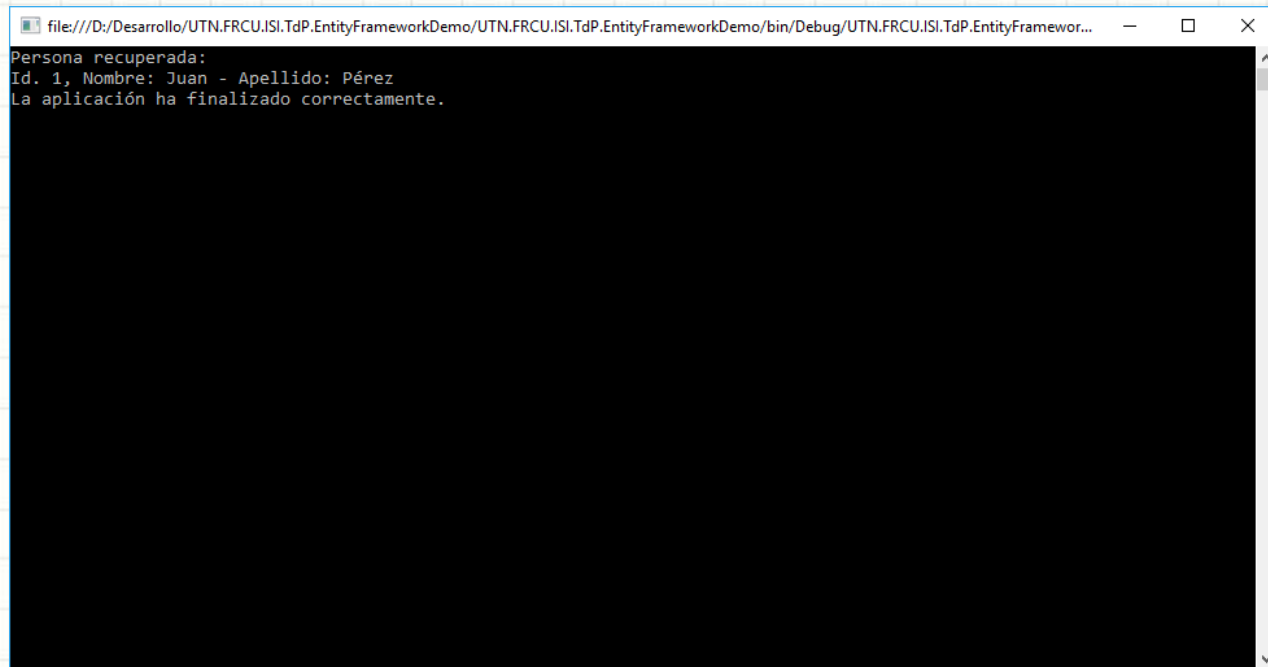
                    Console.WriteLine("La aplicación ha finalizado correctamente.");
                }
            }
            catch (Exception bEx)
            {
                Console.WriteLine("Ha ocurrido un error: {0}", bEx);
            }

            Console.ReadKey();
        }
    }
}
```



# Uso

- Resultado de ejecutar la aplicación:

A screenshot of a Windows console window. The title bar shows the file path: file:///D:/Desarrollo/UTN.FRCU.ISI.TdP.EntityFrameworkDemo/UTN.FRCU.ISI.TdP.EntityFrameworkDemo/bin/Debug/UTN.FRCU.ISI.TdP.EntityFramework... The console output is as follows:

```
Persona recuperada:  
Id. 1, Nombre: Juan - Apellido: Pérez  
La aplicación ha finalizado correctamente.
```



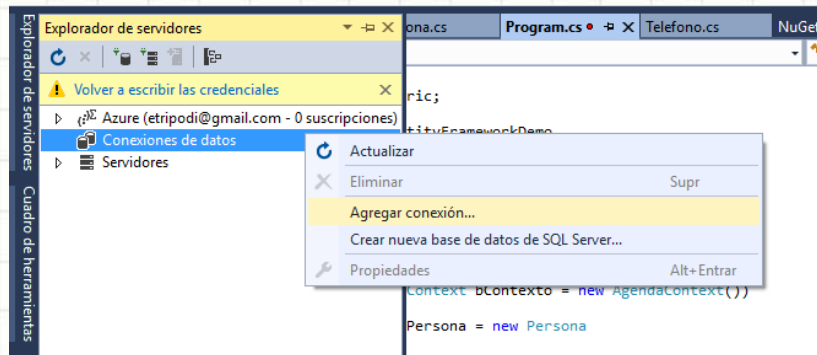
**¿ESTÁN LOS DATOS?**

**¿DÓNDE?**

**¿PODEMOS VERLOS?**

# Verificación

- Se agrega una conexión de datos dentro de Visual Studio:

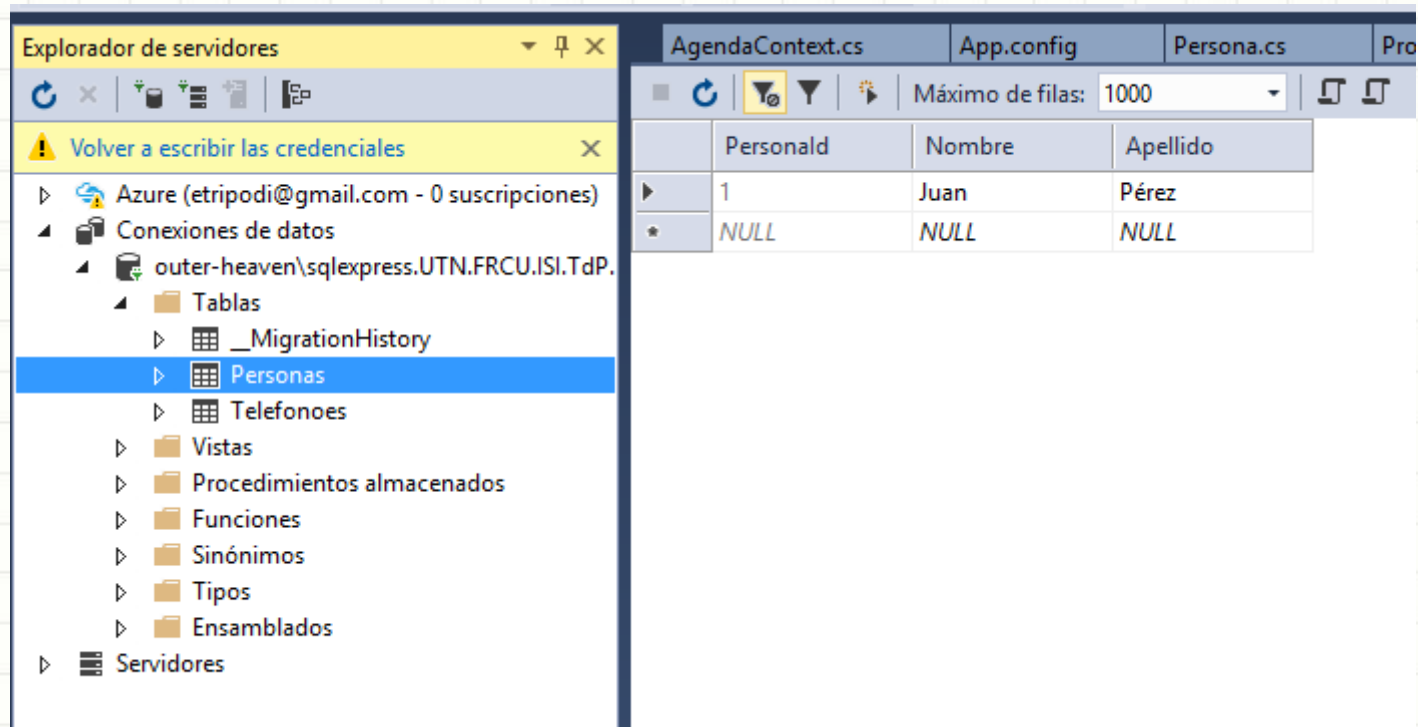
A screenshot of the 'Agregar conexión' (Add connection) dialog box. The dialog is titled 'Agregar conexión' and contains the following sections:

- Origen de datos:** A text box containing 'Microsoft SQL Server (SqlClient)' with a 'Cambiar...' button to its right.
- Nombre del servidor:** A dropdown menu showing '.\SQLExpress' with an 'Actualizar' button to its right.
- Conexión con el servidor:** A section with a dropdown menu for 'Autenticación' (Authentication) set to 'Autenticación de Windows'. Below it are text boxes for 'Nombre de usuario:' and 'Contraseña:', and a checkbox for 'Guardar mi contraseña'.
- Establecer conexión con una base de datos:** A section with two radio buttons. The first, 'Seleccionar o escribir el nombre de la base de datos:', is selected and has a dropdown menu showing 'UTN.FRCU.ISI.TdP.EntityFrameworkDemo.AgendaContext'. The second, 'Adjuntar un archivo de base de datos:', is unselected and has a text box and an 'Examinar...' button.

At the bottom of the dialog are buttons for 'Probar conexión', 'Aceptar', and 'Cancelar', along with an 'Avanzadas...' link.

# Verificación

- Se consultan las tablas:



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Explorador de servidores' pane shows the tree structure: 'Azure (etripodi@gmail.com - 0 suscripciones)' > 'Conexiones de datos' > 'outer-heaven\squlexpress.UTN.FRCU.ISI.TdP.' > 'Tablas' > 'Personas' (selected). A yellow banner at the top of the tree pane says 'Volver a escribir las credenciales'. On the right, the 'Personas' table is displayed with the following structure and data:

| Personald | Nombre | Apellido |
|-----------|--------|----------|
| 1         | Juan   | Pérez    |
| NULL      | NULL   | NULL     |

# Funcionó... ¿por qué?

- Convención sobre configuración (Convention Over Configuration):
  - Si la Base de Datos no existe, se crea una con el nombre del contexto.
  - Si no existe mapeo, cada clase es una tabla y cada atributo un campo.
  - Si no se define clave para la tabla, se toma el campo con el nombre Id o <nombre-tabla>Id.
  - Entre otras reglas.



# ¿Y qué sucede si no quiero seguir las convenciones?

- Hay varias opciones que permiten configurar el modelo de BBDD que se utilizará.
- En el esquema CodeFirst, se puede utilizar DataAnnotations o FluentAPI.

# ¿Y qué sucede si no quiero seguir las convenciones?

- Hay varias opciones que permiten configurar el modelo de BBDD que se utilizará.
- Dentro del esquema CodeFirst hay dos opciones:
  - DataAnnotations.
  - Fluent API.
- Se pueden utilizar DataAnnotations y Fluent API en forma conjunta, donde la precedencia es la siguiente:
  1. Fluent API.
  2. DataAnnotations.
  3. Convenciones por defecto.

# DataAnnotations

- Son atributos que permiten decorar las clases o las propiedades de las clases de dominio, sobrescribiendo las convenciones por defecto en CodeFirst.
- Los atributos que están en el espacio de nombres *System.ComponentModel.DataAnnotations* permiten modificar, entre otras cosas, el tamaño de una columna y si se permiten o no valores nulos.
- Los atributos que están en el espacio de nombres *System.ComponentModel.DataAnnotations.Schema* permiten manipular el esquema de la Base de Datos.

# DataAnnotations

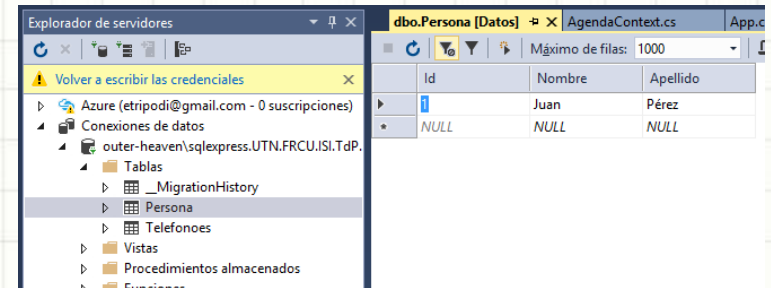
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    [Table("Persona")]
    class Persona
    {
        [Key]
        public int Id { get; set; }

        [Required]
        [StringLength(20)]
        public String Nombre { get; set; }

        [Required]
        [StringLength(20)]
        public String Apellido { get; set; }

        public IList<Telefono> Telefonos { get; set; }
    }
}
```



Explorador de servidores

dbo.Persona [Datos] | AgendaContext.cs | App.c

Máximo de filas: 1000

| Id | Nombre | Apellido |
|----|--------|----------|
| 1  | Juan   | Pérez    |
| *  | NULL   | NULL     |

Conexiones de datos

- outer-heaven\sqlexpress.UTN.FRCU.ISI.TdP.
  - Tablas
    - \_\_MigrationHistory
    - Persona
    - Telefonos
  - Vistas
  - Procedimientos almacenados
  - Funciones

# Fluent API

- Es otra forma de configurar la estructura de la Base de Datos, que en lugar de atributos utiliza los mecanismos de herencia y polimorfismo.
- Permite más opciones de configuración que DataAnnotations, pero es más compleja que ésta.



# Fluent API

```
using System.Data.Entity.ModelConfiguration;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class TelefonoMap : EntityTypeConfiguration<Telefono>
    {
        public TelefonoMap()
        {
            this.ToTable("Telefono");

            this.HasKey(pTelefono => pTelefono.Id);

            this.Property(pTelefono => pTelefono.Id)
                .HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema.DatabaseGeneratedOption.Identity);

            this.Property(pTelefono => pTelefono.Numero)
                .IsRequired()
                .HasMaxLength(50);

            this.Property(pTelefono => pTelefono.Tipo)
                .IsRequired()
                .HasMaxLength(5);
        }
    }
}
```

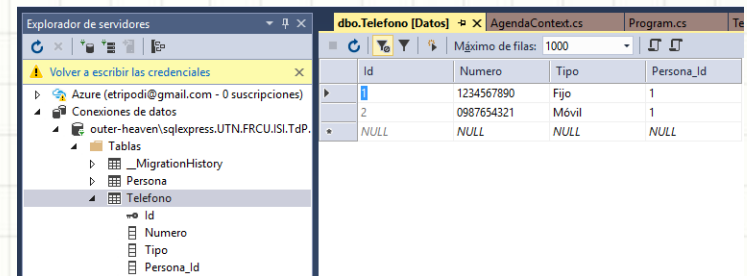
```
using System.Data.Entity;

namespace UTN.FRCU.ISI.TdP.EntityFrameworkDemo
{
    class AgendaContext : DbContext
    {
        public DbSet<Persona> Personas { get; set; }

        public DbSet<Telefono> Telefonos { get; set; }

        protected override void OnModelCreating(DbModelBuilder pModelBuilder)
        {
            pModelBuilder.Configurations.Add(new TelefonoMap());

            base.OnModelCreating(pModelBuilder);
        }
    }
}
```



| Id   | Numero     | Tipo  | Persona_Id |
|------|------------|-------|------------|
| 1    | 1234567890 | Fijo  | 1          |
| 2    | 0987654321 | Móvil | 1          |
| NULL | NULL       | NULL  | NULL       |

# Más información

- <https://msdn.microsoft.com/es-ar/data/ef.aspx>
- <http://www.entityframeworktutorial.net/>



**¿PREGUNTAS?**