Desarrollador de Aplicaciones Web
Programación Web III

Departamento de Ingeniería e Investigaciones Tecnológicas

Entity Framework (Relaciones y LINQ)

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

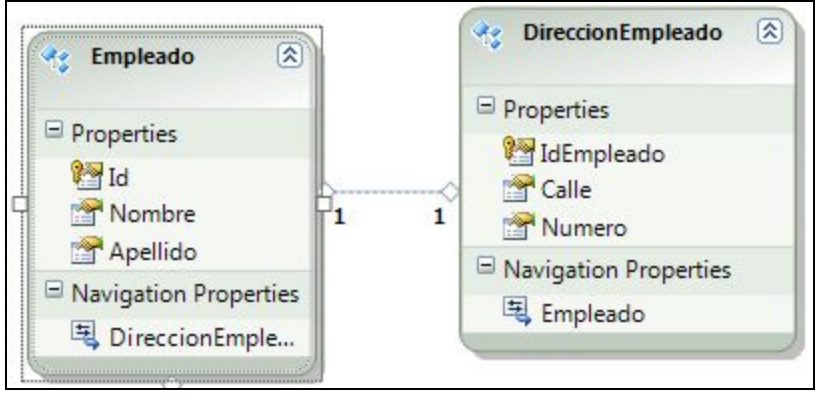
Agenda

- Relaciones 1 a 1
- Relaciones 1 a n
- Relaciones n a n
- Agregar Validaciones al Modelo EF.
- LINQ.
- LINQ To Entities.
- Lazy loading, Eager Loading.

Relaciones entre entidades (Uno a uno):

Cada entidad es una propiedad de la otra entidad. En el ejemplo siguiente se muestra que la entidad DireccionEmpleado es una propiedad de la entidad Empleado; al mismo tiempo la entidad Empleado es una propiedad de la entidad DireccionEmpleado:

Modelo Conceptual (1----1)



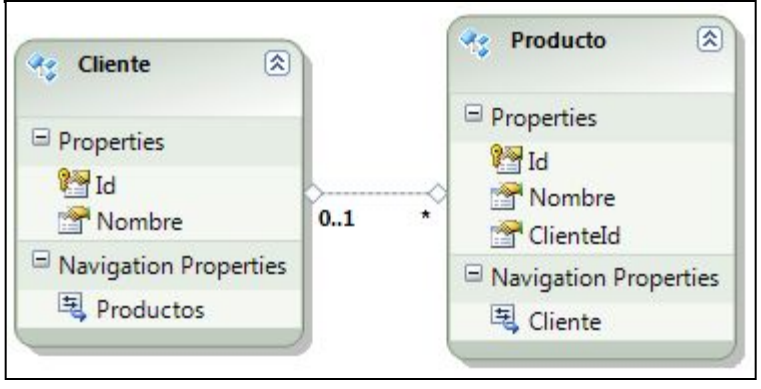
Ejemplo: Nuevo Empleado, nueva Dirección

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

        //Add, nuevo empleado nueva direccion.
        Empleado emp = new Empleado();
        emp.Nombre = "Joaquin J";

        DireccionEmpleado dir = new DireccionEmpleado();

        dir.Calle = "AV Independencia";
        dir.Numero = 3700;

        //
        emp.DireccionEmpleado = dir;

        ctx.Empleados.Add(emp);

        ctx.SaveChanges();
}
```

Relaciones entre entidades (Uno a Muchos):

En el modelo conceptual, en una relación uno a muchos, una entidad contiene una propiedad tipo colección o lista de elementos de otra entidad. En el ejemplo siguiente se muestra que la entidad Cliente contiene una propiedad lista de productos. A su vez, cada elemento Producto contiene una propiedad tipo entidad Cliente.

Modelo Conceptual (1----*)



Ejemplo: Nuevo Cliente, Nuevos Productos

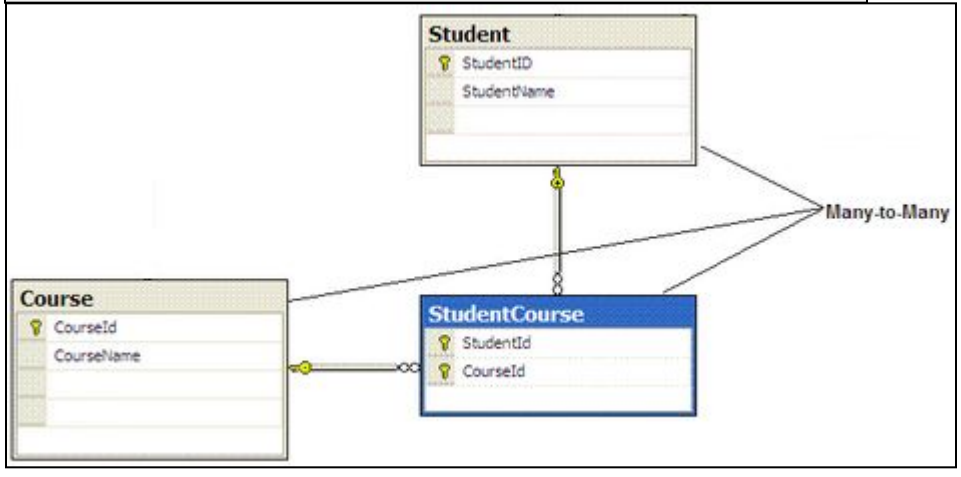
```
protected void NuevoClienteConProductos()
 {
        EFPW3 ctx = new EFPW3()

        //3. a) Entity relacion uno a muchos

        Cliente cli2 = new Cliente();
        cli2.Nombre = "Sancor SA";

        Producto pro = new Producto();
        pro.Nombre = "Leche Larga Vida";
        //pro.NombreFantasia = "Cualquier Cosa";
        cli2.Productos.Add(pro);

        //3. b) Relación uno a muchos con partial
        cli2.Productos.Add(new Producto("Leche Descremada"));

        ctx.Clientes.Add(cli2);
        ctx.SaveChanges();
}
```

Relaciones Muchos a Muchos (EF 6)

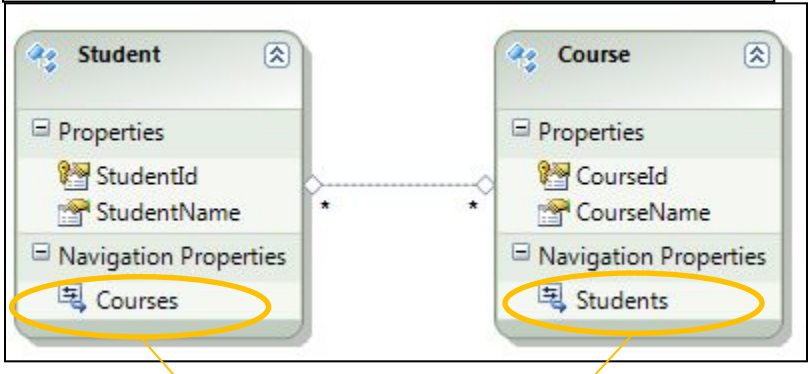
Como es de saberse, en el paradigma relacional, una relación muchos a muchos se representa agregando una tabla intermedia adicional entre dos tablas. De esta forma tenemos dos relaciones del tipo uno a muchos hacia esa nueva tabla adicional.

En el paradigma orientado a objetos, no es necesario agregar una entidad adicional, por el contrario, la relación entre ambas entidades se logra con dos colecciones o listas en cada entidad. De esta manera, en el ejemplo; la entidad Student tiene una colección de cursos y la entidad Cursos tiene una colección de estudiantes.

Modelo Relacional (muchos a muchos)



Modelo Conceptual (muchos a muchos)



Colecciones / Listas

Relaciones Muchos a Muchos (EF 6)

En los siguientes ejemplos de código se muestran operaciones sobre este tipo de relación:

Se crea un nuevo grupo y se le asignan todos los empleados:

```
protected void NuevoGrupoConEmpleados()
{
   EFPW3 context = new EFPW3();

   Grupo g = new Grupo();
   g.Nombre = "Marketing Directo";

   var emps = context.Empleados;

   foreach (Empleado emp in emps)
   {
      g.Empleados.Add(emp);
   }
   context.Grupos.Add(g);

   context.SaveChanges();
}
```

Se eliminan todos los grupos de un Empleado:

```
protected void EliminarGrupos(int IdEmp)
{
   EFPW3 context = new EFPW3();

   var emp = (from e1 in context.Empleados.Include("Grupo")
              where e1.Id == IdEmp
              select e1).First();

  emp.Grupos.Clear();

   context.SaveChanges();
}
```

Se elimina un empleado de un Grupo:

```
protected void EliminarGrupo(int IdEmp, int IdGrupo)
{
   EFPW3 context = new EFPW3();

   var grupo = context.Grupos.Where (g1 => g1.Id == IdGrupo).FirstOrDefault();

   var emp = from e1 in context.Empleados
             where e1.Id == IdEmp
             select e1;

  grupo.Empleados.Remove(emp.First());
   context.SaveChanges();
}
```

Agregar Validaciones al modelo EF:

Se deben crear dos clases:

- 1 clase tipo “partial” **con el mismo nombre** que la entidad de EF. Sobre esta clase se agrega el atributo [ModelMetadataType(...)]

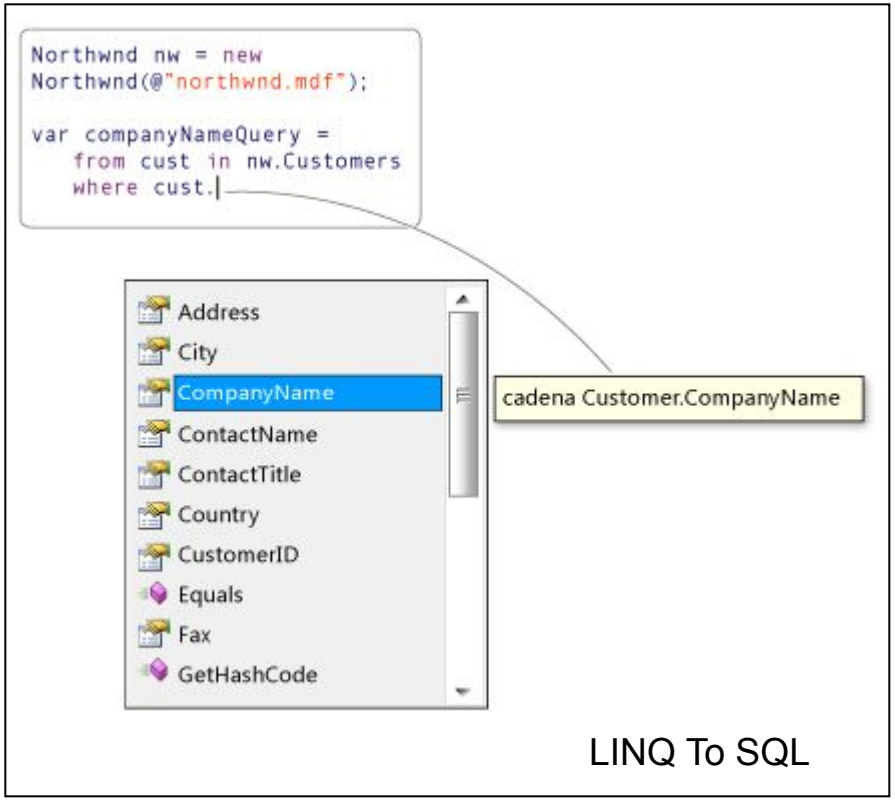
- 1 clase ModelMetaData con los dataannotations que necesite según corresponda.

```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;
namespace EF6_COREClase.Models
{
    [ModelMetadataType(typeof(EmpleadoModelMetaData))]
    public partial class Empleado
    {
    }

    public class EmpleadoModelMetaData
    {
        [Required(ErrorMessage = "Ingrese un nombre para el empleado")]
        [StringLength(30, MinimumLength = 5, ErrorMessage = "El Nombre debe tener al menos 5 caracteres")]
        public string Nombre { get; set; }

        [Required(ErrorMessage = "Debe Ingresar un apellido")]
        public string Apellido { get; set; }
    }
}
```

LINQ

Language-Integrated Query (LINQ) es una innovación introducida en Visual Studio 2008 y .NET Framework versión 3.5 que elimina la distancia que separa el mundo de los objetos y el mundo de los datos.

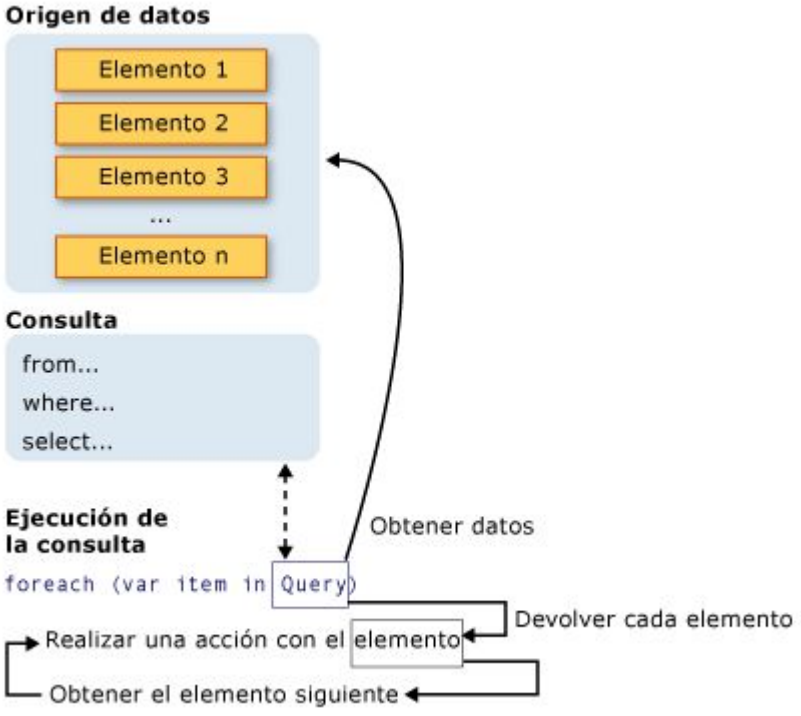
Tradicionalmente, las consultas con datos se expresan como cadenas sencillas, sin comprobación de tipos en tiempo de compilación ni compatibilidad con IntelliSense. Además, es necesario aprender un lenguaje de consultas diferente para cada tipo de origen de datos: bases de datos SQL, documentos XML, etc. LINQ Permite escribir estas consultas directamente en C#. Las consultas se escriben para colecciones de objetos fuertemente tipadas, utilizando palabras clave del lenguaje y operadores con los que se está familiarizado.



LINQ (2)

Las tres partes de una consulta en LINQ

```
//  1. Origen de datos.
int[] numeros = new int[] { 0, 1, 2, 3, 4, 5, 6 };


// 2. Creación de la consulta.
// numQuery es un IEnumerable<int>
var numQuery =
    from num in numeros
    where (num % 2) == 0
    select num;


// 3. Ejecución de la consulta.
foreach (int num in numQuery)
{
    Response.Write(string.Format("{0}", num));
}
```



LINQ To…

LINQ se puede usar con distintos orígenes de datos:

LINQ To SQL: Bases de datos SQL Server

LINQ To Objects: Cadenas, numeros, objetos, etc

LINQ To XML: Documentos XML

LINQ To Datasets: Conjunto de datos de ADO.NET

LINQ To Entities (1)

LINQ se usa en este caso para escribir consultas hacia el EDM. Devuelve las entidades definidas en el modelo conceptual

En este caso usamos la **sintaxis de expresiones de consulta:**

```
private void ListarProductos(int idCliente)
{
    //1) Origen de Datos
    EFPW3 context = new EFPW3();

    //2) Consulta: Sintaxis de consulta
    var productos = from p in context.Producto
                    where p.Cliente.Id == idCliente
                    select p;

    //3) Ejecución de Consulta
    foreach (Producto p in productos)
    {
        lblMensaje.Text += p.Nombre + " - ";
    }
}
```

LINQ To Entities (2)

El mismo ejemplo anterior, pero usando **sintaxis de consulta basada en métodos:**

Es importante destacar que en este tipo de sintaxis se usan las llamadas **expresiones lambdas**, observar: **(p => p.Cliente.Id == idCliente)**

```
private void ListarProductos(int idCliente)
{
    //1) Origen de Datos
    EFPW3 context = new EFPW3();

    //2) Consulta: Sintaxis de Metodo, con expresión lambda
    var productos = context.Producto.Where(p => p.Cliente.Id == idCliente).Select(p1 => p1);

    //3) Ejecución de Consulta
    foreach (Producto p in productos)
    {
        lblMensaje.Text += p.Nombre + " - ";
    }
}
```

LINQ To Entities (2)

Recuerde, existen 3 partes o pasos para realizar una consulta en LINQ:

1) Origen de datos
2) Preparación de la consulta.
3) Ejecución de la consulta.

En LINQ To Entities, es decir, **cuando el origen de datos sea el contexto de EF**, para la ejecución de la consulta (parte o paso 3) tenemos al menos las siguientes formas:

.ToList()
.ToArray()
.FirstOrDefault()
.First()
.SingleOrDefault()
.Single()
.foreach(...)

Lazy Loading (Carga Perezosa)

Lazy loading es un patron de diseño comunmente usado en programación a propósito de posponer la inicialización de un objeto hasta el momento en el cual este es necesitado.
El objetivo es contribuir a la eficiencia; de esta manera los distintos objetos de una clase se irán cargando a medida que los vamos usando.

Estudiemos el siguiente ejemplo: si tenemos una entidad **Empleado,** que tiene como propiedad una entidad **DireccionEmpleado**, y hacemos la siguiente consulta LINQ to entities:

```
EFPW3 ctx = new EFPW3();

var emps = (from em in ctx.Empleados
            select em).ToList();

foreach (Empleado em in emps)
{
    string nombre = em.Nombre;
    string calle = em.DireccionEmpleado.Calle;
}
```

En la siguiente consulta no se recupera información para el objeto **DireccionEmpleado**

Al momento que se quiere acceder a la propiedad **DireccionEmpleado**, es justo en ese preciso instante cuando se recupera su información desde la BD.

```
public EFPW3() : base("name=EFPW3", "EFPW3")
{
    this.ContextOptions.LazyLoadingEnabled = true;
    OnContextCreated();
}
```

Constructor del Contexto:
Por defecto EF trabaja en esta modalidad **LazyLoading (ver)**

Lazy Loading (Carga Perezosa)

Para poder usar Lazy Loading en Net Core 5 se debe:

1) Instalar el package via nuget: **Microsoft.EntityFrameworkCore.Proxies**

2) Activar el uso de LazyLoading en Startup.cs, ConfigureServices:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<EFCoreContext>(ef =>
    ef.UseLazyLoadingProxies().UseSqlServer(ConnectionString));
}
```

Eager Loading (Carga Temprana)

Eager Loading es lo opuesto a Lazy Loading, es decir los objetos relacionados se cargaran o recuperaran la primera vez, en el momento de ejecutar la consulta.

Para eager Loading debemos explicitar el método Include(), pasando como parámetro la-s entidad-es que queremos se recuperen con la consulta.

```
EFPW3 ctx = new EFPW3();

var emps = from em in ctx.Empleados.Include("DireccionEmpleado")
           select em;

foreach (Empleado em in emps)
{
    string nombre = em.Nombre;
    string calle = em.DireccionEmpleado.Calle;
}
```

Se explicita que se recuperará la entidad DireccionEmpleado en la consulta.

NOTA: Include() también se puede usar de la siguiente forma:
...
...**ctx.Empleados.Include(e => e.DireccionEmpleado)**
...

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