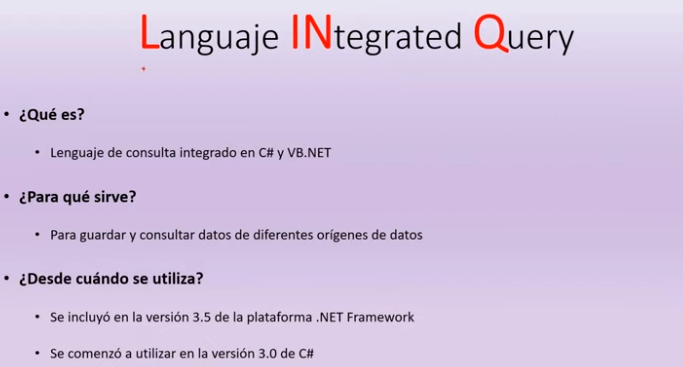
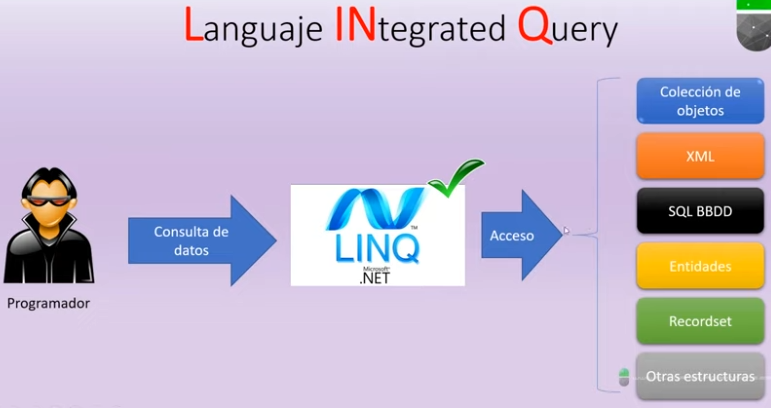
Linq



Es un lenguaje, como Sql.



Ventaja. Un solo lenguaje para muchos tipos de datos distintos. Además, hay una reducción de código, por lo que también se hace mas legible. Ademas hay integración con C# (intellisense disponible).

Enumerable y Queryable son las clases que nos ofrecen todos sus métodos.

No se usan estructuras foreach o for para recorrer.

Hay que importar el espacio de trabajo System.Linq

int[] valoresNumericos = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

Console.WriteLine("Números impares:");

foreach (var item in valoresNumericos)

{

if (item % 2 != 0) Console.Write(item+" / ");

}

Console.WriteLine();

Console.WriteLine();

/\*

Console.WriteLine("Números pares:");

List<int> numerosPares = new List<int>();

for (int i = 0; i < valoresNumericos.Length; i++)

{

if (valoresNumericos[i] % 2 == 0) numerosPares.Add(valoresNumericos[i]);

}

\*/

Console.WriteLine("Números pares:");

IEnumerable<int> numerosPares = from numero in valoresNumericos where numero % 2 == 0 select numero;

foreach (var item in numerosPares)

{

Console.Write(item + " - ");

}

Console.ReadKey();

}

Printeo números pares y números impares. Lo hago de la forma tradicional, luego comento lo que hizo con los números pares. Luego lo de linq:

Aclara que es una interfaz. Usa las palabras reservadas from, in, where, select. numerosPares y números son creadas ahí mismo, solo valoresNumericos (el array) ya existía.

Linq con objetos

class Program

{

static void Main(string[] args)

{

God god = new God();

god.GetBoxeador();

Console.ReadKey();

}

}

class God

{

public List<Empresa> listaEmpresas;

public List<Empleado> listaEmpleados;

public God()

{

listaEmpresas = new List<Empresa>();

listaEmpleados = new List<Empleado>();

listaEmpresas.Add(new Empresa { Id = 1, Nombre = "Lanús" });

listaEmpresas.Add(new Empresa { Id = 2, Nombre = "RapiRobo" });

listaEmpleados.Add(new Empleado { Id = 1, Nombre = "Carlos Ireneo", Cargo = "Panquequero", Salario = 25800, EmpresaId = 1 });

listaEmpleados.Add(new Empleado { Id = 2, Nombre = "Juán Díaz", Cargo = "Boxeador", Salario = 30000, EmpresaId = 2 });

listaEmpleados.Add(new Empleado { Id = 3, Nombre = "Roberto Nazario", Cargo = "Administrativo", Salario = 500000, EmpresaId = 1 });

listaEmpleados.Add(new Empleado { Id = 4, Nombre = "Luis Miguel Majul", Cargo = "Trapeador", Salario = 1250000, EmpresaId = 2 });

}

public void GetBoxeador()

{

IEnumerable<Empleado> boxeadores = from empleado in listaEmpleados where empleado.Cargo == "Boxeador" select empleado;

foreach (var item in boxeadores)

{

item.ShowDatosEmpleado();

}

}

}

public class Empresa

{

public int Id { get; set; }

public string Nombre { get; set; }

public void ShowDatosEmpresa()

{

Console.WriteLine("La empresa N° "+ Id+ " es " + Nombre);

}

}

public class Empleado

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Cargo { get; set; }

public double Salario { get; set; }

//Clave foránea

public int EmpresaId { get; set; }

public void ShowDatosEmpleado()

{

Console.WriteLine("El empleado N° " + Id + " es " + Nombre + ". Su cargo es "+ Cargo + ", su salario es de $"+Salario + " y pertenece a la empresa "+ EmpresaId);

}

}

Mas cosas de Linq

Ordenar datos alfabéticamente: orderby

Ordenar datos alfabéticamente al revés: orderby nombreVariable descending

Luego relaciona empleados y empresas a través de la clave foránea: join.

public void GetEmpleadosLanus()

{

IEnumerable<Empleado> empleadosLanus = from empleado in listaEmpleados join empresa in listaEmpresas on empleado.EmpresaId equals empresa.Id where empresa.Nombre=="Lanús" select empleado ;

foreach (var item in empleadosLanus)

{

item.ShowDatosEmpleado();

}

}

Equals busca la equivalencia entre empleado.EmpresaId y empresa.Id. Luego se asigna cuál tiene que ser el nombre en empresa.nombre. Luego se selecciona a todo el objeto empleado que se corresponda con ese nombre de empresa (a través de EmpresaId.

class Program

{

static void Main(string[] args)

{

God god = new God();

int lineaAux = 0;

bool valido;

do

{

Console.Clear();

valido = true;

Console.WriteLine("Introduzca el N° de Id de la empresa para obtener la información de sus empleados:");

string linea = Console.ReadLine();

try

{

lineaAux = int.Parse(linea);

if (lineaAux < 1 || lineaAux > 2)

{

valido = false;

Console.WriteLine("El número no coincide con ninguna de las empresas existentes");

Console.ReadKey();

}

}

catch (Exception)

{

Console.WriteLine("¡¡Has introducido letritas melonazo!!");

valido = false;

Console.ReadKey();

}

} while (valido == false);

god.GetEmpleadosEmpresa(lineaAux);

Console.ReadKey();

}

}

class God

{

public List<Empresa> listaEmpresas;

public List<Empleado> listaEmpleados;

public God()

{

listaEmpresas = new List<Empresa>();

listaEmpleados = new List<Empleado>();

listaEmpresas.Add(new Empresa { Id = 1, Nombre = "Lanús" });

listaEmpresas.Add(new Empresa { Id = 2, Nombre = "RapiRobo" });

listaEmpleados.Add(new Empleado { Id = 1, Nombre = "Carlos Ireneo", Cargo = "Panquequero", Salario = 25800, EmpresaId = 1 });

listaEmpleados.Add(new Empleado { Id = 2, Nombre = "Juán Díaz", Cargo = "Boxeador", Salario = 30000, EmpresaId = 2 });

listaEmpleados.Add(new Empleado { Id = 3, Nombre = "Roberto Nazario", Cargo = "Administrativo", Salario = 500000, EmpresaId = 1 });

listaEmpleados.Add(new Empleado { Id = 4, Nombre = "Luis Miguel Majul", Cargo = "Trapeador", Salario = 1250000, EmpresaId = 2 });

}

public void GetBoxeador()

{

IEnumerable<Empleado> boxeadores = from empleado in listaEmpleados where empleado.Cargo == "Boxeador" select empleado;

foreach (var item in boxeadores)

{

item.ShowDatosEmpleado();

}

}

public void ShowEmpleadosOrdenados()

{

IEnumerable<Empleado> empleados = from empleado in listaEmpleados orderby empleado.Nombre select empleado;

foreach (var item in empleados)

{

item.ShowDatosEmpleado();

}

}

public void GetEmpleadosLanus()

{

IEnumerable<Empleado> empleadosLanus = from empleado in listaEmpleados join empresa in listaEmpresas on empleado.EmpresaId equals empresa.Id where empresa.Nombre=="Lanús" select empleado ;

foreach (var item in empleadosLanus)

{

item.ShowDatosEmpleado();

}

}

public void GetEmpleadosEmpresa(int Id)

{

IEnumerable<Empleado> empleadosLanus = from empleado in listaEmpleados join empresa in listaEmpresas on empleado.EmpresaId equals empresa.Id where empresa.Id == Id select empleado;

//ahí agrego el Id que se pasa por parámetro para ver de qué empresa se trata.

foreach (var item in empleadosLanus)

{

item.ShowDatosEmpleado();

}

}

}

public class Empresa

{

public int Id { get; set; }

public string Nombre { get; set; }

public void ShowDatosEmpresa()

{

Console.WriteLine("La empresa N° " + Id + " es " + Nombre);

}

}

public class Empleado

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Cargo { get; set; }

public double Salario { get; set; }

//Clave foránea

public int EmpresaId { get; set; }

public void ShowDatosEmpleado()

{

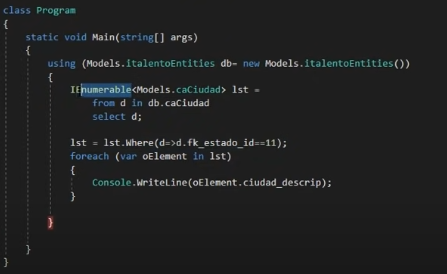
Console.WriteLine("El empleado N° " + Id + " es " + Nombre + ". Su cargo es " + Cargo + ", su salario es de $" + Salario + " y pertenece a la empresa " + EmpresaId);

}

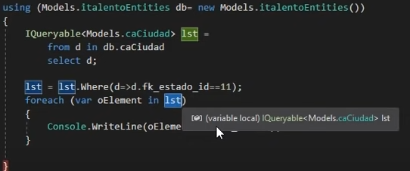
}

IEnumerable IQueryable

hdeleon

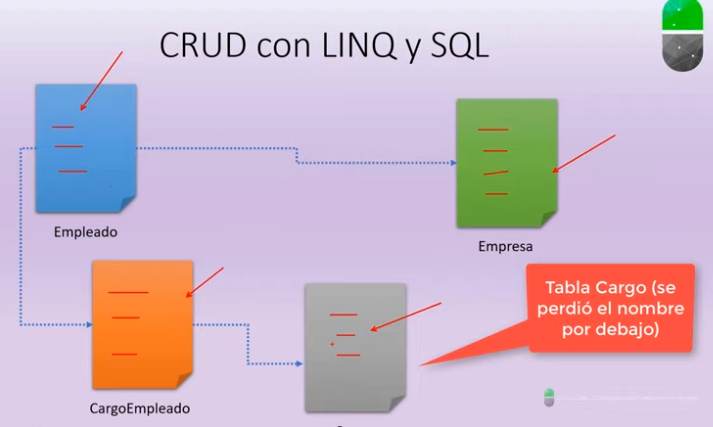


Dice el muchacho que si uno usa el IEnumerable, en este caso, por ejemplo. Se pasa de una base de datos a un dato creado que almacena toda esa info. Lo cual ocupa mucha memoria y hace que funcione mas lento el programa.



Con eso, solamente hace una consulta a la base de datos. Es mas rápido.

Linq y SQL I



Reitera cómo se hace lo de clave foránea con Visual Studio:

Donde dice claves externas, toca agregar clave.

Luego, abajo se escribe esto:

CONSTRAINT [EmpresaFK] FOREIGN KEY ([EmpresaId]) REFERENCES [dbo].[Empresa]([Id]) on delete cascade

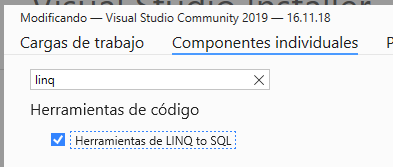
Eso es para que se eliminen los registros en cascada. Si eliminamos un registro en una tabla, se eliminarán en la otra.

Recuerda que hay que tocar origen de datos, y guardar la cadena de conexión:

“ PildorasLinqSQLCRUD “

Hay que habilitar herramientas Linq con SQL en VS.

Desde el instalador de VS.



Reitera que para agregar ConfigurationManager primero hay que agregar la referencia system.configuration luego se agrega el using.

Luego va eso:

InitializeComponent();

string miConexion = ConfigurationManager.ConnectionStrings["Pildoras-Linq-SQL-CRUD.Properties.Settings.PildorasLinqSQLCRUD"].ConnectionString;

//Primero va el nombre del proyecto actual y al final el nombre de la cadena de conexiòn guardada

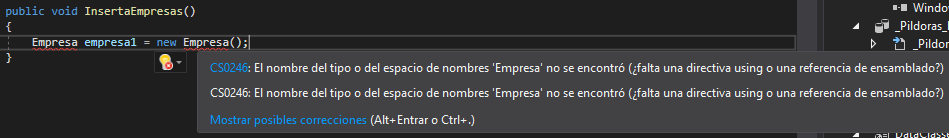
Luego agrega un archivo .dbml que es Clase de Linq a SQL. Para hacer un mapeo (modelo ORM), dice, la bbdd.

Luego declara eso creado con:

DataClasses1DataContext dataContext;

Y lo inicializa pasándole como parámetro miConexion.

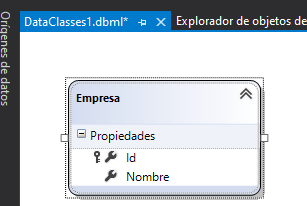
dataContext = new DataClasses1DataContext(miConexion);



Intenta crear un objeto de la clase empresa, pero no reconoce ninguna clase así.

Se va al archivo dbml y ahí te dice que se puede crear clases arrastrando desde el explorador de servidores o el cuadro de herramientas.

Una vez que se arrastra Empresa (la tabla ya creada) ahí, reconoce la clase empresa.



public void InsertaEmpresas()

{

Empresa empresa1 = new Empresa();

empresa1.Nombre = "Lanús";

//Id no porque es autonúmerico.

dataContext.Empresa.InsertOnSubmit(empresa1);

dataContext.SubmitChanges();

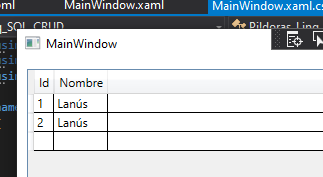
Principal.ItemsSource = dataContext.Empresa;

}

No se usa nada de SQL es todo C# y POO. Con InsertOnSubmit se agrega el objeto a la tabla.

SubmitChanges es como una confirmación, para que los cambios tomen efecto.

La última línea es para que se muestre en pantalla lo hecho:



Lo hice 2 veces y así quedó.

Para que no se cree un objeto cada vez que se ejecute el programa. Se puede comentar lo escrito o agregar esto, para que al iniciarse se borre todo lo previo, se crearía todo de vuelta y estaría solo 1 vez.



Luego crea el método InsertaEmpleados. Es un poco mas complejo ya que hay que asociar empleados y empresas, para eso declara las empresas y les agrega un código jodidongo, utilizando una expresión lambda.

First refiere a la primera aparición de dicho nombre. Empresucha es una variable creada en el momento, usa la palabra clave equals, luegamente.

Empresa empresaLanús = dataContext.Empresa.First(empresucha => empresucha.Nombre.Equals("Lanús"));

Empresa empresaGoogle = dataContext.Empresa.First(empresucha => empresucha.Nombre.Equals("Google"));

En vez de insertar on submit está el insertallonsubmit si hay mas de un valor a agregar.

Así queda:

public void InsertaEmpleados()

{

Empresa empresaLanús = dataContext.Empresa.First(empresucha => empresucha.Nombre.Equals("Lanús"));

Empresa empresaGoogle = dataContext.Empresa.First(empresucha => empresucha.Nombre.Equals("Google"));

List<Empleado> listaEmpleados = new List<Empleado>();

listaEmpleados.Add(new Empleado { Nombre = "Pablo", Apellido = "Krojzl", EmpresaId = empresaLanús.Id });

listaEmpleados.Add(new Empleado { Nombre = "Lucas", Apellido = "Ocampo", EmpresaId = empresaGoogle.Id });

listaEmpleados.Add(new Empleado { Nombre = "Jorge", Apellido = "Goirriarán", EmpresaId = empresaLanús.Id });

listaEmpleados.Add(new Empleado { Nombre = "Antonio", Apellido = "Fernández", EmpresaId = empresaGoogle.Id });

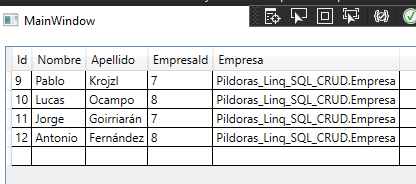
dataContext.Empleado.InsertAllOnSubmit(listaEmpleados);

dataContext.SubmitChanges();

Principal.ItemsSource = dataContext.Empleado;

}

Y esto resulta:



CargoId es un nexo entre cargo y empleado.

public void InsertaEmpleadoCargo()

{

dataContext.ExecuteCommand("delete from CargoEmpleado");

Empleado Pablo = dataContext.Empleado.First(emp => emp.Nombre.Equals("Pablo"));

Empleado Lucas = dataContext.Empleado.First(emp => emp.Nombre.Equals("Lucas"));

Empleado Jorge = dataContext.Empleado.First(emp => emp.Nombre.Equals("Jorge"));

Empleado Antonio = dataContext.Empleado.First(emp => emp.Nombre.Equals("Antonio"));

Cargo pre = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Presidente/a"));

Cargo sec = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Secretario/a"));

Cargo adm = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Administrativo/a"));

Cargo inf = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Informático/a"));

Cargo sic = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Sicario/a"));

Cargo jar = dataContext.Cargo.First(carg => carg.NombreCargo.Equals("Jardinero/a"));

MI FORMA:

CargoEmpleado cargoPablo = new CargoEmpleado();

cargoPablo.Empleado = Pablo;

cargoPablo.CargoId = pre.Id;

CargoEmpleado cargoLucas = new CargoEmpleado();

cargoLucas.Empleado = Lucas;

cargoLucas.CargoId = inf.Id;

CargoEmpleado cargoJorge = new CargoEmpleado();

cargoJorge.Empleado = Jorge;

cargoJorge.CargoId = sic.Id;

CargoEmpleado cargoAntonio = new CargoEmpleado();

cargoAntonio.Empleado = Antonio;

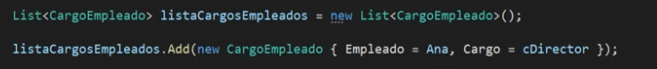
cargoAntonio.CargoId = adm.Id;

dataContext.SubmitChanges();

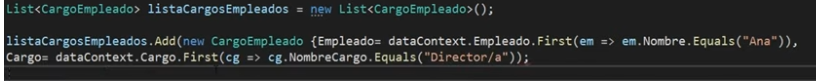
Principal.ItemsSource = dataContext.CargoEmpleado;

}

Además de mi forma podría hacerse así:



O así:



En esta última forma estaría todo dentro del new CargoEmpleado.

Actualizar datos

public void ActualizaEmpleado()

{

Empleado Lucas = dataContext.Empleado.First(emp => emp.Nombre.Equals("Lucas"));

Lucas.Nombre = "Lucas Carlos";

dataContext.SubmitChanges();

Principal.ItemsSource = dataContext.Empleado;

}

Borrar Registros