**ENTITY FRAMEWORK CODE FIRST:**

<https://www.youtube.com/watch?v=_2yMZGIZOso&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=46>

Estas funciones son tanto para SQLServer como par otro motor de base de datos como mysql, Oracle.

Instalar Entity Framework

Captura de pantalla de computadora

Descripción generada automáticamente

**Creamos los modelos:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace \_4\_EntityFramework.Models

{

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

}

}

**Creamos el DbContext:**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace \_4\_EntityFramework.Models

{

public class AppDbContext: DbContext

{

//cada DbSet representa a una tabla que tengamos en la base de datos

//cada tabla esta representada por un modelo en este caso el modelo Persona

public DbSet<Persona> Persona { get; set; }

}

}

**enable-migrations:**

Es un comando que indica que vamos a utilizar Entity Framawork con Code First.

**Comando en consola:**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Migrations-Configurations:**

Podemos indicar que permita realizar los update-database libremente y también indicar que permita la perdida de datos

También agregar registros automáticamente usando el método seed.

namespace \_4\_EntityFramework.Migrations

{

using System;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

internal sealed class Configuration : DbMigrationsConfiguration<\_4\_EntityFramework.Models.AppDbContext>

{

public Configuration()

{

**//ponemos en true por fines didacticos, en la practica no es recomendable poner en true**

**//esto permite realizar libremente el update-database**

**AutomaticMigrationsEnabled = true;**

**//permite la perdida de datos en la base de datos cuando realizamos un update-database**

**AutomaticMigrationDataLossAllowed = true;**

}

protected override void Seed(\_4\_EntityFramework.Models.AppDbContext context)

{

// This method will be called after migrating to the latest version.

// You can use the DbSet<T>.AddOrUpdate() helper extension method

// to avoid creating duplicate seed data.

}

}

}

**Crear la base de datos:**

**Update-database**

Crea la base de datos por medio del DbContext y el modelo que indicamos en el DbSet. Si realizamos cambios en el modelo, con update-database actualizamos la base de datos.

**Crea la base de datos en un localDB. No utiliza un connectionstring**

**Server:** (localdb)\MSSQLLocalDB

**Autenticación:** Autenticación de Windows

Texto

Descripción generada automáticamente

**Connectionstring:**

**Lo usamos si necesitamos indicar la ubicación de una base de datos en especifico.**

Creamos el connectionString en el **web.config**

Y en el **DbContext** indicamos el nombre

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace \_4\_EntityFramework.Models

{

public class AppDbContext: DbContext

{

**//indicamos el nombre del connectionString**

**public AppDbContext():base("DefaultConnection")**

**{**

**}**

//cada DbSet representa a una tabla que tengamos en la base de datos

//cada tabla esta representada por un modelo en este caso el modelo Persona

public DbSet<Persona> Persona { get; set; }

}

}

**Conexión a localDB desde SQL Server Management Studio:**

**Server:** (localdb)\MSSQLLocalDB

**Autenticación:** Autenticación de Windows

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Vemos las bases de datos:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Agregar un campo Edad al modelo y actualizar la base de datos:**

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

public int Edad { get; set; }

}

**Comlando:**

**Update-database**

**Vemos que se ha creado el campo Edad:**

Imagen que contiene Tabla

Descripción generada automáticamente

**Funciones especiales del DbContex:**

<https://www.youtube.com/watch?v=cT4WCRKuJXs&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=47>

* **OnModelCreating:**

Permite realizar cambios sobre los tipos de datos de los campos de las tablas. Que contengan el tipo de dato que se indicó.

**Ejemplo cambiar de DateTime a DateTime2:**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace \_4\_EntityFramework.Models

{

public class AppDbContext: DbContext

{

//cada DbSet representa a una tabla que tengamos en la base de datos

//cada tabla esta representada por un modelo en este caso el modelo Persona

public DbSet<Persona> Persona { get; set; }

//Metosdos Especiales

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

//Permite cambiar el tipo de dato de un campo

modelBuilder.Properties<DateTime>().Configure(x => x.HasColumnType("DateTime2"));

base.OnModelCreating(modelBuilder);

}

}

}

**update-database**

**vemos que el tipo de dato cambio a DateTime2**

**Texto

Descripción generada automáticamente**

**Ejemplo2:**

**Crear una nueva tabla y crear la llave primaria con el campo que se de tipo int y que su nombre inicio con Codigo.**

* **Crear el modelo de la tabla:**

namespace \_4\_EntityFramework.Models

{

public class Direccion

{

public int CodigoDireccion { get; set; }

public string Calle { get; set; }

}

}

* **Crear un nuevo DbSet en el AppDbContext para el modelo Direccion:**

namespace \_4\_EntityFramework.Models

{

public class AppDbContext: DbContext

{

//cada DbSet representa a una tabla que tengamos en la base de datos

//cada tabla esta representada por un modelo en este caso el modelo Persona

public DbSet<Persona> Persona { get; set; }

**//Esta tabla contiene el campo que es de tipo int y que su nombre inicia con Codigo**

**//para que se cree como**

**//primary key como lo indica en el metodo especial**

**public DbSet<Direccion> Direccion { get; set; }**

//Metosdos Especiales

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

//Permite cambiar el tipo de dato de un campo

modelBuilder.Properties<DateTime>().Configure(x => x.HasColumnType("DateTime2"));

**//Permite indicar que campo sera primary Key**

**//en este caso el campo que sea de tipo int y que su nombre inicio con Codigo**

**modelBuilder.Properties<int>().Where(p => p.Name.StartsWith("Codigo")).Configure(p => p.IsKey());**

base.OnModelCreating(modelBuilder);

}

}

}

**Update-database -verbose**

**Vemos que el campo CodigoDireccion es el que se crea como primary Key**

**Texto

Descripción generada automáticamente**

* **ShouldValidateEntity:**

Activar que se pueda validar al momento de eliminar

protected override bool ShouldValidateEntity(DbEntityEntry entityEntry)

{

if(entityEntry.State == EntityState.Deleted)

{

return true;

}

return base.ShouldValidateEntity(entityEntry);

}

* **ValidateEntity:**

Permite que se valide eliminar en un modelo

//Permite que se valide eliminar en el modelo Persona

protected override DbEntityValidationResult ValidateEntity(DbEntityEntry entityEntry, IDictionary<object, object> items)

{

if(entityEntry.Entity is Persona && entityEntry.State == EntityState.Deleted)

{

var entidad = entityEntry.Entity as Persona;

if(entidad.Edad < 18)

{

return new DbEntityValidationResult(entityEntry, new DbValidationError[]

{

new DbValidationError("Edad", "No Esta permitido Eliminar a unn menor de 18 años.")

});

}

}

return base.ValidateEntity(entityEntry, items);

}

* **Actualizamos la base de datos:**

Update-database

**crear el controlador con vistas para ver si se cumple la condición de no eliminar a un menor de 18 años.**

**Texto

Descripción generada automáticamente**

Indicamos el modelo y el DbContext

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**Eliminar registro Edad < 18 no permitido:**

<https://www.entityframeworktutorial.net/EntityFramework4.3/validate-entity-in-entity-framework.aspx>

**en la vista DeleteConfirmed**

capturamos el error

**tutorial obtener mensaje de error:**

<https://www.entityframeworktutorial.net/EntityFramework4.3/validate-entity-in-entity-framework.aspx>

public ActionResult DeleteConfirmed(int id)

{

try

{

Persona persona = db.Persona.Find(id);

db.Persona.Remove(persona);

db.SaveChanges();

return RedirectToAction("Index");

}

catch (DbEntityValidationException dbEx)

{

foreach (DbEntityValidationResult entityErr in dbEx.EntityValidationErrors)

{

foreach (DbValidationError error in entityErr.ValidationErrors)

{

ViewBag.Error = "Error en el Campo:";

ViewBag.Campo = error.PropertyName;

ViewBag.Mensaje = error.ErrorMessage;

}

}

return View("DeleteError");

}

}

**Insertando registros con Entity Framework:**

**AddRange()**

Para agregar una lista o colección de registros.

Ejemplo:

Vamos a insertar 5 registros de forma automática.

public ActionResult Create([Bind(Include = "Id,Nombre,Nacimiento,Edad")] Persona persona)

{

if (ModelState.IsValid)

{

//Crear u listado de personas que se agregaran automaticamente al insertar un registro

var personas = new List<Persona>() { persona }; //agregamos al objeto persona que viene como parametro

//vamos a agregar 5 registoes

for(int i=0; i<=5; i++)

{

personas.Add(new Persona() { Nombre = "Perona\_"+i, Edad = 42, Nacimiento = new DateTime(2022, 07, 07) });

}

db.Persona.AddRange(personas);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(persona);

}

**Formas de Editar:**

public ActionResult EditPesonalizado([Bind(Include = "Id,Nombre,Nacimiento,Edad")] Persona persona)

{

if (ModelState.IsValid)

{

//Método 1: Trae todo el objeto y lo actualiza

//el primer registro que tenga el Id == 2

var personaEditar = db.Persona.FirstOrDefault(x => x.Id == 2);

personaEditar.Nombre = "Nombre Editado Con Método Personalizado 1";

personaEditar.Edad = 2000;

//Método 2: Actualizacion Parcial

//Solo vamos a Actualizar el campo Nombre el Edad no se debe Actualizar

var personaEditar2 = new Persona();

personaEditar2.Id = 6;

personaEditar2.Nombre = "Nombre Editado Con Método Personalizado 2";

personaEditar2.Edad = 500;

//indicamos el objeto que se va actualizar en la db

db.Persona.Attach(personaEditar2);

//indicamos que propiedad campo es el que se va a Actualizar

db.Entry(personaEditar2).Property(x => x.Nombre).IsModified = true;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(persona);

}

**BORRAR REGISTROS:**

[**https://www.youtube.com/watch?v=WycuKIGCa7o&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=50**](https://www.youtube.com/watch?v=WycuKIGCa7o&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=50)

**RemoveRange:**

<https://www.youtube.com/watch?v=MYyqRBpd7Ic>

public ActionResult DeleteLista()

{

//le enviamos una lista con los Id que vamos a eliminar

//en este caso los Id maayores a 13

var personas = db.Persona.Where(x => x.Id >= 13).ToList();

db.Persona.RemoveRange(personas);

db.SaveChanges();

return RedirectToAction("Index");

}

**Formas de Seleccionar varias columnas:**

**Seleccionar Todas Las Columnas de una Tabla:**

**1 Retorna a la vista una lista de tipo string.**

public ActionResult AllCampos()

{

var listaCampos = db.Persona.ToList();

return View(listaCampos);

}

//Seleccionar una Columna:

//La consulta devuelve una lista tipo String

public ActionResult SelectCampo()

{

var model = db.Persona.Select(x => x.Nombre).ToList();

//en la vista debemos indicar que vamos a recibir una lista de tipo string

return View(model);

}

**Vista:**

<!--Esta reciviendo una lista de tipo string-->

@model List<string>

@{

ViewBag.Title = "Seleciona unicamente el campo Nombre de la base de datos:";

}

<h3>@ViewBag.Title</h3>

<h4>

El controlador realiza una consulta con Entity y envia a la vista una Lista de tipo String.

</h4>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table table-bordered">

<tr>

<th>

Nombre:

</th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@item

</td>

</tr>

}

</table>

**2 Seleccionar varios campos objetos anonimos:**

* **Debemos crear un clase modelo para mapear los campos.**

**Esto para enviarlo a la vista para que la pueda recorrer.**

**Ya que utiliza un objeto anonimo en el select y los objetos anónimos no se pueden usar en la vista.**

**Clase para mapper campos PersonaNombreEdad**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace \_4\_EntityFramework.Models.Anonimo

{

public class PersonaNombreEdad

{

public string Nombre { get; set; }

public int Edad { get; set; }

}

}

**Controlador:**

//Seleccionar varios campos y proyectarlos a una clase de tipo anomima, es anomima por que no tiene nombre.

//Para poder usar el objeto anomimo en la Vista es necesario mamper a una nueva clase

//que contengoa solo los campos a utilizar en este caso Nombre y Edad.

public ActionResult VariosCampos()

{

List<PersonaNombreEdad> model = db.Persona.Select(x => new PersonaNombreEdad() { Nombre = x.Nombre, Edad = x.Edad }).ToList();

return View(model);

}

**Vista:**

**<!--indicamos que recibe el modelo PersonaNombreEdad-->**

@model IEnumerable<\_4\_EntityFramework.Models.Anonimo.PersonaNombreEdad>

@{

ViewBag.Title = "Selecionar unicamente los campo Nombre y Edad de la base de datos:";

}

<h3>@ViewBag.Title</h3>

<h4>

El controlador realiza una consulta con Entity y envia a la vista un objeto anonimo que mapeamos

en el modelo PersonaNombreEdad, para poder recorrerlo en la vista.

</h4>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table table">

<tr>

<th>

Nombre:

</th>

<th>

Edad:

</th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@item.Nombre

</td>

<td>

@item.Edad

</td>

</tr>

}

</table>

**3 Seleccionar varios campos y proyectarlos sobre la misma clase modelo**

**Para no utilizar objetos anónimos**

**Controlador:**

//Seleccionar varis campos y proyectarlo sobre la misma clase para no usar objetos anonimos:

//en este metodo retornamos un listado de objeto persona.

//para poder devolver un listado de personas sin usar objeto anonimo. realializamos un select a persona con un objeto anonimo

//y devuelve un list

//sobre ese list realizamos nuevamente otro select pero ahora crearmos un objeto persona que retornamos a la vista

public ActionResult VariosCamposPersona()

{

var model = db.Persona.Select(x => new { Nombre = x.Nombre, Edad = x.Edad }).ToList()

.Select(x => new Persona() { Nombre = x.Nombre, Edad = x.Edad }).ToList();

return View(model);

}

Vista:

<!--indicamos que recibe el modelo PersonaNombreEdad-->

@model IEnumerable<\_4\_EntityFramework.Models.Persona>

@{

ViewBag.Title = "Selecionar unicamente los campo Nombre y Edad de la base de datos sin usar objeto anonimo:";

}

<h3>@ViewBag.Title</h3>

<h4>

El controlador realiza una consulta con Entity y envia a la vista un objeto de tipo persona unicamente con los campos

que le indicamos Nombre Y edad.

</h4>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table table">

<tr>

<th>

Nombre:

</th>

<th>

Edad:

</th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@item.Nombre

</td>

<td>

@item.Edad

</td>

</tr>

}

</table>

**propiedades de navegación con Llaves foráneas:**

<https://www.youtube.com/watch?v=kOoGgP9iXxA&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=53>

Esta propiedad nos permite navegar de un modelo a otro cuando estos están relacionados.

Vamos a relacionar dos modelos, persona y dirección.

Suponemos que una persona puede tener cero o más direcciones.

Y a cada dirección le corresponde una persona.

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

public int Edad { get; set; }

//relacion cero a muchos, una persona puede tener muchas direccione.

public List<Direccion> Direcciones { get; set; }

}

public class Direccion

{

public int CodigoDireccion { get; set; }

public string Calle { get; set; }

//relaciones cero a muchos a una direccion le corresponde muchas personas.

public Persona Persona { get; set; }

}

* **Relacionar las tablas:**

**Lo hacemos en el AppDbContext: DbContext:**

public class AppDbContext: DbContext

{

//Metosdos Especiales API FLUENTE

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

//Relacionar Modelos Doreccion Persona

//toda entidad Direccion debe tener asignada una entidad Persona

//se crea en la tabla Direccion un campo PersonaID que hace referencia una llave foranea a la tabla persona

modelBuilder.Entity<Direccion>().HasRequired(x => x.Persona);

base.OnModelCreating(modelBuilder);

}

* **Crear las relaciones en la base de datos:**

Update-database -force

Vemos que se creo en la tabla Direcciones la llave foránea Persona\_id:

Texto

Descripción generada automáticamente

* Insertar un registro en la tabla Direcciones

También debemos agregar el id de una Persona en el campo Persona\_Id.

* Controlador:

public ActionResult AgregarDireccionFK()

{

//creamos un objeto para indicar que Id vamos a usar

var persona = new Persona() { Id = 9 };

//con Attach indicamos que el registro ya existe en la base de datos, para que no intete crear uno nuevo

//sin no que es un exitente

db.Persona.Attach(persona);

//indicamos que queremos asignar una nueva direccion a persona con Id=9, si no usamos Attach intenta crear uno nuevo.

db.Direccion.Add(new Direccion() { Calle = "Calle principal 1", Persona = persona });

db.SaveChanges();

return View();

}

* Vista:

@model \_4\_EntityFramework.Models.Persona

@{

ViewBag.Title = "Agregar Direccion FK";

}

<h2>@ViewBag.Title</h2>

<hr />

<div class="container">

<h3>Se ha agregado un registro nuevo a la tabla Direcciones y una llave foranea a persona\_Id = 9</h3>

</div>

<hr />

<div>

@Html.ActionLink("Back to List", "Index")

</div>

* **Registro en la tabla:**

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

**Lazy loading:**

Permite utilizar propiedades de navegación entre las propiedades

Utiliza, solamente cuando accedamos a ellas.

**Virtual:**

Para utilizarlo debemos indicar que es de forma virtual la propiedad del modelo.

Modelo:

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

public int Edad { get; set; }

//relacion cero a muchos, una persona puede tener muchas direccione.

public virtual List<Direccion> Direcciones { get; set; }

}

**Controlador:**

//Obtener la direccion de una persona usando Virtual en el modelo

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public ActionResult PersonaDireccionVirtual()

{

//SE REALIZAN DOS CONSULTAS SEPARADAS:

//obtenemos a la persona con Id ==11

var persona = db.Persona.FirstOrDefault(x => x.Id == 11);

//de la persona con Id = 11 Obtenemos las direcciones que tiene registradas

//para obtener las direcciones devemos agregar Virtual en el modelo.

var model = persona.Direcciones.ToList();

ViewBag.PersonaId11 = persona;

return View(model);

}

**Vista:**

@model IEnumerable<\_4\_EntityFramework.Models.Direccion>

@{

ViewBag.Title = "Persona Direccion usando Virtual ó Lazy Loading en el modelo";

}

<h2>@ViewBag.Title</h2>

<h4>Se están realizanado dos consultas una para los datos de persona y otro pra las direcciones.</h4>

<hr />

<div>

<h3>Desplegar El Objeto con ViewBag</h3>

<table class="table">

<tr>

<th>

ID

</th>

<th>

Nombre

</th>

<th>

Edad

</th>

<th>

Nacimiento

</th>

<th>

Direcciones

</th>

</tr>

<tr>

<td>

@ViewBag.PersonaId11.Id

</td>

<td>

@ViewBag.PersonaId11.Nombre

</td>

<td>

@ViewBag.PersonaId11.Edad

</td>

<td>

@ViewBag.PersonaId11.Nacimiento

</td>

<td>

<ul>

@foreach (var item in Model)

{

<li>

@item.Calle

</li>

}

</ul>

</td>

</tr>

</table>

</div>

<hr />

<h3>Desplegar un objeto completon con Display</h3>

<h4>Para dar formato con clases debemos usar un DisplayTemplate</h4>

<h4>

@Html.Display("PersonaId11")

</h4>

<hr />

<h3>Direcciones las está enviado la vista en el model Direccion:</h3>

<h4>Como es una lista la recorremos con un Foreach</h4>

<ul>

@foreach (var item in Model)

{

<li>

@item.Calle

</li>

}

</ul>

**INCLUDE EAGER LOADING:**

**No utiliza virtual. Es mas recomendado utilizar eager loading.**

Obtener la dirección de una sola persona

Modelo persona:

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

public int Edad { get; set; }

//relacion cero a muchos, una persona puede tener muchas direccione.

public List<Direccion> Direcciones { get; set; }

}

**Controlador:**

**retorna un solo Objeto en la vista no necesitamos un IEnumerable para recorrerlo.**

//Obtener a una sola persona:

public ActionResult UnaPersonaInclude()

{

//buscamos a la persona con Id = 11, y incluimos su direccion en una sola consulta:

**var model = db.Persona.Include("Direcciones").FirstOrDefault(x => x.Id == 11);**

//retorna un solo Objeto en la vista no necesitamos un IEnumerable para recorrerlo.

return View(model);

}

**Vista:**

<!--NO SE USA UN IEnumerable para recorrerlo-->

@model \_4\_EntityFramework.Models.Persona

@{

ViewBag.Title = "Una Persona Include";

}

<h2>@ViewBag.Title</h2>

<h3>Devuleve un objeto un solo objeto persona, No se necesita un IEnumerable para recorrerlo.</h3>

<div>

<h4>Persona</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Nombre)

</dt>

<dd>

@Html.DisplayFor(model => model.Nombre)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Nacimiento)

</dt>

<dd>

@Html.DisplayFor(model => model.Nacimiento)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Edad)

</dt>

<dd>

@Html.DisplayFor(model => model.Edad)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Direcciones)

</dt>

@for (int i = 0; i <= Model.Direcciones.Count - 1; i++)

{

<!--el campo Direcciones contien todos los campos de la tabla Direcion -->

<!--Elegimos cual necesitamos utilizar en este caso calle-->

<dd>@Html.DisplayFor(model => model.Direcciones[i].Calle)</dd>

}

</dl>

</div>

<p>

@Html.ActionLink("Back to List", "Index")

</p>

**Ontener Nombre por medio de su Direccion:**

**Usando virtual lazy loading:**

**Modelos:**

public class Direccion

{

public int CodigoDireccion { get; set; }

public string Calle { get; set; }

//relaciones cero a muchos a una direccion le corresponde muchas personas.

public virtual Persona Persona { get; set; }

}

public class Persona

{

public int Id { get; set; }

public string Nombre { get; set; }

public DateTime Nacimiento { get; set; }

public int Edad { get; set; }

//relacion cero a muchos, una persona puede tener muchas direccione.

public virtual List<Direccion> Direcciones { get; set; }

}

**Controlador:**

//Obtener el Nombre de la pesona por medio de su dirección:

//para tener acceso al nombre debemos colocar virtual en campo persona del modelo direccion:

public ActionResult ObtenerNombrePorDireccion()

{

//hacemos una consulta para obtener la direccion con CodigoDireccion = 4:

var direccion = db.Direccion.FirstOrDefault(d => d.CodigoDireccion == 4);

//Enviar solo el nombre usando un Viewbag

ViewBag.nombre = direccion.Persona.Nombre;

//retornamos un objeto direccion que contiene todos los campos

return View(direccion);

}

**Vista:**

@model \_4\_EntityFramework.Models.Direccion

@{

ViewBag.Title = "Obtener Nombre Por Medio de su Direccion";

}

<h2>@ViewBag.Title</h2>

<hr />

<h4>Recibimos solo el nombre por medio de un ViewBag</h4>

<dl class="dl-horizontal">

<dt>

Nombre:

</dt>

<dd>

@ViewBag.nombre

</dd>

</dl>

<hr />

<h4>Recibimos un modelo direccion y mostramos el nombre y demas campos por medio de su dirección</h4>

<p>Nombre es un campo del modelo Persona y Calle, Edad Son campos del modelo Direcion</p>

<p>Para tener acceso a un campo de otra tabla debemos agregar virtual al campo en el modelo, o usar include en la consulta del controlador.</p>

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Persona.Nombre):

</dt>

<dd>

@Html.DisplayFor(model => model.Persona.Nombre)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Persona.Edad):

</dt>

<dd>

@Html.DisplayFor(model => model.Persona.Edad)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Calle):

</dt>

<dd>

@Html.DisplayFor(model => model.Calle)

</dd>

</dl>

<hr />

<p>

@Html.ActionLink("Edit", "Edit", new { id = Model.CodigoDireccion }) |

@Html.ActionLink("Back to List", "Index")

</p>

**INNER JOIN Y GROUP JOIN SIN PROPIEDADE DE NAVEGACIÓN:**

**USANDO OBJETO Y CLAASE ANINIMA:**

**Creamos dos modelos:**

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public DateTime ContractDate { get; set; }

}

Cuando usamos inner join debemos agregar manualmente cual será la FK:

public class Job

{

public int JobCode { get; set; }

public string JobDescription { get; set; }

**//este campo sera la FK lo agregamos manualmente**

**public int EmployeeId { get; set; }**

public Employee Employee { get; set; }

}

**Agregamos un controlador:**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**appDbContext:**

indicamos en el DbContext:

el DbSet para las tablas Empledo y Puesto y cuál será la FK en la tabla Puesto, para el INNER JOIN:

public class AppDbContext: DbContext

{

**//TABLAS USADAS EN EL INNER JON:**

**//cada DbSet representa a una tabla que tengamos en la base de datos**

**public DbSet<Employee> Employees { get; set; }**

**//Esta tabla contiene el campo que es de tipo int y que su nombre inicia con Codigo para que se cree como**

**//primary key como lo indica en el metodo especial**

**public DbSet<Job> Jobs { get; set; }**

//API FLUENTE:

//Metosdos Especiales API FLUENTE

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

//INNER JOIN

//Permite indicar que campo sera primary Key

//en este caso el campo que sea de tipo int y que su nombre termine con Code

modelBuilder.Properties<int>().Where(p => p.Name.EndsWith("Code")).Configure(p => p.IsKey());

//debemos indicar cual sera la FK en la tabla Job, aqui es EmployeeId

modelBuilder.Entity<Job>().HasRequired(p => p.Employee).WithMany().HasForeignKey(P => P.EmployeeId);

base.OnModelCreating(modelBuilder);

}…

**Actulizamos la base de datos:**

Update-datebase -force

**Se han creado las tablas:**

Texto

Descripción generada automáticamente

**Clase anónima:**

public class JobsInnerEmployeeAnonimo

{

public int JobCode { get; set; }

public string JobDescription { get; set; }

//este campo sera la FK lo agregamos manualmente

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public DateTime ContractDate { get; set; }

}

**CONTROLADOR:**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INNER JOIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//obtener el Employee y su tabajo con codigo JobCode == 1:

//ANONIMO utilizamos una un objeto anomino y su clase para maperar los campos

public ActionResult OneEmployeeInnerJob()

{

//Hacemos el INNER JOIN ENTRE Jobs y Employees:

var JobInnerJoinEmployee = db.Jobs.Join(db.Employees, //Indicamos la union entre la tabla Jobs y la tabla Employees.

job => job.EmployeeId, emp => emp.Id, //la union sera usando los campos FK Job.EmployeeId y PK Employee.Id, usando lambda.

(job, emp) => new JobsInnerEmployeeAnonimo() { //Mapeamos los campos a la clase anonima

JobCode = job.JobCode,

JobDescription = job.JobDescription,

EmployeeId = job.EmployeeId,

Employee = job.Employee,

//campos tabla Employee,

Id = emp.Id,

Name = emp.Name,

Age = emp.Age,

ContractDate = emp.ContractDate

}).FirstOrDefault(x => x.JobCode == 1); //De la union (job, emp) Creamos un objeto anonimo y obtenemos el primer resultado de JobCode == 1.

//Retornamos la consulta con los campos del objeto y la clase anonima JobsInnerEmployeeAnonimo:

return View(JobInnerJoinEmployee);

}

**Vista:**

**La vista utiliza el modelo anonimo:**

@model \_4\_EntityFramework.Models.Anonimo.JobsInnerEmployeeAnonimo

@{

ViewBag.Title = "INNER JOIN CON OBJETO Y CLASE ANONIMA:";

}

<h2>@ViewBag.Title</h2>

<h3>Consulta INNER JOIN a las tablas Jobs y Employees CON JobCode = 1</h3>

<div>

<h3>Campos de la tabla Employees</h3>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Id)

</dt>

<dd>

@Html.DisplayFor(model => model.Id)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd>

@Html.DisplayFor(model => model.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Age)

</dt>

<dd>

@Html.DisplayFor(model => model.Age)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ContractDate)

</dt>

<dd>

@Html.DisplayFor(model => model.ContractDate)

</dd>

</dl>

<hr />

<h3>Campos de la tabla Jobs </h3>

<p>el campo Employee de la tabla Jobs contiene todos los campos relacionados de la tabla Employee:</p>

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.JobCode)

</dt>

<dd>

@Html.DisplayFor(model => model.JobCode)

</dd>

<dt>

@Html.DisplayNameFor(model => model.JobDescription)

</dt>

<dd>

@Html.DisplayFor(model => model.JobDescription)

</dd>

<dt>

@Html.DisplayNameFor(model => model.EmployeeId)

</dt>

<dd>

@Html.DisplayFor(model => model.EmployeeId)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Employee)

</dt>

<dd>

@Html.DisplayFor(model => model.Employee)

</dd>

</dl>

</div>

<p>

@Html.ActionLink("Edit", "Edit", new { id = Model.JobCode }) |

@Html.ActionLink("Back to List", "Index")

</p>

GROUPJOIN:

USA CLASE Y OBJETO ANONIMO: Para mapear los campos de ambas tablas y enviarlos a la vista.

los campos de una taba serán de tipo IEnumerable, en este caso la tabla Job

<https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.groupjoin?view=net-6.0>

CLASE Y OBJETO ANONIMO:

public class groupJoinAnonimo

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public DateTime ContractDate { get; set; }

//los campos de la tabla Jobs debenser de tipo IEnumerable

public IEnumerable<int> JobCode { get; set; }

public IEnumerable<string> Jobs { get; set; }

}

CONTROLADOR:

EmployeesController:

//GROUPJOIN

//USA CLASE Y OBJETO ANONIMO: Para mapear los campos de ambas tablas y enviarlos a la vista.

//los campos de una taba seran de tipo IEnumerable, en este caso la tabla Job

//Obtener a una persoan y su listado de trabajos:

public ActionResult GroupJoinEmployeeJob()

{

var query = db.Employees.GroupJoin(db.Jobs,

emp => emp.Id,

job => job.EmployeeId,

(emp, jobCollection) =>

new groupJoinAnonimo()

{

Id = emp.Id,

Name = emp.Name,

Age = emp.Age,

ContractDate = emp.ContractDate,

//los campos de la tabla job son de tipo IEnumerable,

//los definimos asi enel objeto y clae anonimo

JobCode = jobCollection.Select(job => job.JobCode),

Jobs = jobCollection.Select(job => job.JobDescription)

}).FirstOrDefault(x => x.Id == 1);

return View(query);

}

VISTA:

Recibe el modelo anonimo groupJoinAnonimo,

En donde mapeamos los campos de las vistas que se usan el GroupJoin.

En esta caso los campos de Job son de tipo IEnumerable los cuales recorremos con un ForEach().

@model \_4\_EntityFramework.Models.Anonimo.groupJoinAnonimo

@{

ViewBag.Title = "GroupJoinEmployeeJob";

}

<h2>@ViewBag.Title</h2>

<h3>GroupJoin devuelve los campos de la tabla Job de tipo IEnumerable</h3>

<h4>Usamos una clase y objeto Anonimo para mapear los campos que se utilizarán de ambas tablas.</h4>

<br />

<p>Se está mostrando el empleado con id = 1</p>

<div>

<h4>Employee</h4>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Id)

</th>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th>

@Html.DisplayNameFor(model => model.ContractDate)

</th>

<th>

@Html.DisplayNameFor(model => model.JobCode)

</th>

<th>

@Html.DisplayNameFor(model => model.Jobs)

</th>

</tr>

<tr>

<td>

@Html.DisplayFor(model => model.Id)

</td>

<td>

@Html.DisplayFor(model => model.Name)

</td>

<td>

@Html.DisplayFor(model => model.Age)

</td>

<td>

@Html.DisplayFor(model => model.ContractDate)

</td>

<td>

<ul>

@foreach (var item in Model.JobCode)

{

<li> @item </li>

}

</ul>

</td>

<td>

<ul>

@foreach (var item in Model.Jobs)

{

<li> @item </li>

}

</ul>

</td>

</tr>

</table>

</div>

<hr />

<p>

@Html.ActionLink("Back to List", "Index")

</p>

**GROUPJOIN**

**obtener a todos los empleados y sus trabajos:**

**CLASE ANONIMA:**

Mapea los campos de ambas tablas en este caso Employees y Jobs.

Los campos de Jobs seran de tipo IEnumerable.

public class groupJoinAnonimo

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public DateTime ContractDate { get; set; }

//los campos de la tabla Jobs debenser de tipo IEnumerable

public IEnumerable<int> JobCode { get; set; }

public IEnumerable<string> Jobs { get; set; }

}

**CONTROLADOR:**

//GROUPJOIN

//obtener a todos los empleados y sus trabajos:

public ActionResult GroupJoinAllEmployeeJob()

{

var query = db.Employees.GroupJoin(db.Jobs,

emp => emp.Id,

job => job.EmployeeId,

(emp, jobCollection) =>

new groupJoinAnonimo()

{

Id = emp.Id,

Name = emp.Name,

Age = emp.Age,

ContractDate = emp.ContractDate,

JobCode = jobCollection.Select(job => job.JobCode),

Jobs= jobCollection.Select(job => job.JobDescription)

}

);

return View(query);

}

**VISTA:**

**RECIBE UN IEnumerable del moselo groupJoinAnonimo.**

**Ya que lista a todos los empleados y todos sus trabajos:**

@model IEnumerable<\_4\_EntityFramework.Models.Anonimo.groupJoinAnonimo>

@{

ViewBag.Title = "GroupJoinAllEmployeeJob";

}

<h2>@ViewBag.Title</h2>

<h3>Mustra todos los Empleados con un listado de todos los Trabajos.</h3>

<h3>Utilizamos una clae y un objeto ANONIMO.</h3>

<p>REcibimos un IEnumerable de tipo model groupJoinAnonimo, para recorrer el listado de trabajos. </p>

<br />

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Id)

</th>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th>

@Html.DisplayNameFor(model => model.ContractDate)

</th>

<th>

@Html.DisplayNameFor(model => model.JobCode)

</th>

<th>

@Html.DisplayNameFor(model => model.Jobs)

</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.DisplayFor(modelItem => item.ContractDate)

</td>

<td>

<ul>

@foreach (var jcode in item.JobCode)

{

<li>

@jcode

</li>

}

</ul>

</td>

<td>

<ul>

@foreach (var jobs in item.Jobs)

{

<li>

@jobs

</li>

}

</ul>

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id = item.Id }) |

@Html.ActionLink("Details", "Details", new { id = item.Id }) |

@Html.ActionLink("Delete", "Delete", new { id = item.Id })

</td>

</tr>

}

</table>

**VER LOS QUERY GENERADOS POR ENTITY FRAMEWORK:**

Debemos convertir a ToString las consultas:

Y luego colocar un breakPoint y ver el código sql Generado.

**Ejemplo:**

Ver el código sql generado por la constula GroupJoinAllEmployeeJob()

Group Join de todos los empleados y los trabajos que tiene asignados:

**Convertir a ToString:**

public ActionResult GroupJoinAllEmployeeJob()

{

var query = db.Employees.GroupJoin(db.Jobs,

emp => emp.Id,

job => job.EmployeeId,

(emp, jobCollection) =>

new groupJoinAnonimo()

{

Id = emp.Id,

Name = emp.Name,

Age = emp.Age,

ContractDate = emp.ContractDate,

JobCode = jobCollection.Select(job => job.JobCode),

Jobs= jobCollection.Select(job => job.JobDescription)

}

**).ToString();**

return View(query);

}

**Colocar un BreakPoint:**

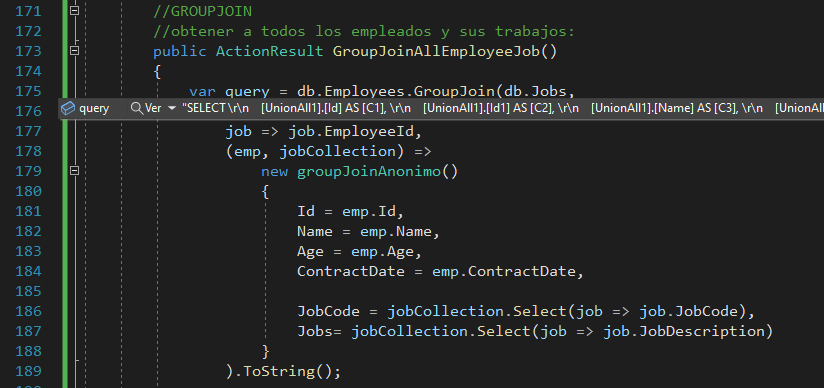
Y ejecutamos la aplicacion

**Texto

Descripción generada automáticamente**

**Nos posicionamos sobre la consulta:**

Y seleccionamos Ver en donde esta la lupa.

****

**Nos muestra el código Sql Generado:**

Texto

Descripción generada automáticamente

**Código Generado:**

SELECT

[UnionAll1].[Id] AS [C1],

[UnionAll1].[Id1] AS [C2],

[UnionAll1].[Name] AS [C3],

[UnionAll1].[Age] AS [C4],

[UnionAll1].[ContractDate] AS [C5],

[UnionAll1].[C1] AS [C6],

[UnionAll1].[JobCode] AS [C7],

[UnionAll1].[C2] AS [C8]

FROM (SELECT

CASE WHEN ([Extent2].[JobCode] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C1],

[Extent1].[Id] AS [Id],

[Extent1].[Id] AS [Id1],

[Extent1].[Name] AS [Name],

[Extent1].[Age] AS [Age],

[Extent1].[ContractDate] AS [ContractDate],

[Extent2].[JobCode] AS [JobCode],

CAST(NULL AS varchar(1)) AS [C2]

FROM [dbo].[Employees] AS [Extent1]

LEFT OUTER JOIN [dbo].[Jobs] AS [Extent2] ON [Extent1].[Id] = [Extent2].[EmployeeId]

UNION ALL

SELECT

2 AS [C1],

[Extent3].[Id] AS [Id],

[Extent3].[Id] AS [Id1],

[Extent3].[Name] AS [Name],

[Extent3].[Age] AS [Age],

[Extent3].[ContractDate] AS [ContractDate],

CAST(NULL AS int) AS [C2],

[Extent4].[JobDescription] AS [JobDescription]

FROM [dbo].[Employees] AS [Extent3]

INNER JOIN [dbo].[Jobs] AS [Extent4] ON [Extent3].[Id] = [Extent4].[EmployeeId]) AS [UnionAll1]

ORDER BY [UnionAll1].[Id1] ASC, [UnionAll1].[C1] ASC

* **Obtener a una persoan y su listado de trabajos**

**GroupJoinEmployeeJob():**

En esta caso para ver el código SQL como String devemos

Agregar un **Where** y luego convertir a to String:

**NOTA:**

En where retorna al avista un System.Data.Entity.Infrastructure.**DbQuery,**

Es diferente al **FirstOrDefault.**

//Obtener a una persoan y su listado de trabajos:

public ActionResult GroupJoinEmployeeJob()

{

var query = db.Employees.GroupJoin(db.Jobs, //la tabla que devolvera IEnumerable

emp => emp.Id, //la relación entre las llaves PK y FK

job => job.EmployeeId,

(emp, jobCollection) => //Lambda para emp y jobCollection indicamos que job sera de tipo collction para tnenerlo presente

new groupJoinAnonimo() //Objeto de tipo groupJoinAnonimo, la clase que definimos para mapear los campos de ambas tablas

{

Id = emp.Id,

Name = emp.Name,

Age = emp.Age,

ContractDate = emp.ContractDate,

//los campos de la tabla job son de tipo IEnumerable, los definimos asi enel objeto y clae anonimo

JobCode = jobCollection.Select(job => job.JobCode), //seleccina todos los registros

Jobs = jobCollection.Select(job => job.JobDescription)

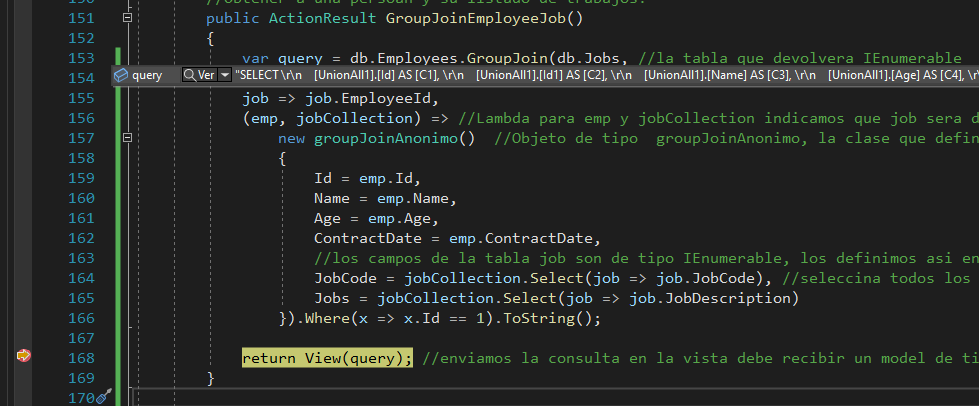
**}).Where(x => x.Id == 1).ToString();**

return View(query); //enviamos la consulta en la vista debe recibir un model de tipo groupJoinAnonimo,

}

**Colocamos un breakPoint:**

**Y entramos en ver:**

****

**Nos genera el código sql:**

SELECT

[UnionAll1].[Id] AS [C1],

[UnionAll1].[Id1] AS [C2],

[UnionAll1].[Name] AS [C3],

[UnionAll1].[Age] AS [C4],

[UnionAll1].[ContractDate] AS [C5],

[UnionAll1].[C1] AS [C6],

[UnionAll1].[JobCode] AS [C7],

[UnionAll1].[C2] AS [C8]

FROM (SELECT

CASE WHEN ([Extent2].[JobCode] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C1],

[Extent1].[Id] AS [Id],

[Extent1].[Id] AS [Id1],

[Extent1].[Name] AS [Name],

[Extent1].[Age] AS [Age],

[Extent1].[ContractDate] AS [ContractDate],

[Extent2].[JobCode] AS [JobCode],

CAST(NULL AS varchar(1)) AS [C2]

FROM [dbo].[Employees] AS [Extent1]

LEFT OUTER JOIN [dbo].[Jobs] AS [Extent2] ON [Extent1].[Id] = [Extent2].[EmployeeId]

WHERE 1 = [Extent1].[Id]

UNION ALL

SELECT

2 AS [C1],

[Extent3].[Id] AS [Id],

[Extent3].[Id] AS [Id1],

[Extent3].[Name] AS [Name],

[Extent3].[Age] AS [Age],

[Extent3].[ContractDate] AS [ContractDate],

CAST(NULL AS int) AS [C2],

[Extent4].[JobDescription] AS [JobDescription]

FROM [dbo].[Employees] AS [Extent3]

INNER JOIN [dbo].[Jobs] AS [Extent4] ON [Extent3].[Id] = [Extent4].[EmployeeId]

WHERE 1 = [Extent3].[Id]) AS [UnionAll1]

ORDER BY [UnionAll1].[Id1] ASC, [UnionAll1].[C1] ASC

Consultas Sql Personalizadas:

<https://www.youtube.com/watch?v=3jJQmBe5sZg&list=PL0kIvpOlieSNWR3YPSjh9P2p43SFnNBlB&index=55>

en las consultas personalizadas podemos enviar los datos obtenido de un query de una tabla de la base de datos, la podemos enviar a un modelo clase externa con los campos de indiquemos,

esto nos servirá también para generar la vista.

**GROUP BY:**

En este ejemplo hacemos una consulta GROUP BY a la tabla de la base de datos Employees

Para agrupar los hombres y mujeres y mostrar la cantidad de cada uno.

**Modelos externo:**

public class HombresMujers

{

public Sexo Sexo { get; set; }

public int Cantidad { get; set; }

}

**Modelo Enum:**

En este caso usamos el campo Sexo de tipo Enum.

public enum Sexo

{

Masculino = 1,

Femenino = 2,

}

**Controller:**

La clase externa HombresMujers nos servirá para almacenar los datos obtenidos de la consulta, y es la que usaremos para mostrar en la vista:

//CONCULTA PERSONZLIZADA:

//Cargar a un modelo no mapeado en al base de datos, la consulta de una tabla de la base de datos:

//Debemos crear el modelo no mapeado.

//GROUPBY

//usanso group by contar la cantidad de hombre y mujeres.

//creamo una clase HombresMujeres para almacenar la consulta

//creamos una clase Enum para el tipo Sexo que pueda tener valores de 1 y 2.

public ActionResult SqlPersonalizado()

{

var query = db.Database.SqlQuery<HombresMujers>(

@"SELECT Sexo, COUNT(\*) AS Cantidad

FROM dbo.Employees

GROUP BY Sexo").ToList();

return View(query);

}

**Vista:**

Usamos el modelo personalizado para mostrar los datos en la vista:

**@model IEnumerable<\_4\_EntityFramework.Models.SqlPeersonalizado.HombresMujers>**

@{

ViewBag.Title = "SqlPersonalizado";

}

<h2>SqlPersonalizado</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Sexo)

</th>

<th>

@Html.DisplayNameFor(model => model.Cantidad)

</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Sexo)

</td>

<td>

@Html.DisplayFor(modelItem => item.Cantidad)

</td>

</tr>

}

</table>