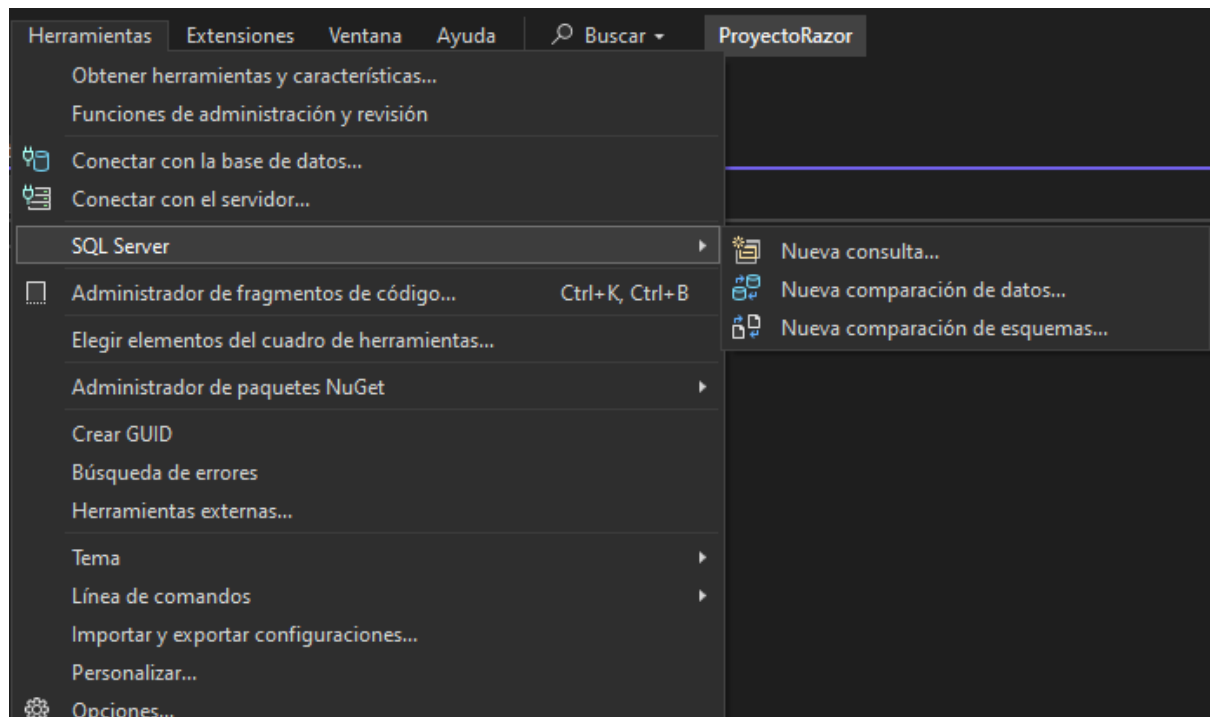


PROYECTO RAZOR - ALEJANDRO AFONSO BARBER



Lo primero que haremos será realizar el proceso de instalación y conexión con la base de datos, así que tras crear el proyecto, abriremos el menú de herramientas y pulsaremos en la opción SQL Server, y luego en Nueva consulta.

Conectar

History **Examinar**

🔍 Escriba aquí para filtrar la lista

- ▶ Local
- ▶ Red
- ▶ Azure

Nombre del servidor: (localdb)\MSSQLLocalDB

Autenticación: Autenticación de Windows ▼

Nombre del usuario: DESKTOP-CFHH8AP\aries

Contraseña:

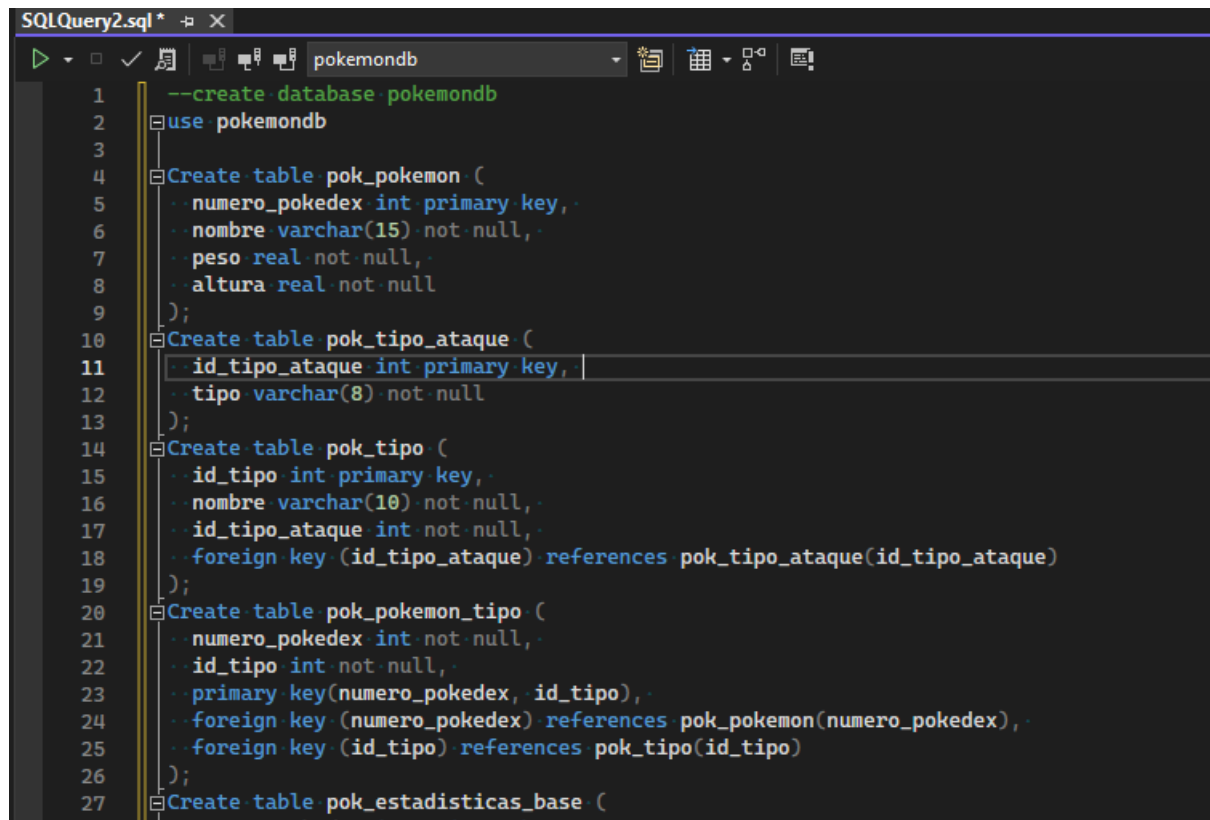
☐ Recordar contraseña

Nombre de la base de datos: <predeterminado> ▼

[Avanzadas...](#)

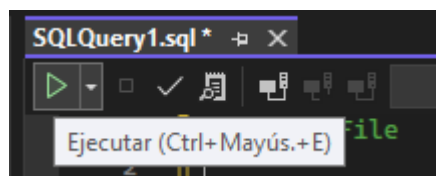
Conectar Cancelar

Cuando le demos, se nos abrirá una pequeña ventana en la que tendremos que seleccionar nuestra base de datos, en este caso, es una base de datos local. Usaremos Windows de modo de autenticación y dejaremos el nombre de la base de datos como predeterminado. Luego le daremos a conectar y se cerrará la ventana. Ahora se nos abrirá una pestaña en Visual Studio que será una SQL Query. El siguiente paso será pegar el script que contiene la base de datos con la que vamos a trabajar.

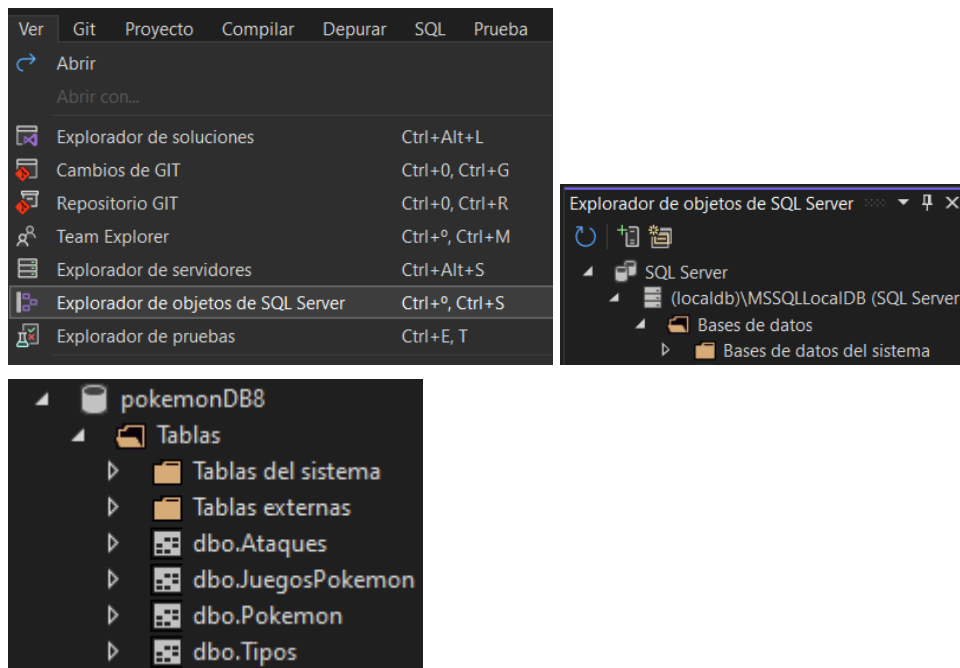


```
SQLQuery2.sql *
1  --create database pokemondb
2  use pokemondb
3
4  Create table pok_pokemon (
5      numero_pokedex int primary key,
6      nombre varchar(15) not null,
7      peso real not null,
8      altura real not null
9  );
10 Create table pok_tipo_ataque (
11     id_tipo_ataque int primary key,
12     tipo varchar(8) not null
13 );
14 Create table pok_tipo (
15     id_tipo int primary key,
16     nombre varchar(10) not null,
17     id_tipo_ataque int not null,
18     foreign key (id_tipo_ataque) references pok_tipo_ataque(id_tipo_ataque)
19 );
20 Create table pok_pokemon_tipo (
21     numero_pokedex int not null,
22     id_tipo int not null,
23     primary key(numero_pokedex, id_tipo),
24     foreign key (numero_pokedex) references pok_pokemon(numero_pokedex),
25     foreign key (id_tipo) references pok_tipo(id_tipo)
26 );
27 Create table pok_estadisticas_base (
```

Creación de la base de datos.

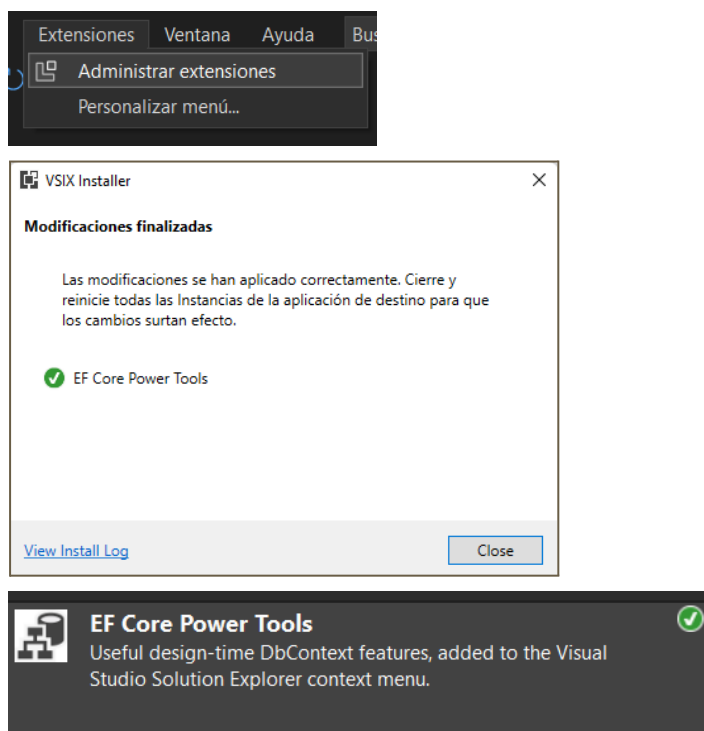


Pulsamos el botón de Ejecutar una vez hayamos puesto el script para crear la base de datos.



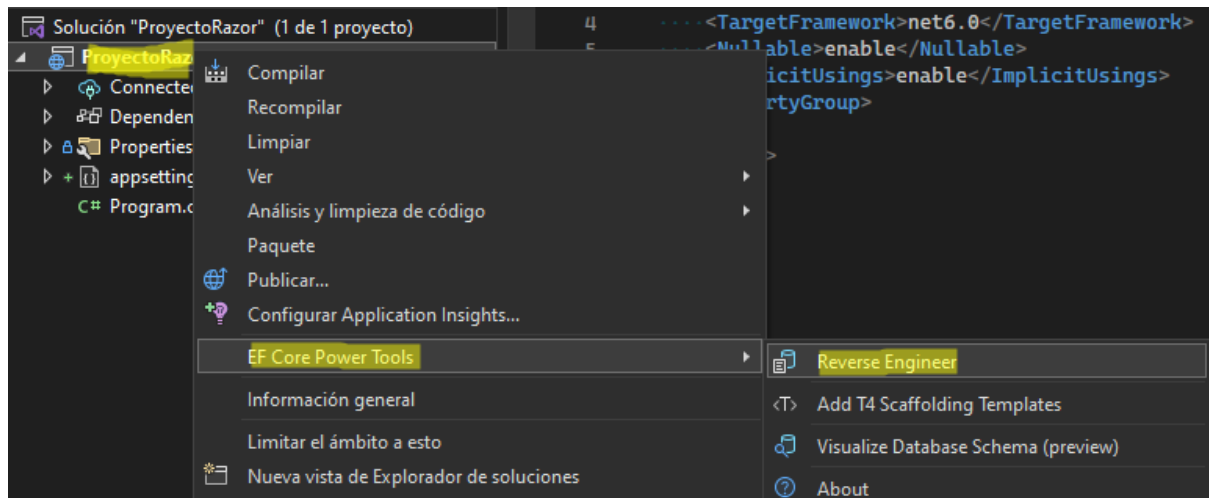
Desde el menú Ver, podemos acceder al explorador de objetos de SQL y ver toda la información que tenemos en la base de datos, confirmando que las tablas han sido creado correctamente.

Nuestro siguiente paso será descargar la extensión y enlazarla con nuestro proyecto. Para instalarla correctamente tendremos que reiniciar el programa.

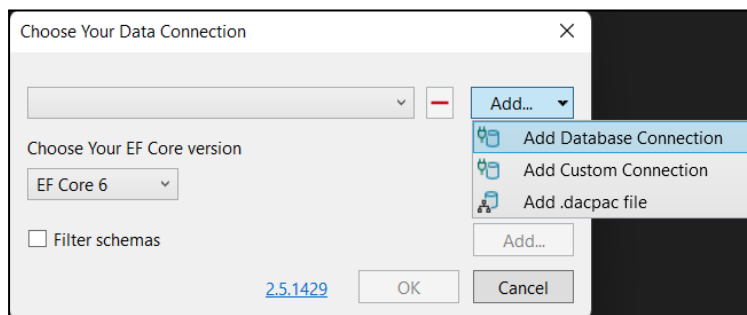


Una vez hayamos instalado la extensión, la usaremos para hacer el enlace entre la base de datos y nuestro proyecto, para ello seguiremos estos pasos.

Haremos click derecho en nuestro proyecto, y haremos click en los botones resaltados.



Se nos abrirá la siguiente ventana.



Le damos a la opción de añadir conexión a la base de datos.

Propiedades de la conexión

Especifique la información para conectarse al origen de datos seleccionado o haga clic en "Cambiar" para elegir otro origen o proveedor de datos.

Origen de datos:
Microsoft SQL Server (Microsoft SqlClient) Cambiar...

Nombre del servidor:
(localdb)\MSSQLLocalDB Actualizar

Conexión con el servidor

Autenticación: Autenticación de Windows

Nombre de usuario:

Contraseña:

☐ Guardar mi contraseña

Establecer conexión con una base de datos

☒ Seleccionar o escribir el nombre de la base de datos:
pokemondb

☐ Adjuntar un archivo de base de datos:
 Examinar...

Nombre lógico:

Avanzadas...

Probar conexión Aceptar Cancelar

Escribimos el nombre del servidor local y de la base de datos que vamos a usar. En este caso, pokemondb.

Choose Your Data Connection

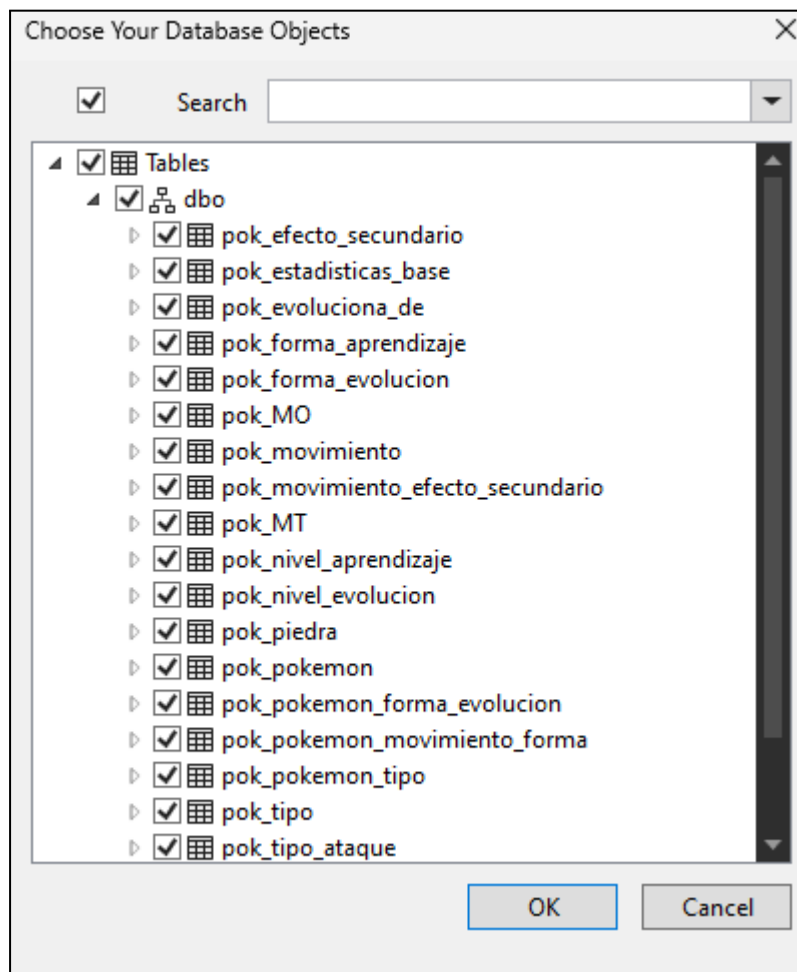
(localdb)\MSSQLLocalDB.pokemondb + Add...

Choose Your EF Core version
EF Core 6

☐ Filter schemas Add...

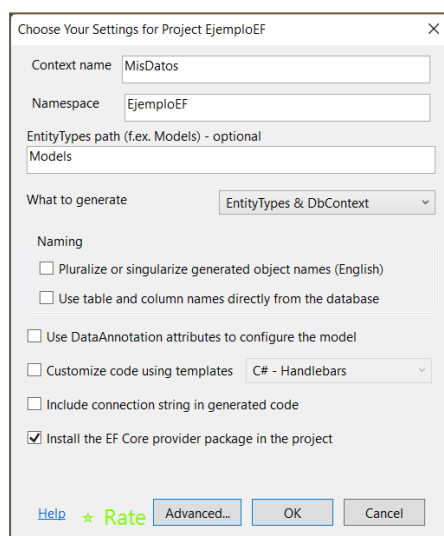
[2.5.1429](#) OK Cancel

Seleccionamos las tablas que queremos enlazar con nuestro proyecto.

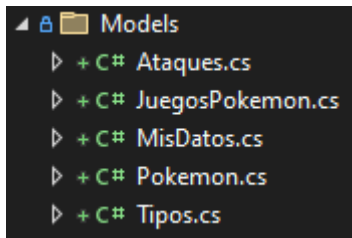


Captura antigua, ahora estamos usando otra base de datos con menos tablas.

Creamos la clase de contexto



Se nos creará en Models (nombre que hemos elegido nosotros) las clases que estarán enlazadas con las tablas de la base de datos.



```
namespace Proyecto_Razor.Models
{
    public partial class MisDatos : DbContext
    {
        public MisDatos()
        {
        }

        public MisDatos(DbContextOptions<MisDatos> options)
            : base(options)
        {
        }

        public virtual DbSet<Ataques> Ataques { get; set; }
        public virtual DbSet<JuegosPokemon> JuegosPokemon { get; set; }
        public virtual DbSet<Pokemon> Pokemon { get; set; }
        public virtual DbSet<Tipos> Tipos { get; set; }
    }
}
```

El siguiente paso será introducir en los ajustes una serie de parámetros, donde pondremos el nombre del servidor local y el nombre de la base de datos, entre otras opciones y configuraciones.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MiDataBase": "Data Source=(localdb)\\MSSQLLocalDB;Database=pokemondb;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False"
  }
}
```

También en Program.cs tenemos que especificar la clase de contexto y el servidor de la base de datos.


```
Program.cs * X
EjemploEF
1 using Microsoft.EntityFrameworkCore;
2 using EjemploEF.Models;
3
4 var builder = WebApplication.CreateBuilder(args);
5 builder.Services.AddRazorPages();
6
7 builder.Services.AddDbContext<MisDatos>(options =>
8     options.UseSqlServer(builder.Configuration.GetConnectionString("MiDataBase") ?? throw new InvalidOperationException
9         ("Connection string 'MiDataBase' not found.")));
10
11 var app = builder.Build();
12 app.MapRazorPages();
13
14 app.Run();
15
```

Incluiremos los datos en cada página donde queramos usar la información de la base de datos.

```
public readonly MisDatos Datos;
[ActivatorUtilitiesConstructor]
0 referencias
public SelecciónModel(MisDatos datos)
{
    Datos = datos;
}
```

Con esto ya hemos hecho la conexión a la base de datos, hemos generado las clases y hemos dispuesto la lectura de estos datos en todas las páginas de nuestra página que necesiten acceder. A continuación especificaremos como hemos hecho esas consultas en los diferentes ejercicios de la página.

Ejercicio 1.

En el primer ejercicio le daremos al usuario la opción de elegir una o varias opciones, y posteriormente, una segunda opción única a elegir entre dos. Usando condicionales if / else if, sacamos diferentes datos y los mostramos.

```
if (Request.Form.ContainsKey("1"))
{
    if (Request.Form["evolution"] == "evolution")
    {
        List<string> pokemonTipofuegoEvolucionados = (from pokemon in Model.Datos.Pokemon
        join tipo in Model.Datos.Tipos on pokemon.IdTipo equals tipo.IdTipo
        where tipo.Nombre == "fuego" && pokemon.EsEvolucion
        select pokemon.Nombre).ToList();

        foreach (string nombre in pokemonTipofuegoEvolucionados)
        {
            <p>@nombre</p>
        }
    }

    else if (Request.Form["evolution"] == "noevolution")
    {
        List<string> pokemonTipofuegoNoEvolucionados = (from pokemon in Model.Datos.Pokemon
        join tipo in Model.Datos.Tipos on pokemon.IdTipo equals tipo.IdTipo
        where tipo.Nombre == "fuego" && !pokemon.EsEvolucion
        select pokemon.Nombre).ToList();

        foreach (string nombre in pokemonTipofuegoNoEvolucionados)
        {
            <p>@nombre</p>
        }
    }
}

if (Request.Form.ContainsKey("2"))
{
}
```

En el primer IF validamos si el usuario ha presionado esa opción, y en caso de que lo haya hecho, validamos en el segundo IF cuál de las dos opciones ha seleccionado el usuario para saber que datos exactamente mostrar, donde haremos una consulta diferente en cada caso y guardaremos los resultados en una lista que mostraremos posteriormente. Los siguientes IF del ejercicio para validar las tres posibles opciones que el usuario puede elegir (todas a la vez o con la combinación que sea) para mostrar las otras posibles listas una debajo de la otra.

Ejercicio 2.

En este ejercicio hacemos un formulario en el que el usuario introduce una fecha, luego esa fecha la recogemos en otra página (donde mostraremos los datos) y la guardamos en una variable, ya que esta vez los datos que mostraremos serán de tres en tres y tendremos que poder movernos entre ellos con unas botones tanto hacia delante como hacia atrás. Como ahora vamos a mostrar varios datos, en vez de guardarlo en una lista de Strings como antes, hemos creado un objeto y una lista de esos objetos, y con esta consulta lo guardamos tras hacer la validación.

```
// Obtener los juegos válidos según la fecha y aplicar la paginación
var juegosFiltrados = _datos.JuegosPokemon
    .Where(juegoPokemon => juegoPokemon.FechaLanzamiento < fechaSeleccionada)
    .Select(juegoPokemon => new Juego
    {
        Nombre = juegoPokemon.Nombre,
        Plataforma = juegoPokemon.Plataforma,
        FechaLanzamiento = juegoPokemon.FechaLanzamiento.ToString("dd/MM/yyyy")
    });
```

```
JuegosValidos = juegosFiltrados
    .Skip((pagina - 1) * PageSize)
    .Take(PageSize)
    .ToList();
```

En esta segunda captura establecemos que los Juegos Validos son solo los que se mostrarán en la página, ya que como expliqué antes, se mostrarán de tres en tres y con la opción de movernos entre todos los resultados que saquemos de la base de datos.

Ejercicio 3 y 4.

Ambos ejercicios son similares y por eso están agrupados. En ellos les pedimos al usuario seleccionar una opción de una combobox, y en función a la opción elegida, les aparecerá una segunda combobox que contendrá unos datos únicos en relación a esa opción.

Para rellenar el segundo combobox, usamos una consulta en la que le pasamos una variable (del primer combobox) y así saber que datos necesitamos.

```
List<string> pokemonsTipoSeleccionado = (from pokemon in Model.Datos.Pokemon
.....join tipo in Model.Datos.Tipos on pokemon.IdTipo equals tipo.IdTipo
.....where tipo.Nombre == tipoSeleccionado
.....select pokemon.Nombre).ToList();
```

Y luego con ambas variables (ambos combobox) definidas, podemos hacer la consulta final que mostraremos al usuario:

```
List<string> ataquesPokemon = (from pokemon in Model.Datos.Pokemon
.....join ataques in Model.Datos.Ataques on pokemon.NumeroPokedex equals ataques.IdPokemon
.....where pokemon.Nombre == pokemonSeleccionado
.....select ataques.Nombre).ToList();
if (ataquesPokemon.Count > 0)
{
    <h4>Ataques:</h4>
    <ul>
    @foreach (string ataque in ataquesPokemon)
    {
        <li>@ataque</li>
    }
    </ul>
}
else
{
    Console.WriteLine("No se encontraron ataques en la tabla 'Ataques'.");
}
```

El ejercicio 4 es bastante similar, solo que en la consulta final, queremos mostrar varios datos, así que de nuevo volvemos a crear un Objeto que contenga los atributos que queremos mostrar y lo rellenamos de esta manera:

```
public void RellenarPoke(string poke)
{
    PokemonSeleccionado = Datos.Pokemon
    .Where(pokemon => pokemon.Nombre == poke)
    .Select(pokemon => new Pokemon
    {
        NumeroPokedex = pokemon.NumeroPokedex,
        Nombre = pokemon.Nombre,
        Peso = pokemon.Peso,
        Altura = pokemon.Altura
    })
    .ToList();
}
```

Y luego lo mostramos al usuario.

Ejercicio 5.

En este ejercicio le pedimos al usuario introducir un número, luego lo guardaremos y validaremos si es positivo o no, y en caso de que sea positivo, usaremos ese peso para hacer una consulta y mostrar los datos que sean iguales o superiores a ese número.

```
decimal peso = Convert.ToDecimal(Request.Form["Peso"]);

if (peso > 0)
{
    List<string> listaPokemon = (from pokemon in Model.Datos.Pokemon
                                ..... where pokemon.Peso >= peso
                                ..... select pokemon.Nombre).ToList();

    @foreach (string poke in listaPokemon)
    {
        <ul>
        .....<li>@poke</li>
        .....</ul>
    }
}
else
{
    <p>Introduce un número positivo para poder mostrar los Pokémons que superen ese peso.</p>
}
```

Ejercicio 6.

En este ejercicio, de forma parecida a lo que hicimos en el ejercicio 4, vamos a pedirle, esta por escrito, un nombre al usuario, validaremos que ese nombre esté dentro de una tabla en concreto de la base de datos, y creando un objeto con los atributos de esa tabla, mostraremos los datos.

```
if (!string.IsNullOrEmpty(nombre))
{
    string? nombrePokemon = (from pokemon in Model.Datos.Pokemon
                              where pokemon.Nombre == nombre
                              select pokemon.Nombre).SingleOrDefault();
    if (!string.IsNullOrEmpty(nombrePokemon))
    {
        Model.RellenarPoke(nombre);
        @foreach (var poke in Model.PokemonSeleccionado)
        {
            <table class="table">
            <tr>
            <th>Número Pokedex</th>
            <th>Nombre</th>
            <th>Peso</th>
            <th>Altura</th>
            </tr>
            <tr>
            <td>@poke.NumeroPokedex</td>
            <td>@poke.Nombre</td>
            <td>@poke.Peso</td>
            <td>@poke.Altura</td>
            </tr>
            </table>
        }
    }
}
```

Aquí primero validamos que el dato introducido exista y solo si existe, rellenamos el objeto con los valores de ese dato en concreto, y luego lo mostramos.

