



# .Net

## Teoría 11

# Una introducción rápida a LINQ

## LINQ

En la práctica de delegados, se pidió extender al tipo `int[]` con los métodos **Seleccionar** y **Donde**. Por ejemplo si `v` es un vector de enteros, la expresión

```
v.Donde(n => n % 2 == 1).Seleccionar(n => n * n)
```

debía devolver un nuevo vector con todos los elementos impares de `v` elevados al cuadrado.

Los delegados como parámetros aportan muchísima versatilidad.



# LINQ

Si en lugar de extender `int[]` extendiésemos `IEnumerable<T>` sería mucho más provechoso porque afectaría a todas las colecciones que implementan esta interfaz.

Afortunadamente no tenemos que hacerlo  
**LINQ** ya lo hace por nosotros

Veamos algunos ejemplos ...





## Crear una aplicación de consola llamada LINQ



1. Abrir una terminal del sistema operativo
2. Cambiar a la carpeta `proyectosDotnet`
3. Crear la aplicación de consola `LINQ`
4. Abrir code en la carpeta `LINQ`



Codificar Program.cs de la siguiente manera y ejecutar



```
int[] vector = new int[] { 1, 2, 3, 4, 5 };  
IEnumerable<int> secuencia = vector.Select(n => n * 3);  
Mostrar(secuencia);
```

```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```

```
int[] vector = new int[] { 1, 2, 3, 4, 5 };  
IEnumerable<int> secuencia = vector.Select(n => n * 3);  
Mostrar(secuencia);
```

`T[]` implementa la interfaz  
`IEnumerable<T>`

```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```

Obtenemos los  
elementos de `vector`  
multiplicados por 3



3 6 9 12 15



Completar la línea que falta para que la salida por consola sea la que se indica



→ `List<string> lista = new List<string>() { "uno", "dos", "tres" };  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
Mostrar(secuencia);`

```
void Mostrar<T>(IEnumerable<T> secuencia)
{
    foreach (T elemento in secuencia)
    {
        Console.Write(elemento + " ");
    }
    Console.WriteLine();
}
```

A black computer monitor with a silver stand. The screen is black and displays the text "(UNO) (DOS) (TRES)" in white. A small green light is visible at the bottom right of the monitor frame.

(UNO) (DOS) (TRES)





## Posible solución

→ 

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);
```

```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```

(UNO) (DOS) (TRES)



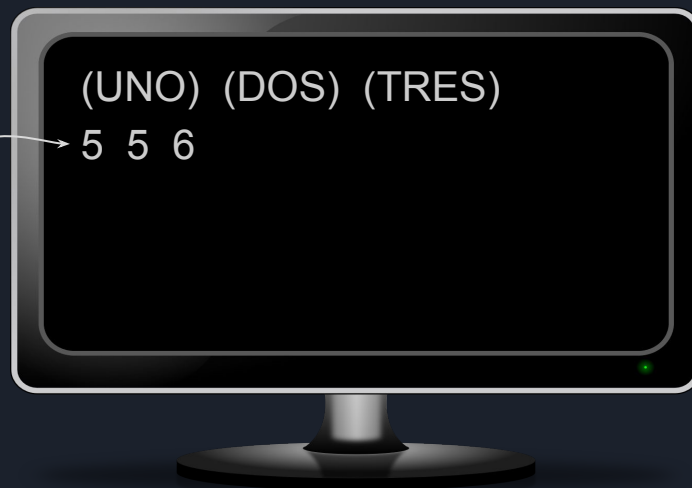
Completar la línea que falta para que la salida por consola sea la que se indica



```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
→ IEnumerable<int> secuencia2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
Mostrar(secuencia2);
```

```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```

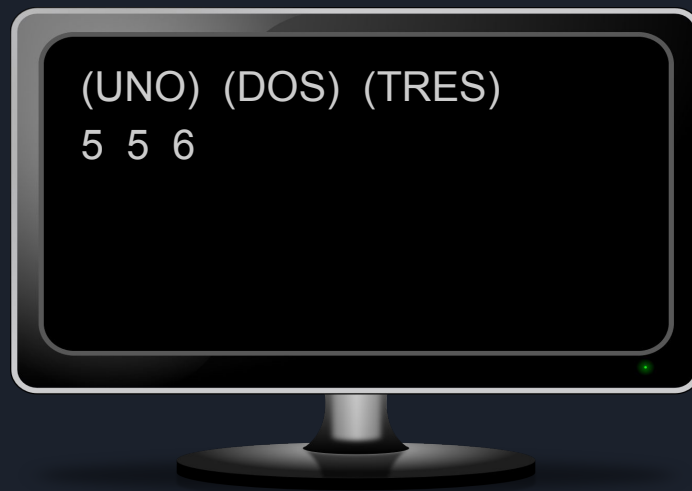
Longitud de los  
strings de secuencia



## Posible solución

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);
```


```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```



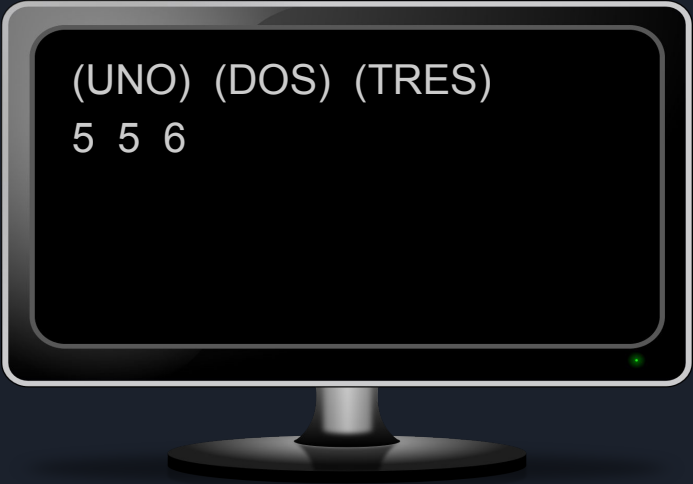
(UNO) (DOS) (TRES)  
5 5 6

## Posible solución

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);
```



Observar que `secuencia2` es de un tipo distinto a `secuencia` (el método `Select` es un método genérico, se está haciendo inferencia de parámetros de tipos a partir del argumento, en este caso de tipo `Func<string,int>`), por lo tanto se está invocando a `Select<string,int>`



```
(UNO) (DOS) (TRES)  
5 5 6
```



## Posible solución

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);
```

Este es el encabezado del método de extensión `Select` definido en la clase estática `System.Linq.Enumerable`

```
public static IEnumerable<TResult> Select<TSource, TResult>(  
    this IEnumerable<TSource> source, Func<TSource, TResult> selector)
```

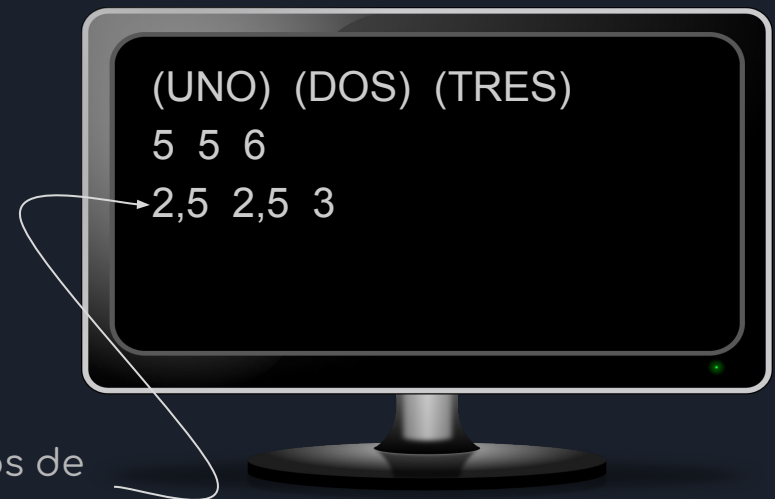


Completar la línea que falta para que la salida por consola sea la que se indica



```
List<string> lista = new List<string>() { "uno", "dos", "tres" };
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");
Mostrar(secuencia);
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);
Mostrar(secuencia2);
→ IEnumerable<double> secuencia3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxx
Mostrar(secuencia3);
```

```
void Mostrar<T>(IEnumerable<T> secuencia)
{
    foreach (T elemento in secuencia)
    {
        Console.Write(elemento + " ");
    }
    Console.WriteLine();
}
```

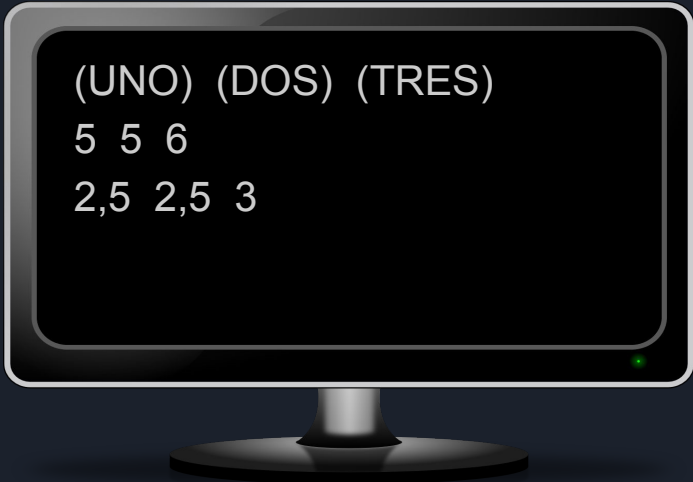


la mitad de los elementos de  
la secuencia anterior

## Posible solución

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);  
IEnumerable<double> secuencia3 = secuencia2.Select(n => n / 2.0);  
Mostrar(secuencia3);
```

```
void Mostrar<T>(IEnumerable<T> secuencia)  
{  
    foreach (T elemento in secuencia)  
    {  
        Console.Write(elemento + " ");  
    }  
    Console.WriteLine();  
}
```



(UNO) (DOS) (TRES)  
5 5 6  
2,5 2,5 3



## Posible solución

```
List<string> lista = new List<string>() { "uno", "dos", "tres" };  
IEnumerable<string> secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
IEnumerable<int> secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);  
IEnumerable<double> secuencia3 = secuencia2.Select(n => n / 2.0);  
Mostrar(secuencia3);
```



Dividimos por 2.0 los elementos enteros de secuencia2 obteniendo un `IEnumerable<double>`  
Los argumentos de tipo del método `Select` se infieren por medio del argumento que en este caso es de tipo `Func<int,double>`

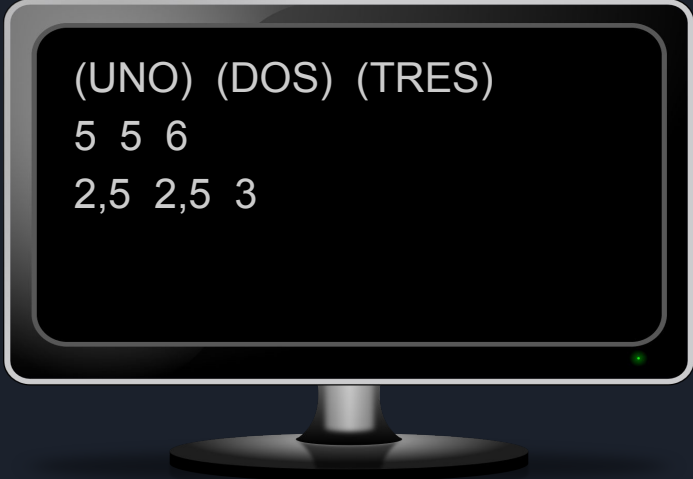
(UNO) (DOS) (TRES)  
5 5 6  
2,5 2,5 3



## Posible solución

```
var lista = new List<string>() { "uno", "dos", "tres" };  
var secuencia = lista.Select(st => "(" + st.ToUpper() + ")");  
Mostrar(secuencia);  
var secuencia2 = secuencia.Select(st => st.Length);  
Mostrar(secuencia2);  
var secuencia3 = secuencia2.Select(n => n / 2.0);  
Mostrar(secuencia3);
```

Es muy común utilizar LINQ con inferencia de tipos (palabra clave `var`) para simplificar la escritura y lectura del código




(UNO)	(DOS)	(TRES)
5	5	6
2,5	2,5	3

# Interfaz Fluida de LINQ

```
var lista = new List<string>() { "uno", "dos", "tres" };  
var secuencia = lista.Select(st => "(" + st.ToUpper() + ")")  
                      .Select(st => st.Length)  
                      .Select(n => n / 2.0);  
Mostrar(secuencia);
```

Si sólo nos interesa el último resultado es muy común utilizar la interfaz fluida (**fluent API**) de LINQ



2,5 2,5 3

```
var numeros = new List<int>() { 1, 10, 7, 3, 11 };  
Mostrar(numeros);
```

Además de `Select`, LINQ  
provee muchos otros  
métodos de extensión:  
`Where`, `Reverse`, `OrderBy`,  
`Sum`, `Average` son  
sólo alguno de ellos

A computer monitor with a black bezel and a silver stand. The screen is black, and the numbers '1 10 7 3 11' are displayed in a blue box at the top left of the screen. A white arrow points from the 'Mostrar(numeros);' line of code to the blue box.

1 10 7 3 11

```
var numeros = new List<int>() { 1, 10, 7, 3, 11 };  
Mostrar(numeros);  
var mayores6 = numeros.Where(n => n > 6);  
Mostrar(mayores6);
```

Además de Select, LINQ  
provee muchos otros  
métodos de extensión:  
Where, Reverse, OrderBy,  
Sum, Average son  
sólo alguno de ellos

A computer monitor with a black frame and a silver base. The screen displays the output of the LINQ query. The first line shows the original list: "1 10 7 3 11". The second line shows the filtered list: "10 7 11", which is highlighted with a blue background. A white arrow points from the "Mostrar(mayores6);" line of code to the second line of the monitor's output.

1 10 7 3 11  
10 7 11

```
var numeros = new List<int>() { 1, 10, 7, 3, 11 };  
Mostrar(numeros);  
var mayores6 = numeros.Where(n => n > 6);  
Mostrar(mayores6);  
var reverso = mayores6.Reverse();  
Mostrar(reverso);
```

Además de Select, LINQ  
provee muchos otros  
métodos de extensión:  
Where, Reverse, OrderBy,  
Sum, Average son  
sólo alguno de ellos

A computer monitor with a black frame and a silver stand. The screen displays the output of the LINQ query. The first line shows the original list: '1 10 7 3 11'. The second line shows the filtered list: '10 7 11'. The third line shows the reversed filtered list: '11 7 10', which is highlighted with a blue background. A white arrow points from the 'Mostrar(reverso);' line of code to the third line of output.

1	10	7	3	11
10	7	11		
11	7	10		

```
var numeros = new List<int>() { 1, 10, 7, 3, 11 };  
Mostrar(numeros);  
var mayores6 = numeros.Where(n => n > 6);  
Mostrar(mayores6);  
var reverso = mayores6.Reverse();  
Mostrar(reverso);  
var ordenados = reverso.OrderBy(n => n);  
Mostrar(ordenados);
```

Además de Select, LINQ  
provee muchos otros  
métodos de extensión:  
Where, Reverse, OrderBy,  
Sum, Average son  
sólo alguno de ellos

A computer monitor displaying the output of the LINQ query. The output is a list of numbers: 1 10 7 3 11, 10 7 11, 11 7 10, and 7 10 11. The last line, 7 10 11, is highlighted in blue. A white arrow points from the 'var ordenados' line in the code block to the highlighted line on the monitor.

1	10	7	3	11
10	7	11		
11	7	10		
7	10	11		

```
var numeros = new List<int>() { 1, 10, 7, 3, 11 };  
Mostrar(numeros);  
var mayores6 = numeros.Where(n => n > 6);  
Mostrar(mayores6);  
var reverso = mayores6.Reverse();  
Mostrar(reverso);  
var ordenados = reverso.OrderBy(n => n);  
Mostrar(ordenados);  
var suma = ordenados.Sum();  
var promedio = ordenados.Average();  
Console.WriteLine($"suma: {suma} promedio:{promedio:0.00}");
```

Además de Select, LINQ  
provee muchos otros  
métodos de extensión:  
Where, Reverse, OrderBy,  
Sum, Average son  
sólo alguno de ellos



```
1 10 7 3 11  
10 7 11  
11 7 10  
7 10 11
```

```
suma: 28 promedio:9,33
```

//calculamos la suma de los primeros  
//20 cuadrados: 1 + 4 + 9 + ... + 400

```
var suma = Enumerable.Range(1, 20)           // 1, 2, 3, ..., 20
                        .Select(n => n * n)    // 1, 4, 9, ... 400
                        .Sum();                // 1 + 4 + 9 + ... + 400
Console.WriteLine($"Resultado: {suma}");
```

Primer valor del rango

Cantidad de elementos  
del rango



El método estático  
Range(start,count) de la  
clase System.Linq.Enumerable  
devuelve un IEnumerable<int>  
con la secuencia de enteros  
comenzando por start  
y con count elementos

Resultado: 2870





Codificar la clase  
**Persona** que vamos a  
utilizar para mostrar  
más ejemplos de las  
facilidades que brinda  
**LINQ**

```
class Persona
{
    public string Nombre { get; private set; }
    public int Edad { get; private set; }
    public string Pais { get; private set; }
    public Persona(string nombre, int edad, string pais)
    {
        Nombre = nombre;
        Edad = edad;
        Pais = pais;
    }
    public override string ToString()
    {
        return $"{Nombre} ({Edad} años) {Pais.Substring(0, 3)}.";
    }

    // vamos a hardcodear una lista de personas
    // que usaremos en los siguientes ejemplos
    // para ello definimos el siguiente método estático
    public static List<Persona> GetLista()
    {
        return new List<Persona>() {
            new Persona("Pablo", 15, "Argentina"),
            new Persona("Juan", 55, "Argentina"),
            new Persona("José", 9, "Uruguay"),
            new Persona("María", 33, "Uruguay"),
            new Persona("Lucía", 16, "Perú"),
        };
    }
}
```

Copiar el código del archivo  
11\_Teoria\_codigo.txt



Completar el código para que la salida por consola sea la indicada (mayores de 18)



```
var personas = Persona.GetLista();  
personas.ForEach(p => Console.WriteLine(p)); // lista todas las personas  
Console.WriteLine();  
. . .
```

Listar las personas  
mayores de edad

Pablo (15 años) Arg.  
Juan (55 años) Arg.  
José (9 años) Uru.  
María (33 años) Uru.  
Lucía (16 años) Per.

[ Juan (55 años) Arg.  
María (33 años) Uru.



## Posible solución

```
var personas = Persona.GetLista();  
personas.ForEach(p => Console.WriteLine(p)); // lista todas las personas  
Console.WriteLine();  
personas.Where(p => p.Edad >= 18) // un IEnumerable<Persona> no tiene método Foreach  
    .ToList() // lo convierte en un List<Persona>  
    .ForEach(p => Console.WriteLine(p)); // lista personas mayores de edad
```

Pablo (15 años) Arg.  
Juan (55 años) Arg.  
José (9 años) Uru.  
María (33 años) Uru.  
Lucía (16 años) Per.

Juan (55 años) Arg.  
María (33 años) Uru.

```
var personas = Persona.GetLista();  
  
personas.OrderBy(p => p.Edad)  
    .Select(p => new { Nombre = p.Nombre, Condición = p.Edad < 18 ? "Menor" : "Mayor" })  
    .ToList()  
    .ForEach(obj => Console.WriteLine(obj));
```



También es común  
devolver tipos anónimos.  
En este caso el método  
Select devuelve un  
IEnumerable de un tipo  
anónimo que tiene las  
propiedades  
Nombre y Condición

A black computer monitor with a silver stand, displaying the output of the LINQ query. The output is a list of five anonymous objects, each with 'Nombre' and 'Condición' properties.

```
{ Nombre = José, Condición = Menor }  
{ Nombre = Pablo, Condición = Menor }  
{ Nombre = Lucía, Condición = Menor }  
{ Nombre = María, Condición = Mayor }  
{ Nombre = Juan, Condición = Mayor }
```

```
var personas = Persona.GetLista();  
Console.WriteLine($"Primero: {personas.First()}");  
Console.WriteLine($"Último: {personas.Last()}");  
Console.WriteLine($"Edad máxima: {personas.Max(p => p.Edad)}");  
Console.WriteLine($"Edad mínima: {personas.Min(p => p.Edad)}");  
Console.WriteLine($"Son todos mayores? {personas.All(p => p.Edad >= 18)}");  
Console.WriteLine($"Hay algún mayor? {personas.Any(p => p.Edad >= 18)}");
```



First, Last, Max, Min,  
All, Any son algunos  
métodos más que  
brinda LINQ

A black computer monitor with a silver stand, displaying the output of the LINQ code. The text is white on a black background.

Primero: Pablo (15 años) Arg.  
Último: Lucía (16 años) Per.  
Edad máxima: 55  
Edad mínima: 9  
Son todos mayores? False  
Hay algún mayor? True

```
var personas = Persona.GetLista();  
var grupos = personas.GroupBy(p => p.Pais);  
foreach (var grup in grupos)  
{  
    Console.WriteLine($"{grup.Key} ({grup.Count()})");  
    foreach (var p in grup)  
    {  
        Console.WriteLine("    " + p.Nombre);  
    }  
}
```

Agrupamos por país.  
grupos es un IEnumerable de  
IGrouping<string,Persona>

Cada grup representa un  
grupo de personas junto a  
su clave de agrupación.  
grup es de tipo  
IGrouping<string,Persona>,  
esta interfaz hereda de  
IEnumerable<Persona>



LINQ Nos permite  
agrupar fácilmente  
por algún criterio

A computer monitor with a black bezel and a silver stand. The screen displays a list of names grouped by country. The text on the screen is: Argentina (2), Juan, Pablo, Uruguay (2), María, José, Perú (1), Lucía.

```
Argentina (2)  
    Juan  
    Pablo  
Uruguay (2)  
    María  
    José  
Perú (1)  
    Lucía
```

```
var personas = Persona.GetLista();
personas.GroupBy(p => p.Pais)
    .ToList()
    .ForEach(grup =>
    {
        Console.WriteLine($"{grup.Key} ({grup.Count()})");
        grup.ToList().ForEach(p => Console.WriteLine("  " + p.Nombre));
    });
```

También podemos  
hacerlo de esta  
manera, evitando  
la estructura de  
control foreach



Argentina (2)

Juan

Pablo

Uruguay (2)

María

José

Perú (1)

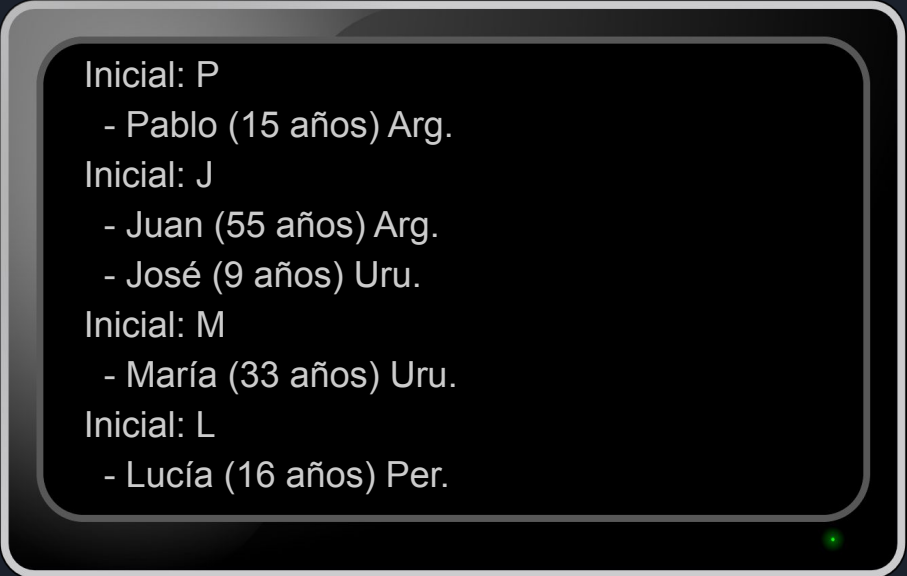
Lucía



# Utilizar GroupBy para listar las personas agrupadas por la inicial de su nombre



```
var personas = Persona.GetLista();  
var agrupadas = personas.GroupBy(xxxxxxxxxxxxxxxxxx);  
foreach(var grupo in agrupadas)  
    . . .
```



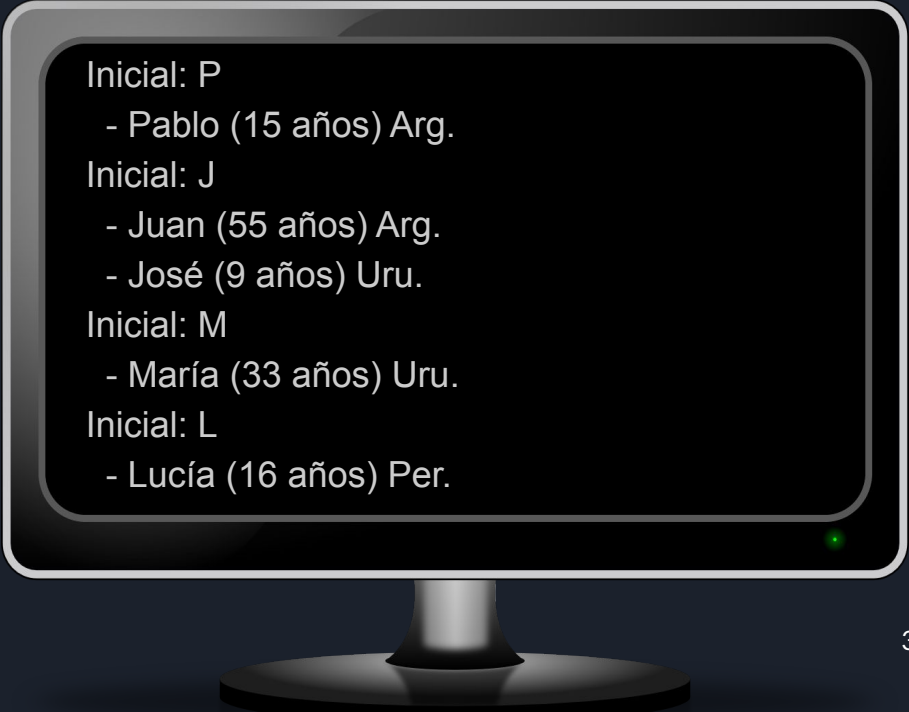
```
Inicial: P  
- Pablo (15 años) Arg.  
Inicial: J  
- Juan (55 años) Arg.  
- José (9 años) Uru.  
Inicial: M  
- María (33 años) Uru.  
Inicial: L  
- Lucía (16 años) Per.
```





## Posible solución

```
var personas = Persona.GetLista();  
var agrupadas = personas.GroupBy(p => p.Nombre[0]);  
foreach(var grupo in agrupadas)  
{  
    Console.WriteLine($"Inicial: {grupo.Key}");  
    grupo.ToList().ForEach(p => Console.WriteLine("  - " + p));  
}
```



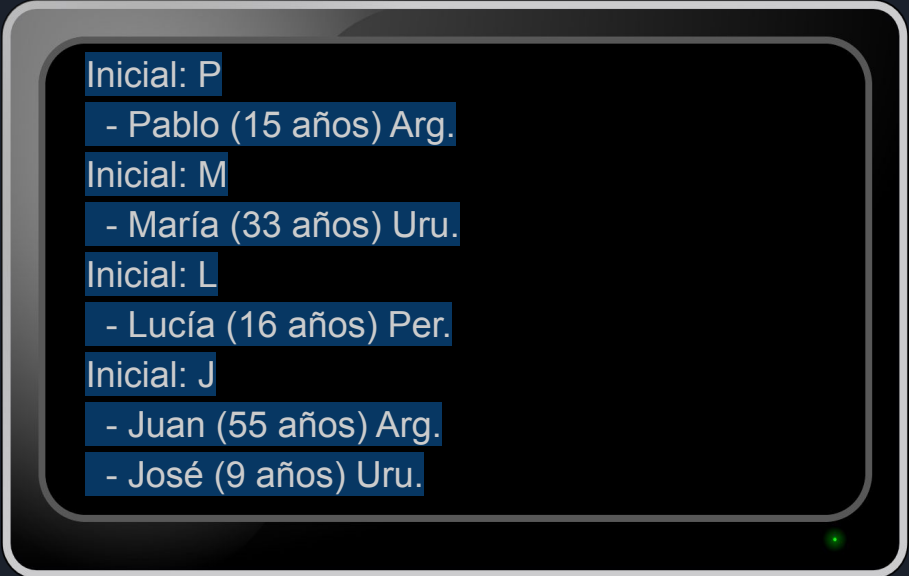
```
Inicial: P  
  - Pablo (15 años) Arg.  
Inicial: J  
  - Juan (55 años) Arg.  
  - José (9 años) Uru.  
Inicial: M  
  - María (33 años) Uru.  
Inicial: L  
  - Lucía (16 años) Per.
```



## Ordenar el listado por inicial de manera descendente



```
var personas = Persona.GetLista();  
var agrupadas = personas.GroupBy(p => p.Nombre[0]).xxxxxxxxxxxxxxxxxx;  
foreach(var grupo in agrupadas)  
{  
    Console.WriteLine($"Inicial: {grupo.Key}");  
    grupo.ToList().ForEach(p => Console.WriteLine(" - " + p));  
}
```

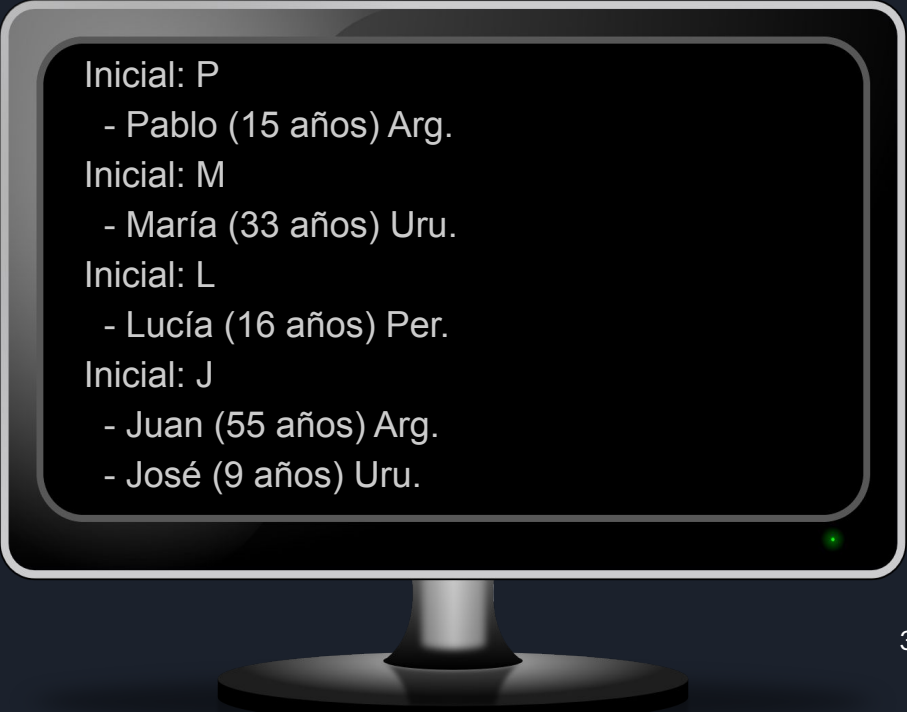


```
Inicial: P  
- Pablo (15 años) Arg.  
Inicial: M  
- María (33 años) Uru.  
Inicial: L  
- Lucía (16 años) Per.  
Inicial: J  
- Juan (55 años) Arg.  
- José (9 años) Uru.
```



## Posible solución

```
var personas = Persona.GetLista();  
var agrupadas = personas.GroupBy(p => p.Nombre[0]).OrderByDescending(g=>g.Key);  
foreach(var grupo in agrupadas)  
{  
    Console.WriteLine($"Inicial: {grupo.Key}");  
    grupo.ToList().ForEach(p => Console.WriteLine("    - " + p));  
}
```



```
Inicial: P  
    - Pablo (15 años) Arg.  
Inicial: M  
    - María (33 años) Uru.  
Inicial: L  
    - Lucía (16 años) Per.  
Inicial: J  
    - Juan (55 años) Arg.  
    - José (9 años) Uru.
```



Vamos a utilizar  
estas dos clases  
Alumno y  
Examen en los  
siguientes  
ejemplos

```
class Alumno
{
    public int Id { get; private set; }
    public string Nombre { get; private set; }
    public Alumno(int id, string nombre)
    {
        Id = id;
        Nombre = nombre;
    }
}

class Examen
{
    public int AlumnoId { get; private set; }
    public string Materia { get; private set; }
    public double Nota { get; private set; }
    public Examen(int alumnoId, string materia, double nota)
    {
        AlumnoId = alumnoId;
        Materia = materia;
        Nota = nota;
    }
}
```

```
var alumnos = new List<Alumno>() {  
    new Alumno(1,"Juan"),  
    new Alumno(2,"Ana"),  
    new Alumno(3,"Laura")  
};
```

```
var examenes = new List<Examen>() {  
    new Examen(2,"Inglés",9),  
    new Examen(1,"Inglés",5),  
    new Examen(1,"Álgebra",10)  
};
```

```
var listado = alumnos.Join(examenes,  
    a => a.Id, //clave de matching en alumnos  
    e => e.AlumnoId, //clave de matching en notas  
    (a, e) => new  
    {  
        Alumno = a.Nombre,  
        Materia = e.Materia,  
        Notas = e.Nota  
    });  
listado.ToList().ForEach(obj => Console.WriteLine(obj));
```

Alumnos

Nombre	Id
Juan	1
Ana	2
Laura	3

Exámenes

IdAlumno	Materia	Nota
2	Inglés	9
1	Inglés	5
1	Álgebra	10

Join es como inner  
join de SQL, devuelve  
sólo los elementos  
donde las claves  
coinciden.

```
{ Alumno = Juan, Materia = Inglés, Notas = 5 }  
{ Alumno = Juan, Materia = Álgebra, Notas = 10 }  
{ Alumno = Ana, Materia = Inglés, Notas = 9 }
```

```
var alumnos = new List<Alumno>() {  
    new Alumno(1,"Juan"),  
    new Alumno(2,"Ana"),  
    new Alumno(3,"Laura")  
};
```

```
var examenes = new List<Examen>() {  
    new Examen(2,"Inglés",9),  
    new Examen(1,"Inglés",5),  
    new Examen(1,"Álgebra",10)  
};
```

```
var listado = alumnos.GroupJoin(examenes,  
    a => a.Id, //clave de matching en alumnos  
    e => e.AlumnoId, //clave de matching en notas  
    (a, examenes) => new  
    {  
        NombreAlumno = a.Nombre,  
        Examenes = examenes  
    });
```

```
foreach (var grup in listado)  
{  
    Console.WriteLine($"Alumno: {grup.NombreAlumno}");  
    Console.WriteLine($" Cant. exams.: {grup.Examenes.Count()}");  
    foreach (var e in grup.Examenes)  
    {  
        Console.WriteLine($" {e.Materia} Nota:{e.Nota}");  
    }  
}
```

Alumnos

Nombre	Id
Juan	1
Ana	2
Laura	3

Exámenes

IdAlumno	Materia	Nota
2	Inglés	9
1	Inglés	5
1	Álgebra	10

Can GroupJoin  
agrupamos a los  
alumno con sus  
exámenes. También  
obtengo información  
de quienes no rindieron  
ninguno

```
Alumno: Juan, Cant. exams.: 2  
Inglés Nota:5  
Álgebra Nota:10  
Alumno: Ana, Cant. exams.: 1  
Inglés Nota:9  
Alumno: Laura, Cant. exams.: 0
```

# Persistencia de datos

## Vamos a usar SQLite para persistir datos

SQLite es una biblioteca que implementa un motor de base de datos SQL “en proceso”, autónomo, sin servidor, de configuración cero.

SQLite es de código abierto y, por lo tanto, es gratuito para su uso para cualquier propósito.







## Crear una aplicación de consola llamada Escuela

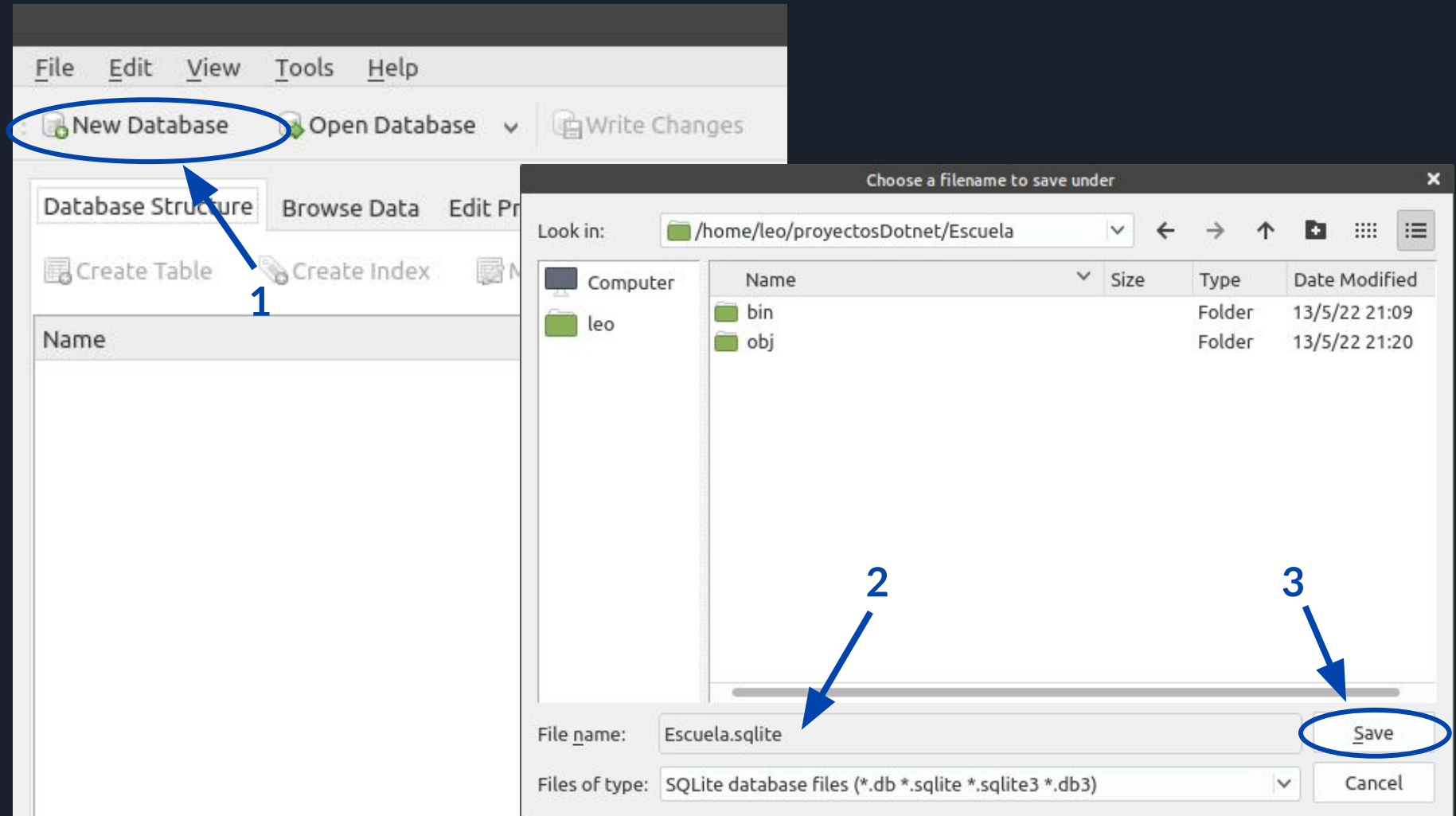


1. Abrir una terminal del sistema operativo
2. Cambiar a la carpeta `proyectosDotnet`
3. Crear la aplicación de consola `Escuela`
4. En la carpeta raíz del proyecto crear la base de datos `Escuela.sqlite` utilizando `DB Browser for SQLite`

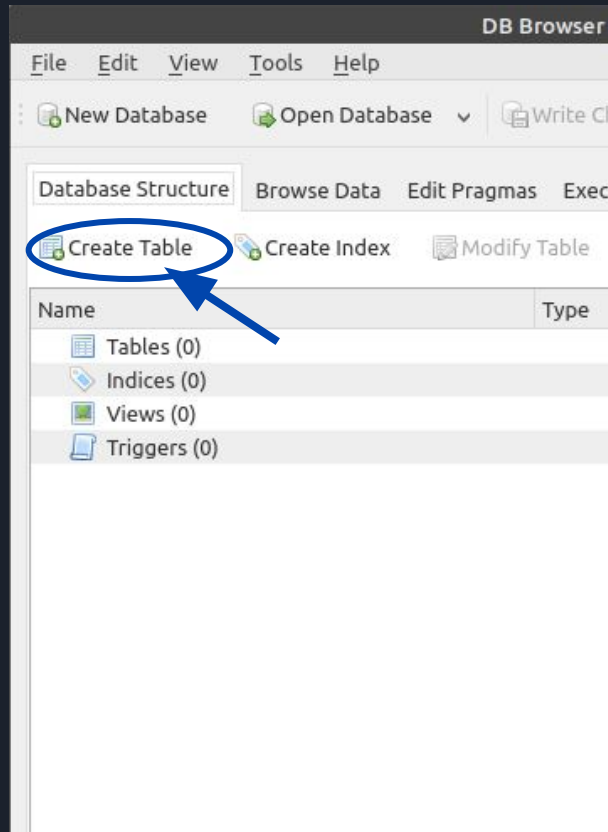
# Descargar e instalar DB Browser for SQLite (<https://sqlitebrowser.org>)



Utilizando **DB Browser** crear la base **Escuela.sqlite** en la carpeta raíz del proyecto que acabamos de crear (donde se encuentra el archivo **Escuela.csproj**)



Crear la tabla **Alumnos** con los campos **Id** de tipo **INTEGER**, **Nombre** y **Email** de tipo **TEXT**



Marcamos el campo **Id** como clave de la tabla (PK). Además marcamos (AI) para que este campo sea establecido automáticamente por SQLite, de manera incremental, al momento de agregar una nueva fila a la tabla. Tanto **Nombre** como **Id** deben ser no nulos

## Agregar algunos datos a la tabla Alumnos

The screenshot shows the SQLite GUI interface. The 'Browse Data' tab is selected, and the 'Alumnos' table is chosen. The table contains three records. A blue callout points to the 'Browse Data' tab, and another points to the 'New Record' button.

Pestaña  
Hoja de datos

Table: Alumnos

	Id	Nombre	Email
	Filter	Filter	Filter
1	1	Juan	juan@gmail.com
2	2	Ana	NULL
3	3	Laura	NULL

Presionando este botón se agrega un registro (fila) en blanco que podemos editar. El campo Id se completa solo

Crear la tabla **Exámenes** con los campos que se observan en esta diapositiva

Edit table definition

Table

Exámenes

Advanced

Fields

Add field

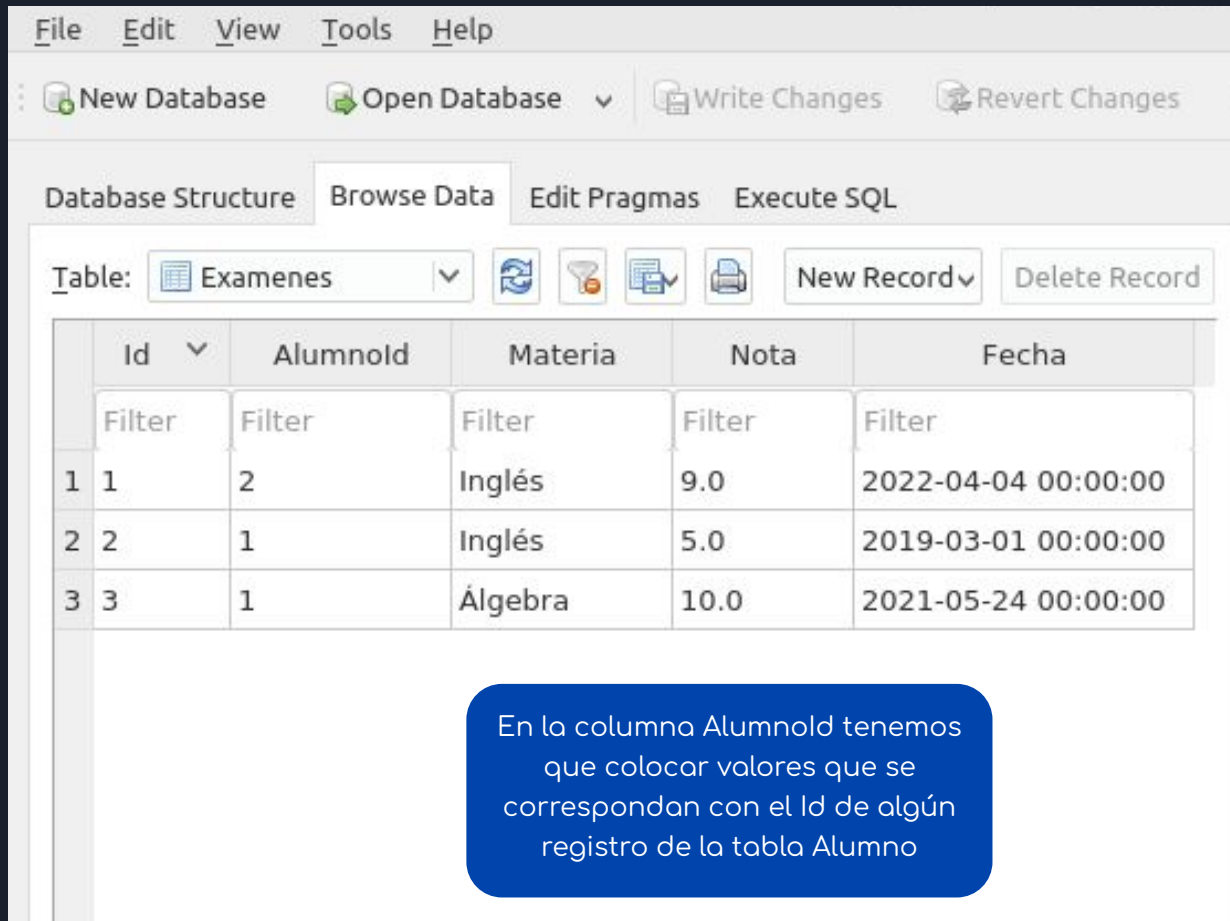
Remove field

Move field up

Move field down

Name	Type	NN	PK	AI	U	Default	Check	Foreign Key
Id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
Alumnoid	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
Materia	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
Nota	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
Fecha	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			

# Agregar algunos datos a la tabla Exámenes



The screenshot shows the SQLite Browser application with the 'Exámenes' table selected. The table has five columns: Id, Alumnold, Materia, Nota, and Fecha. The data is as follows:

	Id	Alumnold	Materia	Nota	Fecha
1	1	2	Inglés	9.0	2022-04-04 00:00:00
2	2	1	Inglés	5.0	2019-03-01 00:00:00
3	3	1	Álgebra	10.0	2021-05-24 00:00:00

A blue callout box contains the following text:

En la columna Alumnold tenemos que colocar valores que se correspondan con el Id de algún registro de la tabla Alumno

## Entity Framework Core

Nuestra aplicación de consola accederá a los datos de **Escuela.sqlite** mediante **Entity Framework Core**.

**EF Core** es un **ORM** (object-relational mapper) que permite trabajar con una base de datos utilizando objetos .Net y ahorrando la escritura de mucho código de acceso a datos







## Instalar el paquete Nuget Entity Framework Core para SQLite



En la terminal del sistema operativo (o en la que provee el **Visual Studio Code**) posicionarse en la carpeta del proyecto (donde se encuentra el archivo **Escuela.csproj**) y tipear el siguiente comando:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

Este es el nombre del paquete  
que se requiere instalar

## Entity Framework Core

Con **EF Core**, el acceso a datos se realiza mediante un **modelo**. Un modelo se compone de **clases de entidad** y un **objeto de contexto** que representa **una sesión con la base de datos**. Este objeto de contexto permite consultar y guardar datos.





# Codificar las clases de entidad Alumno y Examen



-----Alumno.cs-----

```
namespace Escuela;
public class Alumno
{
    public int Id { get; set; }
    public string Nombre { get; set; } = "";
    public string? Email { get; set; }
}
```

-----Examen.cs-----

```
namespace Escuela;
public class Examen
{
    public int Id { get; set; }
    public int AlumnoId { get; set; }
    public string Materia { get; set; } = "";
    public double Nota { get; set; }
    public DateTime Fecha { get; set; }
}
```

Observar que se corresponden con la estructura de los datos guardados en las tablas **Alumnos** y **Exámenes** en la base de datos



## Codificar la clase EscuelaContext



```
using Microsoft.EntityFrameworkCore;

namespace Escuela;

public class EscuelaContext : DbContext
{

    public DbSet<Alumno> Alumnos { get; set; }
    public DbSet<Examen> Exámenes { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
        optionsBuilder)
    {
        optionsBuilder.UseSqlite("data source=Escuela.sqlite");
    }
}
```

Esta clase debe heredar de `DbContext` e invalidar el método `OnConfiguring` que se utiliza para identificar la fuente de datos

El nombre de estas propiedades coincide con el nombre de las tablas en la base de datos



## Codificar la clase EscuelaContext



```
using Microsoft.EntityFrameworkCore;
```

```
namespace Escuela;
```

```
public class EscuelaContext : DbContext
{
    #nullable disable
    public DbSet<Alumno> Alumnos { get; set; }
    public DbSet<Examen> Exámenes { get; set; }
    #nullable restore
```

Las propiedades `Alumnos` y `Exámenes` serán inicializadas por Entity Framework. Para evitar los warnings del compilador podemos deshabilitar el contexto nullable en esta sección

```
protected override void OnConfiguring(DbContextOptionsBuilder
    optionsBuilder)
{
    optionsBuilder.UseSqlite("data source=Escuela.sqlite");
}
}
```

Invalidando `OnConfiguring` podemos establecer el motor de la base de datos y el archivo



# Codificar Program.cs de la siguiente manera



```
using Escuela;

using (var db = new EscuelaContext())
{
    Console.WriteLine("-- Tabla Alumnos --");
    foreach (var a in db.Alumnos)
    {
        Console.WriteLine($"{a.Id} {a.Nombre}");
    }

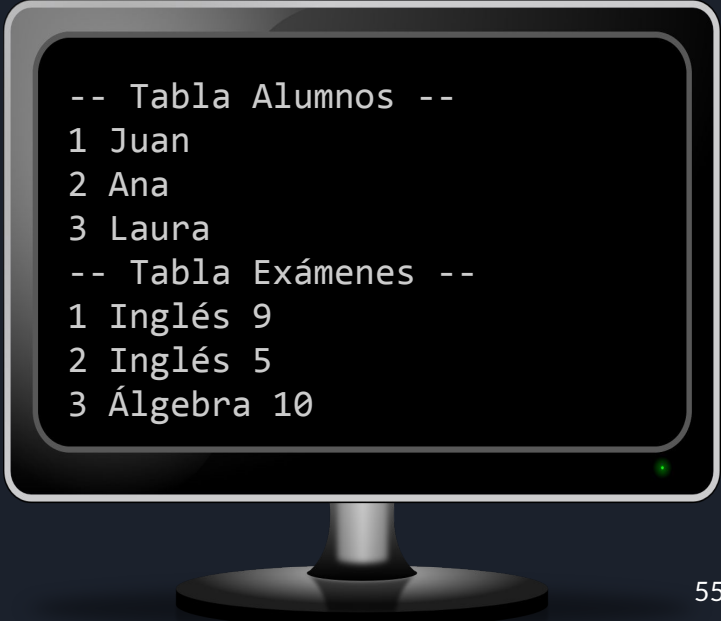
    Console.WriteLine("-- Tabla Exámenes --");
    foreach (var ex in db.Examenes)
    {
        Console.WriteLine($"{ex.Id} {ex.Materia} {ex.Nota}");
    }
}
```

```
using Escuela;

using (var db = new EscuelaContext())
{
    Console.WriteLine("-- Tabla Alumnos --");
    foreach (var a in db.Alumnos)
    {
        Console.WriteLine($"{a.Id} {a.Nombre}");
    }

    Console.WriteLine("-- Tabla Exámenes --");
    foreach (var ex in db.Examenes)
    {
        Console.WriteLine($"{ex.Id} {ex.Materia} {ex.Nota}");
    }
}
```

Un objeto `DbContext` está diseñado para usarse durante una única unidad de trabajo, por lo que la duración de una instancia de un `DbContext` suele ser muy breve.



```
-- Tabla Alumnos --
1 Juan
2 Ana
3 Laura
-- Tabla Exámenes --
1 Inglés 9
2 Inglés 5
3 Álgebra 10
```



## Entity Framework Core - Code First

Si empezamos un proyecto nuevo, para el cual no existe una base de datos creada, podemos crearla fácilmente a partir del código C# que ya tenemos codificado. A esta estrategia se la conoce con el nombre de "Code First"







Borrar la base de datos Escuela.sqlite del proyecto, codificar Program.cs de esta manera y ejecutar



```
using Escuela;

using (var context = new EscuelaContext())
{
    context.Database.EnsureCreated();
}
```

Este método no hace nada si la base de datos ya existe. En caso contrario la crea en base al modelo definido en `EscuelaContext`

# Persistencia de datos - SQLite - Entity Framework Core

Escuela.sqlite - Escuela - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- ESCUELA
  - .vscode
  - bin
  - obj
  - Alumno.cs
  - Escuela.csproj
  - Escuela.sqlite
  - EscuelaContext.cs
  - EscuelaIni.cs
  - Examen.cs
  - Program.cs

Escuela.sqlite

Search tables... Reset Filters Records: 0 Search 0 records...

Tables (3)

- Alumnos
  - Id
  - Nombre
  - Email
- sqlite\_sequence
- Examenes

Id Nombre Email

Search column... Search column... Search column...

Se creó la base de datos Escuela.sqlite

En Escuela.sqlite se crearon las tablas Alumnos y Examenes de acuerdo al modelo definido

OUTLINE

TIMELINE

0 0 .NET Core Launch (console) (Escuela) Escuela

Page 1 / 1 Try SQLite Viewer Web

# Persistencia de datos - SQLite - Entity Framework Core

The screenshot shows the Visual Studio Code interface with the 'Escuela.sqlite - Escuela - Visual Studio Code' window. The Extensions Marketplace is open on the left, displaying a list of extensions. The 'SQLite Viewer' extension by Florian Klampfer is highlighted with a white box. A blue callout bubble points to this extension with the text: 'Para visualizar el contenido de la base de datos en Visual Studio Code instalar alguna extensión para SQLite'. The main editor area shows the 'Escuela.sqlite' file with a table view of 'Alumnos' containing columns 'Id', 'Nombre', and 'Email'. The status bar at the bottom shows '.NET Core Launch (console) (Escuela)' and 'Escuela'.

File Edit Selection View Go Run Terminal Help

EXTENSIONS: MARKETPLACE

sqlite

**SQLite Viewer** 3ms  
SQLite Viewer for VSCode  
Florian Klampfer

**SQLite & MySQL Snippets** 13K ★ 5  
A snippet for MySQL and SQLite...  
Rohit Chouhan **Install**

**SQLTools** 1.7M ★ 3.5  
Database management done rig...  
Matheus Teixeira **Install**

**Flutter ORM M8 Gener...** 15K  
Vs Code generator for orm-m8 f...  
Mircea-Tiberiu MATEI **Install**

**dBizzy** 2K ★ 4  
ER Diagram Visualizer  
dBizzy **Install**

**GrayCat SQL Format...** 8K ★ 3  
A VSCode formatter extension f...  
Adam Rybak **Install**

**MySQL** 499K ★ 4.5  
Database manager for MySQL/...  
cweijan **Install**

**Database Client** 62K ★ 5  
Database manager for MySQL/...

Escuela.sqlite x Program.cs

Escuela.sqlite

Search tables.. Reset Filters Records: 0 Search 0 records...

Tables (3)

Alumnos

Id Nombre Email

Search column... Search column... Search column...

sqlite\_sequence

Examenes

Para visualizar el contenido de la base de datos en Visual Studio Code instalar alguna extensión para SQLite

0 0 .NET Core Launch (console) (Escuela) Escuela

Page 1 / 1 Try SQLite Viewer Web

# Persistencia de datos - SQLite - Entity Framework Core

Table:

Advanced

Fields

Add field Remove field Move field up Move field down

Name	Type	NN	PK	AI	U
Id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Nombre	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Email	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

```
1 CREATE TABLE "Alumnos" (  
2     "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
3     "Nombre" TEXT NOT NULL,  
4     "Email" TEXT  
5 );
```

Cancel

Table:

Advanced

Fields

Add field Remove field Move field up Move field down

Name	Type	NN	PK	AI	U	Default	Check
Id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
AlumnoId	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Materia	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Nota	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Fecha	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

```
1 CREATE TABLE "Exámenes" (  
2     "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
3     "AlumnoId" INTEGER NOT NULL,  
4     "Materia" TEXT NOT NULL,  
5     "Nota" REAL NOT NULL,  
6     "Fecha" TEXT NOT NULL  
7 );
```

Cancel OK

Las tablas Alumnos y Exámenes se crearon con estas configuraciones a partir de nuestro modelo utilizando ciertas convenciones por defecto

Table

Alumnos

Advanced

Fields

Add field Remove field Move field up Move field down

Name	Type	NN	PK	AI	U
Id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Nombre	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Email	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

```
1 CREATE TABLE "Alumnos" (  
2     "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
3     "Nombre" TEXT NOT NULL,  
4     "Email" TEXT  
5 );
```

Table

Exámenes

Advanced

Fields

Add field Remove field Move field up Move field down

Name	Type	NN	PK	AI	U	Default	Check
Id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
AlumnoId	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Materia	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Nota	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Fecha	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

```
1 CREATE TABLE "Exámenes" (  
2     "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
3     "AlumnoId" INTEGER NOT NULL,  
4     "Materia" TEXT NOT NULL,  
5     "Nota" REAL NOT NULL,  
6     "Fecha" TEXT NOT NULL  
7 );
```

Cancel OK

## Convenciones por defecto utilizadas:

- El nombre de las tablas se determinó a partir del nombre de las propiedades DbSet<> de EscuelaContext
- La propiedad Id de las entidades se establecen como claves de la tablas
- Si la propiedad Id de una entidad es entera, se establece como clave autoincremental
- Sqlite soporta pocos tipos de datos, se utiliza un mapeo adecuado, por ejemplo las propiedades DateTime de .NET se establecen como campos TEXT en las tablas Sqlite

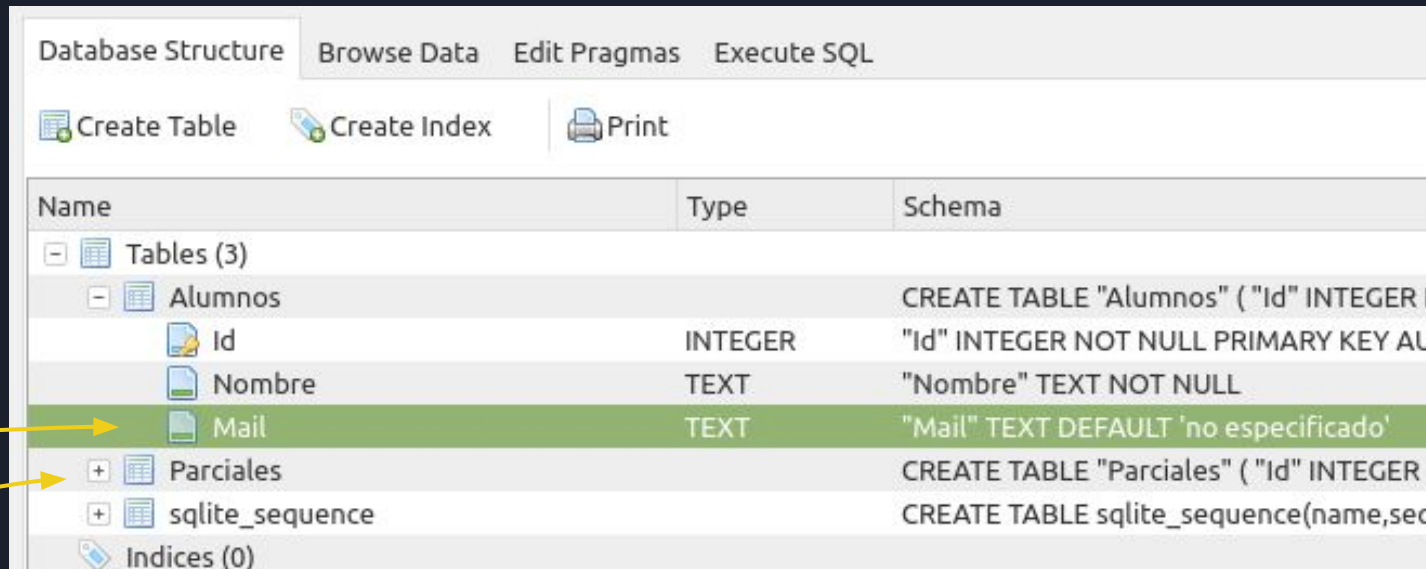
# Persistencia de datos - SQLite - Entity Framework Core

```
public class EscuelaContext : DbContext
{
    #nullable disable
    public DbSet<Alumno> Alumnos { get; set; }
    public DbSet<Examen> Exámenes { get; set; }
    #nullable restore

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("data source=Escuela.sqlite");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Examen>().ToTable("Parciales");
        modelBuilder.Entity<Alumno>()
            .Property(a => a.Email)
            .HasColumnName("Mail").HasDefaultValue("no especificado");
    }
}
```

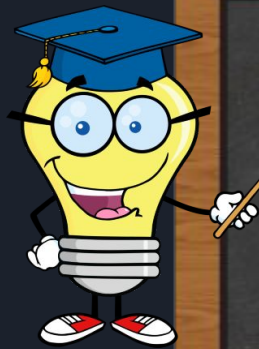
Se podría invalidar OnModelCreating para especificar distintos aspectos sobre cómo se realiza el mapeo entre el modelo y la base de datos utilizando Fluent API  
(es sólo ilustrativo, no codificarlo para seguir estas diapositivas)



Database Structure		
Browse Data Edit Pragmas Execute SQL		
Create Table Create Index Print		
Name	Type	Schema
Tables (3)		
Alumnos		CREATE TABLE "Alumnos" ("Id" INTEGER NOT NULL PRIMARY KEY, "Nombre" TEXT NOT NULL, "Mail" TEXT DEFAULT 'no especificado')
Id	INTEGER	"Id" INTEGER NOT NULL PRIMARY KEY
Nombre	TEXT	"Nombre" TEXT NOT NULL
Mail	TEXT	"Mail" TEXT DEFAULT 'no especificado'
Parciales		CREATE TABLE "Parciales" ("Id" INTEGER NOT NULL PRIMARY KEY, "Nombre" TEXT NOT NULL, "Mail" TEXT DEFAULT 'no especificado')
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (0)		



## Entity Framework Core - Code First



Existe un mecanismo más sofisticado para implementar “Code First” que se llama migraciones.

Con las **migraciones** es posible realizar modificaciones incrementales en la base de datos e incluso volver para atrás removiendo la última migración

Para utilizarlas es necesario instalar las herramientas de EF necesarias



## Codificar la siguiente clase para agregar algunos datos a las tablas



```
namespace Escuela;

public class EscuelaInit
{
    public static void Inicializar(EscuelaContext context)
    {
        if (context.Alumnos.Count() > 0) // ya fue inicializada
        {
            return;
        }

        context.Add(new Alumno() { Nombre = "Juan", Email="juan@gmail.com" });
        context.Add(new Alumno() { Nombre = "Ana" });
        context.Add(new Alumno() { Nombre = "Laura" });

        context.Add(new Examen() { AlumnoId = 2, Materia = "Inglés", Nota = 9,
            Fecha = DateTime.Parse("4/4/2022") });
        context.Add(new Examen() { AlumnoId = 1, Materia = "Inglés", Nota = 5,
            Fecha = DateTime.Parse("1/3/2019") });
        context.Add(new Examen() { AlumnoId = 1, Materia = "Álgebra", Nota = 10,
            Fecha = DateTime.Parse("24/5/2021") });

        context.SaveChanges();
    }
}
```

Copiar el código del archivo  
11\_Teoria\_codigo.txt



```
namespace Escuela;
```

```
public class EscuelaInit
```

```
{
```

```
    public static void Inicializar(EscuelaContext context)
```

```
    {
```

```
        if (context.Alumnos.Count() > 0) // ya fue inicializada
```

```
        {
```

```
            return;
```

```
        }
```

```
        context.Add(new Alumno() { Nombre = "Juan", Email="juan@gmail.com" });
```

```
        context.Add(new Alumno() { Nombre = "Ana" });
```

```
        context.Add(new Alumno() { Nombre = "Laura" });
```

```
        context.Add(new Examen() { AlumnoId = 2, Materia = "Inglés", Nota = 9,  
            Fecha = DateTime.Parse("4/4/2022") });
```

```
        context.Add(new Examen() { AlumnoId = 1, Materia = "Inglés", Nota = 5,  
            Fecha = DateTime.Parse("1/3/2019") });
```

```
        context.Add(new Examen() { AlumnoId = 1, Materia = "Álgebra", Nota = 10,  
            Fecha = DateTime.Parse("24/5/2021") });
```

```
        context.SaveChanges();
```

```
    }
```

```
}
```

Observar que no es necesario especificar la propiedad DbSet<> donde agregar un Alumno o un Examen, porque existe una única propiedad por cada entidad



## Modificar Program.cs y ejecutar



```
using Escuela;

using (var context = new EscuelaContext())
{
    context.Database.EnsureCreated();
    EscuelaInit.Inicializar(context);
}
```

# Persistencia de datos - SQLite - Entity Framework Core

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Alumnos ↺ ↻ 🔍 📄 🖨 New Record Delete Record

	Id	Nombre	Email
	Filter	Filter	Filter
1	1	Juan	juan@gmail..
2	2	Ana	NULL
3	3	Laura	NULL

⏪ ⏩ 1 - 3 of 3 ⏪ ⏩ Go to:

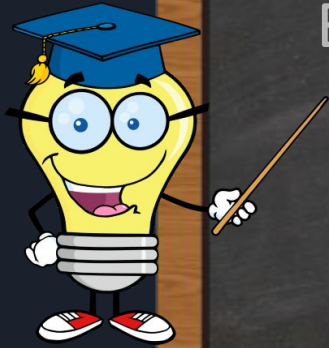
Database Structure Browse Data Edit Pragmas Execute SQL

Table: Exámenes ↺ ↻ 🔍 📄 🖨 New Record Delete Record

	Id	Alumnoid	Materia	Nota	Fecha
	Filter	Filter	Filter	Filter	Filter
1	1	2	Inglés	9.0	2022-04-04 ...
2	2	1	Inglés	5.0	2019-03-01 ...
3	3	1	Álgebra	10.0	2021-05-24 ...

⏪ ⏩ 1 - 3 of 3 ⏪ ⏩ Go to: 1

## Entity Framework Core - Code First



Está claro que existe una relación “uno a muchos” entre Alumnos y Exámenes

Alumnos

Nombre	Id
Juan	1
Ana	2
Laura	3

Exámenes

Alumnoid	Materia	Nota	Id
2	Inglés	9	1
1	Inglés	5	2
1	Álgebra	10	3

Por cada alumno podemos tener 0, 1 o más exámenes



## Modificar Program.cs y ejecutar

```
using Escuela;
using (var db = new EscuelaContext())
{
    var query = db.Alumnos.Join(db.Examenes,
        a => a.Id,
        e => e.AlumnoId,
        (a, e) => new
        {
            Alumno = a.Nombre,
            Materia = e.Materia,
            Nota = e.Nota
        });
    foreach (var obj in query)
    {
        Console.WriteLine(obj);
    }
}
```

Consultando  
datos

Utilizamos LINQ  
para realizar  
consultas

Copiar el código del archivo  
11\_Teoria\_codigo.txt

```
using Escuela;

using (var db = new EscuelaContext())
{
    var query = db.Alumnos.Join(db.Examenes,
        a => a.Id,
        e => e.AlumnoId,
        (a, e) => new
        {
            Alumno = a.Nombre,
            Materia = e.Materia,
            Nota = e.Nota
        });

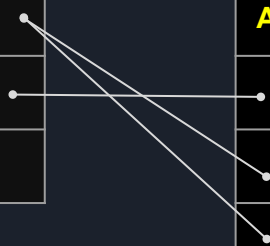
    foreach (var obj in query)
    {
        Console.WriteLine(obj);
    }
}
```

Alumnos

Nombre	Id
Juan	1
Ana	2
Laura	3

Exámenes

AlumnoId	Materia	Nota	Id
2	Inglés	9	1
1	Inglés	5	2
1	Álgebra	10	3



```
{ Alumno = Ana, Materia = Inglés, Nota = 9 }
{ Alumno = Juan, Materia = Inglés, Nota = 5 }
{ Alumno = Juan, Materia = Álgebra, Nota = 10 }
```



## Modificar Program.cs y ejecutar



```
using Escuela;

using (var db = new EscuelaContext())
{
    //Agregamos un nuevo alumno
    var alumno = new Alumno() { Nombre = "Pablo" }; // el Id será establecido por SQLite
    db.Add(alumno); // se agregará realmente con el db.SaveChanges()
    db.SaveChanges(); //actualiza la base de datos. SQLite establece el valor de alumno.Id

    // Agregamos un examen para el nuevo alumno
    var examen = new Examen()
    {
        AlumnoId = alumno.Id,
        Materia = "Historia",
        Nota = 9.5
    };
    db.Add(examen);
    db.SaveChanges();
}
```

Agregando  
registros

Copiar el código del archivo  
11\_Teoria\_codigo.txt

# Estado de las tablas luego de ejecutar el código

```
using Escuela;
```

```
using (var db = new EscuelaContext())
```

```
{
```

```
//Agregamos un n
```

```
var alumno = ne
```

```
db.Add(alumno);
```

```
db.SaveChanges()
```

```
// Agregamos un
```

```
var examen = ne
```

```
{
```

```
AlumnoId =
```

```
Materia = "
```

```
Nota = 9.5
```

```
};
```

```
db.Add(examen);
```

```
db.SaveChanges();
```

```
}
```

Así queda la base de datos

Alumnos		Exámenes			
Nombre	Id	AlumnoId	Materia	Nota	Id
Juan	1				
Ana	2	2	Inglés	9	1
Laura	3	1	Inglés	5	2
Pablo	4	1	Álgebra	10	3
		4	Historia	9.5	4





## Modificar Program.cs y ejecutar



```
using Escuela;
```

```
using (var db = new EscuelaContext())  
{
```

```
    //borramos de la tabla Alumnos el registro con Id=3
```

```
    var alumnoBorrar = db.Alumnos.Where(a => a.Id == 3).SingleOrDefault();
```

```
    if (alumnoBorrar != null)
```

```
    {
```

```
        db.Remove(alumnoBorrar); //se borra realmente con el db.SaveChanges()
```

```
    }
```

```
    //La nota en Inglés del alumno id=1 es un 5. La cambiamos a 7.5
```

```
    var examenModificar = db.Examenes.Where(  
        e => e.AlumnoId == 1 && e.Materia == "Inglés").SingleOrDefault();
```

```
    if (examenModificar != null)
```

```
    {
```

```
        examenModificar.Nota = 7.5; //se modifica el registro en memoria
```

```
    }
```

```
    db.SaveChanges(); //actualiza la base de datos.
```

```
}
```

Borrando y  
actualizando  
registros

Copiar el código del archivo  
11\_Teoria\_codigo.txt

# Persistencia de datos - SQLite - Entity Framework Core

```
using Escuela;
```

```
using (var db = new EscuelaContext())  
{
```

```
    //borramos de la tabla Alumnos el registro con Id=3
```

```
    var alumnoBorrar = db.Alumnos.Where(a => a.Id == 3).SingleOrDefault();
```

```
    if (alumnoBorrar != null)
```

```
    {
```

```
        db.Remove(alumnoBorrar); //se borra realmente con el db.SaveChanges()
```

```
    }
```

```
    //La nota en Inglés del alumno id=1 es un 5. La cambiamos a 7.5
```

```
    var examenModificar = db.Examenes.Where(e => e.AlumnoId == 1 && e.Materia == "Inglés").SingleOrDefault();
```

```
    if (examenModificar != null)
```

```
    {
```

```
        examenModificar.Nota = 7.5; //se modifica el registro en memoria
```

```
    }
```

```
    db.SaveChanges(); /
```

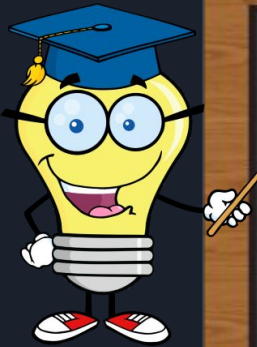
```
}
```

Si la secuencia contiene un único elemento, devuelve ese elemento. De lo contrario devuelve el valor por default (null en este caso)

Así queda la base de datos

Alumnos		Exámenes			
Nombre	Id	Alumnoid	Materia	Nota	Id
		2	Inglés	9	1
Juan	1	1	Inglés	7.5	2
Ana	2	1	Álgebra	10	3
Pablo	4	4	Historia	9.5	4

# Entity Framework Core - Code First



EF Core facilita la navegación entre entidades relacionadas por medio de las propiedades de navegación

Alumnos		Exámenes			
Nombre	Id	Alumnoid	Materia	Nota	Id
Juan	1	2	Inglés	9	1
Ana	2	1	Inglés	7.5	2
Pablo	4	1	Álgebra	10	3
		4	Historia	9.5	4

Sería conveniente tener una propiedad “Exámenes” en la entidad **Alumno** que me permitiera acceder a la lista de exámenes de cada alumno



## Modificar la clase Alumno



```
namespace Escuela;

public class Alumno
{
    public int Id { get; set; }
    public string Nombre { get; set; } = "";
    public string? Email { get; set; }

    public List<Examen>? Examenes { get; set; }
}
```

Propiedad de  
navegación



## Modificar Program.cs y ejecutar



```
using Microsoft.EntityFrameworkCore;  
using Escuela;
```

```
using (var db = new EscuelaContext())  
{
```

```
    foreach (Alumno a in db.Alumnos.Include(a => a.Examenes))  
    {
```

```
        Console.WriteLine(a.Nombre);
```

```
        a.Examenes?.ToList()
```

```
            .ForEach(ex => Console.WriteLine($" - {ex.Materia} {ex.Nota}"));
```

```
    }
```

```
}
```


Es necesario incluir  
explícitamente los datos  
relacionados



## Modificar Program.cs y ejecutar

```
using Microsoft.EntityFrameworkCore;
using Escuela;

using (var db = new EscuelaContext())
{
    foreach (Alumno a in db.Alumnos.Include(a => a.Examenes))
    {
        Console.WriteLine(a.Nombre);
        a.Examenes?.ToList()
            .ForEach(ex => Console.WriteLine($" - {ex.Materia} {ex.Nota}"));
    }
}
```

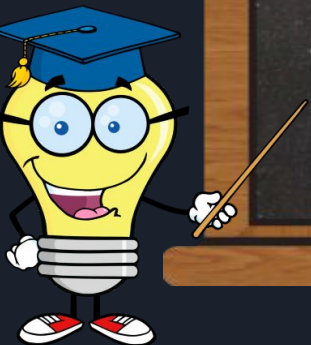


```
Juan
- Inglés 7,5
- Álgebra 10
Ana
- Inglés 9
Pablo
- Historia 9,5
```

## Entity Framework Core - Code First

Las propiedades de navegación, también facilitan el alta de información en las tablas relacionadas de la base de datos

Podemos agregar un nuevo alumno con la información de sus exámenes en una única operación `SaveChanges()`





## Modificar Program.cs y ejecutar



```
using Microsoft.EntityFrameworkCore;
using Escuela;

using (var db = new EscuelaContext())
{
    Alumno nuevo = new Alumno()
    {
        Nombre = "Andrés",
        Examenes = new List<Examen>() {
            new Examen() {Materia="Lengua",Nota=7,Fecha = DateTime.Parse("5/5/2022") },
            new Examen() {Materia="Matemática",Nota=6,Fecha = DateTime.Parse("6/5/2022") }
        }
    };
    db.Add(nuevo);
    db.SaveChanges();

    foreach (Alumno a in db.Alumnos.Include(a => a.Examenes))
    {
        Console.WriteLine(a.Nombre);
        a.Examenes?.ToList()
            .ForEach(ex => Console.WriteLine($" - {ex.Materia} {ex.Nota}"));
    }
}
```

Se crea un alumno con su lista de exámenes y se salva de una sola vez





## Modificar Program.cs y ejecutar

```
using Microsoft.EntityFrameworkCore;
using Escuela;
```

```
using (var db = new EscuelaContext())
{
```

```
    Alumno
```

```
    {
```

```
    }
```

```
    db
```

```
    .Add
```

```
    (Alumno)
```

```
    );
```

```
    db
```

```
    .Save
```

```
    ();
```

```
    }
```

```
}
```

Así queda la base de datos

Exámenes

Alumnos

Nombre	Id
Juan	1
Ana	2
Pablo	4
Andrés	5

Alumnoid	Materia	Nota	Id
2	Inglés	9	1
1	Inglés	7.5	2
1	Álgebra	10	3
4	Historia	9.5	4
5	Lengua	7	5
5	Matemática	6	6

Juan  
 - Inglés 7,5  
 - Álgebra 10  
 Ana  
 - Inglés 9  
 Pablo  
 - Historia 9,5  
 Andrés  
 - Lengua 7  
 - Matemática 6

```
{2"} }
```

# Fin de Teoría 11