LINQ

Héctor Pérez

Microsoft MVP

¿Qué es LINQ?

Language-Integrated Query

¿Qué es LINQ?

- Conjunto de tecnologías basadas en la integración de funciones de consulta directamente en el lenguaje C#.

Ejemplo - Tabla

Employees Table

IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

Ejemplo - SQL

```sql
SELECT IdNum, LNmae

FROM Employees

WHERE Salary > 50000
```

Ejemplo – C#

```
public void Query()
{
    string connectionString = "connectionString";
    string query = "SELECT IdNum, LNmae FROM Employees WHERE Salary > 50000";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);

        try
        {
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                Console.WriteLine($"IdNum: {reader["IdNum"]}, LNmae: {reader["LNmae"]}");
            }
            reader.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

Ejemplo – Problemas

- Código propenso a errores
- No hay validación en tiempo de compilación
- No hay Soporte IntelliSense
- Una super query puede ser muy confuse
- Hay que aprender diferentes lenguajes de consulta para cada tipo de fuente

LINQ

- En LINQ, una consulta es una construcción de lenguaje de primera clase, al igual que las clases, los métodos y los eventos.

Ejemplo - LINQ

```
public void Query()
{
    List<Employee> employees = new List<Employee>
    {
        new Employee { IdNum = 1, LName = "Smith", Salary = 60000 },
        new Employee { IdNum = 2, LName = "Johnson", Salary = 45000 }
    };

    var query = from employee in employees
                where employee.Salary > 50000
                select new { employee.IdNum, employee.LName };

    foreach (var employee in query)
    {
        Console.WriteLine($"IdNum: {employee.IdNum}, LName: {employee.LName}");
    }
}
```

Expresiones Lambda

- Son funciones anónimas que proporcionan una forma concisa de representar una función inline.
- Se utilizan comúnmente con métodos de la clase System.Linq.

Expresiones Lambda

(arguments) => expression

Expresiones Lambda

$$(x, y) => x + y$$

Expresiones Lambda

- ¿Cuáles son los argumentos?
- ¿Cuál es la expression?

```
void SayHello()
{
    Console.WriteLine("Hello");
}
```

Expresiones Lambda

```
void SayHello()
{
    Console.WriteLine("Hello");
}

() => Console.WriteLine("Hello");
```

Expresiones Lambda

- ¿Cuáles son los argumentos?
- ¿Cuál es la expression?

```
void SayHello(string name)
{
    Console.WriteLine(name);
}
```

Expresiones Lambda

```
void SayHello(string name)
{
    Console.WriteLine(name);
}

(name) => Console.WriteLine(name);
```

Expresiones Lambda

- ¿Cuáles son los argumentos?
- ¿Cuál es la expression?

```
int Add(int arg1, int arg2)
{
    return arg1 + arg2;
}
```

Expresiones Lambda

```
int Add(int arg1, int arg2)
{
    return arg1 + arg2;
}
```

```
(arg1, arg2) => { return arg1 + arg2 }
```

Expresiones Lambda

```
1 reference
int DoSomething(int arg1, int arg2)
{
    return arg1 + arg2;
}

var result = DoSomething(2, 2);

var result2 = (arg1, arg2) => arg1 + arg2;
```

(parameter) ? arg2

CS8917: The delegate type could not be inferred.

Show potential fixes (Alt+Enter or Ctrl+.)

Expresiones Lambda

- ¿Qué devuelve esta expression?

```
var operation = (arg1, arg2, arg3) => arg1 + arg2 + arg3;
```

Action

- Representa un método que puede ser llamado y que realiza una acción pero no devuelve ningún valor (es decir, su tipo de retorno es void)

```
void SayHello()
{
    Console.WriteLine("Hello");
}
```

Action

```
void SayHello()
{
    Console.WriteLine("Hello");
}
```

```
Action sayHello = () => Console.WriteLine("Hello");
```

Action

Demo

Func

- Es un tipo de delegado que se utiliza para encapsular un método que devuelve un valor.
- A diferencia de Action, que se usa para métodos que no retornan nada (void), Func<> está diseñado específicamente para métodos que devuelven un valor.
- Puede tener de 0 a 16 parámetros de entrada y un parámetro de salida obligatorio
- El último tipo genérico en la definición de Func<> siempre es el tipo de valor de retorno.

Func

```
double GetPI()
{
    return 3.1416;
}
```

```
Func<double> getPI = () => 3.1416;
```

Func

```
double Add(double arg1, double arg2)
{
    return arg1 + arg2;
}
```

```
Func<double, double, double> add = (arg1, arg2) => arg1 + arg2;
```

Func

Demo

Información de los estudiantes

Id	First Name	Last Name	University Id
1	Héctor	Pérez	1
2	Ana	Nepomuceno	2
3	Pedro	Sánchez	3
4	José	Infante	3
5	Regina	Bustamante	2
6	Rodrigo	Jiménez	4
7	Miguel	Hernández	5
8	Marilyn	Monroe	5
9	Leonardo	Estrada	4
10	Ricardo	Rojas	1

Direcciones de las Universidades

Id	Name	City	Country
1	Real de Brasil	Brasilia	Brasil
2	Oxford	Oxford	Reino Unido
3	Harvard	Cambridge	Estados Unidos
4	Brooklyn	Nueva York	Estados Unidos
5	UNAM	Ciudad de México	México

DBContext

Demo

Operaciones de proyección (Select)

- La proyección se refiere a la operación de transformar un objeto en una nueva forma que a menudo consiste sólo en aquellas propiedades que se utilizarán posteriormente.

Operaciones de proyección (Select)

- Lista Original

```
+-----------------------------------------+
| Employee                                |
+-----------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000 |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
+-----------------------------------------+
```

```
var simplifiedEmployees = employeesList.Select(emp => new { emp.Name, emp.Department });
```

- Select aplicado

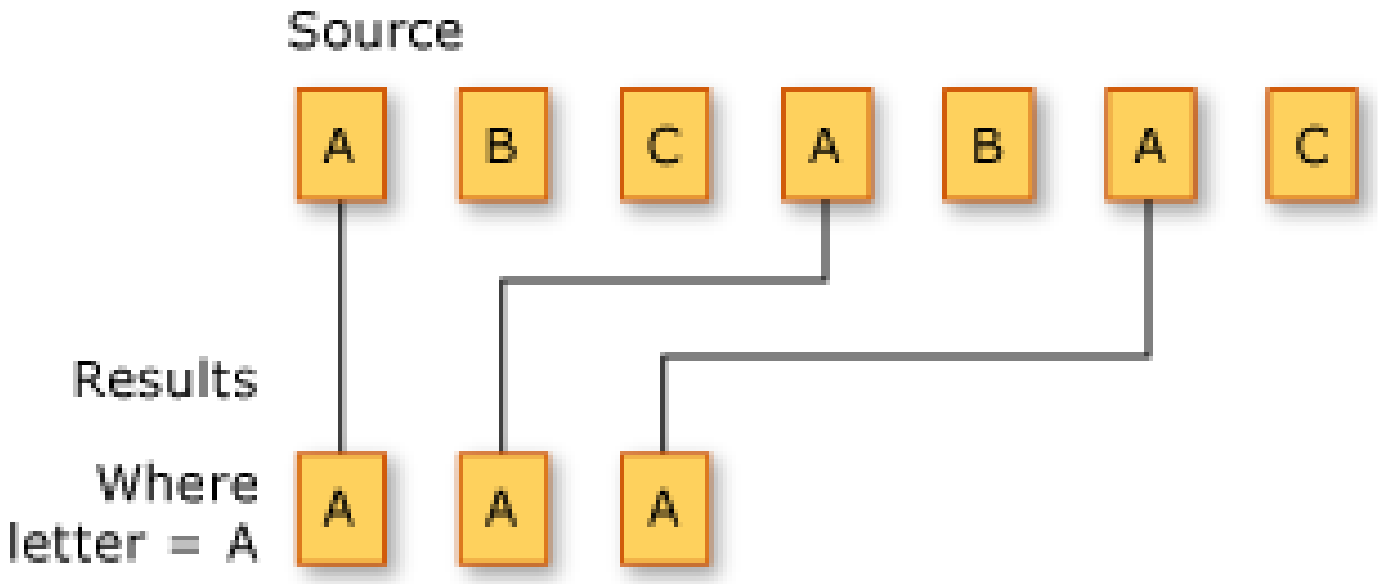
```
+---------------------+
| Simplified Employees |
+---------------------+
| Name: John, Department: HR       |
| Name: Sarah, Department: IT      |
| Name: Mike, Department: Marketing|
| Name: Emma, Department: Sales    |
+---------------------+
```

Select

Demo

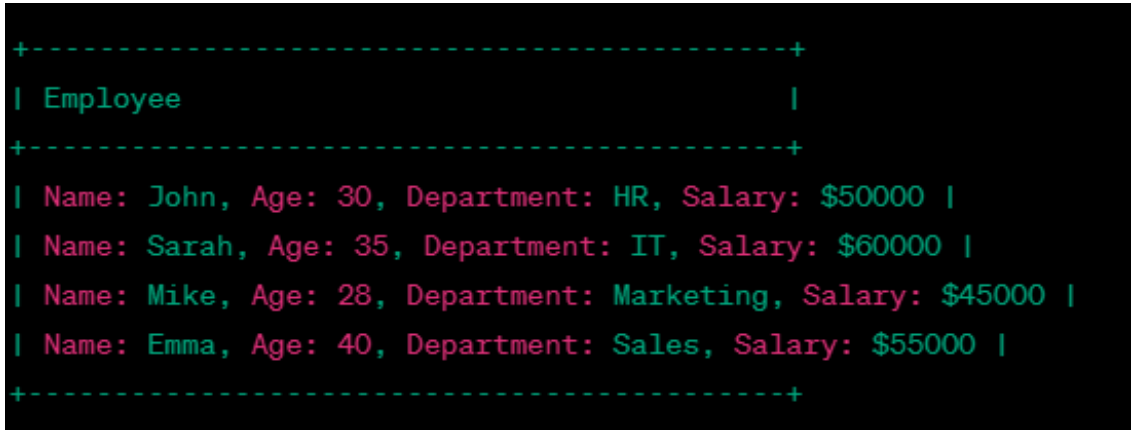
Operaciones de filtrado (Where)

- Por filtrado se entiende la operación de restringir el conjunto de resultados para que sólo contenga los elementos que satisfacen una condición especificada. También se conoce como selección.



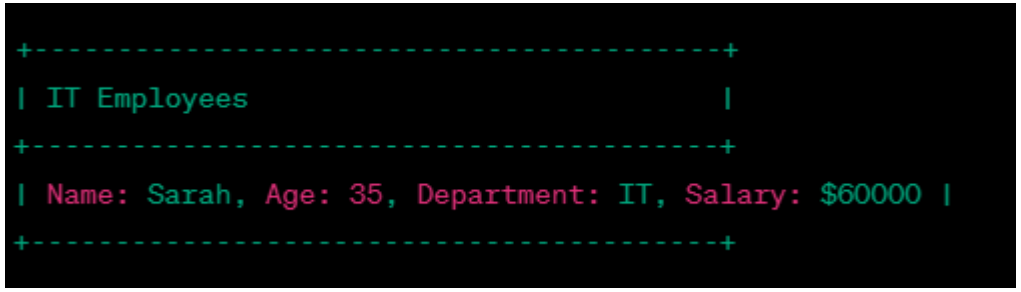
Operaciones de filtrado (Where)

- Lista Original




```
var itEmployees = employeesList.Where(emp => emp.Department == "IT");
```

- Where aplicado

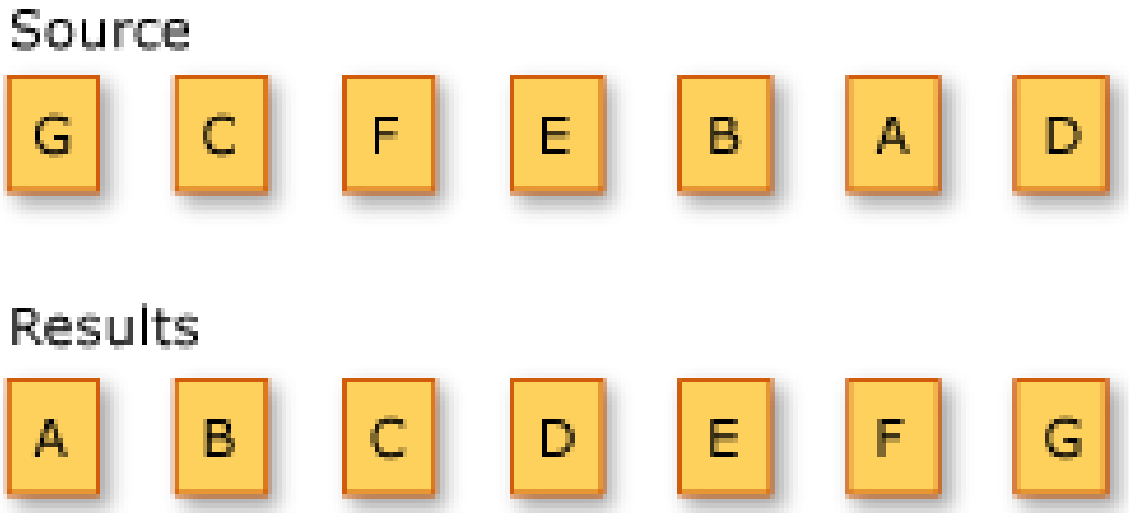


Where

Demo

Operaciones de ordenamiento (OrderBy)

- Una operación de ordenación ordena los elementos de una secuencia basándose en uno o más atributos.
- El primer criterio de ordenación realiza una ordenación primaria de los elementos.
- Especificando un segundo criterio de ordenación, puede ordenar los elementos dentro de cada grupo de ordenación primaria.



Operaciones de ordenamiento (OrderBy)

- Lista Original

```
+-----------------------------------------+
| Employee                                |
+-----------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000 |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
+-----------------------------------------+
```

```
var sortedByAge = employeesList.OrderBy(emp => emp.Age);
```

- Where aplicado

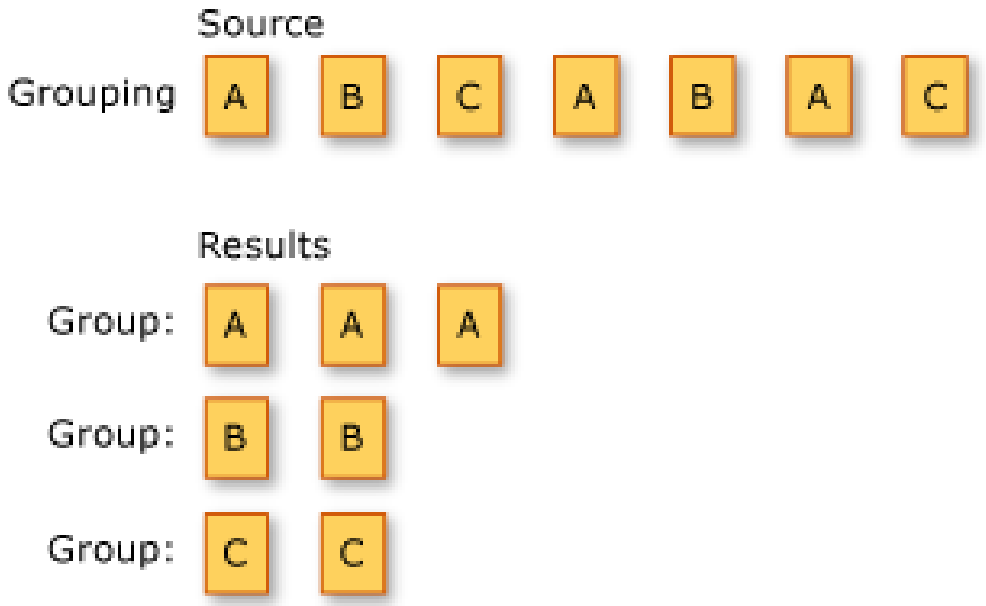
```
+-------------------------------------------------+
| Employees Ordered by Age                        |
+-------------------------------------------------+
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: John, Age: 30, Department: HR, Salary: $50000  |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
+-------------------------------------------------+
```

OrderBy y OrderByDescending

Demo

Operaciones de agrupamiento (GroupBy)

- Por agrupación se entiende la operación de poner los datos en grupos de forma que los elementos de cada grupo compartan un atributo común.



Operaciones de agrupamiento (GroupBy)

```
+-----------------------------------------+
| Employee                                |
+-----------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000 |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
+-----------------------------------------+
```

var groupedByDepartment = employeesList.GroupBy(emp => emp.Department);

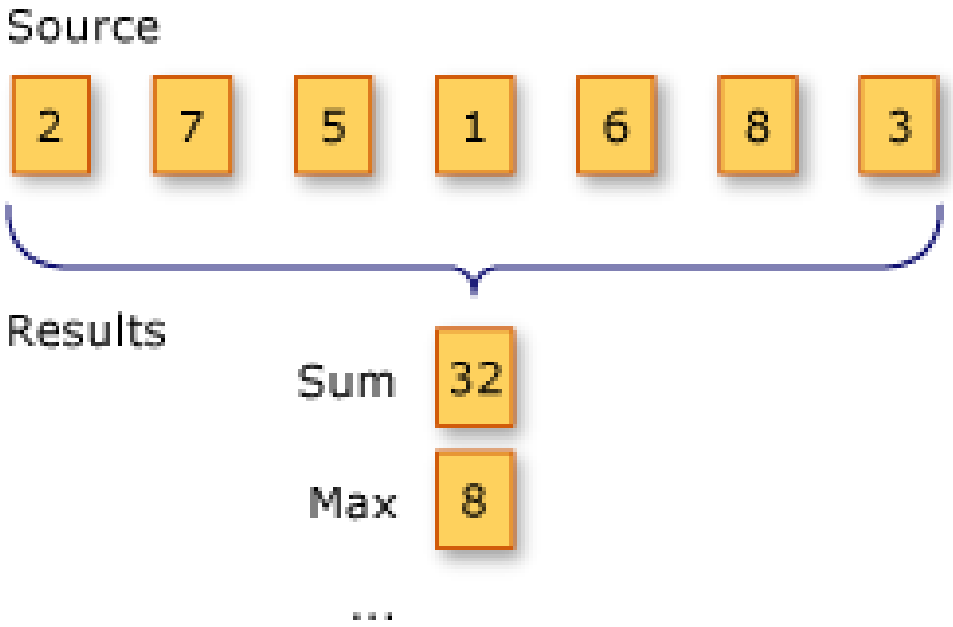
```
+---------------------------------+
| Employees Grouped by Department |
+---------------------------------+
| Department: HR                  |
| - Name: John, Age: 30, Salary: $50000 |
| - Name: Bob, Age: 36, Salary: $48000  |
| Department: IT                  |
| - Name: Sarah, Age: 35, Salary: $60000 |
| - Name: Mike, Age: 28, Salary: $45000 |
| Department: Sales               |
| - Name: Emma, Age: 40, Salary: $55000 |
+---------------------------------+
```

GroupBy

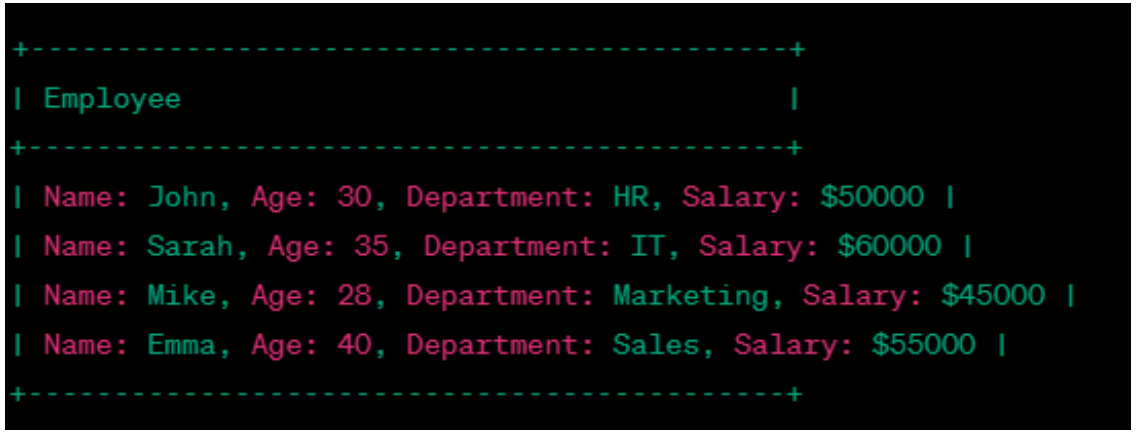
Demo

Operaciones de agregación (Count)

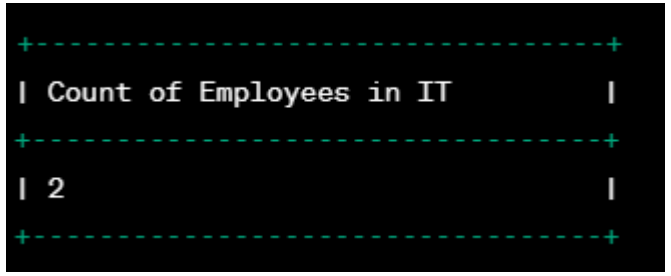
- Una operación de agregación calcula un único valor a partir de una colección de valores.
- Un ejemplo de operación de agregación es calcular la temperatura media diaria a partir de los valores de temperatura diaria de un mes.



Operaciones de agregación (Count)




```
int countIT = employeesList.Count(emp => emp.Department == "IT");
```



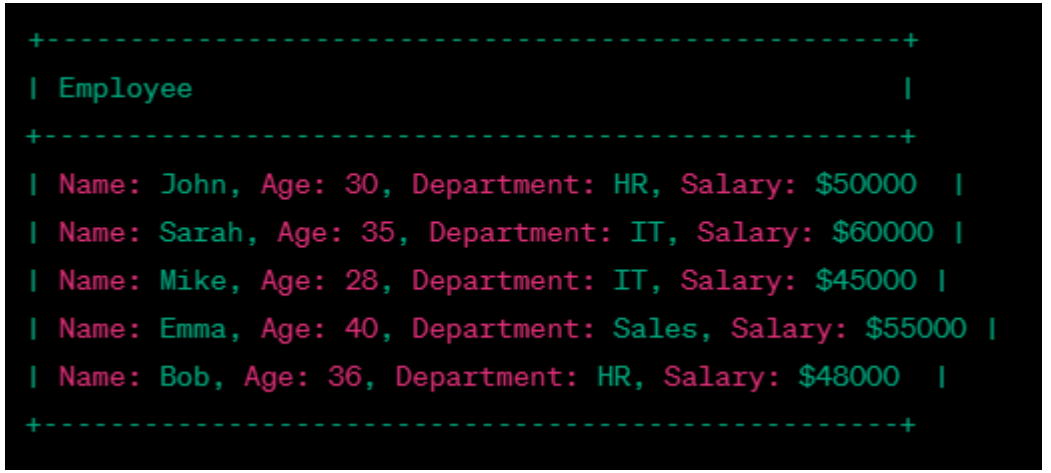
Count

Demo

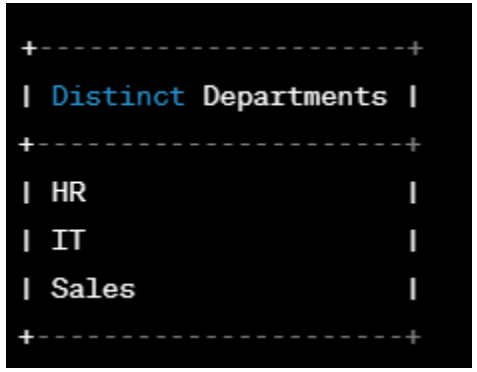
Operaciones de conjunto (Distinct)

- Las operaciones de conjunto en LINQ se refieren a operaciones de consulta que producen un conjunto de resultados que se basa en la presencia o ausencia de elementos equivalentes dentro de la misma colección (o conjunto) o de colecciones separadas.

Operaciones de conjunto (Distinct)




```
var distinctDepartments = employeesList.Select(emp => emp.Department).Distinct();
```

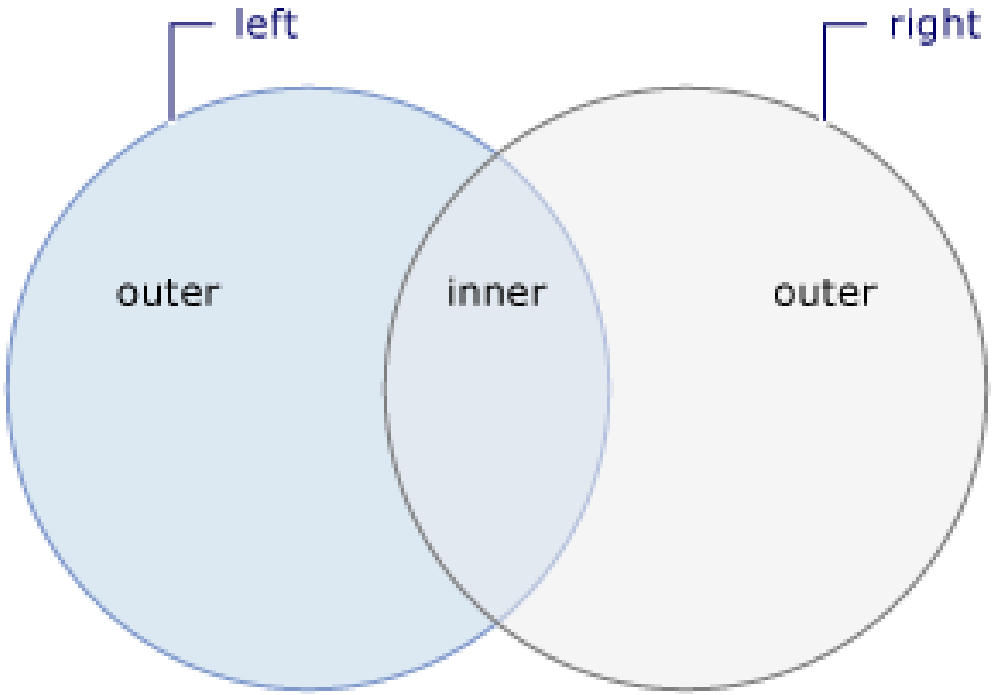


Distinct

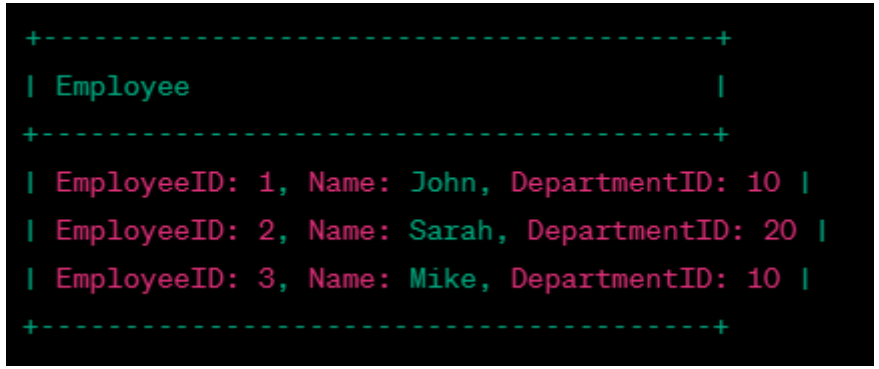
Demo

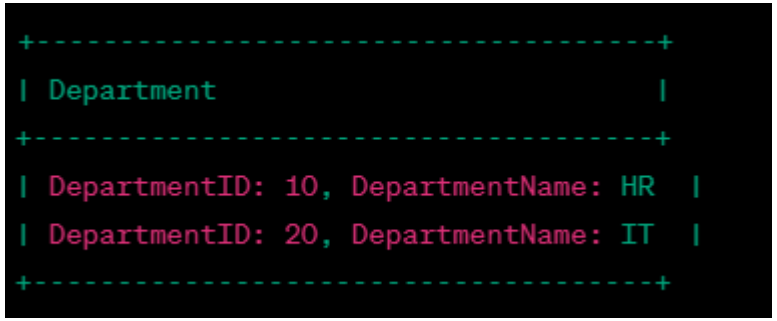
Operaciones Join (Join)

- Las operaciones de conjunto en LINQ se refieren a operaciones de consulta que producen un conjunto de resultados que se basa en la presencia o ausencia de elementos equivalentes dentro de la misma colección (o conjunto) o de colecciones separadas.

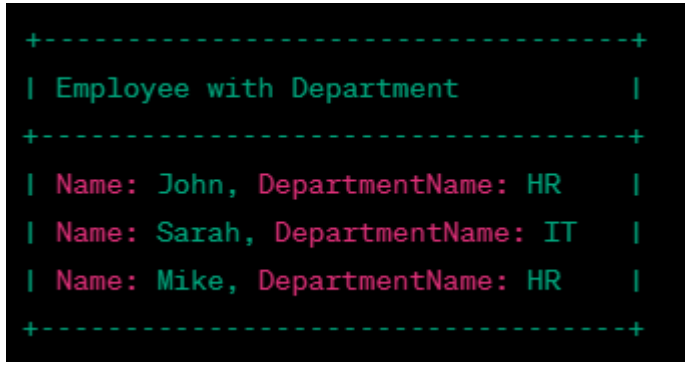


Operaciones Join (Join)






```
var employeeWithDepartment = employeesList.Join(
    departmentsList,
    emp => emp.DepartmentID,
    dept => dept.DepartmentID,
    (emp, dept) =>
       new {
                emp.Name,
                DepartmentName = dept.DepartmentName }
);
```



Join

Demo

LINQ – Operadores de Consulta

Héctor Pérez

Microsoft MVP

Operadores de Consulta

- Sintaxis más parecida a SQL

```
IEnumerable<string> names =
    DBContext.Students.Select((student) => student.FirstName);
```

```
var names =
    from n in DBContext.Students
    select n.FirstName;
```

Operadores de consulta

Demo

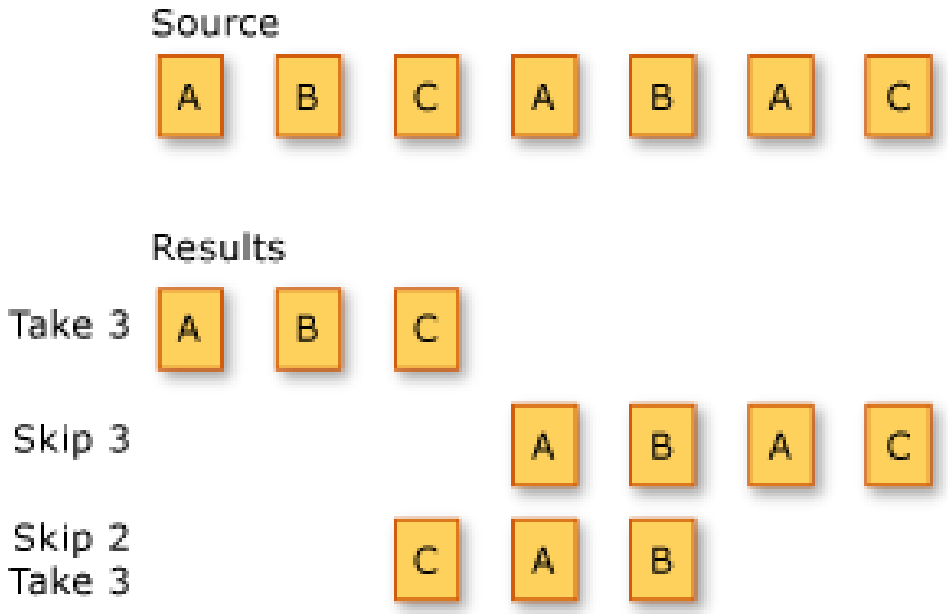
LINQ – Otros métodos útiles

Héctor Pérez

Microsoft MVP

Operaciones de partición (Take)

- La partición en LINQ se refiere a la operación de dividir una secuencia de entrada en dos secciones, sin reordenar los elementos, y luego devolver una de las secciones.



Operaciones de partición (Take)

```
+-------------------------------------------------+
| Employee                                        |
+-------------------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000  |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
| Name: Bob, Age: 36, Department: HR, Salary: $48000  |
| ... (más empleados) ...                         |
+-------------------------------------------------+
```

var firstThreeEmployees = employeesList.Take(3);

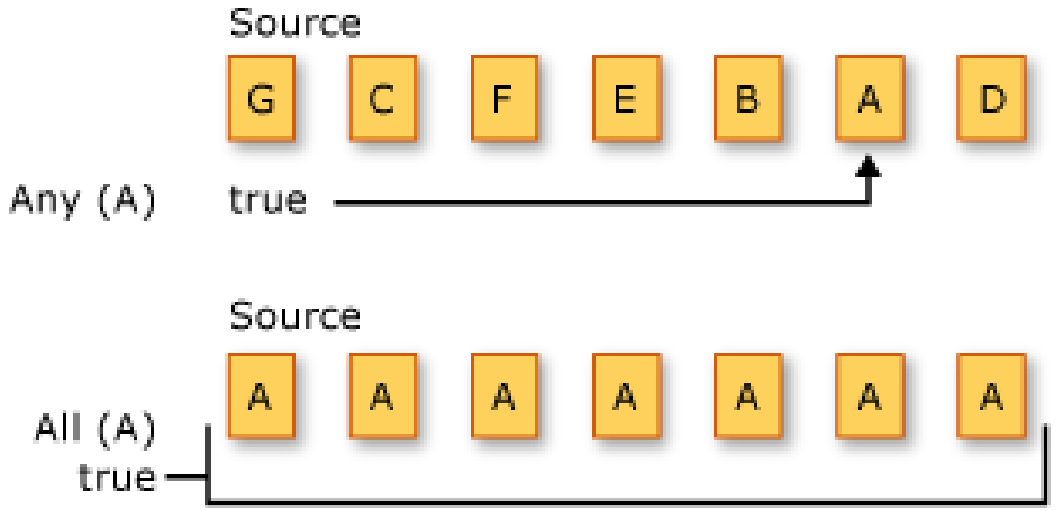
```
+-------------------------------------------------+
| First Three Employees                           |
+-------------------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000  |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
+-------------------------------------------------+
```

Take and Skip

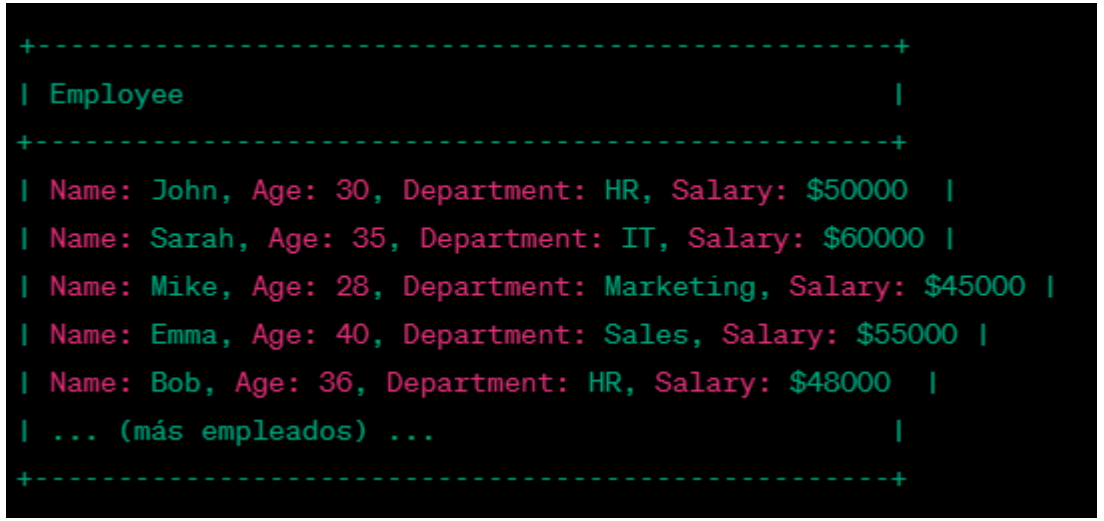
Demo

Operaciones de cuantificación (Any y All)

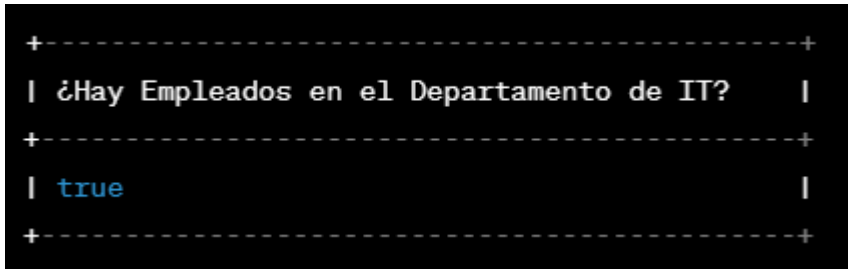
- Las operaciones con cuantificadores devuelven un valor booleano que indica si algunos o todos los elementos de una secuencia cumplen una condición.



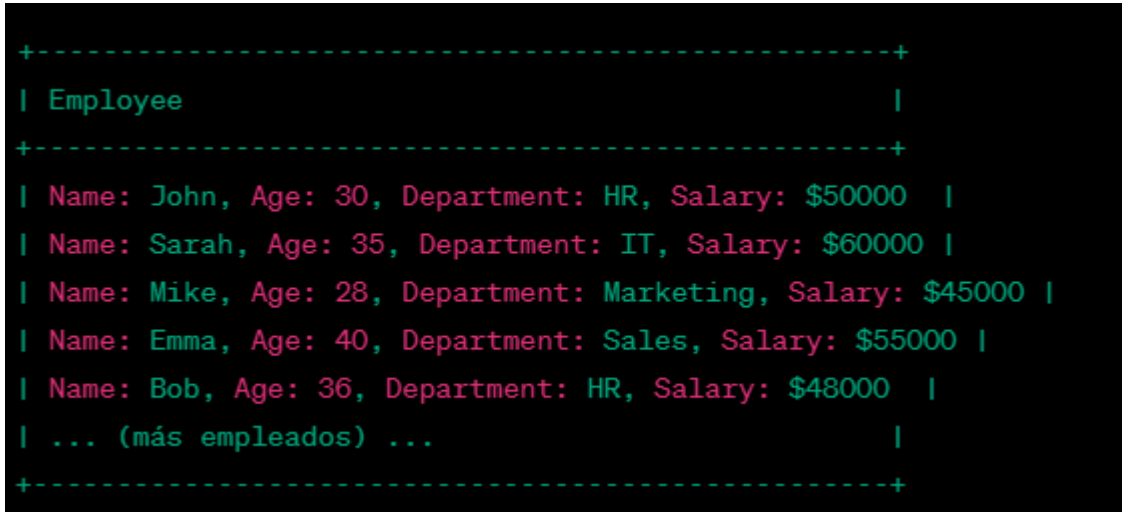
Operaciones de cuantificación (Any y All)




```
bool anyInIT = employeesList.Any(emp => emp.Department == "IT");
```

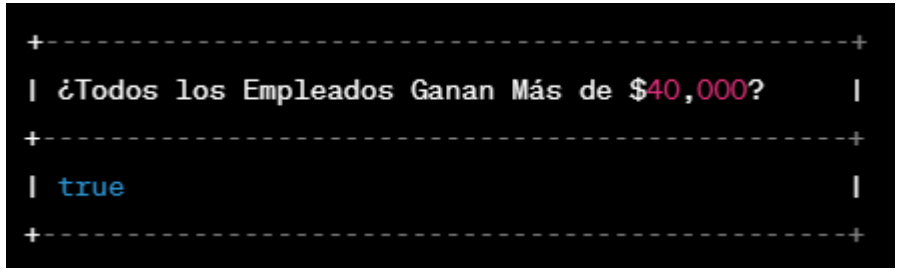


Operaciones de cuantificación (Any y All)



```
+-------------------------------------------------+
| Employee                                        |
+-------------------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000  |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Emma, Age: 40, Department: Sales, Salary: $55000 |
| Name: Bob, Age: 36, Department: HR, Salary: $48000  |
| ... (más empleados) ...                         |
+-------------------------------------------------+
```

```
bool allAbove40K = employeesList.All(emp => emp.Salary > 40000);
```



```
+-------------------------------------------+
| ¿Todos los Empleados Ganan Más de $40,000? |
+-------------------------------------------+
| true                                      |
+-------------------------------------------+
```

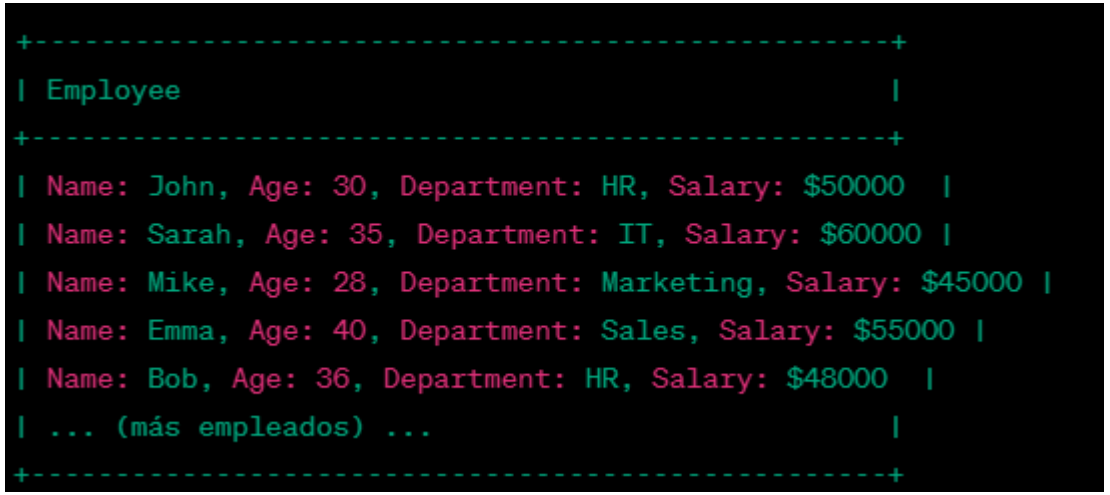
Any and All

Demo

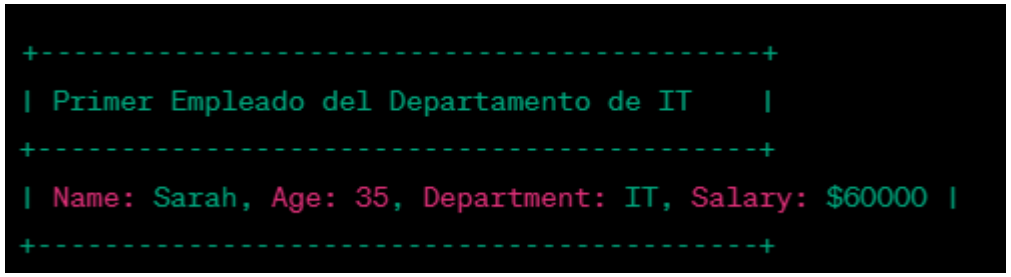
Operaciones de elementos (First y FirstOrDefault)

- Las operaciones con elementos devuelven un único elemento específico de una secuencia.

Operaciones de elementos (First y FirstOrDefault)




```
var firstITEmployee = employeesList.FirstOrDefault(emp => emp.Department == "IT");
```



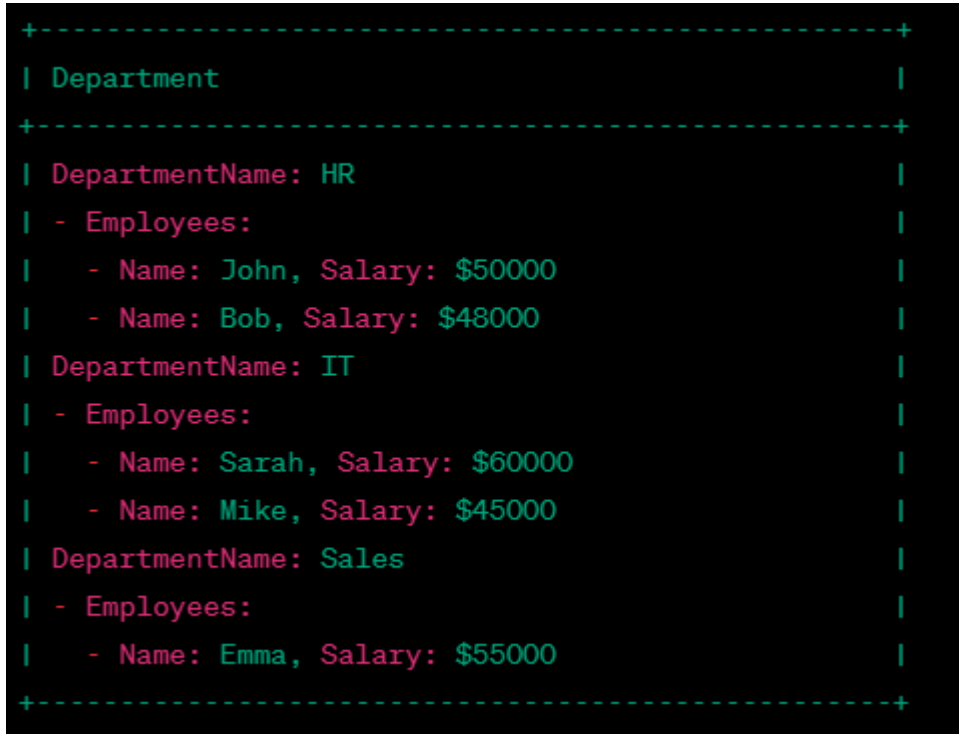
First and FirstOrDefault

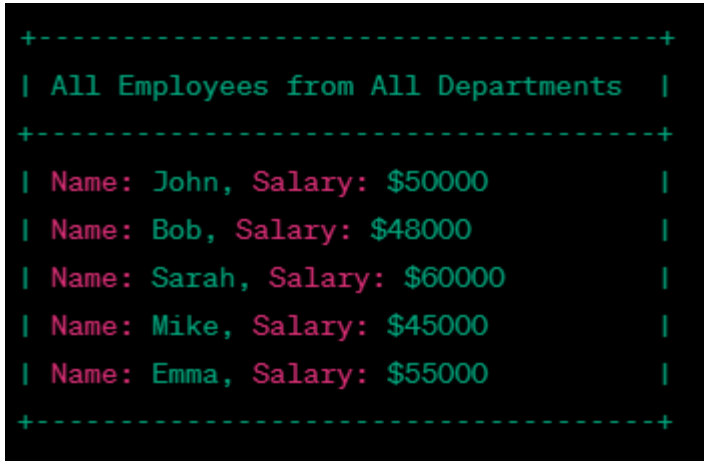
Demo

SelectMany

- Proyecta secuencias de valores que se basan en una función de transformación y luego las aplana en una secuencia.

SelectMany






```
var allEmployees = departmentsList.SelectMany(dept => dept.Employees);
```

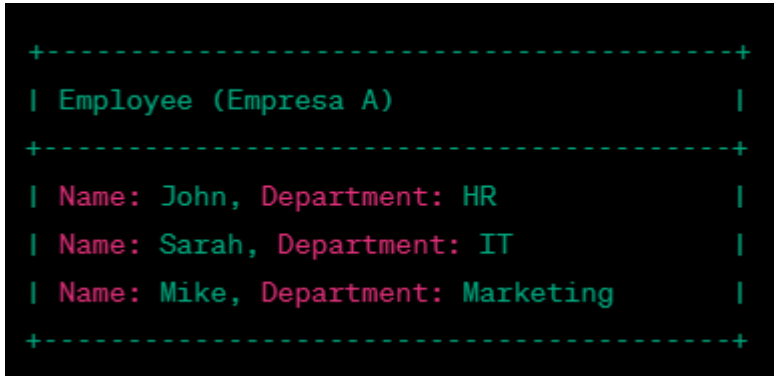
SelectMany

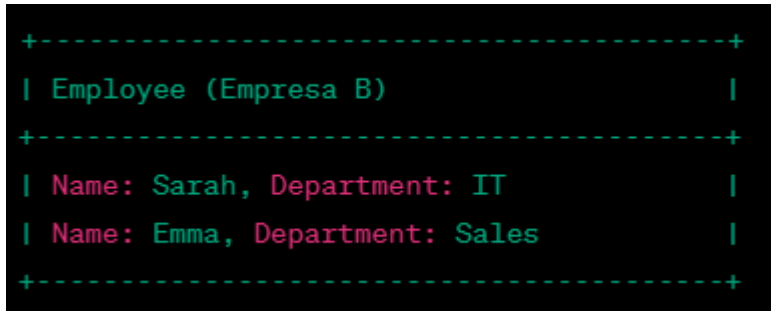
Demo

Except (Set Operation)

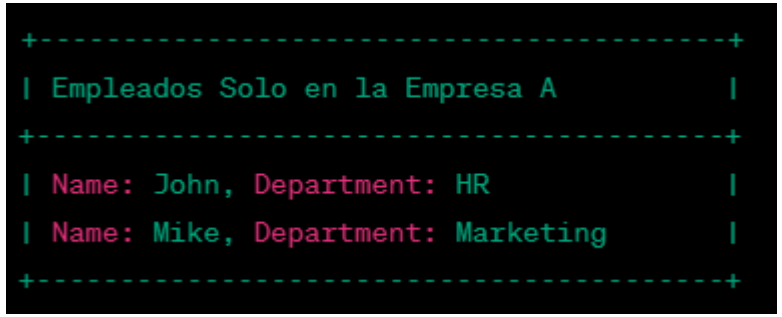
- Devuelve la diferencia de conjuntos, es decir, los elementos de una colección que no aparecen en una segunda colección.

Except (Set Operation)






```
var employeesOnlyInA = employeesListA.Except(employeesListB);
```

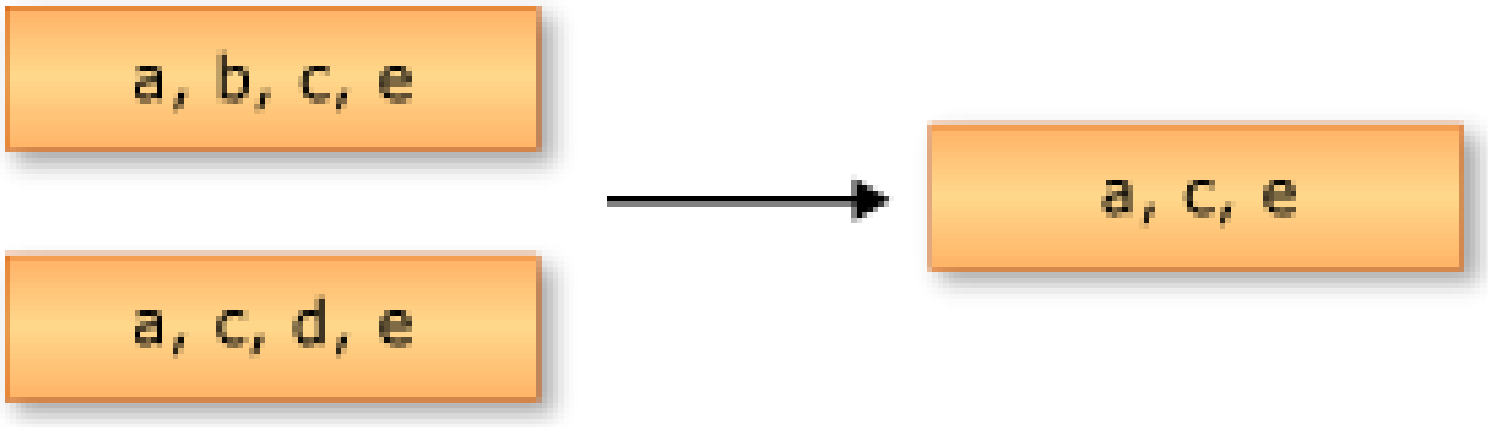


Except

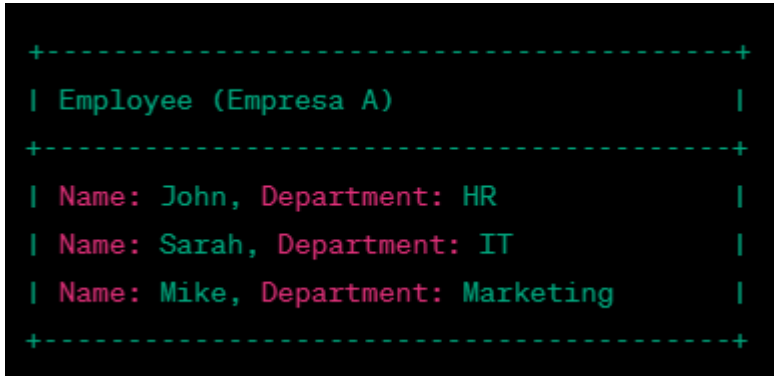
Demo

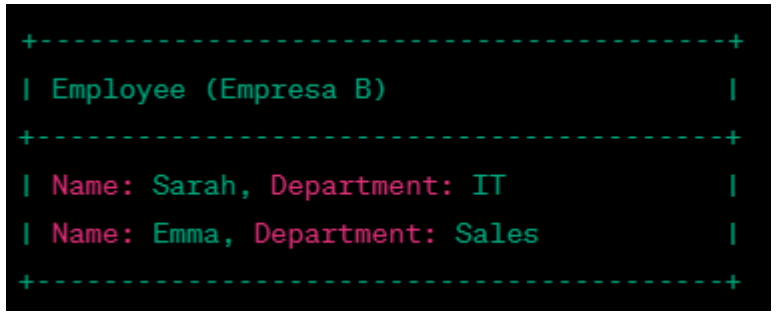
Intersect (Set Operation)

- Devuelve la intersección de conjuntos, es decir, los elementos que aparecen en cada una de dos colecciones.

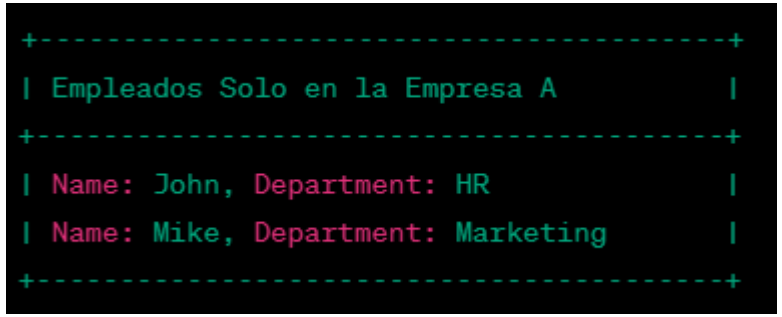


Intersect (Set Operation)






```
var employeesInBoth = employeesListA.Intersect(employeesListB);
```

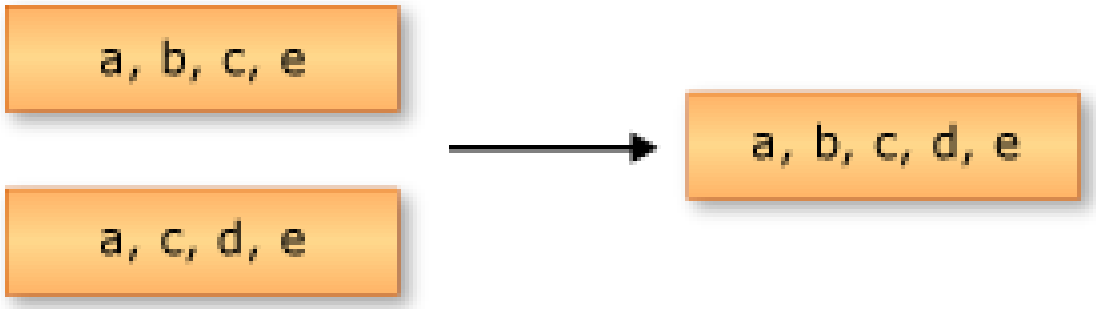


Intersect

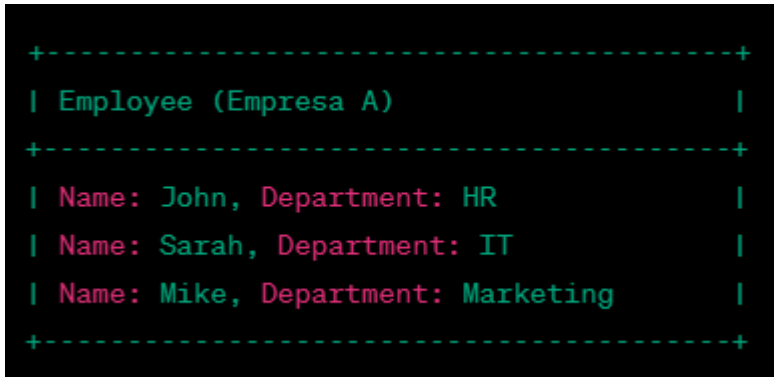
Demo

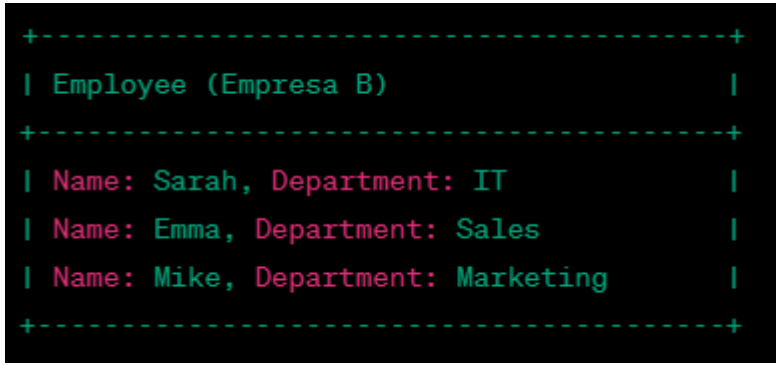
Union (Set Operation)

- Devuelve la unión de conjuntos, es decir, los elementos únicos que aparecen en cualquiera de las dos colecciones.

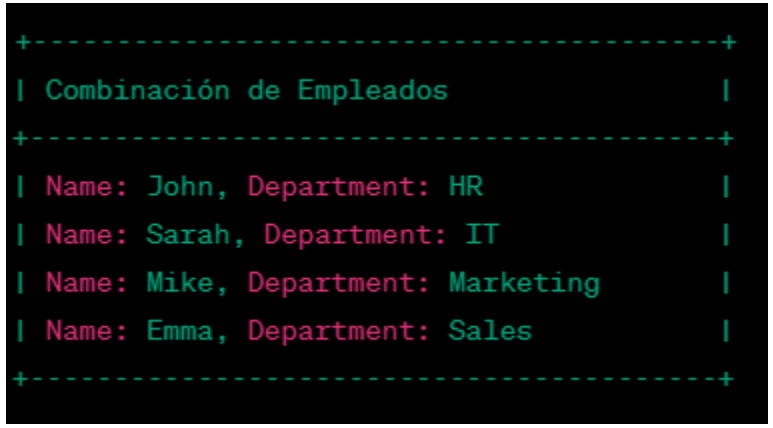


Union (Set Operation)






```
var combinedEmployees = employeesListA.Union(employeesListB);
```



Union

Demo

Reverse (Sorting Data)

- Invierte el orden de los elementos de una colección.

Reverse (Sorting Data)

```
+-------------------------------------------------+
| Employee                                        |
+-------------------------------------------------+
| Name: John, Age: 30, Department: HR, Salary: $50000  |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
+-------------------------------------------------+
```

employeesList.Reverse();

```
+-------------------------------------------------+
| Lista Invertida de Empleados                    |
+-------------------------------------------------+
| Name: Mike, Age: 28, Department: Marketing, Salary: $45000 |
| Name: Sarah, Age: 35, Department: IT, Salary: $60000 |
| Name: John, Age: 30, Department: HR, Salary: $50000  |
+-------------------------------------------------+
```

Reverse

Demo