Для группировки данных по определенным параметрам применяется оператор **group by** и метод **GroupBy()**.

Оператор group by

Допустим, у нас есть набор из объектов следующего типа:

```
record class Person(string Name, string Company);
```

Данный класс представляет пользователя и имеет два свойства: Name (имя пользователя) и Company (компания, где работает пользователь). Сгруппируем набор пользователей по компании:

```
Person[] people =
{
    new Person("Tom", "Microsoft"), new Person("Sam", "Google"),
    new Person("Bob", "JetBrains"), new Person("Mike", "Microsoft"),
    new Person("Kate", "JetBrains"), new Person("Alice", "Microsoft"),
};

var companies = from person in people
                group person by person.Company;

foreach(var company in companies)
{
    Console.WriteLine(company.Key);

    foreach(var person in company)
    {
        Console.WriteLine(person.Name);
    }
    Console.WriteLine(); // для разделения между группами
}

record class Person(string Name, string Company);
```

Если в выражении LINQ последним оператором, выполняющим операции над выборкой, является `group`, то оператор `select` не применяется.

Оператор `group` принимает критерий по которому проводится группировка:

```
group person by person.Company
```

в данном случае группировка идет по свойству Company. Результатом оператора `group` является выборка, которая состоит из групп. Каждая группа представляет объект IGrouping<K, V>: параметр K указывает на тип ключа - тип свойства, по которому идет группировка (здесь это тип string). А параметр V представляет тип сгруппированных объектов - в данном случае группируем объекты Person.

Каждая группа имеет ключ, который мы можем получить через свойство Key: `g.Key`. Здесь это будет название компании.

Все элементы внутри группы можно получить с помощью дополнительной итерации. Элементы группы имеют тот же тип, что и тип объектов, которые передавались оператору `group`, то есть в данном случае объекты типа `Person`.

В итоге мы получим следующий вывод:

```
Microsoft
Tom
Mike
Alice

Google
Sam

JetBrains
Bob
Kate
```

GroupBy

В качестве альтернативы можно использовать метод расширения **GroupBy**. Он имеет ряд перегрузок, возьмем самую простую из них:

```
GroupBy<TSource,TKey> (Func<TSource,TKey> keySelector);
```

Данная версия получает делегат, которые в качестве параметра принимает каждый элемент коллекции и возвращает критерий группировки.

Перепишем предыдущий пример с помощью метода GroupBy:

```
Person[] people =
{
    new Person("Tom", "Microsoft"), new Person("Sam", "Google"),
    new Person("Bob", "JetBrains"), new Person("Mike", "Microsoft"),
    new Person("Kate", "JetBrains"), new Person("Alice", "Microsoft"),
};

var companies = people.GroupBy(p => p.Company);

foreach(var company in companies)
{
    Console.WriteLine(company.Key);

    foreach(var person in company)
    {
        Console.WriteLine(person.Name);
    }
    Console.WriteLine(); // для разделения между группами
}

record class Person(string Name, string Company);
```

Создание нового объекта при группировке

Теперь изменим запрос и создадим из группы новый объект:

```
Person[] people =
{
    new Person("Tom", "Microsoft"), new Person("Sam", "Google"),
    new Person("Bob", "JetBrains"), new Person("Mike", "Microsoft"),
    new Person("Kate", "JetBrains"), new Person("Alice", "Microsoft"),
};

var companies = from person in people
                group person by person.Company into g
                select new { Name = g.Key, Count = g.Count() }; ;

foreach(var company in companies)
{
    Console.WriteLine($"{company.Name} : {company.Count}");
}

record class Person(string Name, string Company);
```

Выражение

```
group person by person.Company into g
```

определяет переменную **g**, которая будет содержать группу. С помощью этой переменной мы можем затем создать новый объект анонимного типа (хотя также можно под данную задачу определить новый класс):

```
select new { Name = g.Key, Count = g.Count() }
```

Теперь результат запроса LINQ будет представлять набор объектов таких анонимных типов, у которых два свойства Name и Count.

Результат программы:

```
Microsoft : 3
Google : 1
JetBrains : 2
```

Аналогичная операция с помощью метода `GroupBy()`:

```
var companies = people
                    .GroupBy(p=>p.Company)
                    .Select(g => new { Name = g.Key, Count = g.Count() });
```

Вложенные запросы

Также мы можем осуществлять вложенные запросы:

```
Person[] people =
{
    new Person("Tom", "Microsoft"), new Person("Sam", "Google"),
    new Person("Bob", "JetBrains"), new Person("Mike", "Microsoft"),
    new Person("Kate", "JetBrains"), new Person("Alice", "Microsoft"),
};
var companies = from person in people
                group person by person.Company into g
                select new
                {
                    Name = g.Key,
                    Count = g.Count(),
                    Employees = from p in g select p
                };

foreach (var company in companies)
{
    Console.WriteLine($"{company.Name} : {company.Count}");
    foreach(var employee in company.Employees)
    {
        Console.WriteLine(employee.Name);
    }
    Console.WriteLine(); // для разделения компаний
}

record class Person(string Name, string Company);
```

Здесь свойство Employees каждой группы формируется с помощью дополнительного запроса, который выбирает всех пользователей в этой группе. Консольный вывод программы:

```
Microsoft : 3
Tom
Mike
Alice

Google : 1
Sam

JetBrains : 2
Bob
Kate
```

Аналогичный запрос с помощью метода GroupBy:

```
var companies = people
                    .GroupBy(p=>p.Company)
                    .Select(g => new
                    {
                        Name = g.Key,
                        Count = g.Count(),
                        Employees = g.Select(p=> p)
                    });
```