How to “let” in lambda expression?

Asked 7 years, 10 months ago Active 2 months ago Viewed 29k times

40

13

How can I rewrite this linq query to Entity on with lambda expression?
I want to use *let* keyword or an equivalent in my lambda expression.

```
var results = from store in Stores
              let AveragePrice =  store.Sales.Average(s => s.Price)
              where AveragePrice < 500 && AveragePrice > 250
```

For some similar questions like what is commented under my question, it's suggested to

```
.Select(store=> new { AveragePrice = store.Sales.Average(s => s.Price), store})
```

which will calculate AveragePrice for each item, while in Query style I mentioned, *let* expression prevents to calculate average many times.

linq entity-framework c#-4.0 lambda let

edited Feb 11 '12 at 14:29

asked Feb 11 '12 at 12:55

Reza Owliaei
2,833 4 26 49

2 possible duplicate of Code equivalent to the 'let' keyword in chained LINQ extension method calls – Eranga Feb 11 '12 at 13:04

@Eranga: I that Question, Marc had select animalName.Length for each item. Here, I don't want to calculate Average of all items, for every item. – Reza Owliaei Feb 11 '12 at 14:24

1 @Reza: the average is computed just once per store object, exactly as in your query... – digEmAll Feb 11 '12 at 14:49

4 Answers

46

So, you can use the extension method syntax, which would involve one lambda expression more than you are currently using. There is no `let`, you just use a multi-line lambda and declare a variable:

```
var results = Stores.Where(store =>
{
    var averagePrice = store.Sales.Average(s => s.Price);
    return averagePrice > 250 && averagePrice < 500;
});
```

Note that I changed the average price comparison, because yours would never return any results (more than 500 AND less that 250).

The alternative is

```
var results = Stores.Select(store => new { Store = store, AveragePrice =
store.Sales.Average(s => s.Price})
    .Where(x => x.AveragePrice > 250 && x.AveragePrice < 500)
    .Select(x => x.Store);
```

edited Feb 11 '12 at 13:09

answered Feb 11 '12 at 13:02

Jay
48.7k 8 88 114

6 Error: A lambda expression with a statement body cannot be converted to an expression tree. – Reza Owliaei Feb 11 '12 at 14:09

5 Your first offer just works on memory and could not be used in LINQ providers like EF. A lambda expression with a statement body cannot be converted to an expression tree. – Amir Karimi Feb 11 '12 at 14:12

2 @amkh: I'm almost sure EF was not mentioned in the first version of the question. Or at least neither Jay nor me have noticed that... – digEmAll Feb 11 '12 at 14:44

1 @Reza: the average is computed just once per store object, exactly as in your query... – digEmAll Feb 11 '12 at 14:48

2 @Reza No, I wouldn't expect it to necessarily improve EF query performance. The answer is just about how to do a `let`-like operation with the extension method syntax. Why not just stick with the query syntax for this? – Jay Feb 15 '12 at 17:01

Basically, you need to use Select and an anonymous type to add the average to your object, followed by the rest of your statement.

Not tested but it should look like this:

24

```
Stores.Select(
x => new { averagePrice = x.Sales.Average(s => s.Price), store = x})
.Where(y => y.averagePrice > 500 && y.averagePrice < 250)
.Select(x => x.store);
```

Warning, be careful with these constructs. Using let creates a new anonymous type per object in your collection, it consumes a lot of memory with large collections ...

Look here for details: Let in chained extension methods

edited May 23 '17 at 11:33 — Community ♦ — 1 1

answered Feb 11 '12 at 13:03 — Yoeri — 1,742 12 22

1 It does not consume *any* memory if those are LINQ-to-Entities queries or similar. – svick Feb 11 '12 at 15:04

2 indeed, only for linq-to-objects it is. Thx – Yoeri Feb 12 '12 at 12:57

3 This should be the accepted answer, as it lets you use your variable across different "clauses". – alexbchr Mar 5 '14 at 13:19

4

Another option is to define this extension method:

```
public static class Functional
{
    public static TResult Pipe<T, TResult>(this T value, Func<T, TResult> func)
    {
        return func(value);
    }
}
```

Then write your query like this:

```
var results = Stores
    .Where(store => store.Sales.Average(s => s.Price)
        .Pipe(averagePrice => averagePrice < 500 && averagePrice > 250));
```

edited Aug 17 '15 at 19:36

answered Aug 7 '13 at 17:35 — Timothy Shields

56.9k 16 98 149

1

We can avoid the overhead of the lambda used in all the other answers with an inline out declaration:

```
public static class FunctionalExtensions
{
    public static T Assign<T>(this T o, out T result) =>
        result = o;
}
```

And call it like this

```
var results = Stores
    .Where(store => store.Sales
        .Average(s => s.Price)
        .Assign(out var averagePrice) < 500 && averagePrice > 250);
```

answered Oct 5 at 15:12

daw

731 7 14