Dynamic WHERE clause in LINQ

Ask Question

51

22

What is the best way to assemble a dynamic WHERE clause to a LINQ statement?

I have several dozen checkboxes on a form and am passing them back as: Dictionary<string, List<string>> (Dictionary<fieldName,List<values>>) to my LINQ query.

```
public IOrderedQueryable<ProductDetail> GetProductList(string productGroupName, string
productTypeName, Dictionary<string,List<string>> filterDictionary)
{
    var q = from c in db.ProductDetail
            where c.ProductGroupName == productGroupName && c.ProductTypeName ==
productTypeName
            // insert dynamic filter here
            orderby c.ProductTypeName
            select c;
    return q;
}
```

c# linq dynamic where-clause

edited Feb 18 '15 at 4:46

abatishchev

70.9k 70 267 397

asked May 11 '09 at 14:34

Keith Barrows

14.5k 25 70 120

52

```
Dim Northwind As New NorthwindDataContext

Dim query = Northwind.Products _
                     .Where("CategoryID=2 And UnitPrice>3") _
                     .OrderBy("SupplierId")

GridView1.DataSource = query
GridView1.DataBind()
```

(source: scottgu.com)

You need something like this? Use the Linq Dynamic Query Library (download includes examples).

Check out ScottGu's blog for more examples.

edited Mar 16 at 8:00

Glorfindel
16.9k 11 52 74

answered May 11 '09 at 14:39

Thomas Stock
6,864 11 53 76

1 There is a ported version on github (github.com/kahanu/System.Linq.Dynamic), which I contribute to and help manage. – Ryan Gates Mar 31 '16 at 21:29

13

You can also use the PredicateBuilder from LinqKit to chain multiple typesafe lambda expressions using Or or And.

http://www.albahari.com/nutshell/predicatebuilder.aspx

answered May 13 '09 at 1:08

Linus
841 10 22

9

I have similar scenario where I need to add filters based on the user input and I chain the where clause.

Here is the sample code.

```
var votes = db.Votes.Where(r => r.SurveyID == surveyId);
if (fromDate != null)
{
    votes = votes.Where(r => r.VoteDate.Value >= fromDate)
}
if (toDate != null)
{
    votes = votes.Where(r => r.VoteDate.Value <= toDate);
}
votes = votes.Take(LimitRows).OrderByDescending(r => r.Vot
```

answered Jul 30 '14 at 0:28

Xavier John

3,626 3 19 26

Best suited for my need and easy to use. Thank you. – user6121177 Aug 24 '17 at 13:56

8

A simple Approach can be if your Columns are of Simple Type like String

```
public static IEnumerable<MyObject> WhereQuery(IEnumerable
columnName, string propertyValue)
{
   return source.Where(m => { return m.GetType().GetProper
null).ToString().StartsWith(propertyValue); });
}
```

edited Jan 19 '17 at 13:17

answered Mar 20 '13 at 9:34

Nitin Bourai
313 5 18

5

I came up with a solution that even I can understand... by using the 'Contains' method you can chain as many WHERE's as you like. If the WHERE is an empty string, it's ignored (or evaluated as a select all). Here is my example of joining 2 tables in LINQ, applying multiple where clauses and populating a model class to be returned to the view. (this is a select all).

```
public ActionResult Index()
    {
        string AssetGroupCode = "";
        string StatusCode = "";
        string SearchString = "";

        var mdl = from a in _db.Assets
                  join t in _db.Tags on a.ASSETID equals t
                  where a.ASSETGROUPCODE.Contains(AssetGro
                  && a.STATUSCODE.Contains(StatusCode)
                  && (
                  a.PO.Contains(SearchString)
                  || a.MODEL.Contains(SearchString)
                  || a.USERNAME.Contains(SearchString)
                  || a.LOCATION.Contains(SearchString)
                  || t.TAGNUMBER.Contains(SearchString)
                  || t.SERIALNUMBER.Contains(SearchString)
                  )
                  select new AssetListView
                  {
                      AssetId = a.ASSETID,
                      TagId = t.TAGID,
                      PO = a.PO,
                      Model = a.MODEL,
                      UserName = a.USERNAME,
```

```
                    };

            return View(mdl);
        }
```

edited Feb 28 '14 at 23:26

answered Feb 28 '14 at 22:51

mike
51 1 3

2

I had same question (User defined filter for linq), and @tvanfosson told me about Dynamic Linq (http://code.msdn.microsoft.com/csharpsamples).

edited May 23 '17 at 11:46

Community ♦
1 1

answered May 11 '09 at 14:40

TcKs
21.3k 6 54 93

1

You could use the Any() extension method. The following seems to work for me.

```
XStreamingElement root = new XStreamingElement("Results",
                from el in StreamProductItem(file)
                where fieldsToSearch.Any(s => el.Element(s
```

```
);
Console.WriteLine(root.ToString());
```

Where 'fieldsToSearch' and 'fieldsToReturn' are both List objects.

answered Aug 27 '13 at 17:52

Todd DeLand
1,933 1 17 13

1

This project on CodePlex have what you want.

System.Linq.Dynamic - http://dynamiclinq.codeplex.com/

> **Project Description**
>
> Extends System.Linq.Dynamic to support Execution of Lambda expressions defined in a string against Entity Framework or any provider that supports IQueryable.

As it is an extension of the source code you can find on Scott Guthrie's Blog it will allow you to do things like this:

```
NorthwindDataContext northwind = new NorthwindDataContext();
var result = northwind.Products
                      .Where("CategoryID = 3 AND UnitPrice > 3")
                      .OrderBy("SupplierID");
result.ToArray();
```

And things like this:

```
NorthwindDataContext northwind = new NorthwindDataContext();
IQueryable<Product> queryableData = northwind.Products.AsQueryable<Product>();
string query = "Products.Where(Product => (Product.CategoryID = 3 And (Product.UnitPrice > 10))).Take(10)";
var externals = new Dictionary<string, object>();
externals.Add("Products", queryableData);
var expression = System.Linq.Dynamic.DynamicExpression.Parse(typeof(IQueryable<Product>), query, new[] { externals });
var result = queryableData.Provider.CreateQuery<Product>(expression);
result.ToArray();
```

answered Sep 21 '13 at 20:03

Zignd
2,747 11 29 53

1

This is the solution I came up with if anyone is interested.

https://kellyschronicles.wordpress.com/2017/12/16/dynamic-predicate-for-a-linq-query/

First we identify the single element type we need to use (Of TRow As DataRow) and then identify the “source” we are using and tie the identifier to that source ((source As TypedTableBase(Of TRow)). Then we must specify the predicate, or the WHERE clause that is going to be passed (predicate As Func(Of TRow, Boolean)) which will either be returned as true or false. Then we identify how we want the returned information ordered (OrderByField As String). Our function will then return a EnumerableRowCollection(Of TRow), our collection of datarows that have met the conditions of our predicate(EnumerableRowCollection(Of TRow)). This is a basic example. Of course you must make sure your order field doesn’t contain nulls, or have handled that situation properly and make sure your column names (if you are using a strongly typed datasource never mind this, it will rename the columns for you) are standard.

edited Dec 17 '17 at 18:59

answered Dec 17 '17 at 18:53

KJM

A link to a solution is welcome, but please ensure your answer is useful without it: add context around the link so your fellow users will have some idea what it is and why it's there, then quote the most relevant part of the page you're linking to in case the target page is unavailable. Answers that are little more than a link may be deleted. – FelixSFD Dec 17 '17 at 18:56

I do apologize. I am new here. – KJM Dec 17 '17 at 18:59

0

It seems much simpler and simpler to use the ternary operator to decide dynamically if a condition is included

List productList = new List();

```
        productList =
                db.ProductDetail.Where(p => p.ProductDetai
                && (String.IsNullOrEmpty(iproductGroupName
(p.iproductGroupName.Equals(iproductGroupName)) ) //use te
condition dynamic
                && (ID == 0 ? (true) : (p.ID == IDParam))
                ).ToList();
```

answered Feb 22 at 0:15

Josué Camacho

1