Does LINQ-to-SQL Support Composable Queries?

Asked 16 years, 3 months ago Modified 16 years, 3 months ago Viewed 1k times

5

Speaking as a non-C# savvy programmer, I'm curious as to the evaluation semantics of LINQ queries like the following:

```
var people = from p in Person
             where p.age < 18
             select p

var otherPeople = from p in people
                  where p.firstName equals "Daniel"
                  select p
```

Assuming that `Person` is an ADO entity which defines the `age` and `firstName` fields, what would this do from a database standpoint? Specifically, would the `people` query be run to produce an in-memory structure, which would then be queried by the `otherPeople` query? Or would the construction of `otherPeople` merely pull the data regarding the query from `people` and then produce a new database-peered query? So, if I iterated over both of these queries, how many SQL statements would be executed?

sql linq linq-to-sql code-reuse

Share Improve this question Follow

asked Sep 18, 2008 at 1:28

Daniel Spiewak
55.1k ● 14 ● 111 ● 120

5 Answers

Sorted by: Highest score (default)

12

They are composable. This is possible because LINQ queries are actually expressions (code as data), which LINQ providers like LINQ-to-SQL can evaluate and generate corresponding SQL.

Because LINQ queries are lazily evaluated (e.g. won't get executed until you iterate over the elements), the code you showed won't actually touch the database. Not until you iterate over otherPeople or people will SQL get generated and executed.

Share Improve this answer Follow

answered Sep 18, 2008 at 1:34


Judah Gabriel Himango
60k ● 39 ● 161 ● 215

4

```
var people = from p in Person
             where p.age < 18
             select p
```

Translates to:

```
SELECT [t0].[PersonId], [t0].[Age], [t0].[FirstName]
FROM [dbo].[Person] AS [t0]
WHERE [t0].[Age] < @p0
```

where @p0 gets sent through as 18

```
var otherPeople = from p in people
                  where p.firstName equals "Daniel"
                  select p
```

Translates to:

```
SELECT [t0].[PersonId], [t0].[Age], [t0].[FirstName]
FROM [dbo].[Person] AS [t0]
WHERE [t0].[FirstName] = @p0
```

where @p0 gets sent through as "Daniel"

```
var morePeople = from p1 in people
                 from p2 in otherPeople
                 where p1.PersonId == p2.PersonId
                 select p1;
```

Translates to:

```
SELECT [t0].[PersonId], [t0].[Age], [t0].[FirstName]
FROM [dbo].[Person] AS [t0], [dbo].[Person] AS [t1]
WHERE ([t0].[PersonId] = [t1].[PersonId]) AND ([t0].[Age] < @p0) AND ([t1].
[FirstName] = @p1)
```

where @p0 is 18, @p1 is "Daniel"

When in doubt, call the ToString() on your IQueryable or give a TextWriter to the DataContext's Log property.

answered Sep 18, 2008 at 4:22

Ant

1,156 ● 11 ● 15

3

Yes, the resulting query is composed. It includes the full where clause. Turn on SQL profiling and try it to see for yourself.

Linq does this through expression trees. The first linq statement produces an expression tree; it doesn't execute the query. The second linq statement builds on the expression tree created by the first. The statement is only executed when you enumerate the resulting collection.

Share Improve this answer Follow

answered Sep 18, 2008 at 1:35

Michael L Perry

7,435 ● 3 ● 38 ● 34

1

`people` and `otherPeople` contain objects of type `IQueryable<Person>`.

If you iterate over both, separatly, it will run two queries. If you only iterate over `otherPeople`, it will run the expected query, with two where clauses.

If you do `.ToList()` on `people` and use the returned `List<Person>` in the second query instead of people, it becomes LINQ-to-Objects and no SQL is executed.

This behavior is referred to as deferred execution. Meaning no query is done until it is needed. Before execution they are just expression trees that get manipulated to formulate the final query.

Share

Improve this answer

Follow

edited Sep 18, 2008 at 1:38

answered Sep 18, 2008 at 1:33

David Thibault

8,736 ● 3 ● 39 ● 51

0

Both these queries will be executes when you'll try to access final results. You can try to view original SQL generated from DataContext object properties.

Share Improve this answer Follow

answered Sep 18, 2008 at 1:33

dimarzionist

18.7k ● 4 ● 24 ● 23