Performance Tuning in EF

# Auto-Compiling LINQ Queries

#### Problem

You want to improve the performance of queries that are reused several times, and you would like to achieve this performance upgrade with no additional coding or configuration.

When executing against a database, Entity Framework must *translate* your strongly typed LINQ query to a corresponding SQL query, based upon your database provider (SQL Server, Oracle, and so on). Beginning with version 5 of Entity Framework, each query translation is cached by default. This process is referred to as *auto-caching*. With each subsequent execution of a given LINQ query, the corresponding SQL query is retrieved directly from query plan cache, bypassing the translation step. For queries containing parameters, changing parameter values will still retrieve the same query. Interestingly, this query plan cache is shared among all instances of a context object instantiated with the application's AppDomain, meaning that, once cached, any context object in the AppDomain has access to it.

In [Listing 13-10](http://viewer.books24x7.com/assetviewer.aspx?bkid=59088&destid=1533#1533), we compare performance with caching enabled and then disabled. To illustrate the performance benefit, we've instrumented the code in [Listing 13-10](http://viewer.books24x7.com/assetviewer.aspx?bkid=59088&destid=1533#1533) to print the number of ticks for each of ten iterates taken for both the uncompiled and the compiled versions of the LINQ query. In this query, we can see that we get roughly a 2X performance boost. Most of this, of course, is due to the relatively high cost of compiling versus the low cost for actually performing this simple query

#### How It Works

When you execute a LINQ query, Entity Framework builds an expression tree object for the query, which is then converted, or compiled, into an internal command tree. This internal command tree is passed to the database provider to be converted into the appropriate database commands (typically SQL). The cost of converting an expression tree can be relatively expensive depending on the complexity of the query and the underlying model. Models with deep inheritance or horizontal splitting introduce enough complexity in the conversion process that the compile time may become significant relative to the actual query execution time. However, in Version 5 of the Entity Framework, automatic query caching for LINQ queries was introduced. You can get an idea of the performance benefits of this feature by examining the results of [Listing 13-20](http://viewer.books24x7.com/assetviewer.aspx?bookid=59088&chunkid=808579635).

Additionally, as shown in [Listing 13-20](http://viewer.books24x7.com/assetviewer.aspx?bookid=59088&chunkid=808579635), you can disable the auto-compiling features by dropping down from the DbContext object into the underlying ObjectContext object, obtaining a reference to the entity object and setting its EnablePlanCaching property to false.

To track each compiled query, Entity Framework walks the nodes of the query expression tree and creates a hash, which becomes the key for that compiled query in the underlying query cache. For each subsequent call, Entity Framework will attempt to locate the hash key from the cache, eliminating the overhead cost of the query translation process. It's important to note that the cached query plan is independent of the context object, instead being tied to the AppDomain of the application, meaning that the cached query is available to all instances of a given Entity Framework context object.

Once the underlying query cache contains 800 or more query plans, a cache eviction process automatically kicks off. Each minute, a sweeping process removes entries based upon a LFRU (least frequently/recently used) algorithm, driven by hit count and age of the query.

Compiled queries are especially helpful in ASP.NET search page scenarios where parameter values may change, but the query is the same and can be reused on each page rendering. This works because a compiled query is *parameterized*, meaning that it can accept different parameter values.