Using the WITH RECOMPILE option in a Stored Procedure

Posted At : April 13, 2005 4:07 PM | Posted By : Mark Kruger
Related Categories: SQL tips

This is an excellent article by Arthur Fuller that was sent as a **Builder.com** newsletter. If you've ever wondered why a stored procedure doesn't save quite as much time as you expected - or why equivelent query code can even be faster - this may be the answer.

STYLE type=text/css> .normalCourier { FONT-SIZE: 13px; FONT-FAMILY: Courier } .titleArialBold { FONT-WEIGHT: bold; FONT-SIZE: 18px; FONT-FAMILY: Arial, Helvetica, sans-serif } .normalArial { FONT-SIZE: 13px; FONT-FAMILY: Arial, Helvetica, sans-serif } .normalArialItalic { FONT-SIZE: 13px; FONT-STYLE: italic; FONT-FAMILY: Arial, Helvetica, sans-serif } .smallArial { FONT-SIZE: 11px; FONT-FAMILY: Arial, Helvetica, sans-serif } .smallArialWhiteUnderline { FONT-SIZE: 11px; COLOR: #ffffff; FONT-FAMILY: Arial, Helvetica, sans-serif; TEXT-DECORATION: underline } .smallArialWhite { FONT-SIZE: 11px; COLOR: #ffffff; FONT-FAMILY: Arial, Helvetica, sans-serif } .subhead1 { FONT-WEIGHT: bold; FONT-SIZE: 16px; FONT-FAMILY: Arial, Helvetica, sans-serif } .smallVerdanaBoldWhite { FONT-WEIGHT: bold; FONT-SIZE: 11px; COLOR: #ffffff; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif } .smallVerdanaBold { FONT-WEIGHT: bold; FONT-SIZE: 11px; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif } .smallArialBoldOrange { FONT-WEIGHT: bold; FONT-SIZE: 11px; COLOR: #ffcc66; FONT-FAMILY: Arial, Helvetica, sans-serif } .contentborder { BORDER-RIGHT: #000 1px solid; BORDER-LEFT: #000 1px solid; BACKGROUND-COLOR: #f0f0df } #sky { BORDER-RIGHT: #e6e6e6 4px solid; PADDING-RIGHT: 1px; BORDER-TOP: #e6e6e6 4px solid; FLOAT: right; BORDER-LEFT: #e6e6e6 4px solid; BORDER-BOTTOM: #e6e6e6 4px solid; BACKGROUND-COLOR: #e6e6e6 } #title { BACKGROUND-COLOR: #66717d } #contentcopy { MARGIN: 2px 8px }

The *WITH RECOMPILE* option in MS SQL

Understanding the WITH RECOMPILE option

The generally accepted wisdom about stored procedures (or sprocs) is that because SQL can optimize and compile them, they run more quickly than the equivalent SQL statements executed from Query Analyzer (or perhaps passed in from some front-end app such as a Web page or VB program). This is true, as far as it goes; the trouble is, it doesn't go very far.

To understand this, you need to know what SQL Server does with a new sproc. At creation time, it checks the syntax. If it doesn't find any errors, then it adds the sproc to the system tables: sysobjects, sysdepends, and syscomments (the latter stores the body of the sproc). By default, it doesn't compile the sproc at creation time.

Upon first execution of the sproc, SQL Server optimizes and compiles it. This is when SQL Server devises a query plan and stores it in its procedure cache. On subsequent invocations, SQL looks in the cache first, finds the sproc there, and doesn't compile it. If the sproc isn't in the cache, then SQL Server compiles it and places it in the cache.

My experience with the WITH RECOMPILE option

A while back, I was supporting a search page that allowed its users to search by any of several columns. Then the page called a sproc, passing a parameter to indicate which column to search. I examined the parameter using a CASE block, and then executed

one of several queries, depending upon the column to search.

I knew something was wrong when I began to test my allegedly clever stored procedure. In theory, the performance of each search should at least be approximately the same, but that isn't what happened. When I performed multiple searches, regardless of the order, the first would be fast, and subsequent searches were much slower.

Finally, I realized that the first time the procedure was called, a query plan was devised and stored in the cache. As long as I searched on that particular column, everything would work as expected. The moment I switched columns, however, performance plummeted. Why did this happen?

The first search I performed created a query plan and stored it in the cache. For instance, say I was searching on the column OrderDate. If I switched the search to the CompanyName column, SQL would blindly use the cached query plan, searching for the target company name using the OrderDate index. No wonder performance would plummet so dramatically.

The fix is quite simple. I executed the sproc supplying the WITH RECOMPILE option:

This tells SQL Server to throw away the existing query plan and build another one--but

only this once.

You can also add the WITH RECOMPILE directly to the stored procedure right before the AS keyword. This tells SQL Server to throw out the query plan on every execution of the sproc.

There is also a third option. I could have created a separate sproc for each search method, and then decide which one to execute within the CASE block. That way, the query plan associated with the sub-sprocs remains in the cache, where SQL can take advantage of them. Since each of the sprocs searched exactly one column, there is no need to recompile.

SQL Server's ability to optimize and compile a stored procedure is great but, if you aren't careful, it can bite you when you least expect it. Now that you know how to deal with the problem, perhaps there are a few situations in your own database that you might want to revisit.

Arthur Fuller has been developing database applications for more than 20 years. He frequently works with Access ADPs, Microsoft SQL 2000, MySQL, and .NET.