**SQL SERVER – Stored Procedure Optimization Tips – Best Practices**

We will go over how to optimize Stored Procedure with making simple changes in the code. Please note there are many more other tips, which we will cover in future articles.

* **Include SET NOCOUNT ON statement:** With every SELECT and DML statement, the SQL server returns a message that indicates the number of affected rows by that statement. This information is mostly helpful in debugging the code, but it is useless after that. By setting SET NOCOUNT ON, we can disable the feature of returning this extra information. For stored procedures that contain several statements or contain Transact-SQL loops, setting SET NOCOUNT to ON can provide a significant performance boost because network traffic is greatly reduced.

**CREATE PROC dbo.ProcName  
AS  
SET NOCOUNT ON;  
--Procedure code here  
SELECT column1 FROM dbo.TblTable1  
-- Reset SET NOCOUNT to OFF  
SET NOCOUNT OFF;  
GO**

* **Use schema name with object name:** The object name is qualified if used with schema name. Schema name should be used with the stored procedure name and with all objects referenced inside the stored procedure. This help in directly finding the complied plan instead of searching the objects in other possible schema before finally deciding to use a cached plan, if available. This process of searching and deciding a schema for an object leads to COMPILE lock on stored procedure and decreases the stored procedure’s performance. Therefore, always refer the objects with qualified name in the stored procedure like

**SELECT \* FROM dbo.MyTable -- Preferred method  
-- Instead of  
SELECT \* FROM MyTable -- Avoid this method  
--And finally call the stored procedure with qualified name like:  
EXEC dbo.MyProc -- Preferred method  
--Instead of  
EXEC MyProc -- Avoid this method**

* **Do not use the prefix “sp\_” in the stored procedure name**: If a stored procedure name begins with “SP\_,” then SQL server first searches in the master database and then in the current session database. Searching in the master database causes extra overhead and even a wrong result if another stored procedure with the same name is found in master database.
* **Use IF EXISTS (SELECT 1) instead of (SELECT \*):** To check the existence of a record in another table, we uses the IF EXISTS clause. The IF EXISTS clause returns True if any value is returned from an internal statement, either a single value “1” or all columns of a record or complete recordset. The output of the internal statement is not used. Hence, to minimize the data for processing and network transferring, we should use “1” in the SELECT clause of an internal statement, as shown below:

**IF EXISTS (SELECT 1 FROM sysobjects  
WHERE name = 'MyTable' AND type = 'U')**

* **Use the sp\_executesql stored procedure instead of the EXECUTE statement.**  
  The sp\_executesql stored procedure supports parameters. So, using the sp\_executesql stored procedure instead of the EXECUTE statement improve the re-usability of your code. The execution plan of a dynamic statement can be reused only if each and every character, including case, space, comments and parameter, is same for two statements. For example, if we execute the below batch:

**DECLARE @Query VARCHAR(100)  
DECLARE @Age INT  
SET @Age = 25  
SET @Query = 'SELECT \* FROM dbo.tblPerson WHERE Age = ' + CONVERT(VARCHAR(3),@Age)  
EXEC (@Query)**

If we again execute the above batch using different @Age value, then the execution plan for SELECT statement created for @Age =25 would not be reused. However, if we write the above batch as given below,

**DECLARE @Query NVARCHAR(100)  
SET @Query = N'SELECT \* FROM dbo.tblPerson WHERE Age = @Age'  
EXECUTE sp\_executesql @Query, N'@Age int', @Age = 25**

the compiled plan of this SELECT statement will be reused for different value of @Age parameter. The reuse of the existing complied plan will result in improved performance.

* **Try to avoid using SQL Server cursors whenever possible:** Cursor uses a lot of resources for overhead processing to maintain current record position in a recordset and this decreases the performance. If we need to process records one-by-one in a loop, then we should use the WHILE clause. Wherever possible, we should replace the cursor-based approach with SET-based approach. Because the SQL Server engine is designed and optimized to perform SET-based operation very fast. Again, please note cursor is also a kind of WHILE Loop.
* **Keep the Transaction as short as possible:** The length of transaction affects blocking and deadlocking. Exclusive lock is not released until the end of transaction. In higher isolation level, the shared locks are also aged with transaction. Therefore, lengthy transaction means locks for longer time and locks for longer time turns into blocking. In some cases, blocking also converts into deadlocks. So, for faster execution and less blocking, the transaction should be kept as short as possible.
* **Use TRY-Catch for error handling**: Prior to SQL server 2005 version code for error handling, there was a big portion of actual code because an error check statement was written after every t-sql statement. More code always consumes more resources and time. In SQL Server 2005, a new simple way is introduced for the same purpose. The syntax is as follows:

**BEGIN TRY  
--Your t-sql code goes here  
END TRY  
BEGIN CATCH  
--Your error handling code goes here  
END CATCH**

Reference: **Pinal Dave (**[**http://blog.SQLAuthority.com**](http://blog.sqlauthority.com/)**)**