

SQLskills Immersion Event

SQL Server Programming for Developers

Module 6: Optimizing Procedural Code

Part 1: Stored Procedures

Bob Beauchemin
BobB@SQLskills.com



Overview

- Stored procedure overview
- Temporary Structures
- When to optimize T-SQL code
- Why queries run slowly
 - Query plans
 - Plan caching and reuse
- How to find query information
- How to fix it
 - Parameterization
 - Recompile
 - Statistics
 - Hints and Plan Guides



2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Stored Procedures

- **Modules that are cataloged to database**
 - Written in T-SQL
 - Can also be written in SQLCLR
 - Data access usually slower this way
- **SQL statements embedded in procedural code**

Stored Procedures

- **Full power of SQL language**
 - Procedural code can be included
- **Static SQL is syntax checked**
 - At procedure creation time
- **Database object names referred to in SQL**
 - Not checked at creation time
 - Column names are checked

Stored Procedure Logistics

- **All database code is in one place**
 - Easier for database upgrade
 - SQL Upgrade Advisor checks modules
 - System views for dependencies
 - VSTS does static SQL code analysis
- **Facilitates performance troubleshooting**
 - Push all cataloged T-SQL code into database tuning advisor

Static SQL vs. Dynamic SQL

- **Procedures can contain**
 - Static SQL
 - select col1, col2 from table
 - Dynamic SQL
 - Execute('select col1, col2 from table')
 - sp_executesql – parameterized dynamic SQL
- **Security repercussions for composed SQL**
 - Any SQL created with string concatenation subject to SQL injection

Stored Procedure Performance

- **Stored Procedures have**
 - One query plan that contains
 - One or more statement query plans
- **Separate plan cache for sproc plans**
- **Sprocs execute through RPC**
 - Not slower SQL:Statement calls

Stored Procedures and Metadata

- **SQL Server procedure metadata contains**
 - Data type of parameters
 - Direction of parameters
 - Defaults
- **This specifies a contract for parameters**
- **There is no contract for resultsets**
 - Number of resultsets
 - Data types returned in resultset columns

EXECUTE WITH RESULT SETS

- **Resultsets can be specified on EXECUTE statements in SQL Server 2012**
 - For dynamic SQL, OPENROWSET
 - Stored Procedures
- **Specify shape of resultset**
 - If data type or collation does not match, conversion will be done on return
 - If conversion not possible, returns error
 - Multiple results specifications possible

Resultset Metadata

- **Resultset Metadata Before SQL Server 2012**
 - SELECT... FOR BROWSE
 - EXECUTE... FOR BROWSE
- **Resultset Metadata in SQL Server 2012**
 - sys.dm_exec_describe_first_result_set
 - sp_describe_first_result_set
 - With query
 - sys.dm_exec_describe_first_result_set_for_object
 - With OBJECT_ID (e.g. procedure ID)
 - sp_describe_first_undeclared_parameters
 - For query

Scalar Parameters

- **Stored Procedures allow**
 - Input parameters – default
 - Output parameters – OUTPUT keyword
 - Return code – integer (defaults to zero)
- **Input parameters passed by value**
- **Output parameters passed by reference**
- **Parameter defaults can be specified**
- **Passed in order or as name/value pairs**
 - Parameters with defaults can be omitted

Table-Valued Parameters

- **Inserts into structures with 1-n cardinality problematic**
 - One order -> N order line items
 - "N" is variable and can be large
 - Don't want to force a new order for every 20 line items
- **One database round-trip / line item slows things down**
 - No ARRAY data type in SQL Server
 - XML composition/decomposition used as an alternative
- **Table-valued parameters solve this problem**

Table Types

- SQL Server has table variables
`DECLARE @t TABLE (id int);`
- SQL Server 2008 adds strongly typed table variables

```
CREATE TYPE mytab AS TABLE (id int);  
DECLARE @t mytab;
```

- Parameters must use strongly typed table variables

Table Variables are Input Only

- Declare and initialize TABLE variable

```
DECLARE @t mytab;  
INSERT @t VALUES (1), (2), (3);  
EXEC myproc @t;
```

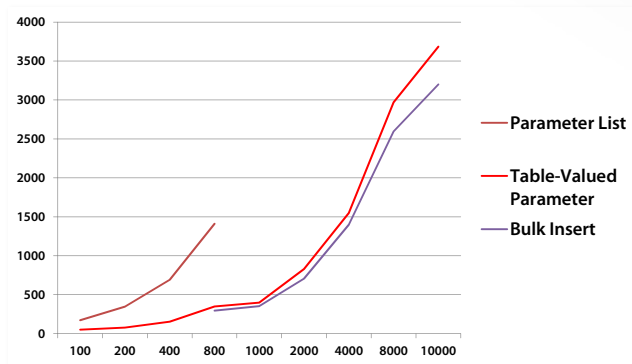
- Procedure must declare variable READONLY

```
CREATE PROCEDURE usetable (  
    @t mytab READONLY)  
AS  
BEGIN  
    INSERT INTO lineitems SELECT * FROM @t;  
    -- UPDATE @t SET... no!  
END
```

TVP Implementation and Performance

- **Table Variables materialized in TEMPDB**

- Faster than parameter arrays
- BCP APIs still fastest



Temporary Structures

- **Temp Tables**

- Allow statistics and DDL

- **Table Variables**

- Limitations might not affect you
- Might be the most optimal

- **Derived Tables**

- Nested Subquery in FROM clause
- May optimize better than temp tables/variables

- **Views**

- Another option – rewrite existing temp table code to use views instead (simple rewrite)
- May optimize better than temp tables

- **Common Table Expressions – similar to views**

- **Table-valued Functions**

- **Table-valued Parameters**

Temporary Tables

- **Scope depends on where its defined**
 - In scope for current and lower-level scopes
- **Temp Table can create excessive recompilations for procedures.**
Consider creating permanent tables (with indexes) and manipulating data there.
- **Consider dropping and re-creating or rebuilding indexes as part of the procedure instead!**
- **Use Profiler to see if there are significant recompiles (watch EventSubClass)**
- **Use KEEP PLAN on SELECT statements if data changes more than 6 times but the plan should not change.**

Table Variables (1)

- **Scope is limited to the local batch/procedure**
- **Does not cause excessive recompiles (local only)**
 - No re-resolution on CREATE/ALTER
 - Temp Tables need re-resolution for nested procedures
- **Less logging as it's not a permanent object**

Table Variables (2)

- **Indexes must be declared in table definition**
 - Does not support CREATE INDEX
- **No statistics**
 - Use TEMP TABLES if large volumes of data will be manipulated – create the right indexes for access
- **Population**
 - Does not support SELECT INTO
- **Rolling back transaction does not affect table variables**

Temp Table vs. Table Variables

- **Temp Table**
 - PROs
 - Supports CREATE INDEX and statistics
 - Can access from other nested procedures
 - Can populate with INSERT EXEC or SELECT INTO
 - CON – Potential for excessive recompiles due to resolution
- **Table Variable Table**
 - PRO – Local only – no excessive recompiles
 - CONs
 - No CREATE INDEX or statistics
 - Not as flexible on population

When to Optimize T-SQL Code

- The users will point out when to optimize
- "SQL Server is running slowly"
 - Is it all queries?
 - It is a specific set of queries?
 - It is specific timeframes?
- Or proactive optimization

What's Running Slowly

- If the whole instance is running slowly
 - Data Collection
 - Bottleneck Analysis
 - sys.dm_os_wait_stats
 - DBCC SQLPERF(WAITSTATS)
 - sys.dm_os_waiting_tasks

Which Queries are Running Slowly

- **Types of queries to tune**
 - Frequently executed queries
 - Small improvement is worthwhile if query runs 10000 times/hour
 - Parameterization helps detecting similar queries
 - May need to combine similar queries
 - sys.dm_exec_query_stats (aggregates)
 - Built-in SSMS reports (by CPU, by IO)
 - Heavy resource consumption queries
 - Month-end reports
 - Analysis queries
 - Watch for high response time in profiler
 - Queries that are slow because in lock waits

Why Code Runs Slowly

- **Queries run slowly because of**
 - Suboptimal physical structures
 - Missing indexes
 - Lack of covering indexes
 - Lack of index on computed columns
 - Out of date statistics
 - Non-parameterized queries
 - Excessive compilation time
 - Parameter sniffing - too little compilation
 - Overly complex queries
 - cardinality estimates incorrect

Troubleshooting Queries - Tools

- Dynamic Management Views
- SQL Profiler
- Showplan
- Performance Monitor
- Database Tuning Advisor
- SQL Server 2008 Adds
 - Data Collection
 - Extended Events

Missing Indexes

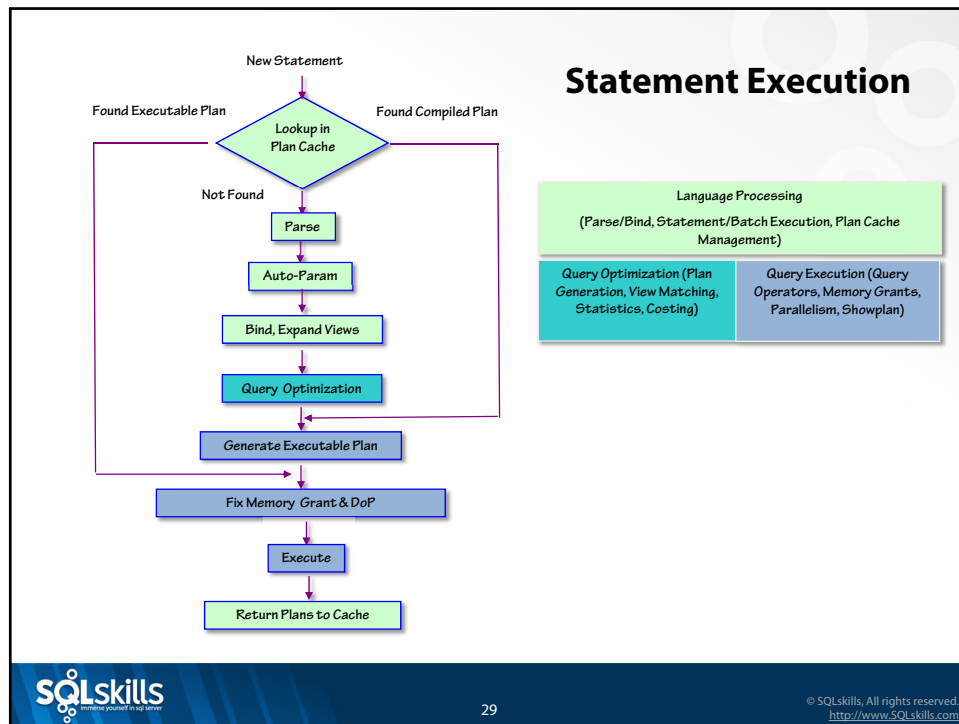
- Missing indexes appear in
 - Query Plans
 - sys.dm_db_missing_index_*
- Covering indexes may not appear as missing
 - Must cover all columns to be useful
 - Index included columns in SQL Server 2005 expand possibilities of covering indexes

Computed Columns

- **Query processor can use index**
 - when search argument on one side of query
 - when formula uses one column
- **Query processor cannot use index**
 - non-SARGable queries
 - formulas that use multiple columns
- **Solution: persist computed columns**
 - can add index if functions are deterministic
 - SQL Server 2005/2008 will use index even if same formula is embedded in queries

Query Processing

- **Statement processing**
- **The plan caches**
- **Statistics**
- **Parameterization**



Compilation and Plan Selection

- **Parsing and object resolution phase**
 - View substitution
- **Normalization phase**
 - Logical simplification - uses constraints
 - normalize predicates
 - detect contradictions
 - remove redundant operations
 - evaluate constant expressions
 - Push down filters to evaluate as early as possible
 - Convert subqueries to joins (add outer join or group by if needed)
 - Remove unneeded joins and outer joins
 - Turn outer joins into inner joins, if later predicates reject nulls
 - Predicates in the query as well as check constraints
- **Cost-based optimization phase is based on**
 - Data access paths
 - Data volume
 - Distribution

SQLskills
Master your SQL skills

30

© SQLskills. All rights reserved.
<http://www.SQLskills.com>

Automatic Statistics

- **Out of date statistics can yield bad plans**
 - Compare estimates and actual in plans
- **Statistics configurable on database**
 - AUTO_CREATE_STATISTICS
 - AUTO_UPDATE_STATISTICS
 - AUTO_UPDATE_STATISTICS_ASYNC
- **AUTO_CREATE and AUTO_UPDATE defaults**
 - View current setting in sys.databases
- **Statistics auto-created on index columns**
 - Index (A,B,C) -> stats on (A), (A,B), (A,B,C)

Manual Statistics

- **You can create statistics manually**
 - CREATE STATISTICS
 - sp_createstats
 - UPDATE STATISTICS ON
table/view/index/statistics
sample {percent/number} or fullscan
norecompute
 - sp_updatestats
- **You must create multicolumn manually**

Plan Reuse

- **Plan reuse is usually is good thing**
 - Compilation takes CPU
 - Less total plans -> more plans in plan cache
- **Plan reuse can be achieved by**
 - Using stored procedures
 - Using parameterized queries
 - Autoparameterization
- **Plans can be aged out of cache**
 - Under memory pressure
 - Same algorithm for all caches

Plan Caches

- **Three major query plan caches**
 - Procedure cache
 - Adhoc query cache
 - XP cache
- **Query plans are reentrant**
- **Execution plans are also cached**
 - Execution plans not reentrant
 - Execution plans are reuseable

Where to Find Query Plan Info

- **SQL Server 2000/2005**
 - Access syscacheobjects
- **SQL Server 2005/2008 – DMVs**
 - sys.dm_exec_cached_plans
 - sys.dm_exec_cached_plan_dependent_objects
 - For list of query plans, execution plans
 - sys.dm_exec_sql_text(sql_handle)
 - For the text of the sql statement executed
 - sys.dm_exec_query_plan(plan_handle)
 - sys.dm_exec_text_query_plan(plan_handle) SP2
 - For the execution plan of the sql statement executed
 - sys.dm_exec_query_stats
 - For a variety of query statistics – like number of executions, plan creation time (first execution into cache), last execution time, etc.
 - sys.dm_exec_plan_attributes(plan_handle)
 - Determine why plan is/is not being reused

Plans and Parameterization

- **Parameterization helps plan reuse**
 - Stored Procedures
 - Parameterized Queries
 - sp_executesql
 - Autoparameterization
- **Autoparameterization**
 - can be observed in plan cache
 - exact data type based on parameter value
 - not as useful as explicit parameterization
 - SIMPLE vs FORCED
 - default set per database
 - can be changed with plan guide

Naming to Encourage Plan Reuse

- **Owner Qualify to Eliminate Ambiguity**
 - On execution
`EXEC dbo.procname`
 - On creation
`CREATE PROC dbo.procname`
`AS`
`SELECT alias.columnlist`
`FROM dbo.tablename AS alias`
`EXEC dbo.subprocname`
- **Minimize Blocking**
 - Excessive lookups can cause significant blocking and cache misses
- **Do not use sp_ in stored procedure names**
 - See KB Article Q263889

Modifying Procedures

- **DROP and RECREATE**
 - Loses the permissions already granted
 - Invalidates all plans
- **ALTER PROC**
 - Retains the permissions
 - Invalidates all plans
- **To retain the dependency chain you must also ALTER all procedures that depend on the procedure being altered**
 - Until SQL Server 2008...

Keeping Track of Dependencies

- **New dependency views replace sp_depends**
 - Views are kept in sync as changes occur
- **sys.sql_object_dependencies**
 - Object dependencies by name
 - Replacement for sys.sql_dependencies
- **sys.dm_sql_referenced_entities**
 - All named entities that an object references
 - Example: which objects does this stored procedure use?
- **sys.dm_sql_referencing_entities**
 - All named entities that use an object
 - Example: which objects use this table?
- **Can see references at OBJECT, DATABASE DDL TRIGGER, SERVER DDL TRIGGER level**

Changing SESSION Settings

- **Certain Session Settings can be set within a stored procedure – some can be desired:**
 - SET NOCOUNT ON
 - SET QUOTED_IDENTIFIER OFF
(not recommended except for backward compatibility and upgrades)
- **Some Session Settings will cause EVERY execution to force a recompile:**
 - ANSI_DEFAULTS
 - ANSI_NULLS
(tip: do not use WHERE col = null, use col IS NULL)
 - ANSI_PADDING
 - ANSI_WARNINGS
 - CONCAT_NULL_YIELDS_NULL
(tip: use the ISNULL function to concatenate strings)
- **Recommendation: DO NOT Change these session settings in the client or the server!**
- **See “SET Options that Affect Results” in the BOL**

Parameter Sniffing

- **Queries in procedure optimized on compile**
 - First set of parameter values used
 - Never reoptimized for different values
- **Parameters not sniffed (or used)**
 - If changed in procedure code
 - If passed to queries from variables
- **If parameter sniffing is a problem,**
 - Use explicit recompiles
 - Modify code to be modular
 - multiple procs not CASE
 - OPTION (OPTIMIZE FOR VALUES/UNKNOWN)

Statement Recompilation

- **Plans are recompiled when**
 - Metadata changes
 - Statistics change
 - All plan options do not match (cache_keys)
 - Other reasons
 - SELECT v.subclass_name, v.subclass_value
 - from sys.trace_events e inner join sys.trace_subclass_values v
 - on e.trace_event_id = v.trace_event_id
 - where e.name = 'SP:Recompile'
- **You can force a recompile**
 - OPTION (RECOMPILE)
 - CREATE PROCEDURE ...WITH RECOMPILE
 - EXECUTE ...WITH RECOMPILE
 - sp_recompile

sp_recompile

- Can be used to periodically and directly force recompilation of a procedure (or trigger)
- Can be used on tables and views to indirectly force the recompilation of all procedures and triggers that reference the specified table or view
- Does not actually recompile the procedures ⇒ Instead it invalidates plans for next execution
- SQL Server invalidates plans as data changes
- Never really negative – especially if you run it at night as part of batch processing after index rebuilds or statistics updates with FULLSCAN

Conditional SQL

- **Avoid conditional SQL to modularize sprocs**
IF (expression operator expression)
SQL Statement Block1
ELSE
SQL Statement Block2
- **If same parameters are used, all SQL optimized based on first set of parameter values**
- **Use separate stored procedures instead**
IF (expression operator expression)
EXECUTE Procedure1
ELSE
EXECUTE Procedure2

How Do You Know?

- **Test optimization plans consistency**
 - EXECUTE WITH RECOMPILE
- **Choose what needs to be recompiled**
 - Whole Procedure
 - Portions of the procedure
- **Test final performance using strateg**
 - Procedure Recompilation
 - CREATE with RECOMPILE
 - Statement Recompilation
 - RECOMPILE hint or string execution
 - Modularized Code
 - Sub procedures with or without WITH RECOMPILE

Profiling SP Performance

- **Create New Trace (SQLProfilerTSQL_sps)**
- **Replace SP:StmtStarting w/SP:StmtCompletion**
 - Better if you want to see a duration
 - Add Duration as a Column Value
- **Look at Recompiles, Statistics Events**
 - Performance:Performance Statistics
 - SQL:StmtRecompile, SP:Cache...
- **If short term profiling for performance:**
 - Add columns: Reads, Writes, Execution Plan
- **Always use Filters**
 - Database Name (only the db you want)
 - Exclude system IDs (checkbox on filter dialog)

Hints

- **SQL Server has two types of hints**
 - **QUERY hints**
 - Many of these
 - **TABLE hints**
 - Use table scan/seek
 - Use one or more indexes
 - Locking hints
 - NOEXPAND
 - Spatial_Window_Max_Cells
 - Some can be used as query hints

Query Hints

- { LOOP | MERGE | HASH } JOIN
- { HASH | ORDER } GROUP
- { CONCAT | HASH | MERGE } UNION
- FAST number_rows
- FORCE ORDER
- IGNORE_NONCLUSTERED_COLUMNSTORE_INDEX
- MAXDOP number_of_processors
- OPTIMIZE FOR (@variable_name = literal_constant [,...n])
- OPTIMIZE FOR UNKNOWN
- PARAMETERIZATION { SIMPLE | FORCED }
- RECOMPILE
- ROBUST PLAN
- KEEP PLAN
- KEEPFIXED PLAN
- EXPAND VIEWS
- MAXRECURSION number
- USE PLAN N'xml_plan'

Beyond Query Hints

- **Query hints cannot be applied if you don't have access to the source code**
 - Dynamically generated
 - Purchased package
- **Query hints tie the hint to the source code**
 - No straightforward way to disable
 - Always using procedures helps this
- **Enter plan guides...**

Plan Guide

- **Plan guide is a named database object**
 - Only available in Standard and Enterprise
 - Associates a hint with a query
 - Lives at database scope
 - Defined and maintained through system procs
 - `sp_create_plan_guide`
 - Can be enabled or disabled
 - `sp_control_plan_guide`
 - `sp_create` and `sp_control` flush appropriate query cache entries
 - `sys.plan_guides` holds plan guide metadata

SQL Server 2008 Plan Guides

- **More table hints as query hints**
 - Allows use in plan guides
- **Easier to create and check**
 - `sp_create_plan_guide_from_cache`
 - `sys.fn_validate_plan_guides`
- **Detectable**
 - Profiler Events
 - Performance Monitor Counters

Summary

- **Stored procedures vs. static SQL**
 - Use appropriate parameters, table structures
- **Determine which queries to tune**
 - Frequently used
 - Big resource consumers
- **Look at query plans for**
 - Missing indexes
 - Out-of-date statistics
- **Check plan cache for query reuse**
 - Use stored procs or parameterize
 - Beware of parameter sniffing
 - Recompile explicitly when needed
- **Query hints and plan guides as last resort**

Resources

- **Statistics Used by the Query Optimizer in Microsoft SQL Server 2005, Eric Hanson and Lubor Kollar,**
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/qrystats.mspx>
- **Batch Compilation, Recompilation, and Plan Caching Issues in SQL Server 2005, Arun Marathe,**
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.mspx>
- **Troubleshooting stored procedure recompilation**
 - <http://support.microsoft.com/default.aspx?scid=kb;en-us;Q243586>

Resources (Blogs)

- **Tips, Tricks, and Advice from the SQL Server Query Optimizer Team**
 - <http://blogs.msdn.com/programmability>
- **SQL Server Engine Tips**
 - <http://blogs.msdn.com/sqltips>
- **SQL Programmability Team Blog**
 - <http://blogs.msdn.com/sqlqueryprocessing>
- **Craig Freedman's Web Log**
 - <http://blogs.msdn.com/craigfr>

Questions?

