# Understanding, Improving, and Resolving Issues with the SQL Server Procedure Cache

Becoming a Query Plan Caching Superhero!

Charlie Arehart, Independent Consultant charlie@carehart.org

# Topics

- Thinking through DB request processing
- Observing the plan cache
  - Code for evaluating query plan usage
- Cache Plan Management
  - How cache is managed, may be cleared
- Understanding query parameterization
  - Parameterized queries
  - Forced
  - Simple/Auto
- Monitoring & performance implications
- Changes in 2005, especially SP2

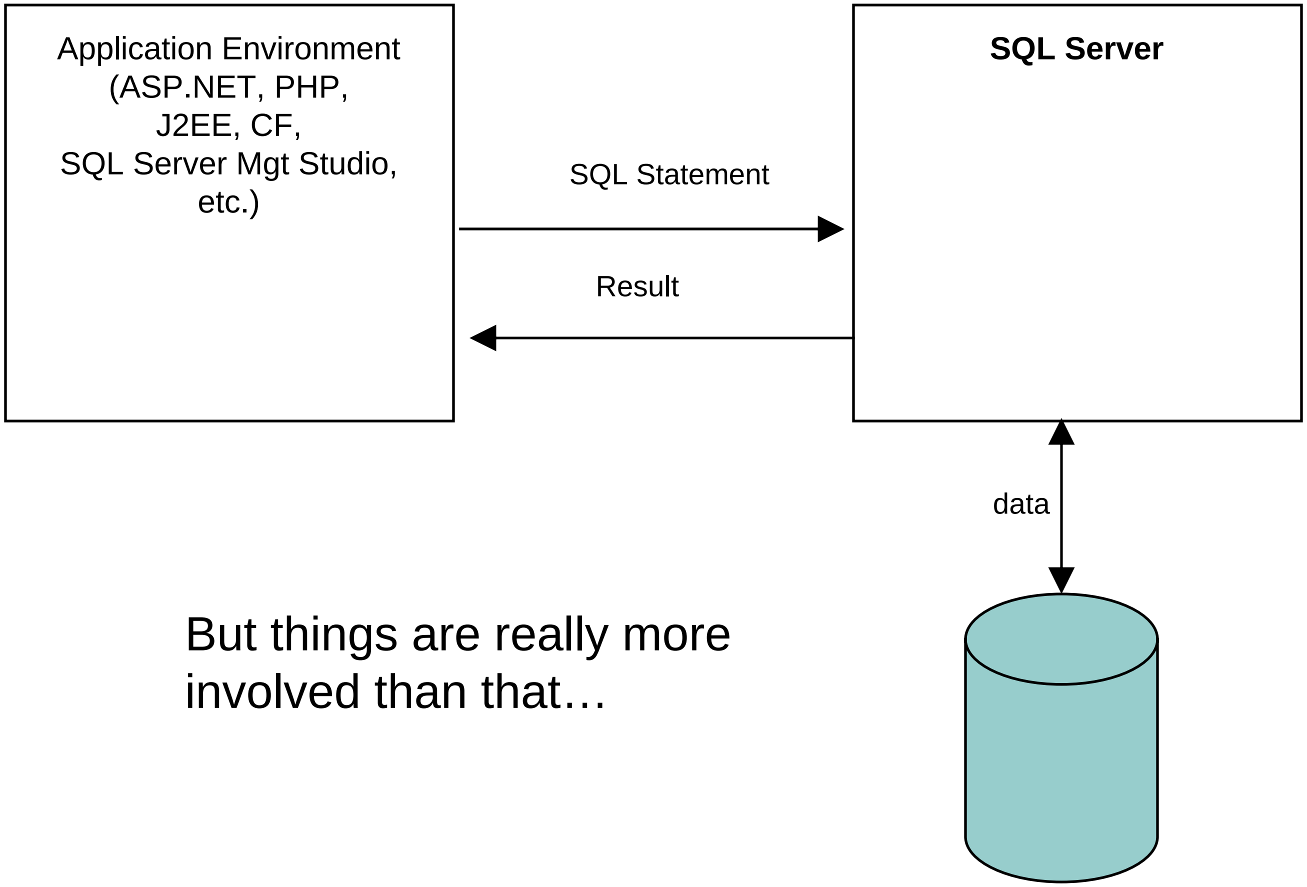
# About Charlie Arehart

- Member of Atlanta MDF and .NET UGs about 2 years
  - Just sharing, returning favor to community that's helped me
- 25 Yrs IT Experience:
  - Databases (25), Web Apps (10), SQL Server (7), .NET (3)
- Frequent speaker to user groups, conferences worldwide, including recent MS events
  - Atlanta MDF August 2006
  - 2006 Atlanta and Greenville
  - PASS 2006
- Frequent writer, speaker on other web app dev topics
- Not claiming to be a SQL Server expert
  - Nor even expert on this topic
  - Just sharing things I've learned
    - I welcome observations, questions tonight
- Let's also realize people are at very different experience levels

# Sources for this talk

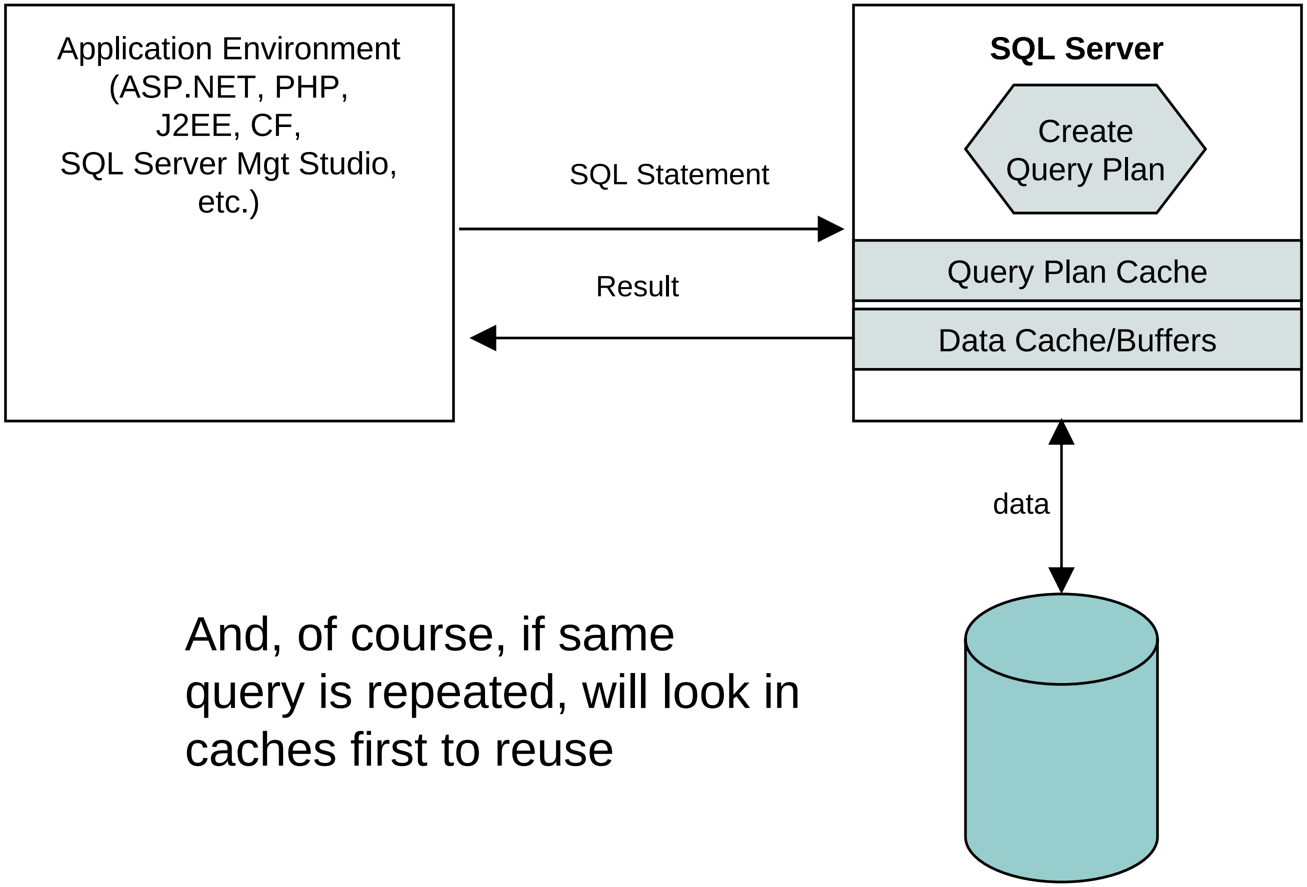
- Where to find much more...
  - SQL Pass talk by Bart Duncan
  - His chapter in the book, “SQL Server 2005 Practical Troubleshooting, The Database Engine”
    - Safari chapter online http://safari.oreilly.com/0321447743/ch04
    - And I have a copy to give away tonight! ☺
  - Of course, BOL
  - Perhaps best of all, 16-part plan cache article series
    - Bart and other MS reps put together incredibly resourceful series of blog entries on this topic
      - http://blogs.msdn.com/sqlprogrammability/archive/2007/01/08/plan-cache-concepts-explained.aspx
    - Find links to each at:
      - http://carehart.org/blog/client/index.cfm/2007/8/13/resources_on_sql_server_query_plan_cache

# Thinking through DB request processing



But things are really more involved than that…

# Thinking through DB request processing



Application Environment
(ASP.NET, PHP,
J2EE, CF,
SQL Server Mgt Studio,
etc.)

SQL Statement

Result

**SQL Server**

Create
Query Plan

Query Plan Cache

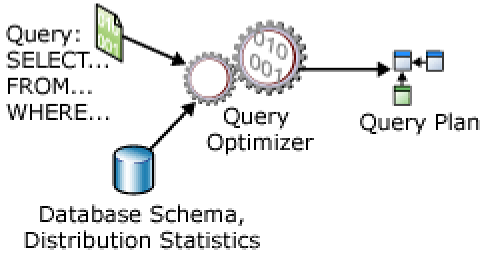
Data Cache/Buffers

data

And, of course, if same query is repeated, will look in caches first to reuse

# Understanding query plan creation

- Query plan creation analyzes many datapoints
  - Influenced by indexing, row count, value distribution within columns, and more



- This takes time, that's separate from getting data
  - It **is** good that DBMS does this for us
    - But it's also relatively expensive
  - Fortunately, SQL Server caches this
    - We can influence *how often* and *how well* it does plan caching

# Some terminology, variations

- **Query plan** also called “execution plan”, “compiled plan”
- Technically, “Execution Plan” comprised of
  - Query Plan – reusable by many users
  - Execution Contexts – user-specific details, parm vals
  - (http://msdn2.microsoft.com/en-us/library/ms181055.aspx)
- **Query plan cache** also referred to as “procedure cache”, “statement cache”

# Observing the cache

- Simple count of query plan (proc) cache size
  - DBCC PROCCACHE
    - And its Proc Cache Size value
- SQL Server 2000 and 7
  - syscacheobjects system table
    - Cacheobjtype=Compiled Plan, Executable Plan, etc.
    - Objtype=Ad hoc query, Prepared statement, Stored Procedure, etc
    - Usecounts – number of times used since cached
    - Refcounts – count of other objects using it
    - And more
- SQL Server 2005 DMVs have this and much more

# SQL 2005 Cache Plan DMVs

- sys.dm_exec_cached_plans
  - Offers similar info, and more
  - Usecounts - Number of times the query plan has been used
  - Size_in_bytes - Number of bytes consumed by this cache object
  - Cacheobjtype - Type of the cache object i.e. if it’s a compiled plan, or a parse tree or an xproc
  - Memory_object_address - Memory address of the cache object
  - And much more

## Some simple code to observe query plans

```
-- simple queries to observe cached plans
select count(*) from
  sys.dm_exec_cached_plans cp
select count(*) from
  sys.dm_Exec_Cached_plans where
  cacheobjtype = 'Compiled Plan'
select count(*) from
  sys.dm_Exec_Cached_plans where
  cacheobjtype = 'Compiled Plan' and
  objtype = 'Adhoc'
select count(*) from
  sys.dm_Exec_Cached_plans where
  cacheobjtype = 'Compiled Plan' and
  objtype = 'Prepared'
```

# More SQL 2005 DMVs

- sys.dm_exec_sql_text
  - Text – actual SQL statement of the query

- Can combine with previous to get more info about each query plan

# More code to observe query plans

```
-- list query plans
select st.text, cp.plan_handle,
  cp.usecounts, cp.size_in_bytes,
  cp.cacheobjtype, cp.objtype,
  cp.memory_object_address
from sys.dm_exec_cached_plans cp
cross apply
  sys.dm_exec_sql_text(cp.plan_handle) st
order by cp.usecounts desc
go
```

# Some More SQL 2005 DMVs

- sys.dm_exec_requests
  - returns execution and resource consumption information for currently executing requests in the server
  - can be used to identify the long running queries among the requests currently executing
- sys.dm_exec_query_stats
  - returns aggregate performance statistics for cached query plans
  - contains a row per query/statement within the cached plan (plan could be for a batch of SQL statements)
  - If cache or cache entry is flushed, will no longer be present
- Can combine these in a variety of ways

# Long running current queries

```
-- long running queries among the requests currently
   executing
select top 10 substring(st.text,
   (er.statement_start_offset/2) + 1,
((case statement_end_offset
when -1
      then datalength(st.text)
else
      er.statement_end_offset
end
- er.statement_start_offset)/2) + 1) as statement_text
, *
from sys.dm_exec_requests er
cross apply sys.dm_exec_sql_text(er.sql_handle) st
order by total_elapsed_time desc
go
```

# Top queries in proc cache by CPU

```
-- top 10 queries in proc cache, with highest CPU time
select top 10 substring(st.text,
  (qs.statement_start_offset/2) + 1,
((case statement_end_offset
when -1
      then datalength(st.text)
else
      qs.statement_end_offset
end
- qs.statement_start_offset)/2) + 1) as statement_text,
*
from sys.dm_exec_query_stats qs
cross apply sys.dm_exec_sql_text(qs.sql_handle) st
order by total_elapsed_time/execution_count desc
go
```

# Top 10 compiled plans by usecount

```
-- top 10 used compiled plans
SELECT TOP 100 objtype, usecounts,
  size_in_bytes, cacheobjtype,
REPLACE (REPLACE ([text], CHAR(13), ' '),
  CHAR(10), ' ') AS sql_text
FROM sys.dm_exec_cached_plans AS p
CROSS APPLY sys.dm_exec_sql_text
  (p.plan_handle)
WHERE p.objtype in
  ('Proc','Prepared','Adhoc') AND
  cacheobjtype = 'Compiled Plan'
ORDER BY objtype, usecounts DESC
```

# More details about cached plans

```
-- show more details, worker_time, etc.
select substring(st.text,
      (qs.statement_start_offset/2) + 1,
      ((case qs.statement_end_offset when -1
            then datalength(st.text)
            else qs.statement_end_offset
            end
      - qs.statement_start_offset)/2) + 1) as
  query,
      qs.execution_count, qs.last_worker_time,
      qs.max_worker_time, qs.last_execution_time
  from sys.dm_exec_cached_plans cp
      inner join sys.dm_exec_query_stats qs
      on cp.plan_handle = qs.plan_handle
  cross apply sys.dm_exec_sql_text(qs.sql_handle)
  st
  order by execution_count desc
```

# Other queries

- To get a count of the number of compiled plans use:

  -
    ```
    select count(*) from sys.dm_Exec_Cached_plans
    where cacheobjtype = 'Compiled Plan'
    ```
- To get a count of the number of adhoc query plans use:

  -
    ```
    select count(*) from sys.dm_Exec_Cached_plans
    where cacheobjtype = 'Compiled Plan'
    and objtype = 'Adhoc'
    ```
- To get a count of the number of prepared query plans use:

  -
    ```
    select count(*) from sys.dm_Exec_Cached_plans
     where cacheobjtype = 'Compiled Plan‘
    and objtype = 'Prepared'
    ```
- For the number of prepared query plans with a given usecount use:

  -
    ```
    select usecounts, count(*) as no_of_plans
    from sys.dm_Exec_Cached_plans
    where cacheobjtype = 'Compiled Plan'
    and objtype = 'Prepared'
    group by usecounts
    ```

# Other queries

- For the number of adhoc query plans with a given usecount use:

  -
    ```
    select usecounts, count(*) as no_of_plans
    from sys.dm_Exec_Cached_plans
    where cacheobjtype = 'Compiled Plan'
    and objtype = 'Adhoc'
    group by usecounts
    ```
- For the top 1000 adhoc compiled plans with usecount of 1 use:

  -
    ```
    select top(1000) * from sys.dm_Exec_cached_plans
    cross apply sys.dm_exec_sql_text(plan_handle)
    where cacheobjtype = 'Compiled Plan'
    and objtype = 'Adhoc' and usecounts = 1
    ```

# Internal Cache Management

- SQL Server manages cache objects
  - **Data cache / buffers** cleared based on use counts, underlying data being changed
  - **Query plan cache** cleared by aging execution plans
- Can clear query plan cache manually
  - **DBCC FreeProcCache**
    - Useful for testing, but beware in production
    - We’ll come to appreciate value of cache, so don’t cavalierly clear it in production!

# Plan cache aging

- Cache plan saved with an “age” and “cost factor”
  - Latter determined by cost of creating the plan
- Each time a plan is referenced, age field is incremented by the compilation cost factor
- Cache swept periodically
  - age field of each object decremented by 1
- Execution plan removed if all are true:
  - memory manager requires memory
  - all available memory is currently in use
  - age field for object is 0
  - not currently referenced by a connection
- Source: http://msdn2.microsoft.com/en-us/library/ms181055.aspx

# Observing Cached Plan Management

- Our goal in processing is generally to have high reuse of cache entries
  - Avoid thrashing by removing/inserting same entries into cache
  - Have a couple of hi-level ways to monitor this
- SQL Profiler
  - Stored Procedures event (even for dynamic SQL)
    - SP:CacheHit/CacheInsert/CacheMiss/CacheRemove
- PerfMon
  - SQLServer:PlanCache event
    - “Cache Hit Ratio” for “SQL Plans” instance
    - “Cache Object Counts”/”Cache Objects in use”

# Non-Cached Plans

- Certain query plans are never inserted into the procedure cache
  - plans used to create or update statistics
  - those used to execute DBCC commands,
  - those used to create or rebuild indexes
  - dynamic SQL executed via EXEC()
  - any stored procedure or query executed with RECOMPILE
  - query that contains string/binary literal > 8KB

# Zero Cost Plans

- Some query plans may not be cached, called “zero-cost plans”
  - extremely inexpensive to compile
  - cache could fill up quickly with invaluable plans
- SQL 2000 never caches them
  - SQL 2005 makes some exceptions
- SQL 2005 exceptions
  - zero cost cursor fetch plans are cached because they are likely to be reused many times
  - any batch that includes BEGIN TRAN, COMMIT TRAN, or a SET statement will be cached
    - in an attempt to avoid repeated parsing of certain batches that are frequently executed by the SQL Server ODBC driver and OLEDB provider
  - any plan that undergoes full optimization
    - encompasses nearly every SELECT, INSERT, UPDATE, or DELETE query

# Plan Recompilation

- Certain changes in db cause an execution plan to be either inefficient or invalid
  - based on the new state of the database
- Conditions that invalidate a plan
  - Changes made to a table or view referenced by the query (ALTER TABLE and ALTER VIEW)
  - Changes to any indexes used by the execution plan
  - Updates on statistics used by the execution plan, generated either explicitly from a statement, such as UPDATE STATISTICS, or generated automatically
  - Dropping an index used by the execution plan
  - An explicit call to sp_recompile
  - Large numbers of changes to keys (generated by INSERT or DELETE statements from other users that modify a table referenced by the query)
  - For tables with triggers, if the number of rows in the inserted or deleted tables grows significantly
  - Executing a stored procedure using the WITH RECOMPILE option
- Source: http://msdn2.microsoft.com/en-us/library/ms181055.aspx

# Plan Recompilation

- SQL 2000
  - whole batch, whether submitted through a stored procedure, trigger, ad-hoc batch, or prepared statement, is recompiled
- SQL 2005
  - only the statement inside the batch that causes recompilation is recompiled
  - Statement-level recompilation benefits performance
- Observe in Profiler
  - SP:Recompile
    - In 2005, TextData column of this event is populated
  - 2005 adds new event
    - SQL:StmtRecompile, reports statement-level recompilations

# Operations that clear Procedure Cache in 2k5

- Certain database maintenance operations or regular transaction operations clear the whole procedure cache
  - Any of these database operations
    - Detach, restore, change state to Off/Online
    - Rebuild transaction log for a database
    - Specify various options via ALTER Database
    - Run DBCC CheckDB (though no longer as of SP2)
  - Have DB with AUTO_CLOSE option set to ON
    - When no user connection references or uses the database, the background task tries to close and shut down the database automatically
  - Run several queries against DB w/ default options, then drop database
- Source: http://support.microsoft.com/default.aspx?scid=kb;en-us;917828

# Impact of Unqualified Tablenames

- Plan sharing: ability for different users to reuse the same compiled plan
- Failure to use schema-qualified table names will prevent plan sharing
  - the dbo in dbo.MyTable is the schema
- Example
  - SELECT * FROM MyTable
    - For user Joe, could refer to table Joe.MyTable, owned by Joe
    - If user Dan executes same query, could refer to a different table named Dan.MyTable
  - A table name without a schema reference is ambiguous
- Indicated in indicated sys.dm_exec_plan_attributes by a special user_id value of -2
- Why worry? ...

# Impact of Unqualified Tablenames

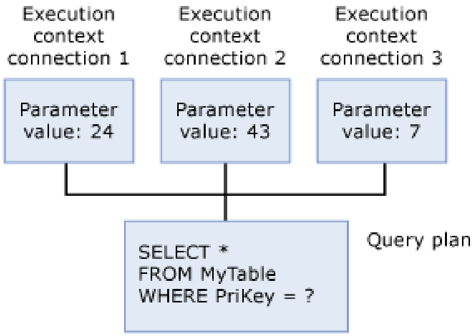
- Important for applications that connect to db with different login for each end user
  - Consider 100 logins executing same unqualified query
  - Will have 100 copies of query plan in cache
- Cache will be 100 times larger than it needs to be
  - And overall CPU cost paid for query compilation will be up to 100 times higher
- Applies only to queries in ad hoc or prepared batches
  - Queries in stored procedures, triggers, or user-defined functions are unambiguous even without schema-qualification
  - tables referenced by a T-SQL object always resolve as if they were being referenced by the user that created the object

# Understanding parameterization

- SQL Server creates new query plan for each unique SQL statement
  - Even if statement varies by a WHERE value, such as a search by one state, then one by another state
    - Engine will not reuse the same query plan
- Parameterization increases the ability of SQL Server to reuse compiled
  - **In Code**: many may know this can be controlled in code
    - Also known as explicit parameterization
  - **By Config**: can also be controlled in the database
    - And for particular queries that were not handled in code!

# Parameterized queries

- One solution: parameterized queries
  - Can cause DBMS to reuse previous query plan
    - If a given statement has variable values
  - Also reused by multiple users, if same SQL statement



  - Could save significant time (for query compilation)

b,c

# Parameterized queries

- In ADO.NET, ODBC, ADO, or via sp_executesql
  - specify parameter holders, data types, lengths
  - Tells DB to expect variations by given value(s)
- Parameterized queries also known as
  - Using “bind variables”, prepared statements
  - Opposite of “direct” or “immediate” execution

# Non-parameterized query (C#)

```
static private bool RunQueriesNotParameterized()
{
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cn;

    for (int i = 0; i < QueryIterations; i++)
    {
      cmd.CommandText = @”
      SELECT *
      FROM Sales.SalesOrderHeader h
      LEFT OUTER JOIN Sales.SalesOrderDetail d ON h.SalesOrderID =
                   d.SalesOrderID
      LEFT OUTER JOIN Sales.vSalesPerson p ON h.SalesPersonID =
                   p.SalesPersonID
      LEFT OUTER JOIN Sales.vSalesPersonSalesByFiscalYears sfy
                   ON sfy.SalesPersonID = h.SalesPersonID
      WHERE h.OrderDate BETWEEN ‘20040101’ AND ‘20040110’
                   AND h.SalesPersonID = “ + i.ToString();
    cmd.ExecuteNonQuery();
    }
return true;
}
```

# Parameterized query

```
static private bool RunQueriesParameterized()
{
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cn;
    cmd.CommandText = @”
      SELECT *
      FROM Sales.SalesOrderHeader h
      LEFT OUTER JOIN Sales.SalesOrderDetail d ON h.SalesOrderID =
                  d.SalesOrderID
      LEFT OUTER JOIN Sales.vSalesPerson p ON h.SalesPersonID =
                  p.SalesPersonID
      LEFT OUTER JOIN Sales.vSalesPersonSalesByFiscalYears sfy
                  ON sfy.SalesPersonID = h.SalesPersonID
      WHERE h.OrderDate BETWEEN @StartDate AND @EndDate
                  AND h.SalesPersonID = @SalesPersonID”;
                  cmd.Parameters.Add(“@StartDate”, SqlDbType.DateTime);
                  cmd.Parameters.Add(“@EndDate”, SqlDbType.DateTime);
                  cmd.Parameters.Add(“@SalesPersonID”, SqlDbType.Int);
    ...
```

# Parameterized query

```
...
   for (int i = 0; i < QueryIterations; i++)
   {
     cmd.Parameters[“@SalesPersonID”].Value = i;
     cmd.Parameters[“@StartDate”].Value = DateTime.Parse(“2004/01/01”,
               CultureInfo.InvariantCulture);
     cmd.Parameters[“@EndDate”].Value = DateTime.Parse(“2004/01/10”,
               CultureInfo.InvariantCulture);
     cmd.ExecuteNonQuery();
   }
return true;
}
```

# Other ways to parameterize in code

- In ODBC
  - use SQLExecDirect or SQLPrepare/SQLExecute with ? as a placeholder for each parameter
- In ADO
  - execute query with a Command object after explicitly populating Command object’s Parameters collection
- In T-SQL
  - use the sp_executesql stored procedure to explicitly parameterize a query
- In CFML
  - Use CFQUERYPARAM

# Observing parameterized queries

- Using SQL Server Profiler
  - Stored Procedures event (yes, even for SQL not in SPs)
    - RPC:Starting/Completed; SQL:StmtStarting/Completed
  - Performance event
    - Showplan All/XML for Query Compile
- Query plans include those compiled for
  - stored procedures
  - parameterized queries
  - ad-hoc SQL queries
    - Queries that are not SP or parameterized

# What if you can’t change the code?

- SQL Server comes to the rescue
  - In some ways, automatically
  - In others, requires configuration
- Forced parameterization
- Simple, or auto-parameterization

# Forced parameterization

- New db level option in 2005
  - ALTER DATABASE dbname SET PARAMETERIZATION FORCED|SIMPLE
- Any query that arrives at the server while that database has context will be implicitly parameterized
- Can’t work for all queries
  - literals in a SELECT’s column list or certain query hints will prevent
  - See BOL for more examples
- Also also prevent queries from making use of indexes on views
- Sounds very compelling. Be sure to test!

# Simple Parameterization

- If Forced is not enabled, SQL Server uses a simple, more limited form of auto-parameterization: Simple
  - In SQL Server 2000, was indeed simply called “auto-parameterization”
- Implicitly parameterizes queries as long as they don’t use the folllowing:
  - Reference to more than one table
  - SELECT with TOP, IN clauses, or OR expressions, Subqueries, GROUP BY, UNION, SELECT with DISTINCT, SELECT INTO
  - Any query hint
  - UPDATE SET @variable=...
  - DELETE or UPDATE with a FROM clause
  - Reference to fulltext catalogs, linked server queries, or table variables
  - A predicate of the form Expression <> Constant
  - References to most intrinsic functions
- More examples: http://www.sqlteam.com/article/introduction-to-parameterization-in-sql-server

## Plan Guides to Force Parameterization

- Plan guides are yet another solution
  - Instead of global setting for entire database, declare way to handle particular query
  - Can then either:
    - force paramerization for that query
    - or optimize the query plan built using a particular representative value
- For more examples, features, restrictions, best practices, useful case study
  - http://www.microsoft.com/technet/prodtechnol/sql/2005/frcqupln.mspx#ETD

# Plan Guides to Force Parameterization

- Consider the example below:
  - `select t1.col2, t2.col2 from t1 join t2 on t1.col1 = t2.col1 and t1.col2 > 5`
  - does not get parameterized under parameterization
- Can create plan guide for this query

```
declare @stmt nvarchar(max), @params
  nvarchar(max);
exec sp_get_query_template N'select t1.col2,
  t2.col2 from t1 join t2 on t1.col1 = t2.col1
  and t1.col2 > 5', @stmt output, @params
  output
exec sp_create_plan_guide N'JoinGuide', @stmt,
  N'Template', NULL, @params,
  N'OPTION(PARAMETERIZATION FORCED)‘
```

# Parameter Sniffing

- Be careful about forced parameterization
  - Parameter values specified during first execution of a query cause query optimizer attempts to pick best plan for that
  - That may not make sense for all query values
  - Can cause severe problems, very slow queries
    - Definitely take time to learn more on detecting, resolving
  - See the book or blog series for more info

# How to determine if DBs use forced parameterization

- sys.databases - name, is_parameterization_forced
  - 0 - means no, uses simple parameterization
  - Source: http://blogs.msdn.com/sqlprogrammability/archive/2007/01/20/trouble-shooting-query-performance-issues-related-to-plan-cache-in-sql-2005-rtm-and-sp1.aspx

# When Parameterized Queries Can Hurt

- Query plan may be created for one value that does not work well for other values
  - Obvious example is NULL
  - A query plan created for SELECT ... WHERE col=null would not use index
    - So could force tablescan
  - Subsequent query using “real” value could end up using that query plan...BAD!
  - Could happen for other than null (value with low distribution across records vs value with high)
- SQL Server offers some solutions to help
  - Can offer WITH RECOMPILE hint
  - Can also create plan guides and template plan guides
  - Beyond the scope of this talk to discuss in details

# Performance implication of query plan caching

- The query plan cache requires memory
  - So using it most effectively can be important
- Failure to use parameterized queries means unique entries for statements varying by some arg
  - Parameterized queries would reuse one cached statement
- DMV analysis can help address evaluating performance
  - For determining which so parameterize, find SQL statements that look alike
    - But vary by some one parameter
  - For multiple user/no schema problem, will find entries with identical sql

f,g,h

# Change in SQL2k5 SP2

- As compared to RTM and SP1
- Reduced contention in the creation and eviction of plan cache entries
  - To increase throughput and eliminating an Out of Memory error
- Implemented fairness policy in evicting cache entries across all NUMA nodes
  - To avoid a drop in throughput due to contention
- Decreased number of plan cache entries by not caching some zero-cost plans
  - dynamic batches containing at least one set (option) statement and/or at least one transaction statement (begin/commit/save/rollback transaction statement) are no longer cached, with some exceptions
- Aligned plan cache size limit to be similar to that of SQL Server 2000
  - More on next page
- Source: http://blogs.msdn.com/sqlprogrammability/archive/2007/01/23/4-0-useful-queries-on-dmv-s-to-understand-plan-cache-behavior.aspx
  - “3.2 Improvements made to Plan Cache behavior in SQL Server 2005 SP2”

# Change in Max Cachestore Memory Use in SP2

| **SQL Server Version** | **Internal Memory Pressure Indication in Cachestore** |
|---|---|
| SQL Server 2005 RTM & SP1 | 75% of server memory from 0-8GB + 50% of server memory from 8Gb-64GB + 25% of server memory > 64GB |
| SQL Server 2005 **SP2** | 75% of server memory from **0-4GB** + **10%** of server memory from **4Gb**-64GB + **5%** of server memory > 64GB |
| SQL Server 2000 | SQL Server 2000 4GB upper cap on the plan cache |

- Example for a SQL Server with 32Gb total SQL server memory:
  - SQL Server 2005 RTM and SP1, limit will be 75% X 8 + 50% X (32 - 8) = **18GB**
  - SQL Server 2005 SP2, limit will be 75% X 4 + 10% X (32-4) = **5.8GB**

# Indicators that this has happened

- Performance object: Process>Counter: %Processor Time>Instance: sqlservr
  - The value of this counter will increase because of increased CPU activity
  - Essentially, the whole procedure cache is cleared if this issue occurs. Therefore, subsequent requests must generate new plans to be cached
  - This behavior will slightly increase CPU activity.
- Performance object: SQLServer:Plan Cache>Counter: Cache Object Counts>Instance: _Total
  - Performance object: SQLServer:Plan Cache>Counter: Cache Pages>Instance: _Total
  - The values of these counters will suddenly decrease.
  - Note For a named instance of SQL Server 2005, the performance object is named MSSQL$InstanceName:Plan Cache.
- Performance object: SQLServer:SQL Statistics>Counter: SQLCompilations/sec
  - The value of this counter will significantly increase after this incident.
  - Note For a named instance of SQL Server 2005, the performance object is named MSSQL$InstanceName: SQL Statistics.
- If you capture SQL Profiler Trace by using the SP:CacheRemove event
  - notice that this event is generated together with the following TextData column value when this issue occurs: "Entire Procedure Cache Flushed"

# Other topics covered in the article series

- Structure of the Plan Cache and Types of Cached Objects
- Sql_Handle and Plan_Handle Explained
- How Cache Lookups Work
- Query Parameterization
- Retrieving Query Plans from Plan Cache DMV's
- Best Programming Practices
- Costing Cache Entries
- Factors that affect Batch Cache-ability
- Memory Pressure Limits
- Plan Cache Flush
- Temporary Tables, Table Variables and Recompiles
- Plan Cache Trace Events and Performance
-
- Machine Configuration Information That Can Impact Plan Cache Size/Performance
- Diagnosing Plan Cache Related Performance Problems and Suggested Solutions
- Changes in Caching Behavior between SQL Server 2000, SQL Server 2005 RTM and SQL Server 2005 SP2
- Useful Queries on DMV's to understand Plan Cache Behavior

# Other resources

- http://www.sqlteam.com/article/what-query-plans-are-in-sql-server-memory
- http://www.sqlteam.com/article/what-data-is-in-sql-server-memory
- http://www.sqlteam.com/article/introduction-to-parameterization-in-sql-server
- http://blogs.msdn.com/sqlprogrammability/archive/2007/01/23/4-0-useful-queries-on-dmv-s-to-understand-plan-cache-behavior.aspx
  - Many examples
- http://blogs.msdn.com/irenak/archive/2007/04/20/sysk-333-what-query-plans-are-cached-in-sql-server.aspx
- http://blogs.msdn.com/cbiyikoglu/archive/2005/11/03/488920.aspx

# Summary

- Thinking through DB request processing
- Observing the plan cache
    - Code for evaluating query plan usage
- Cache Plan Management
    - How cache is managed, may be cleared
- Understanding query parameterization
    - Parameterized queries
    - Forced
    - Simple/Auto
- Monitoring & performance implications
- Changes in 2005, especially SP2

# Questions on presentation

- Charlie Arehart
  - charlie@carehart.org
- So, who learned something new?
- I’d really appreciate your feedback
  - http://carehart.org/feedback/
- Also available for consulting
  - Remote or on-site