

How to save the Plan Cache

by Alessandro Mortola



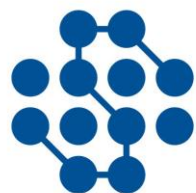


UNIVERSITÀ DEGLI STUDI DI PARMA



Bi Factory

DATA KNOWLEDGE ADVISOR



DATA SKILLS
UNDERSTANDING THE WORLD



Lucient⁴
ITALIA



- I have worked with all versions from Sql Server 7.0 to Sql Server 2022
- I spoke several times at Sql Saturday, Data Saturday and SqlStart! editions
- I am an author for sqlservercentral.com
- I am certified Sql Server 2016 Dev
- I currently work for a well-known Italian company as
 - Sql Server DBA
 - Analyst



@AlexMortola



alessandro.mortola@gmail.com



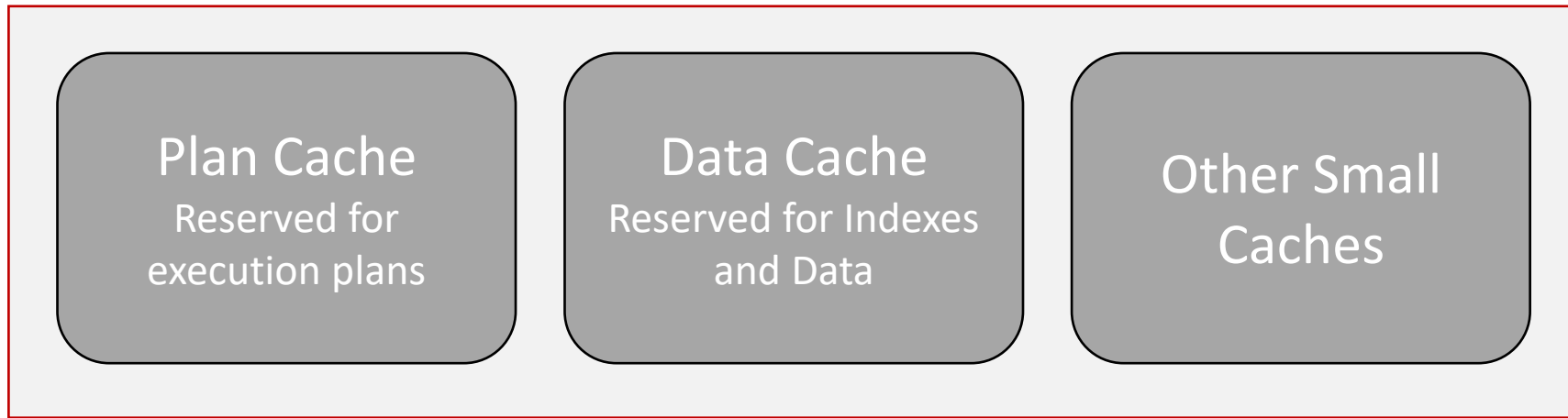
<https://www.linkedin.com/in/alessandromortola>



Agenda

- **The Plan Cache and its limits**
- **How to use the Plan Cache properly**
 - **Ad hoc queries vs Parameterized**
- **Parameterization**
- **What if your queries are not parameterized?**
 - **Simple Parameterization**
 - **Forced Parameterization**
- **Other options for Ad hoc queries**
 - **Optimize for Ad hoc Workloads**
 - **TF 253**

Sql Server Cache



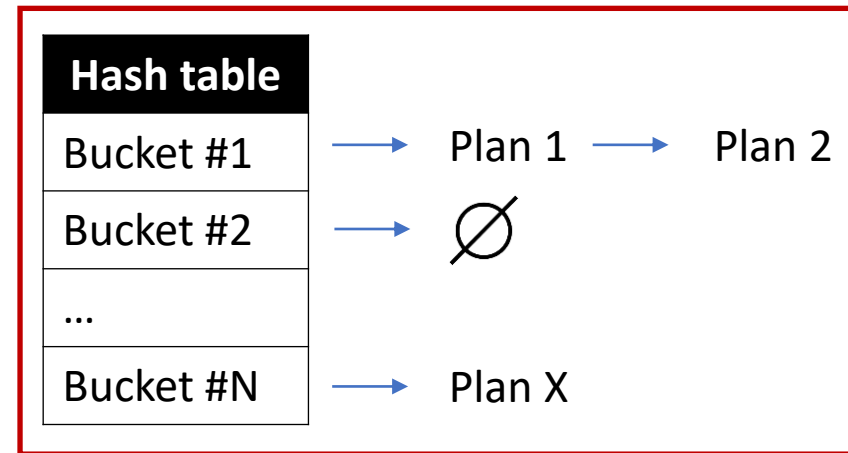
STORES

- ➔ **Object Plans** (stored procedures, triggers, functions)
- ➔ **Sql Plans** (Ad hoc, autoparameterized, dynamic, prepared queries)
- ➔ **Bound Trees** (data structure used during compilation for views, constraints and defaults)
- ➔ **Extended Stored Procedures** (system procedures. E.g. sp_executesql, xp_cmdshell)

The Plan Cache

Each Store (Object Plans, Sql Plans, Bound Trees and Extended Stored Procedures) contains a hash table to keep track of all the plans in that particular store. Each bucket in the hash table contains zero, one, or more cached plans.

The DMV **sys.dm_os_memory_cache_hash_tables** gives us information about each hash table.



Name	Plan cache store	Buckets_count	Buckets_max_length	Buckets_avg_length
Object Plans	CACHESTORE_OBJCP	40009
SQL Plans	CACHESTORE_SQLCP	40009
Bound Trees	CACHESTORE_PHDR	4001
Extended Stored Procedures	CACHESTORE_XPROC	127

The Plan Cache

The DMV **sys.dm_os_memory_cache_counters** provides information about the cache entries allocated.

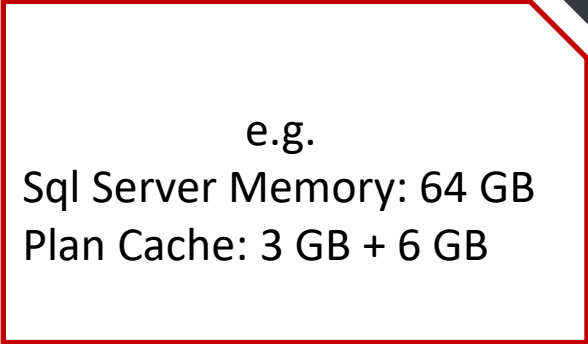
Name	Plan cache store	Pages_kb	Entries_count
Object Plans	CACHESTORE_OBJCP
SQL Plans	CACHESTORE_SQLCP
Bound Trees	CACHESTORE_PHDR
Extended Stored Procedures	CACHESTORE_XPROC

The Plan Cache Pressure Limit

Starting from Sql Server 2005 SP2, there are two limitations related to the plan cache:

Total size (depending on the server memory)


- 75% of visible target memory from 0 to 4 GB
- + 10% of visible target memory from 4 GB to 64 GB
- + 5% of visible target memory > 64 GB



e.g.
Sql Server Memory: 64 GB
Plan Cache: 3 GB + 6 GB

Number of entries

SQL Server also indicates memory pressure when the number of plans in a store exceeds four times the hash table size for that store, regardless of the actual size of the plans.



These limits can be raised with the documented 8032 (total size) and trace flag 174 (number of entries).

Two query types to deal with

Adhoc vs Parameterized

The Cache

Ad hoc

```
select SalesOrderID, OrderQty, ProductID
from Sales.SalesOrderDetailEnlarged
where ProductID = 777;
```

Compiled

objtype	usecounts	text
Adhoc	1	select SalesOrderID, OrderQty, ProductID from Sales.SalesOrderDetailEnlarged where ProductID = 777;

```
select SalesOrderID, OrderQty, ProductID
from Sales.SalesOrderDetailEnlarged
where ProductID = 778;
```

Compiled

objtype	usecounts	text
Adhoc	1	select SalesOrderID, OrderQty, ProductID from Sales.SalesOrderDetailEnlarged where ProductID = 778;

Parameterized

```
exec sp_executesql
@stmt = N'select SalesOrderID,
OrderQty, ProductID
from Sales.SalesOrderDetailEnlarged
where ProductID = @pid;',
@params = N'@pid int', @pid = 777;
```

Compiled

objtype	usecounts	text
Prepared	2	(@pid int)select SalesOrderID, OrderQty, ProductID from Sales.SalesOrderDetailEnlarged where ProductID = @pid;

Uses the plan in the cache

```
exec sp_executesql
@stmt = N'select SalesOrderID,
OrderQty, ProductID
from Sales.SalesOrderDetailEnlarged
where ProductID = @pid;',
@params = N'@pid int', @pid = 778;
```

To keep in mind

#1

Compiling a query has a cost in terms of time and resources (e.g. CPU, RAM)

#2

SQL Server has an efficient algorithm to find an existing plan related to a specific statement. The minimal resources that are used by this scan are less than the resources that should be used for compiling every statement

#3

The way SQL Server looks at the plan in cache is by calculating a hash value of the query text

#4

Ad hoc queries are the main responsible for plan pollution

#5

Execution plans remain in the cache as long as there is enough memory to store them. When memory pressure exists, the SQL Server uses a cost-based approach to determine which plans to remove from the cache





How to use the
Plan Cache
properly

PARAMETERIZATION

#1 – Stored procedures

CacheObjType	ObjType	Estimated number of rows based on
Compiled Plan	Proc	Statistics Histogram

```
create procedure dbo.spGetOrderDetailsFromProductID @productID int
as
    select SalesOrderID, CarrierTrackingNumber
    from Sales.SalesOrderDetailEnlarged
    where ProductID = @productID
    order by rowguid;
go

exec dbo.spGetOrderDetailsFromProductID @productID = 870;
go
```



PARAMETERIZATION

#2 – Dynamic Sql with sp_executesql

CacheObjType	ObjType	Estimated number of rows based on
Compiled Plan	Prepared	Statistics Histogram

```
declare @stmt nvarchar(max),
        @params nvarchar(max);

set @stmt = N'select SalesOrderID, CarrierTrackingNumber
            from Sales.SalesOrderDetailEnlarged
            where ProductID = @productID
            order by rowguid;';

set @params = N'@productID int';

exec sp_executesql @stmt, @params, 870;
```

PARAMETERIZATION

#3 – Using sp_prepare and sp_execute

CacheObjType	ObjType	Estimated number of rows based on
Compiled Plan	Prepared	Statistics Header & Density Vector

```
DECLARE @PreparedStatementNumber INT;  
  
EXEC sp_prepare @PreparedStatementNumber OUTPUT,  
    N'@productID int',  
    N'select SalesOrderID, CarrierTrackingNumber  
    from Sales.SalesOrderDetailEnlarged  
    where ProductID = @productID  
    order by rowguid;'  
  
exec sp_execute @PreparedStatementNumber, 870;  
  
exec sp_unprepare @PreparedStatementNumber;
```



Pay attention: Variables are NOT parameters

- Queries that filter by literals or **parameters** (with the exception of `sp_prepare`) use statistics **HISTOGRAM**
- Queries that filter by **variables** use statistics **DENSITY_VECTOR**

```
declare @pid int = 870;
```

```
select SalesOrderID, CarrierTrackingNumber  
from Sales.SalesOrderDetailEnlarged  
where ProductID = @pid  
order by rowguid;
```



CacheObjType	ObjType
Compiled Plan	Adhoc

```
create procedure [dbo].[spTest] (@prodid int)  
as  
declare @pid int = @prodid;
```

```
select SalesOrderID, CarrierTrackingNumber  
from Sales.SalesOrderDetailEnlarged  
where ProductID = @pid  
order by rowguid;  
go
```



CacheObjType	ObjType
Compiled Plan	Proc



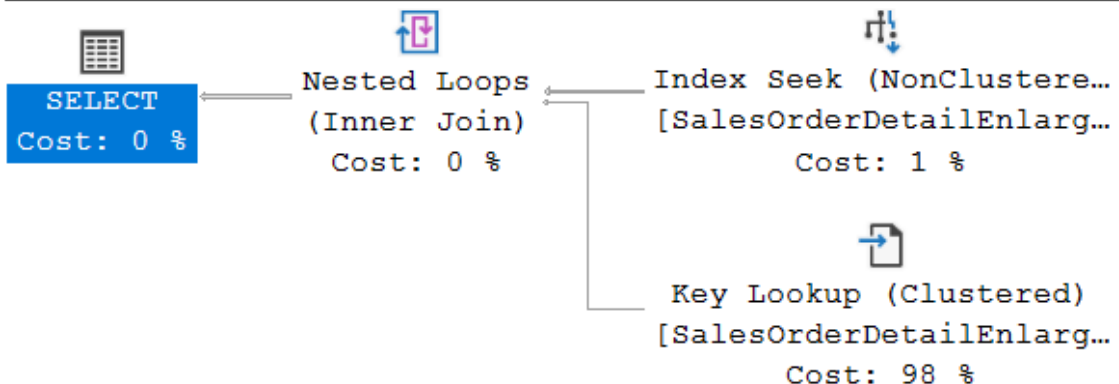
Parameter Sniffing (aka Parameter Sensitivity)

It is a performance optimization. The first time a user runs a parameterized query, Sql Server chooses a “good enough” plan basing the decision on the parameter values. That plan is used for the next executions.

What can go wrong? If the data distribution is unevenly distributed, the compiled plan may not fit all possible parameter values and performance problems may occur.

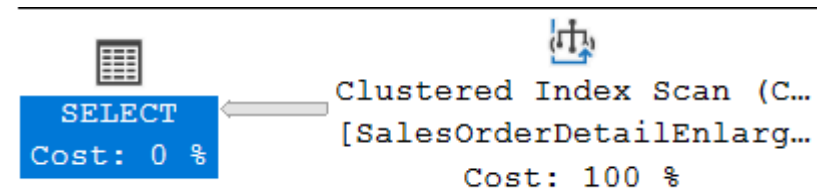
--@pid = 897 → 78 rows

```
select en.SalesOrderID, en.CarrierTrackingNumber
from Sales.SalesOrderDetailEnlarged en
where en.ProductID = @pid;
```



--@pid = 870 → 182832 rows

```
select en.SalesOrderID, en.CarrierTrackingNumber
from Sales.SalesOrderDetailEnlarged en
where en.ProductID = @pid;
```



Parameter Sniffing – An interesting metaphor

By Guy Glanster



Parameter Sniffing – How to deal with it

```
select SalesOrderID, CarrierTrackingNumber  
from Sales.SalesOrderDetailEnlarged  
where ProductID = @pid
```

Optimizing for a typical parameter

```
option (optimize for (@pid = 123));
```

Optimizing on every execution

```
option (recompile);
```

Optimizing for UNKNOWN

```
option (optimize for unknown);
```

Using local variables

```
declare @lpid int = @pid;  
select SalesOrderID, CarrierTrackingNumber  
from Sales.SalesOrderDetailEnlarged  
where ProductID = @lpid;
```

Disabling PS at Query level

```
option (use hint ('DISABLE_PARAMETER_SNIFFING'));
```

Disabling PS at Database level

```
alter database scoped configuration set parameter_sniffing = off
```


Disabling PS at Instance level

```
Trace Flag 4136
```



Parameter Sensitive Plan Optimization

New in Sql Server 2022 !!!

- 
- It is part of the Intelligent Query Processing family of features
 - It is available for each edition
 - It addresses the scenario where a single cached plan for a parameterized query isn't optimal for all possible incoming parameter values. This is the case with non-uniform data distributions
 - The *dispatcher plan* contains the optimization logic called a *dispatcher expression*. The dispatcher plan maps to *query variants* based on the cardinality range boundary values predicates

Parameter Sensitive Plan Optimization

New in Sql Server 2022 !!!

```
select *  
from t1  
where id = @pid;
```

It checks if PSPO requirements are met

It sets the high and low boundary values looking at the statistics

Cardinality

Query Variant 1

Query Variant 2

Query Variant 3

Low
boundary

High
boundary

Seek + Lookup + small
memory grant

Scan + medium
memory grant

Scan + large memory
grant

PSP optimization skipped reasons

- ❖ There are 40 possible reasons why PSP optimization can be skipped and they are returned from the following query:

```
SELECT map_key, map_value  
FROM sys.dm_xe_map_values  
WHERE name = 'psp_skipped_reason_enum';
```

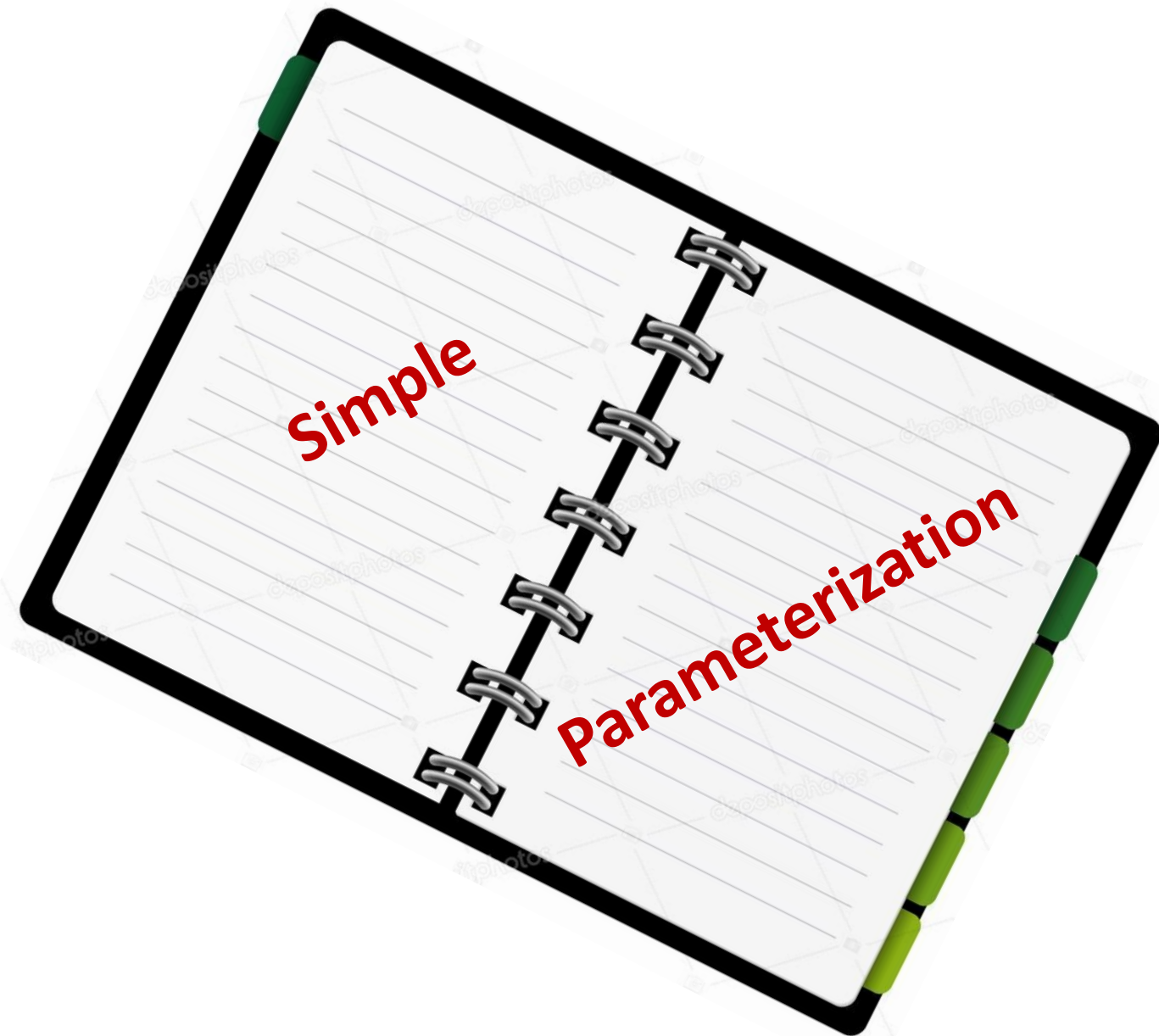
- NoParameter
- CompatLevelBelow160
- UnsupportedComparisonType
- SkewnessThresholdNotMet
- HasLocalVar
- SystemDB
- ...

- ❖ The Extended Event **parameter_sensitive_plan_optimization_skipped_reason** occurs when the PSP optimization feature is skipped.

PSP optimization is not perfect yet

- We have more possible execution plans in cache, each of which can be sniffed and the medium plans are the more vulnerable
- PSP optimization currently only works with equality predicates
- It is a young technology. It will be better in future releases 😊

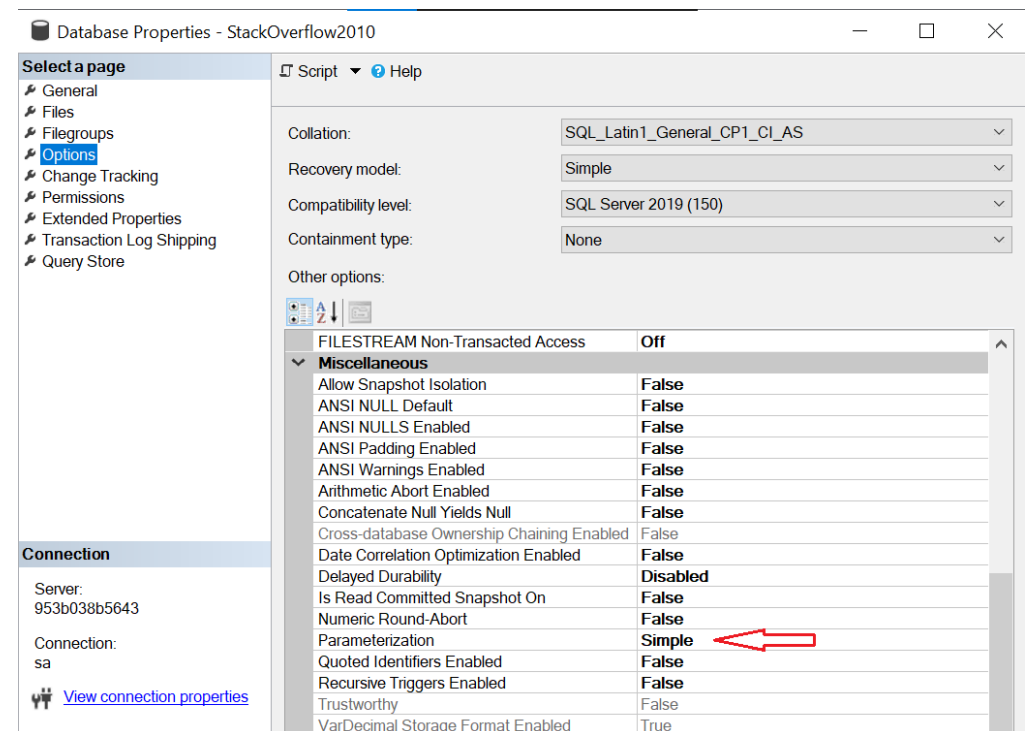




#1. What if your queries are not parameterized ?

How Sql Server protects itself against “single use plans”: Simple Parameterization

- With Simple Parameterization, before Sql Server 2005 known as auto-parameterization, Sql Server attempts to replace constant literal values with parameter markers
- It is set at database level and **Simple** is the default value



Simple Parameterization – When it is applied

The Simple Parameterization is applied to “**simple and safe**” queries

❖ **Simple** Parameterization will NOT happen if the statement contains any of these:

JOIN	TOP	BULK INSERT	UNION
INTO	DISTINCT	GROUP BY	HAVING
Sub queries	Query hint	Constant comparison	...

❖ **Safe** Execution Plan: *regardless of the provided input parameter values, the query must always lead to the same execution plan [Klaus Aschenbrenner]*

What is supported and not supported by the Simple Parameterization

	Supported	Not Supported
Functions	<ul style="list-style-type: none">• CAST & CONVERT• ABS• FLOOR• ...	<ul style="list-style-type: none">• LOWER• UPPER• CEILING• ...
Global Variables	<ul style="list-style-type: none">• @@SPID• @@TRANCOUNT• ...	<ul style="list-style-type: none">• @@ROWCOUNT• @@IDENTITY• ...

The list of intrinsic functions compatible with simple parameterization is quite limited and undocumented. [Paul White]

Simple Parameterization – How it works

- Starting from an Ad hoc, Simple and Safe query, Sql Server substitutes literal values with parameters obtaining a parameterized plan
- Sql Server tries guessing the data type using the smallest compatible subtype.
E.g.: with “**WHERE BusinessEntityId = 1**” the parameter will be defined as **tinyint**
- In the plan cache we have one **Compiled Plan / Prepared** statement with the full plan and N Adhoc shells (smaller than a full plan) that are really pointers to the Prepared
- Shell plans allow to bypass the parsing, parameter replacement, normalization and decoding stages
- The original text is manipulated and standardized (more details later)



Simple Parameterization – Data Type Inference



Numbers

- For strings of numbers without decimal points, Sql Server chooses from tinyint, smallint and int. For numbers beyond the range of integer, Sql Server uses numeric with the smallest possible precision
- Strings of numbers with decimal precision are interpreted as numeric with a precision and scale just large enough to contain the value provided
- Strings prefixed with a currency symbol are interpreted as money
- Strings in scientific notation translate to float

Simple Parameterization – Data Type Inference

Datetime and Uniqueidentifier

The literal value must be provided in ODBC escape format. e.g:

- {d '1901-01-01'}
- {ts '1900-01-01 12:34:56.790'}
- {guid 'F85C72AB-15F7-49E9-A949-273C55A6C393'}

General string and Binary

They are typed as varchar(8000), nvarchar(4000) or varbinary(8000). MAX is used if the literal exceeds 8000 bytes

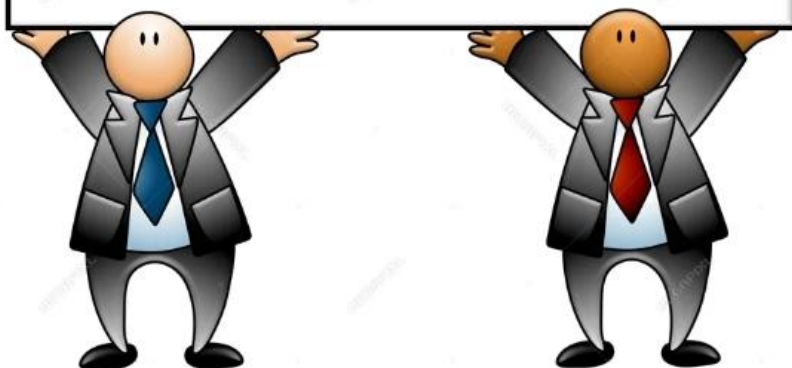
[Ref. Paul White]

Simple Parameterization – Text manipulation

Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

Text Manipulation



Query text manipulated by the Simple Parameterization

```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```

Simple Parameterization – Text manipulation

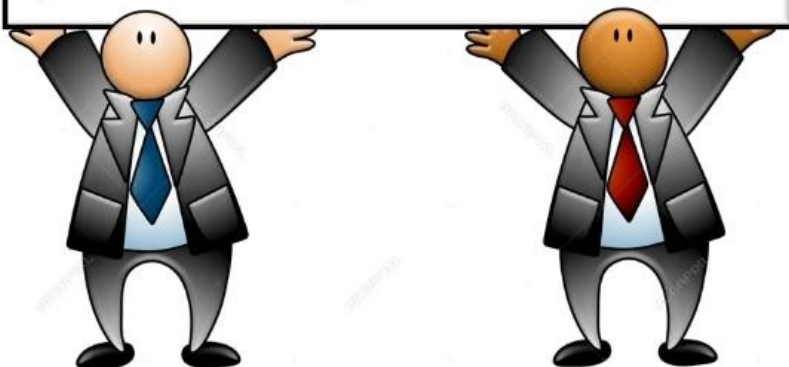
Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

Query text manipulated by the Simple Parameterization

```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```

- **Keywords: UPPERCASE**
- **Data types: lowercase**
- **ASC is added in ORDER BY clause**

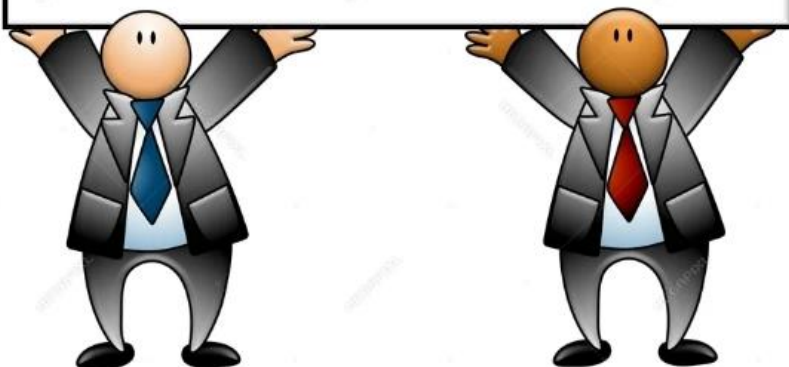


Simple Parameterization – Text manipulation

Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

- Global variables: lowercase
- Objects are delimited by square brackets



Query text manipulated by the Simple Parameterization

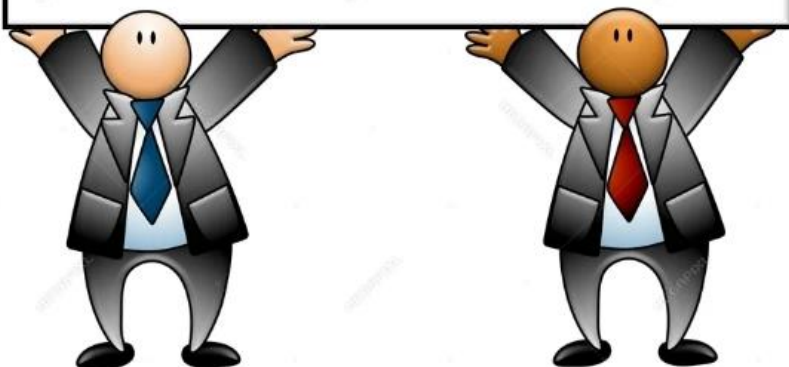
```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```


Simple Parameterization – Text manipulation

Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

- Unnecessary spaces and comments are removed
- CAST is replaced by CONVERT



Query text manipulated by the Simple Parameterization

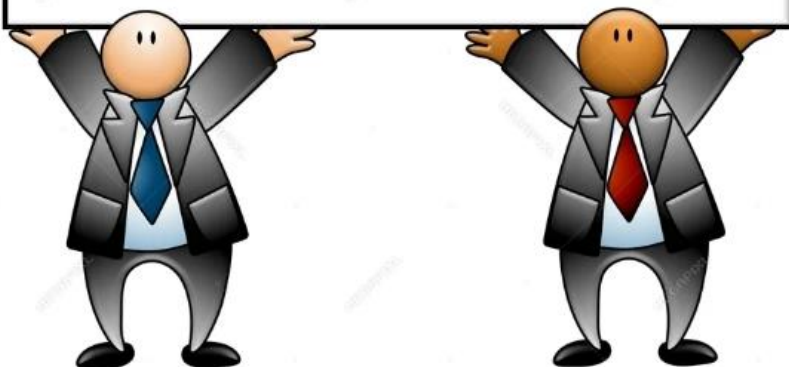
```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```

Simple Parameterization – Text manipulation

Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

- Parameter enumeration starts from @1
- Intrinsic functions: lowercase (but not CONVERT!)



Query text manipulated by the Simple Parameterization

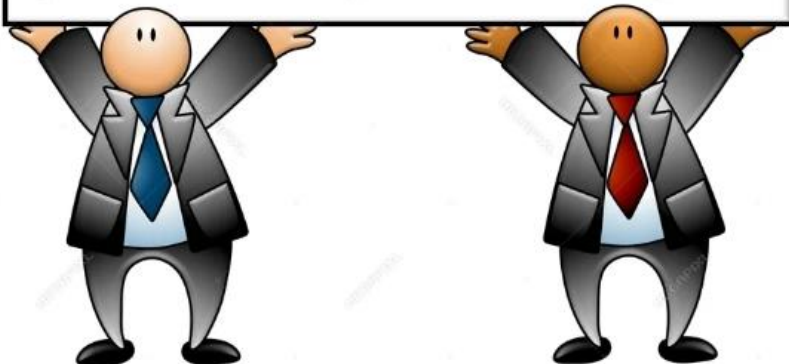
```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```

Simple Parameterization – Text manipulation

Original query text

```
select @@SPID,  
        CAST(ProductId as VARCHAR(10)) as ProdId,  
        FLOOR(StandardCost) as      IntStdCost --Comment!!!  
from Production.Product  
where ProductID = 1 and Name != 'X'  
order by ProductID
```

- AS used for aliases are removed
- != is substituted by <>
- ...



Query text manipulated by the Simple Parameterization

```
(@1 tinyint,@2 varchar(8000))  
SELECT @@spid,  
        CONVERT([varchar](10),[ProductId]) [ProdId],  
        floor([StandardCost]) [IntStdCost]  
FROM [Production].[Product]  
WHERE [ProductID]=@1 AND [Name]<>@2  
ORDER BY [ProductID] ASC
```

Simple Parameterization – Final notes

- It is set at database level
- The main drawback is due to the Data Type inference. E.g.:

```
select Name from Production.Product where ProductID = 1;
```

```
go
```

```
select Name from Production.Product where ProductID = 256;
```

```
go
```

```
select Name from Production.Product where ProductID = 32768;
```

```
go
```



tinyint



smallint



int





#2. What if your queries are not parameterized?

Forced Parameterization

- It was introduced in Sql Server 2005
- It tells SQL Server to parameterize all DML queries
- When the PARAMETERIZATION option is set to FORCED, any literal value that appears in a SELECT, INSERT, UPDATE, or DELETE statement, submitted in any form, is converted to a parameter during query compilation
- In the plan cache we have one **Compiled Plan / Prepared** statement with the full plan and N Compiled Plan / Adhoc shells (smaller than a full plan) that are really pointers to the Prepared statement

Forced Parameterization – When it is not applied

- Statements inside the bodies of stored procedures, triggers, or user-defined functions
- Statements inside a T-SQL cursor
- Statements that reference variables, such as WHERE T.col2 >= @bb
- Statements that contain more than 2,097 literals that are eligible for parameterization
- Statements that contain the RECOMPILE query hint
- *The <select_list> of any SELECT statement*
- *Constant-foldable expressions that are arguments of the +, -, *, /, and % operators. E.g.: WHERE T.col2 = 1 + 2*

Forced Parameterization – Data Type Inference

Numbers

- Integer literals that fit within int data type parameterize to int
- Larger integer literals that are part of predicates that involve comparison operators, parameterize to numeric (38,0)
- Larger literals that aren't parts of predicates that involve comparison operators parameterize to numeric whose precision is just large enough to support its size and whose scale is 0
- Fixed-point numeric literals parameterize to numeric
- Floating point numeric literals parameterize to float
- Money type literals parameterize to money



Forced Parameterization – Data Type Inference



NON parameterizable data types: Datetime and Uniqueidentifier

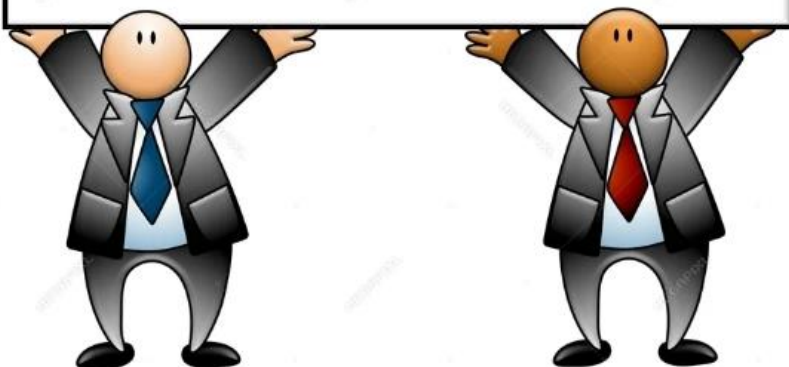
Constants specified by using ODBC extension syntax are NOT parameterized

General string and Binary

- Non-Unicode string literals parameterize to varchar(8000) or varchar(MAX)
- Unicode string literals parameterize to nvarchar(4000) or nvarchar(MAX)
- Binary literals parameterize to varbinary(8000) or varbinary(MAX)

Forced Parameterization – Text manipulation

Text Manipulation



Original query text

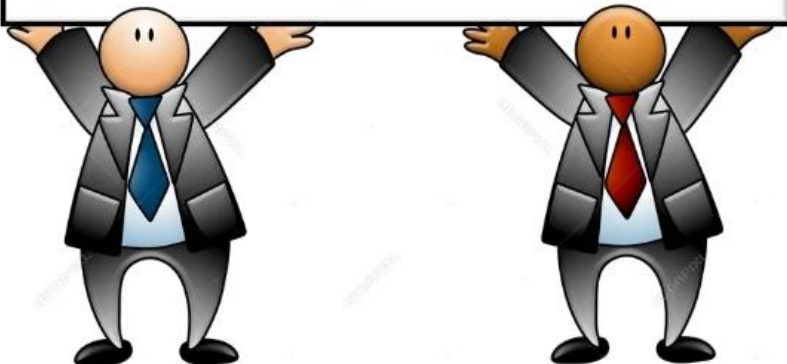
```
SELECT p.ProductId as ProductIdentifier,  --Comment!!!
       sc.Name AS SubCatName
FROM Production.Product p
INNER JOIN Production.ProductSubcategory sc
       ON p.ProductSubcategoryID = sc.ProductSubcategoryID
WHERE p.ProductID = 1
ORDER BY p.ProductID
```

Query text manipulated by the Forced Parameterization

```
(@0 int)
select p . ProductId as ProductIdentifier ,
       sc . Name as SubCatName
from Production . Product p
inner join Production . ProductSubcategory sc
       on p . ProductSubcategoryID = sc . ProductSubcategoryID
where p . ProductID = @0
order by p . ProductID
```

Forced Parameterization – Text manipulation

- **Keywords: lowercase**
- **Data type: lowercase**
- **Parameter enumeration starts from @0**



Original query text

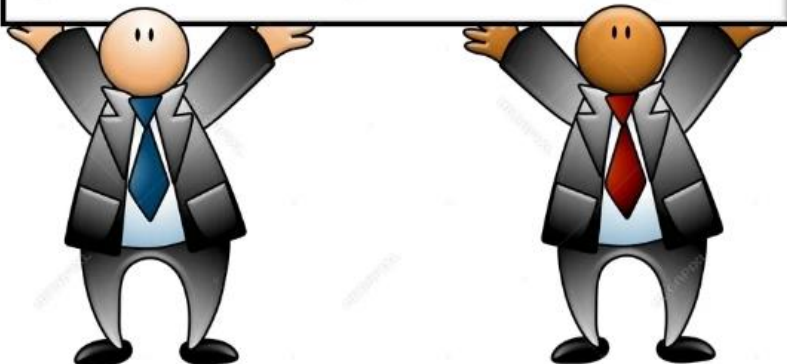
```
SELECT p.ProductId as ProductIdentifier,  --Comment!!!
       sc.Name AS SubCatName
FROM Production.Product p
INNER JOIN Production.ProductSubcategory sc
         ON p.ProductSubcategoryID = sc.ProductSubcategoryID
WHERE p.ProductID = 1
ORDER BY p.ProductID
```

Query text manipulated by the Forced Parameterization

```
(@0 int)
select p . ProductId as ProductIdentifier ,
       sc . Name as SubCatName
from Production . Product p
inner join Production . ProductSubcategory sc
         on p . ProductSubcategoryID = sc . ProductSubcategoryID
where p . ProductID = @0
order by p . ProductID
```

Forced Parameterization – Text manipulation

- Unnecessary spaces and comments are removed
- There is a space either side of the 'dot' between schema and object names



Original query text

```
SELECT p.ProductId as ProductIdentifier,  --Comment!!!
       sc.Name AS SubCatName
FROM Production.Product p
INNER JOIN Production.ProductSubcategory sc
       ON p.ProductSubcategoryID = sc.ProductSubcategoryID
WHERE p.ProductID = 1
ORDER BY p.ProductID
```

Query text manipulated by the Forced Parameterization

```
(@0 int)
select p . ProductId as ProductIdentifier ,
       sc . Name as SubCatName
from Production . Product p
inner join Production . ProductSubcategory sc
       on p . ProductSubcategoryID = sc . ProductSubcategoryID
where p . ProductID = @0
order by p . ProductID
```

Forced Parameterization – Final notes

- It is set at database level
- It can be set at query level with `OPTION (PARAMETERIZATION FORCED)`
- It doesn't have any particular drawbacks, even if it may cause a sub-optimal plan usage mainly in environments that rely heavily on indexed views and indexes on computed columns





Another option
for Ad hoc
queries

Optimize for Ad hoc Workloads

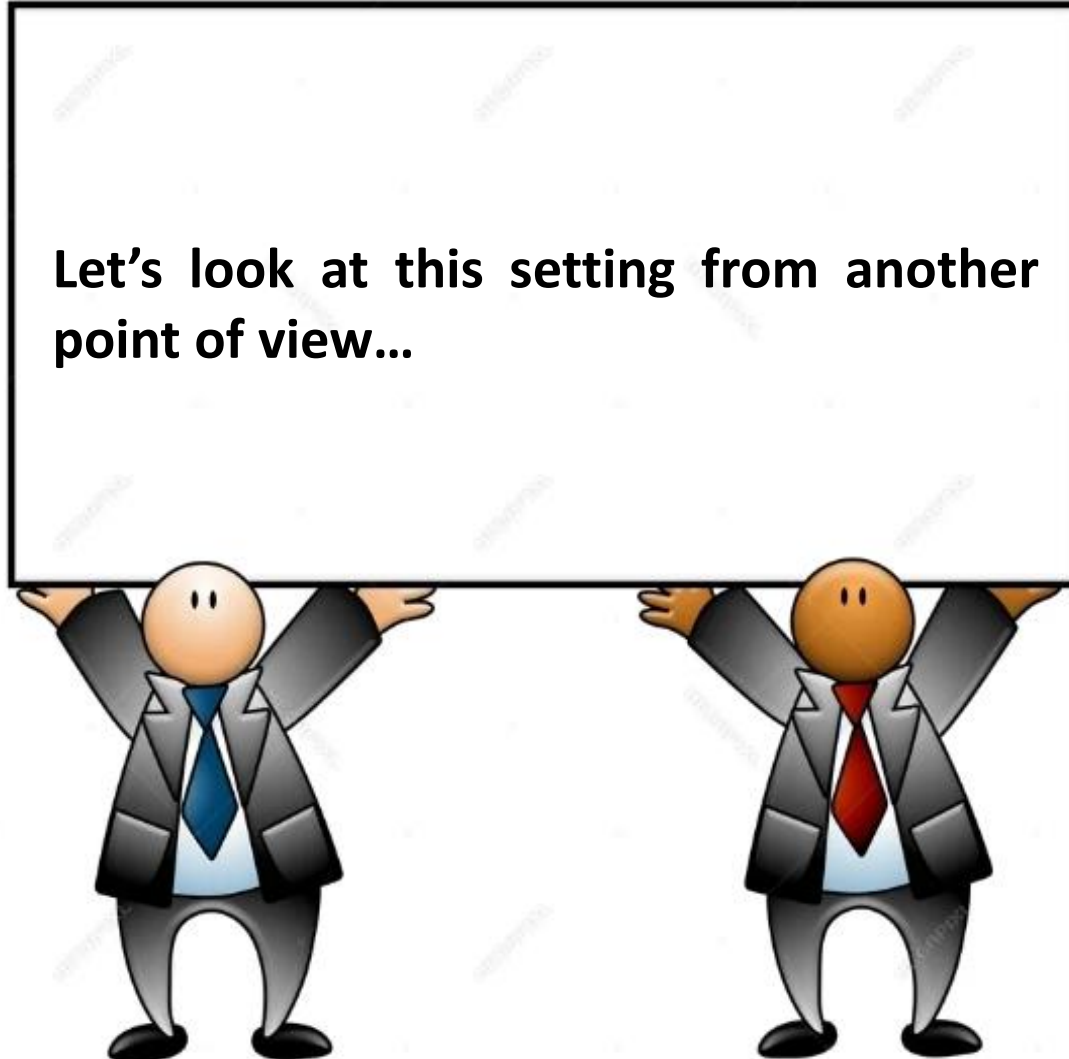
It is set at instance level
It has been available since Sql Server 2008

The behaviour

- When the Ad hoc query is compiled for the first time the Engine does not store the full plan, but only a small stub
- The compiled plan stub doesn't have an execution plan associated with it
- If the query is executed again, the stub is removed from the cache, the query is compiled again and the full execution plan is cached, ready for further reuse



Optimize for Ad hoc Workloads



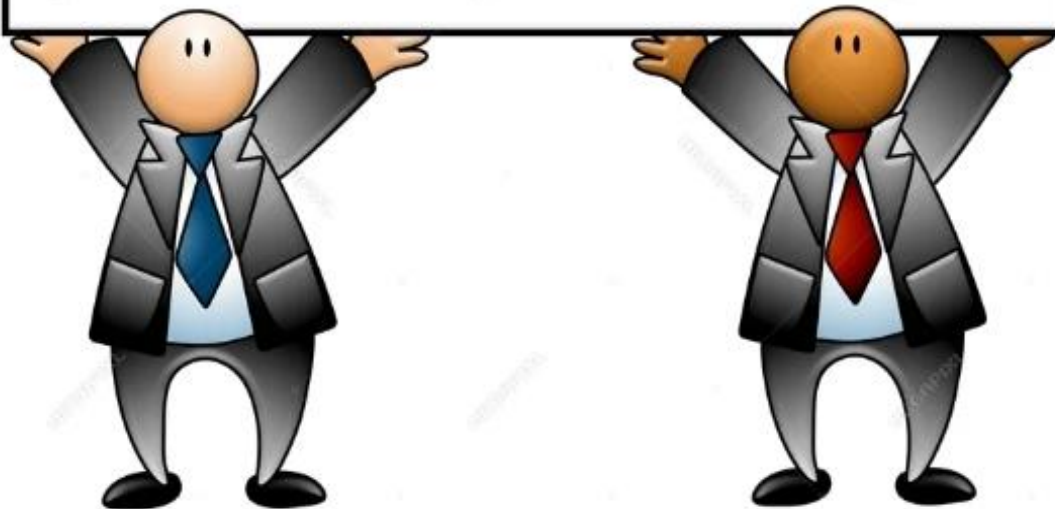
Optimize for Ad hoc Workloads



#1

The stubs are very small (generally < 1 Kb) but they still count for the total number of plans in the cache.

The only advantage is space saving.



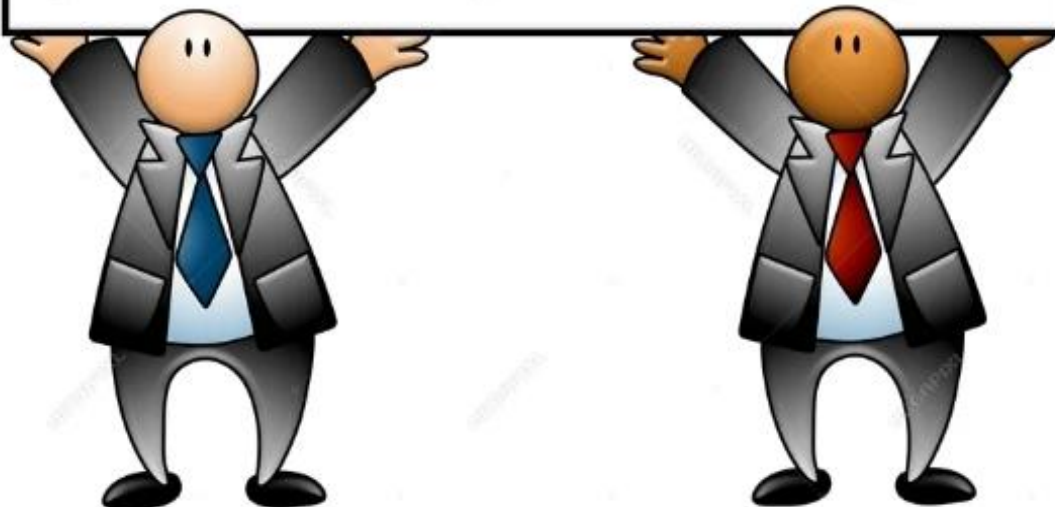
Optimize for Ad hoc Workloads



#2

There aren't plans in the cache to analyse for the stubs.

Do you capture query plans in a different way? (Query Store, monitoring tool, etc.)

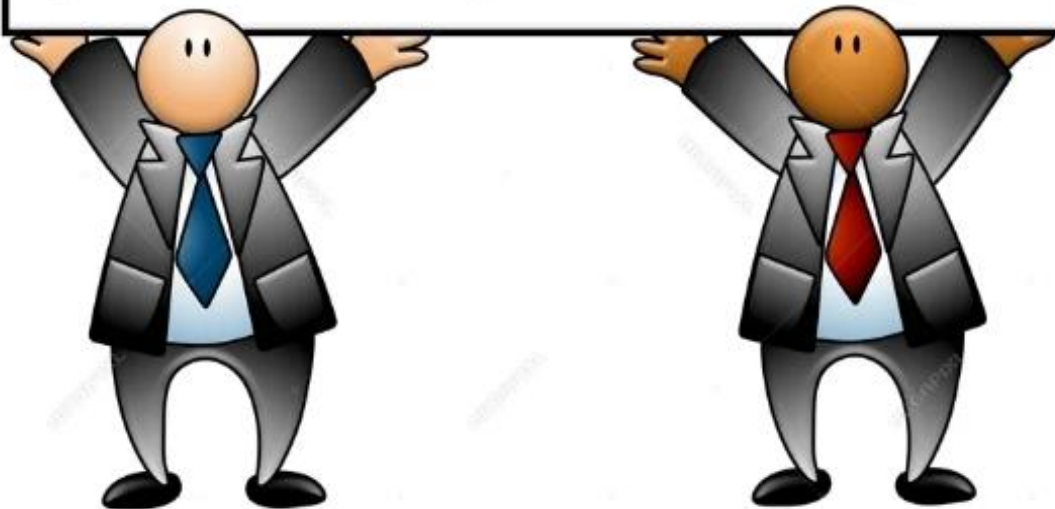


Optimize for Ad hoc Workloads



#3

A second execution of an Ad hoc query needs recompiling, that means *time* and usage of resources.

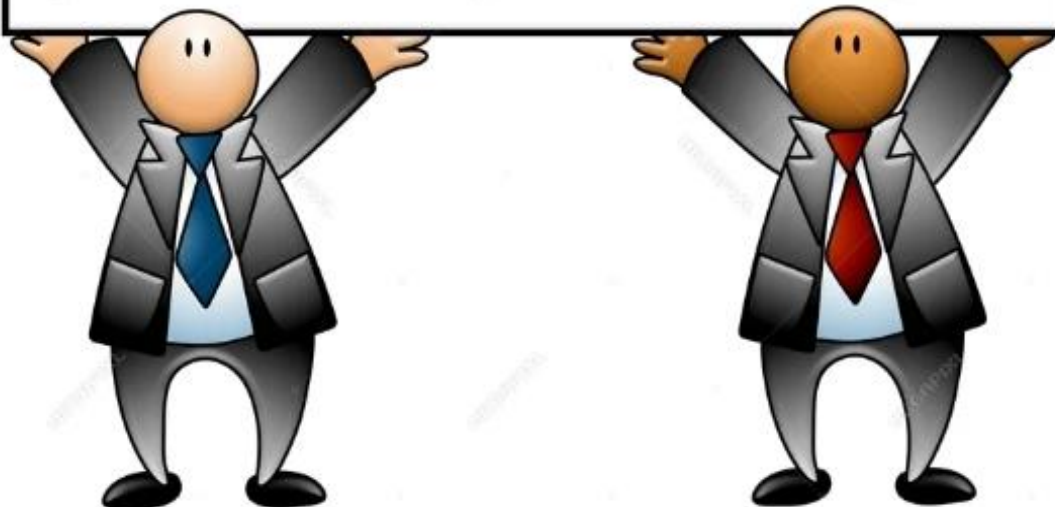


Optimize for Ad hoc Workloads



#4

Is there memory pressure related to space usage?

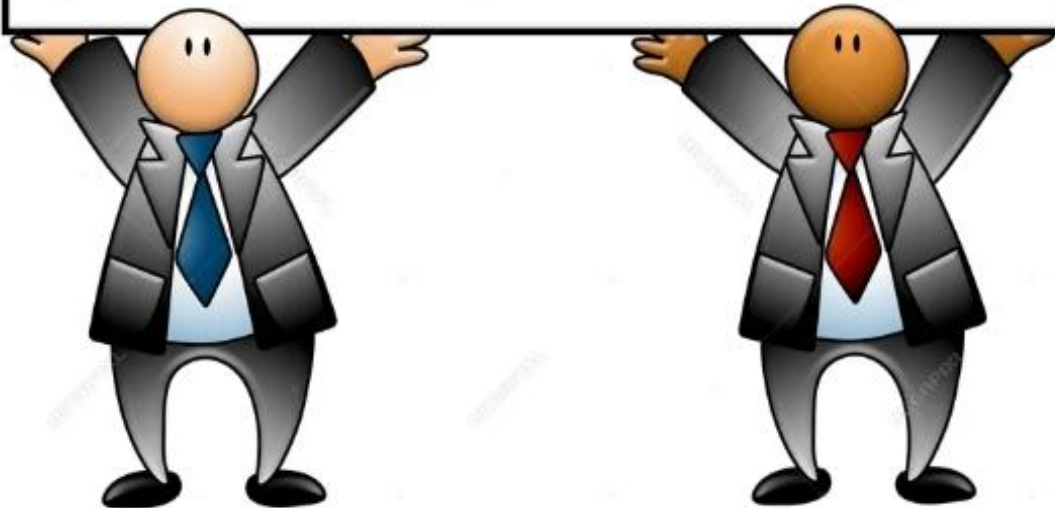


Optimize for Ad hoc Workloads



#5

How many queries are in the cache with `execution_count = 1`? And how many with `execution_count > 1`?





Another option
for Ad hoc
queries

The Trace Flag 253

- It prevents the caching of Adhoc plans
- If enabled *at query level, it prevents simple parameterization*
- It is *undocumented*



Resources

Resources

Official documentation – Query processing architecture guide

<https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16>

Erin Stellato – SQL Server Plan Cache Limits

<https://www.sqlskills.com/blogs/erin/sql-server-plan-cache-limits/>

Guy Glantser - How to Use Parameters Like a Pro and Boost Performance

<https://www.youtube.com/watch?v=PR39RcJV1K8>

Sergio Govoni – SQL Server 2022 Parameter Sensitive Plan Optimization

<https://t.ly/Pk2bM>

Resources

Paul White – Simple Parameterization and Trivial Plans

<https://sqlperformance.com/2022/03/sql-performance/simple-param-trivial-plans-1>

Erik Darling – No, Really: Don't Optimize For Ad Hoc Workloads As A Best Practice

<https://erikdarlingdata.com/no-really-dont-optimize-for-ad-hoc-workloads-as-a-best-practice/>

Randolph West – Don't optimize for ad hoc workloads as a best practice

<https://bornsql.ca/blog/dont-optimize-for-ad-hoc-workloads-as-a-best-practice/>

Jonathan Kehayias – Enlarging the AdventureWorks Sample Databases

<https://www.sqlskills.com/blogs/jonathan/enlarging-the-adventureworks-sample-databases/>



Are there any
questions?



Grazie!!!

Data Saturday #37 Feedback Form

