Why is the Same Query Sometimes Slow?

Slides & demos: BrentOzar.com/go/sniff

Abstract

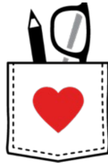
Sometimes the exact same query goes slow out of nowhere. Your current fix is to update statistics, rebuild indexes, or restart the SQL Server. It works, but you don't know why.

The single most common reason is parameter sniffing. SQL Server "sniffs" the first set of parameters for a query, builds an execution plan for it, and then reuses that same plan no matter what parameters get called. It's been the bane of our performance tuning for decades.

In this session, Brent Ozar will explain how it happens, how SQL Server 2022 is trying to fix it, and how you can work around emergencies in the meantime.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.



BRENT OZAR
UNLIMITED

99-05: dev, architect, DBA
05-08: DBA, VM, SAN admin
08-10: MCM, Quest Software
Since: consulting DBA

www.BrentOzar.com
Help@BrentOzar.com

Slides & demos: BrentOzar.com/go/sniff

I'm going to teach you 3 things.

1. What parameter sniffing is
2. How SQL Server 2022 tries to reduce it
3. Learning resources for reducing the stench

Slides, demos, resources: cleverly hidden below

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

Tools I’m using

Microsoft SQL Server 2022
(but all SQL Server versions, Azure SQL DB flavors, Amazon RDS, etc. have the exact same issue)

Stack Overflow db: BrentOzar.com/go/querystack
(any size will work – I’m using a big one)

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

ct Explorer

nect

- dbo.Users
  - Columns
    - Id (PK, int, not null)
    - AboutMe (nvarchar(max), null)
    - Age (int, null)
    - CreationDate (datetime, not null)
    - DisplayName (nvarchar(40), not null)
    - DownVotes (int, not null)
    - EmailHash (nvarchar(40), null)
    - LastAccessDate (datetime, not null)
    - Location (nvarchar(100), null)
    - Reputation (int, not null)
    - UpVotes (int, not null)
    - Views (int, not null)
    - WebsiteUrl (nvarchar(200), null)
    - AccountId (int, null)
  - Keys
  - Constraints
  - Triggers
  - Indexes
  - Statistics
- dbo.Votes

SQLQuery1.sql*

```sql
SELECT TOP 100 * FROM dbo.Users;
```

200 %

Results | Messages

	Id	AboutMe	DisplayName	Location	Reputation
1	-1	<p>Hi, I'm not really a person....	Community	on the server farm	1
2	1	<p><a href="http://www.codi...	Jeff Atwood	El Cerrito, CA	43628
3	2	<p>Developer on the Stack ...	Geoff Dalgas	Corvallis, OR	3431
4	3	<p><a href="http://blog.stac...	Jarrod Dixon	Raleigh, NC, United States	13323
5	4	I am: the co...	Joel Spolsky	New York, NY	28403
6	5	<p>Technical Evangelist at ...	Jon Galloway	San Diego, CA	38706
7	8	<p>This is a puppet test acco...	Eggs McLaren		942
8	9	<p>Independent software en...	Kevin Dente	Oakland, CA	14059
9	10	I'm not takin' my sneakers off!...	Sneakers O'Toole	Lakehead, CA	101
10	11	NULL	Anonymous User	NULL	1837
11	13	Quick links: <li...	Chris Jester-Young	Raleigh, NC, United States	173767
12	16	You don't know me because ...	Rodrigo Sieiro	Uberlândia, Minas Gerais, Brazil	507
13	17	<p>In my spare time when no...	Nick Berardi	Issaquah, WA	43921
14	19	<p>I work mainly with Python ...	Mads Kristiansen	Denmark	1222

Slides & demos: BrentOzar.com/go/sniff

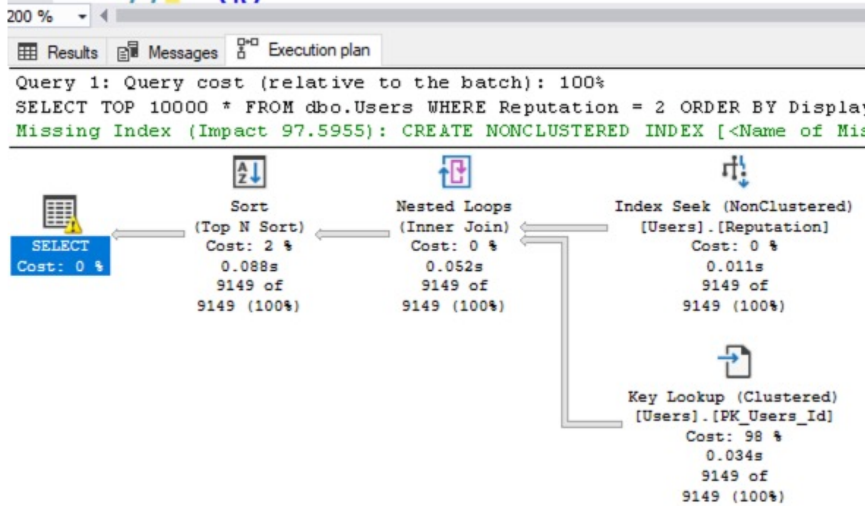
Create an index on Reputation

But only Reputation – not enough to cover this query:

```sql
CREATE INDEX Reputation ON dbo.Users(Reputation)
GO
SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation = 2
ORDER BY DisplayName;
```

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.
BrentOzar.com/go/sniff



SQL Server uses the index.

Measuring the plan

	Reputation 2 (Small Data)
Plan shape	Index seek + key lookup
Logical reads	28,048
Parallel	No
Memory grant	56 MB
Duration	~1 second

Your metrics will be different depending on the size of Stack Overflow db you're using.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

Now try Reputation = 1.

```
59 SELECT TOP 10000 *
60 FROM dbo.Users
61 WHERE Reputation = 1
62 ORDER BY DisplayName;
```

200 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation = 1 ORDER BY DisplayName

SELECT Cost: 0 % ← Top Cost: 0 % 12.432s 10000 of 10000 (100%) ← Parallelism (Gather Streams) Cost: 0 % 12.431s 10000 of 10000 (100%) ← Sort (Top N Sort) Cost: 100 % 12.341s 10047 of 10000 (100%) ← Clustered Index Scan (Clustered) [Users].[PK_Users_Id] Cost: 0 % 2.279s 6044557 of 6044560 (99%)

Slides & demos: BrentOzar.com/go/sniff

Measuring the plans

	Reputation 2 (Small Data)	Reputation 1 (Big Data)
Plan shape	Index seek + key lookup	Table scan
Logical reads	28,048	141,666
Parallel	No	Yes
Memory grant	56 MB	~1,500 MB
Duration	~1 second	~10-15 seconds

Your metrics will be different depending on the size of Stack Overflow db you're using, but overall you'll see the same small/large pattern.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

```sql
65  /* How SQL Server decides plans: */
66  DBCC SHOW_STATISTICS('dbo.Users', 'Reputation');
```

200 %

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	Reputation	Oct 18 2022 9:17AM	8917507	8917507	200	0.0400632	8	NO	NULL	8917507	0

	All density	Average Length	Columns
1	4.918355E-05	4	Reputation
2	1.12139E-07	8	Reputation, Id

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	1	0	6044557	0	1
2	2	0	9149	0	1
3	3	0	191747	0	1
4	4	0	39094	0	1
5	5	0	26163	0	1
6	6	0	322794	0	1
7	7	0	14294	0	1
8	8	0	129703	0	1

Slides & demos: BrentOzar.com/go/sniff

But it doesn’t look every time.

It only looks when there’s no plan cached in memory.

Building plans is computationally intensive, so when you run a query, SQL Server:

- Checks the plan cache to see if a valid plan has already been compiled
- If so, reuses that plan
- If not, builds the plan, puts it in cache, and then starts running your query

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

This saves CPU & time.

SQLQuery3.sql*

```
57
58 SELECT TOP 10000 *
59 FROM dbo.Users
60 WHERE Reputation = 1
61 ORDER BY DisplayName;
62 GO
```

200 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation = 1 ORDER…

SELECT Cost: 0 % ← Top Cost: 0 % 12.583s 10000 of 10000 (100%) ← Parallelism (Gather Streams) Cost: 0 % 12.581s 10000 of 10000 (100%) ← Sort (Top N Sort) Cost: 100 % 12.483s 10048 of 10000 (100%)

Properties

SELECT

Misc	
Cached plan size	40 KB
CardinalityEstimationModelVersion	150
CompileCPU	4
CompileMemory	272
CompileTime	4
DatabaseContextSettingsId	2
Degree of Parallelism	4
Estimated Number of Rows for All Exe	0
Estimated Number of Rows Per Execut	10000
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	34247.5
Memory Grant	2320 MB
MemoryGrantInfo	
Optimization Level	FULL
OptimizerHardwareDependentPropert	
OptimizerStatsUsage	
ParentObjectId	0

2 different queries, 2 plans.

SQL Server builds a hash of the query text.

Different query text = new plans get generated.

Sometimes they're different.

Sometimes they're the same.

```
SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation = 2
ORDER BY DisplayName;
GO

SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation = 1
ORDER BY DisplayName;
GO
```

© Brent Ozar Unlimited®. All rights reserved.
BrentOzar.com/go/sniff

Let’s put it in a stored procedure.

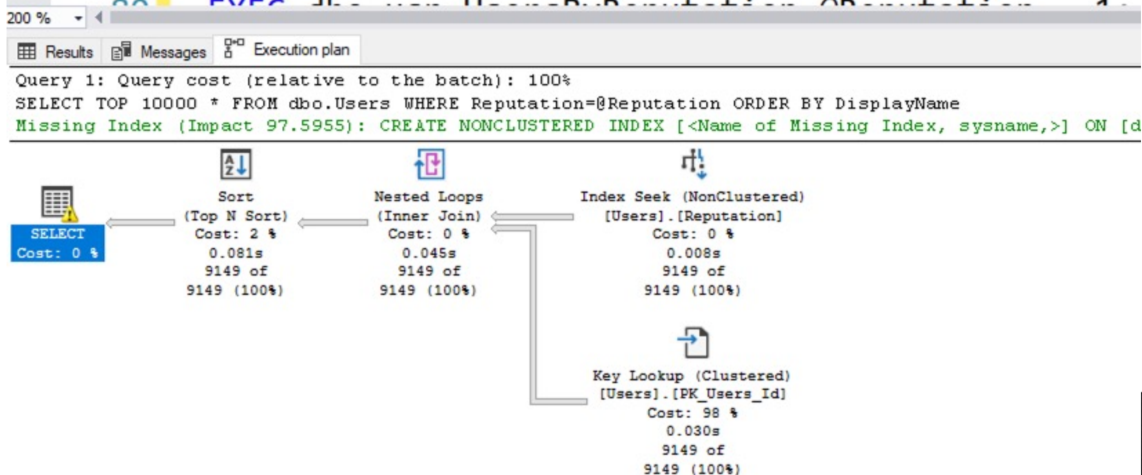
Same query – just parameterized.

```
CREATE OR ALTER PROCEDURE dbo.usp_UsersByReputation
  @Reputation int
AS
SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation=@Reputation
ORDER BY DisplayName;
GO
```

Slides & demos: BrentOzar.com/go/sniff

The first time, it runs for 2...

```
78  EXEC dbo.usp_UsersByReputation @Reputation = 2;
79  GO
```



© Brent Ozar Unlimited®. All rights reserved.

Measuring the plans

	Reputation 2 (Small Data)
Plan shape	Index seek + key lookup
Logical reads	28,048
Parallel	No
Memory grant	56 MB
Duration	~1 second

Your metrics will be different depending on the size of Stack Overflow db you're using, but overall you'll see the same small/large pattern.

Slides & demos: BrentOzar.com/go/sniff

But now run it for 1...

SQLQuery3.sql*

```
78  EXEC dbo.usp_UsersByReputation @Reputation = 2;
79  GO
80  EXEC dbo.usp_UsersByReputation @Reputation = 1;
81  GO
```

200 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputation ORDER BY DisplayName
Missing Index (Impact 97.5955): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [d…
```

SELECT Cost: 0 %

Sort (Top N Sort) Cost: 2 % 37.134s 10000 of 9149 (109%)

Nested Loops (Inner Join) Cost: 0 % 15.488s 6044557 of 9149 (66067%)

Index Seek (NonClustered) [Users].[Reputation] Cost: 0 % 1.256s 6044557 of 9149 (66067%)

Key Lookup (Clustered) [Users].[PK_Users_Id] Cost: 98 % 11.093s 6044557 of 9149 (66067%)

Query executed successfully. | SQL2019\SQL2022 (16.0 RC1) | SQL2019\Administrator ... | StackOverflow | 00:00:37 | 10,000 rows

Ready Ln 81 Col 1 Ch 1 INS

© Brent Ozar Unlimited®. All rights reserved.

What USED to happen...

	Reputation 2 (Small Data)	Reputation 1 (Big Data)
Plan shape	Index seek + key lookup	Table scan
Logical reads	28,048	141,666
Parallel	No	Yes
Memory grant	56 MB	~1,500 MB
Duration	~1 second	~10-15 seconds

Your metrics will be different depending on the size of Stack Overflow db you're using, but overall you'll see the same small/large pattern.

Slides & demos: BrentOzar.com/go/sniff

What's happening now:

	Reputation 2 (Small Data)	Reputation 1 (Big Data)
Plan shape	Index seek + key lookup	**Index seek + key lookup**
Logical reads	28,048	**18,521,946**
Parallel	No	**No (and needs it!)**
Memory grant	56 MB	**56 MB (spills to disk)**
Duration	~1 second	**>30 seconds**

Customers call in complaining that queries are timing out.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

The plan was compiled for 2.

```
78  EXEC dbo.usp_UsersByReputation @Reputation = 2;
79  GO
80  EXEC dbo.usp_UsersByReputation @Reputation = 1;
81  GO
```

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputation ORDER BY DisplayName
Missing Index (Impact 97.5955): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON

SELECT Cost: 0 %
Sort (Top N Sort) Cost: 2 % 37.134s 10000 of 9149 (109%)
Nested Loops (Inner Join) Cost: 0 % 15.488s 6044557 of 9149 (66067%)
Index Seek (NonClustered) [Users].[Reputation] Cost: 0 % 1.256s 6044557 of 9149 (66067%)
Key Lookup (Clustered) [Users].[PK_Users_Id] Cost: 98 % 11.093s 6044557 of 9149 (66067%)

Query executed successfully. | SQL2019\SQL2022 (16.0 RC1) | SQL2019\Administrator ... | StackOverflow | 00:00:37 | 10,000 rows

Properties – SELECT

Property	Value
Memory Grant	2136 KB
MemoryGrantInfo	
MissingIndexes	
Optimization Level	FULL
OptimizerHardwareDependentPropert	
OptimizerStatsUsage	
Parameter List	@Reputation
Column	@Reputation
Parameter Compiled Value	(2)
Parameter Data Type	int
Parameter Runtime Value	(1)
ParentObjectId	1557580587
QueryHash	0xA6F28073109D7278
QueryPlanHash	0x52551CBB40CE8499
QueryTimeStats	
RetrievedFromCache	true
SecurityPolicyApplied	False
Set Options	ANSI_NULLS: True, ANSI_PADDING: True,
Statement	SELECT TOP 10000 * FROM dbo.Users WH
StatementParameterizationType	0
StatementSqlHandle	0x0900A49D3B684BB6381139A538C0236E
WaitStats	

Parameter Compiled Value
Parameter Compiled Value

This is sniffing.

SQL Server sniffs the input parameters used when putting the plan into cache.

As long as that plan is in cache, everyone gets the same plan.

Properties – SELECT

Property	Value
Memory Grant	2136 KB
MemoryGrantInfo	
MissingIndexes	
Optimization Level	FULL
OptimizerHardwareDependentPropert	
OptimizerStatsUsage	
Parameter List	@Reputation
Column	@Reputation
Parameter Compiled Value	(2)
Parameter Data Type	int
Parameter Runtime Value	(1)
ParentObjectId	1557580587
QueryHash	0xA6F28073109D7278

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.
BrentOzar.com/go/sniff

“But you have to fix it!”

Customers are screaming

Small data parameters run quickly

Large data parameters:

- Cause app timeouts at 30+ seconds
- Hammer TempDB due to spills

So you try to “fix” it...

Slides & demos: BrentOzar.com/go/sniff

In order of career seniority

1. Restart the operating system
2. Restart the SQL Server service
3. Fail over the cluster/AG/mirror
4. DBCC FREEPROCCACHE
5. Rebuild indexes
6. Update statistics

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

In order of career seniority

1. Restart the operating sy
2. Restart th

xes

Update statistics

THESE ARE ALL DOING THE SAME BASIC THING: freeing parts of the plan cache.

Slides & demos: BrentOzar.com/go/sniff

What really happens

You do one of those “fixes”

Plans are removed from the plan cache

Someone runs the query that used to be slow

SQL Server checks the plan cache, and since there’s no cached plan for that query, it builds a new one

And sniffs the parameters that used to be slow (thereby building a good plan for it)

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

Simulating it

```
83  ALTER TABLE dbo.Users REBUILD;
84  GO
85  EXEC dbo.usp_UsersByReputation @Reputation = 1;
86  GO
```

200 %

Results | Messages | Execution plan

	Id	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location	Reputation	UpVotes	Views	WebsiteUrl	AccountId
1	2417926	NULL	NULL	2013-05-24 14:33:01.300	[illegible]	0	NULL	2013-05-24 15:18:35.350	NULL	1	0	0	NULL	2811847
2	1955288	NULL	NULL	2013-01-07 14:25:43.087	[illegible]	0	NULL	2013-03-24 03:07:39.383	NULL	1	0	31	NULL	2214217
3	2140934	NULL	NULL	2013-03-06 16:56:20.987	[illegible]	0	NULL	2013-03-06 16:56:20.987	NULL	1	0	6	NULL	2455633
4	2211070	NULL	NULL	2013-03-26 10:08:00.757	[illegible]	0	NULL	2017-11-08 06:42:20.087	NULL	1	0	4	NULL	2546337
5	4880433	NULL	NULL	2015-05-08 20:09:46.237	[illegible]	0	NULL	2015-05-08 20:09:46.237	NULL	1	0	1	NULL	6278521
6	2441368	NULL	NULL	2013-05-31 17:20:51.627	[illegible]	0	NULL	2013-06-27 13:02:42.197	NULL	1	0	1	NULL	2841927
7	2882213	NULL	NULL	2013-10-15 10:58:20.253	[illegible]	0	NULL	2016-11-24 18:29:54.837	NULL	1	0	1	NULL	3440046
8	4208097	NULL	NULL	2014-11-02 18:38:35.490	[illegible]	0	NULL	2014-11-03 00:24:33.157	NULL	1	0	6	NULL	5269276
9	4950782	NULL	NULL	2015-05-28 20:05:02.003	[illegible]	0	NULL	2015-07-13 22:46:03.283	NULL	1	0	0	NULL	6381504
10	2515912	NULL	NULL	2013-06-24 10:46:08.763	[illegible]	0	NULL	2016-05-18 13:50:16.647	NULL	1	0	0	NULL	2937277
11	4567326	NULL	NULL	2015-02-14 19:27:35.423	[illegible]	0	NULL	2015-02-16 10:04:52.517	NULL	1	0	0	NULL	5789171
12	4267203	NULL	NULL	2014-11-18 20:18:46.287	[illegible]	0	NULL	2014-11-18 20:18:46.287	NULL	1	0	0	NULL	5353896
13	4999305	NULL	NULL	2015-06-11 13:10:51.843	[illegible]	0	NULL	2015-06-11 13:10:51.843	NULL	1	0	0	[illegible]	6452943
14	2700752	NULL	NULL	2013-08-20 16:50:09.347	[illegible]	0	NULL	2013-08-20 17:05:55.087	NULL	1	0	0	[illegible]	3197995
15	4804652	NULL	NULL	2015-04-18 13:55:48.320	[illegible]	0	NULL	2015-05-01 12:52:54.100	NULL	1	0	0	[illegible]	[illegible]
16	4281429	<p>htt...	NULL	2014-11-22 10:04:51.877	[illegible]	0	NULL	2014-11-22 10:04:51.877	NULL	1	0	1	[illegible]	[illegible]

Query executed successfully. | SQL2019\SQL2022 (16.0 RC1) | SQL2019\Administrator ... | StackOv... | 00:00:13 | 10,000 rows

Ready | Ln 85 | Col 48 | Ch 48 | INS

Yay!

Yay! Things are better, right?

	Reputation 1 (Big Data)	Reputation 1 (Big Data)
Plan shape	Table scan	Table scan
Logical reads	141,666	141,666
Parallel	Yes	Yes
Memory grant	~1,500 MB	~1,500 MB
Duration	~10-15 seconds	~10-15 seconds

At first, yes.

© Brent Ozar Unlimited®. All rights reserved.

But now run it for reputation 2…

```
84  GO
85  EXEC dbo.usp_UsersByReputation @Reputation = 1;
86  GO
87  EXEC dbo.usp_UsersByReputation @Reputation = 2;
```

	Id	AboutMe	Age	CreationDate	DisplayName	DownVotes	EmailHash	LastAccessDate	Location	Reputation	UpVotes	Views	WebsiteUrl	AccountId
1	5295935	NULL	NULL	2015-09-03 08:54:32.923	Azize	0	NULL	2017-06-27 18:23:13.210	NULL	2	0	13	NULL	6893166
2	7906230	NULL	NULL	2017-04-22 15:37:40.153	J.Dewey	0	NULL	2017-04-26 00:26:09.030	NULL	2	0	3	NULL	10744332
3	7869476	NULL	NULL	2017-04-14 22:55:20.940	Michael	0	NULL	2018-04-03 14:39:29.123	NULL	2	0	0	NULL	10690291
4	9860070	NULL	NULL	2018-05-28 16:51:37.597	0000 00	0	NULL	2018-06-01 15:12:08.593	NULL	2	0	6	NULL	13666228
5	2170291		NULL	2013-03-14 14:40:00.850	01101988	0	NULL	2015-06-17 09:26:34.117	Nagpur, India	2	0	8		2493207
6	8166914	NULL	NULL	2017-06-15 14:54:38.607	0Tommy	0	NULL	2018-06-01 20:18:17.813	Mexico	2	0	3	NULL	11125186
7	6629209	<p>Coding : C++, Java, PHP, MySQL, Visual Basi...	NULL	2016-07-23 14:07:20.287	0x38	0	NULL	2017-08-13 15:04:47.540	Germany	2	0	6		8877715
8	6846980	NULL	NULL	2016-09-19 02:49:28.710	0x41	0	NULL	2016-09-29 22:36:44.057	NULL	2	0	4	NULL	6695362
9	5937483	NULL	NULL	2016-02-16 22:24:19.120	1210saad	0	NULL	2018-04-18 07:31:10.443	NULL	2	0	6	NULL	7855466
10	8332345	NULL	NULL	2017-07-19 14:19:44.910	123	0	NULL	2017-09-12 09:06:54.927	Ankara, Turkey	2	0	8		11365251
11	5242019	NULL	NULL	2015-08-19 06:19:33.543	12345	0	NULL	2015-09-11 20:13:22.707	NULL	2	0	2	NULL	6799752
12	2398966		NULL	2013-05-19 13:39:44.800	1234567	0	NULL	2018-05-30 22:16:51.973	NULL	2	0	3		2788104
13	9268796		NULL	2018-01-25 17:10:09.397	123556666	0	NULL	2018-04-25 14:17:45.387	NULL	2	[illegible]	3		12810379
14	8714925	NULL	NULL	2017-10-03 15:22:41.730	123qwe	0	NULL	2017-10-03 15:53:31.917	NULL	2	[illegible]	[illegible]	[illegible]	11909168
15	6138613	NULL	NULL	2016-03-31 07:37:24.397	1Hund TV	0	NULL	2016-05-26 08:50:24.073	NULL	2	[illegible]	[illegible]	[illegible]	8151890
16	4396550	NULL	NULL	2014-12-26 19:09:22.723	1nzlinger	0	NULL	2015-01-24 11:50:39.617	NULL	2	0	[illegible]	[illegible]	5539367

Query executed successfully. | SQL2019\SQL2022 (16.0 RC1) | SQL2019\Administrator ... | 00:00:00 | 9,149 rows

Yay!

Actually… not so bad, right?

	Reputation 1 (Big Data)	Reputation 2 (Small Data)
Plan shape	Table scan	**Table scan**
Logical reads	141,666	**141,666**
Parallel	Yes	**Yes**
Memory grant	~1,500 MB	**~1,500 MB**
Duration	~10-15 seconds	<1 second

At first, this seems like a winner: all we have to do is make sure the "big" plan goes into memory first, and users are happy.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

Well, hang on…

87 EXEC dbo.usp_UsersByReputation @Reputation = 2;

200 %

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputation ORDER BY DisplayName

SELECT Cost: 0 % ← Top Cost: 0 % 0.702s 9149 of ← Parallelism (Gather Streams) Cost: 0 % 0.700s 9149 of ← Sort (Top N Sort) Cost: 100 % 0.613s 9149 of 10000 (91%) ← Clustered Index Scan (Clustered) [Users].[PK_Users_Id] Cost: 0 % 0.598s 9149 of 6044560 (0%)

SELECT

Cached plan size	40 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	4
Estimated Subtree Cost	34247.5
Memory Grant	1363 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	10000

Statement
SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation=@Reputation
ORDER BY DisplayName

Warnings
The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 1395232 KB, Final 1395232 KB, Used 2216 KB.

Well, hang on…

Warnings
The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 1395232 KB, Final 1395232 KB, Used 2216 KB.

Every time any parameter runs, it gets ~1.5 GB RAM.

- SQL Server erases those pages for this query
- The query runs nearly instantly, turns the RAM in
- Whatever was cached before is gone.

Page Life Expectancy dive-bombs constantly.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

There is no one good plan here.

If we cache the small data plan,
big data parameters time out, spill to disk,
read more pages than there are in the table.

If we cache the big data plan,
query duration is fine, but
we can't cache anything in memory
because we keep granting queries too much RAM.

Slides & demos: BrentOzar.com/go/sniff

How SQL Server tried in the past

SQL Server 2016: Query Store

SQL Server 2017:

- Automatic Plan Regression (Automatic Tuning)
- Adaptive Joins

SQL Server 2019: Adaptive Memory Grant Feedback

None of these solve the problem we're seeing here.

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

2. How SQL Server 2022 tries to reduce it

Slides & demos: BrentOzar.com/go/sniff

In 2019 compatibility level...

When a query comes in:

1. SQL Server checks to see if a query plan has been compiled for it
2. If yes, then run with it
3. If no, sniff the parameters, build a plan for them, and cache that plan for everyone

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

In 2022 compatibility level…

When a query comes in:

1. SQL Server checks to see if a query plan has been compiled for it
2. If yes, then run with it
3. sniff the parameters, build a plan for them, and cache that plan for everyone

This changes

Slides & demos: BrentOzar.com/go/sniff

How sniffing works in 2022

SQL Server looks at equality search parameters

Looks at their statistics histograms

If the histograms have outliers for *any* value
(not just the ones getting sniffed), new logic kicks in:
Parameter Sensitive Plan Optimization (PSPO)

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

The query itself was changed

```
96  ALTER DATABASE CURRENT
97      SET COMPATIBILITY_LEVEL = 160; /* 2022 */
98  GO
99  EXEC dbo.usp_UsersByReputation @Reputation = 2;
```

New

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputati... option (PLAN PER VALUE(ObjectID = 1557580587, QueryVariantID = 2, predicate_range([StackOverflow].[dbo].[Users].[Reputation] = @Reputation
Missing Index (Impact 97.5955): CREATE NONCLUSTERED INDEX [<Name of Missing...x, sysname,>] ON [dbo].[Users] ([Reputation]) INCLUDE ([AboutMe],[Age],[CreationDate],[DisplayName],[DownVotes],[EmailHash],[LastAccessDate],[
```

SELECT Cost: 0 % ← Sort (Top N Sort) Cost: 2 % 0.085s 9149 of 9149 (100%) ← Nested Loops (Inner Join) Cost: 0 % 0.048s 9149 of 9149 (100%) ← Index Seek (NonClustered) [Users].[Reputation] Cost: 0 % 0.008s 9149 of 9149 (100%); Key Lookup (Clustered) [Users].[PK_Users_Id] Cost: 98 % 0.032s 9149 of 9149 (100%)

(Don’t worry, I’ll copy that to a new window)

Slides & demos: BrentOzar.com/go/sniff

```
SELECT TOP 10000 *
FROM dbo.Users
WHERE Reputation=@Reputation
ORDER BY DisplayName
option (PLAN PER VALUE
(ObjectID = 1557580587,
QueryVariantID = 2,
predicate_range(
    [StackOverflow].[dbo].[Users].[Reputation]
    = @Reputation, 100.0, 1000000.0)))
```

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

```
option (PLAN PER VALUE
(ObjectID = 1557580587,
QueryVariantID = 2,
predicate_range(
    [StackOverflow].[dbo].[Users].[Reputation]
    = @Reputation, 100.0, 1000000.0)))
```

The equality predicate on Reputation has outliers.

It will build separate plans for Reputations that have:

- < 100.0 estimated rows
- 100.0 – 1,000,000.0 estimated rows
- > 1,000,000.00 estimated rows

Slides & demos: BrentOzar.com/go/sniff

This is a neat idea!

The plans aren't hard-coded to specific Reputations.

Only the plan needed now actually gets compiled.

The rest are postponed for later compilation:
they'll get sniffed for whatever value gets used at the time we need that plan.

Now, both Reputation 2 and 1 get different plans!

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

```
 99  EXEC dbo.usp_UsersByReputation @Reputation = 2;
100  GO
101  EXEC dbo.usp_UsersByReputation @Reputation = 1;
```

200 % | Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 0%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputation ORDER BY DisplayName option (PLAN PER VALUE(ObjectID = 1557580587, QueryVariantID = 2, pr
Missing Index (Impact 97.5955): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Users] ([Reputation]) INCLUDE ([AboutMe],[Age],
```

SELECT Cost: 0 % | Sort (Top N Sort) Cost: 2 % 0.080s 9149 of 9149 (100%) | Nested Loops (Inner Join) Cost: 0 % 0.041s 9149 of 9149 (100%) | Index Seek (NonClustered) [Users].[Reputation] Cost: 0 % 0.007s 9149 of 9149 (100%)

Key Lookup (Clustered) [Users].[PK_Users_Id] Cost: 98 % 0.028s 9149 of 9149 (100%)

```
Query 2: Query cost (relative to the batch): 100%
SELECT TOP 10000 * FROM dbo.Users WHERE Reputation=@Reputation ORDER BY DisplayName option (PLAN PER VALUE(ObjectID = 1557580587, QueryVariantID = 3, pr
```

SELECT Cost: 0 % | Top Cost: 0 % 15.774s 10000 of 10000 (100%) | Parallelism (Gather Streams) Cost: 0 % 15.772s 10000 of 10000 (100%) | Sort (Top N Sort) Cost: 100 % 15.670s 10048 of 10000 (100%) | Clustered Index Scan (Clustered) [Users].[PK_Users_Id] Cost: 0 % 3.132s 6044557 of 6044560 (99%)

The good news

The implementation logic is conservative:

- Only shows up in compat level >160
- Only shows up for equality searches (and a limited number of them at that)
- Only caches a limited number of plans (less worry about plan cache bloat)
- Where it shows up in plans, it will help!

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.
BrentOzar.com/go/sniff

The less-good, but not bad news

It only fires for equality searches,
not range searches like StartDate/EndDate

It only fires for direct comparisons,
not joined filters (PostTypeId)

Each of the small/medium/large plans is still vulnerable to parameter sniffing

Slides & demos: BrentOzar.com/go/sniff

The truly awful news

The PSPO implementation breaks query monitoring.

Monitoring tools can’t tell where queries are from.

All PSPO queries act like dynamic SQL:
they have no parent/child relationship to the code.

```
Query 1: Query cost (relative to the batch): 100%
CREATE PROCEDURE dbo.usp_UsersByReputation @Reputation int AS

T-SQL

MULTIPLE PLAN
Cost: 0 %
```

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.

3. Where to go to learn more

Slides & demos: BrentOzar.com/go/sniff

Resources:

Slides, demos: https://BrentOzar.com/go/sniff

More details on SQL Server 2022's PSPO:
https://BrentOzar.com/go/pspo

Erland Sommarskog's Slow in the App, Fast in SSMS:
https://www.sommarskog.se/query-plan-mysteries.html

Slides & demos: BrentOzar.com/go/sniff

© Brent Ozar Unlimited®. All rights reserved.