



WHAT'S NEW IN SQL SERVER 2019 FOR DEVELOPERS

MILOŠ RADIVOJEVIĆ, DATA PLATFORM MVP, BWIN, VIENNA

BRATISLAVA, 19.11. 2019

Milos Radivojevic

- **Data Platform MVP**
- Principal Database Consultant, **bwin GVC Vienna, Austria**
- Co-Founder: **SQL Pass Austria**
- Conference Speaker, Book Author

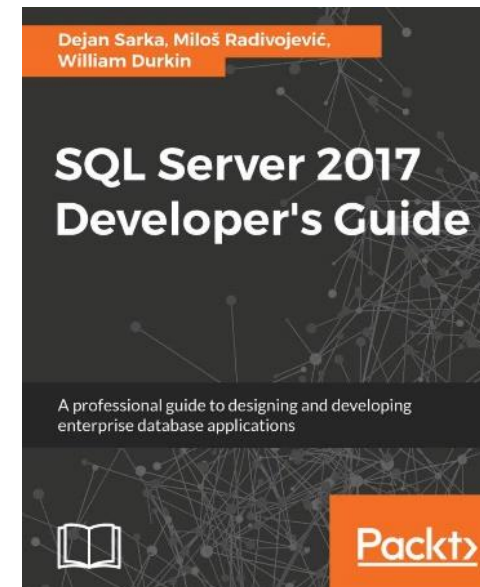
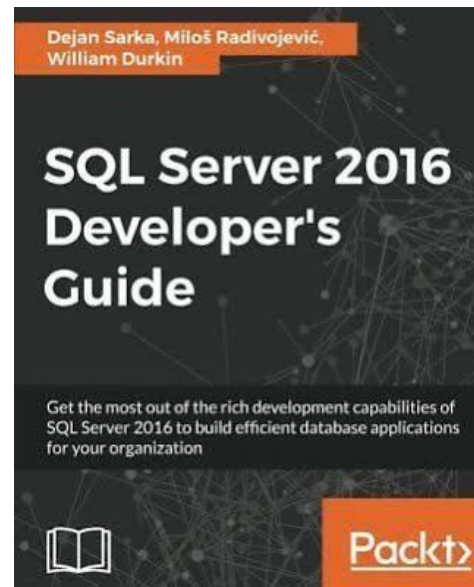


Contact:

MRadivojevic@gvcgroup.com

www.bwinparty.com

Twitter: **@MilosSQL**



Agenda

- Small Database Engine and Programming Enhancements
- Memory-Optimized TempDB Metadata
- Accelerated Database Recovery
- Optimize for Sequential Key
- UTF-8 Support
- Lightweight Statistics
- Intelligent Query Processing
- SQL Graph

SQL Server 2019

- Generally available since 4th November
- Download: <https://go.microsoft.com/fwlink/?linkid=866662>

Developer

SQL Server 2019 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

[Download now](#) ↓

Verbose Truncation Warnings

- Old error message

Msg 8152, Level 16, State 30, Line 13

String or binary data would be truncated.

- New error message

Msg 2628, Level 16, State 1, Line 13

String or binary data would be truncated in table 'TestDb.dbo.T', column 'col2'. Truncated value: 'B5678'.

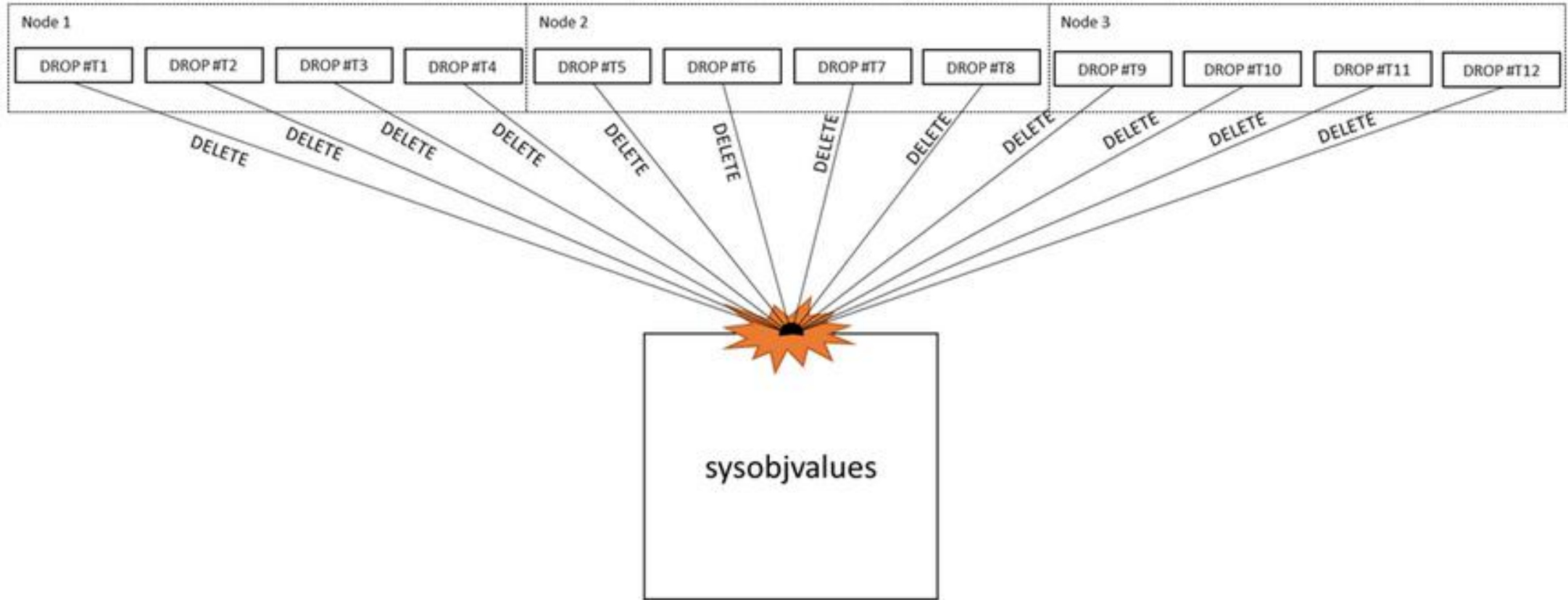
- New error number: 2628 (vs. 8152)
- If you refer to the old error number, you have to turn off the feature

```
ALTER DATABASE SCOPED CONFIGURATION SET VERBOSE_TRUNCATION_WARNINGS = OFF;
```

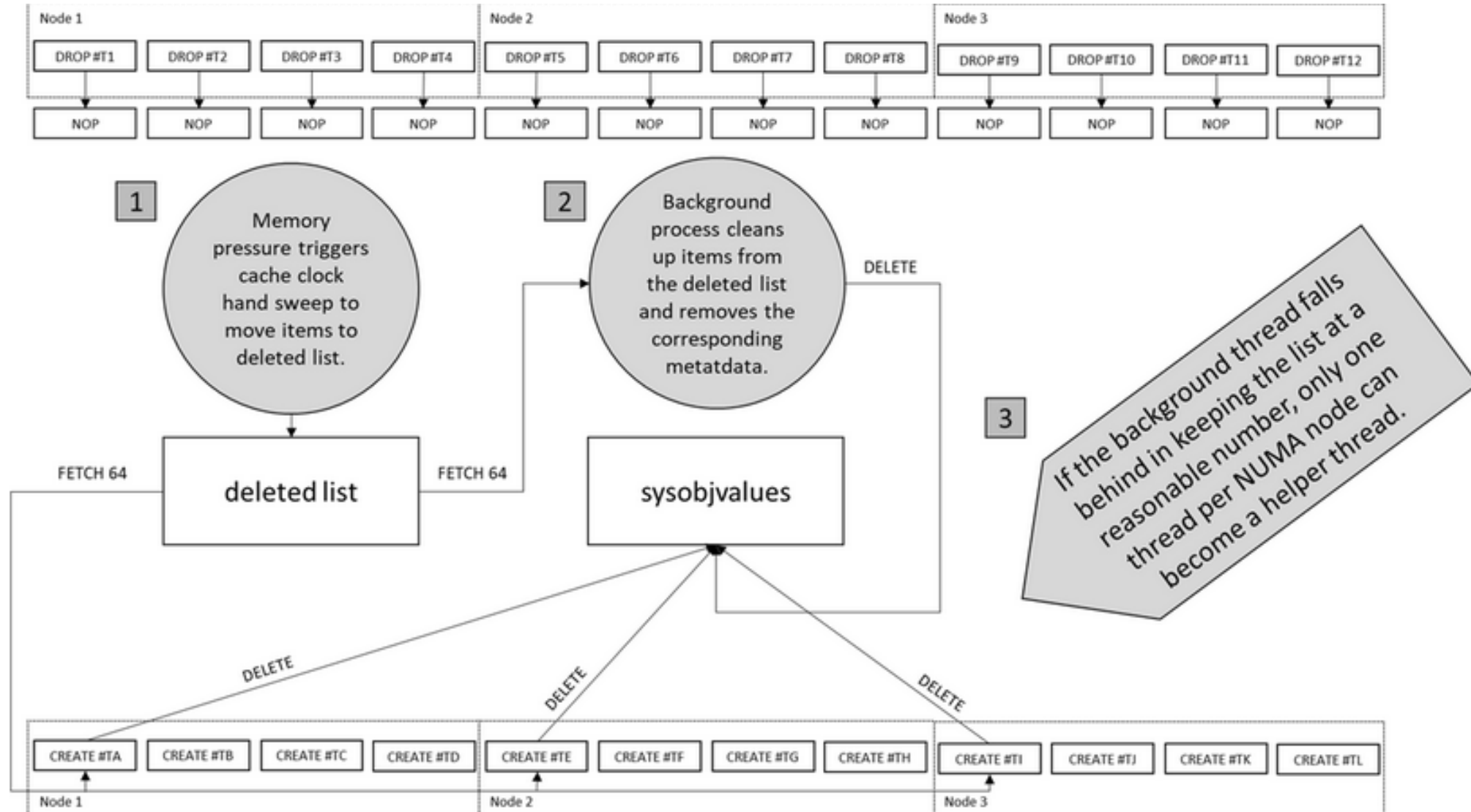
Memory-Optimized TempDB Metadata

- Issues with TempDB-heavy workloads
- **Allocation Page Latch Waits**
 - Multiple users needing to allocate pages for temp tables
 - **Workaround: multiple tempdb files during setup**
- **System Table Page Latch Waits**
 - High rates of create/drop require system table modifications

TempDb Metadata Contention



TempDb Removal Object Optimization



TempDb Removal Object Optimization

- Process changed from synchronous to asynchronous.
 - table is not immediately dropped, it is moved to a deleted list and dropped later by another thread
- Number of helper threads reduced to one per NUMA node
- A helper thread will remove 64 objects at a time rather than 1
- Optimized the latching strategy used when scanning for metadata
- **But still PAGE LATCH contention!!!**

Temporary Tables - Best Practice (regarding latch contention)

- Do not explicitly drop temp tables at the end of a stored procedure
 - they will get cleaned up when the session that created them ends
- Do not alter temp tables after they have been created
- Do not truncate temp tables
- Move index creation statements on temp tables to the new inline index creation syntax

Temporary Tables - Best Practice (regarding latch contention)



```
CREATE TABLE #T1(  
  c1 INT NOT NULL,  
  c2 INT NOT NULL,  
  c3 VARCHAR(10)  
);  
CREATE UNIQUE CLUSTERED INDEX ixT1 ON #T1(c1,c2);
```



```
CREATE TABLE #T1(  
  c1 INT NOT NULL,  
  c2 INT NOT NULL,  
  c3 VARCHAR(10)  
INDEX ixT1 UNIQUE CLUSTERED (c1,c2)  
);
```

BUT... Page Latch contention is still possible

- Tempdb Metadata Contention

Results								
wait_time	wait_type	total_elapsed_time	(No column name)	cpu_time	text	client_net_address	num_reads	num_writes
90	PAGELATCH_EX	17906	tempdb	365	----- - W...	10.33.20.31	17	970
87	PAGELATCH_EX	28654	tempdb	586	----- - W...	10.33.20.31	15	845
81	PAGELATCH_EX	22654	test	447	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	11	463
78	PAGELATCH_EX	18051	test	384	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	15	797
77	PAGELATCH_EX	31569	test	607	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	9	321
77	PAGELATCH_EX	26764	test	544	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	9	310
76	PAGELATCH_EX	11238	tempdb	275	----- - W...	10.33.20.31	15	773
73	PAGELATCH_EX	30884	tempdb	660	----- - W...	10.33.20.31	13	683
71	PAGELATCH_EX	15251	tempdb	324	----- - W...	10.33.20.31	15	786
70	PAGELATCH_EX	34876	tempdb	722	----- - W...	10.33.20.31	17	1051
68	PAGELATCH_EX	27200	test	558	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	14	838
68	PAGELATCH_EX	29202	tempdb	599	----- - W...	10.33.20.31	15	844
67	PAGELATCH_EX	4843	tempdb	101	----- - W...	10.33.20.31	17	905
62	PAGELATCH_EX	10253	tempdb	225	----- - W...	10.33.20.31	22	1459
55	PAGELATCH_EX	21440	tempdb	438	----- - W...	10.33.20.31	19	1160
54	PAGELATCH_EX	29363	tempdb	598	----- - W...	10.33.20.31	15	847
53	PAGELATCH_EX	25905	test	557	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	17	1012
52	PAGELATCH_EX	31193	test	631	CREATE PROCEDURE dbo.P AS BEGIN DECLARE @...	10.33.20.31	17	1026

Memory-Optimized TempDB Metadata



- Key tempdb system tables become SCHEMA_ONLY memory optimized tables
- Latch and lock free
- This is just **metadata** NOT user data!

- Server feature:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON;
```

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = OFF;
```

- Requires server restart

TempDB Enhancements in SQL Server 2019

Default

Opt-in

Temp table
cache
improvements

Concurrent PFS
updates

Memory-
optimized
metadata tables



Results:

.\ostress -E -imem.sql -SAT03W00308\SQL2019RC1 -n100 -r100 -q

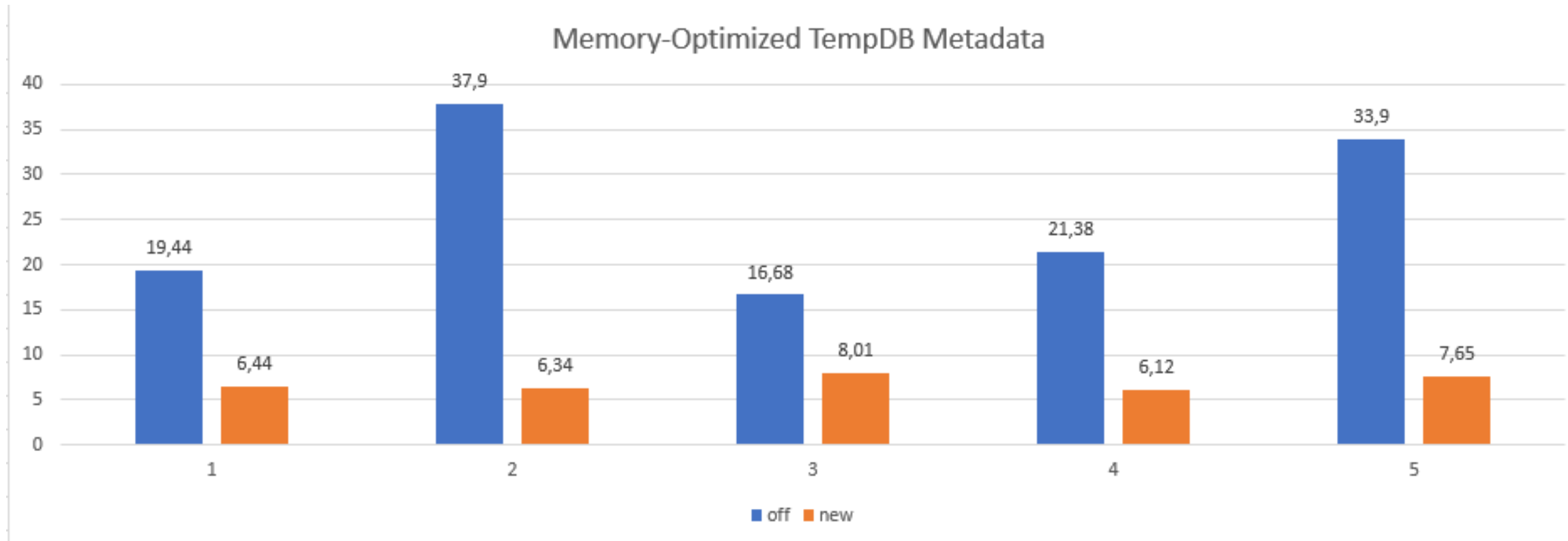
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = OFF;

```
11/18/19 23:10:53.316 [0x00020A38] Starting query execution...
11/18/19 23:10:53.326 [0x00020A38] RsFx I/O completion thread starting
11/18/19 23:10:53.327 [0x00020A38] File Stream support enabled.
11/18/19 23:10:53.329 [0x00020A38] BETA: Custom CLR Expression support enabled.
11/18/19 23:10:53.329 [0x00020A38] Creating 100 thread(s) to process queries
11/18/19 23:10:53.389 [0x00020A38] Worker threads created, beginning execution...
11/18/19 23:13:40.730 [0x00020A38] Total IO waits: 0, Total IO wait time: 0 (ms)
11/18/19 23:13:40.730 [0x00020A38] OSTRESS exiting normally, elapsed time: 00:02:47.697
11/18/19 23:13:40.731 [0x00020A38] RsFx I/O completion thread ended.
```

ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON;

```
11/18/19 23:14:24.744 [0x00002080] Starting query execution...
11/18/19 23:14:24.765 [0x00002080] RsFx I/O completion thread starting
11/18/19 23:14:24.766 [0x00002080] File Stream support enabled.
11/18/19 23:14:24.770 [0x00002080] BETA: Custom CLR Expression support enabled.
11/18/19 23:14:24.771 [0x00002080] Creating 100 thread(s) to process queries
11/18/19 23:14:24.826 [0x00002080] Worker threads created, beginning execution...
11/18/19 23:14:57.390 [0x00002080] Total IO waits: 0, Total IO wait time: 0 (ms)
11/18/19 23:14:57.390 [0x00002080] OSTRESS exiting normally, elapsed time: 00:00:32.898
11/18/19 23:14:57.391 [0x00002080] RsFx I/O completion thread ended.
```

Memory-Optimized TempDB Metadata



Memory-Optimized TempDB Metadata

Temporary Tables

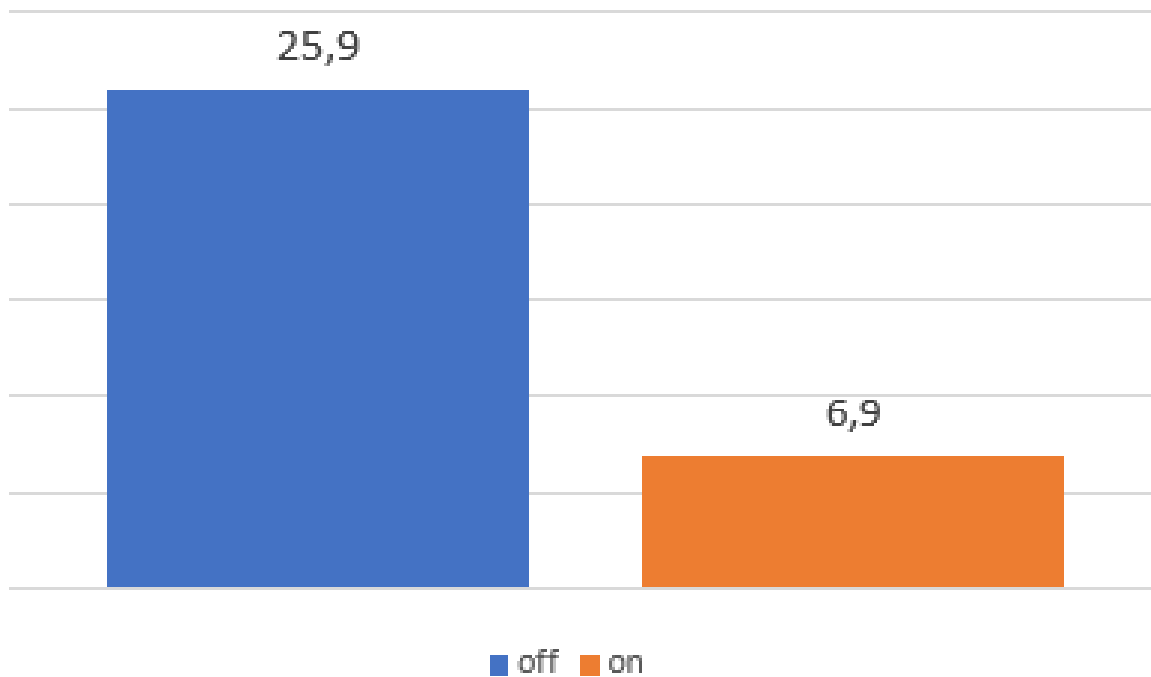
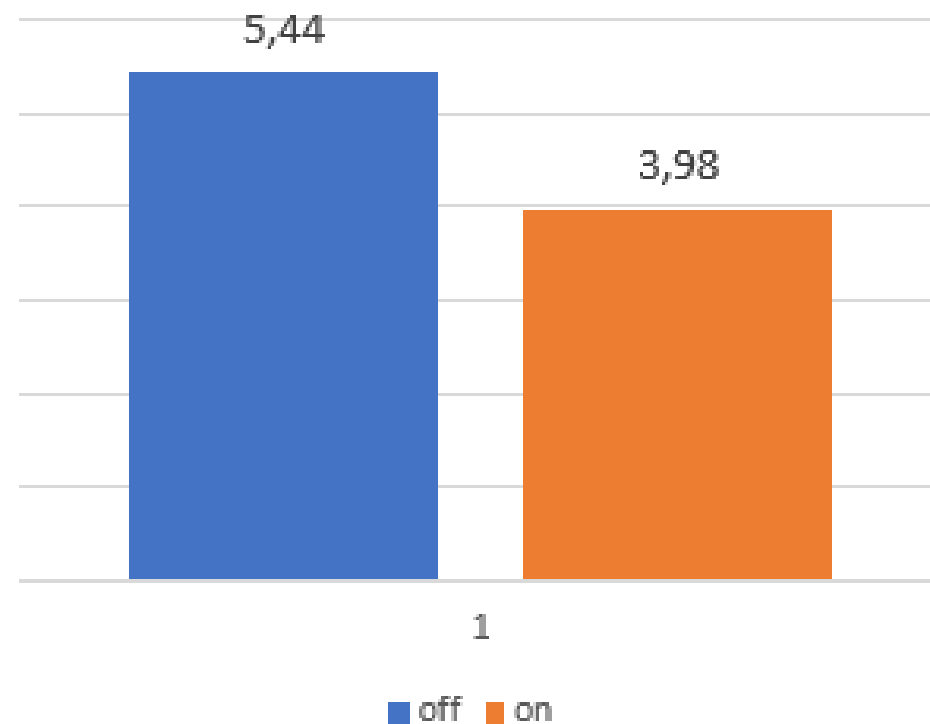


Table Variables



Limitations

- **Cannot create columnstore indexes on temporary tables** when Memory-Optimized TempDB Metadata is enabled
- Transactions that involve a Hekaton table will not be able to access TempDB system views not allowed
- *sp_estimate_data_compression_savings* does not work

This won't work

```
BEGIN TRAN
    SELECT * FROM tempdb.sys.tables;
    INSERT INTO WideWorldImporters.dbo.M1(c1) VALUES(5);
COMMIT TRAN
```

Msg 41317, Level 16, State 0, Line 27

A user transaction that accesses memory optimized tables or natively compiled modules cannot access more than one user database or databases model and msdb, and it cannot write to master.

This will work (more common case)

```
BEGIN TRAN
    CREATE TABLE #t0(id INT);

    INSERT INTO #t0(id) VALUES(4);

    INSERT INTO WideWorldImporters.dbo.M1(c1)
    SELECT id FROM #t0;
COMMIT TRAN
```

(1 row affected)

(1 row affected)

Completion time: 2019-11-04T12:47:26.7622698+01:00

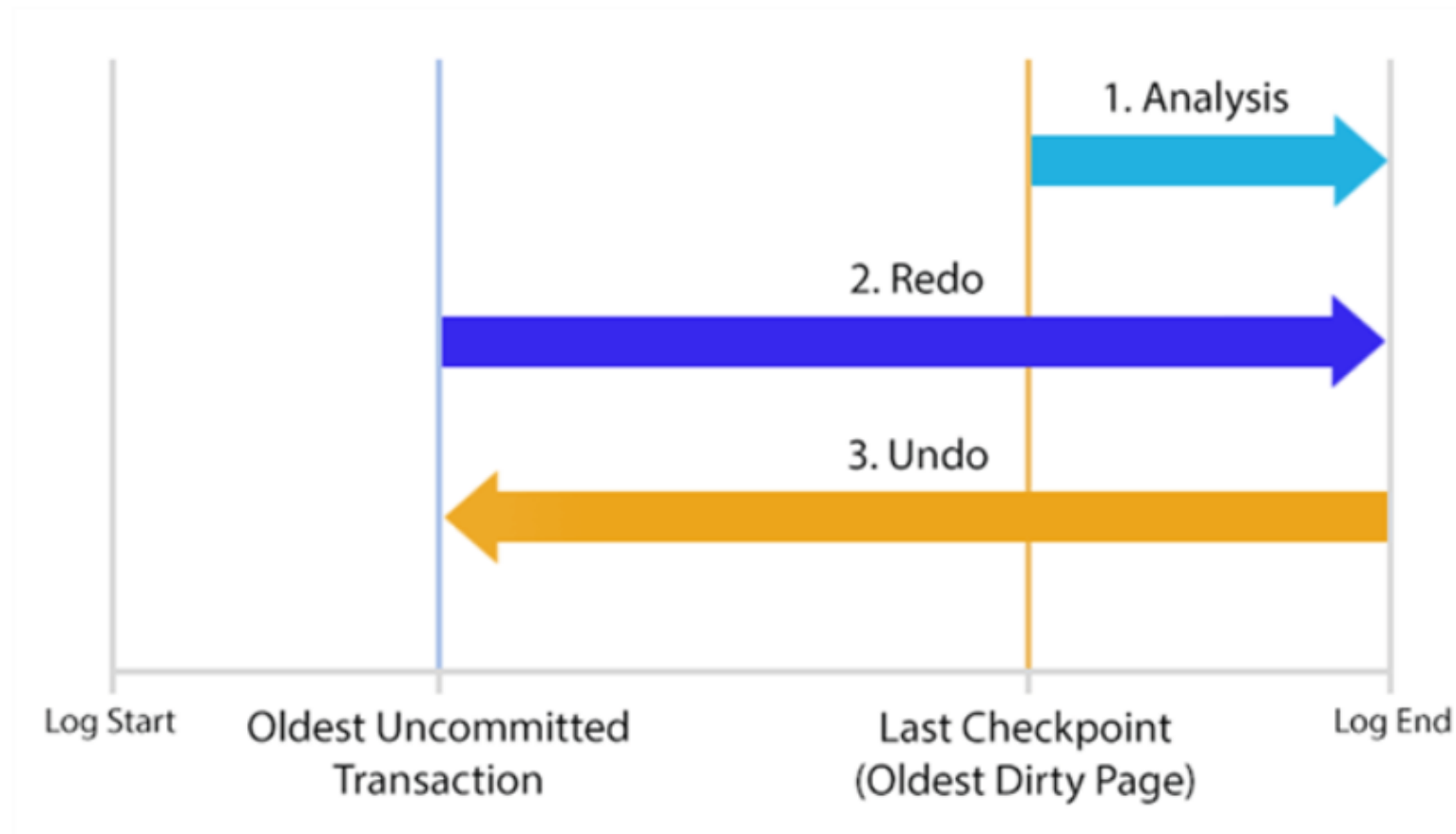
Accelerated Database Recovery (ADR)

- Current Database Recovery Process
 - Recovery time is proportional to the size of the longest active transaction
 - Transaction log cannot be truncated
- Accelerated Database Recovery
 - Significantly improves database availability especially in the presence of long running transactions
 - Fast and consistent database recovery
 - Instantaneous transaction rollback
 - Aggressive log truncation
- More info:

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-accelerated-database-recovery>

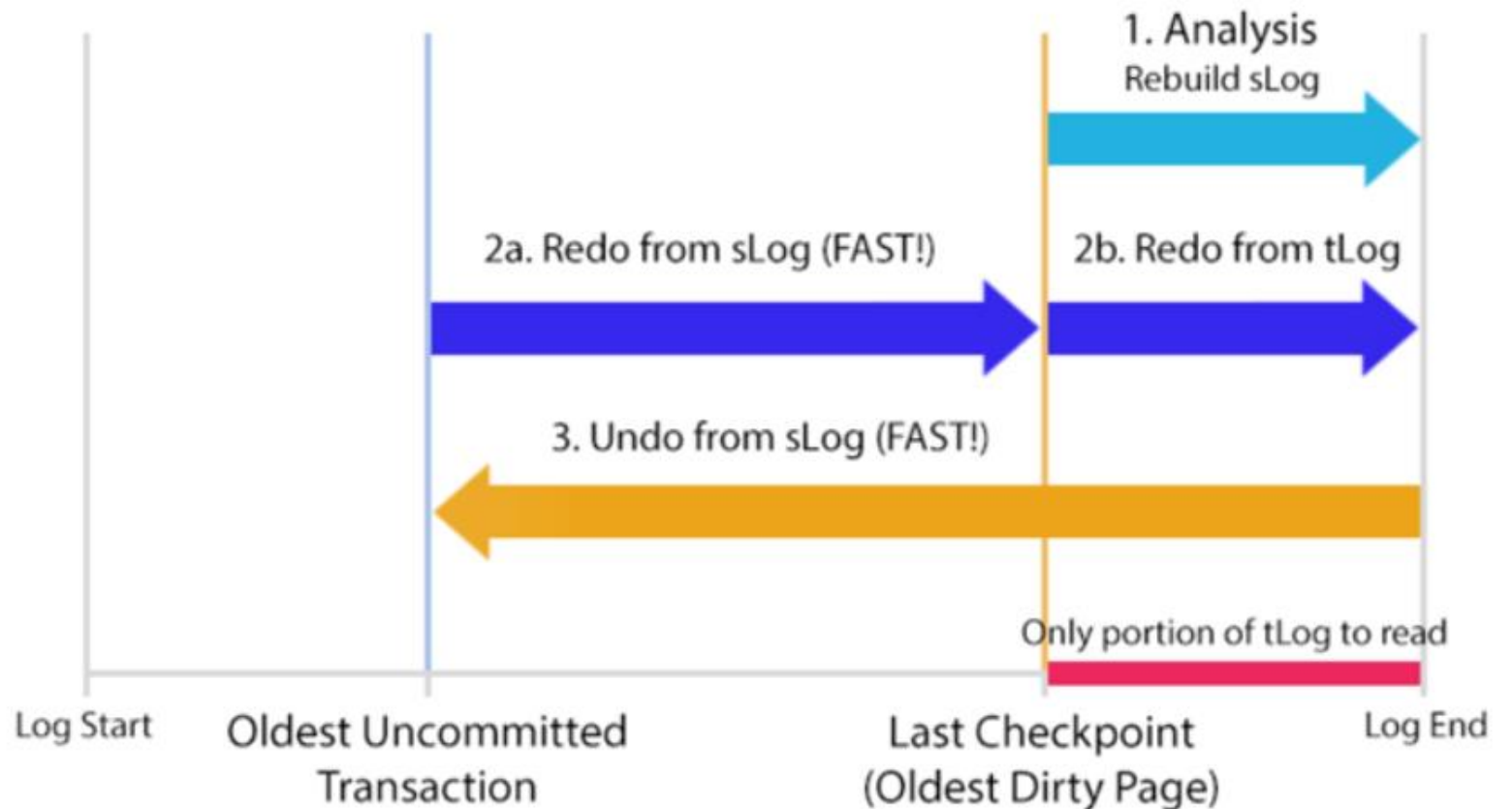
Current Database Recovery Process

- Current Database Recovery Process



Accelerated Database Recovery

- **persisted version store** in the user database
- Logical revert



Accelerated Database Recovery (ADR)

- Components:
 - **Persisted Version Store (PVS)** - Contains previous versions of data modified by transactions. In the case of an update, the previous version of each row is written to the PVS.
 - **Logical revert** - During rollback of a transaction, active transactions read records from the PVS.
 - **sLog** - In-memory log stream that contains activity that is not written to the PVS.
 - **Cleaner** - Cleans up unneeded row versions from the PVS.

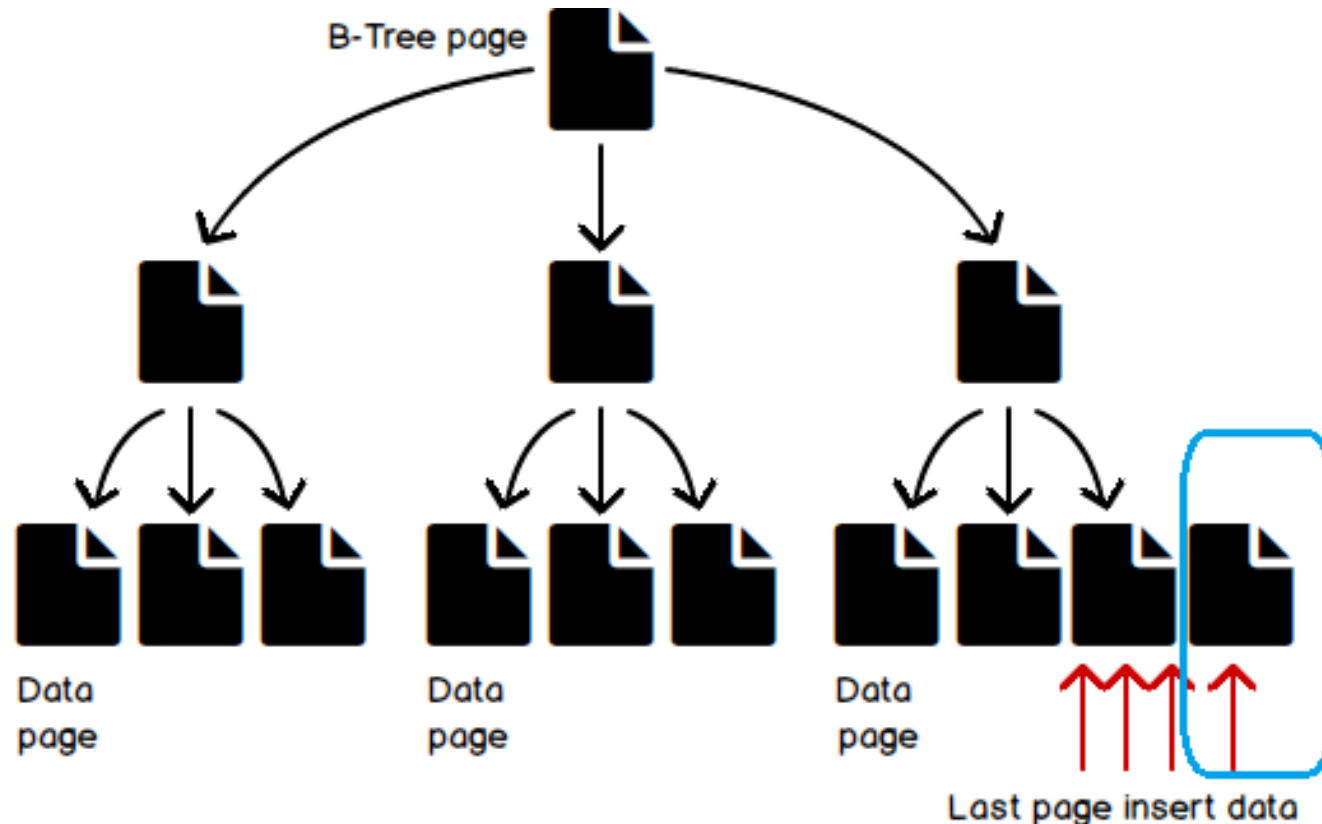
OPTIMIZE_FOR_SEQUENTIAL_KEY

- New option to deal with the last-page insert contention
- OPTIMIZE_FOR_SEQUENTIAL_KEY improves throughput for high-concurrency inserts into the index

```
CREATE TABLE dbo.T(  
  
    id BIGINT IDENTITY(1,1) NOT NULL,  
  
    c1 NVARCHAR(50) NOT NULL,  
  
    CONSTRAINT PK_T PRIMARY KEY CLUSTERED (id)  
  
    WITH (OPTIMIZE_FOR_SEQUENTIAL_KEY = ON)  
)
```

Last-Page Insert Contention

- Increasing leading index column (identity) => all new rows are written into the same page(s)
- PAGELATCH_EX waits



OPTIMIZE_FOR_SEQUENTIAL_KEY

- number of insert threads increases => queue for the page latch increases => throughput will decrease
- if something slows down one of the threads that is holding the latch, this can trigger a convoy and throughput will be almost stopped
- Critical situation => when a new page should be added to the index – a „convoy“ problem
- **OPTIMIZE_FOR_SEQUENTIAL_KEY**
 - controls the rate at which new threads are allowed to request the latch
 - favors threads that are likely to keep the throughput high
- New wait type BTREE_INSERT_FLOW_CONTROL

Testing

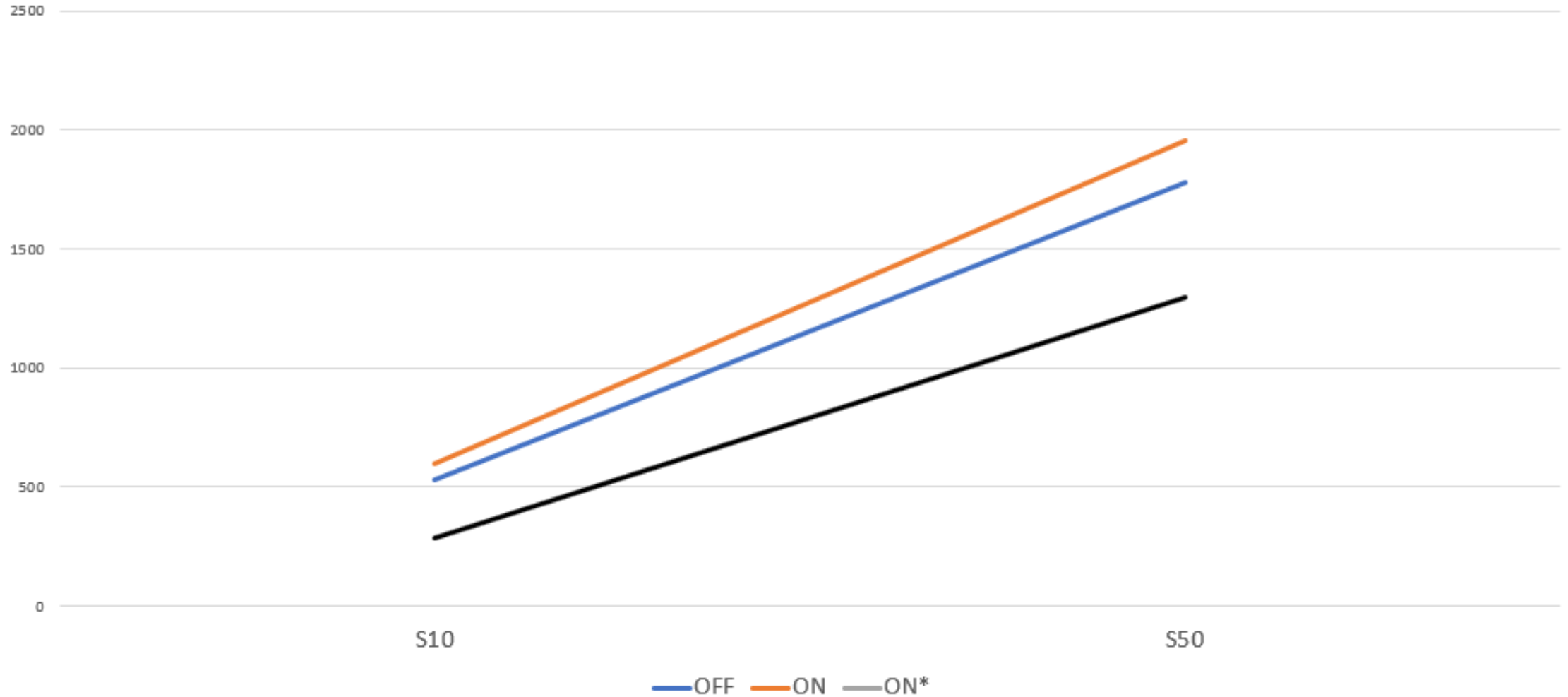
```
CREATE TABLE [dbo].[T0](  
    [id] [bigint] IDENTITY(1,1) NOT NULL,  
    [c1] [smallint] NOT NULL,  
    [c2] [bigint] NOT NULL,  
    [c3] [bigint] NULL,  
    [c4] [nvarchar](128) NOT NULL,  
    [c6] [int] NOT NULL,  
    [c7] [int] NOT NULL,  
    [c8] [nvarchar](256) NULL,  
    [c10] [nvarchar](50) NOT NULL,  
    [c11] [varchar](128) NOT NULL,  
    CONSTRAINT [PK_T0] PRIMARY KEY CLUSTERED  
    (  
        [id] ASC  
    )  
    WITH (OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)  
)  
GO
```

```
CREATE TABLE [dbo].[T0_Opt](  
    [id] [bigint] IDENTITY(1,1) NOT NULL,  
    [c1] [smallint] NOT NULL,  
    [c2] [bigint] NOT NULL,  
    [c3] [bigint] NULL,  
    [c4] [nvarchar](128) NOT NULL,  
    [c6] [int] NOT NULL,  
    [c7] [int] NOT NULL,  
    [c8] [nvarchar](256) NULL,  
    [c10] [nvarchar](50) NOT NULL,  
    [c11] [varchar](128) NOT NULL,  
    CONSTRAINT [PK_T0_Opt] PRIMARY KEY CLUSTERED  
    (  
        [id] ASC  
    )  
    WITH (OPTIMIZE_FOR_SEQUENTIAL_KEY = ON)  
)  
GO
```

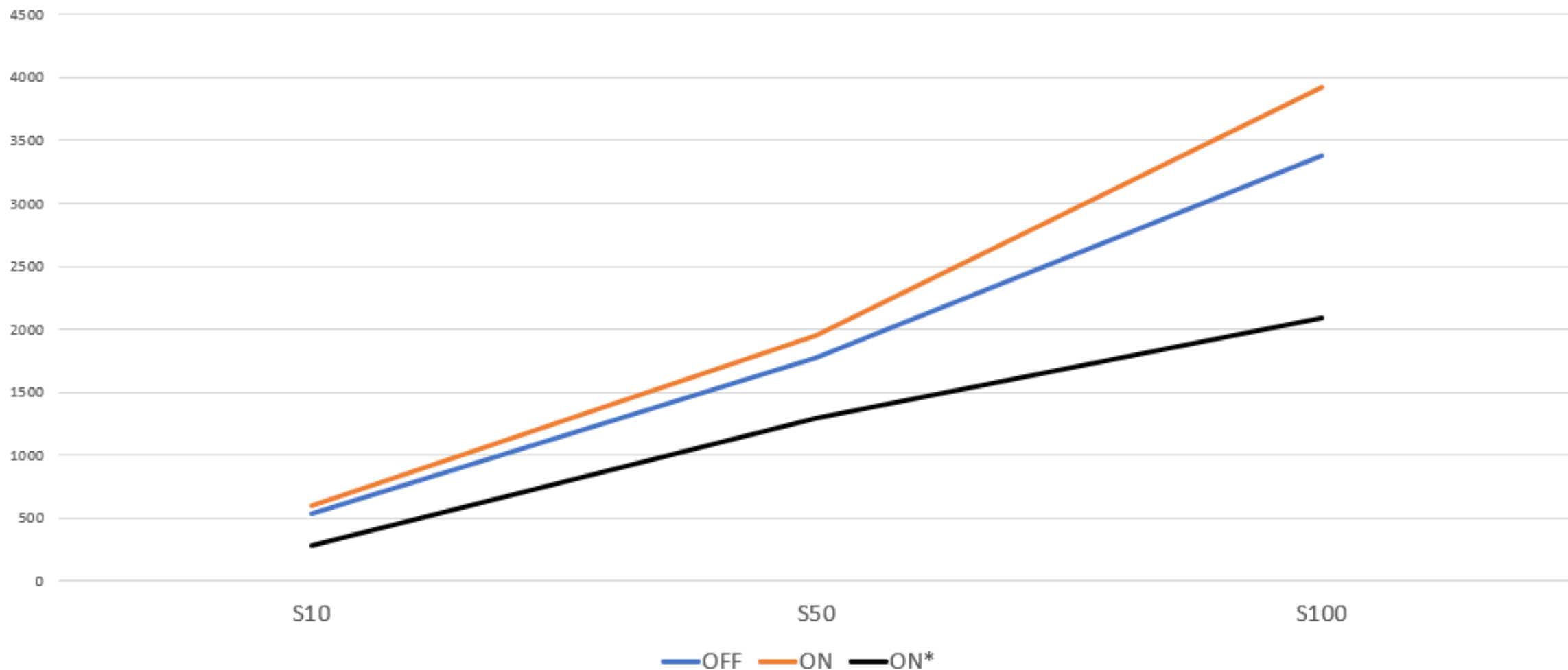
OPTIMIZE_FOR_SEQUENTIAL_KEY - Results

# of parallel sessions	normal table	optimized for seq key	improvement in %
10	531	598	-12,62
50	1 776	1 956	-10,14
100	3 380	3 922	-16,04
200	7 449	6 441	13,53
300	12 292	8 798	28,42
400	17 683	14 901	15,73
500	30 782	25 758	16,32

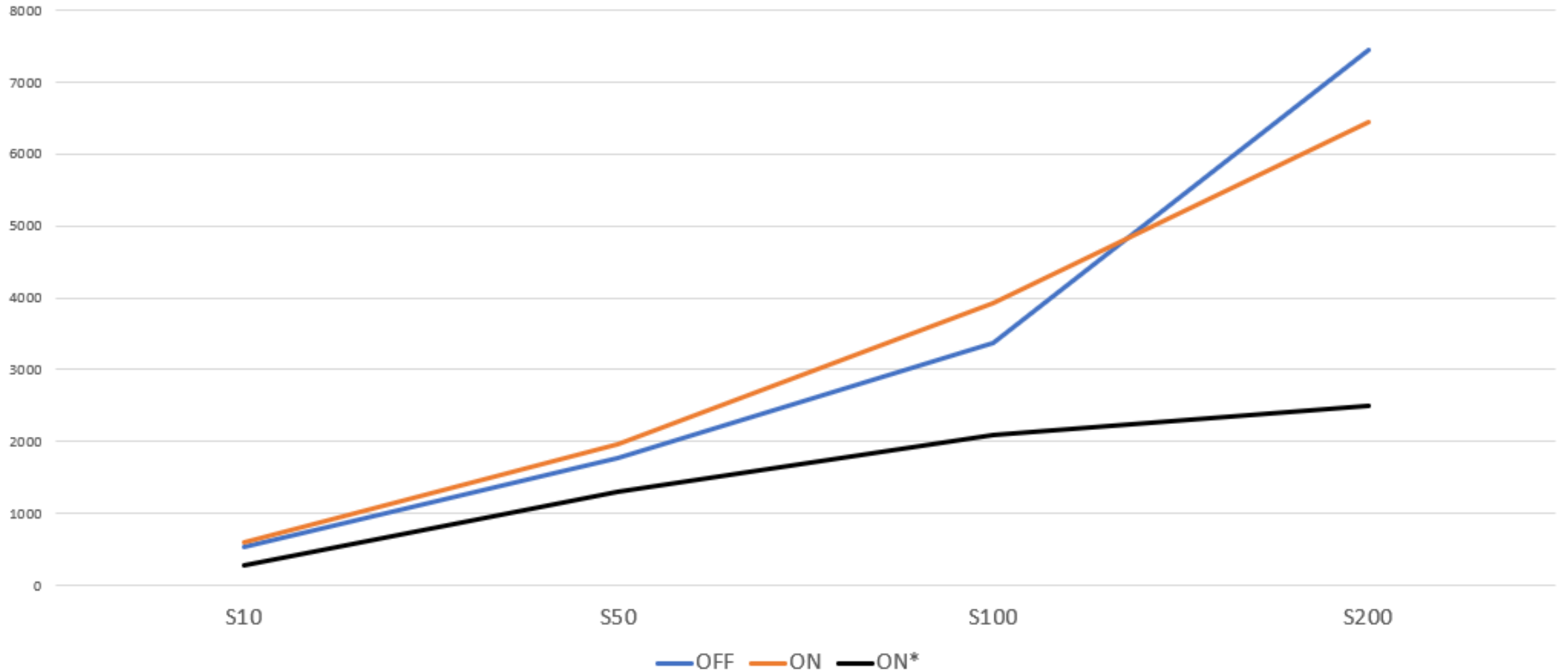
OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



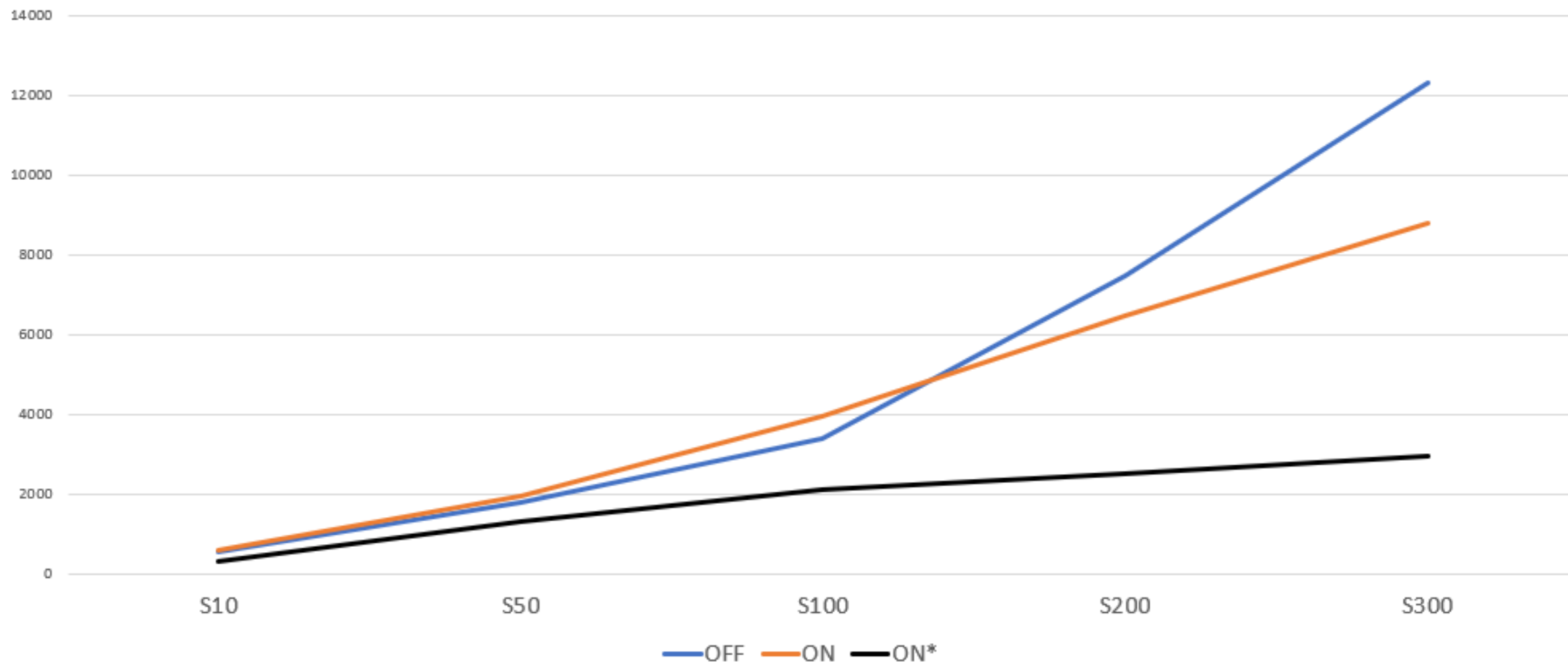
OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



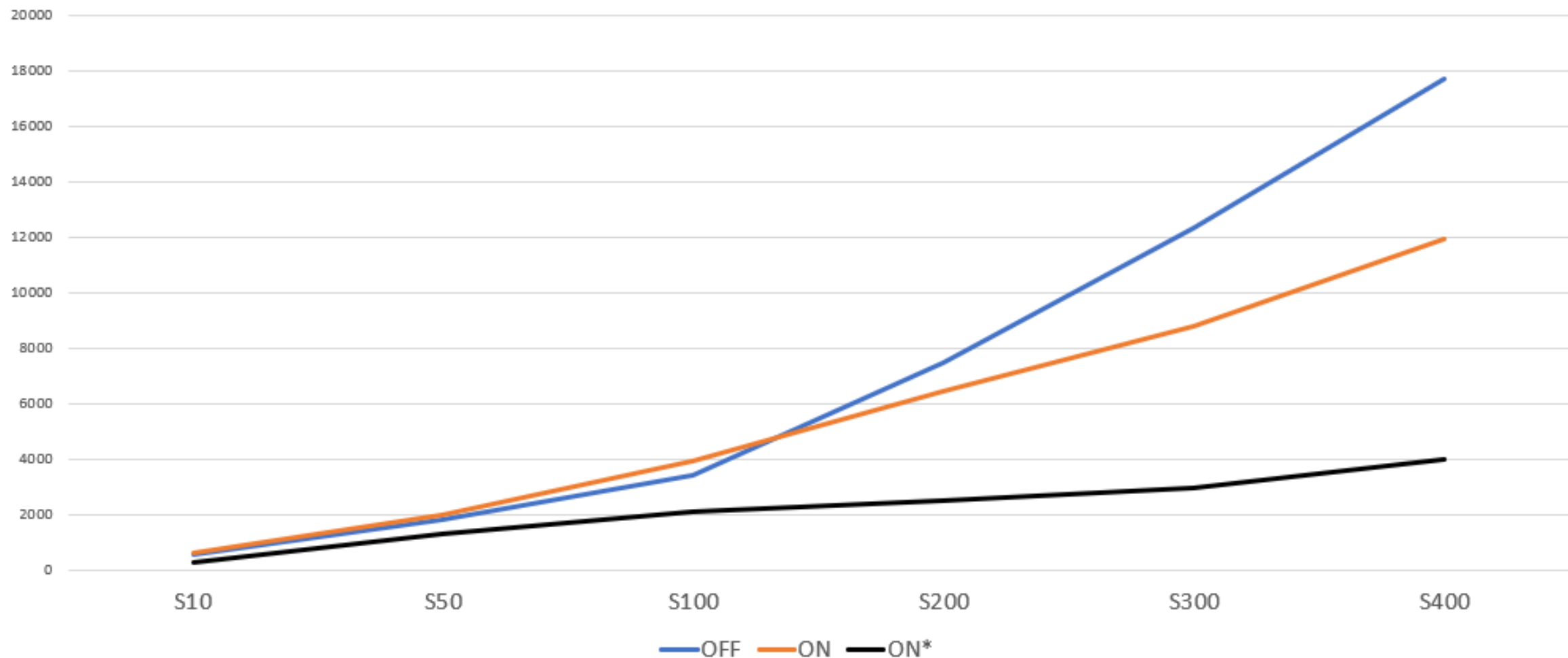
OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



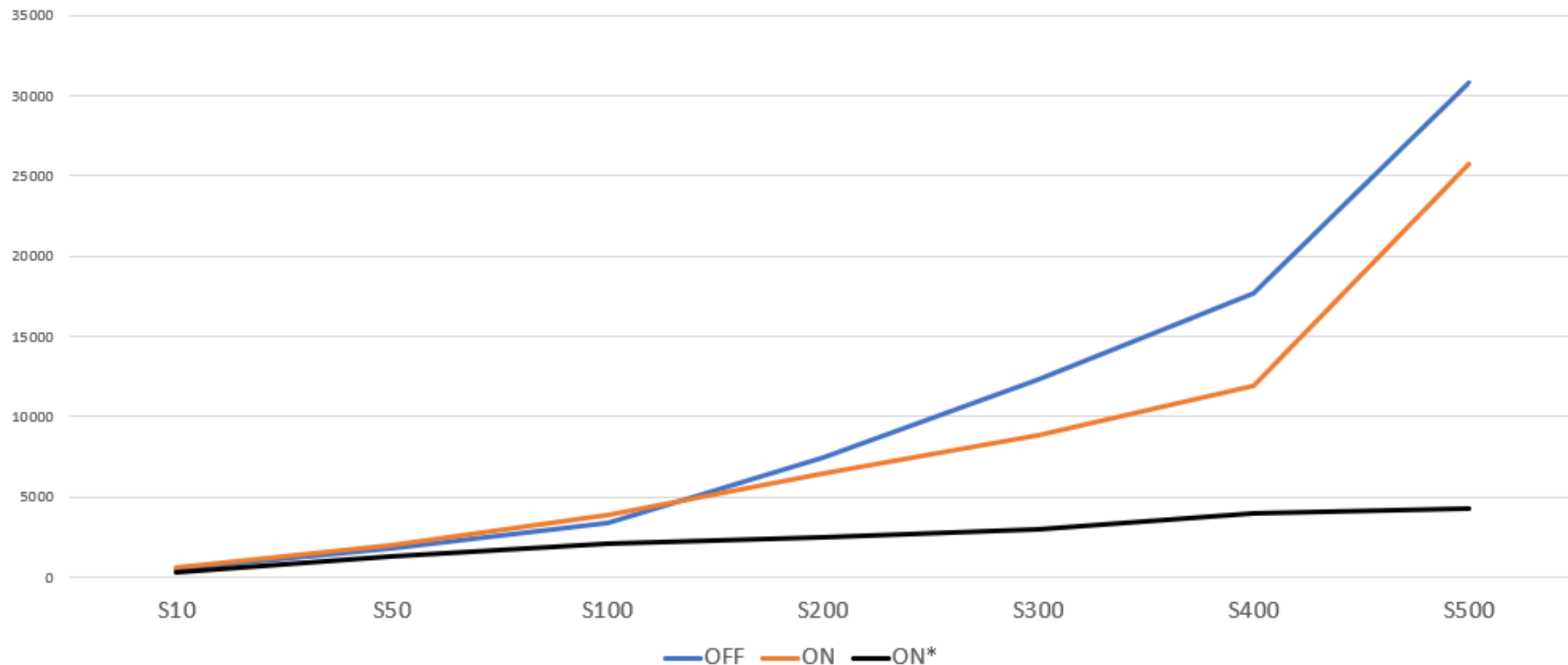
OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



OPTIMIZE_FOR_SEQUENTIAL_KEY - Results



Conclusion

- Alternative to a Hekaton solution
- Does not affect business logic and existing physical implementation (no new indexes, no changes in execution plans)
- Use it only for tables with the last-page contention or where this phenomenon is expected
- It does not bring benefits for not so heavy workload, you can even have a small performance degradation

UTF-8 Support

- To allow applications' internationalization without converting all strings to Unicode
- Implemented as new collation (1.553 new collations)
- UTF-8 document with ASCII characters equal to a classic ASCII document

UTF-8 Support

```
USE AdventureWorks2019;  
DECLARE @v VARCHAR(200) = N'FC ViOn Zlaté Moravce - ЦСКА Москва';  
SELECT DATALENGTH(@v)  
SELECT @v  
GO  
USE OptSeqKey;  
DECLARE @v VARCHAR(200) = 'FC ViOn Zlaté Moravce - ЦСКА Москва';  
SELECT DATALENGTH(@v);  
SELECT @v;
```

100 %
Results Messages

	(No column name)
1	35

	(No column name)
1	FC ViOn Zlaté Moravce - ????? ?????

	(No column name)
1	46

	(No column name)
1	FC ViOn Zlaté Moravce - ЦСКА Москва

```
USE AdventureWorks2019;  
DECLARE @v NVARCHAR(200) = N'FC ViOn Zlaté Moravce - ЦСКА Москва';  
SELECT DATALENGTH(@v)  
SELECT @v  
GO  
USE OptSeqKey;  
DECLARE @v VARCHAR(200) = 'FC ViOn Zlaté Moravce - ЦСКА Москва';  
SELECT DATALENGTH(@v);  
SELECT @v;
```

100 %
Results Messages

	(No column name)
1	70

	(No column name)
1	FC ViOn Zlaté Moravce - ЦСКА Москва

	(No column name)
1	46

	(No column name)
1	FC ViOn Zlaté Moravce - ЦСКА Москва

How does UTF-8 store the data?

Code Range (hexadecimal)	Code Range (decimal)	Storage bytes with UTF-8	Storage bytes with UTF-16
000000 – 00007F (ASCII)	0 - 127	1	2
000080 – 00009F 0000A0 – 0003FF 000400 – 0007FF	128 – 159 160 – 1,023 1,024 – 2,047	2	2
000800 – 003FFF 004000 – 00FFFF	2,048 - 16,383 16,384 – 65,535	3	2
010000 – 03FFFF 040000 – 10FFFF	65,536 – 262,143 262,144 – 1,114,111	4	4

Ideal for typical ASCII documents with a few non-ASCII characters

Query Store Changes in SQL Server 2019

- New default values:

Attribute	SQL Server 2017	SQL Server 2019
MAX_STORAGE_SIZE_MB	100	1 000
QUERY_CAPTURE_MODE	ALL	AUTO

- Asynchronous Loading Query Store is default now (you needed TF 7752 in the previous versions)
- New configuration option for the QUERY_CAPTURE_MODE attribute - CUSTOM

QUERY_CAPTURE_MODE = CUSTOM

Database Properties - WideWorldImporters

Select a page

- General
- Files
- Filegroups
- Options
- Change Tracking
- Permissions
- Extended Properties
- Mirroring
- Transaction Log Shipping
- Query Store

Script ? Help

General

Operation Mode (Actual)	Read write
Operation Mode (Requested)	Read write

Monitoring

Data Flush Interval (Minutes)	50
Statistics Collection Interval	15 Minutes

Query Store Retention

Max Plans Per Query	1000
Max Size (MB)	500
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

Query Store Capture Policy

Execution Count	30
Stale Threshold	1 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

Query Store Capture Mode

Select All to capture all queries.
Select Auto to capture queries based on resource consumption.
Select None to stop the process to capture new queries. Select Custom to capture queries based on custom ...

Lightweight Statistics

- ALTER DATABASE SCOPED CONFIGURATION SET LAST_QUERY_PLAN_STATS = ON;

```
SELECT deqps.query_plan
```

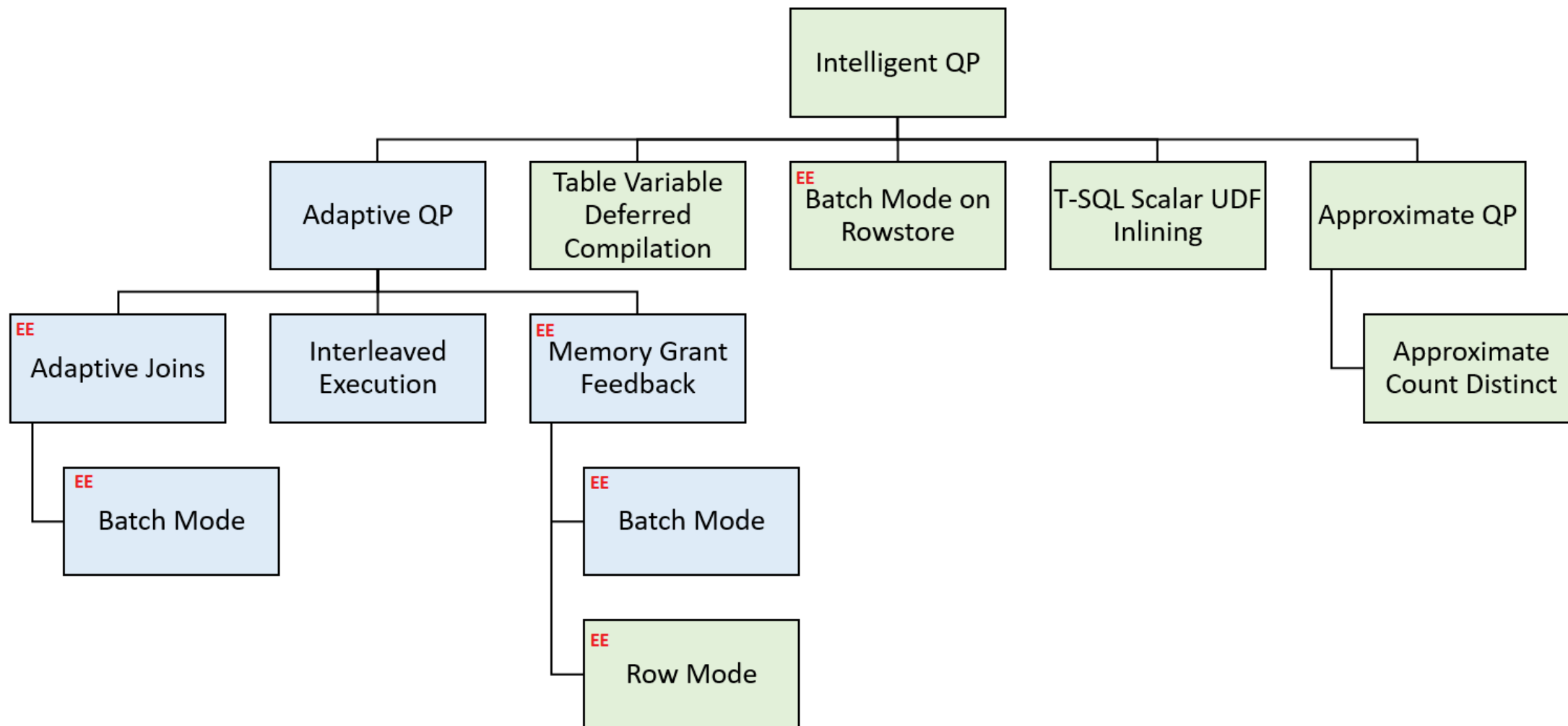
```
FROM sys.dm_exec_procedure_stats AS deps
```

```
    CROSS APPLY sys.dm_exec_query_plan_stats(deps.plan_handle) AS deqps
```

```
WHERE deps.object_id = OBJECT_ID('dbo.x');
```

- This is the actual execution plan, the last one run on the system
- All plans are estimated plans. You can get an estimated plan with runtime metrics. This is what we call an “actual” plan. Just remember that these are the same as estimated plans.

Intelligent Query Processing



INEFFICIENT EXECUTION PLANS IN SQL SERVER

- Execution plans are sometimes suboptimal due to significant inaccuracy in cardinality estimations
- Affected Queries:
 - Queries using table variables
 - Queries with scalar user-defined functions
 - Queries referencing multi-statement table valued functions
 - Complex queries
 - Queries with tables with skew data distribution
- Issues:
 - Operator choice
 - Memory Grant under- or overestimation

LET 'S MAKE EXECUTION PLAN
GREAT AGAIN!



Microsoft®
SQL Server®

QUERY PROCESSING IN SQL SERVER

- **SQL Server 2016 and prior**
 - After the execution plan is created, it is used in consecutive query executions, without changes (with the same operators and memory grants)
- **SQL Server 2017 Adaptive Query Processing**
 - Breaking the pipeline between query optimization and execution
 - Executing a part of the query during the execution plan creation
 - Updating a part of the cached plan during consecutive query executions (Memory Grant)
- **SQL Server 2019 Intelligent Query Processing**
 - Additional improvements, not only adaptive; therefore new name => Intelligent QP

What does batch mode mean?

- Batch mode allows query operators to work on a batch of rows, instead of just one row at a time
- At the CPU level multiple rows processed at once instead of one row
 - Number of processing instructions restricted
 - Better CPU cache utilization and increased memory throughput
 - Not documented how to get number of rows in batch
- Can be beneficial for queries that are CPU bound

What does batch mode mean?

- Batch Mode on Columnstore introduced in SQL Server 2012
 - Improvements – up to 20x faster queries
- Batch Mode on Rowstore introduced in SQL Server 2019
 - Instead of 1 row, a batch with 900 rows is processed at time
 - **Some queries could be significantly faster**
- **In my examples 2-6x faster!**

Batch Mode on Rowstore

- A sample workload comparison row mode vs. batch mode

CL 140

CL 150

The screenshot shows the SQLQueryStress application window. The left pane contains SQL queries. The right pane displays performance metrics for a workload labeled CL 140. The metrics are as follows:

Metric	Value
Elapsed Time	00:00:22.4115
Iterations Completed	10
Client Seconds/Iteration (Avg)	2,2316
Total Exceptions	0
CPU Seconds/Iteration (Avg)	14,3122
Logical Reads/Iteration (Avg)	28662,0000
Actual Seconds/Iteration (Avg)	2,2299

The screenshot shows the SQLQueryStress application window. The left pane contains SQL queries. The right pane displays performance metrics for a workload labeled CL 150. The metrics are as follows:

Metric	Value
Elapsed Time	00:00:07.8514
Iterations Completed	10
Client Seconds/Iteration (Avg)	0,7786
Total Exceptions	0
CPU Seconds/Iteration (Avg)	5,3315
Logical Reads/Iteration (Avg)	28662,0000
Actual Seconds/Iteration (Avg)	0,7764

Batch Mode on Rowstore

- Native support
 - No tricks with fake columnstore indexes or other optimizer delusions
- Initial heuristics considers potential benefits of batch mode for operators
 - At least one table has \geq **131.072** rows
 - At least one potentially efficient batch operators: join, aggregate or window aggregate
 - At least one of the batch operator's input should have not less than 131.072 rows

Batch Mode on Rowstore – More info

- Extended event *batch_mode_heuristics*

- More info:

- Blog: Niko Neugebauer

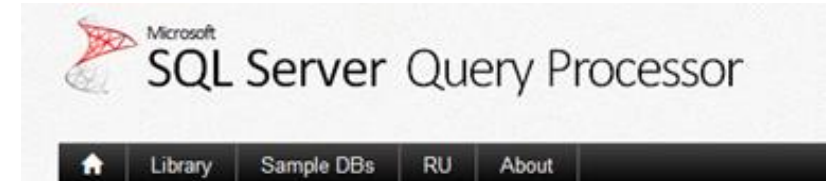
- <http://www.nikoport.com/2018/10/07/batch-mode-part-3-basic-heuristics-analysis/>

Niko Neugebauer

SQL Server, Columnstore, Data Platform & Community

- Blog: Dima Pilugin

- <http://www.queryprocessor.com/batch-mode-on-row-store/>



Batch Mode on Rowstore

Enabling

```
ALTER DATABASE current SET COMPATIBILITY_LEVEL = 150;
```

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ON_ROWSTORE = ON;
```

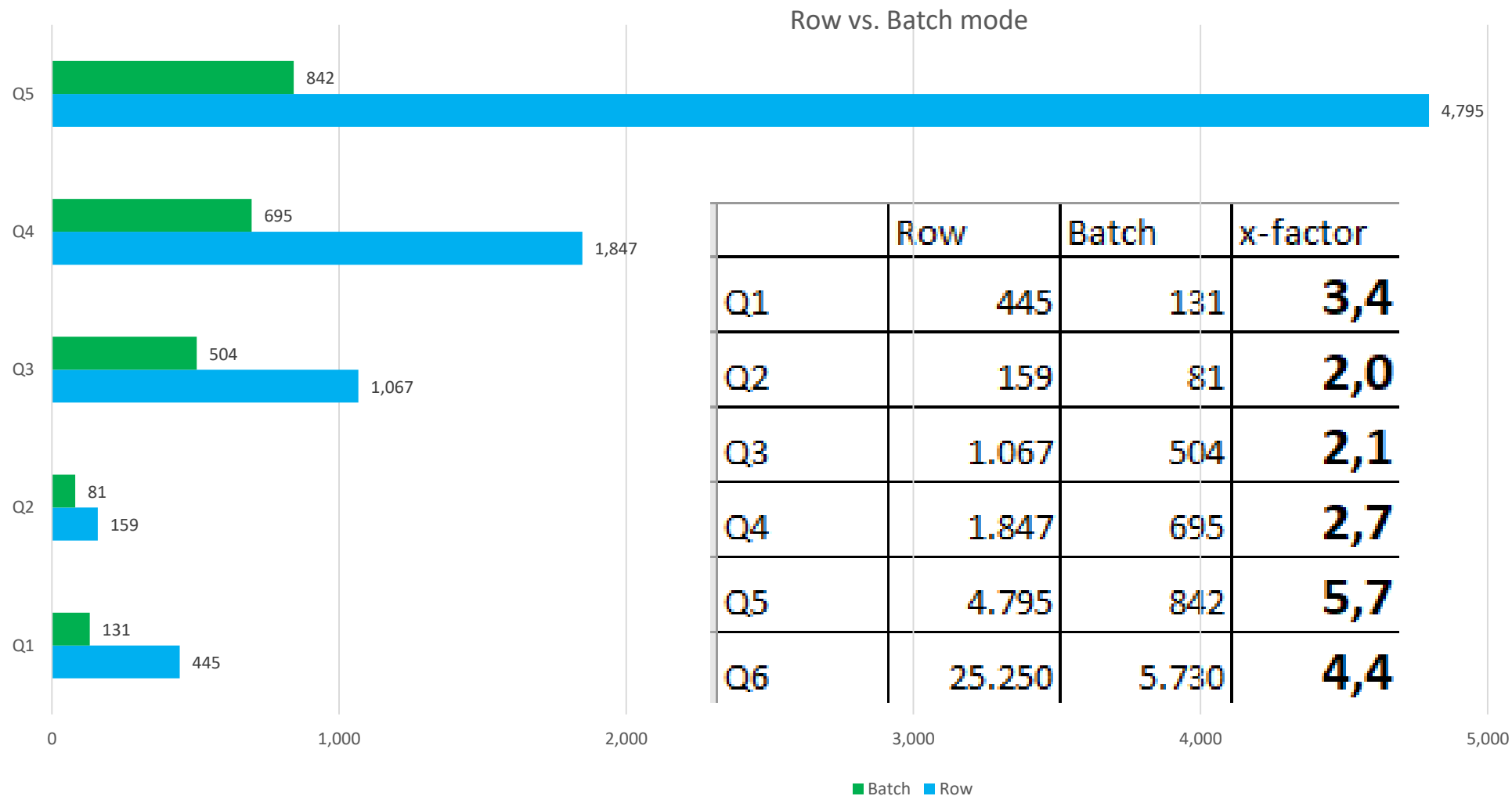
```
OPTION (USE HINT('ALLOW_BATCH_MODE'));
```

Disabling

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ON_ROWSTORE = OFF;
```

```
OPTION (USE HINT('DISALLOW_BATCH_MODE'));
```

Batch Mode on Rowstore – Results

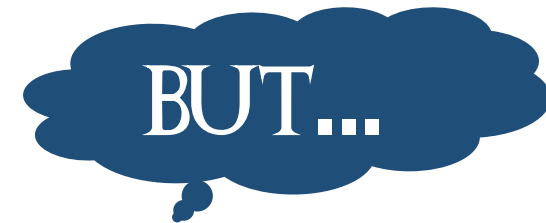


Batch Mode on Rowstore - Conclusion

- Very promising feature
 - Improvements with no efforts
 - It could be a reason for upgrade for some companies
- First version, probably will not optimize some queries that need an optimization
- It brings benefits for queries with large tables and datasets
- Possible regressions!
- **It could significantly affect an OLTP workload!**

Scalar UDFs in SQL Server

- Code reuse, encapsulation and modularity
- Complex business rules or computations
- Single place change
- Written once, invoke from many modules
- Reduce network traffic



Scalar UDFs in SQL Server

Why do SQL Server Scalar-valued functions get slower?

Refactor SQL Server scalar UDF to inline TVF to improve performance

Why SQL Server scalar functions are bad?

T-SQL Best Practices - Don't Use Scalar Value Functions in Column .

Are SQL Server Functions Dragging Your Query Down?

SQL functions rarely perform well.

Scalar UDF Inlining

- <http://www.vldb.org/pvldb/vol11/p432-ramachandra.pdf>

Froid: Optimization of Imperative Programs in a Relational Database

Karthik Ramachandra
Microsoft Gray Systems Lab

karam@microsoft.com

Alan Halverson
Microsoft Gray Systems Lab

alanhal@microsoft.com

Kwanghyun Park
Microsoft Gray Systems Lab

kwpark@microsoft.com

César Galindo-Legaria
Microsoft

cesarg@microsoft.com

K. Venkatesh Emani^{*}
IIT Bombay

venkateshek@cse.iitb.ac.in

Conor Cunningham
Microsoft

conorc@microsoft.com

ABSTRACT

For decades, RDBMSs have supported declarative SQL as well as imperative functions and procedures as ways for users to express data processing tasks. While the evaluation of declarative SQL has received a lot of attention resulting in highly sophisticated techniques, the evaluation of imperative programs has remained naïve and highly inefficient. Imperative programs offer several benefits over SQL and hence are

expressing intent has on one hand provided high-level abstractions for data processing, while on the other hand, has enabled the growth of sophisticated query evaluation techniques and highly efficient ways to process data.

Despite the expressive power of declarative SQL, almost all RDBMSs support procedural extensions that allow users to write programs in various languages (such as Transact-SQL, C#, Java and R) using imperative constructs such

Scalar UDF Inlining

- **Goal** – improve queries where scalar UDFs are problem
- **Scalar UDF Inlining feature:**
 - Transforms scalar UDF into relational expressions or subqueries
 - IF => CASE WHEN
 - RETURN => SELECT
 - Embeds them in the calling query
 - Optimize expressions or subqueries
- **Result:**
 - Performance improved (more efficient plan)
 - Execution plan could be parallel

Scalar UDF Inlining

Table 1: Relational algebraic expressions for imperative statements (using standard T-SQL notation from [33])

Imperative Statement (T-SQL)	Relational expression (T-SQL)
DECLARE {@var data_type [= expr]}[, ... n];	SELECT {expr null AS var}[, ... n];
SET {@var = expr}[, ... n];	SELECT {expr AS var}[, ... n];
SELECT {@var1 = prj_expr1}[, ... n] FROM sql_expr;	{SELECT prj_expr1 AS var1 FROM sql_expr}; [, ... n]
IF (pred_expr) {t_stmt; [, ... n]} ELSE {f_stmt; [, ... n]}	SELECT CASE WHEN pred_expr THEN 1 ELSE 0 END AS pred_val; {SELECT CASE WHEN pred_val = 1 THEN t_stmt ELSE f_stmt; }[, ... n]
RETURN expr;	SELECT expr AS returnVal;

Froid: Optimization of Imperative Programs in a Relational Database

Scalar UDFs in SQL Server

- Not all scalar UDFs can be inlined
- Check whether a function can be inlined:

```
SELECT CONCAT(SCHEMA_NAME(o.schema_id), '.', o.name), is_inlineable
FROM sys.sql_modules m
INNER JOIN sys.objects o ON o.object_id = m.object_id
WHERE o.type = 'FN';
```

- **Is_inlineable** does not imply that it will always be inlined
- Decision is made when the query referencing a scalar UDF is compiled

Scalar UDF Inlining

- Enable

```
ALTER DATABASE current SET COMPATIBILITY_LEVEL = 150;
```

```
ALTER DATABASE SCOPED CONFIGURATION SET TSQL_SCALAR_UDF_INLINING = ON;
```

```
CREATE OR ALTER FUNCTION dbo.getMaxOrderDate(@CustID INT) RETURNS DATETIME WITH INLINE = ON
```

Disable

```
ALTER DATABASE SCOPED CONFIGURATION SET TSQL_SCALAR_UDF_INLINING = OFF;
```

```
OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'));
```

```
CREATE OR ALTER FUNCTION dbo.getMaxOrderDate(@CustID INT) RETURNS DATETIME WITH INLINE = OFF
```

Scalar UDF Inlining - Results

	Scalar CL 140	Scalar CL 150	Inline TVF	
F1 Quantity*UnitPrice	23.50	1.90	1.48	13x faster
F2 SUM(Quantity* UnitPrice)	75.40	0.7	0.21	>100x faster
F3 GetMaxOrderDateForCustID	5.60	4.45	0.14	25% faster

Scalar UDF Inlining - Limitations

- UDF does not invoke any intrinsic function that is either time-dependent or has side effects such as **GETDATE()** or NEWSEQUENTIALID
- Cannot reference table variables, table-valued parameters and user-defined types
- The query invoking a scalar UDF
 - does not reference a scalar UDF call in its GROUP BY clause
 - in its select list with DISTINCT clause does not reference a scalar UDF call in its ORDER BY clause
- UDF is not natively compiled (interop is supported)
- UDF is not used in a computed column or a check constraint definition
- The UDF is not a partition function

Memory Grant Feedback

- Adjust memory grant parameter in the execution plan AFTER the plan is generated
 - after a few query executions
- Memory is adjusted for a query when
 - It used less than 50% of granted memory
 - Is spilling out to tempdb
- In SQL Server 2017 requires a columnstore index on the affected table

Memory Grant Feedback

- If memory grant memory values oscillate, the feature is disabled
- New plan attributes in the XML plan

- It works with cache
- It does not work with...
- It is not persisted if...
- **In SQL Server 2019**

```
<MemoryGrantInfo SerialRequiredMemory="152"  
SerialDesiredMemory="1240"  
RequiredMemory="1352"  
DesiredMemory="2440"  
RequestedMemory="2440"  
GrantWaitTime="0"  
GrantedMemory="2440"  
MaxUsedMemory="1752"  
MaxQueryMemory="1334464"  
LastRequestedMemory="670408"  
IsMemoryGrantFeedbackAdjusted="Yes: Adjusting"  
/>
```

Problems with Table Variables

- Queries using table variables with large number of rows (> 10K) can have an inefficient plan
- Inappropriate operators in the execution plan
 - mostly Nested Loop Join instead of Hash Join
- Insufficient memory grants
 - Number of processing rows is usually underestimated => less Memory Grant reserved for the query => spills to tempdb

Table Variable Deferred Compilation

- Improves plan quality and overall performance for queries referencing table variables
- Cardinality estimates are based on actual table variable row counts
- This accurate row count information will be used for optimizing downstream plan operations

Table Variable Deferred Compilation

```
DECLARE @T AS TABLE (ProductID INT)  
INSERT INTO @T SELECT ProductID  
FROM Production.Product  
WHERE ProductLine IS NOT NULL;
```

```
SELECT * FROM @T t  
INNER JOIN Sales.SalesOrderDetail o  
ON t.ProductID = od.ProductID  
INNER JOIN Sales.SalesOrderHeader h  
ON h.SalesOrderID = od.SalesOrderID  
ORDER BY od.UnitPrice DESC;
```



SQL Server 2017 and all previous versions

- Content of the table variable unknown at the compile time
=> cardinality 1



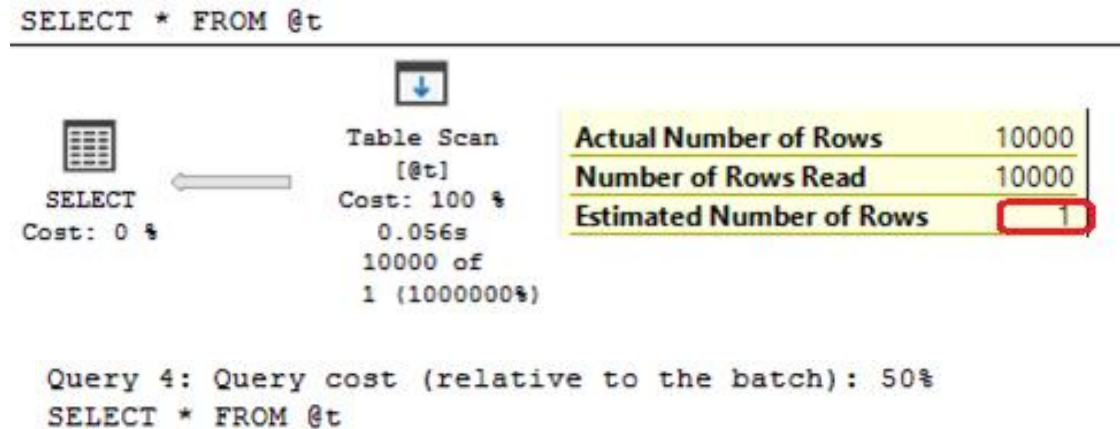
SQL Server 2019

- Content of the table variable known at the compile time =>
cardinality = actual number of rows

Table Variable Deferred Compilation

```
DECLARE @t TABLE(id INT)
INSERT INTO @t SELECT n FROM dbo.GetNums(10000);
SELECT * FROM @t;
```

Compatibility Level 140



Compatibility Level 150

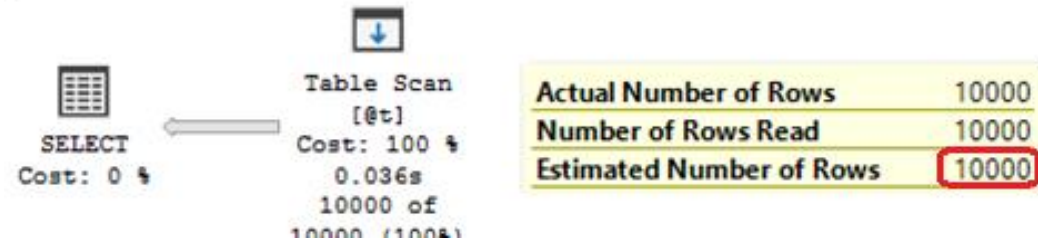


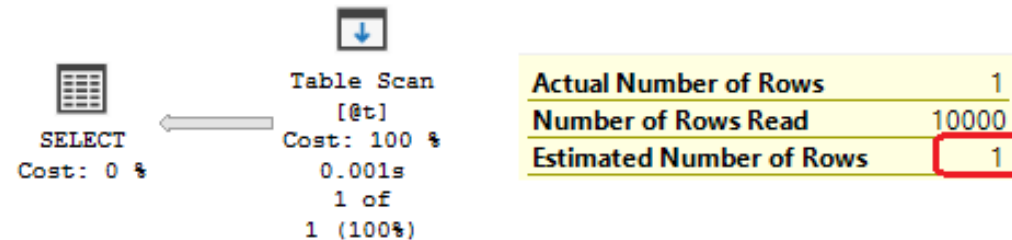
Table Variable Deferred Compilation

- `DECLARE @t TABLE(id INT)`
- `INSERT INTO @t SELECT n FROM dbo.GetNums(10000);`
- `SELECT * FROM @t WHERE id = 5;`

SQRT (C)

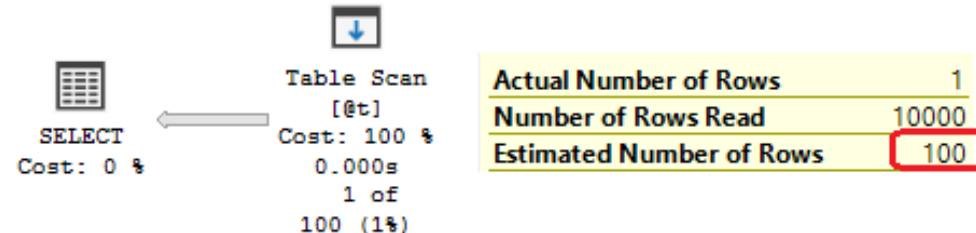
Compatibility Level 140

Query 2: Query cost (relative to the batch): 0%
`SELECT * FROM @t WHERE id = 5`



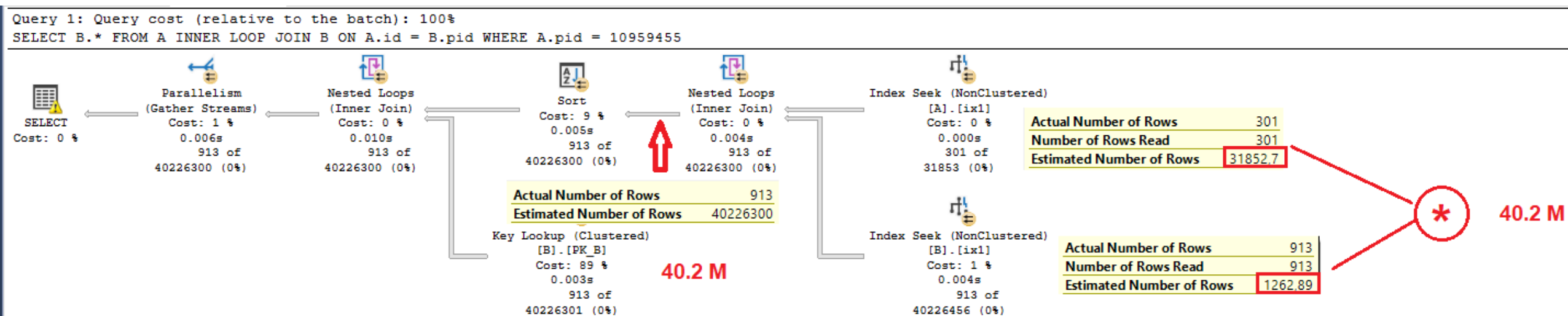
Compatibility Level 150

Query 4: Query cost (relative to the batch): 50%
`SELECT * FROM @t WHERE id = 5`



Cardinality Issue

```
SELECT B.* FROM A  
INNER LOOP JOIN B ON A.id = B.pid  
WHERE A.pid = 413032;
```



Using TV to fix a cardinality issue

```
DECLARE @t TABLE(id INT PRIMARY KEY);  
INSERT INTO @t SELECT A.id FROM A WHERE A.pid = 413032;  
SELECT B.* FROM @t A  
INNER JOIN B ON A.id = B.pid;
```

Query 2: Query cost (relative to the batch): 36%
SELECT B.* FROM @t A INNER JOIN B ON A.id = B.pid

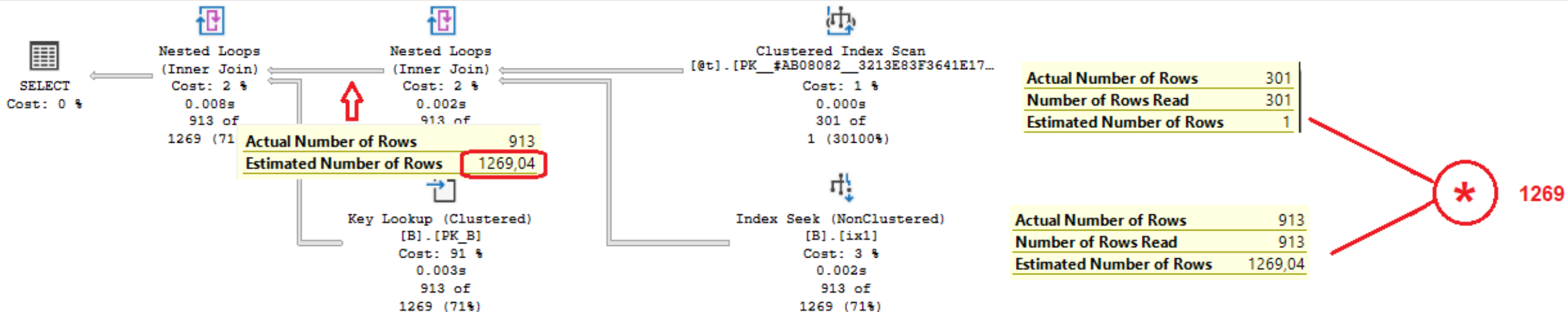





Table Variable Deferred Compilation

- Designed to address cardinality issues caused by fixed estimation:
 - Nested Loop Joins where Hash Joins are more appropriate
 - Memory grant underestimation issues
-  • Better estimation for execution plans for new queries
-  • Prone to parameter sniffing
-  • Can break existing workarounds

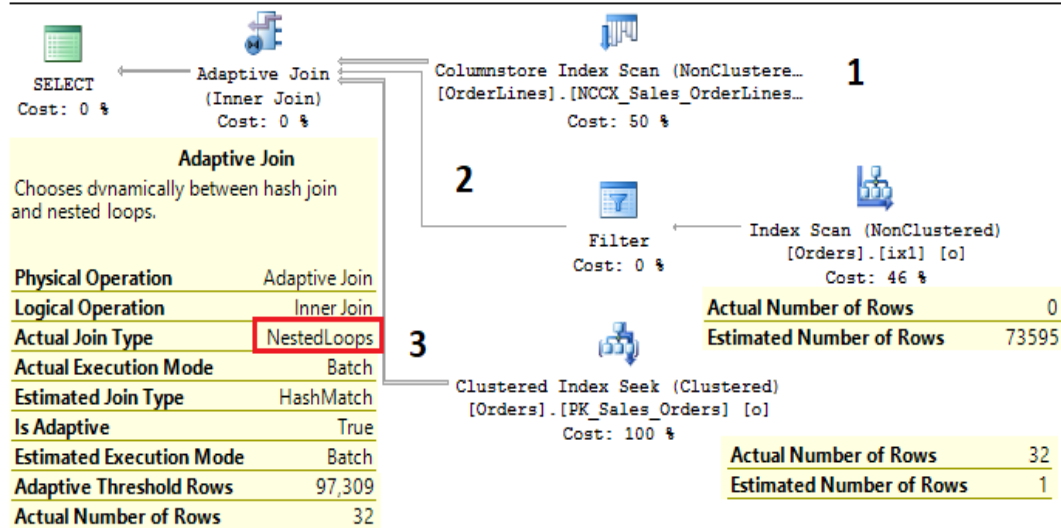
Batch Mode Adaptive Join

- **New operator- Adaptive Join**
 - Allows to choose between Hash Join and Nested Loop Join at runtime
 - It starts as Hash Join and if after input scanning
 - estimated number of rows < threshold => switches to Nested Loop Join
 - estimated number of rows >= threshold => continues as Hash Join
- It will better handle queries with variety of parameters, but it won't solve all issues caused by wrongly chosen Join operator
- **It (still) requires at least one columnstore index**

Batch Mode Adaptive Join

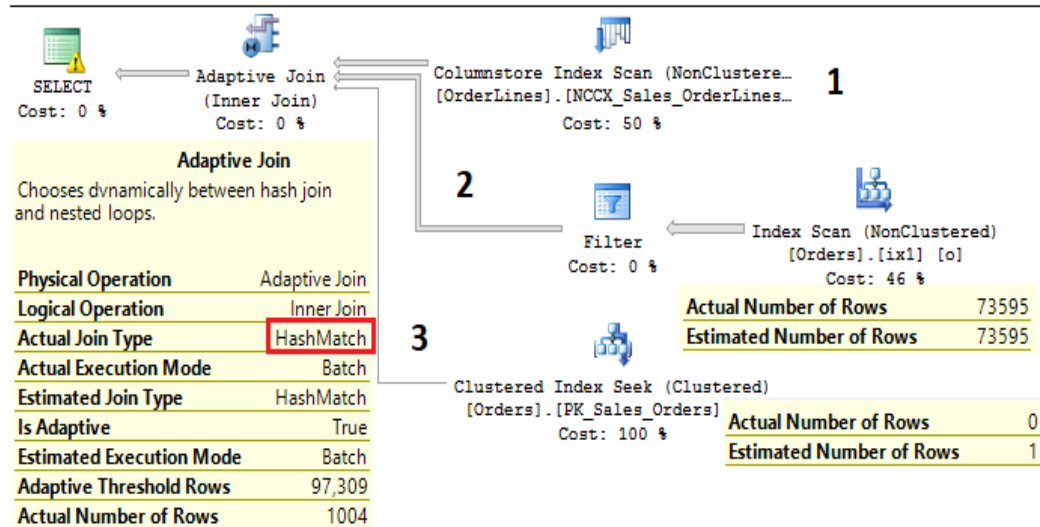
EXEC dbo.GetOrderDetails 1;

```
SELECT o.OrderID, o.OrderDate, ol.OrderLineID, ol.Quantity, ol.UnitPrice FROM Sales
Missing Index (Impact 49.4219): CREATE NONCLUSTERED INDEX [<Name of Missing Index,
```



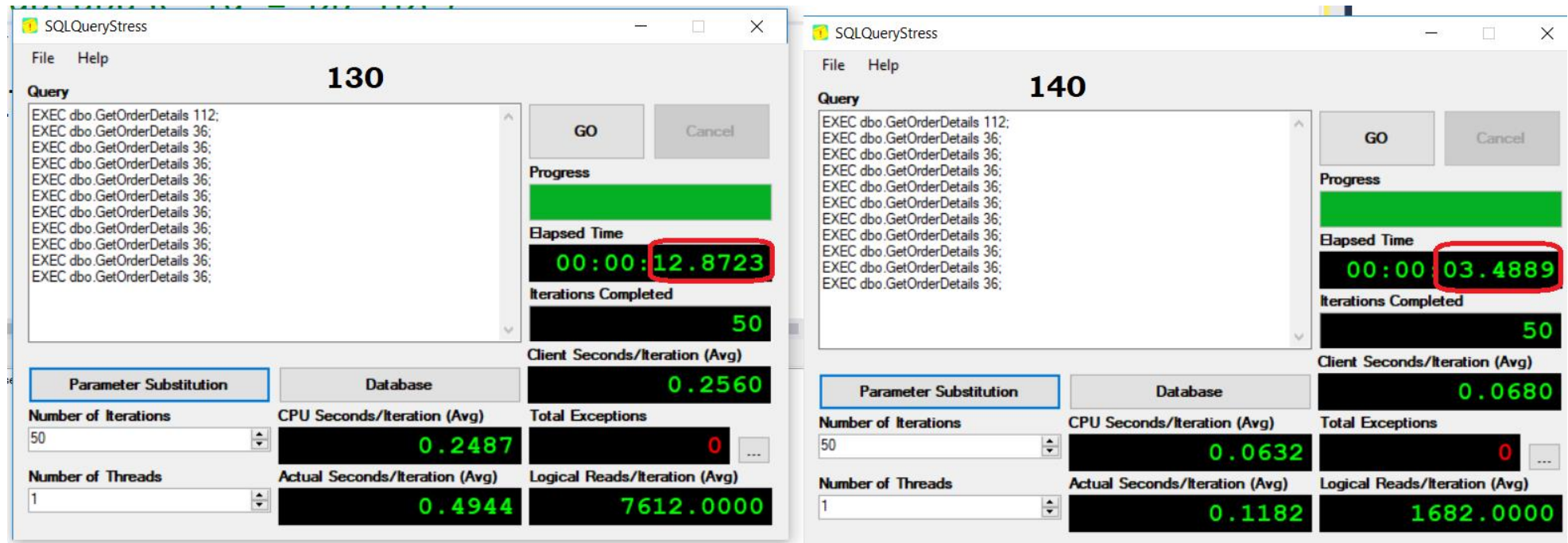
EXEC dbo.GetOrderDetails 112;

```
SELECT o.OrderID, o.OrderDate, ol.OrderLineID, ol.Quantity, ol.UnitPrice FROM Sales
Missing Index (Impact 49.4219): CREATE NONCLUSTERED INDEX [<Name of Missing Index,
```



Batch Mode Adaptive Join

Q: Is it better with new Adaptive Join operator?

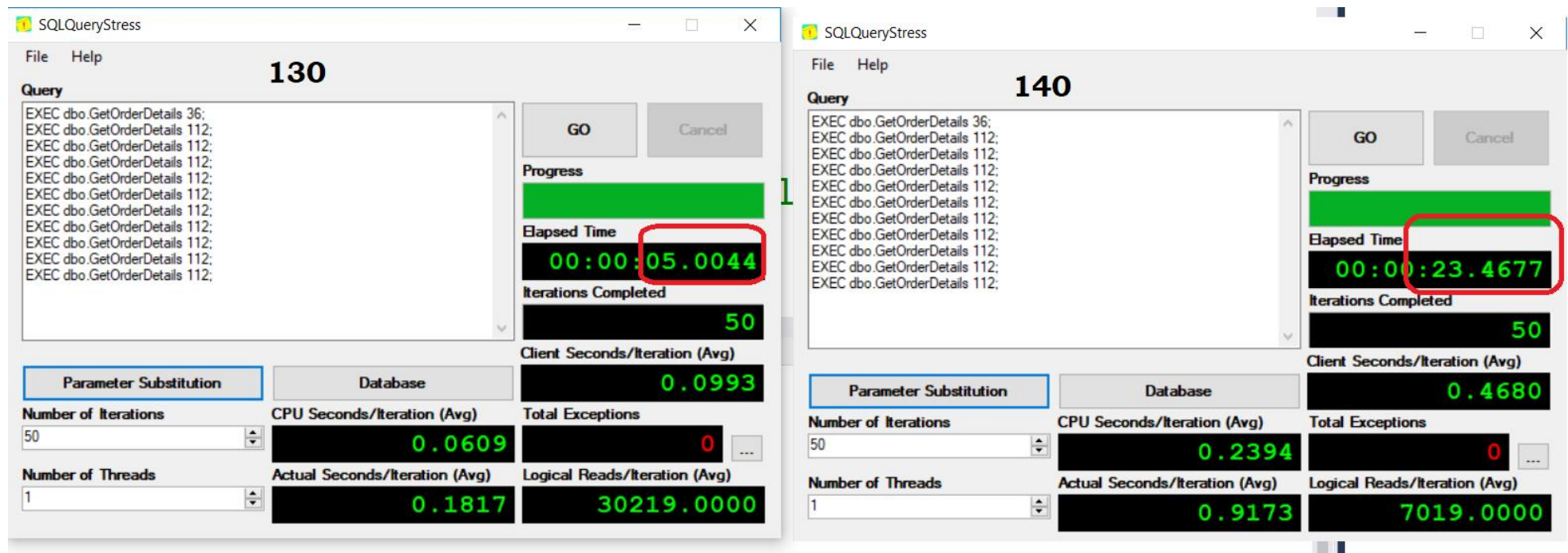


- A: **Yes!!!** Under the CL 140, the query runs **4x faster!**

Batch Mode Adaptive Join

Q: Is it better with new Adaptive Join operator?

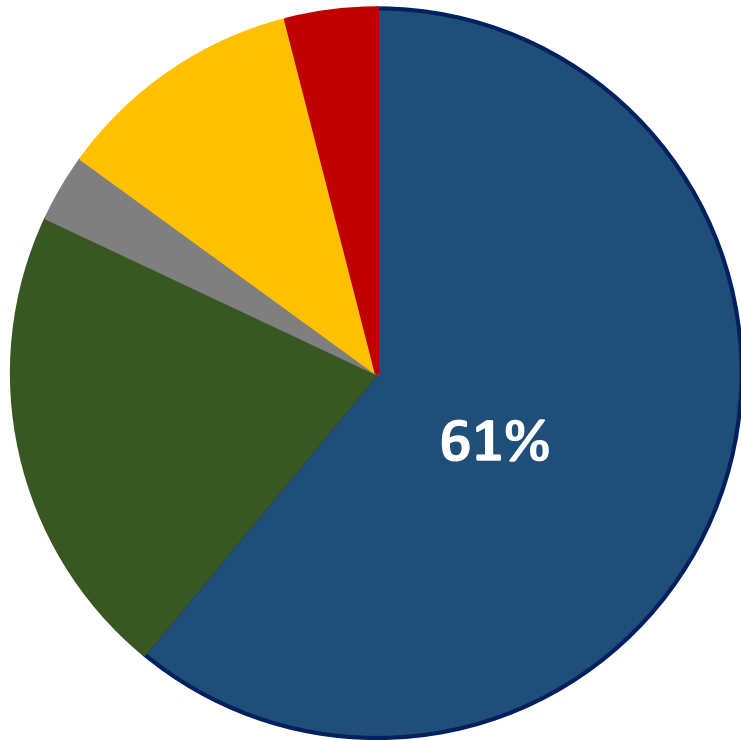
IT DEPENDS!



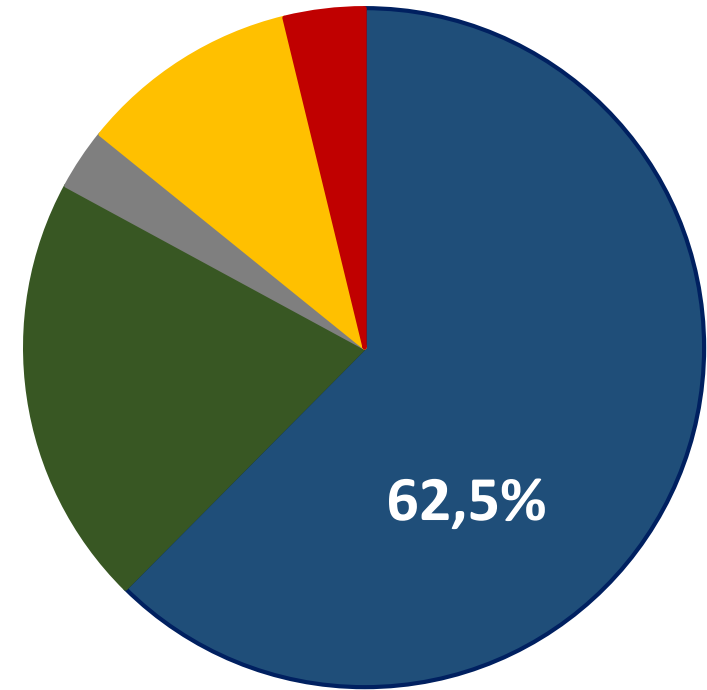
- A: Actually **NO** - under the CL 140, the query runs **5x slower!**

Approximate Query Processing

You don't need always exact answers



■ AIK Stockholm ■ Djurgårdens IF ■ IFK Göteborg ■ Malmö ■ IFK Norrköping



■ AIK Stockholm ■ Djurgårdens IF ■ IFK Göteborg ■ Malmö ■ IFK Norrköping

Approximate Query Processing

- Implemented or supported
 - Apache Spark, PostgreSQL, Oracle, Teradata, Vertica Analytics, Amazon Redshift DWH, Google BigQuery, BlinkDB (Facebook), Conviva, Verdict
- Microsoft Research

Approximate Query Processing: No Silver Bullet

Surajit Chaudhuri, Bolin Ding, Srikanth Kandula
Microsoft Research
{surajitc,bolind,srikanth}@microsoft.com

Approximate Query Processing: No Silver Bullet

Surajit Chaudhuri, Bolin Ding, Srikanth Kandula

May 2017

ACM - Association for Computing Machinery



https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/sigmod17_aqp_sb.pdf

ABSTRACT

In this paper, we reflect on the state of the art of Approximate Query Processing. Although much technical progress has been made in this area of research, we are yet to see its impact on products and services. We discuss two promising avenues to pursue towards integrating Approximate Query Processing into data platforms.

1. INTRODUCTION

While Big Data opens the possibility of gaining unprecedented insights, it comes at the price of increased need for computational resources (or risk of higher latency) for answering queries over voluminous data. The ability to provide approximate answers to queries at a fraction of the cost of executing the query in the traditional way, has the disruptive potential of allowing us to explore large datasets efficiently. Specifically, such techniques could prove effective in helping data scientists identify the subset of data that

approximation is interesting for applications, approximation at the query processing layer has proven ineffective for applications, because either the semantics of error models and the accuracy guarantees of AQP or the extent of savings in work accrued by AQP have been unsatisfactory.

We firmly believe that the value proposition of AQP, outlined in the opening of this article, is considerable in the world of big data. We should not give up the pursuit of such systems. However, critical rethinking of our approach to AQP research is warranted with the exclusive goal of making such systems practical [30]. The first step in such rethinking is to be clear about what combinations of the four dimensions of AQP will make it possible for applications to find AQP systems attractive. In this article, we suggest two research directions to pursue based on our reflection.

One promising approach may be to cede control over accuracy to the user. This approach is based on accepting the reality that AQP systems will not be able to offer a priori (*i.e.*, before the query

Approximate Query Processing

- Provide aggregations across very large data sets where responsiveness is more critical than absolute precision
- Currently only one function **APPROX_COUNT_DISTINCT**
 - It uses significantly less memory resources
 - Error from the precise COUNT DISTINCT equivalent
 - within 2% for most workloads
 - within 20% for all workloads
- Access of data sets that are millions of rows or higher
- Aggregation of a column or columns that have a large number of distinct values

APPROX_COUNT_DISTINCT

- It uses HyperLogLog algorithm
- Philippe Flajolet et al.
- <http://algo.inria.fr/flajolet/Publications/FIFuGaMe07.pdf>

2007 Conference on Analysis of Algorithms, AofA 07

DMTCS proc. AH, 2007, 127–146

HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm

Philippe Flajolet¹ and Éric Fusy¹ and Olivier Gandouet² and Frédéric Meunier¹

¹Algorithms Project, INRIA–Rocquencourt, F78153 Le Chesnay (France)

²LIRMM, 161 rue Ada, 34392 Montpellier (France)

This extended abstract describes and analyses a near-optimal probabilistic algorithm, HYPERLOGLOG, dedicated to estimating the number of *distinct* elements (the *cardinality*) of very large data ensembles. Using an auxiliary memory of m units (typically, “short bytes”), HYPERLOGLOG performs a single pass over the data and produces an estimate of the cardinality such that the relative accuracy (the *standard error*) is typically about $1.04/\sqrt{m}$. This improves on the best previously known cardinality estimator, LOGLOG, whose accuracy can be matched by consuming only 64% of the original memory. For instance, the new algorithm makes it possible to estimate cardinalities well beyond 10^9 with a typical accuracy of 2% while using a memory of only 1.5 kilobytes. The algorithm parallelizes optimally and adapts to the sliding window model.

Approximate Query Processing

- It hashes each element to make the data distribution more uniform
- After hashing all the elements, it looks for the binary representation of each hashed element
- HLL looks number of leading zero bits in the hash value of each element and finds maximum number of leading zero bits
- *Number of distinct elements* = $2^{(k+1)}$
where k is maximum number of leading zeros in data set
- HyperLogLog algorithm documentation
- <http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf>

APPROX_COUNT_DISTINCT

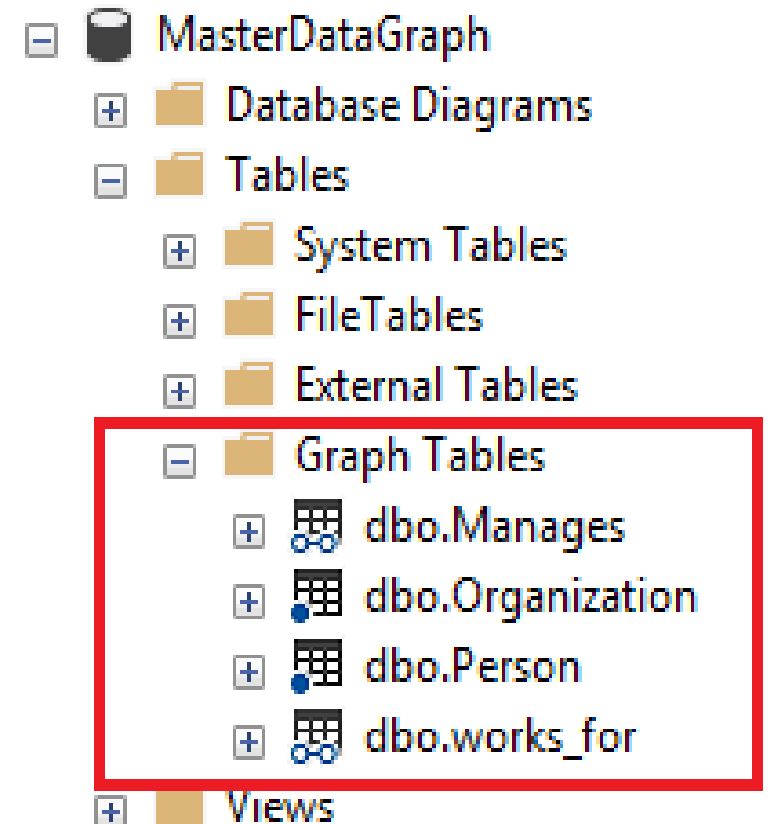
- **Sometimes** better plan
 - Stream Aggregate instead of Hash Match Aggregate, for instance
- **Significantly** less memory
- **Sometimes** faster
 - speed is not main reason for using it, it could be even slower
- Useful for
 - data sets > millions of rows
 - columns with large number of distinct values

SQL Graph in SQL Server 2017

- Graph – collection of node and edge tables
- Language Extensions
 - **DDL Extensions** – create node/edge tables
 - **Query Language Extensions** –

New built-in: **MATCH**, to support pattern matching ar

- Tooling and Eco-system
- Integrated into the SQL Engine
 - Existing tools all work out of the box, for example ! export, etc.
 - Queries can join existing tables and graph node / ec



SQL Graph in SQL Server 2019

- Edge Constraints
- New graph function - `SHORTEST_PATH`
- Partition tables and indexes
- Use derived table or view aliases in graph match query