



In-Memory Tables and Natively Compiled T-SQL

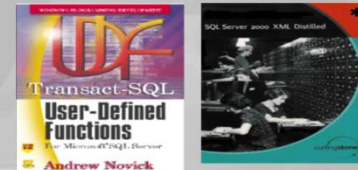
Blazing Speed for OLTP and MOre



Andy Novick



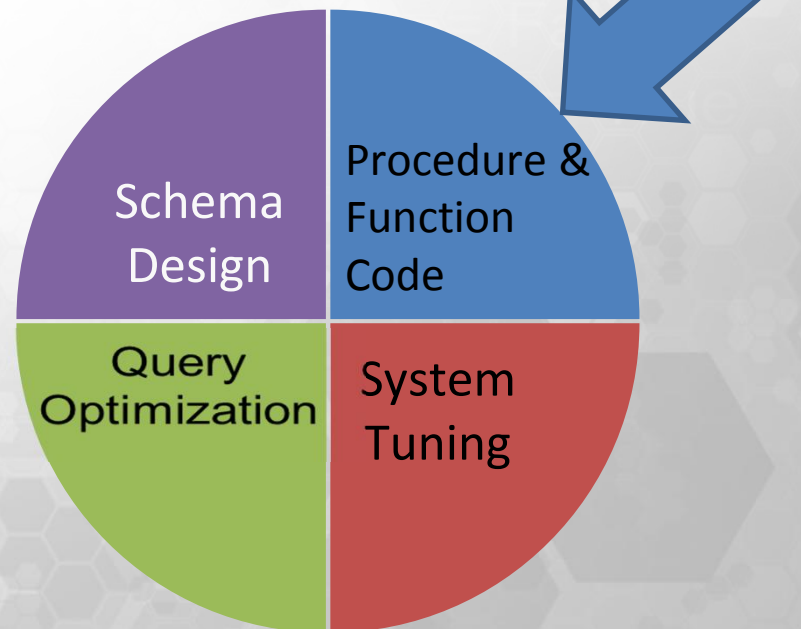
- SQL Server Consultant
- SQL Server MVP since 2010
- Author of 2 books on SQL Server
- anovick@NovickSoftware.com
- www.NovickSoftware.com



Agenda

- The Hekaton Overview
- In-Memory Tables 2016
- The Natively Compiled Difference

Focus of this Presentation



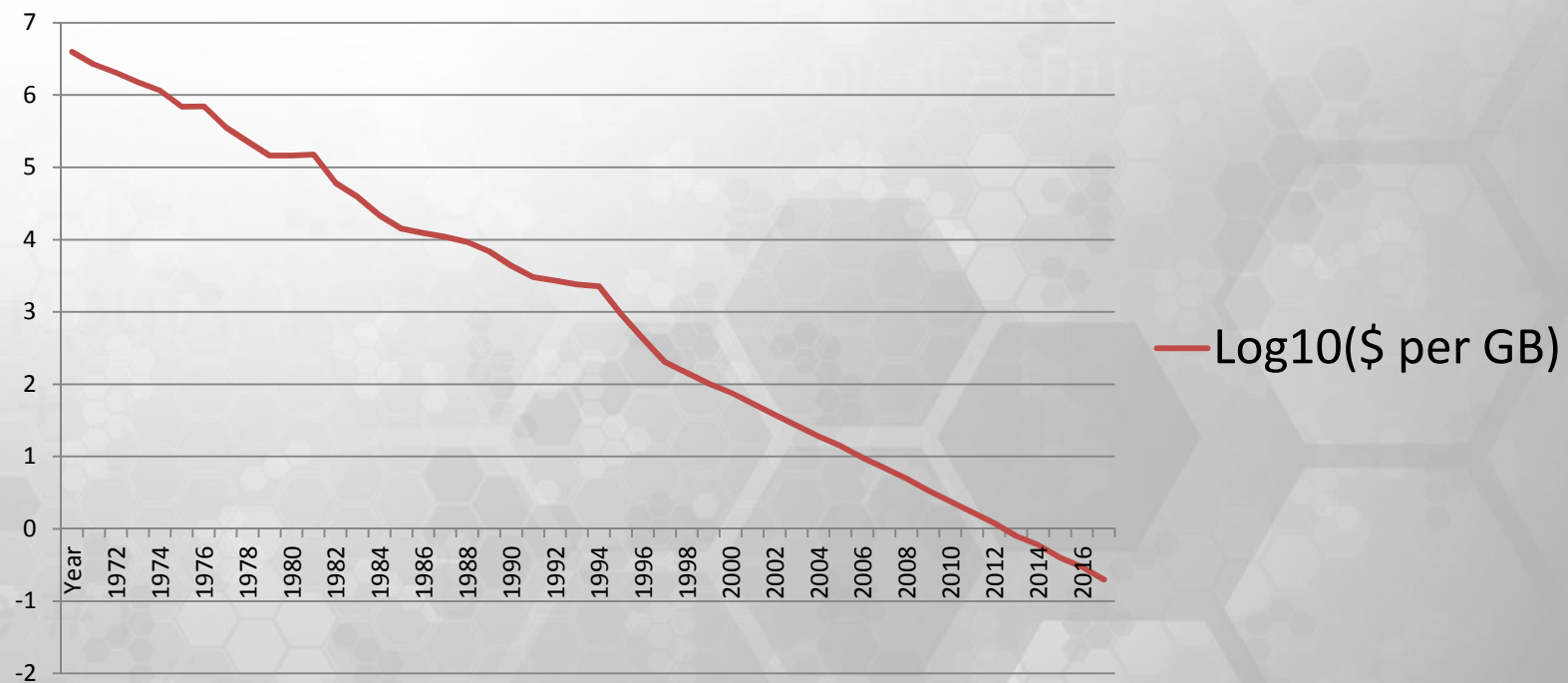
HEKATON

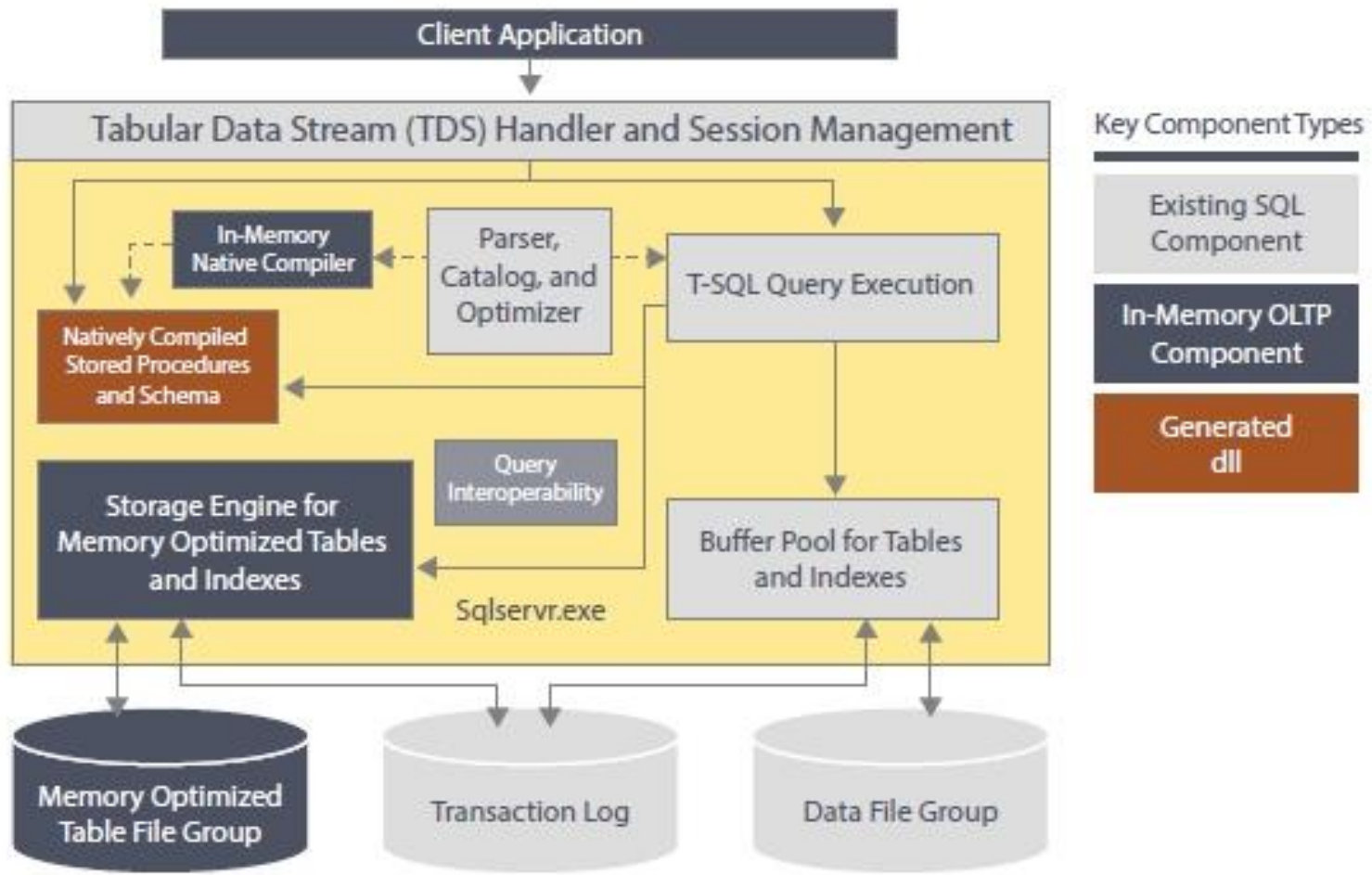
Hekaton

- Aimed to be 100 times faster with
 - In-Memory Data
 - Natively Compiled Code
- Work started 2010
- Initial Release SQL 2014

RAM prices over the last 40 Years

RAM - Log10(\$ per GB)

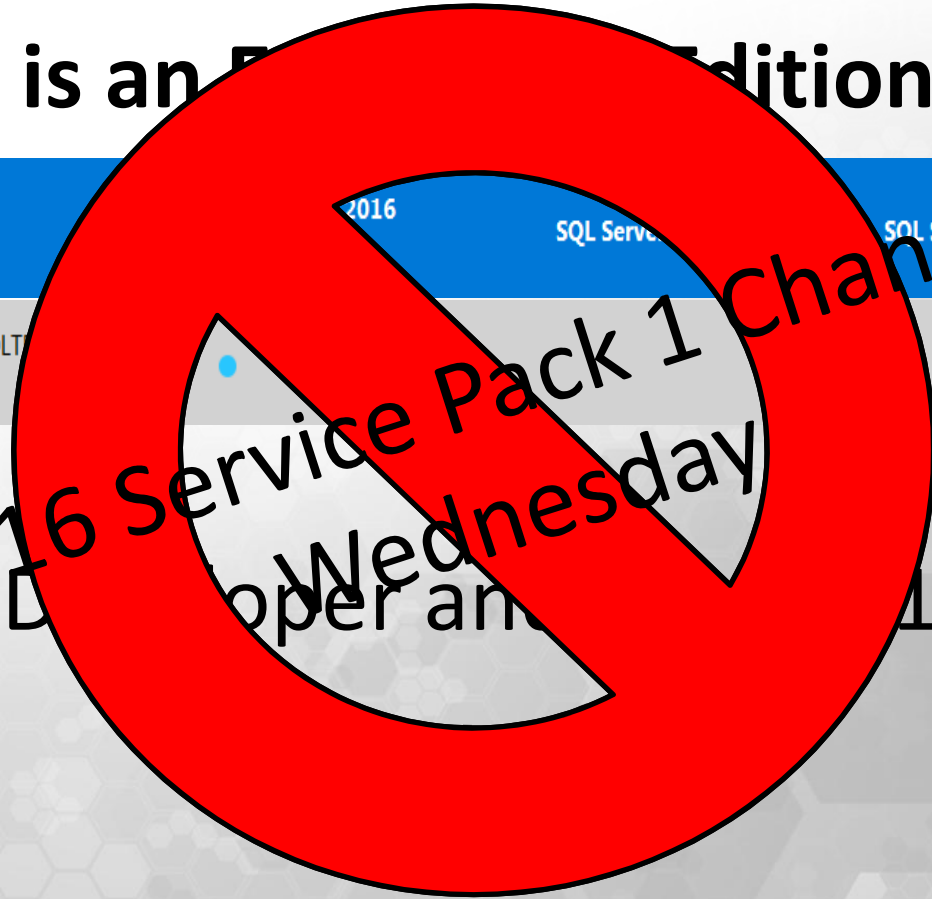




This is an Enterprise Edition feature

SQL Server 2016	SQL Server 2016 Express	SQL Server 2016 Developer
Advanced OLTP: In-memory OLTP analytics		

Also in Developer and SQL Server 2012 Premium



This is an Standard Edition Feature

	SQL Server 2016 Enterprise	SQL Server 2016 Standard	SQL Server 2016 Express	SQL Server 2016 Developer
Advanced OLTP: In-memory OLTP, operational analytics	•			•
Advanced OLTP: in-memory OLTP, operational analytics ²	•	•	•	•

Many Features Moved to Standard Edition

Feature	RTM				SP1			
	Standard	Web	Express	Local DB	Standard	Web	Express	Local DB
Row-level security	Yes	No	No	No	Yes	Yes	Yes	Yes
Dynamic Data Masking	Yes	No	No	No	Yes	Yes	Yes	Yes
Change data capture*	No	No	No	No	Yes	Yes	No*	No*
Database snapshot	No	No	No	No	Yes	Yes	Yes	Yes
Columnstore	No	No	No	No	Yes	Yes	Yes	Yes
Partitioning	No	No	No	No	Yes	Yes	Yes	Yes
Compression	No	No	No	No	Yes	Yes	Yes	Yes
In Memory OLTP	No	No	No	No	Yes	Yes	Yes	No**
Always Encrypted	No	No	No	No	Yes	Yes	Yes	Yes
PolyBase	No	No	No	No	Yes	Yes	Yes	No
Fine grained auditing	No	No	No	No	Yes	Yes	Yes	Yes
Multiple filestream containers	No	No	No	No	Yes	Yes	Yes	Yes

In-Memory Tables – CREATE TABLE 2016

```
CREATE TABLE Product_inmem(  
    ProdID int IDENTITY(1,1) NOT NULL,  
    Name nvarchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,  
    Color nvarchar(15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,  
    ModifiedDate datetime2(7) NOT NULL  
        CONSTRAINT DF_Product_ModifiedDate DEFAULT (sysdatetime()),  
    Notes nvarchar(max) NULL,  
        CONSTRAINT IMPK_Product_ProdID PRIMARY KEY NONCLUSTERED HASH  
            (ProdID) WITH (BUCKET_COUNT = 1048576),  
  
    INDEX IX_Name NONCLUSTERED (Name ASC),  
  
    )WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA )
```

In-Memory Tables – What's the Difference 1

- **A**tomaticity Yes! Transactions are all or nothing
- **C**onsistency Yes! The database is always consistent
- **I**solation Yes! Concurrent transactions isolated
- **D**urability Yes! But Durability is OPTIONAL!
- Tables are in Memory – No reading the disk!
- Indexes – Up to 8 – In memory only
- Data Files - Checkpoint Files – Append only

In-Memory Tables – What's the Difference 2

- No Locks
- No Latches
- No Spinlocks
- Achieved using a newer CPU Instruction
interlocked-compare and exchange

In-Mem Tables – Limitations - 2014

- No LOB – Varchar(MAX), NVARCHAR(MAX)
 - Text, ntext, image, varbinary(max)
- No computed columns
- No CHECK, Unique or Foreign Key constraints
- No Triggers
- No Replication
- Only Bin2 character cols in in indexes

In-Mem Tables – Limitations - 2016

- No LOB – ~~Varchar(MAX), NVARCHAR(MAX)~~
– Text, ntext, image, ~~varbinary(max)~~
- No computed columns
- ~~No CHECK, Unique or Foreign Key constraints~~
- ~~No Triggers Only AFTER~~
- No Replication
- ~~Only Bin2 character cols in in indexes~~

In-Memory Tables – 2016 Changes

- ALTER
- Row sizes > 8060 Bytes
- BLOB types VARCHAR(MAX), etc.
- Constraints: Foreign Key and CHECK
- Triggers – Natively Compiled Only
- Auto-Update Statistics
- Indexes on Non-BIN Character Columns

In-Memory Tables – What's Not Available

- XML types
- CLR types
- Legacy LOB (Text, Ntext, Image)

In-Memory Tables 2016 – Scalability

- 2 TB of In-Memory tables per database
- 1.2 Million Transactions/Sec
- 900 MB per Second Written

In-Memory Table Usability

- Interop
 - Use In-Memory tables with standard T-SQL
- Natively Compiled T-SQL
 - Can only use In-Memory Tables

In-Memory Tables - Durability

- SCHEMA_AND_DATA (PK required)
- SCHEMA_ONLY (Index required)
- DELAYED DURABILITY

In-Memory – Concurrency Control

- Optimistic Locking
- Multi-version Concurrency Control – MVCC
- Isolation Levels
 - SNAPSHOT
 - REPEATABLE READ
 - SERIALIZABLE

First Two In-Memory Use Cases

- Replace @temp tables
 - Scales (rows) better than @temp tables
- Replace #temp tables
 - Faster than #temp tables

In-Memory Table Types

```
CREATE TYPE dbo.EAV_tt_inmem AS TABLE (  
    [entity_id] int NOT NULL,  
    attribute_id smallint NOT NULL ,  
    [value] VARCHAR(256) NOT NULL,  
    INDEX IX_entity_attribute  
        NONCLUSTERED ([entity_id], attribute_id)  
    ) WITH (MEMORY_OPTIMIZED = ON)
```

```
DECLARE @eav_temp dbo.EAV_tt_inmem;
```


Second In-Memory Use-Case

- Replace #temp tables with in-memory tables
- Difference is the scope
- Add session_id column
- Use Row Level Security to limit access

In-Memory – Temp Table Replacement 1

```
CREATE TABLE EAV_temp_inmem (
    [entity_id] int NOT NULL,
    attribute_id smallint NOT NULL ,
    value VARCHAR(256) NOT NULL,
    session_id smallint NOT NULL DEFAULT (@@SPID),
INDEX IL_session_id (session_id),
    INDEX IX_entity_attribute
        NONCLUSTERED (session_id, [entity_id], attribute_id),
    CONSTRAINT CHK_temp1_session_id
        CHECK ( session_id = @@SPID )
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY=SCHEMA_ONLY);
```

In-Memory – Temp Table Replacement 2

```
CREATE FUNCTION dbo.fn_SessionFilter(@session_id smallint)
    RETURNS TABLE
    WITH SCHEMABINDING, NATIVE_COMPILATION
    AS
        RETURN SELECT 1 as fn_SessionFilter
                WHERE @session_id=@@spid;
```

```
CREATE SECURITY POLICY dbo.temp1Filter
    ADD FILTER PREDICATE dbo.fn_SessionFilter(session_id)
    ON dbo.EAV_temp_inmem
    WITH (STATE = ON);
```

NATIVELY COMPILED T-SQL

Natively Compiled Code

- Stored Procedures
- Scalar Functions
- After Triggers

- T-SQL with Limits
 - In-Memory Data only
 - Reduced T-SQL Syntax

Why is it faster

- T-SQL -> C C-> machine code
- Query Plans are compiled in
 - sp_recompile to get new query plans

CREATE PROCEDURE

```
CREATE PROC dbo.myProc (@myStr VARCHAR(64) NOT NULL)
    WITH NATIVE_COMPILATION, SCHEMABINDING
AS BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT
    language = N'English')

    -- BODY GOES HERE

END
```

- WITH NATIVE_COMPILATION
- SCHEMABINDING
- BEGIN ATOMIC

```
CREATE PROC dbo.myProc (@myStr VARCHAR(64) NOT NULL)
    WITH NATIVE_COMPILATION, SCHEMABINDING
AS BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT,
    language = N'English')

DECLARE @eav_temp dbo.EAV_tt_inmem;

insert dbo.EAV_temp_inmem (entity_id, attribute_id, value)
    OUTPUT inserted.entity_id,inserted.attribute_id,inserted.[value]
    into @eav_temp ([entity_id], attribute_id, [value])
    values (1,2, 'abcd')







SELECT [entity_id], attribute_id, [value] FROM @eav_temp

END
```


TAKE A LOOK AT THE C CODE

Files that get produced

- Stored in the XTP directory
 - ...\\MSSQL13.instance\Data\XTP\db_id()
- 6 Files per object

Name	Date modified	Type	Size
 xtp_t_9_581577110_182639204430509	6/5/2016 2:00 PM	C Source	5 KB
 xtp_t_9_581577110_182639204430509.dll	6/5/2016 2:00 PM	Application extension	73 KB
 xtp_t_9_581577110_182639204430509	6/5/2016 2:00 PM	Object File	82 KB
 xtp_t_9_581577110_182639204430509.out	6/5/2016 2:00 PM	OUT File	1 KB
 xtp_t_9_581577110_182639204430509	6/5/2016 2:00 PM	Program Debug Database	780 KB
 xtp_t_9_581577110_182639204430509	6/5/2016 2:00 PM	XML File	2 KB

BENCHMARK



Adventure Works
Bicycles
Corporation

Chapter 7 Bankruptcy

Wide World Importers – The new sample

- Located on GitHub
- .bak files or construct from scripts
- WideWorldOLTP
- WideWorldDW
- InMemDB (From script only)

Bechmark 1 - VehicleLocation

- Azure VM – 2 Core , 14 GB RAM
- Insert 500,000 Rows
- Single Threaded

	insert into disk-based table (in ms)
1	159725

	insert into memory-optimized table (in ms)
1	131741

	insert into memory-optimized table using native compilation (in ms)
1	1859

Benchmark Magic

- Use Binary Collations
- Fixed # of detail lines instead of table
- Compare to bad code
- No error handling

STORED PROCEDURE STRUCTURE


```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name
    [ { @parameter data_type } [ NULL | NOT NULL ] [ = default ]
      [ OUT | OUTPUT ] [READONLY] ]
    [ ,... n ]
    WITH NATIVE_COMPILATION, SCHEMABINDING
    [ , EXECUTE AS clause ]
AS
{
    BEGIN ATOMIC WITH (set_option [ ,... n ] )

        sql_statement [;] [ ... n ]
    [ END ]
} [;]
```

NOT NULL Parameters and Variables

- Parameters can be NOT NULL

```
CREATE PROC myProc (@myStr VARCHAR(64) NOT NULL)
```

- Variables can be NOT NULL

```
DECLARE @myStr VARCHAR(64) NOT NULL = 'Navigabimus Nunc'
```

- Assignment of NULL does NOT fail

```
SET @myStr = NULL; -- Compiles but Fails at runtime
```

BEGIN ATOMIC - SET OPTIONS

```
LANGUAGE = [ N ] 'language'
```

```
TRANSACTION ISOLATION LEVEL =  
    { SNAPSHOT | REPEATABLE READ | SERIALIZABLE }
```

```
[ DATEFIRST = number ]  
[ DATEFORMAT = format ]  
[ DELAYED_DURABILITY = { OFF | ON } ]
```

BEGIN ATOMIC Required Settings

- TRANSACTION ISOLATION LEVEL
 - SNAPSHOT
 - REPEATABLE READ
 - SERIALIZABLE
- LANGUAGE
 - All languages in sys.syslanguages

BEGIN ATOMIC Optional Settings

- DATEFORMAT = number
- DATEFIRST = format
- DELAYED_DURABILITY = OFF | ON

BEGIN ATOMIC standard settings (1 of 2)

SET OPTION	System Default for ATOMIC Blocks
ANSI_NULLS	ON
ANSI_PADDING	ON
ANSI_WARNING	ON
ARITHABORT	ON
ARITHIGNORE	OFF
CONCAT_NULL_YIELDS_NULL	ON
IDENTITY_INSERT	OFF
NOCOUNT	ON

BEGIN ATOMIC standard settings (2 of 2)

SET OPTION	System Default for ATOMIC Blocks
NUMERIC_ROUNDABORT	OFF
QUOTED_IDENTIFIER	ON
ROWCOUNT	0
TEXTSIZE	0
XACT_ABORT	OFF
CONCAT_NULL_YIELDS_NULL	ON

Natively Compiled T-SQL – Added to 2016

- LEFT and RIGHT OUTER JOIN
- SELECT DISTINCT
- OR and NOT operators
- Subqueries in all clauses of a SELECT statement
- Nested stored procedure calls
- UNION and UNION ALL
- All built-in math functions
- Some security functions, including @@spid
- Scalar user-defined functions
- EXECUTE AS CALLER

Error Handling

- TRY CATCH and THROW
- No RAISERROR – use THROW
- Try and have just one TRY CATCH

Natively Compiled T-SQL – Still Missing in 2016

- CASE expressions
- MERGE
- TOP with ORDER BY Limited to 8192 rows
- TOP WITH TIES or PERCENT
- INSERT VALUES with multiple rows
- Most hints
 - LOOP JOIN is supported

Forbidden in Natively Compiled Procedures

- Cross Container Access – Disk Based Tables
- Dynamic SQL NO EXEC (“) or sp_executeSQL
- Cross database transactions
- Distributed Transactions

Parameter Passing - Best Practices

- Pass by position
- Pass exact data type
- Hekaton_slow_parameter_passing EE

```
CREATE EVENT SESSION [Natively Compiled Stored Procedures] ON SERVER
ADD EVENT sqlserver.natively_compiled_proc_slow_parameter_passing,
ADD EVENT sqlserver.natively_compiled_proc_execution_started
ADD TARGET package0.event_file(SET filename=N'Natively Compiled Stored
Procedures'),
ADD TARGET package0.ring_buffer(SET max_memory=(102400))
WITH ( MAX_MEMORY=4096
KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,
MAX_DISPATCH_LATENCY=30 SECONDS,MAX_EVENT_SIZE=0 KB,
MEMORY_PARTITION_MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
GO
--Start Event Session
ALTER EVENT SESSION [Natively Compiled Stored Procedures] ON SERVER STATE =
START
```

FUNCTIONS

Types

- Scalar
- No others

Exponentially Weighted Moving Average

$$EMA_{Today} = (Value_{Today} * \left(\frac{Smoothing}{1 + Days}\right)) \\ + EMA_{Yesterday} * \left(1 - \left(\frac{Smoothing}{1 + Days}\right)\right)$$

Traditional Scalar T-SQL UDF

```
CREATE FUNCTION [dbo].[exp_moving_avg_scalar] (  
    @ema_yesterday FLOAT  
    , @value_today FLOAT  
    , @days INT  
    , @smoothing FLOAT = 2.0  
  
    ) RETURNS FLOAT  
AS BEGIN  
RETURN CASE WHEN @ema_yesterday IS NULL  
    THEN @value_today  
    ELSE (@value_today * (@smoothing/(1.0+@days)))  
        + (@ema_yesterday  
            * (1.0 - (@smoothing/(1.0+@days))))  
    END  
END
```

Natively Compiled Scalar Function

```
CREATE FUNCTION [dbo].[exp_moving_avg_scalar_nc] (  
    @ema_yesterday FLOAT  
    , @value_today FLOAT  
    , @days INT  
    , @smoothing FLOAT = 2.0  
) RETURNS FLOAT  
WITH NATIVE_COMPILATION, SCHEMABINDING, CALLED ON NULL INPUT  
AS BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, language=N'English')  
  
IF @ema_yesterday IS NULL RETURN @value_today  
  
RETURN (@value_today * (@smoothing/(1.0+@days)))  
        + (@ema_yesterday  
           * (1.0 - (@smoothing/(1.0+@days))))  
  
END
```

Options Unique to Scalar Functions

- RETURNS NULL ON NULL INPUT
- CALLED ON NULL INPUT

Interpreted Inline T-SQL UDF

```
CREATE FUNCTION [dbo].[exp_moving_avg_inline] (  
    @ema_yesterday FLOAT  
    , @value_today FLOAT  
    , @days INT  
    , @smoothing FLOAT = 2.0  
) RETURNS TABLE  
AS RETURN  
SELECT CASE WHEN @ema_yesterday IS NULL  
    THEN @value_today  
    ELSE (@value_today * (@smoothing/(1.0+@days)))  
        + (@ema_yesterday  
            * (1.0 - (@smoothing/(1.0+@days))))  
    END  
as ema_today
```

FUNCTIONS DEMO

References

- SQL Server Blog
<https://blogs.msdn.microsoft.com/sqlserverstorageengine/>
- Kalen Delaney
SQL Server In-Memory OLTP Internals for SQL Server 2014
Microsoft June 2016
- Dmitri Korotkevitch
Expert SQL Server In-Memory OLTP Revolutionizing OLTP Performance in SQL Server, APress, New York, 2015 (SQL 2014 book)



anovick@NovickSoftware.com
<http://www.NovickSoftware.com>

Thank you for coming!



Database option

- MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT
- When Txn is READ Committed or read Uncommitted

CROSS CONTAINER TRANSACTIONS

- Access in-memory tables in a TRAN
- Execute a native proc when a TRAN is open

Natively Compiled SP - Dynamic SQL

- No EXEC (“)
- No sp_executesql
- Can be created by Dynamic SQL

Parameter Passing

- Try and match data types
- Use positional parameters instead of @named parms
- Extended events will show where