Microsoft

SQL Server 2016 and 2014

In-Memory OLTP and Data Warehousing

George Walters

Senior Technical Specialist, Northeast US EPG

george.walters@microsoft.com

@gwalters69

SQL Server 2016: Everything built-in

Industry leader in mission critical OLTP

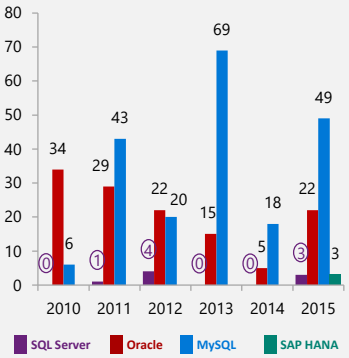
built-in

Industry leader

Most secure database

built-in

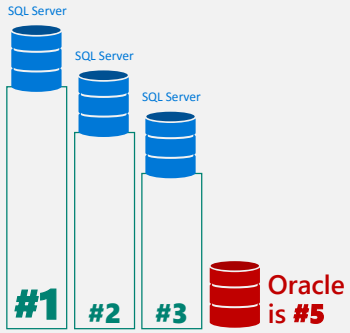
6 years in a row least vulnerable



Highest performing data warehouse

built-in

#1 performance

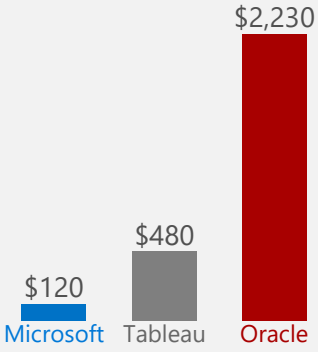


TPC-H

End-to-end mobile BI on any device

built-in

A fraction of the cost

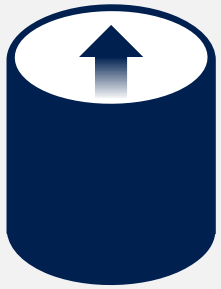


Self-service BI per user

In-database Advanced Analytics

built-in

R + in-memory



at massive scale

In-memory across all workloads

Consistent experience from on-premises to cloud

The above graphics were published by Gartner, Inc. as part of a larger research document and should be evaluated in the context of the entire document. The Gartner document is available upon request from Microsoft. Gartner does not endorse any vendor, product or service depicted in its research publications, and does not advise technology users to select only those vendors with the highest ratings or other designation. Gartner research publications consist of the opinions of Gartner's research organization and should not be construed as statements of fact. Gartner disclaims all warranties, expressed or implied, with respect to this research, including any warranties of merchantability or fitness for a particular purpose.

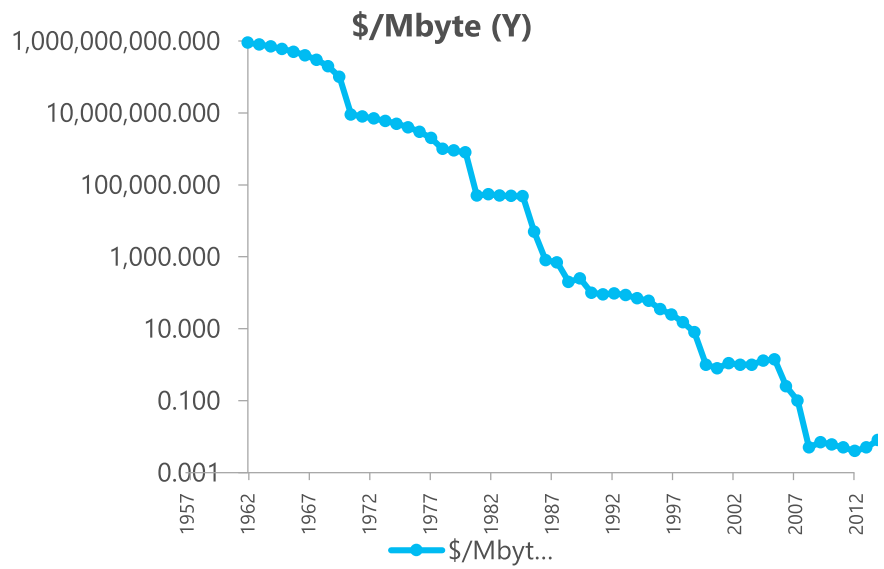
National Institute of Standards and Technology Comprehensive Vulnerability Database update 10/2015

TPC-H non-clustered results as of 04/06/15, 5/04/15, 4/15/14 and 11/25/13, respectively. http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster

Microsoft

Key trends for In-Memory

In-memory now a viable solution



$1.1 trillion

Enabled by cloud innovation

"By 2015, business revenues from IT innovation enabled by the cloud could reach US$1.1 trillion a year."
IDC

Data Explosion

85% from new data types

10x increase every 5 years

In-Memory built-in to SQL 2014

Built-in	On average 10x faster, without having rewrite entire app Leverage full SQL Server capabilities
Flexible	Select only highly utilized tables to be in-memory Optimize in-memory to fit existing hardware
Spans all workloads	In-memory performance across OLTP, DW, and BI All in a single SKU

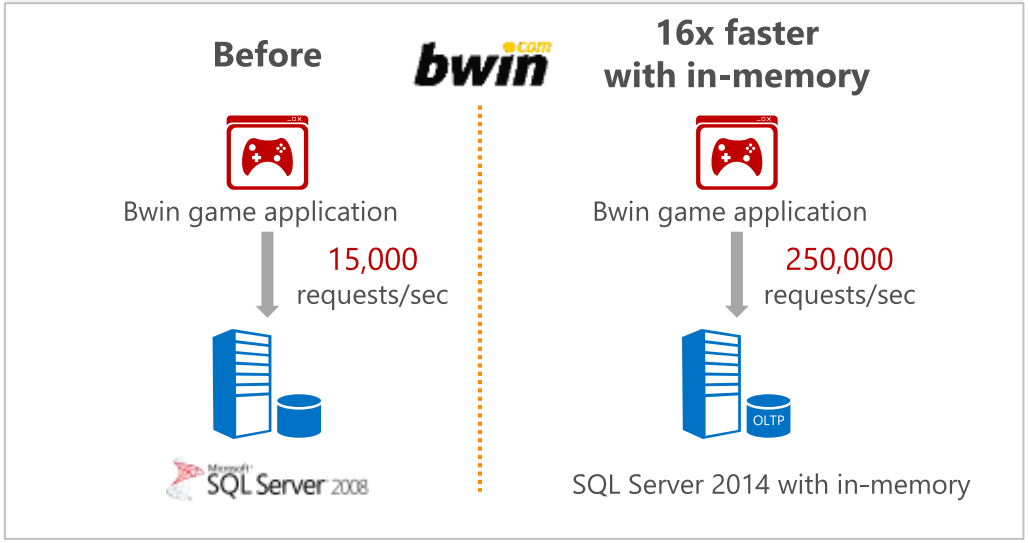
Key features

New in-memory OLTP

Enhanced in-memory ColumnStore for DW

In-memory BI with PowerPivot

Buffer pool extension to SSDs and enhanced query processing



In-Memory OLTP

Benefits

Low latency

Up to 30 times the improvement in performance

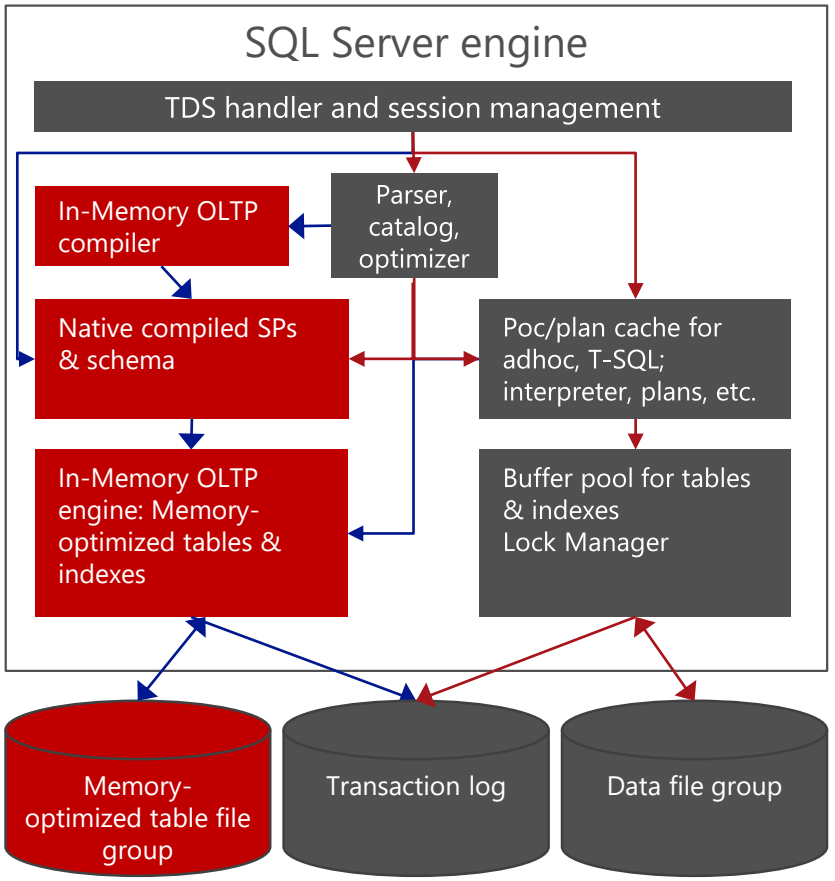
2 to 5 times the improvement in scalability

Takes advantage of investments in Microsoft SQL Server

How it works

New high-performance, memory-optimized online transaction processing (OLTP) engine integrated into SQL Server and architected for modern hardware trends

- Integrated into SQL Server relational database
- Full ACID support
- Memory-optimized
- Non blocking multi-version optimistic concurrency control (no locks or latches)
- T-SQL compiled to native code



Design Considerations For Memory-optimized Tables

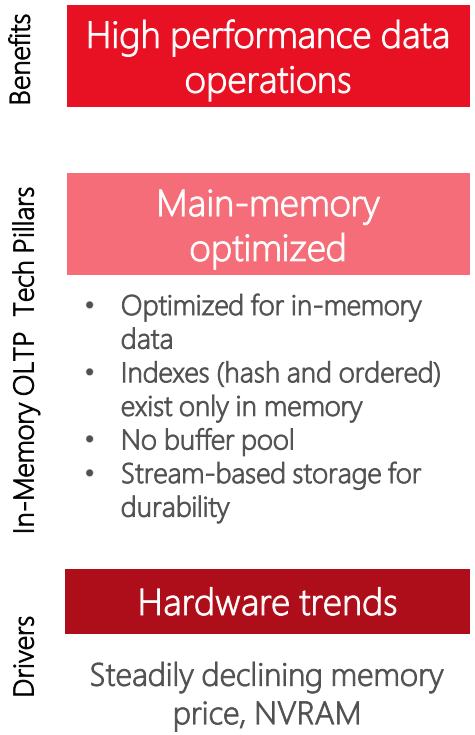


Table constructs

Fixed schema; no ALTER TABLE; must drop/recreate/reload
No LOB data types; row size limited to 8,060
No constraints support (primary key only)
No identity or calculated columns, or CLR

Data and table size considerations

Size of tables = (row size * # of rows)
Size of hash index = (bucket_count * 8 bytes)
Max size SCHEMA_AND_DATA = 512 GB

IO for durability

SCHEMA_ONLY vs. SCHEMA_AND_DATA
Memory-optimized filegroup
Data and delta files
Transaction log
Database recovery

Create Table DDL

```
CREATE TABLE [Customer](
    [CustomerID] INT NOT NULL
            PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1000000),
    [Name] NVARCHAR(250) NOT NULL
      INDEX [IName] HASH WITH (BUCKET_COUNT = 1000000),
    [CustomerSince] DATETIME NULL
)
WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA);
```

Hash index

Secondary indexes are specified inline

This table is memory optimized

This table is durable

Create Procedure DDL

```
CREATE PROCEDURE [dbo].[InsertOrder] @id INT, @date DATETIME
  WITH
    NATIVE_COMPILATION,
    SCHEMABINDING,
    EXECUTE AS OWNER
AS
BEGIN ATOMIC
  WITH
(TRANSACTION
   ISOLATION LEVEL = SNAPSHOT,
 LANGUAGE = 'us_english')

  -- insert T-SQL here
END
```

This proc is natively compiled

Native procs must be schema-bound

Execution context is required

Atomic blocks
- Create a transaction if there is none
- Otherwise, create a savepoint

Session settings are fixed at create time

In-Memory OLTP summary

What's being delivered

High-performance, memory-optimized OLTP engine integrated into SQL Server and architected for modern hardware trends

Main benefits

- Optimized for in-memory data up to 20–30 times throughput
  - Indexes (hash and range) exist only in memory; no buffer pool, B-trees
  - T-SQL compiled to machine code via C code generator and Visual C compiler
  - Core engine uses lock-free algorithms; no lock manager, latches, or spinlocks
- Multiversion optimistic concurrency control with full ACID support
- On-ramp existing applications
- Integrated experience with same manageability, administration, and development experience

DEMO!

- Memory-Optimized Tables
  - Help solve many-threaded contention issues (latch contention)

In-memory OLTP enhancements

In-memory OLTP enhancements

```
ALTER TABLE Sales.SalesOrderDetail
   ALTER INDEX PK_SalesOrderID
   REBUILD
      WITH (BUCKET_COUNT=100000000)
```

```
T-SQL surface area: New

{LEFT|RIGHT} OUTER JOIN
Disjunction (OR, NOT)
UNION [ALL]
SELECT DISTINCT
Subqueries (EXISTS, IN, scalar)
```

ALTER support

Full schema change support: add/alter/drop column/constraint

Add/drop index supported

Surface area improvements

Almost full T-SQL coverage including scaler user-defined functions

Improved scaling

Increased size allowed for durable tables; more sockets

Other improvements

MARS support

Lightweight migration reports

Altering memory-optimized tables

```
| ADD
    {
        <column_definition>
      | <table_constraint>
      | <table_index>
    } [ ,...n ]

| DROP
     {
         [ CONSTRAINT ]
         {
             constraint_name
          } [ ,...n ]
          | COLUMN
          {
             column_name
          } [ ,...n ]
          | INDEX
          {
             index_name
          } [ ,...n ]
     } [ ,...n ]

| ALTER INDEX index_name
    {
REBUILD (WITH <rebuild_index_option>)
    }
}
```

The **ALTER TABLE** syntax is used for making changes to the table schema, as well as for adding, deleting, and rebuilding indexes

Indexes are considered part of the table definition

Key advantage is the ability to change the **BUCKET_COUNT** with an **ALTER INDEX** statement

Altering natively compiled stored procedures

```
CREATE PROCEDURE [dbo].[usp_1]
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS
OWNER
AS BEGIN ATOMIC WITH
(
 TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE
= N'us_english'
)
 SELECT c1, c2 from dbo.T1
END
GO

ALTER PROCEDURE [dbo].[usp_1]
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS
OWNER
AS BEGIN ATOMIC WITH
(
 TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE
= N'us_english'
)
 SELECT c1 from dbo.T1
END
GO
```

You can now perform **ALTER** operations on natively compiled stored procedures using the **ALTER PROCEDURE** statement

Use **sp_recompile** to recompile stored procedures on the next execution

Greater Transact-SQL coverage

- CREATE PROCEDURE (Transact-SQL)
- DROP PROCEDURE (Transact-SQL)
- ALTER PROCEDURE (Transact-SQL)
- SELECT (Transact-SQL) and INSERT SELECT statements
- SCHEMABINDING and BEGIN ATOMIC (required for natively compiled stored procedures)
- NATIVE_COMPILATION
- Parameters and variables can be declared as NOT NULL
- Table-valued parameters.
- EXECUTE AS OWNER, SELF, and user.
- GRANT and DENY permissions on tables and procedures.
- Nesting natively compiled stored procedures
- RIGHT OUTER JOIN, LEFT OUTER JOIN, INNER JOIN, and CROSS JOIN in SELECT statements
- NOT, OR, and IN operators in SELECT, UPDATE and DELETE statement
- UNION ALL and UNION
- SELECT DISTINCT
- GROUP BY clause with no aggregate functions in the SELECT clause (<select> list).
- COLUMNSTORE
- COLLATE

Improved scaling

Other enhancements include:

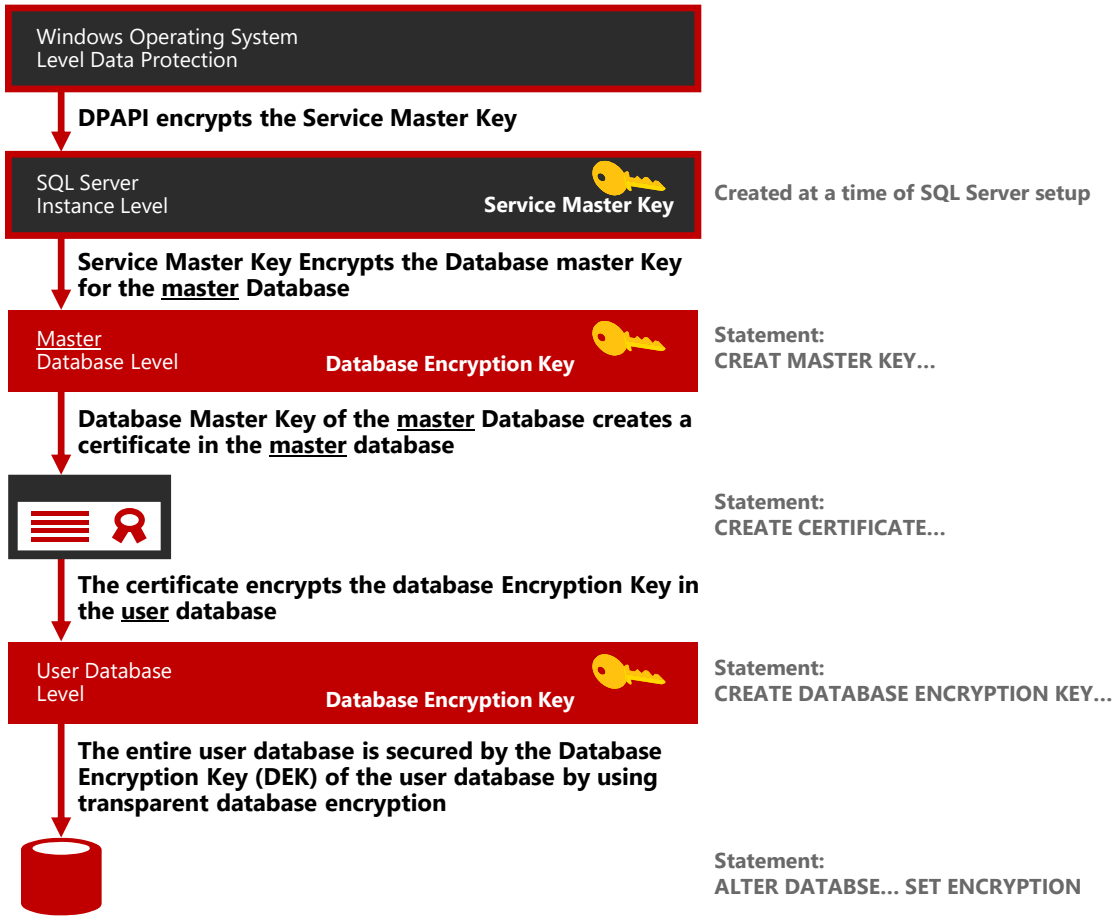
- Multiple threads to persist memory optimized tables
- Multi-threaded recovery
- MERGE operation
- Dynamic management view improvements to `sys.dm_db_xtp_checkpoint_stats` and `sys.dm_db_xtp_checkpoint_files`
- Using multiple active result sets (MARS)
  - Data Source=MSSQL; Initial Catalog=AdventSecurity=SSPIureWorks; Integrated; MultipleActiveResultSets=True

In-memory OLTP engine has been enhanced to scale linearly on servers up to 4 sockets

Setup MARS connection for memory optimized tables using the **`MultipleActiveResultsSets=True`** in your connection string

Support for Transparent Data Encryption

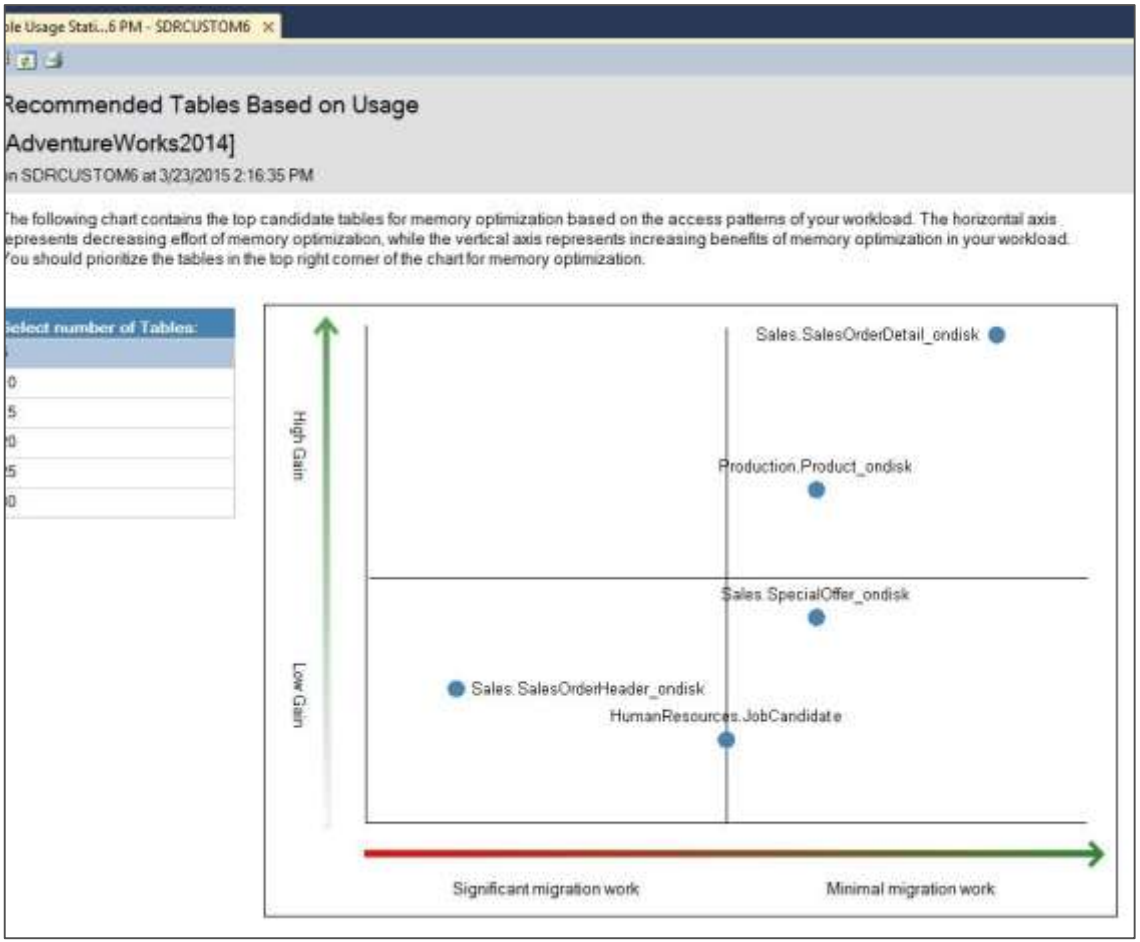
Transparent Database Encryption Architecture



In SQL Server 2016, the storage for memory-optimized tables will be encrypted as part of enabling TDE on the database

Simply follow the same steps as you would for a disk-based database

New Transaction Performance Analysis Overview report



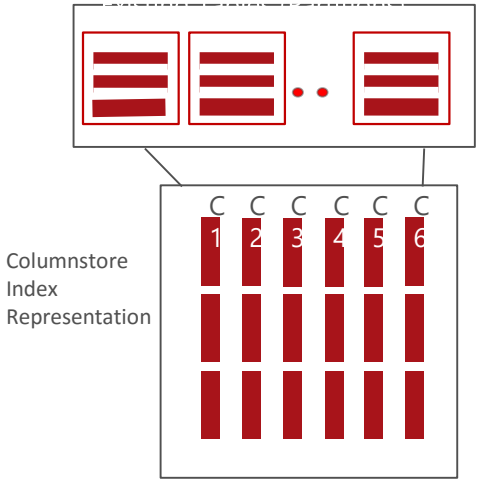
New report replaces the need to use the Management Data Warehouse to analyze which tables and stored procedures are candidates for in-memory optimization

In-Memory In the Data Warehouse

- In-Memory ColumnStore
- Both memory and disk
- Built-in to core RDBMS engine
- Customer Benefits:
  - 10-100x faster
  - Reduced design effort
  - Work on customers' existing hardware
  - Easy upgrade; Easy deployment

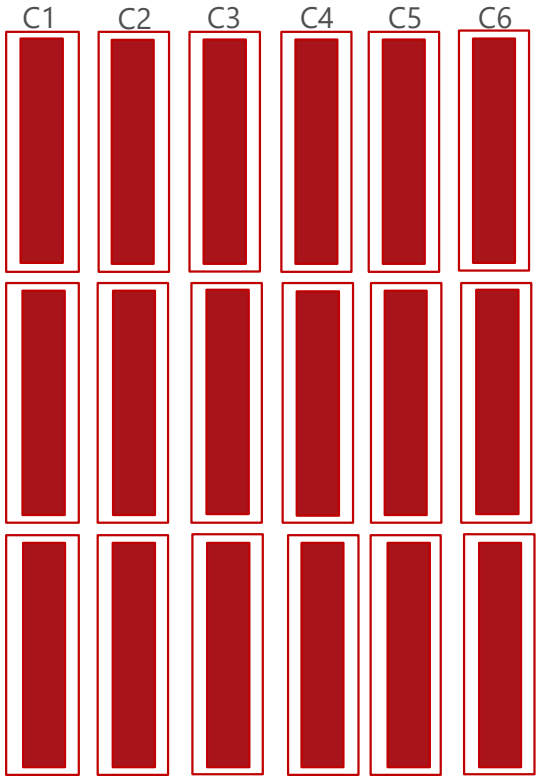
"By using SQL Server 2012 In-Memory ColumnStore, we were able to extract about 100 million records in **2 or 3 seconds** versus the **30 minutes required** previously. "

- Atsuo Nakajima Asst Director, Bank of Nagoya



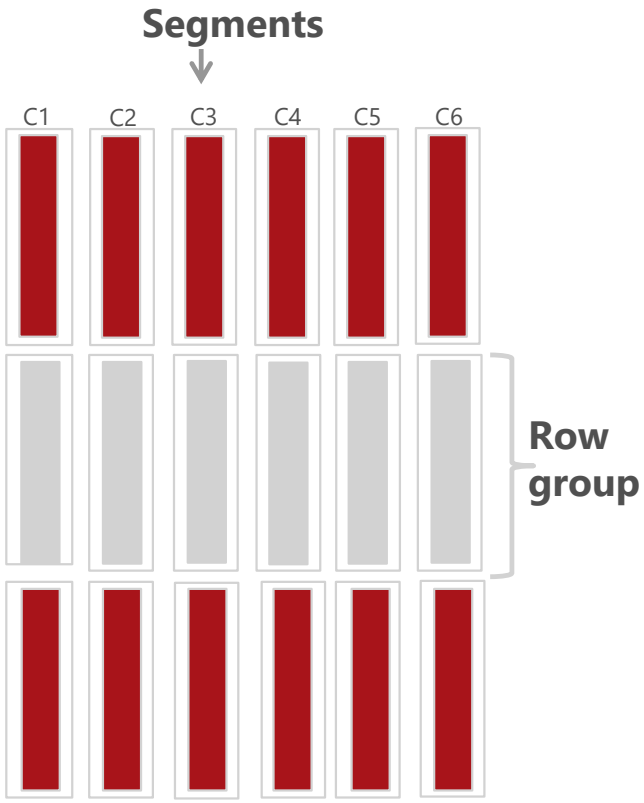
In-Memory DW Storage Model
Data Stored Column-wise

- Each page stores data from a single column
- Highly compressed
  - More data fits in memory
- Each column can be accessed independently
  - Fetch only columns needed
  - Can dramatically decrease I/O



In-Memory DW Index Structure
Row Groups & Segments

- A segment contains values for one column for a set of rows
- Segments for the same set of rows comprise a row group
- Segments are compressed
- Each segment stored in a separate LOB
- Segment is unit of transfer between disk and memory



In-Memory DW Index Processing an Example

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

Horizontally Partition Row Groups

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00

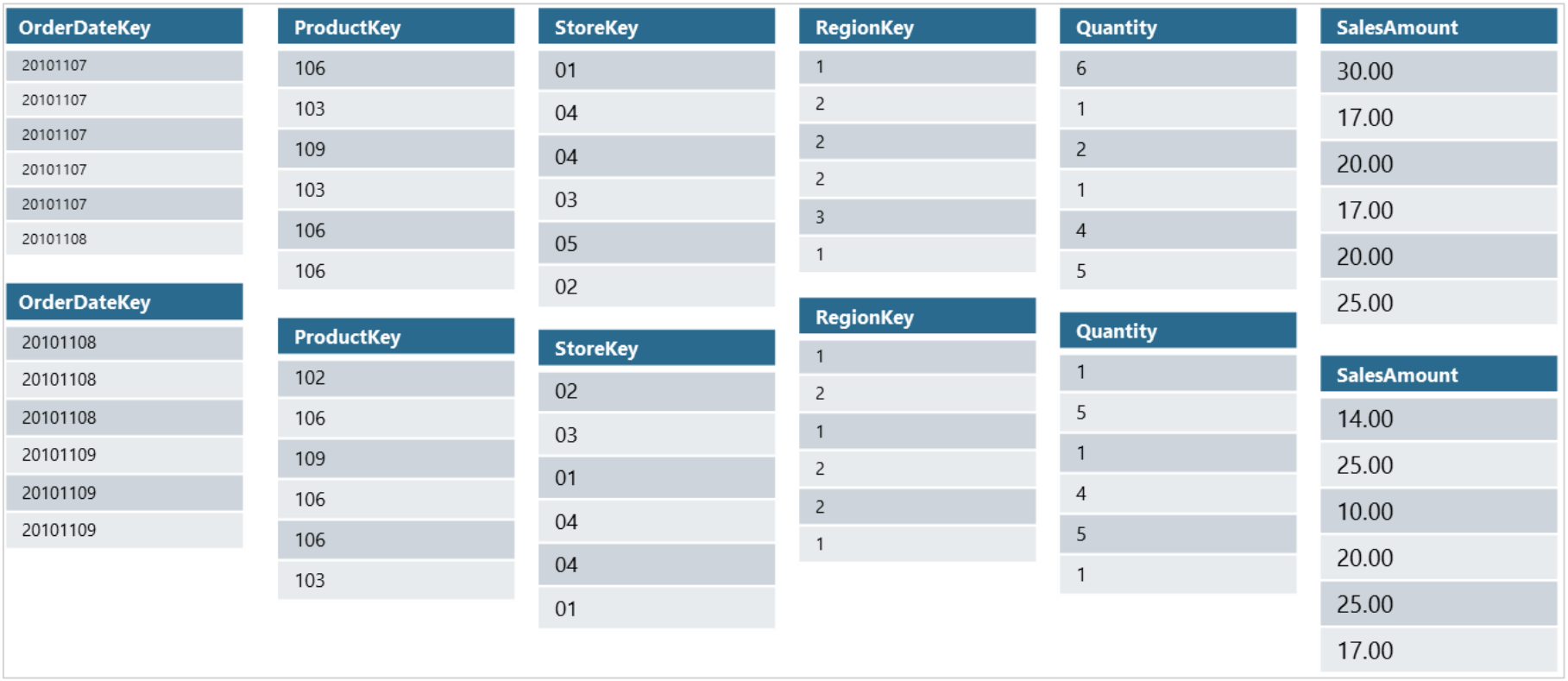
OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

Vertical Partition Segments

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

Compress Each Segment*
Some Compress More than Others



OrderDateKey
20101107
20101107
20101107
20101107
20101107
20101108

OrderDateKey
20101108
20101108
20101108
20101109
20101109
20101109

ProductKey
106
103
109
103
106
106

ProductKey
102
106
109
106
106
103

StoreKey
01
04
04
03
05
02

StoreKey
02
03
01
04
04
01

RegionKey
1
2
2
2
3
1

RegionKey
1
2
1
2
2
1

Quantity
6
1
2
1
4
5

Quantity
1
5
1
4
5
1

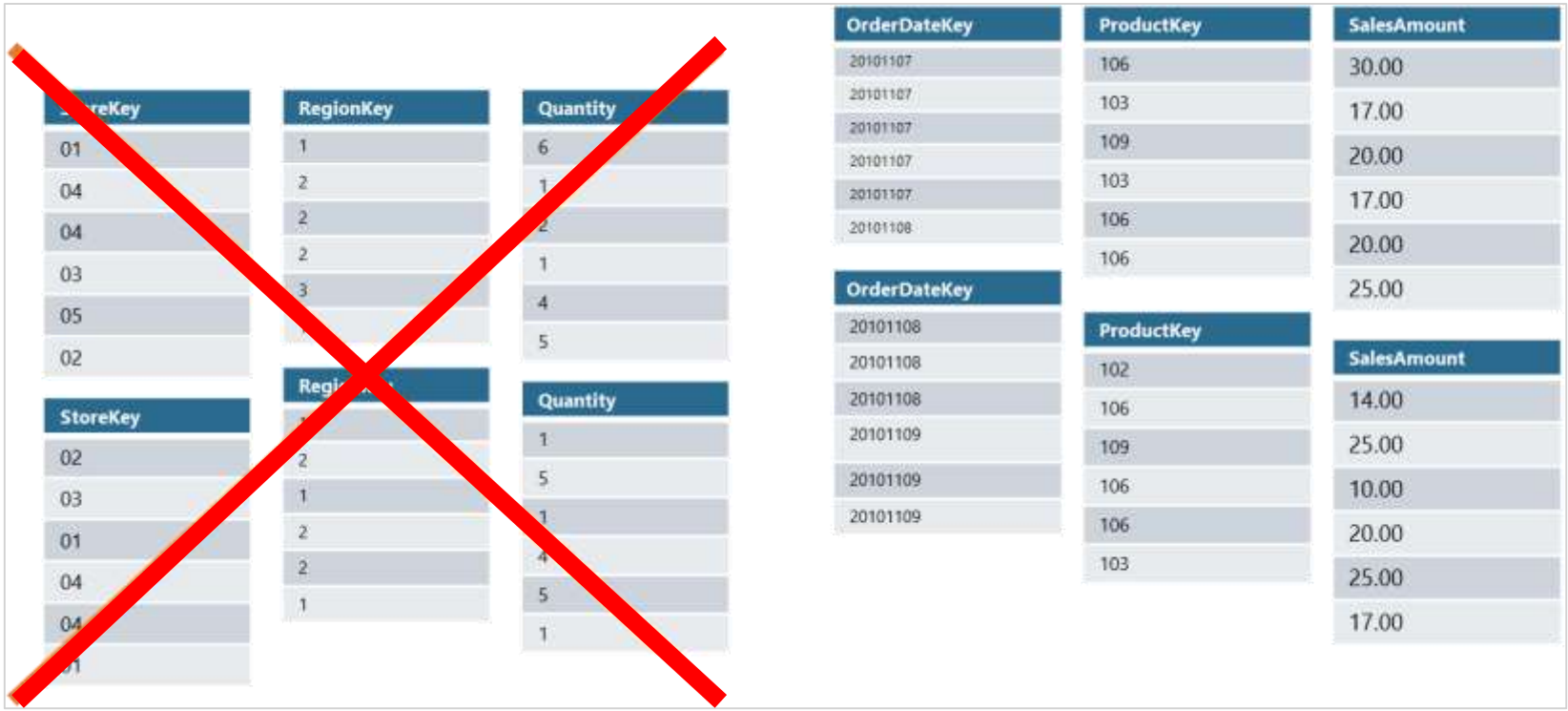
SalesAmount
30.00
17.00
20.00
17.00
20.00
25.00

SalesAmount
14.00
25.00
10.00
20.00
25.00
17.00

*Encoding and reordering not shown

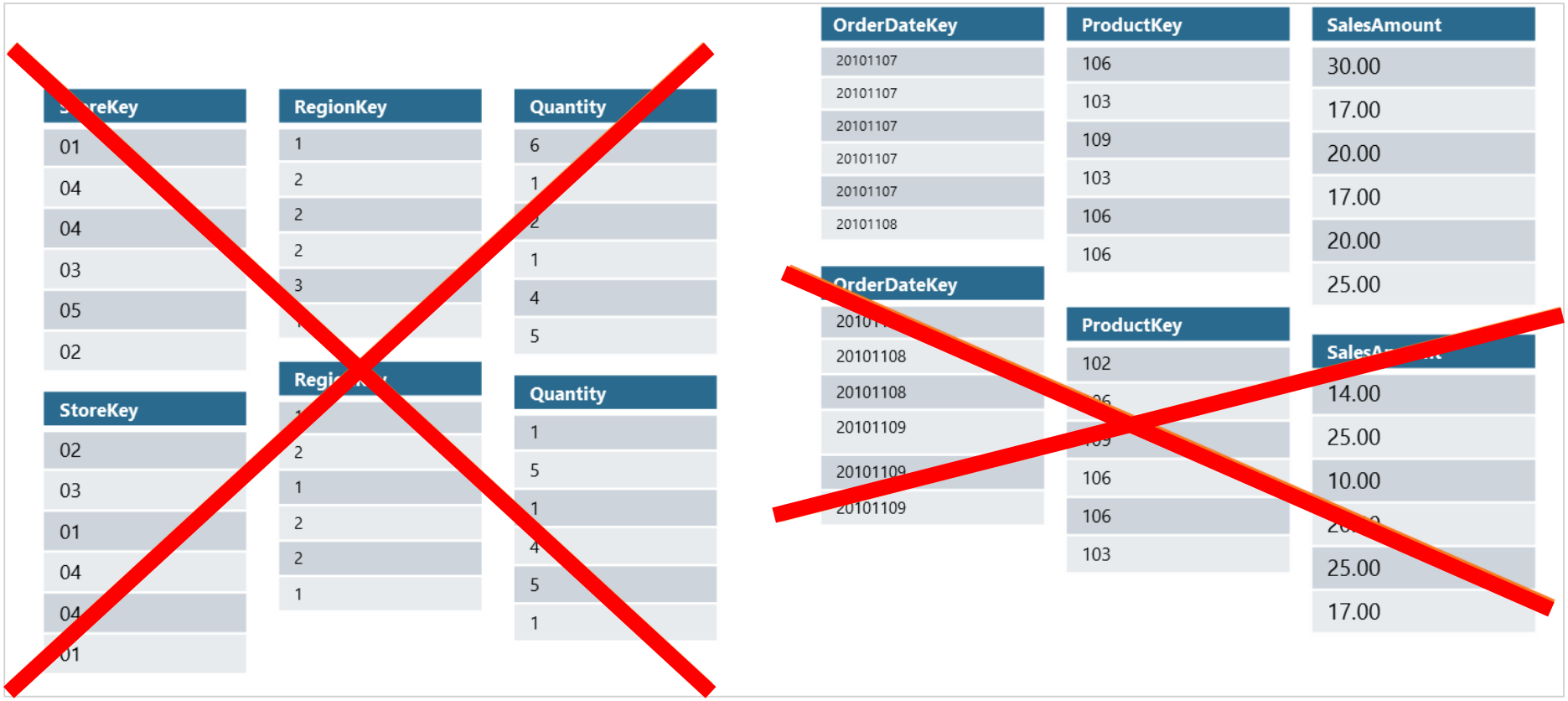
Fetch Only Needed Columns
Segment Elimination

```
SELECT ProductKey, SUM (SalesAmount)
FROM SalesTable
WHERE OrderDateKey < 20101108
```

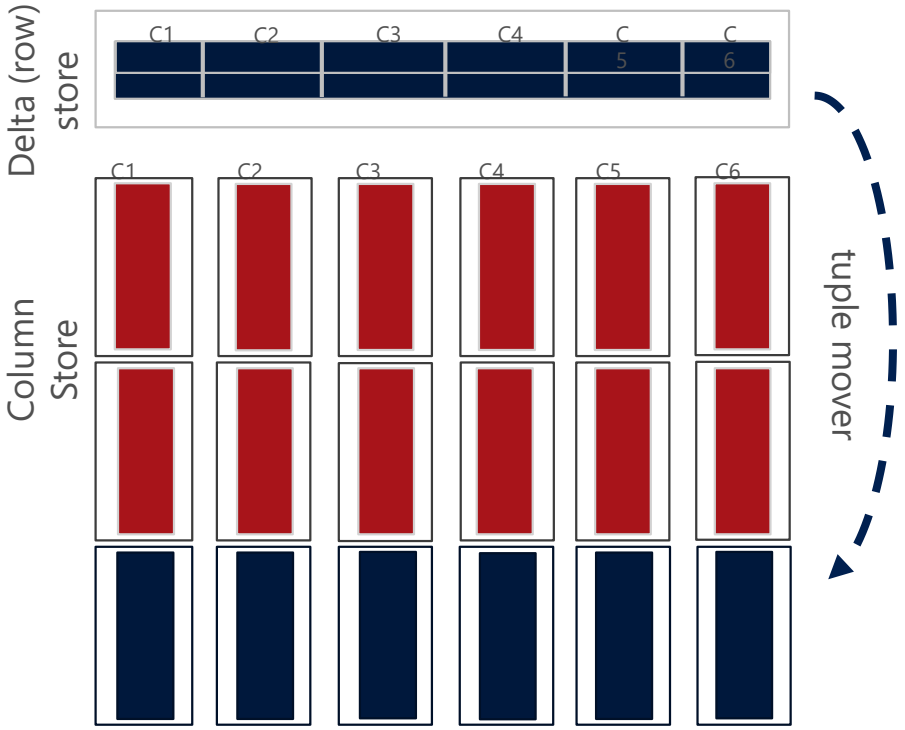


Fetch Only Needed Segments
Segment Elimination

```
SELECT ProductKey, SUM (SalesAmount)
FROM SalesTable
WHERE OrderDateKey < 20101108
```



Updatable Columnstore Index (2014 CCI, 2016 NCCI)

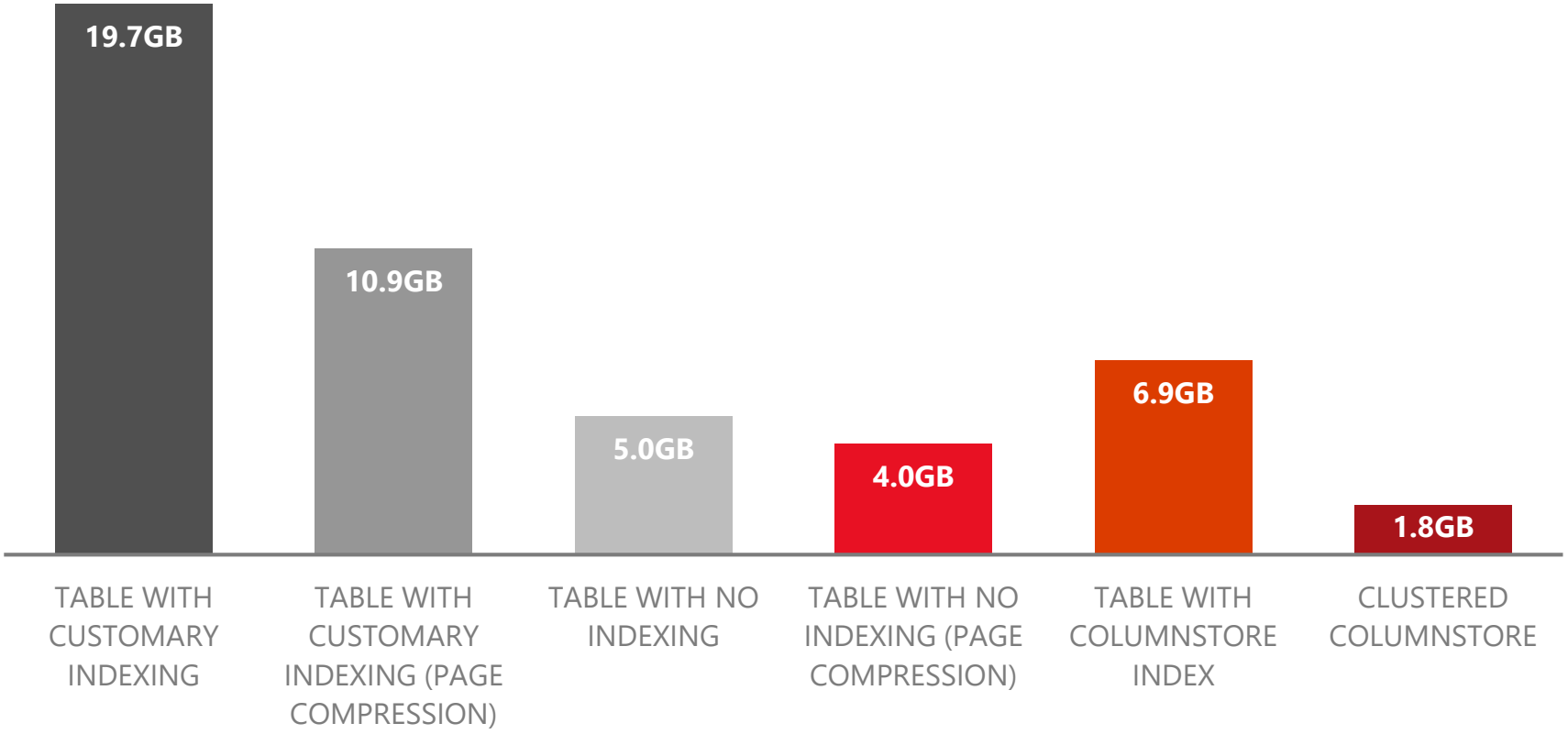


- Table consists of column store and row store
- DML (update, delete, insert) operations leverage delta store

- INSERT Values
  - Always lands into delta store
- DELETE
  - Logical operation
  - Data physically remove after REBUILD operation is performed.
- UPDATE
  - DELETE followed by INSERT.
- BULK INSERT
  - if batch < 100k, inserts go into delta store, otherwise columnstore
- SELECT
  - Unifies data from Column and Row stores - internal UNION operation.

- "Tuple mover" converts data into columnar format once segment is full (1M of rows)
- REORGANIZE statement forces tuple mover to start.

Comparing Space Savings
101 Million Row Table + Index Space

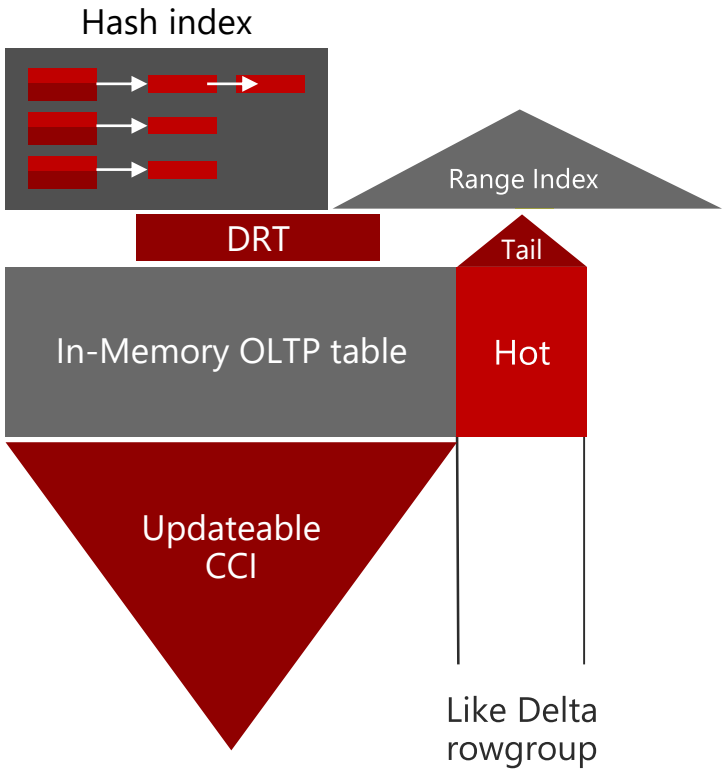


Columnstore enhancements summary

- What's being delivered
  - Clustered and updateable columnstore index
  - Columnstore archive option for data compression
  - Global batch aggregation
- Main benefits
  - Real-time super fast data warehouse engine
    - Ability to continue queries while updating without the need to drop and recreate index or partition switching
  - Huge disk space saving due to compression
    - Ability to compress data 5–15x using archival per-partition compression
  - Better performance and more efficient (less memory) batch query processing using batch mode rather than row mode
- SQL 2016
  - NonClustered updateable columnstore index
  - Filtered Columnstore Indexing

Operational analytics for in-memory tables

Operational analytics: Columnstore on in-memory tables



SQL Server 2016 – CTP2 limitation

You can create columnstore index on empty table
All columns must be included in the columnstore

No explicit delta rowgroup

Rows (tail) not in columnstore stay in in-memory OLTP table
No columnstore index overhead when operating on tail
Background task migrates rows from tail to columnstore in chunks of 1 million rows not changed in last 1 hour.

Deleted Rows Table (DRT) – Tracks deleted rows

Columnstore data fully resident in memory
Persisted together with operational data

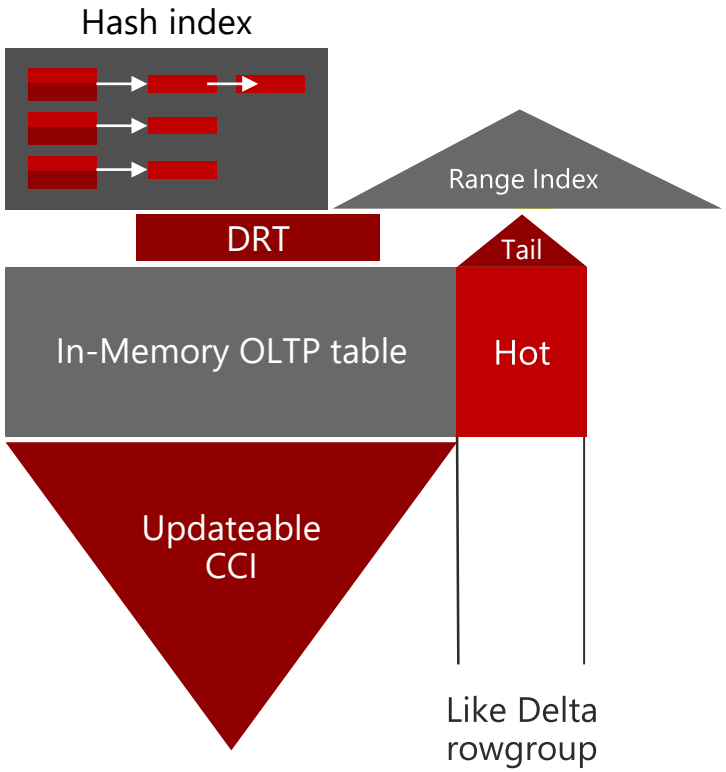
No application changes required.

Operational Analytics: Columnstore Overhead

DML operations on In-Memory OLTP

Operation	Hash or Range Index	HK-CCI
Insert	Insert row into HK	Insert row into HK
Delete	(a) Seek row(s) to be deleted (b) Delete the row	(a) Seek row(s) to be deleted (b) Delete the row in HK (c) If row in TAIL then return else insert <colstore-RID> into DRT
Update	(a) Seek the row(s) (b) Update (delete/insert)	(a) Seek the row(s) (b) Update (delete/insert) in HK (c) If row in TAIL then return else insert <colstore-RID> into DRT

Operational Analytics: Minimizing Columnstore overhead



DML Operations

Keep hot data only in-memory tables
Example – data stays hot for 1 day, 1 week…

CTP2: Work-Around
Use TF – 9975 to disable auto-compression
Force compression using a spec-proc
"sp_memory_optimized_cs_migration"

Analytics queries

Offload analytics to AlwaysOn Readable secondary

Summary: Operational analytics (2016)

Data Warehouse queries can be run on in-memory OLTP workload with no application changes.

These operations have minimal impact on OLTP workload.

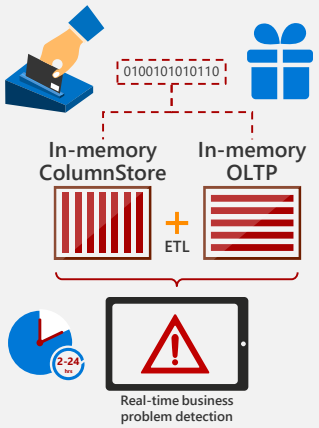
Best performance and scalability available

Offloading analytics workload to Readable Secondary

Minimizing impact on OLTP Example: for HOT/WARM (Predicate)

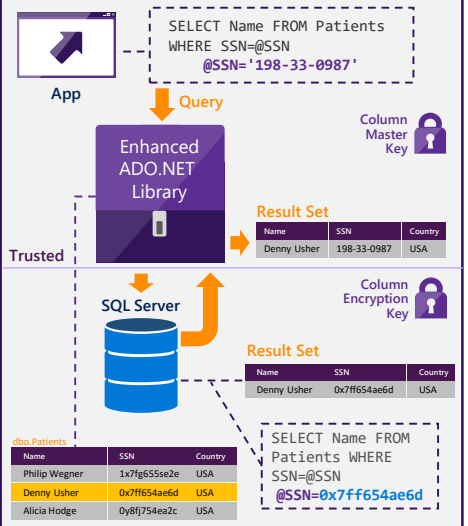
Minimizing impact on OLTP Example: for HOT/WARM (Time Based)

Real-time operational analytics



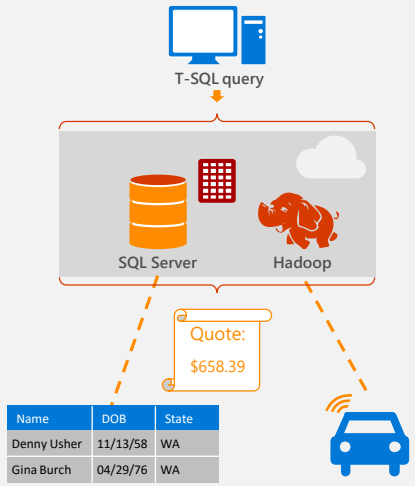
- Up to **30x** faster transactions with in-memory OLTP
- Queries from **minutes to seconds**

Always Encrypted



- Protect data **at rest and in motion**
- **Without impacting** database performance

PolyBase

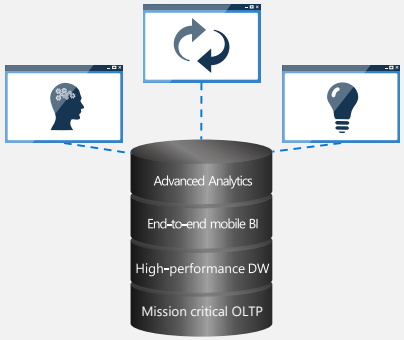


- Manage structured & unstructured data
- **Simple T-SQL** to query Hadoop (HDFS)
- **JSON support**

End-to-end mobile BI

- **In-memory** built-in
- **Real-time** with direct query capabilities
- **Powerful modeling** with 250+ built-in analytical functions
- **Mobile reports** with online & offline access
- **Modern data visualizations** with Reporting Services or Power BI

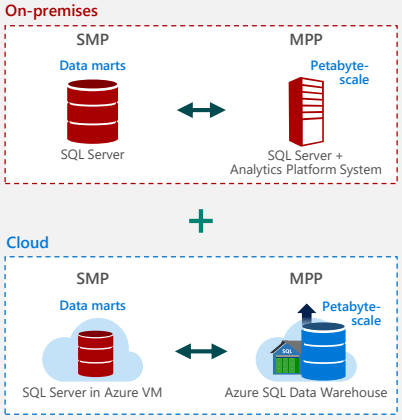
In-database Advanced Analytics



R built-in to SQL Server

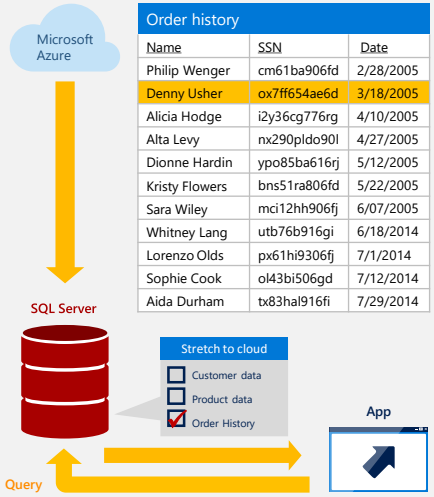
- **R built-in to your T-SQL**
- **Real-time operational analytics** without moving the data
- **Open source R with in-memory & massive scale** – multi-threading and massive parallel processing

Highest performing data warehouse



- **Scale to MPP** on-premises & in the cloud
- **Simple T-SQL** to manage structured and unstructured data
- **½ the cost** of Oracle Exadata

Stretch database



- **Data is encrypted & queryable**
- **Save money** & improve customer experience
- **No application changes**

Learn more!

www.microsoft.com/SQLServer2016

Thank you!

@gwalters69 george.walters@microsoft.com

Resources:

http://blogs.msdn.com/b/data-platform/

http://www.microsoftvirtualacademy.com

http://channel9.msdn.com

Microsoft