Microsoft Services

SQL Server 2016 / 2017
New Features and Capabilities

Reminder: Intersect with Speakers and Attendees

- **Tweet tips and tricks that you learn and follow tweets posted by your peers!**
  - Follow: #SQLintersection and/or #DEVintersection
- **Join us – Tuesday Evening – for SQLafterDark**
  - Doors open at 7:00 pm
  - Trivia game starts at 7:30 pm

    Winning team receives something fun!
  - Raffle at the end of the night

    Lots of great items to win including a seat in a SQLskills Immersion Event!
  - The first round of drinks is sponsored by SentryOne and SQLskills

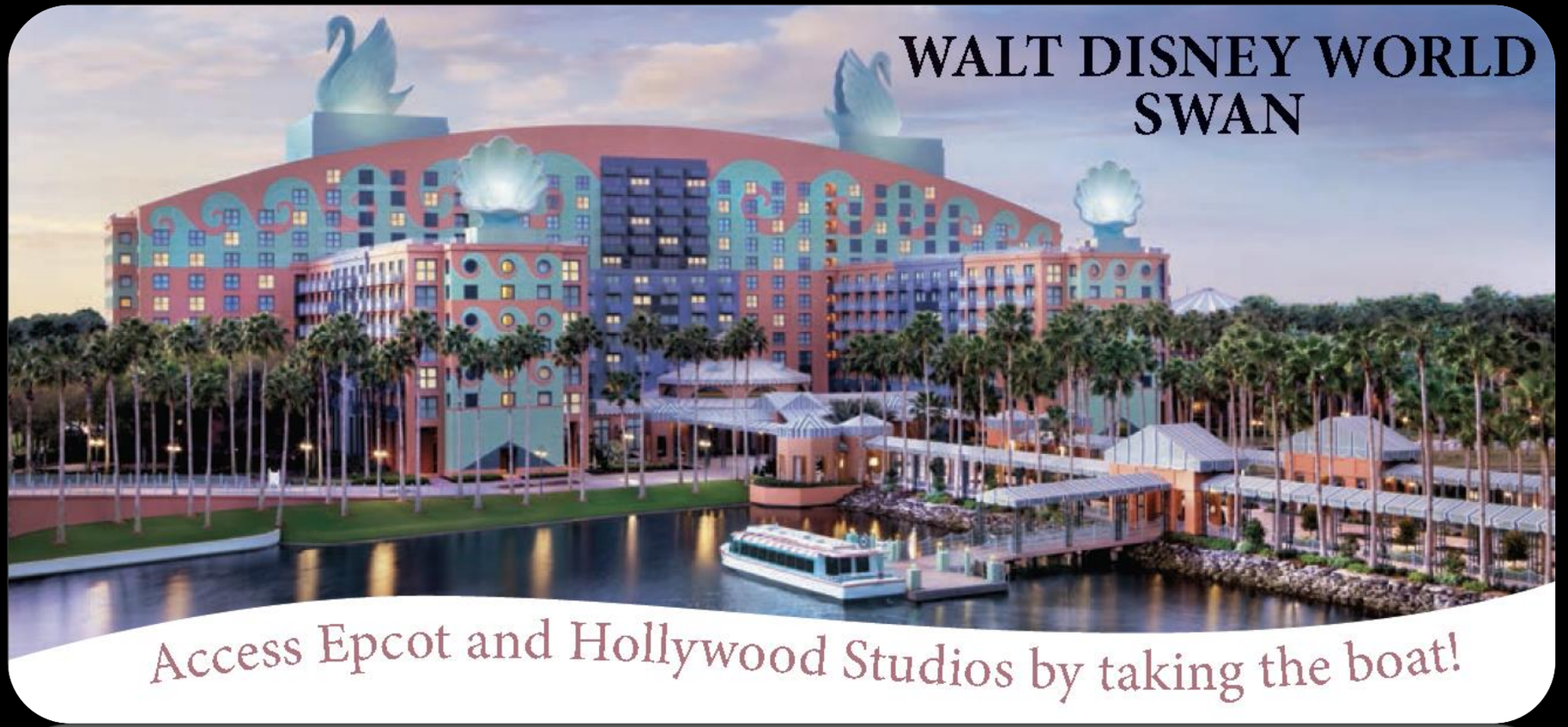
SQLafter Dark

SQL intersection

SQLskills immerse yourself in sql server & SentryOne™

Save the Date!

www.SQLintersection.com



2018

Mar 25-28

We're back in Orlando!

SQL intersection

Leave the every day behind and enter a world of wonder and enchantment at the Walt Disney World® Resort. Located in the heart of the most magical place on earth, the Walt Disney World Swan and Dolphin Resort provides a truly extraordinary backdrop for our event! Beautiful tropical landscaping, tranquil waterways, and classic art and architecture work together to create a stunning landmark!

Workshop Agenda

- SQL Server 2016 / 2017 In-Memory OLTP & Columnstore Enhancements
- SQL Server 2016 / 2017 Monitoring and Tooling
- SQL Server 2016 / 2017 Optimizer Enhancements
- SQL Server 2016 / 2017 T-SQL & Developer Enhancements
- SQL Server 2016 / 2017 Security Improvements

SQL intersection

SQL Server 2016: Everything Built-in

Industry leader in Mission Critical OLTP

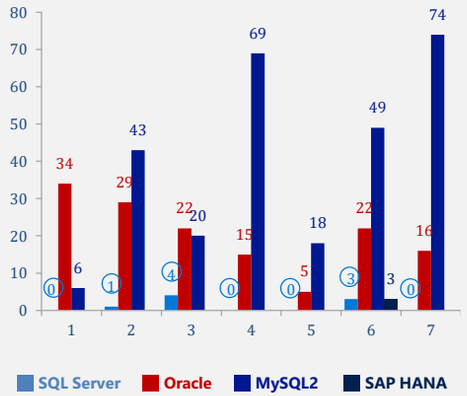
built-in

2 year industry leader

Most secure database 7+ years in a row

built-in

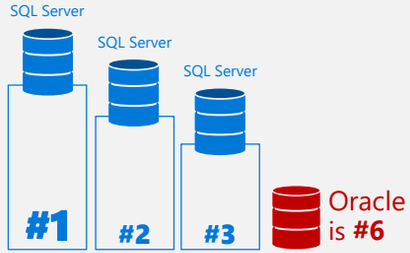
Least vulnerable



Highest performing data warehouse

built-in

#1 performance

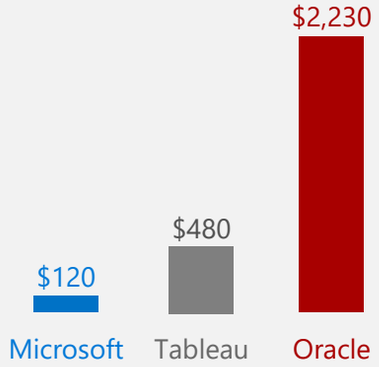


TPC-H

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

Most 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 5/4/2015

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

1/10th the cost of Oracle

SQL Server 2017

INDUSTRY-LEADING PERFORMANCE AND SECURITY NOW ON LINUX AND DOCKER

Choice of platform and language

T-SQL	PHP
Java	Node.js
C/C++	Python
C#/VB.NET	Ruby

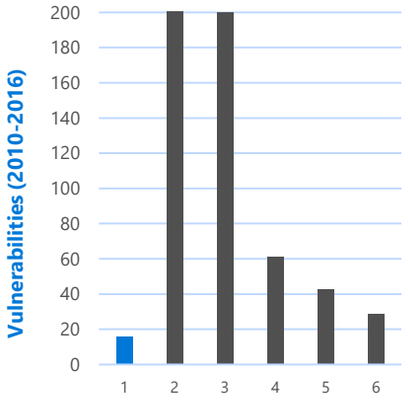
Industry-leading performance

#1 TPC-H performance
1TB, 10TB, 30TB

#1 TPC-E performance

#1 price/performance

Most secure over the last 7 years

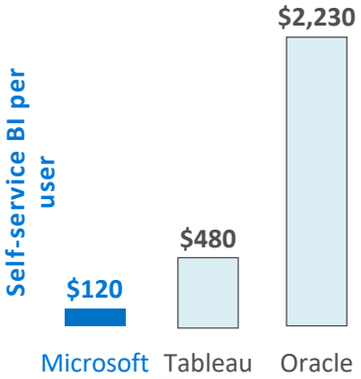


Only commercial DB with AI built-in

R

R and Python + in-memory at massive scale

End-to-end mobile BI on any device



A fraction of the cost

In-memory across all workloads

Private cloud

Public cloud

Most consistent data platform

SQL SERVER 2017

Analyze complex relationships with support for graph data

Advanced Machine Learning with R & Python, now with native T-SQL scoring

Continuous performance with Automatic Plan Correction & Adaptive Query Processing

Enhanced support for Columnstore indexes and In-memory OLTP

SQL intersection

SQL Server 2016 SP1+ Capabilities

- SQL Server SP1 will allow developers and ISVs to build applications which scale across all editions of SQL Server
- Helps developers promote a consistent programming experience
- Core SQL Server OS features are available across Enterprise, Standard, Web, Express, and Local DB*
- New Features extended to lower Editions and DMVs
- Reference: https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/what-s-new-in-sql-server-2016-database-engine

SQL Server 2016 RTM vs. SP1+ Capabilities

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	No**

Limitations of Standard Edition

Max 24 cores

MAXDOP of 2

Max 128GB memory

Max 25% of available server memory for InMemory operations

No pushdown filters for ColumnStore operations

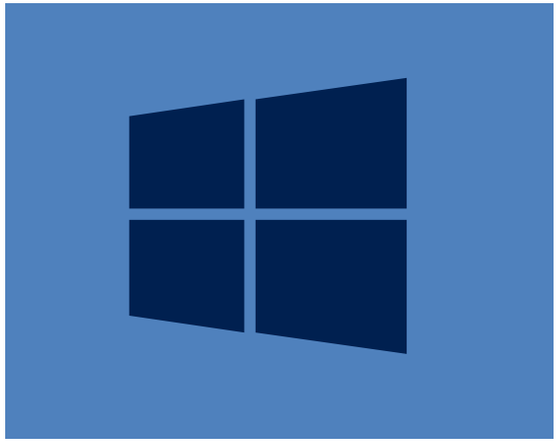
No Polybase scale-out

No R Services parallelism

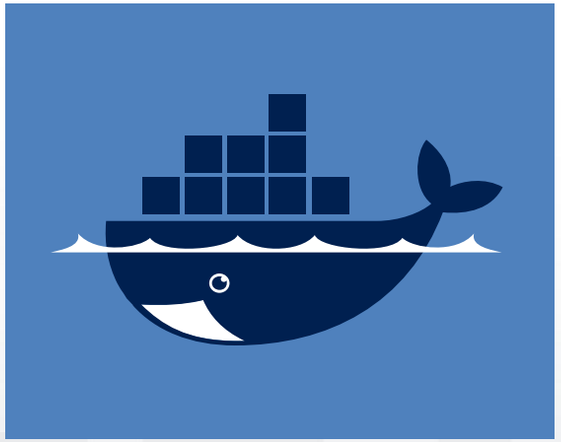
No Mobile BI

Power of SQL Server on the Platform of your Choice

Windows



Linux/Windows Container



Linux

- Linux distributions including RedHat Enterprise Linux (RHEL), Ubuntu, and SUSE Enterprise Linux (SLES)
- Docker: Windows & Linux containers
  - https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker
- Windows Server / Windows 10
- Package-based installation, Yum Install, Apt-Get, and Zypper

In-Memory OLTP

SQL intersection

In-Memory OLTP Overview

- **Technology introduced in SQL Server 2014**
- **Ensures all table data stays in memory at all times**
- **No locking or latching data structures**
- **Everything is organized based on rows**
  - Every table must have at least one index to tie rows into tables
- **Optimized for high-intensive and high-concurrency OLTP environments**
- **Significant improvements made in SQL Server 2016**

SQL intersection

How can In-Memory OLTP help you?

Targeted for high-throughput OLTP systems

- Lots of short, quick transactions

Very high concurrency requirements

- Locking and latching problems

Transaction logging issues

- High WRITELOG waits, LATCH waits

SQL intersection

In-Memory OLTP Hash Indexes

- New In-Memory OLTP Index
- Used exclusively for equality searches that produce small recordsets
- All index fields are required in predicate in order to be used
- No ability to perform sorting
- No ability to perform range-based searches
- Requires defining a bucket_count in order to size the Index array

SQL intersection

Non Clustered (Range) Index

- **Similar in concept to traditional B-tree Indexes**
  - We call these “BW-tree” Indexes
- **Page structures are used to arrange data in a singular sorted order**
- **Traversal starts at a root page and eventually points to a leaf level page**
- **Index pages are never updated (unlike traditional B-tree indexes)**
  - In-place updates require latching
  - No doubly linked list – would require page updates

Non Clustered (Range) Index cont.

- **Index page sizes may vary, but with a max size of 8K**
- **Great for Inequality (Range) based searches**
  - WHERE CreationDate >= ‘5/14/2015’
- **Allows for single direction sorting**
  - Traditional B-tree indexes allow for bi-directional sorting
  - We do not keep track of a “previous” page pointer – would require updates to the pages
- **No Bookmark/Key lookups with these Non Clustered Indexes**
  - The leaf level always points to a row of data

SQL intersection

In-Memory OLTP Logging

- **Two table types: Schema_Only and Schema_And_Data**
- **Schema_Only table data does not survive a restart**
- **Schema_And_Data data is durable and must be saved to disk**
- **Durable table data is written to checkpoint-file pairs**
  - Data is written sequentially to these files (as compared to on-disk)
- **Index Modifications not written to the transaction log**
- **Schema_Only table modifications not written to the transaction log**

SQL intersection

In-Memory OLTP Enhancements

ALTER support

- Full schema change support: add/alter/drop col
- Add/drop index supported

Surface area improvements

- Almost full T-SQL coverage including scalar user-defined functions and triggers
- OUTPUT clause, NULLable key columns, UNIQUE indexes, Foreign Keys, Check Constraints

Improved scaling

- Increased row size allowed for durable tables (>8060 bytes with or without LOB columns)
- Columnstore Indexes on In-Memory OLTP tables
- Parallel scans

Statistics improvements

- Statistics Updated Automatically

SQL intersection

Altering natively compiled stored procedures

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

```
CREATE PROCEDURE dbo.StoredProcedure
WITH
    NATIVE_COMPILATION, SCHEMABINDING,
    EXECUTE AS OWNER
AS BEGIN ATOMIC WITH
(
 TRANSACTION ISOLATION LEVEL = SNAPSHOT,
 LANGUAGE = N'us_english'
)
 SELECT c1, c2 FROM dbo.TableName
END
GO

ALTER PROCEDURE dbo.StoredProcedure
WITH
    NATIVE_COMPILATION, SCHEMABINDING,
    EXECUTE AS OWNER
AS BEGIN ATOMIC WITH
(
 TRANSACTION ISOLATION LEVEL = SNAPSHOT,
 LANGUAGE = N'us_english'
)
 SELECT c1 FROM dbo.TableName
END
```

SQL intersection

Greater Transact-SQL coverage

ALTER PROCEDURE (Transact-SQL)

Nested execution of natively compiled stored procedures

Outer joins

UNION ALL and UNION

Columnstore Indexes

Full Collation Support – BIN2 collation no longer required for indexes on character columns

GROUP BY without aggregation functions in the SELECT clause

Natively compiled Scalar Functions and Triggers

SQL intersection

Natively Compiled Scalar User Defined Functions

- Can be used within Native Stored Procedures or in InterOp code
- Must include same Native Compilation constructs as Natively Compiled Stored Procedures

```
CREATE FUNCTION dbo.GetOrderDate
(
    @SalesOrderID INT
)
RETURNS DATETIME
WITH
    NATIVE_COMPILATION, SCHEMABINDING,
    EXECUTE AS OWNER
AS  BEGIN ATOMIC WITH
    (
     TRANSACTION ISOLATION LEVEL = SNAPSHOT,
     LANGUAGE = N'us_english'
    )
    DECLARE @OrderDate DATETIME

    SELECT @OrderDate = OrderDate
    FROM dbo.SalesOrderMemOpt
    WHERE OrderID = @SalesOrderID
    RETURN(@OrderDate)
END
```

SQL intersection

Natively Compiled Triggers

- AFTER triggers
- Must include the same directives as Functions and Procedures
- Must target an In-Memory table if interacting with other tables

```
CREATE TRIGGER Production.tr_ProductInMem_Insert
ON Production.[Product_inmem]
WITH
    NATIVE_COMPILATION, SCHEMABINDING,
    EXECUTE AS OWNER
    AFTER INSERT
AS BEGIN ATOMIC WITH
(
TRANSACTION ISOLATION LEVEL = SNAPSHOT,
LANGUAGE = N'us_english'
)
DECLARE @Rows INT, @OpType CHAR(1)
SELECT @Rows = COUNT(*) FROM INSERTED

INSERT INTO Production.InMemTracking
(RowsAffected, OptType)
SELECT @Rows, 'I'

END
```

SQL intersection

Additional 2016 In-Memory OLTP Enhancements

- Transparent Data Encryption support
- Nullable Index Key columns
- LOB types
- UNIQUE indexes
- Foreign Key and Check Constraints
- AFTER triggers
- Parallel Scans
- Auto-update of Statistics

SQL intersection

Demo

Natively Compiled Objects

SQL intersection

Columnstore Indexes

SQL intersection

What is a Columnstore Index?

Index that stores data in a columnar fashion (as opposed to a rowstore)

- We typically think of normal indexes as rowstore (b-tree) indexes

Highly compressed - great for operating on large sets of data

- For analytics – can be an order of magnitude better performance as opposed to B-tree indexes

Makes great use of multiple processor machines

- Batch Execution is used to process multiple rows at the same time

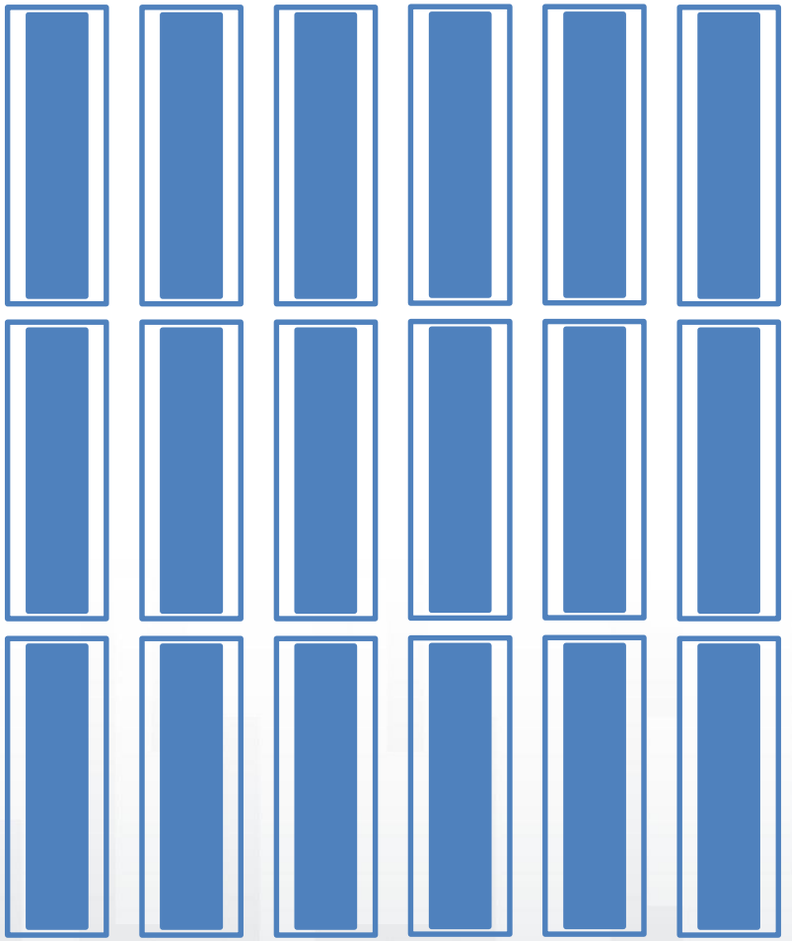
Two types: Clustered and nonclustered

- Much different than their rowstore based counterparts

SQL intersection

Columnar Data Storage

- **Highly compressed – common data type**
  - Columns often have repeating data – which makes it easy to compress
  - More data fits in memory
  - Efficient IO from disk to memory
- **Each column can be accessed independently**
  - Fetch only columns that are needed
  - Can dramatically decrease I/O



SQL intersection

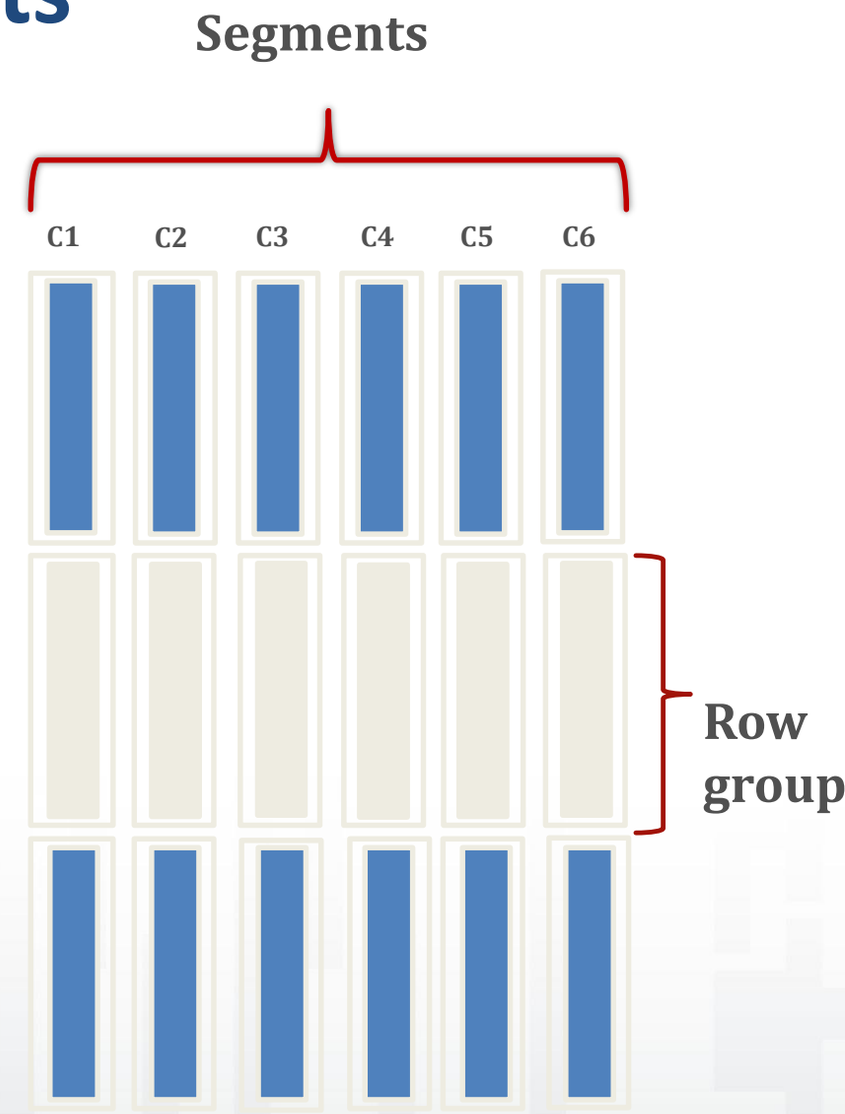
Row Groups & Segments

Segment

- A segment contains values for one column for a set of rows (typically around 1M rows)
- Segments are compressed and LOB stored
- Segment (LOB) is unit of transfer between disk and memory

Row group

- Segments for the same set of rows
- The position of a value in a column indicates to which row it belongs



SQL intersection

Columnstore Target Workloads

Data Warehouse – write once, read often

- Aggregation and filtering are common on large data sets

OLTP Applications (new to 2016)

- Run expensive analytical queries against OLTP workload
- Changes allow for mixture of b-tree indexes and columnstore indexes
- Filtered NC columnstore indexes are great for analytics

SQL intersection

Clustered Columnstore Indexes

- **Columnstore Index IS the table**
  - Similar in concept to a B-tree Clustered Index
- **High rates of compression – smaller disk footprint**
- **Introduced in SQL Server 2014**
  - In 2014 – could be the only index structure – no B-tree indexes allowed
  - This is fixed in SQL Server 2016

SQL intersection

Nonclustered Columnstore Indexes

- **Introduced in SQL Server 2012**
  - Made the table readonly
- **Separate data structure from the table data**
- **No “index key columns” – include all columns that queries may include in large scan operations**
- **In SQL Server 2016 – table is now writeable when a non-clustered Columnstore index is created on table**

SQL intersection

What's New SQL Server 2016 Columnstore Index

- **Updateable non-clustered Columnstore indexes**
  - One per normal B-tree table
  - This index can have a filtering condition
- **Multiple b-tree indexes on Clustered Columnstore Tables**
- **An In-Memory table can have one Columnstore index**
- **Constraints available on a Clustered Columnstore table through B-tree indexes**
- **Columnstore Indexes support RCSI and Snapshot Isolation**
- **Columnstore Indexes available on AG secondaries**
- **Specify Columnstore Indexes when creating tables**

SQL intersection

Use filtered indexes to minimize CS overhead

- **Create a filtered index so the NC CSI index contains “warm” data and the rowstore contains “hot data**
- **When the QO chooses the filtered CS index, the query will return the correct results even when the results contain rows that do not meet the filtered condition.**

```
CREATE TABLE t_account (
    accountkey int PRIMARY KEY,
    accountdescription nvarchar (50),
    accounttype nvarchar(50),
    unitsold int
);

CREATE NONCLUSTERED COLUMNSTORE INDEX account_NCCI
ON T_account (accountkey, accountdescription, unitsold)
WHERE accountkey > 0

SELECT SUM( unitsold )
FROM t_account
WHERE accountkey >=0 AND accountkey < 1000
```

SQL intersection

Columnstore Maintenance & Query Improvements

- Reorganize can be used on CS indexes to remove deleted rows
- Better batch mode execution support for aggregation functions in queries
- Single threaded queries using MAXDOP 1 can execute in batch mode

SQL intersection

Columnstore on in-memory tables

SQL Server 2016

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

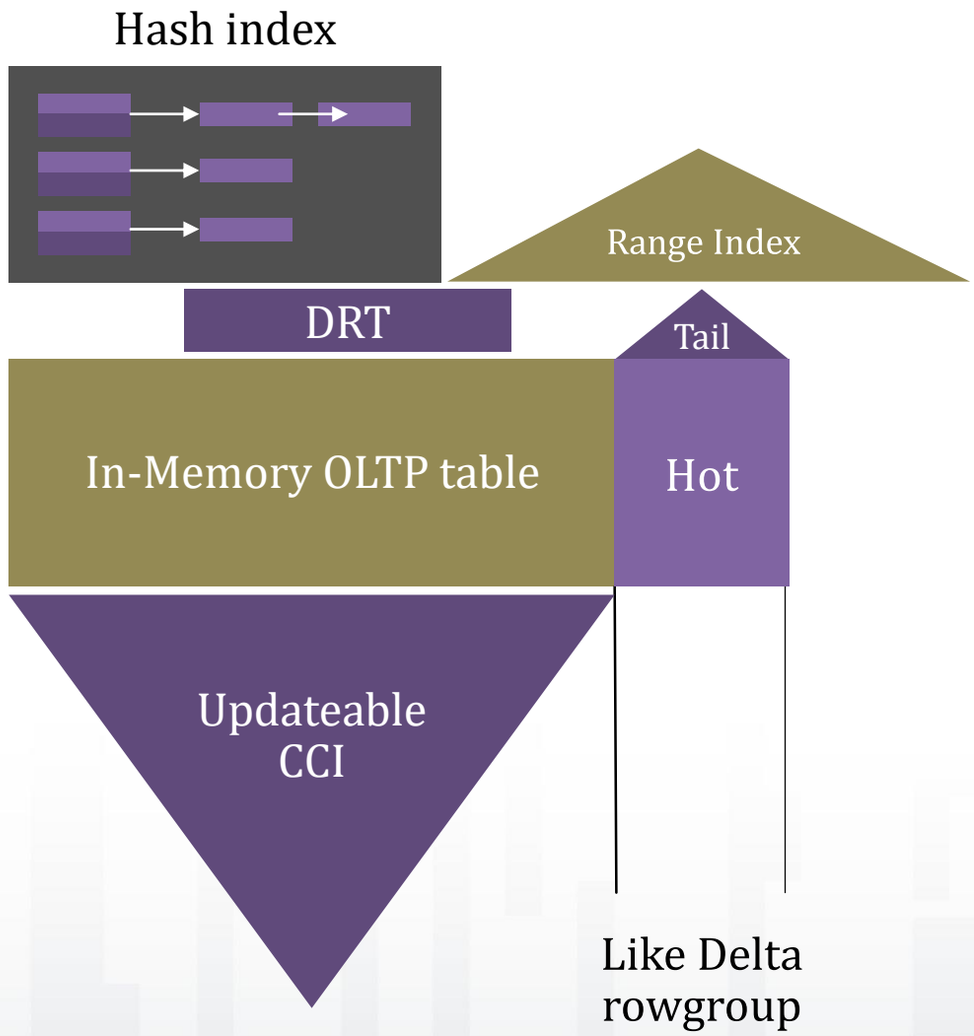
No explicit delta rowgroup

- 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



SQL intersection

Demo

Columnstore Indexes

SQL intersection

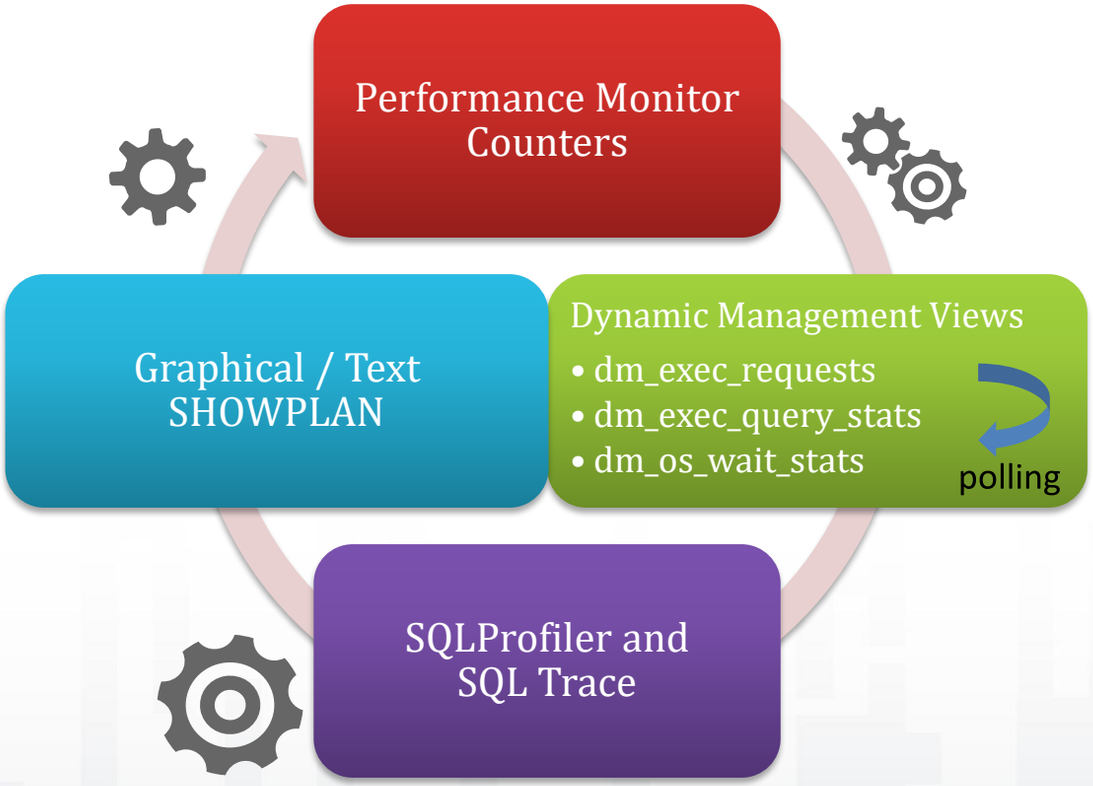
SQL Server Management Studio

Monitoring and Tooling

SQL intersection

SQL Server 2016/2017 Monitoring and Tooling

Traditional Troubleshooting

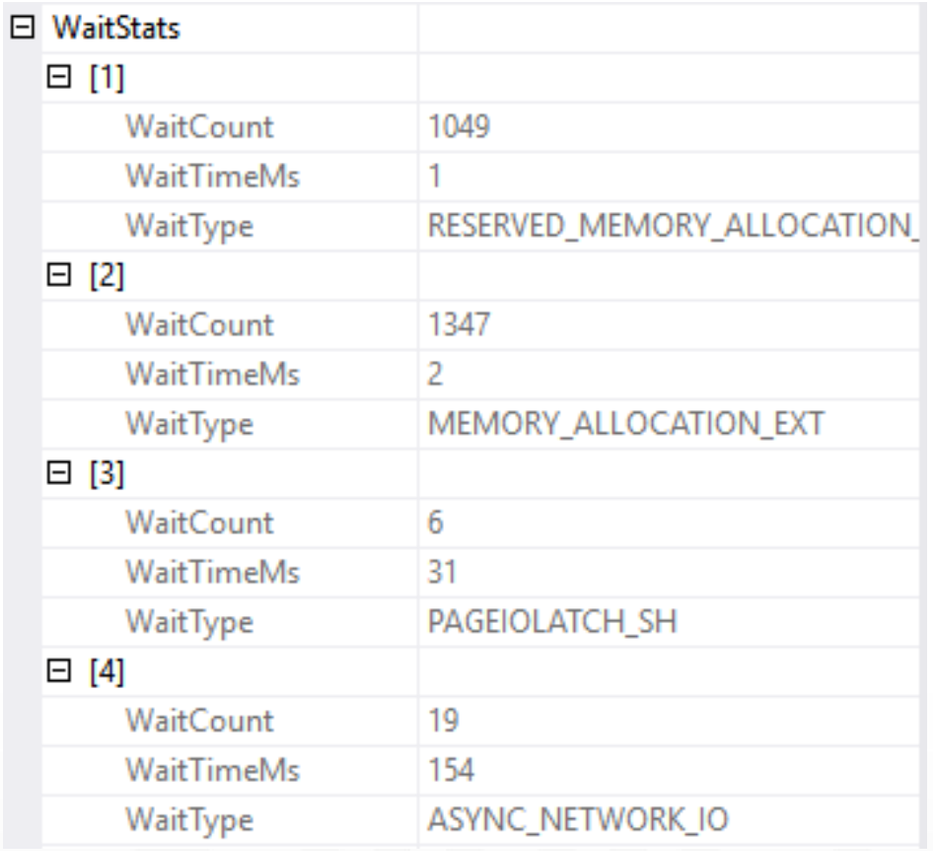


Full SQLTrace Parity+ since 2012

- Extended Events is scalable
- Query Store is persisted and improving
- Performance Dashboard Reports
- Live Query Statistics
- Lightweight Query Profiling
- Expanded Query Plan Diagnostics
- SSMS Dump Analysis (Preview)

Getting all context info in Showplan: Waits

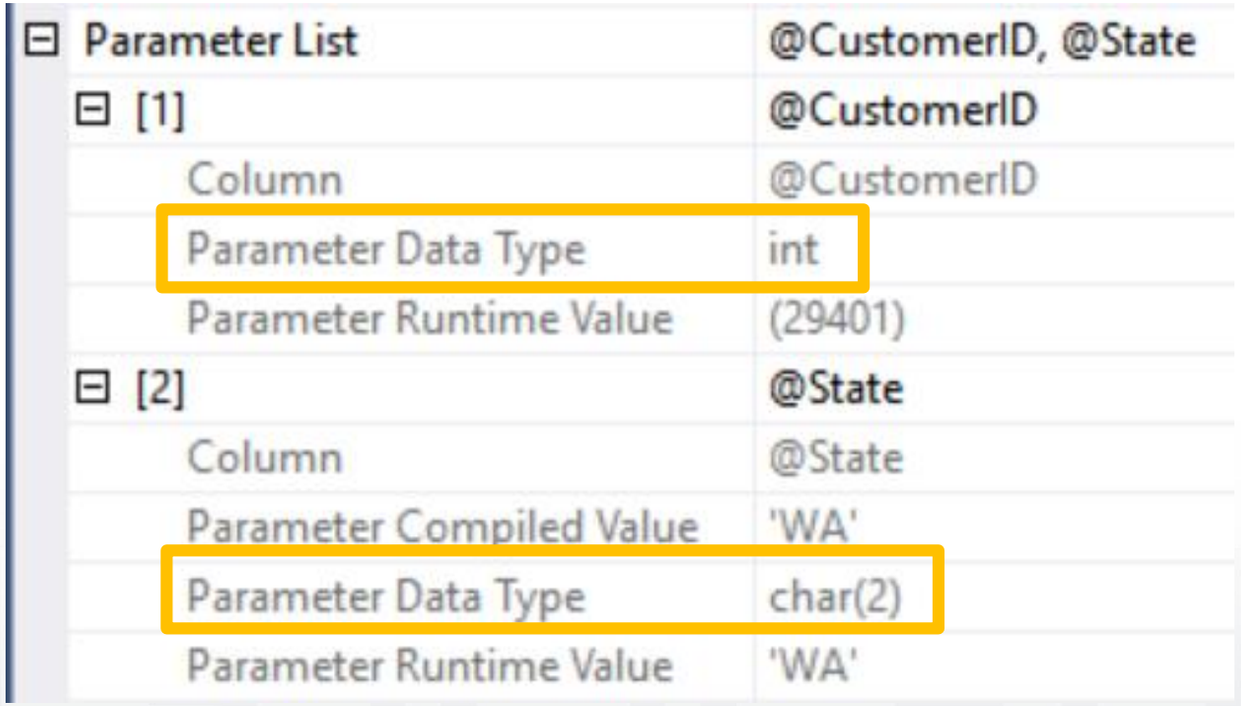
Shows top waits from sys.dm_exec_session_wait_stats



WaitStats	
[1]	
WaitCount	1049
WaitTimeMs	1
WaitType	RESERVED_MEMORY_ALLOCATION_
[2]	
WaitCount	1347
WaitTimeMs	2
WaitType	MEMORY_ALLOCATION_EXT
[3]	
WaitCount	6
WaitTimeMs	31
WaitType	PAGEIOLATCH_SH
[4]	
WaitCount	19
WaitTimeMs	154
WaitType	ASYNC_NETWORK_IO

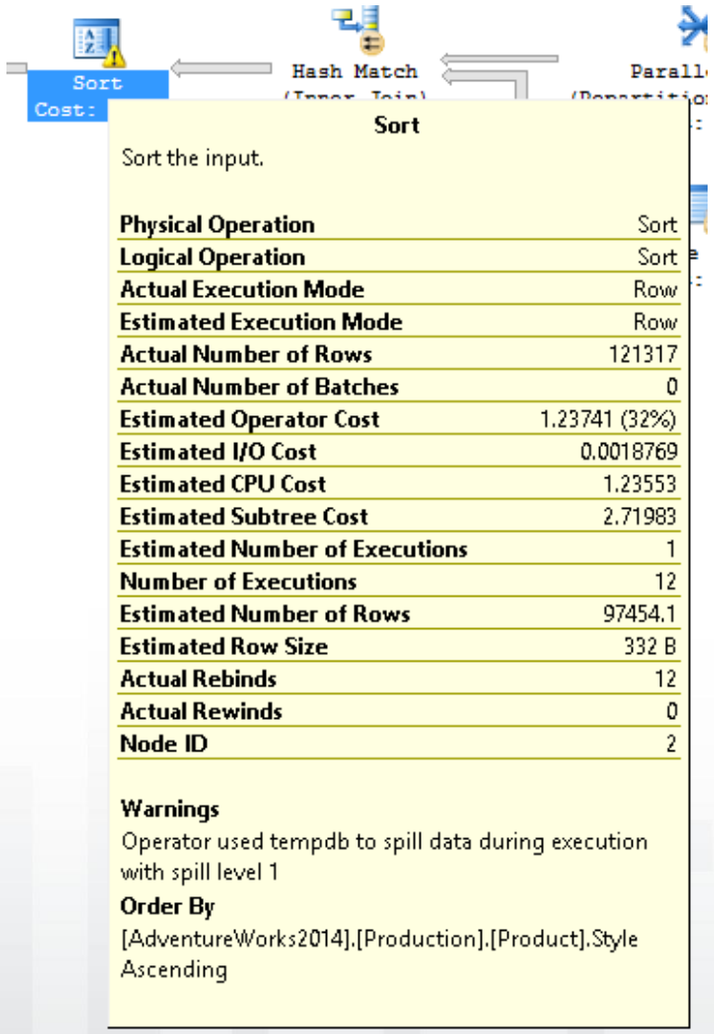
Getting all context info in Showplan: Parameter Data Types

Easier detection of type conversion issues

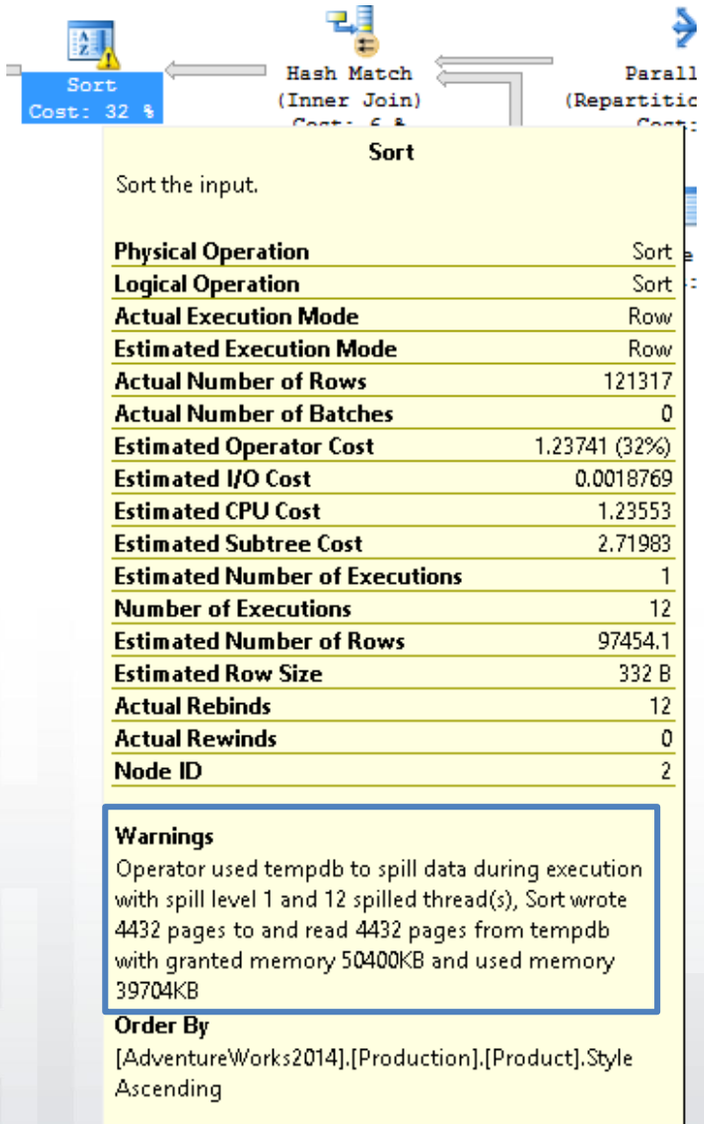


New Spills Warnings - Sort

- Up to SQL Server 2016



- SQL Server 2016 and 2014 SP2



Demo

Performance Monitoring and Tools in SQL Server 2017

- SSMS and Performance Dashboard Reports
- Analyze Single Execution Plan
- XE Profiler
- SSMS Dump Analysis (Preview)

SQL intersection

"A Bad Plan is not the one which failed, but the one which succeeded at the Greatest Cost."

Anonymous DBA

Plan Guides

SQL intersection

Why Use Plan Guides?

- Useful for tuning queries generated by 3rd party applications
- Plan guides work by keeping a list of queries on the server, along with the Hints you want to apply
- You need to provide SQL Server with the query you want to optimize and a query hint using the OPTION clause
- When the query is optimized, SQL Server will apply the hint requested in the plan guide definition

SQL intersection

Plan Guides Stored Procedures

- Use the sp_create_plan_guide stored procedure to create a plan guide
- Use sp_control_plan_guide to drop enable or disable plan guides
- You can see which plan guides are defined in your database using the sys.plan_guides catalog view
- Note: When Using Plan Guides, you must match Query Text and Parameter Names exactly

SQL intersection

Query Store

SQL intersection

Query and Query Plan Fingerprints

- Query Fingerprint
  - query_hash
  - Explicitly identifies a specific query in the cache.
  - *sys.dm_exec_requests*
  - *sys.dm_exec_query_stats*
- SQL Handle
  - sql_handle
  - Token for the SQL text that relates to a batch.
  - *sys.dm_exec_sql_text*
  - *sys.dm_exec_query_stats*
  - *sys.dm_exec_query_memory_grants*
- Query Plan Fingerprints
  - query_plan_hash
  - Useful to determine queries that share the same execution plan.
  - Can be used to determine if the query plan has changed.
  - *sys.dm_exec_requests*
  - *sys.dm_exec_query_stats*
- Plan Handle
  - plan_handle
  - Token for a cached execution plan.
  - *sys.dm_exec_query_plan*
  - *sys.dm_exec_cached_plans*

When performance is not good…

- Database is not working

Website / App is down

- Impossible to predict / root cause

Temporary Perf. issues

- Regression caused by upgrade

System Upgrade

Plan choice change can cause these problems

What are your options today?

- **Most solutions are reactive in nature**
  - Flush the bad plan from the cache with sp_recompile
  - Flush the entire plan cache with DBCC FREEPROCCACHE
  - Force the plan to recompile every time
  - Restart OS / SQL Server (It works for some reason?)
- **Proactive solutions are challenging**
  - Often takes a long time to even detect there is a plan problem
  - Only the latest plan is stored in the cache
  - Need to catch both the good and the bad plan in order to troubleshoot
  - Information is stored in memory only
    - Reboot or memory pressure causes data to be lost
    - No history or timing available – stats are aggregated for what is currently in cache

Why Plan Changes Happen..

- SQL Query Optimizer considers many plans
- When a plan is chosen, it is cached and reused
- As your data changes, it might select a different plan as optimal
- Volume and Data Distribution can affect plan choices
- Sometimes a rare plan choice will be cached
  (The Parameter Sensitive Plan Problem)

SQL intersection

Addressing Plan Choice Regressions

- **First You Have to find the “Slow” Query**
- **Figuring out Why it is slow isn’t Easy**
- **You may not have enough information to fix it**
- **Even if you do know what it is supposed to be…**
  - Can you modify the query to hint it?
  - Can you figure out how to make a plan guide?

Tackling the Problem – What Could We Do?

1. Store the history of plans for each query
2. Baseline the performance of each plan over time
3. Identify queries that have “gotten slower recently”
4. Find a way to force plans quickly and easily
5. Make sure this works across server restarts, upgrades, and query recompiles

This is what the Query Store does for you!

SQL intersection

Introducing the Query Store

- **Plan store persists execution plans per database**
- **Runtime stats store persists execution statistics per database**
- **New views and graphical interface allow you to quickly and easily troubleshoot query performance**
  - Quickly find query plan performance regressions
  - Fix plan regressions by forcing a previous plan
  - Determine the number of times a query was executed in a given time window
  - Identify Top N Queries in the past X hours
  - Audit the history of query plans for a given query
  - Analyze the resource usage patterns for a particular database

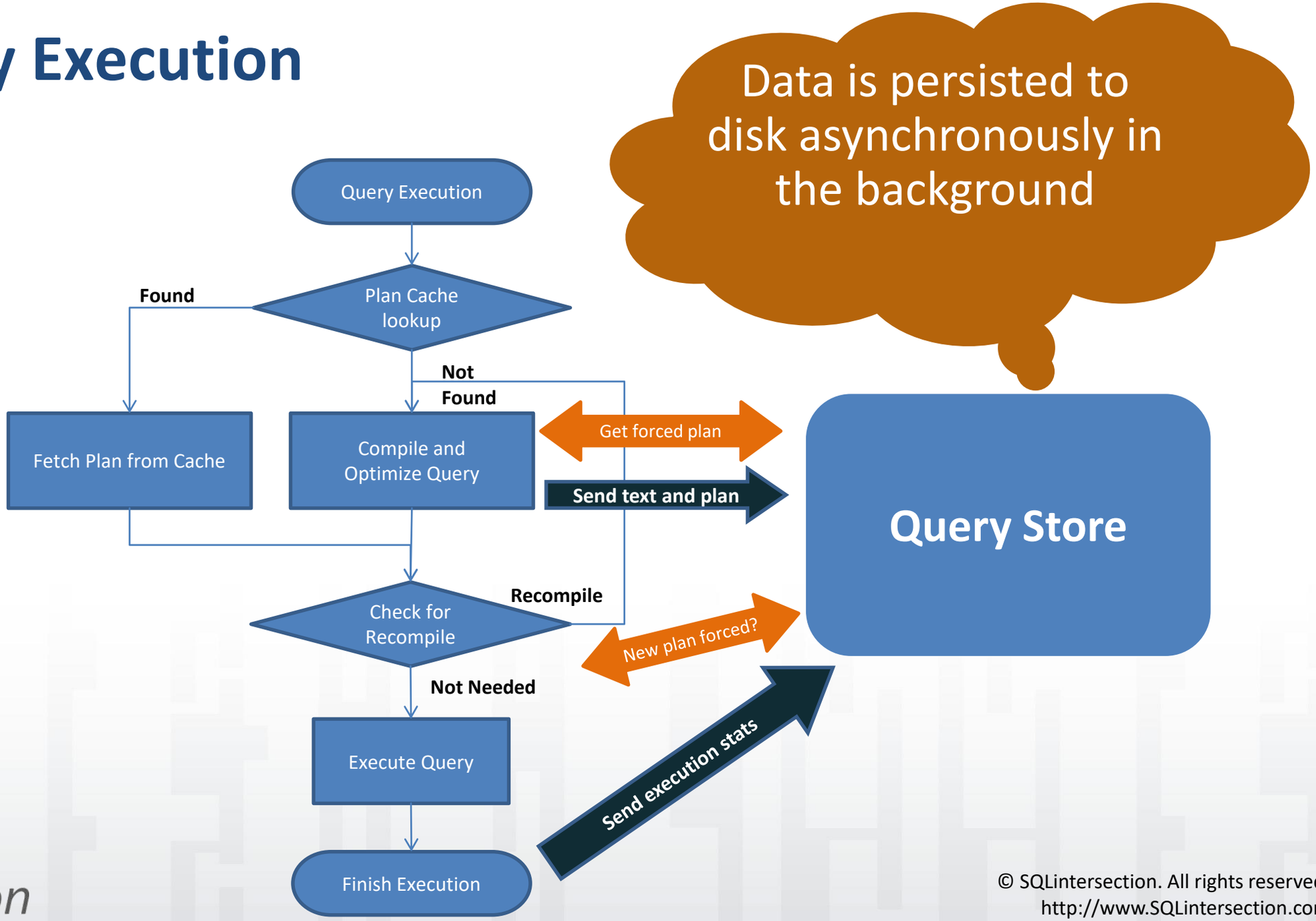
SQL intersection

Key Usage Scenarios

Find and fix query plan regressions	Identify top resource consumers	Reduce risks with server upgrade	Deep analysis of workload patterns/perf

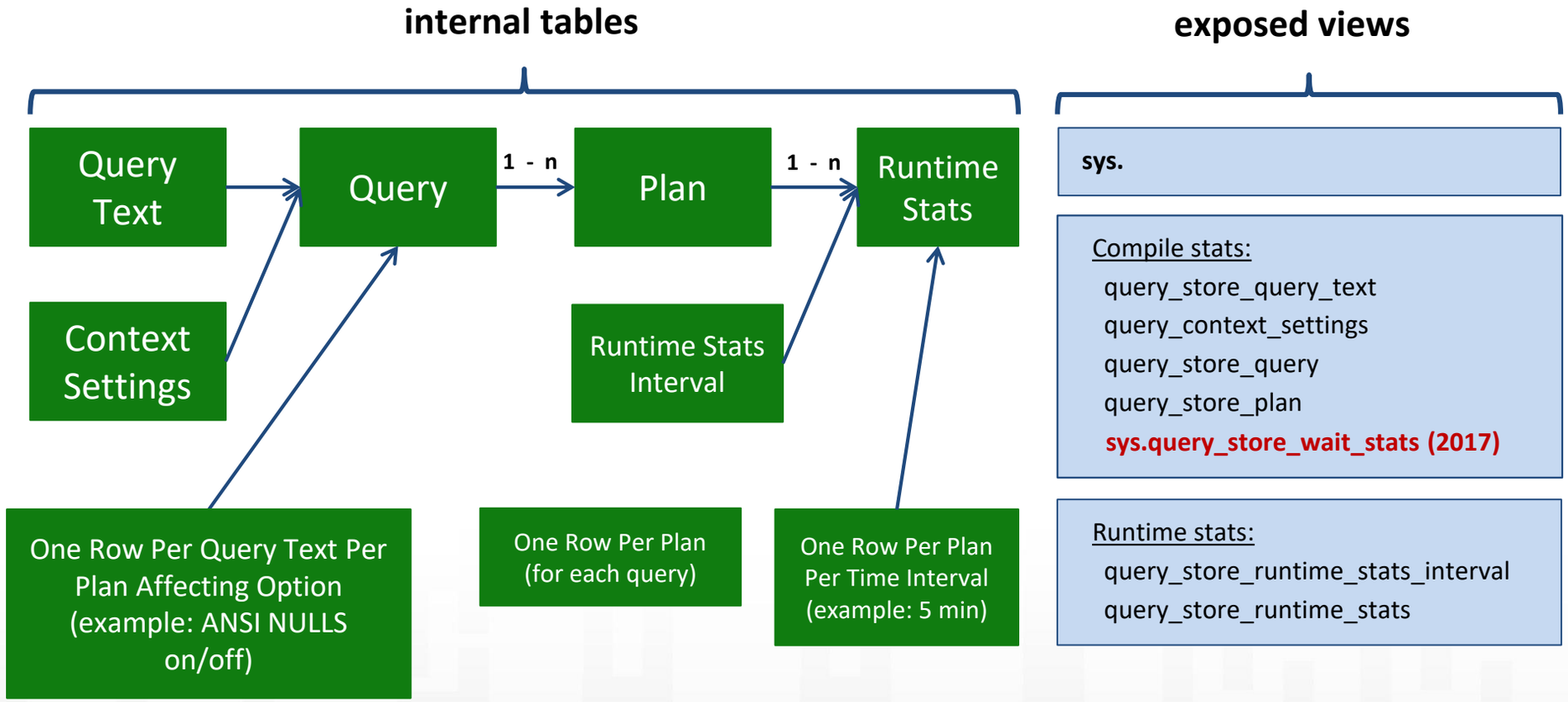
SQL intersection

SQL Query Execution



SQL intersection

Query Store Schema Explained



SQL intersection

Query Store Details

- **Plans and execution data are stored on disk in the user database**
  - Query store data persists reboots, upgrades, restores etc.
  - Plans and statistics are tracked at the database level rather than the server level
- **Query Store is configurable**
  - Settings such as MAX_SIZE_MB, QUERY_CAPTURE_MODE, CLEANUP_POLICY allow you to decide how much data you want to store for how long
  - Can be configured either via the SSMS GUI or T-SQL scripts
- **Query Store can be viewed and managed via scripting or SSMS**

SQL intersection

What does Query Store Track?

- **Query Texts start at the first character of the first token of the statement; end at last character of last token**
  - Comments before/after do not count
  - Spaces and comments inside *do* count
- **Context_settings contains one row per unique combination of plan-affecting settings**
  - Different SET options cause multiple “queries” in the Query Store
  - Plan caching/recompilation behavior unaffected

SQL intersection

Demo

Enabling Query Store and Query Store Properties

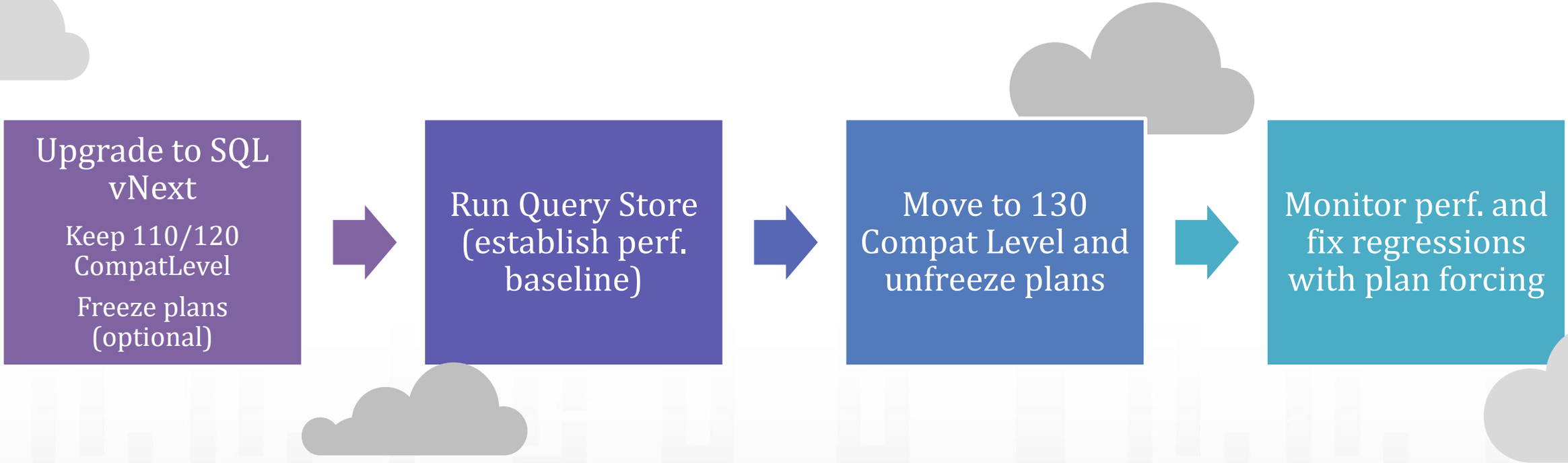
SQL intersection

What Gets Captured?

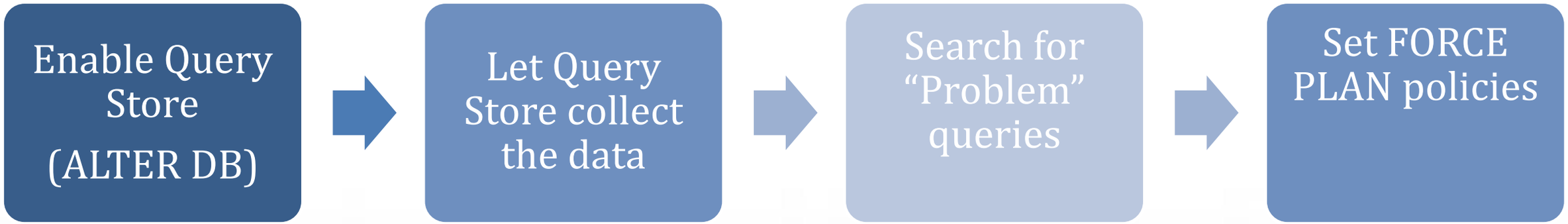
- **Query Texts**
- **Query Plans**
- **Runtime Statistics (per unit of time, default 1 hour)**
  - Count of executions of each captured plan
  - For each metric: average, last, min, max, stddev
  - Metrics: duration, cpu_time, logical_io_reads, logical_io_writes, physical_io_reads, clr_time, DOP, query_max_used_memory, rowcount
  - Data is recorded when a query execution *ends*

SQL intersection

Keeping Stability while Upgrading to SQL Sever 2016/2017

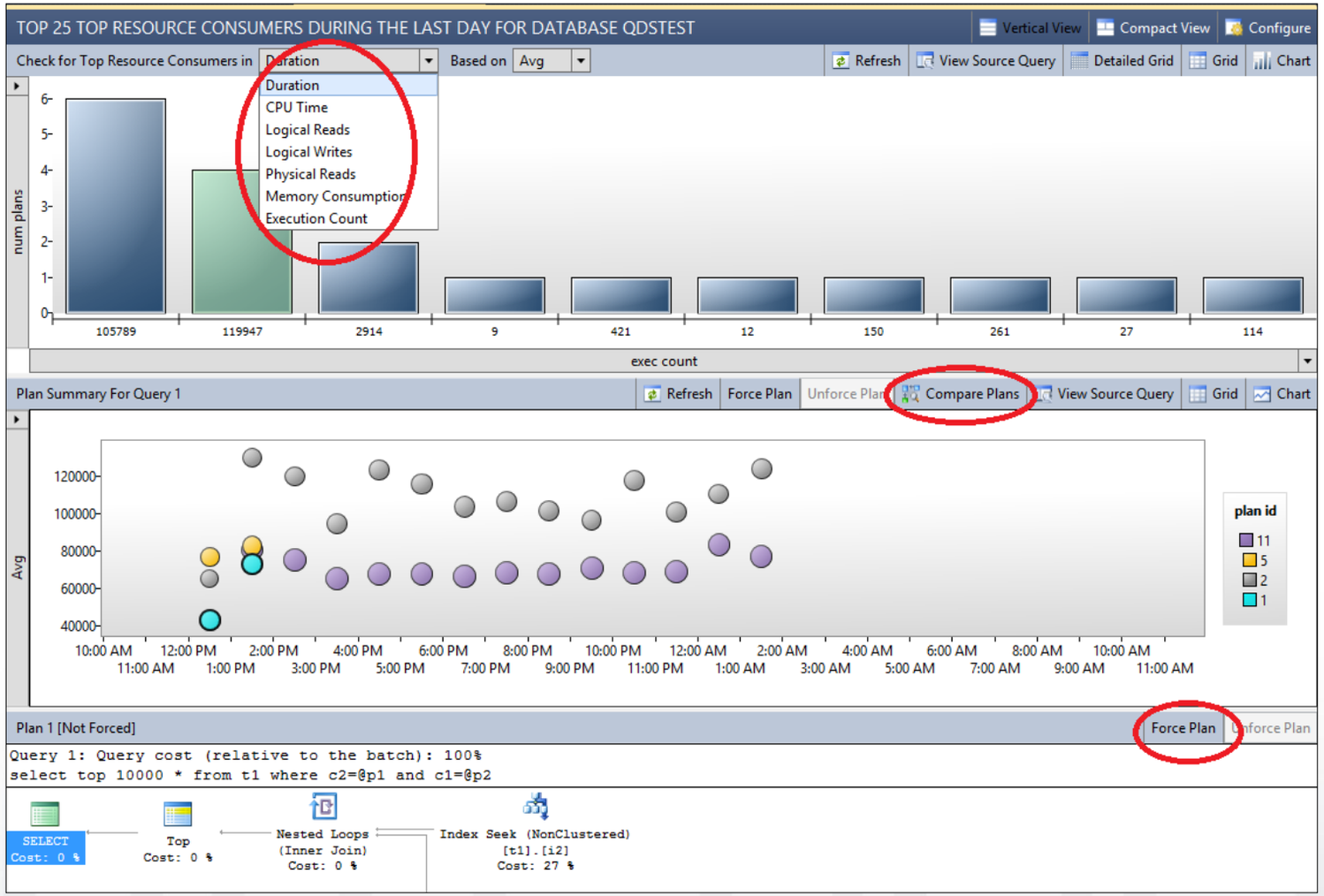


Troubleshooting with Query Store



SQL intersection

Monitoring Performance with Query Store



- The Query Store feature provides DBAs with insight on query plan choice and performance

Working with Query Store

```
/* (6) Performance analysis using Query Store views*/
SELECT q.query_id, qt.query_text_id, qt.query_sql_text,
SUM(rs.count_executions) AS total_execution_count
FROM
sys.query_store_query_text qt JOIN
sys.query_store_query q ON qt.query_text_id =
q.query_text_id JOIN
sys.query_store_plan p ON q.query_id = p.query_id JOIN
sys.query_store_runtime_stats rs ON p.plan_id = rs.plan_id
GROUP BY q.query_id, qt.query_text_id, qt.query_sql_text
ORDER BY total_execution_count DESC

/* (7) Force plan for a given query */
exec sp_query_store_force_plan
12 /*@query_id*/, 14 /*@plan_id*/
```

```
/* (4) Clear all Query Store data */
ALTER DATABASE MyDB SET QUERY_STORE CLEAR;

/* (5) Turn OFF Query Store */
ALTER DATABASE MyDB SET QUERY_STORE = OFF;
```

- DB-level feature exposed through T-SQL extensions
- ALTER DATABASE
- Catalog views (settings, compile & runtime stats)
- Stored Procs (plan forcing, query/plan/stats cleanup)

Troubleshooting Query Store

- **Plan forcing does not always work**
  - Example: If you drop an index, you can't force a plan that uses it
- **Query Store will revert to not forcing if it fails**
  - This keeps the application working if the hint breaks
- **You can see which plans are failing to force by looking at the Plan Table:**
  - There is also a report option in the latest SSMS 17.x

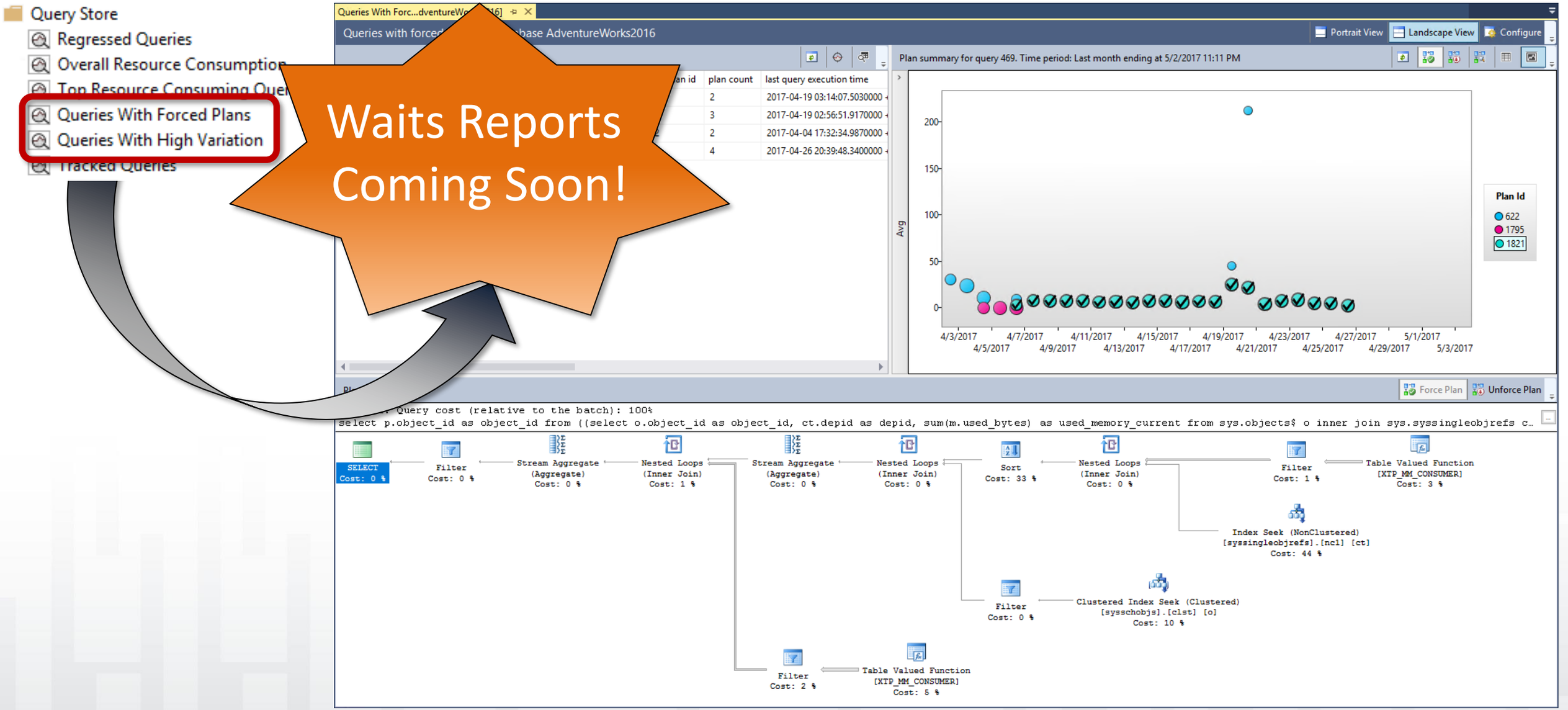
```
SELECT * FROM sys.query_store_plan
WHERE is_forced_plan = 1 AND
force_failure_count > 0
```

Demo

Using Query Store in SQL Server 2016 / 2017

SQL intersection

Queries with Forced Plans in SQL Server 2017

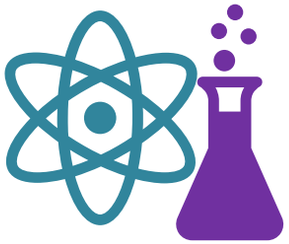


Demo

Query Store in SQL Server 2017

- Query Store Waits
- Queries with Forced Plans
- Queries with High Variations

SQL intersection



SQL Server 2017 – Modern and Intelligent

Query Store – Wait Stats and “Cloud Learnings”

Automatic Tuning and Plan Correction

Query Plan Analysis in SSMS

Adaptive Query Processor

SQL Server 2017 – Query Store Improvements

- **New Query Store Reports**
- **Automatic Tuning Feature Support**

```
ALTER DATABASE AdventureWorks2017
    SET AUTOMATIC_TUNING ( FORCE_LAST_GOOD_PLAN = ON );
```

- **DBCC CLONEDATABASE flushes statistics while cloning to avoid missing query store runtime statistics**
- **New DMVs**
  - sys.query_store_wait_stats
  - sys.dm_db_tuning_recommendations
  - sys.database_automatic_tuning_mode
  - sys.database_automatic_tuning_options

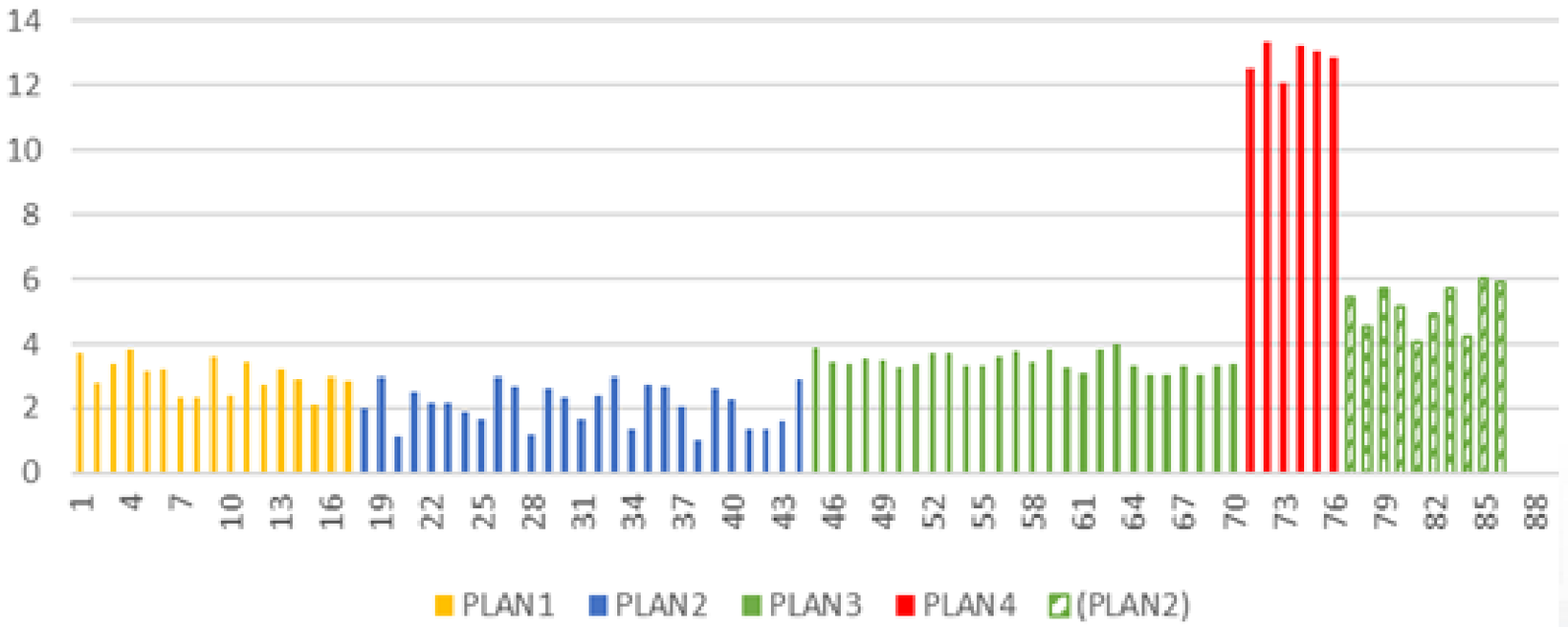
SQL Server 2017 Automatic Tuning

```
ALTER DATABASE CURRENT
SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON);
```

Detect with **dm_db_tuning_recommendations** and force manually

Turn on Auto and system corrects

Reverts back to "Last Known Good"



Perfect to help with parameter sniffing

Demo

Automatic Tuning in SQL Server 2017

SQL intersection

Adaptive Query Processing

SQL intersection

Risks of Misestimation

Slow Query Response Time Due to Bad Plans

Excessive Resource Utilization (CPU, Memory, IO)

Reduced Throughput and Concurrency

T-SQL Refactoring for Off-Model Statements

Adaptive Query Processing (SQL Server 2017)

Interleaved Execution

- Materialize estimates for multi-statement table valued functions (MSTVFs)
- Downstream operations will benefit from the corrected MSTVF cardinality estimate

Batch-mode Memory Grant Feedback

- Adjust memory grants based on execution feedback
- Remove spills and improve concurrency for repeating queries

Batch-mode Adaptive Joins

- Defer the choice of hash join or nested loop until after the first join input has been scanned
- Uses nested loop for small inputs, hash joins for large inputs

Demo

Adaptive Query Processing

- Interleaved Execution
- Batch-Mode Memory Grant Feedback
- Batch-Mode Adaptive Join

SQL intersection

Live Query Statistics

SQL intersection

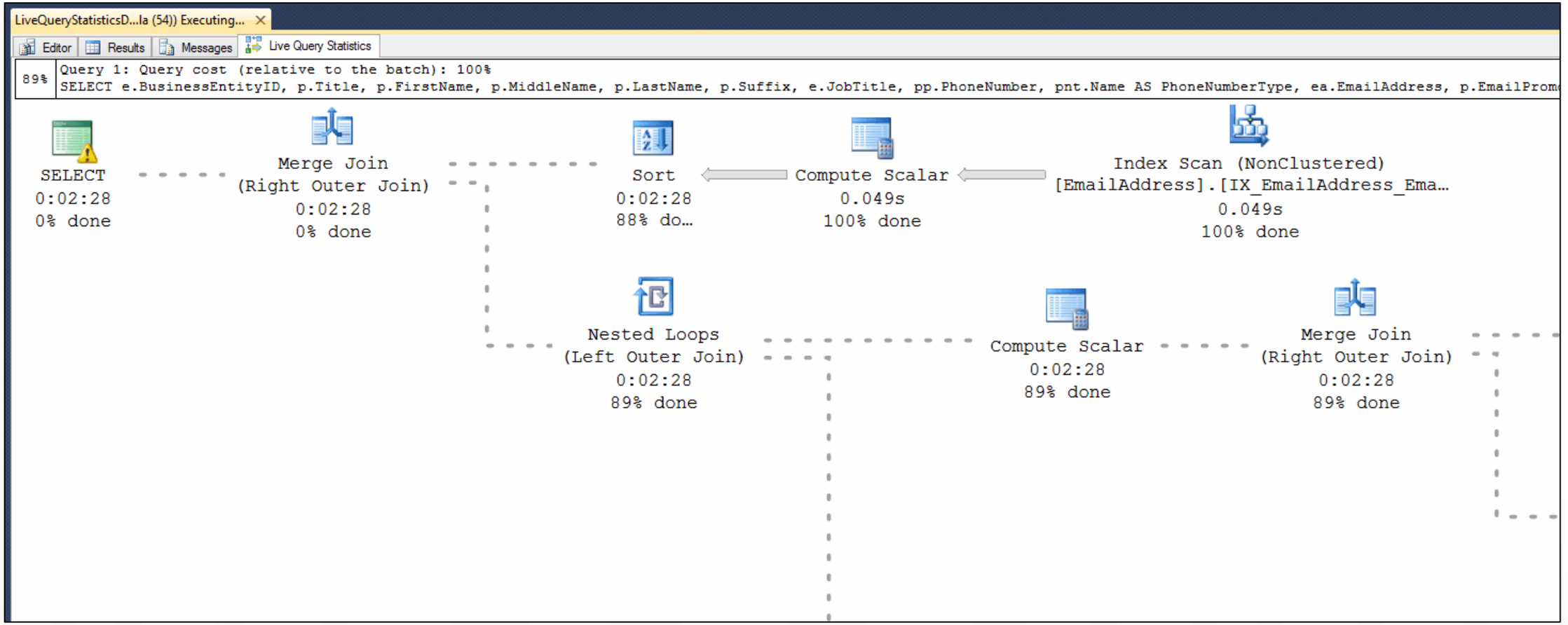
SQL Server 2014+ Query Progress Estimation

Limited feedback mechanism for query execution progress

Very difficult to get insight into long running queries

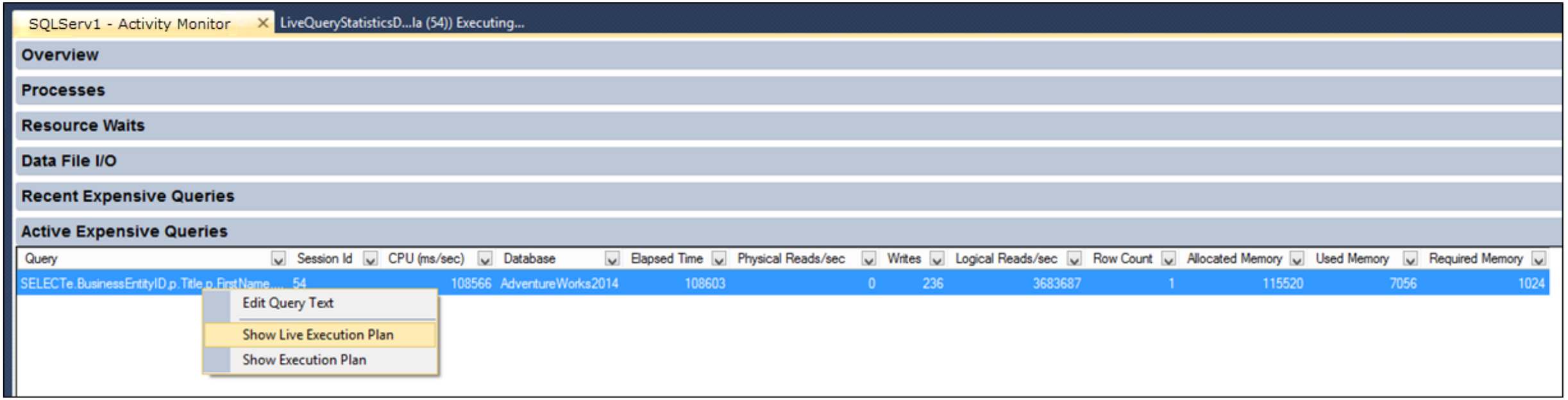
Estimated Query plans do not contain any run-time information
Actual row counts, memory grant usage, execution time warnings

Introducing Live Query Execution Plans!



SQL intersection

Monitor Currently Executing Queries



Live Query Statistics Lightweight Profiling

- **Enable 7412 -** Enables the lightweight query execution statistics profiling infrastructure
- **Much lower overhead for examining Live Query Statistics (Only 2% in 2016 SP1+)**
  - 3.5% overhead in 2014 SP2 / 2016+
  - http://aka.ms/traceflags

Temporal Tables

SQL intersection

Temporal Tables in SQL Server 2016

Track change to table data automagically

Triggers are NOT used to keep track of data changes

Return data from a table AS OF a specific point in time

When a table is tracked, a second table is used to keep historical data

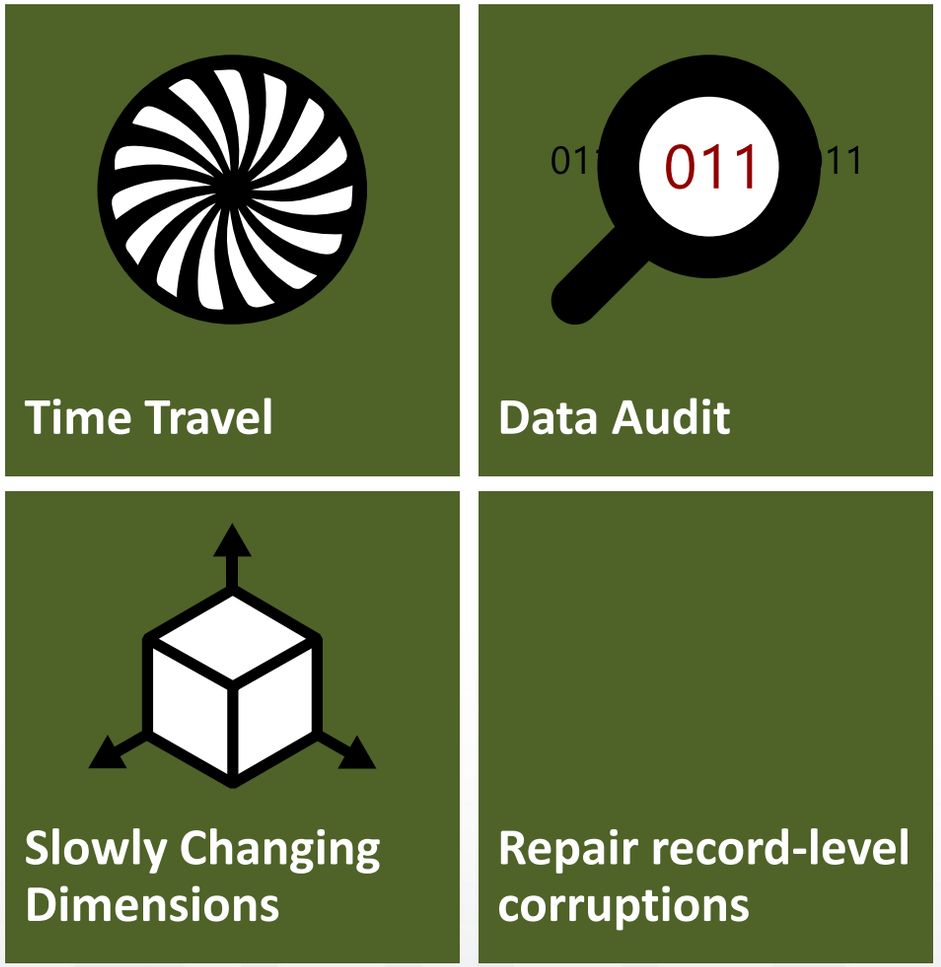
Great for consumption for Data Warehouses and/or auditing

Query the base table and return historical data transparently

SQL intersection

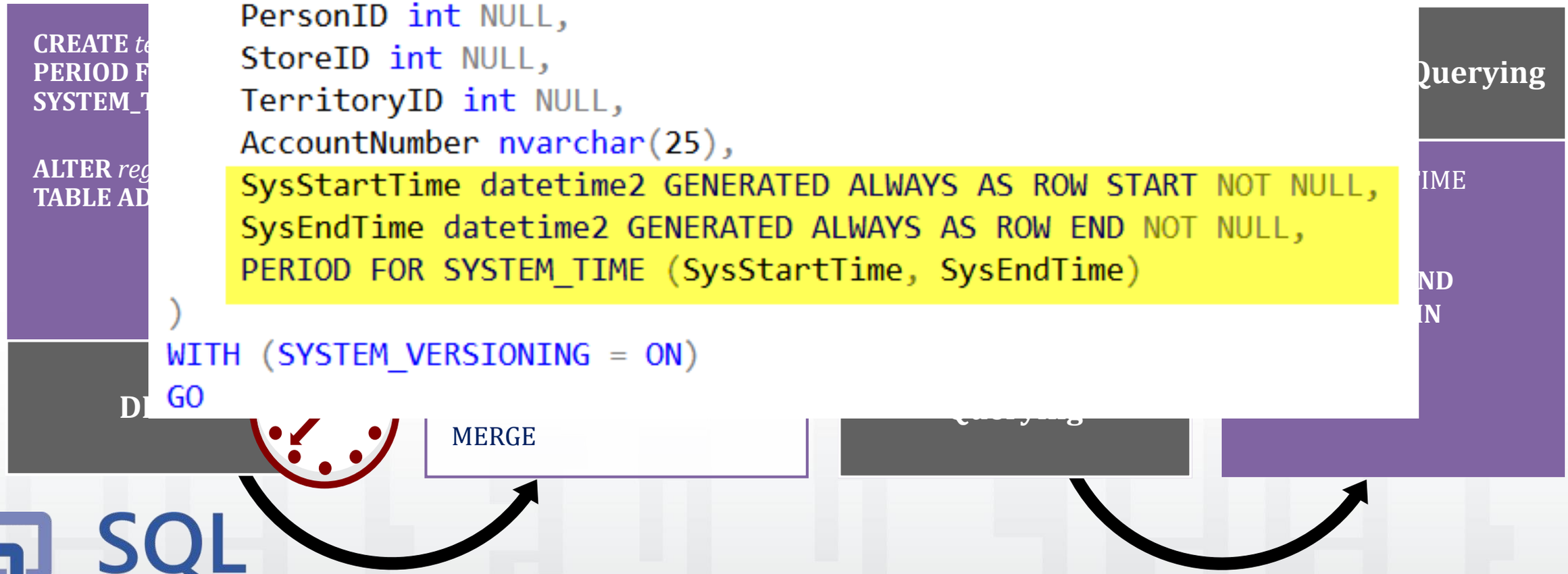
What is Temporal?

- **Real data sources are dynamic**
  - **Historical data may be critical to business success**
  - **Traditional databases fail to provide required insights**
- **Workarounds are…**
  - **Complex, expensive, limited, inflexible, inefficient**
- **SQL Server 2016 makes life easy**
  - **No change in programming model**
  - **Return data as it existed at a certain point in time**

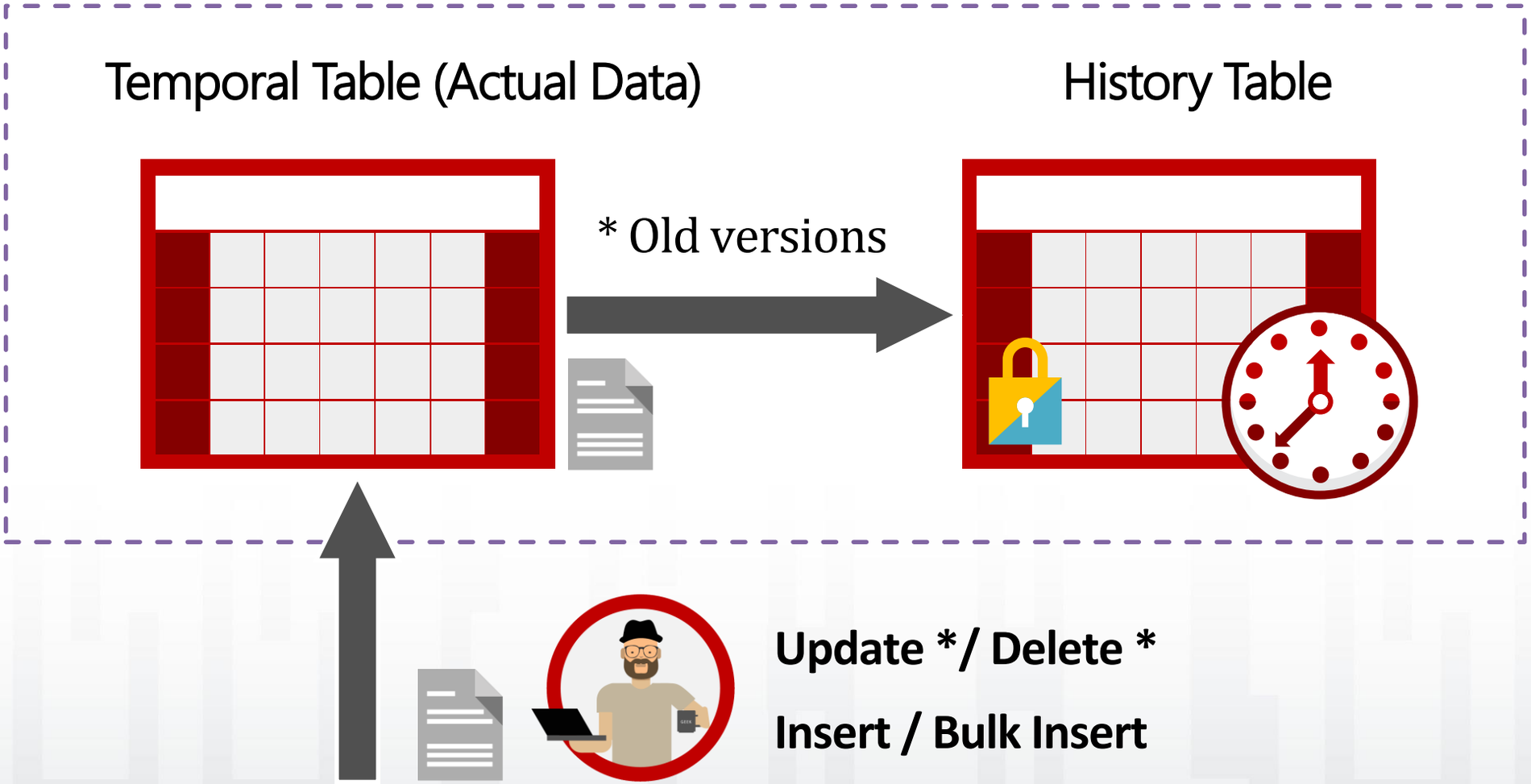


Getting Started with Temporal

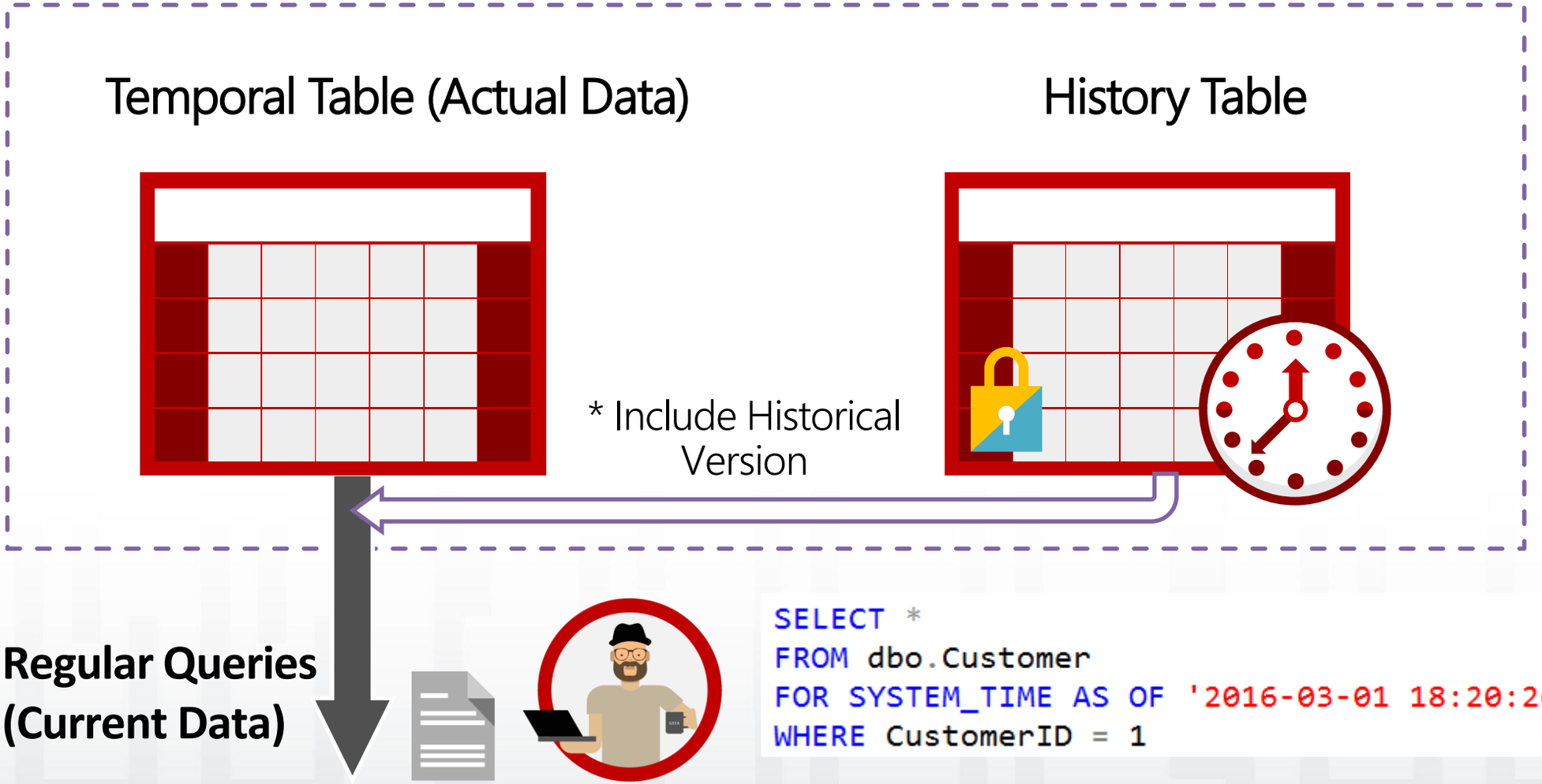
```
CREATE TABLE dbo.Customer
(
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    PersonID int NULL,
    StoreID int NULL,
    TerritoryID int NULL,
    AccountNumber nvarchar(25),
    SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL,
    SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
)
WITH (SYSTEM_VERSIONING = ON)
GO
```



How System Time Works



How System Time Works




```
SELECT *
FROM dbo.Customer
FOR SYSTEM_TIME AS OF '2016-03-01 18:20:26.3997304'
WHERE CustomerID = 1
```

Demo

Temporal Tables

SQL intersection

Development and Admin Enhancements

SQL intersection

Backup and Restore Tracing

- A new XE for backup and restore progress
- Replaces trace flags
- Allows you to keep track of how long each step in the process takes

SQL2016CTP\CTP22 - backup: Live Data

Displaying 31 Events

name	timestamp	trace_message
backup_restore_progress_trace	2015-09-10 01:45:38.4766428	BACKUP DATABASE started
backup_restore_progress_trace	2015-09-10 01:45:38.4766670	Opening the database with S lock
backup_restore_progress_trace	2015-09-10 01:45:38.4767182	Acquiring bulk-op lock on the database
backup_restore_progress_trace	2015-09-10 01:45:38.4767266	Synchronizing with other operations on the database is complete
backup_restore_progress_trace	2015-09-10 01:45:38.4823734	Opening the backup media set
backup_restore_progress_trace	2015-09-10 01:45:38.4836808	The backup media set is open
backup_restore_progress_trace	2015-09-10 01:45:38.4837417	Preparing the media set for writing
backup_restore_progress_trace	2015-09-10 01:45:38.4971657	The media set is ready for backup
backup_restore_progress_trace	2015-09-10 01:45:38.4971770	Effective options: Checksum=0, Compression=0, Encryption=0, Buffe
backup_restore_progress_trace	2015-09-10 01:45:38.4972152	Clearing differential bitmaps
backup_restore_progress_trace	2015-09-10 01:45:38.5341285	Differential bitmaps are cleared
backup_restore_progress_trace	2015-09-10 01:45:38.5341356	Writing a checkpoint
backup_restore_progress_trace	2015-09-10 01:45:39.2769770	Checkpoint is complete (elapsed = 742 ms)
backup_restore_progress_trace	2015-09-10 01:45:39.2770236	Start LSN: 47:202:176, SERepl LSN: 0:0:0

SQL intersection

New Cardinality Estimation

- Estimate of number of rows returned by each operation
- Based on Statistics metadata
- Fixed estimates (assumptions) when no histograms are available
- Query optimization is inherently unpredictable and very sensitive to cardinality estimations
- Cardinality estimation influences the cost, which is an indicator of response time
- Poor estimations can result in inefficient plans

SQL intersection

Why Change Cardinality Estimation?

Business Case

- CE engine was the same since 7.0
- Many changes under trace flag (Trace flag 4199 for Optimizer)
- Goal was to architecturally enhance the quality of cardinality estimation for a broad range of queries and workloads: OLTP, DW and DSS
- Create better plans for most cases, especially complex queries
- Make performance smoother and more predictable

Ramifications

- Can result in worse plans in edge cases
- Can be disabled, New CE is side-by-side.

New Memory Grant Hints

- **MAX_GRANT_PERCENT**
  - The maximum memory grant size in PERCENT. The query is guaranteed not to exceed this limit. The actual limit can be lower if the resource governor setting is lower than this.
- **MIN_GRANT_PERCENT**
  - The minimum memory grant size in PERCENT = % of default limit. The query is guaranteed to get MAX(required memory, min grant) because at least required memory is needed to start a query.

Maintenance Plans

- More options in Maintenance plans that were previously only available in TSQL
- Database Integrity Checks – options for Physical Only and Tablock
- Rebuild Indexes – option for defining lock priority for online rebuild
- Backup Database – specify block size and max transfer size

SQL intersection

More tempdb!

- **Supports caching data with automatic, multiple TempDB files per instance in multi-core environments**
- **Reduces metadata and allocation contention for TempDB workloads, improving performance and scalability**
  - Better scanning algorithms for metadata contention
  - Less exclusive latching and more shared latching for allocations
- **Trace Flags 1117 and 1118 behavior enabled automagically for tempdb**
  - Only applied for tempdb by default – not all use databases

SQL intersection

Online Operations

- **ALTER COLUMN – now happens as an online operation rather than a blocking operation**
  - Uses online Index rebuild infrastructure
- **TRUNCATE WITH PARTITIONS**

New ALTER DATABASE Options

- LEGACY_CARDINALITY_ESTIMATION
- PARAMETER_SNIFFING
- FOR SECONDARY
- QUERY_OPTIMIZER_HOTFIXES
- MAXDOP
- CLEAR PROCEDURE_CACHE
- AUTOGROW_SINGLE_FILE
- AUTOGROW_ALL_FILES
- MIXED_PAGE_ALLOCATION

SQL intersection

New TSQL Functionality

- SPLIT_STRING
- STRING_ESCAPE
- AT TIME ZONE
- SESSION_CONTEXT
- DROP IF

SQL intersection

Indirect Checkpoint

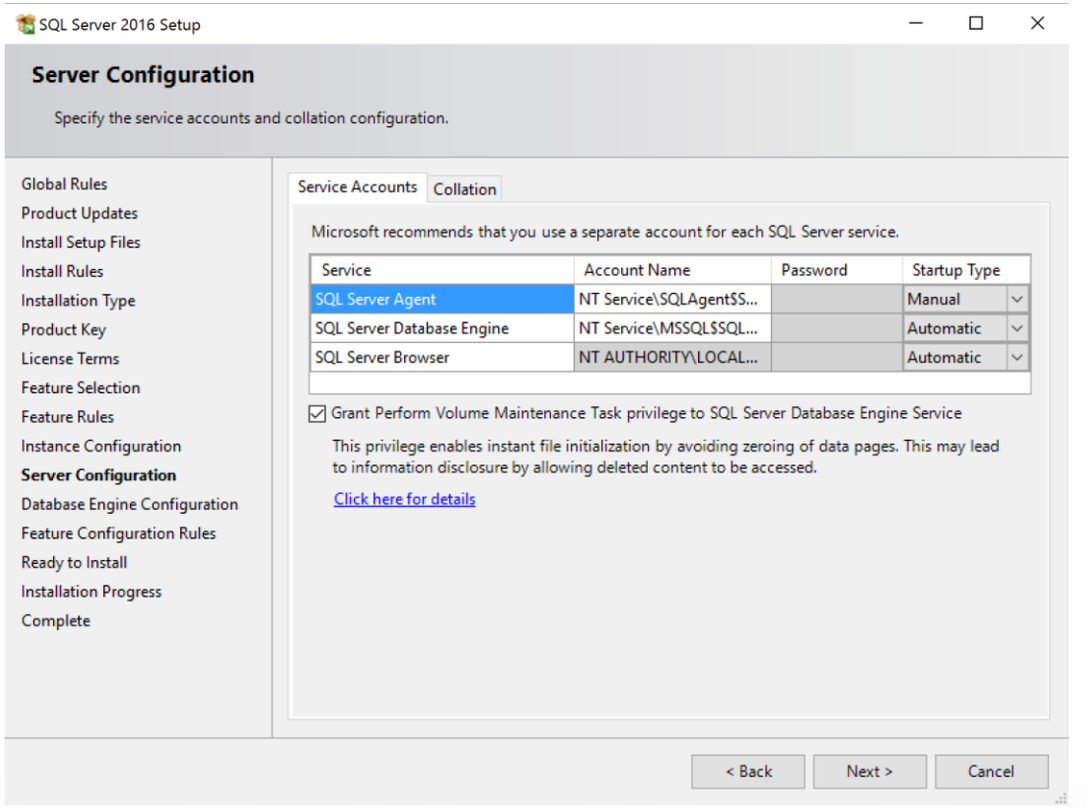
- **Keep track of dirty pages rather than scan the BUF structures to generate a list**
  - Modern disks suffer the Elevator Seek problem less
  - Perform flush once we hit a certain # of dirty pages
  - Auto checkpoint relies upon transaction threshold
- **More reliably control database recovery time**
- **Much better for systems with large memory footprints**
- **Default in SQL 2016**

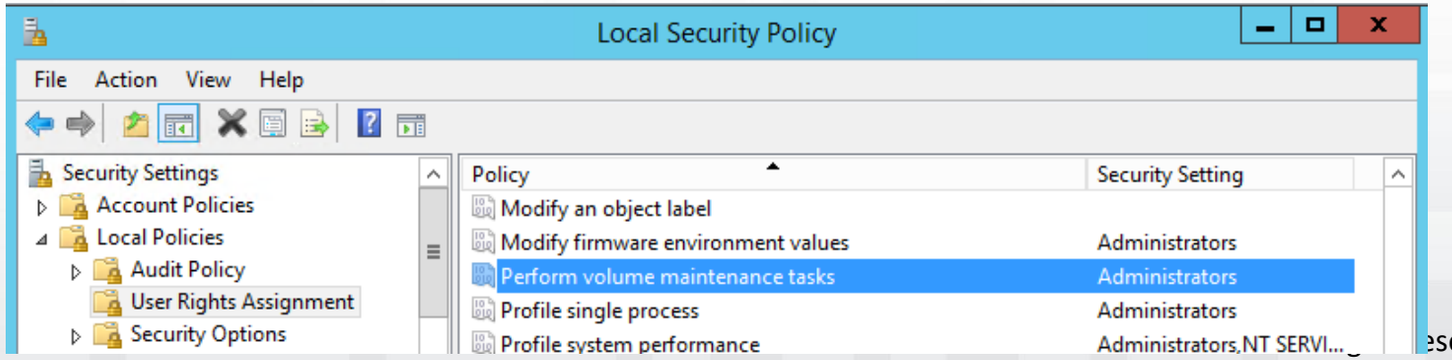
4TB Memory = ~500 million SQL Server BUF structures for older checkpoint
Indirect checkpoint for new database creation dirties ~ 250 BUF structures

SQL intersection

Instant File Initialization

- Grow a SQL *data file* instantly without needing to zero out disk space
- Included as part of the SQL 2016 Install process
- Can dramatically increase the speed of growth and DB creation
- Requires Perform Volume Maintenance Tasks permission





SQL intersection

AES-NI Chips

6 new chip instructions for AES

- **Intel Westmere chip set that introduces New Instructions to implement the AES encryption algorithm at the hardware level**

Availability Groups	• Log transport improvement
Transparent Data Encryption	• Overhead significantly reduced

SQL intersection

New DMV Goodness

sys.dm_exec_function_stats

- Returns aggregate performance statistics for cached functions. The view returns one row for each cached function plan, and the lifetime of the row is as long as the function remains cached

sys.dm_exec_session_wait_stats

- Returns information about all the waits encountered by threads that executed for each session. You can use this view to diagnose performance issues with the SQL Server session and also with specific queries and batches

sys.dm_exec_query_profiles

- Monitors query operator progress while it is executing

SQL intersection

Demo

New Programmatic Improvements

SQL intersection

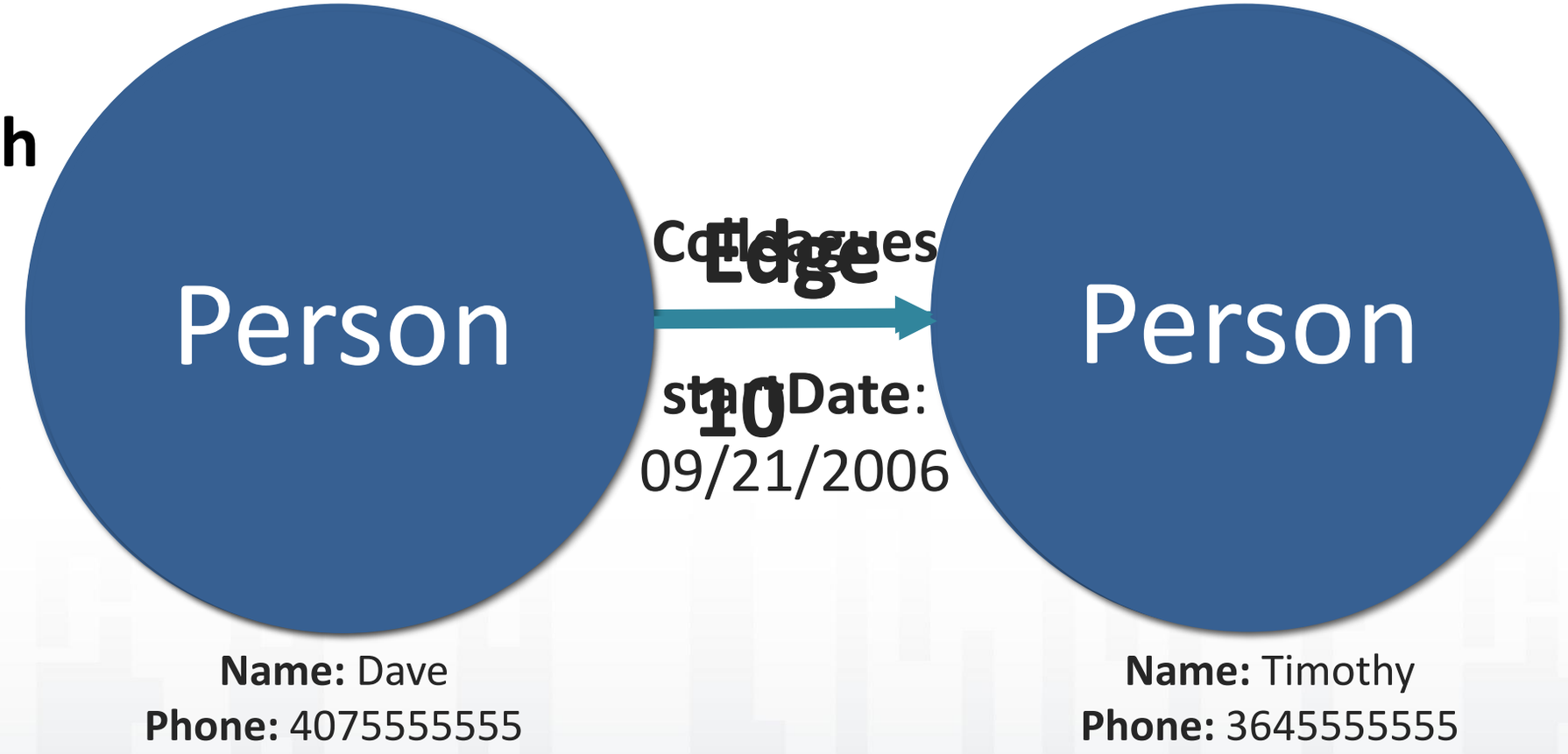
Graph Support in SQL Server 2017

SQL intersection

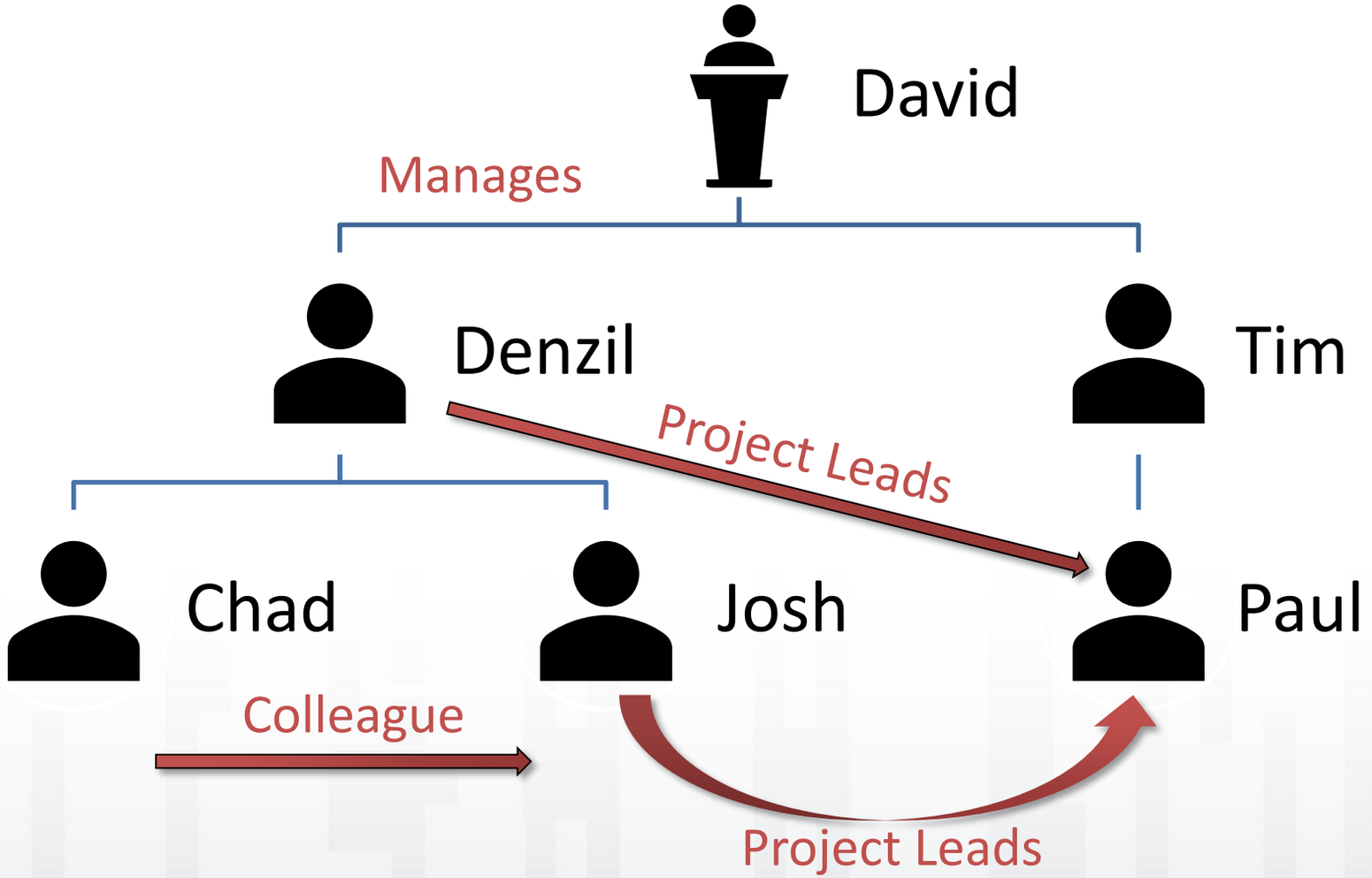
What is a Graph?

A graph is collection of *Nodes* and *Edges*

- **Undirected Graph**
- **Directed Graph**
- **Weighted Graph**
- ***Property Graph**



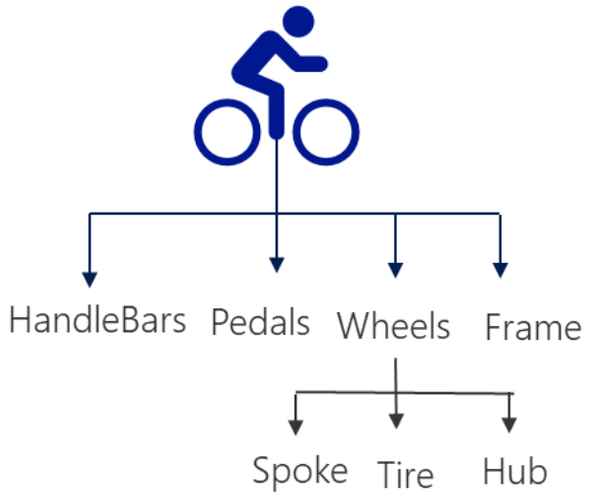
Why Graph Databases?



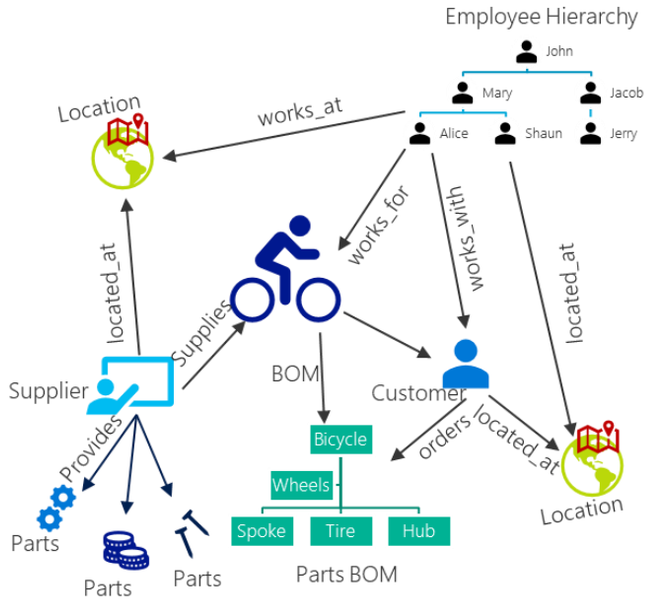
No one likes Paul…

SQL intersection

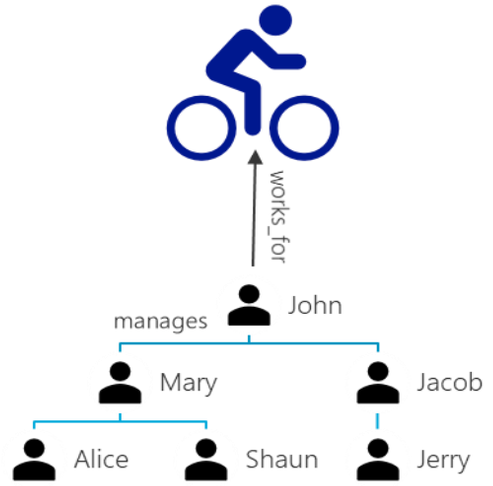
Typical Scenarios for Graph DBs



Hierarchical or interconnected data, entities with multiple parents.



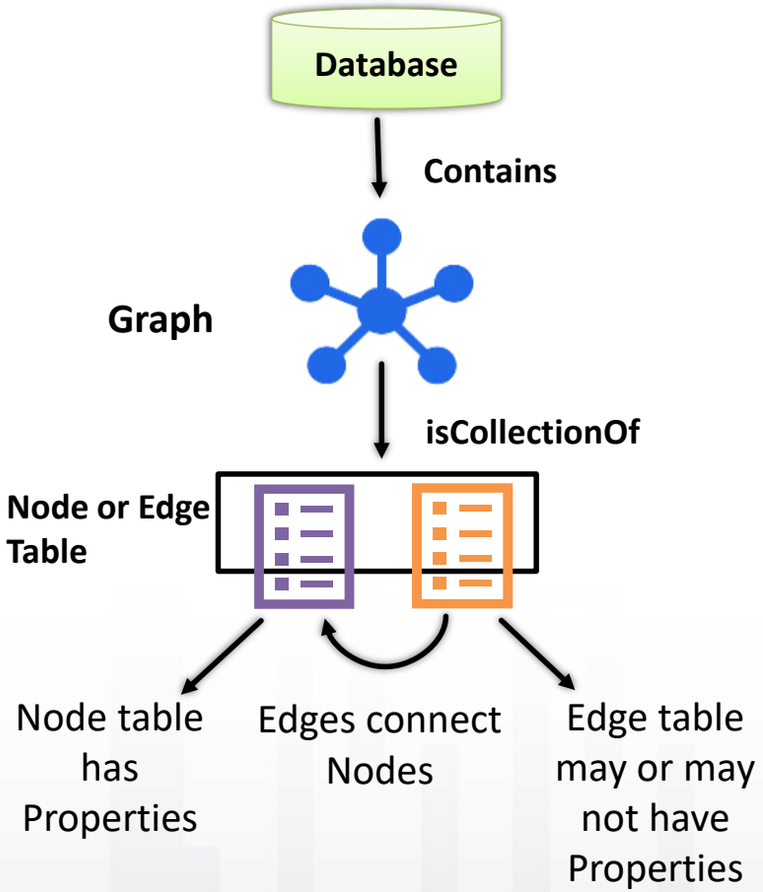
Complex many-to-many relationships. Organically grow connections as the business evolves.



Analyze interconnected data, materialize new information from existing facts. Identify non-obvious connections

Introducing SQL Graph

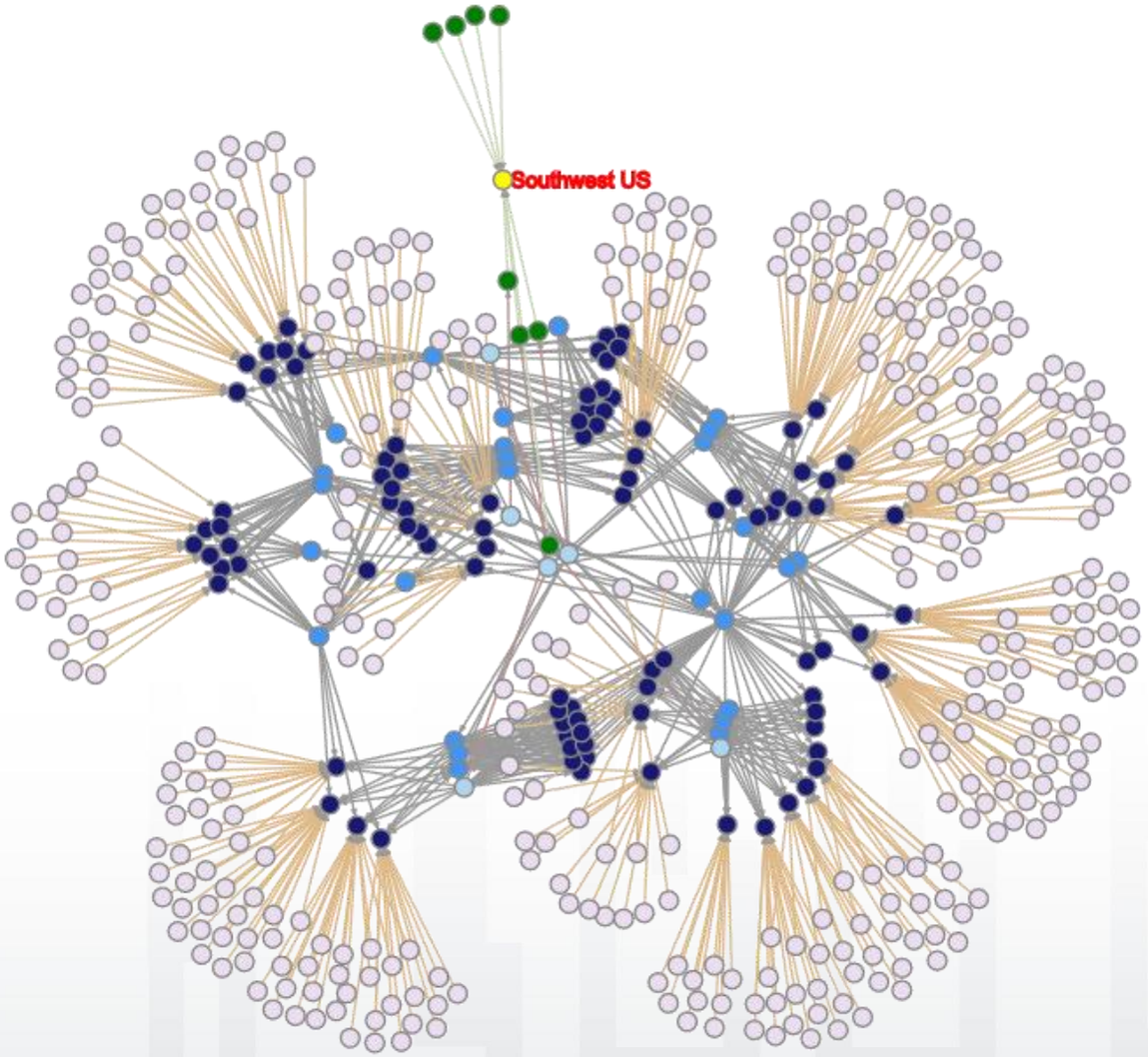
- **A collection of node and edge *tables* in the database**
- **Language Extensions**
  - DDL Extensions – create node/edge tables
  - Query Language Extensions – New built-in: MATCH, to support pattern matching and traversals
- **Tooling and Eco-system**



SQL intersection

Visualization

- There are multiple ways you can visualize your graph data
  - PowerBI
  - R package - iGraph
  - D3.js visualization

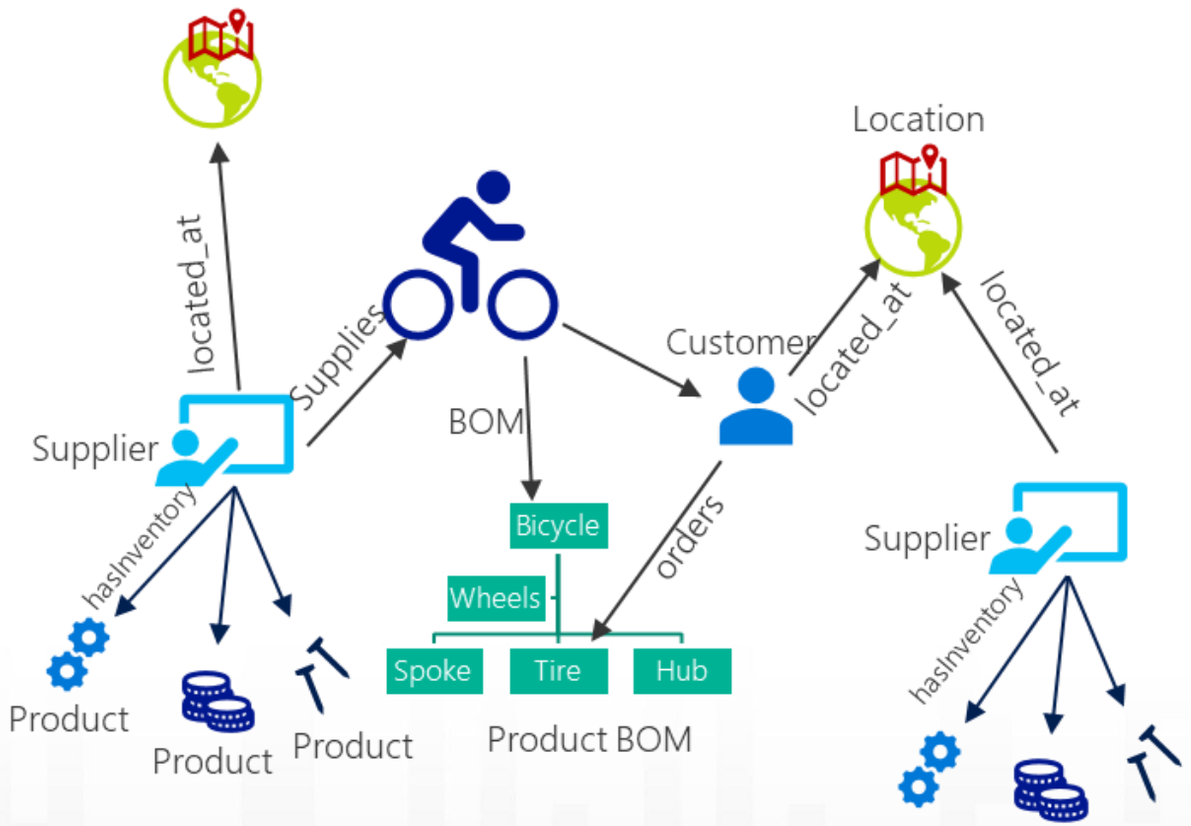


SQL intersection

DDL Extensions

- **Create Nodes (entities) and Edges (relationships)**
- **Properties associated with Nodes and Edges**

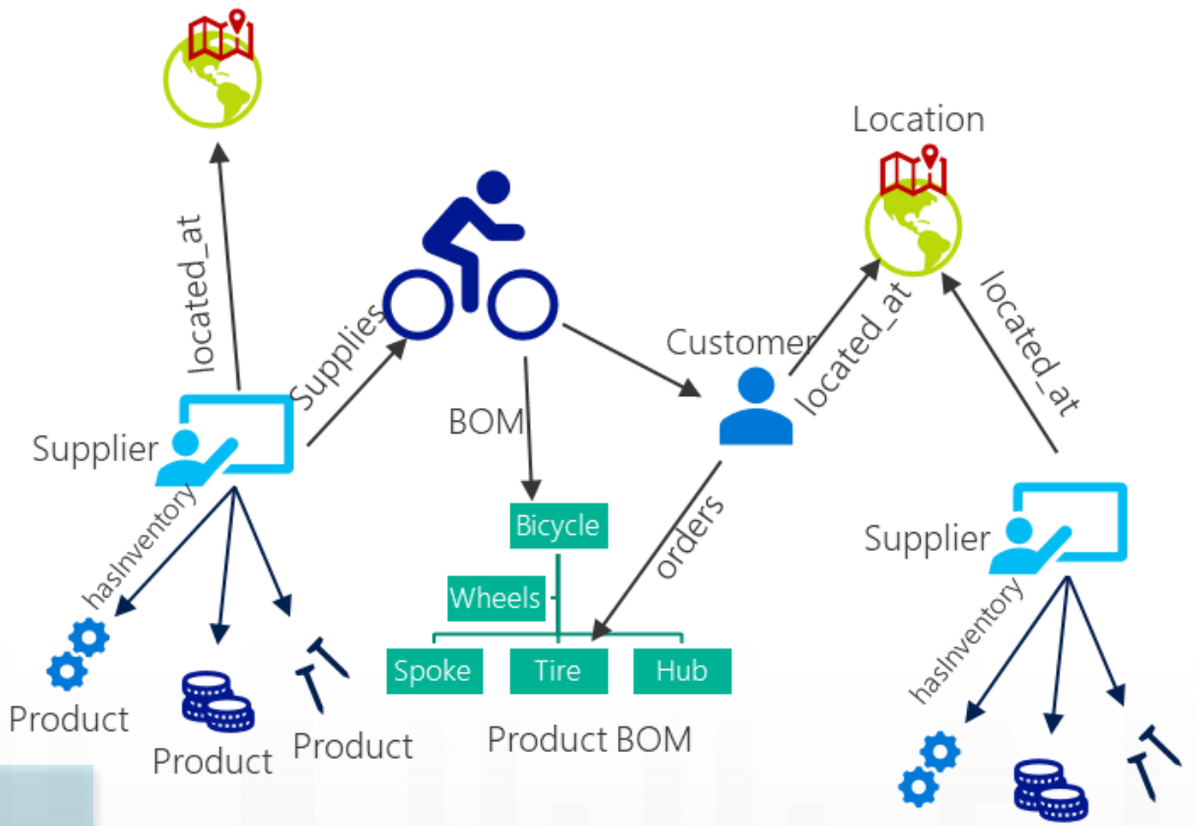
```
CREATE TABLE Product (ID INTEGER
PRIMARY KEY, name VARCHAR(100))
AS NODE;

CREATE TABLE Supplier (ID INTEGER
PRIMARY KEY, name VARCHAR(100))
AS NODE;

CREATE TABLE hasInventory AS EDGE;

CREATE   TABLE   located_at(address
varchar(100)) AS EDGE;
```



Query Language Extensions

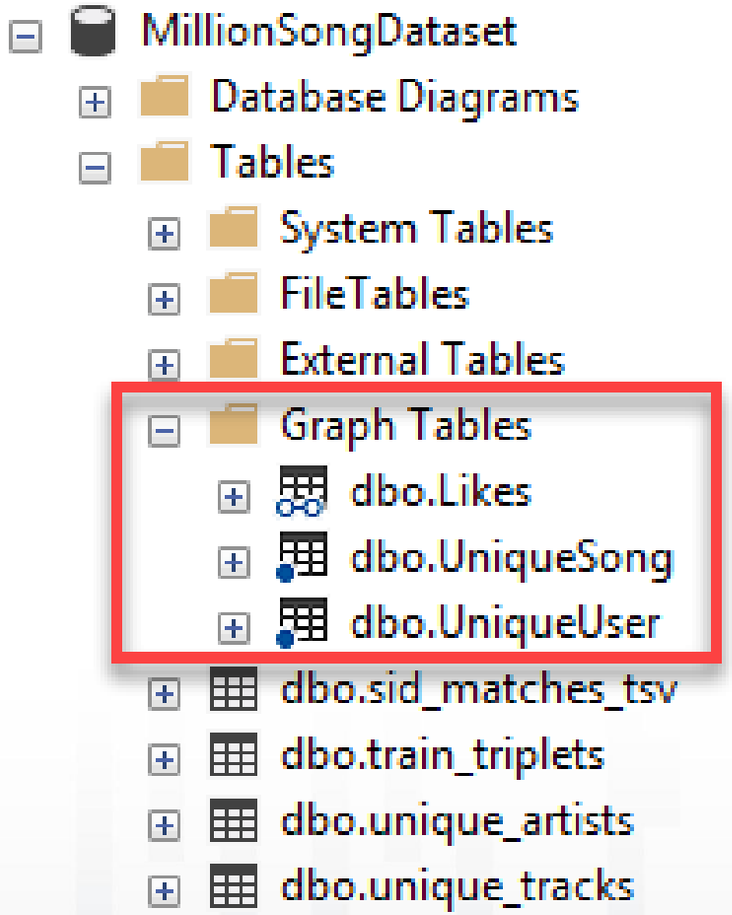
- **Multi-hop navigation and join-free pattern matching using MATCH predicate:**

```
SELECT Prod.name as ProductName,
    Sup.name as SupplierName
FROM Product Prod, Supplier Sup,
  hasInventory hasIn,
  located_at supp_loc,
  Customer Cus,
  located_at cust_loc,
  orders, location loc
WHERE
MATCH(
  cus-(orders)->Prod<-(hasIn)-Sup
  AND
  cus-(cust_loc)->location<-(supp_loc)-Sup)
```

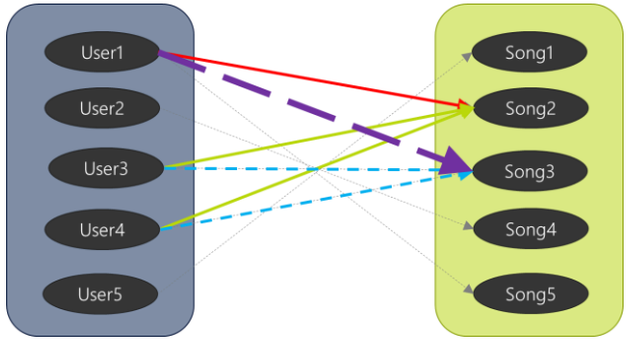


Integrated into the SQL Engine

- Existing tools all work out of the box, for example SSMS, backup and restore, import / export, etc.
- Queries can join existing tables and graph node / edge tables.
- Security and Compliance: Dynamic Data Masking, Row Level Security
- Available on SQL Server on Linux



SQL intersection



Demo

Graph Support in SQL Server 2017

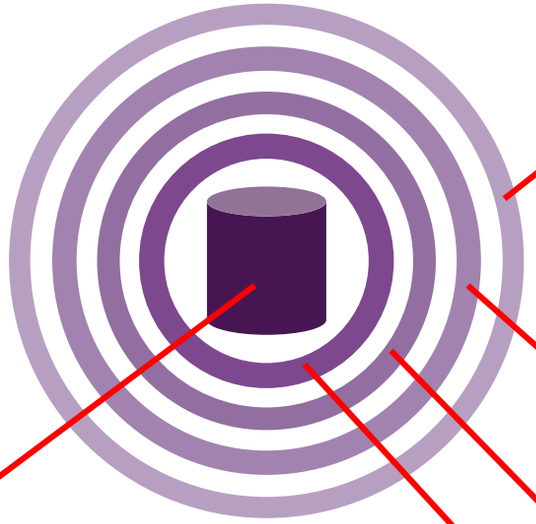
The Million Song Dataset

SQL intersection

Most Secure Database

Surrounded by layers of protection

NEW* As of SQL Server 2016



Identity

- Windows authentication
- Azure Active Directory auth. NEW*

Monitor activity

- SQL threat analytics NEW*
- SQL Server auditing

Control access

- Row-level security NEW*
- Dynamic data masking NEW*

Protect data

- Always encrypted NEW*
- Transparent data encryption

Secure Code

- Secure development lifecycle
- Least vulnerable last 6 years

Row-Level Security (RLS)

SQL intersection

The Need for Row Level Security

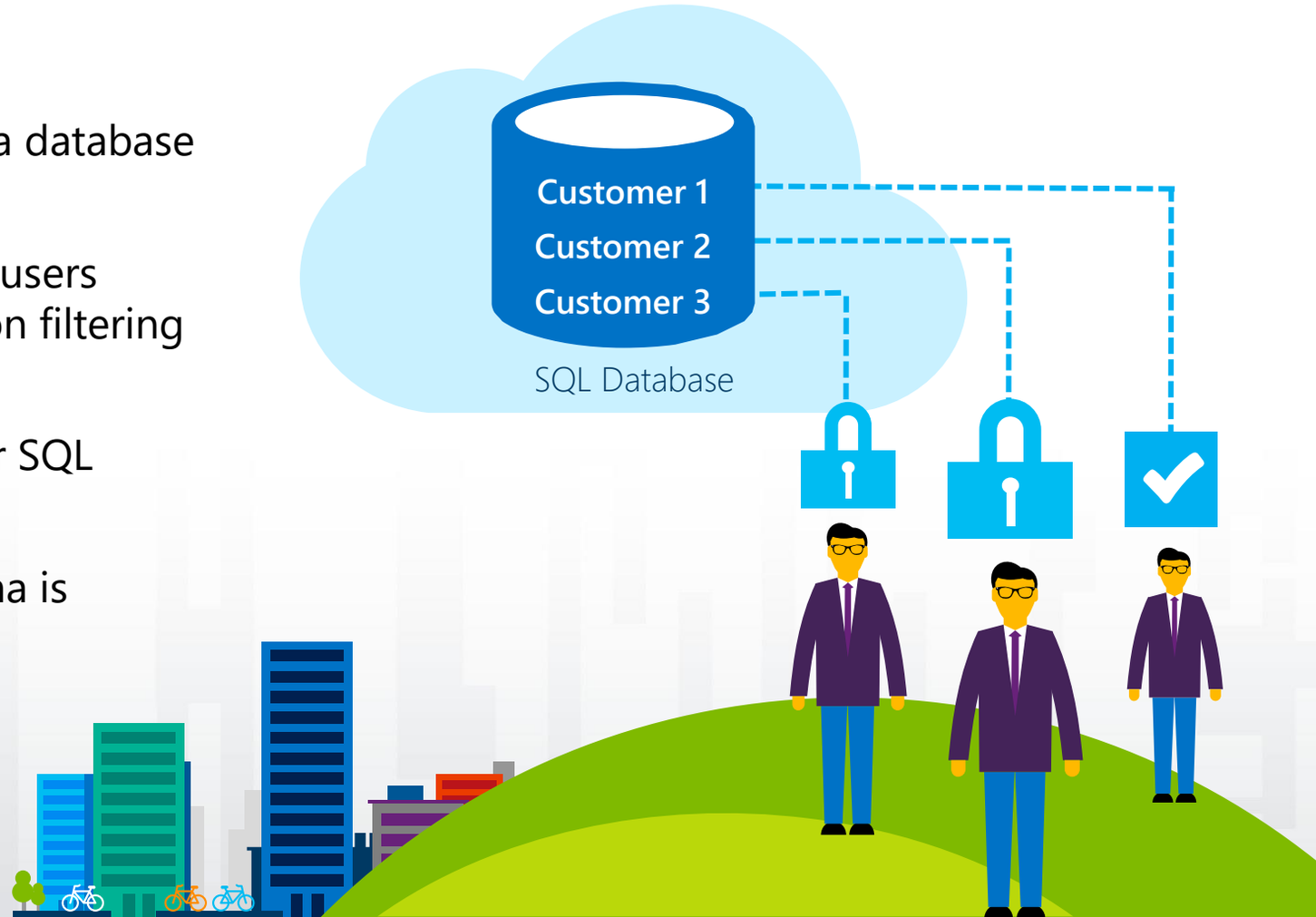
Protect data privacy by ensuring the right access across rows

Fine-grained access control over specific rows in a database table

Help prevent unauthorized access when multiple users share the same tables, or to implement connection filtering in multitenant applications

Administer via SQL Server Management Studio or SQL Server Data Tools

Enforcement logic inside the database and schema is bound to the table



SQL intersection

Benefits of Row-Level Security (RLS)

Fine-grained access control

Keeping multitenant databases secure by limiting access by other users who share the same tables

Application transparency

RLS works transparently at query time, no app changes needed

Compatible with RLS in other leading products

Centralized security logic

Enforcement logic resides inside database and is schema-bound to the table it protects providing greater security. Reduced application maintenance and complexity

Store data intended for many consumers in a single database/table while at the same time restricting row-level read and write access based on users' execution context.

RLS Concepts

Predicate function

User-defined inline table-valued function (iTVF) implementing security logic

Can be arbitrarily complicated, containing joins with other tables

Security predicate

Binds a predicate function to a particular table, applying it for all queries

Two types: Filter predicates and Block predicates

Security policy

Collection of security predicates for managing security across multiple tables

Performance?

Inline functions get optimized to provide comparable performance to views—as if the logic were directly embedded in the original query statement.

```
CREATE SECURITY POLICY mySecurityPolicy
    ADD FILTER PREDICATE dbo.fn_securitypredicate(wing, startTime, endTime)
    ON dbo.patients
```

SQL intersection

How Row-Level Security works

1) Policy manager creates filter predicate and security policy in T-SQL, binding the predicate to the Patients table
2) Nurse, using an app, selects from Patients table
3) Security Policy transparently rewrites query to apply filter predicate

Nurse

(Employee Id = 100986
AND Wing = ‘a1’)

Application

```
SELECT Name FROM Patients
```

Database

Security Policy

Filter Predicate:
INNER JOIN…

Patients

Policy Manager

```
CREATE FUNCTION dbo.fn_securitypredicate(@wing int)
    RETURNS TABLE WITH SCHEMABINDING AS
SELECT Name FROM Patients
   SEMIJOIN APPLY dbo.fn_securitypredicate(patients.Wing);
        ON (d.EmpId = e.EmpId)
SELECT Patients.Name FROM Patients,
   StaffDuties d INNER JOIN Employees e ON (d.EmpId = e.EmpId)
   WHERE e.UserSID = SUSER_SID() AND Patients.wing = d.Wing;

    ADD BLOCK PREDICATE dbo.fn_securitypredicate(Wing) ON Patients
    WITH (STATE = ON)
```

How Row-Level Security works

4) Filtered rows are returned and nurse sees only the patients on her wing

Nurse

(Employee Id = 100986
AND Wing = ‘a1’)

Application

Database

Security Policy

Filter Predicate: INNER JOIN…

Patients

Policy Manager

Patients		
Wing	PatientId	Name
a1	100-207	Jones, Walker
b1	201-677	Smith, Weston
b2	898-045	Jarvis, Anne
a1	009-451	Rickmann, Earnst
b2	922-769	Gray, Jim
a1	801-345	Polka, Dorothy

StaffDuties	
EmployeeId	Wing
100986	a1
206985	a2
345906	a3
331225	b1

Create a Security Policy

```
--Create a security policy that adds this function as a filter
predicate and a block predicate on Sales.
CREATE SECURITY POLICY Security.SalesFilter
ADD FILTER PREDICATE Security.fn_securitypredicate(AppUserId)
  ON dbo.Sales,
ADD BLOCK PREDICATE Security.fn_securitypredicate(AppUserId)
  ON dbo.Sales AFTER INSERT
WITH (STATE = ON);

-- Create a new schema and predicate function, which will use the
application user ID stored in SESSION_CONTEXT to filter rows.
CREATE SCHEMA Security; GO CREATE FUNCTION
Security.fn_securitypredicate(@AppUserId int)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS fn_securitypredicate_result
WHERE DATABASE_PRINCIPAL_ID() =
DATABASE_PRINCIPAL_ID('AppUser') AND
CAST(SESSION_CONTEXT(N'UserId') AS int) = @AppUserId;
GO
```

Creates a security policy for row-level security

The following examples demonstrate the use of the CREATE SECURITY POLICY syntax

For an example of a complete security policy scenario, see Row-Level Security

Security Predicates

RLS supports two types of security predicates

- **Filter predicates silently filter rows available to read operations (SELECT, UPDATE, and DELETE)**
- **Block predicates explicitly block write operations (AFTER INSERT, AFTER UPDATE, BEFORE UPDATE, BEFORE DELETE) that violate predicate***

Access to row-level data in table is restricted by security predicate defined as inline table-valued function, which is invoked and enforced by security policy

- **For filter predicates, no indication to application that rows have been filtered from result set; if all rows are filtered, a null set will be returned**
- **For block predicates, any operations that violate predicate will fail with error**

SQL intersection

Demo

Row Level Security (RLS)

SQL intersection

Dynamic Data Masking

SQL intersection

Dynamic Data Masking

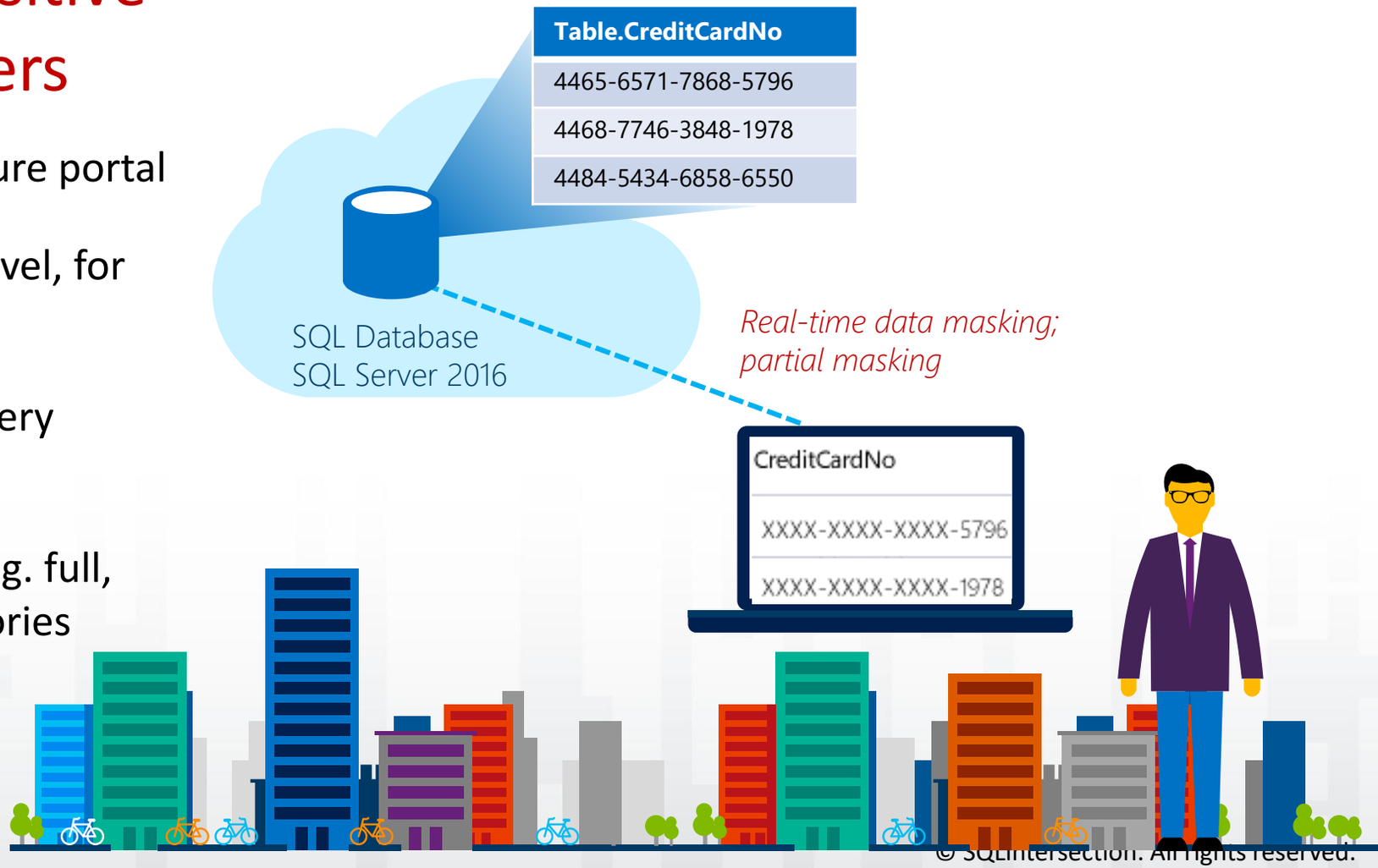
Prevent the abuse of sensitive data by hiding it from users

Configuration made easy in the new Azure portal

Policy-driven at the table and column level, for a defined set of users

Data masking applied in real-time to query results based on policy

Multiple masking functions available (e.g. full, partial) for various sensitive data categories (credit card numbers, SSN, etc.)



SQL intersection

Benefits of Dynamic Data Masking

Regulatory compliance

A strong demand for applications to meet **privacy standards** recommended by regulating authorities

Sensitive data protection

Protects against unauthorized access to sensitive data in the application, and against exposure to developers or DBAs who need access to the production database

Agility and transparency

Data is masked on the fly, with underlying data in the database remaining intact. Transparent to the application and applied according to user privilege

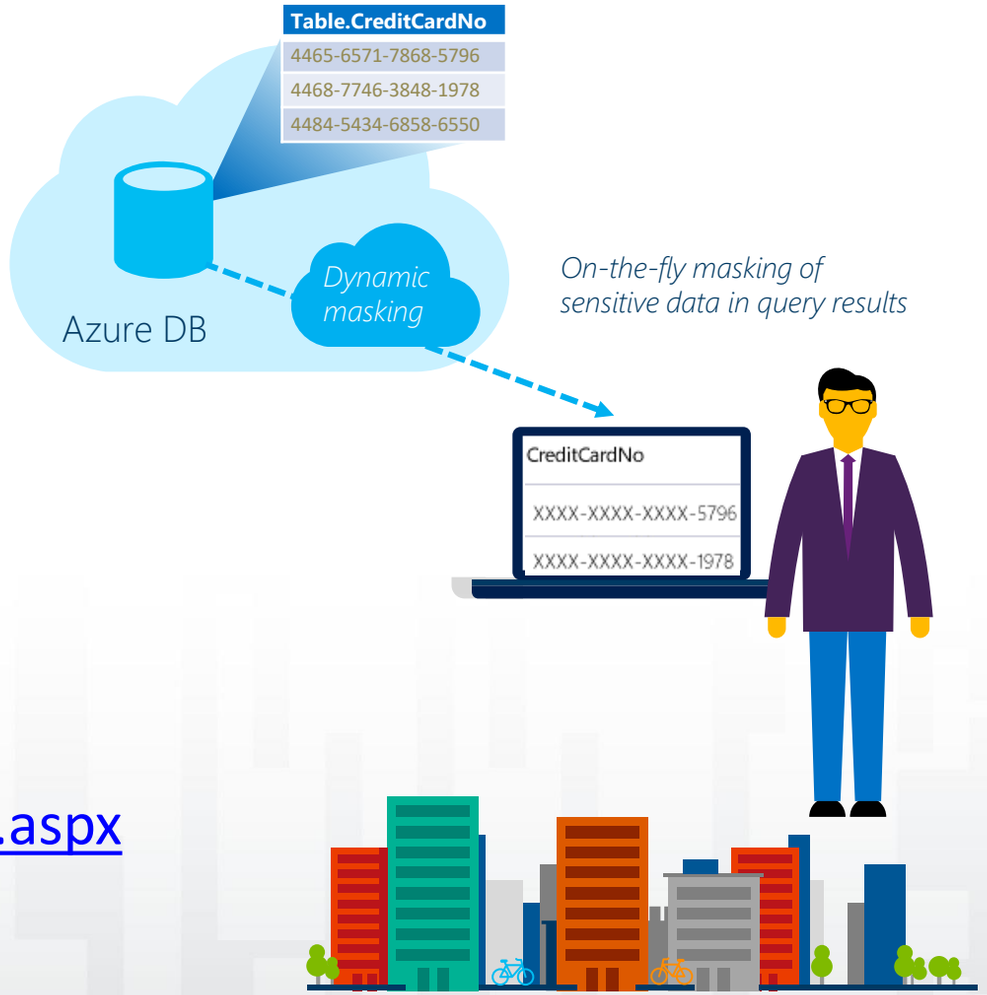
Limit access to sensitive data by defining policies to obfuscate specific database fields, without affecting the integrity of the database.

How Dynamic Data Masking Works

Limit sensitive data exposure by obfuscating it to non-privileged users

- On-the-fly obfuscation of data in query results
- Policy-driven on the table and column
- Multiple masking functions available for various sensitive data categories
- Flexibility to define a set of privileged logins for un-masked data access
- By default, database owner is unmasked

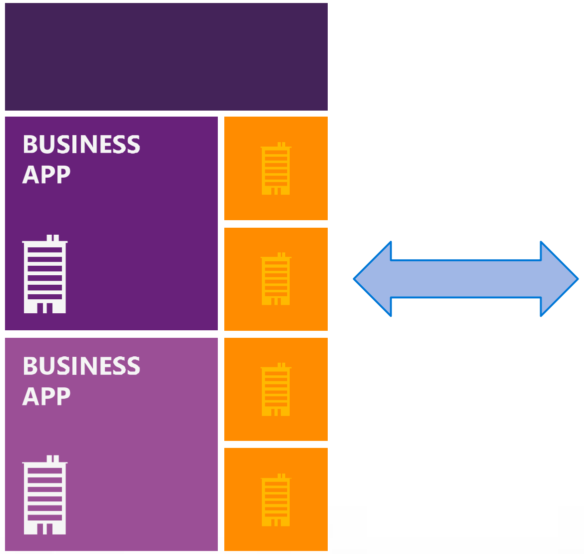
https://msdn.microsoft.com/en-us/library/mt130841.aspx



SQL intersection

How Dynamic Data Masking Works

1) Security officer defines dynamic data masking policy in T-SQL over sensitive data in Employee table
2) App user selects from Employee table
3) Dynamic data masking policy obfuscates the sensitive data in the query results




```
ALTER TABLE [Employee] ALTER COLUMN [SocialSecurityNumber]
ADD MASKED WITH (FUNCTION = 'partial(0,"XXX-XX-",2)')

ALTER TABLE [Employee] ALTER COLUMN [Email]
ADD MASKED WITH (FUNCTION = 'EMAIL()')

ALTER TABLE [Employee] ALTER COLUMN [Salary]
ADD MASKED WITH (FUNCTION = 'RANDOM(1,20000)')

GRANT UNMASK to hrsupervisor
```

Security Officer

```
SELECT [Name],
       [SocialSecurityNumber],
       [Email],
       [Salary]
FROM [Employee]
```

non-privileged login

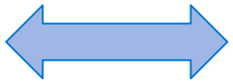
	First Name	Social Security Number	Email	Salary
1	LILA	XXX-XX-XX37	lXX@XXXX.net	8940
2	JAMIE	XXX-XX-XX14	jXX@XXXX.com	19582
3	SHELLEY	XXX-XX-XX28	sXX@XXXX.net	3713
4	MARCELLA	XXX-XX-XX65	mXX@XXXX.net	11572
5	GILBERT	XXX-XX-XX87	gXX@XXXX.net	4487

hrsupervisor login

	First Name	Social Security Num...	Email	Salary
1	LILA	758-10-9637	lila.barnett@comcast.net	1012794
2	JAMIE	113-29-4314	jamie.brown@ntlworld.com	1025713
3	SHELLEY	550-72-2028	shelley.lynn@charter.net	1040131
4	MARCELLA	903-94-5665	marcella.estrada@comcast.net	1040753
5	GILBERT	376-79-4787	gilbert.juarez@verizon.net	1041308

Can be Combined with Row Level Security

4) Data masking obscures columns, row level security filters rows




```
ALTER TABLE [Employee] ALTER COLUMN [SocialSecurityNumber]
ADD MASKED WITH (FUNCTION = 'partial(0,"XXX-XX-",2)')

ALTER TABLE [Employee] ALTER COLUMN [Email]
ADD MASKED WITH (FUNCTION = 'EMAIL()')

ALTER TABLE [Employee] ALTER COLUMN [Salary]
ADD MASKED WITH (FUNCTION = 'RANDOM(1,20000)')

GRANT UNMASK to hrsupervisor
```

Security Officer

```
SELECT [Name],
       [SocialSecurityNumber],
       [Email],
       [Salary]
FROM [Employee]
```

non-privileged login

	First Name	Social Security Number	Email	Salary
1	LILA	XXX-XX-XX37	lXX@XXXX.net	8940
2	JAMIE	XXX-XX-XX14	jXX@XXXX.com	19582
3	SHELLEY	XXX-XX-XX28	sXX@XXXX.net	3713
4	MARCELLA	XXX-XX-XX65	mXX@XXXX.net	11572
5	GILBERT	XXX-XX-XX87	gXX@XXXX.net	4487

hrsupervisor login

	First Name	Social Security Num...	Email	Salary
1	LILA	758-10-9637	lila.barnett@comcast.net	1012794
2	JAMIE	113-29-4314	jamie.brown@ntlworld.com	1025713
3	SHELLEY	550-72-2028	shelley.lynn@charter.net	1040131
4	MARCELLA	903-94-5665	marcella.estrada@comcast.net	1040753
5	GILBERT	376-79-4787	gilbert.juarez@verizon.net	1041308

Demo

Dynamic Data Masking (DDM)

SQL intersection

Always Encrypted

SQL intersection

What is Always Encrypted?

Specific data kept an encrypted state at all times	• Encryption occurs at the column level
Keeps privileged users from accessing sensitive data	• This includes Database Administrators/sysadmins
Tailored for applications that interact with sensitive data	• Not great for applications that need to report off of sensitive data
Requires a certificate	• Can be a self-signed certificate

SQL intersection

Always Encrypted cont.

Driver on client encrypts/ decrypts data	• Requires .NET Framework 4.6
Efficient searches can be done on encrypted data	• Requires a specific type of encryption
Confidently store data in Azure	• Azure Key Vault
Reduce surface attack area	• Data only encrypted/decrypted at the client layer

SQL intersection

Separation of Duties

- **Traditionally the DBA has access to *ALL* sensitive data – even if it is encrypted**
  - No way to prevent this in previous versions unless application rolls custom encryption
- **A security admin can set up the necessary keys while the DBA sets up the encryption**
  - Keeps the DBA from being able to see the data since the SQL Server will never store the actual certificate
  - Powershell cmdlets exist to separate these duties

SQL intersection

Benefits of Always Encrypted

Prevents Data Disclosure

Client-side encryption of sensitive data using keys that are *never* given to the database system.

Happens at the column level in SQL

Queries on Encrypted Data

Support for equality comparison, incl. join, group by and distinct operators.

Application Transparency

Minimal application changes via server and client library enhancements.

The application doesn't realize the data is encrypted inside SQL

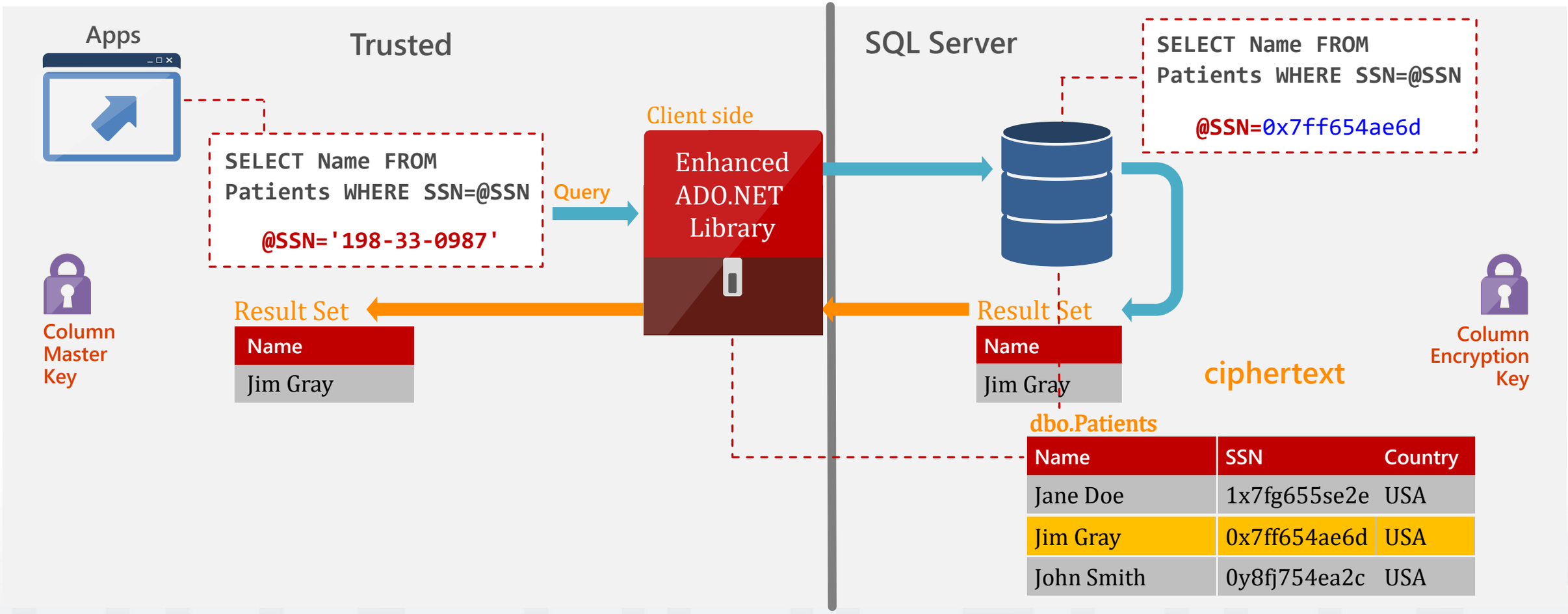
Allows customers to securely store **sensitive** *data outside of their trust boundary.*

Data remains protected from high-privileged, yet unauthorized users.

SQL intersection

Always Encrypted

Help protect data at rest and in motion, on-premises & cloud



Types of Encryption for Always Encrypted

Randomized encryption

- Uses a method that encrypts data in a less predictable manner
- Allows for transparent retrieval of encrypted data but NO operations
- More secure

Deterministic encryption

- Always generates the same encrypted value for any given plain text value
- Allows for transparent retrieval of encrypted data AND comparison operators, GROUP BY, etc

Randomized encryption

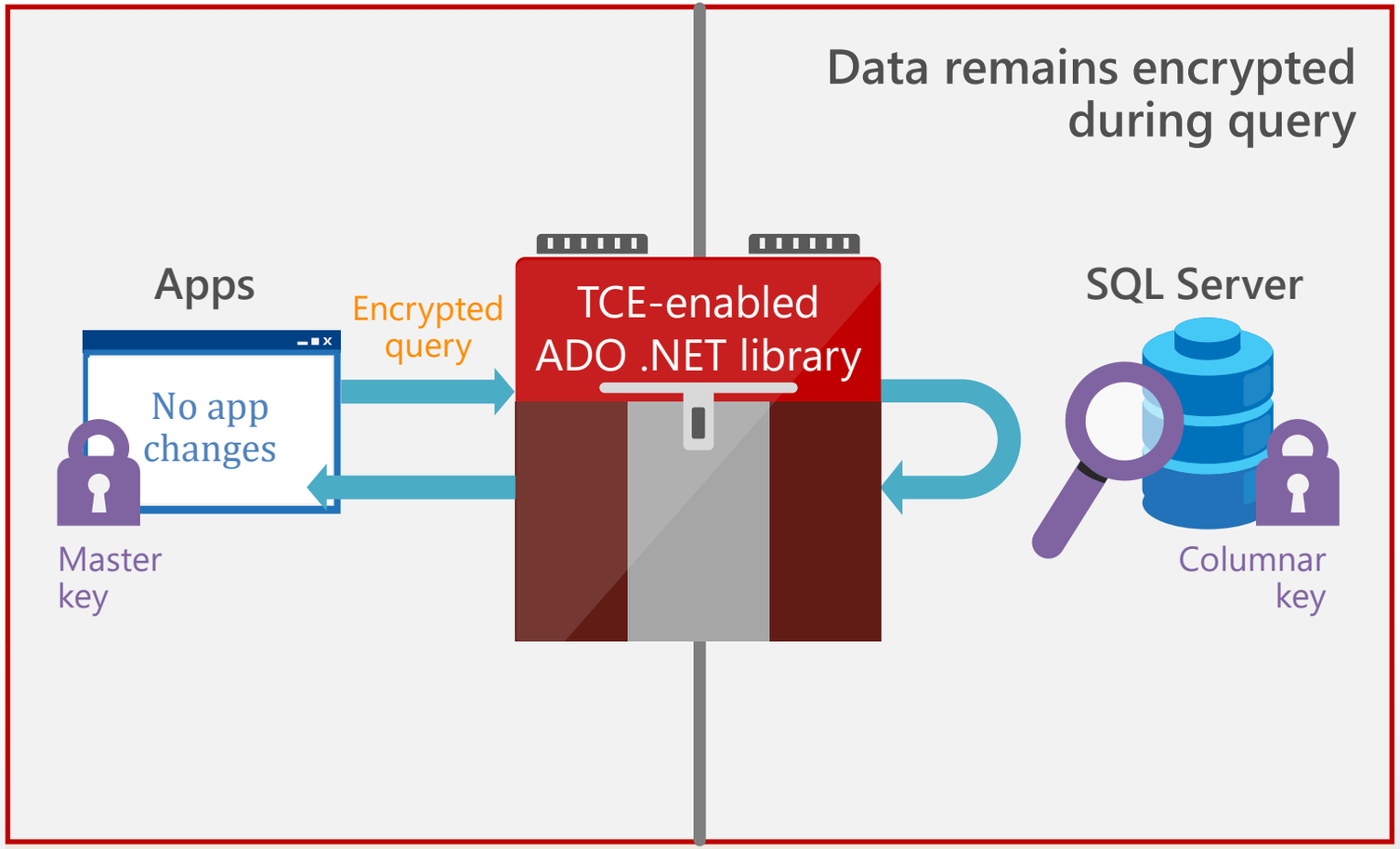
- Encrypt('123-45-6789') = 0x17cfd50a
- Repeat: Encrypt('123-45-6789') = 0x9b1fcf32

Deterministic encryption

- Encrypt('123-45-6789') = 0x85a55d3f
- Repeat: Encrypt('123-45-6789') = 0x85a55d3f

SQL intersection

Summary: Always Encrypted



Capability

- ADO.Net client library provides transparent client-side encryption, while SQL Server executes T-SQL queries on encrypted data

Benefits

- Sensitive data remains encrypted and queryable at all times on-premises & cloud.
- Unauthorized users never have access to data or keys
- No application changes – requires a few database changes

Protect data at rest and in motion, on-premises & cloud

Demo

Always Encrypted

SQL intersection

SQL Server 2016+ Mission-Critical Summary

Performance

Operational analytics
Insights on operational data; Works with in-memory OLTP and disk-based OLTP

In-memory OLTP enhancements
Greater T-SQL surface area, terabytes of memory supported, and greater number of parallel CPUs

Query data store
Monitor and optimize query plans

Temporal database support
Query data as points in time

DMV Improvements

Cardinality Estimates

Security

Always encrypted
Sensitive data remains encrypted at all times with ability to query

Row-level security
Apply fine-grained access control to table rows

Dynamic data masking
Real-time obfuscation of data to prevent unauthorized access

Advanced Threat Detection
Ability to find unusual login patterns, track usage behavior in an auditing database, track SQL injection vulnerability, and more

Other enhancements
Audit success/failure of database operations

TDE support for storage of in-memory OLTP tables

Enhanced auditing for OLTP with ability to track history of record changes

Availability

Enhanced AlwaysOn
Three synchronous replicas for auto failover across domains

Round robin load balancing of replicas

Automatic failover based on database health

DTC for transactional integrity across database instances with AlwaysOn

Support for SSIS with AlwaysOn

StretchDB
Archive historical data transparently and securely to Azure

Queries stretch across local data as well as Azure data

Scalability

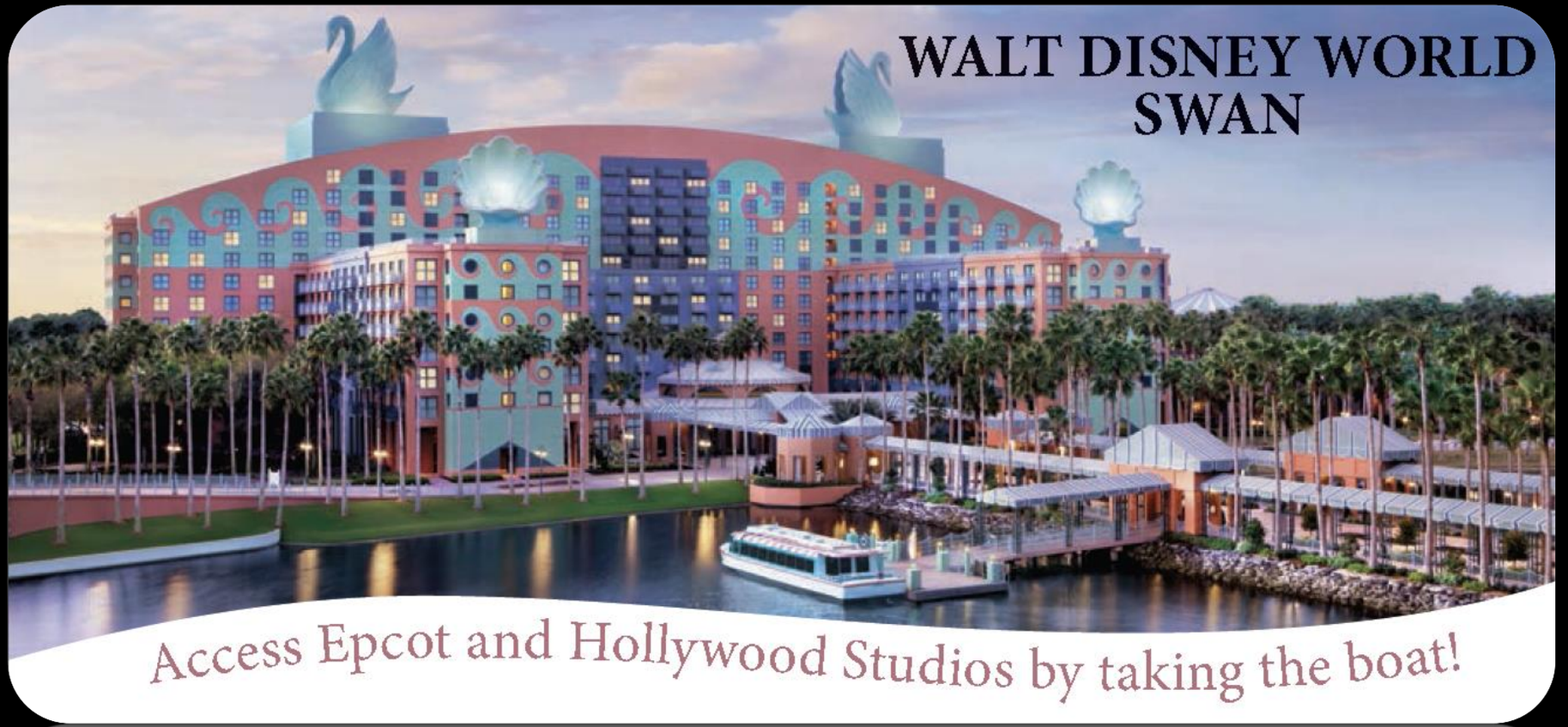
Enhanced database caching
Cache data with automatic, multiple TempDB files per instance in multi-core environments

New Programmatic Improvements
New TSQL Functionality, Maintenance Plan Improvements, New ALTER DATABASE Options

Expanded support for JSON data

Save the Date!

www.SQLintersection.com



2018

Mar 25-28

We're back in Orlando!

SQL intersection

Leave the every day behind and enter a world of wonder and enchantment at the Walt Disney World® Resort. Located in the heart of the most magical place on earth, the Walt Disney World Swan and Dolphin Resort provides a truly extraordinary backdrop for our event! Beautiful tropical landscaping, tranquil waterways, and classic art and architecture work together to create a stunning landmark!