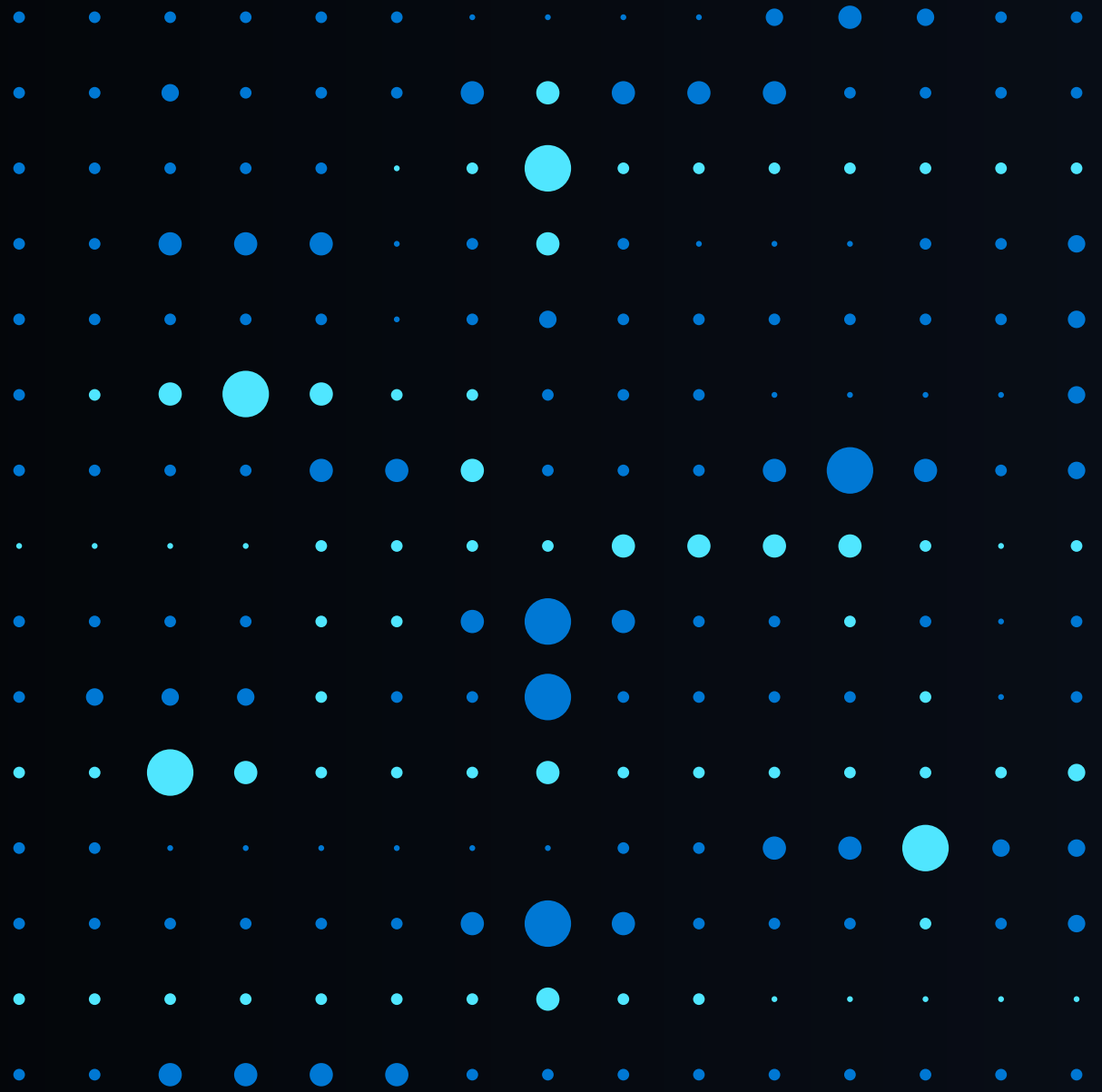
Microsoft

SQL Performance Tuning and Optimization

PTC Inc.

May 8th – 11th

8:00am – 12:00pm EST



Agenda

Day One : SQL Server Architecture

- SQL Operating System
- Waits and Queues
- SQL Server I/O
- Transaction Log Structure

Day Two: Transactions and Concurrency

- SQL Server Transactions
- SQL Server Locking
- Blocking and Deadlocking

Day Three: SQL Server Indexing

- Indexing Internals and Strategies
- Index Monitoring and Fragmentation
- Columnstore Indexes

Day Four: Query Execution Plans

- Query Execution and Plan Analysis
- Troubleshooting with Query Store

Attendee Introductions

Name

Job/Team/Specialty

Time Zone

Experience

Expectations

Microsoft

SQL Server Architecture

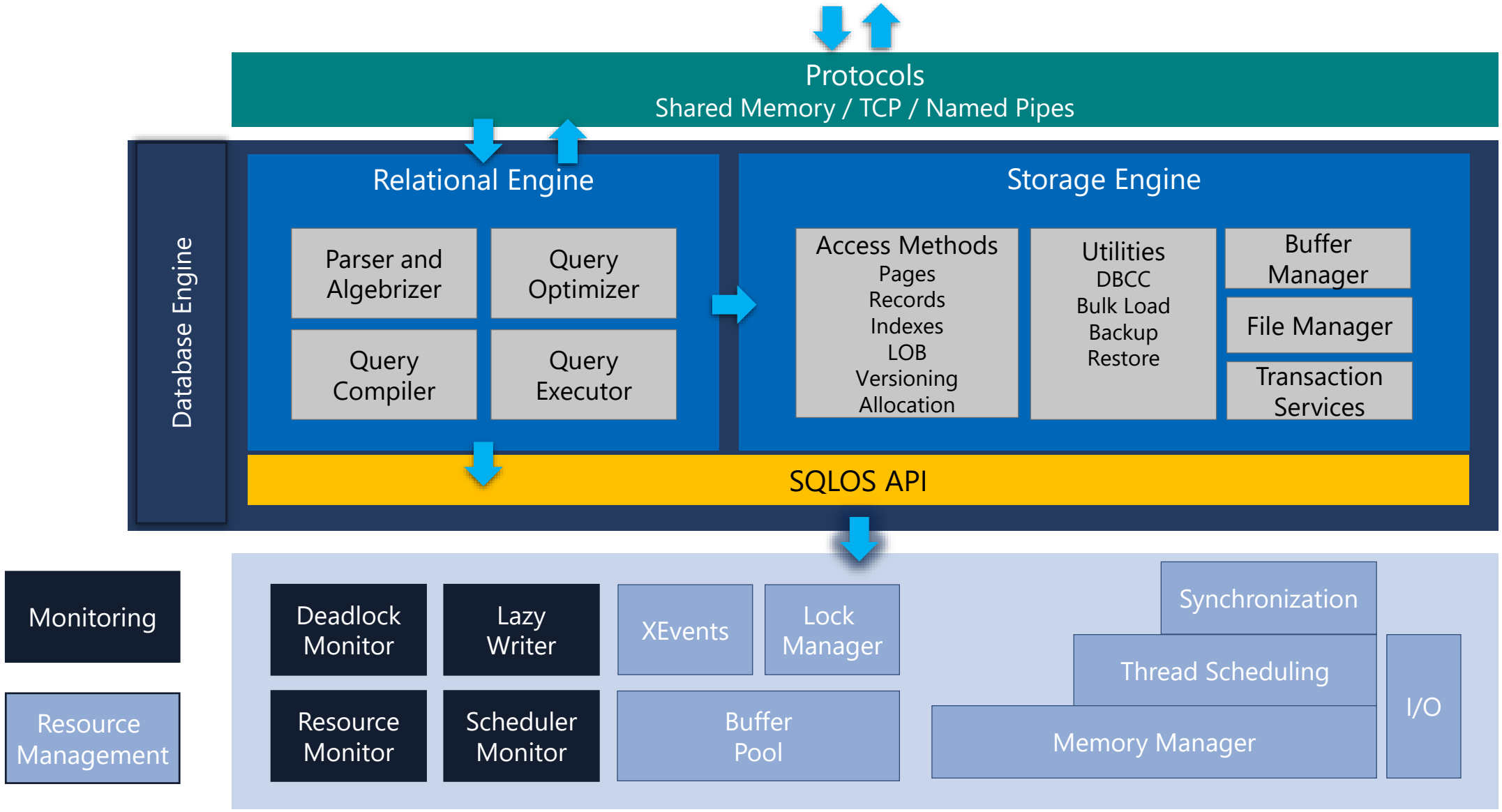
Module 1

Learning Units covered in this Module

- Lesson 1: Introduction to SQL Operating System
- Lesson 2: Waits and Queues

Lesson 1: Introduction to SQL Operating System

Inside the Database Engine



SQL Server Operating System (SQLOS)

Application layer between Microsoft SQL Server components and the Windows Operating System.

Centralizes resource allocation to provide more efficient management and accounting.

The SQLOS is used by the SQL Server relational database engine for system-level services.

Abstracts the concepts of resource management from components, providing:

- **Scheduling and synchronization support**
- **Memory management and caching**
- **Resource governance**
- **Diagnostics and debug infrastructure**
- **Scalability and performance optimization**

Two Main Functions of SQLOS

Management

- Memory Manager
- Process Scheduler
- Synchronization
- I/O
- Support for Non-Uniform Memory Access (NUMA) and Resource Governor

Monitoring

- Resource Monitor
- Deadlock Monitor
- Scheduler Monitor
- Lazy Writer (Buffer Pool management)
- Dynamic Management Views (DMVs)
- Extended Events
- Dedicated Administrator Connection (DAC)

Dynamic Management Views and Functions

Category	Description
sys.dm_exec_%	Execution and connection information
sys.dm_os_%	Operating system related information
sys.dm_tran_%	Transaction management information
sys.dm_io_%	I/O related information
sys.dm_db_%	Database information

Using Dynamic Management Objects (DMOs)

- Must reference using the sys schema
- Two basic types:
  - Real-time state information
  - Historical information

```
SELECT cpu_count, hyperthread_ratio,
    scheduler_count, scheduler_total_count,
    affinity_type, affinity_type_desc,
    softnuma_configuration, softnuma_configuration_desc,
    socket_count, cores_per_socket, numa_node_count,
    sql_memory_model, sql_memory_model_desc
FROM sys.dm_os_sys_info
```

Lesson 2: Waits and Queues

Microsoft SQL Server Scheduling Terminology

Batch

- A statement or set of statements submitted to SQL Server by the user (a query), also referred to as a request
- Monitor with sys.dm_exec_requests

Task

- A batch will have one or more tasks (aligns with statements)
- Monitor with sys.dm_os_tasks

Worker Thread

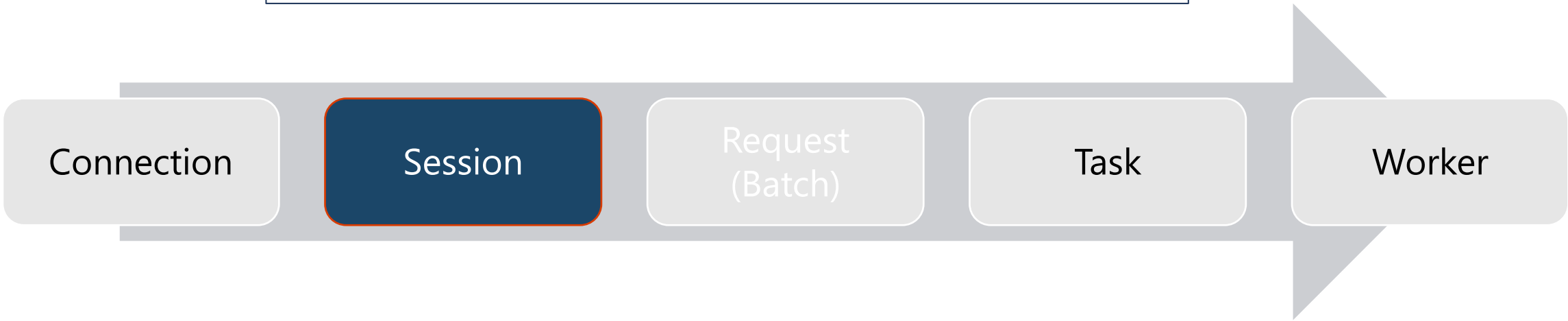
- Each task will be assigned to a single worker thread for the life of the task
- Monitor with sys.dm_os_workers

Hierarchy of Common Terms

```
SELECT *
FROM sys.dm_exec_connections;
-- relevant data:
-- session_id --> spid
-- most_recent_sql_handle --> last query
-- net_transport, protocol_type --> connectivity
```

Hierarchy of Common Terms

```
SELECT *
FROM sys.dm_exec_sessions;
-- relevant data:
-- session_id --> spid
-- host_name, program_name --> client identity
-- login_name, nt_user_name --> login identity
-- status --> activity
-- database_id --> database being accessed
-- open_transaction_count --> blocking identification
```



Hierarchy of Common Terms

```
SELECT *
FROM sys.dm_exec_requests;
-- relevant data:
-- session_id --> spid
-- status --> background, running, runnable, suspended
-- sql_handle, offset --> query text
-- database_id --> database being accessed
-- wait_type, wait_time --> blocking information
-- open_transaction_count --> blocking others
-- cpu_time, total_elapsed_time, reads, writes --> telemetry
```

Hierarchy of Common Terms

```
SELECT *
FROM sys.dm_os_tasks;
-- relevant data:
-- task_state --> running, suspended
-- pending_io_* --> I/O activity
-- scheduler_id -->processor info
-- session_id --> spid
```

Hierarchy of Common Terms

```
SELECT *
FROM sys.dm_os_workers;
-- relevant data:
-- worker_address --> memory address of the worker
-- wait_start_ms_ticks --> Point in time worker Suspended.
-- wait_resumed_ms_ticks --> Worker in Runnable state.
-- state -- > Running, Runnable, Suspended
```

Scheduling Types

Non-Preemptive (Cooperative)

- SQL Server manages CPU scheduling for most activity (instead of the operating system).
- SQL Server decides when a thread should wait or get switched out (known as yielding).
- SQL Server developers also programmed some predetermined voluntary yields to avoid starvation of other threads

Preemptive

- Preemption is the act of an operating system temporarily interrupting an executing task.
- Higher priority tasks can preempt lower priority tasks.
- Preemptive mode used in SQL Server for external code calls, CLR with an UNSAFE assemblies, extended stored procedures

Yielding

In SQL Server, each thread is assigned a quantum (duration 4ms), with SQL Server using a cooperative model to ensure its CPU resources are shared amongst all the threads that are in a runnable state, preventing the 'starving' condition of any individual thread.

By design, a worker owns the scheduler until it yields to another worker on the same scheduler.

When no worker is currently on the Runnable list, the **yielding worker is allowed another quantum** or performs the necessary idle scheduler maintenance.

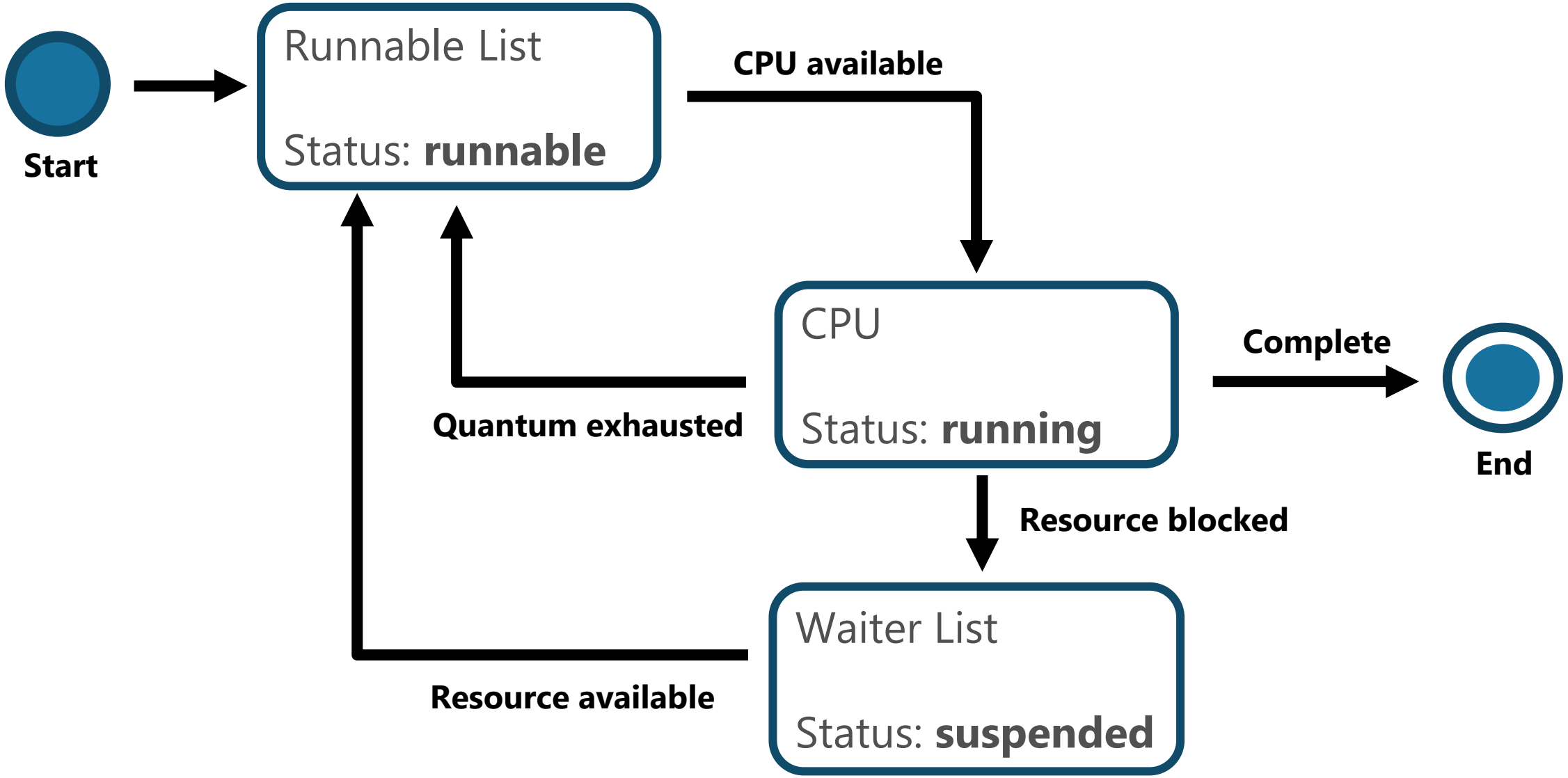
Thread States and Queues

Runnable: The thread is currently in the Runnable Queue waiting to execute. (First In, First Out).

Running: One active thread executing on a processor.

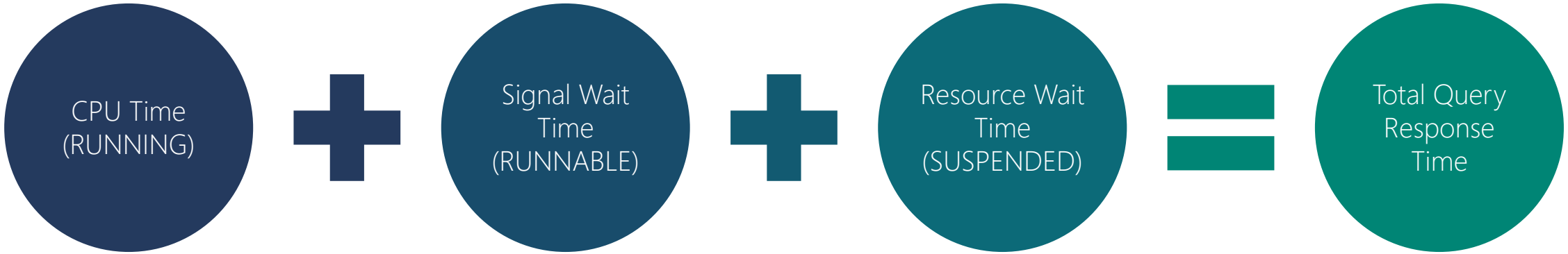
Suspended: Placed on a Waiter List waiting for a resource other than a processor. (No specific order).

Yielding


Start
Runnable List
Status: runnable
CPU available
CPU
Status: running
Complete
End
Quantum exhausted
Resource blocked
Waiter List
Status: suspended
Resource available

Task Execution Model

- The full cycle between the several task states, for how many times it needs to cycle, is what we experience as the total query response time.



Task Execution Model

Status: Running

session_id 51	Running

Runnable Queue (Signal Waits)
Status: Runnable

session_id 51	Runnable
session_id 64	Runnable
session_id 87	Runnable
session_id 52	Runnable
session_id 56	Runnable

SPID56 moved to the bottom of the Runnable queue.

Wait Queue (Resource Waits)
Status: Suspended

session_id 73	LCK_M_S
session_id 59	NETWORKIO
~~session_id 56~~	~~Runnable~~
session_id 55	RESOURCE_SEMAPHORE
session_id 60	IO_Completion

Relevant Dynamic Management Views (DMVs)

sys.dm_os_wait_stats

- Returns information about all the waits encountered by threads that ran.
- Includes wait type, number of tasks that waited in the specific wait type, total and max wait times, and the amount of signal waits.

sys.dm_os_waiting_tasks

- Returns information about the wait queue of tasks actively waiting on some resource.

sys.dm_exec_requests

- Returns information about each request that is in-flight.
- Includes session owning the request and status of the request, which will reflect the status of one or more tasks assigned to the request.

Waiting Tasks DMV

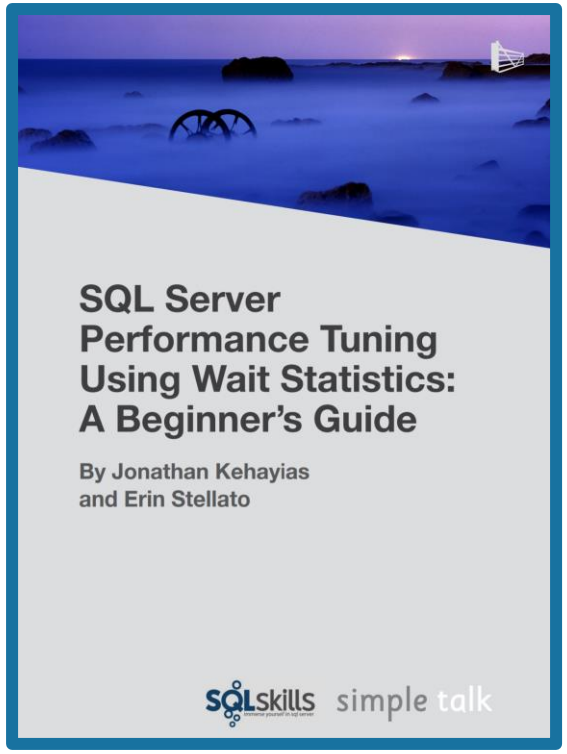
```
SELECT w.session_id, w.wait_duration_ms, w.wait_type,
       w.blocking_session_id, w.resource_description,
       s.program_name, t.text, t.dbid, s.cpu_time, s.memory_usage
 FROM sys.dm_os_waiting_tasks as w
      INNER JOIN sys.dm_exec_sessions as s
         ON w.session_id = s.session_id
      INNER JOIN sys.dm_exec_requests as r
         ON s.session_id = r.session_id
      OUTER APPLY sys.dm_exec_sql_text (r.sql_handle) as t
  WHERE s.is_user_process = 1;
```

session_id	wait_duration_ms	wait_type	blocking_session_id	resource_description
58	8563	LCK_M_S	62	keylock hobtid=72057594047365120 dbid=5 id=lock1...

Troubleshooting Wait Types

Aaron Bertrand – Top Wait Types
https://sqlperformance.com/2018/10/sql-performance/top-wait-stats

Paul Randal – SQL Skills Wait Types Library
https://www.sqlskills.com/help/waits/



Microsoft

SQL Server I/O and Database Structure

Module 2

Learning Units covered in this Module

- Lesson 1: SQL Server Disk I/O
- Lesson 2: SQL Server Log File Structure

Lesson 1: SQL Server Disk I/O

Database files and filegroups

Database files

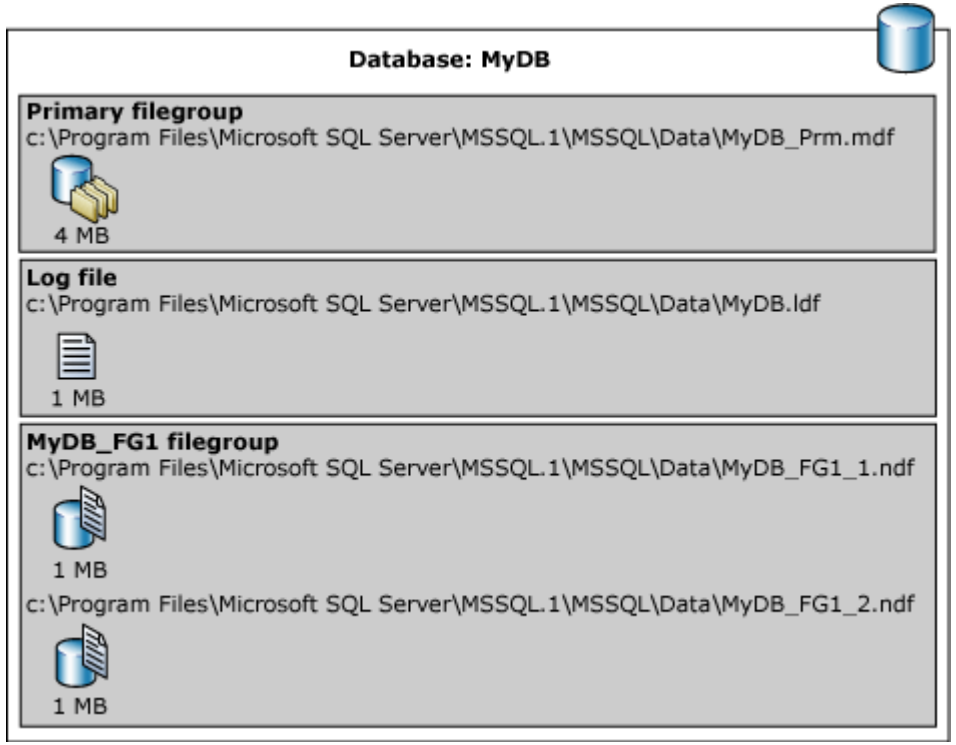
A database is composed by at least two operating system files:

Data files

- Contain database objects and data
- First data file is called primary data file. This file has a .mdf extension
- A database may have additional data files, known as secondary data files. They use .ndf extension
- Can be grouped together in filegroups for allocation and administration purposes

Log file

- Contain Log Records and entries are sequenced



SQL Server disk I/O patterns:

Data Files

- One .mdf file per database
- May have one or more .ndf file
- Random reads and writes
  - Read activity during backups; other activity varies depending on query activity and buffer pool size
  - Write activity during checkpoints, recovery, and lazy writes

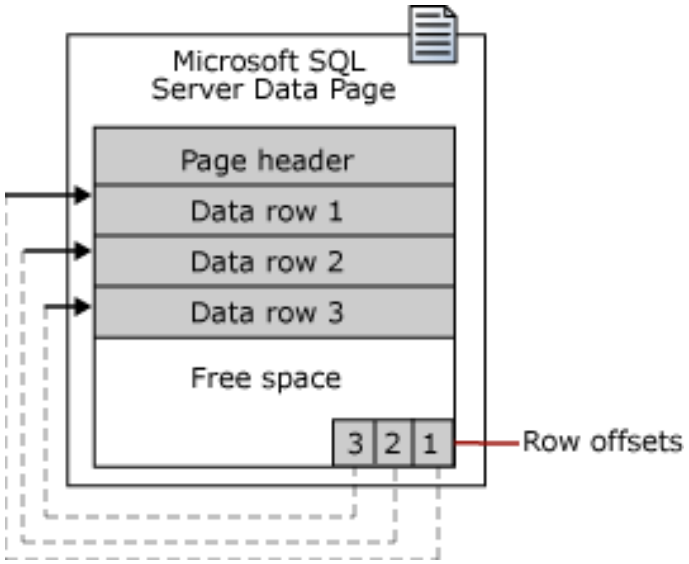
Log Files

- One* .ldf file per database
- Sequential reads and writes
- Write activity during the log buffer flush operations
- Read activity during checkpoints, backups, and recovery
- Features such as database mirroring and replication will increase read and write activity

Pages and Extents architecture

A data page is the fundamental unit of data storage in SQL Server.

- The disk space allocated to a data file (.mdf or .ndf) is logically divided into pages.
- Each page is 8 KB in size
- Pages are numbered contiguously from 0 to n.
- Disk I/O operations are performed at the page level.



Extents are a collection of eight physically contiguous pages (64KB) and are used to efficiently manage the pages.

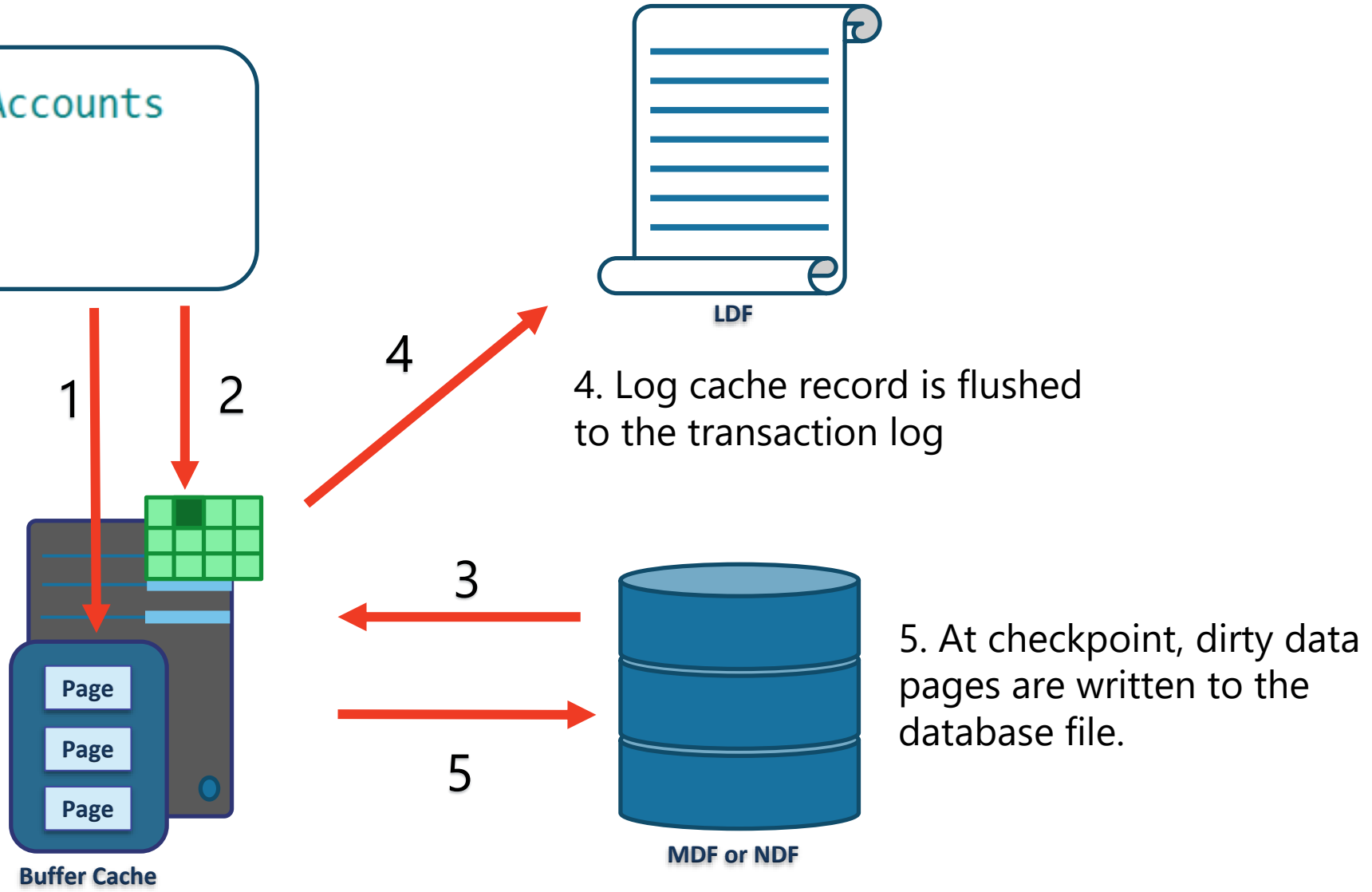
SQL Server Disk I/O (Write-Ahead Logging)

```
UPDATE Accounting.BankAccounts
SET Balance -= 200
WHERE AcctID = 1
```

1. Data modification is sent to buffer cache in memory.

2. Modification is recorded in the log cache.

3. Data pages are located or read into the buffer cache and then modified.



4. Log cache record is flushed to the transaction log

5. At checkpoint, dirty data pages are written to the database file.

Log Buffer Flushing

SQL Server will flush the log buffer to the log file

- SQL Server gets a commit request of a transaction that changes data.
- The log buffer fills up. (Max size 60kb.)
- SQL Server needs to harden dirty data pages (checkpoints)
- Manually request a log buffer flush using the sys.sp_flush_log procedure

Log buffer flushing results in a WRITELOG wait type.

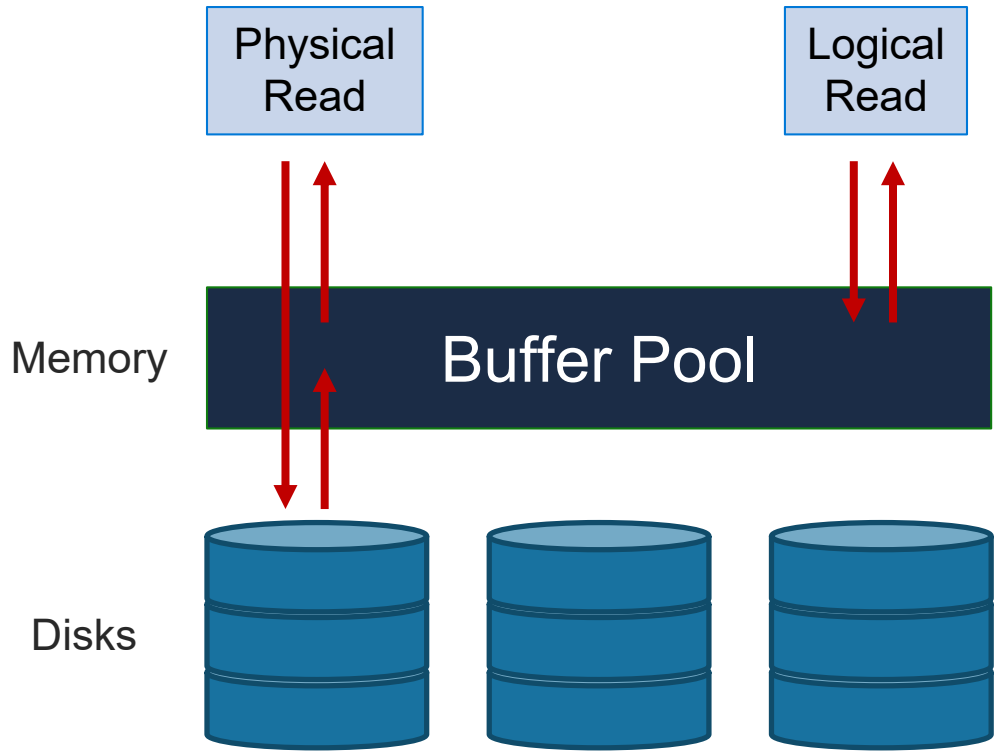
SQL Server Buffer Pool

Stores 8 kilobytes (KB) pages of data to avoid repeated disk I/O.

- Pages held in the buffer until the space is needed by something else.

Lazy Writer searches for eligible buffers.

- If the buffer is dirty, an asynchronous write (lazy write) is posted so that the buffer can later be freed.
- If the buffer is not dirty, it is freed.



SET STATISTICS IO

```
SET STATISTICS IO ON
GO
SET STATISTICS TIME ON
SELECT SOH.SalesOrderID, SOH.CustomerID,
OrderQty, UnitPrice, P.Name
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN Production.Product AS P
ON P.ProductID = SOD.ProductID
SET STATISTICS IO, TIME OFF
```

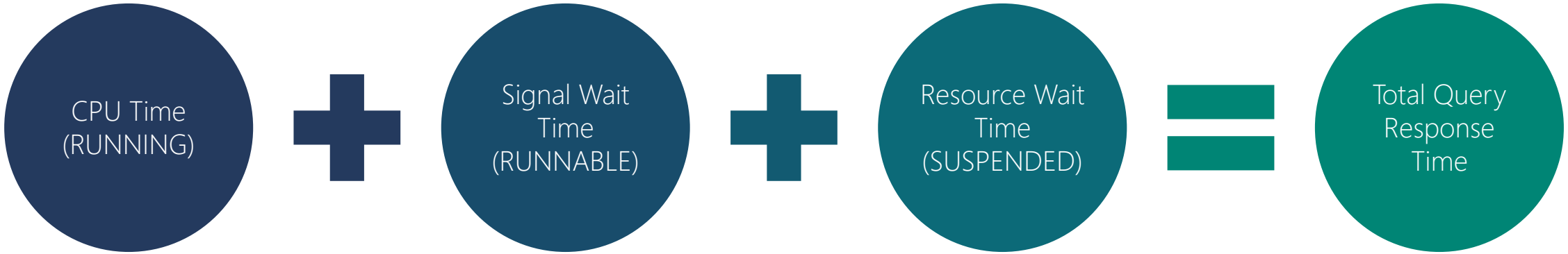
Used to identify physical reads and logical reads for a query

```
(121317 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server r
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server
Table 'SalesOrderDetail'. Scan count 1, logical reads 428, physical reads 0, pag
Table 'Product'. Scan count 1, logical reads 15, physical reads 0, page server r
Table 'SalesOrderHeader'. Scan count 1, logical reads 57, physical reads 0, page

 SQL Server Execution Times:
   CPU time = 94 ms,  elapsed time = 1653 ms.
```

Total Query Response Time

- The full cycle between the several task states, for how many times it needs to cycle, is what we experience as the total query response time.



Checkpoints

Flushes dirty pages from the buffer pool to the disk. Frequency of checkpoints varies based on the database activity and recovery interval.

Automatic (default) – Database engine issues checkpoints automatically based on the server level "recovery interval" configuration option

Indirect (new in SQL Server 2012) – Database engine issues checkpoints automatically based on the database level TARGET_RECOVERY_TIME

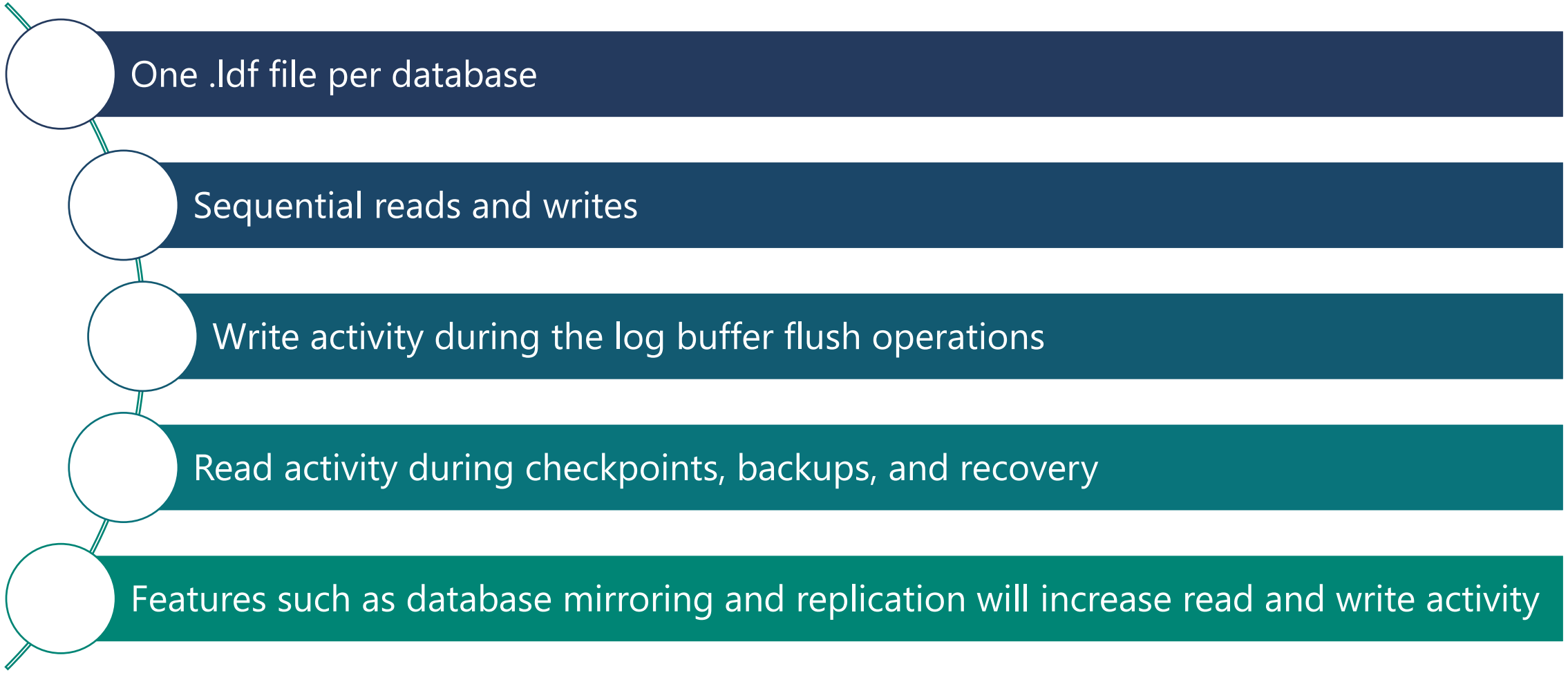
```
ALTER DATABASE [AdventureWorksPTO] SET TARGET_RECOVERY_TIME = 60 SECONDS
```

Manual – Issued in the current database for your connection when you execute the T-SQL CHECKPOINT command

Internal – Issued by various server operations

Lesson 2: SQL Server Log File Structure

SQL Server disk I/O patterns: transaction log



Transaction Log

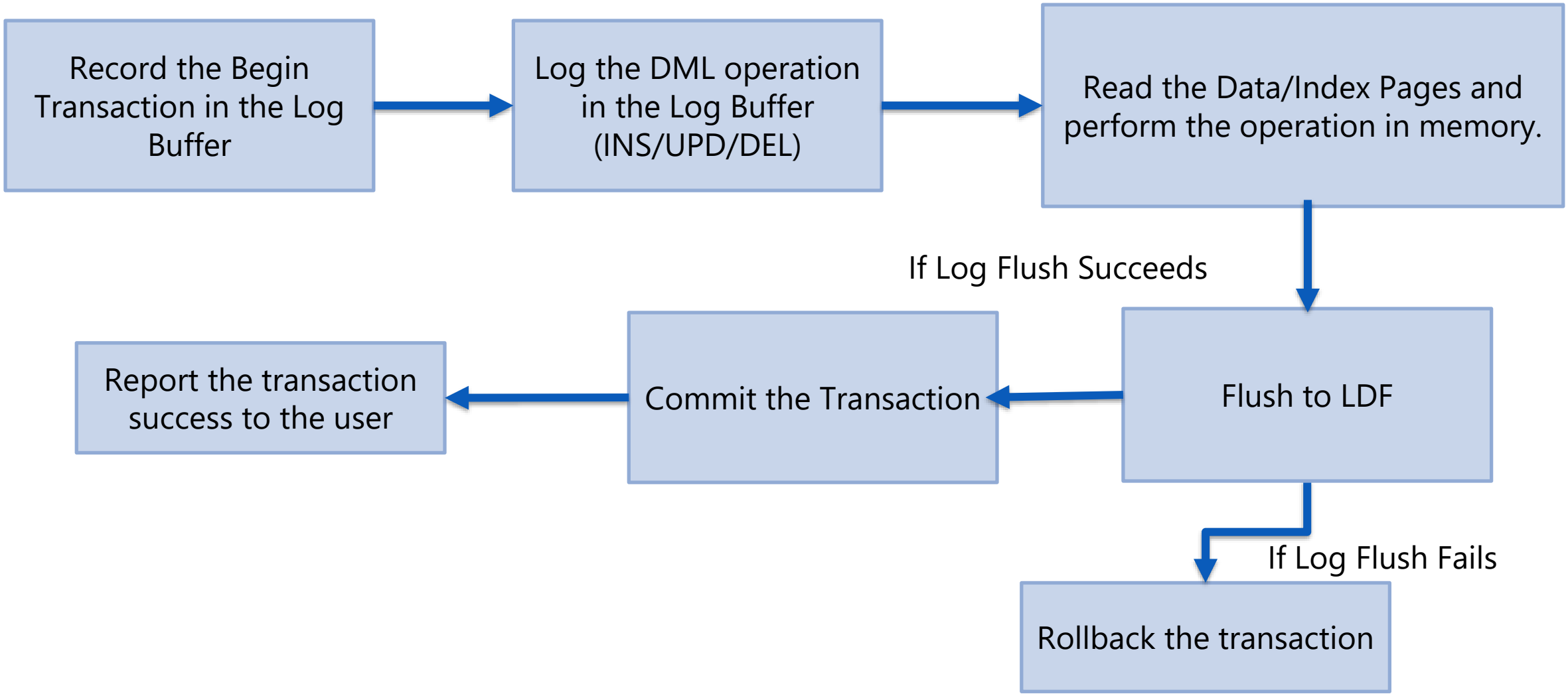
Sequence of log records contained in one or more physical files

Records identified by increasing Logical Sequence Numbers (LSNs)

Recorded operations:

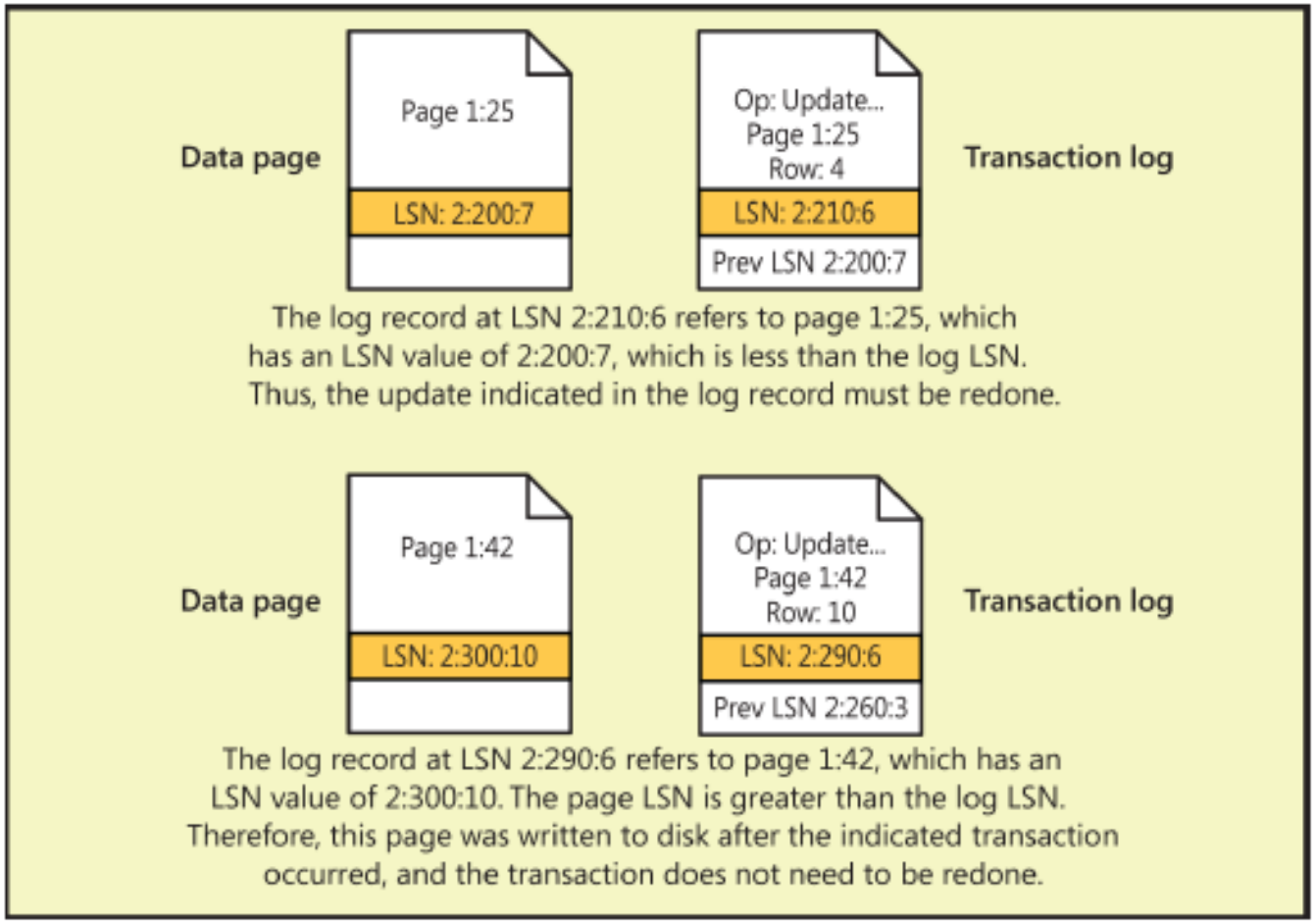
- Start and end of each transaction
- All database modifications
- Extent and page allocation or deallocation
- Creating and dropping objects

Microsoft SQL Server Transaction Logging



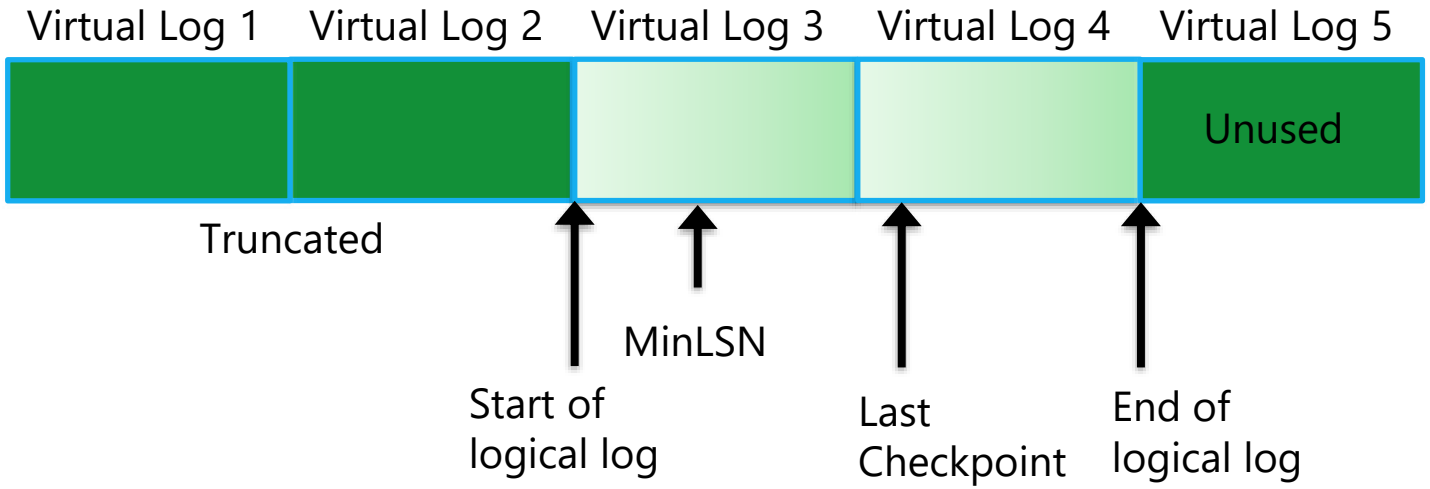
Page LSN and recovery

- Last LSN in the page header
- Log Record has two LSNs
  - Previous LSN from the data page
  - Current log record LSN
- All used for recovery



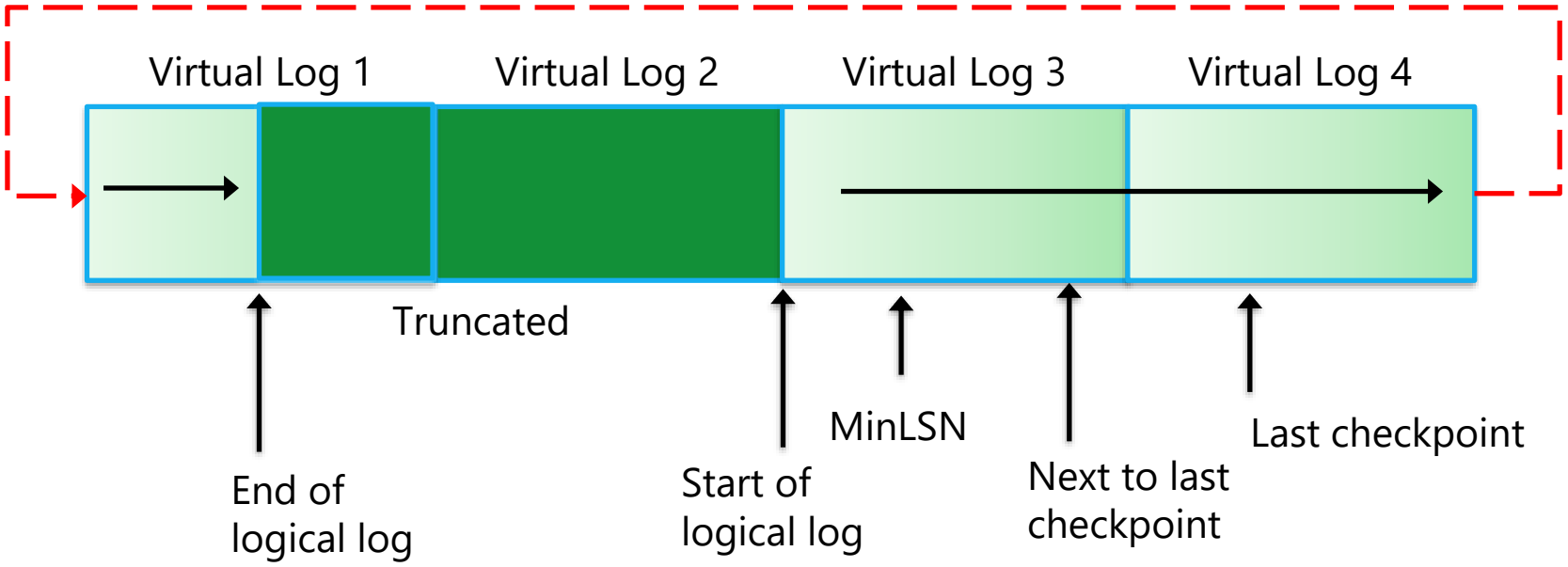
Physical Log File Structure

- The physical file is divided into virtual log files (VLFs).
- SQL Server works to keep the number of VLFs small.
- VLF size and number is dynamic and cannot be configured or set.



Physical Log File Structure (Continued)

- Log file is circular.
- When the end of logical log meets the beginning, the physical log file is extended.



Addressing VLF Counts

VLF count T-LOG file creation

Size <= 64 MB → 4 VLFs

Between 64 MB and 1 GB → 8 VLFs

Size > 1GB →16 VLFs

VLF count at T-LOG file growth

Growth <= 64 MB → 4 VLFs

Between 64 MB and 1GB → 8 VLFs

Growth > 1GB → 16 VLFs

(SQL 2014 and higher)

If growth is less than 1/8th of current size → add 1 VLF

Addressing VLF Counts

To avoid VLF fragmentation

- Pre-size log files to avoid unplanned file growth
- Set autogrowth to an appropriate fixed size

To view VLFs

- DBCC LOGINFO (<database_id>)
- sys.dm_db_log_info (SQL Server 2016 SP2 and higher)
- Active VLFs have a status of 2
- SQL Server Error Log (SQL 2012 and higher)

Microsoft

SQL Server Transactions and Concurrency

Module 3

Learning Units covered in this Module

- Lesson 1: SQL Server Transactions
- Lesson 2: SQL Server Locking
- Lesson 3: Troubleshooting Concurrency Performance

Lesson 1: SQL Server Transactions and Concurrency

What is a Transaction?

A transaction is a sequence of steps that perform a logical unit of work.

Must Exhibit ACID properties, to qualify as a transaction.

A - Atomicity

- A transaction is either fully completed or not at all.

C - Consistency

- A transaction must leave data in a consistent state.

I - Isolation

- Changes made by a transaction must be isolated from other concurrent transactions.

D - Durability

- The modifications persist even in the event of a system failure.

Transaction Modes

Auto-Commit

Individual statements that complete successfully, will be committed. If errors are encountered the statement is rolled back.

Explicit

Transaction is explicitly defined with a BEGIN TRANSACTION and COMMIT TRANSACTION statement.

Implicit

- Transaction starts automatically once first statement of a batch is received. Must still manually COMMIT or ROLLBACK transaction.

Auto-Commit transaction Mode

Default transaction management mode of the SQL Server Database Engine

Each statement is either committed or rolled back automatically upon completion.

A syntax error will result in a batch terminating error.

- This will stop the entire batch from being executed.

A run-time error will result in a statement terminating error.

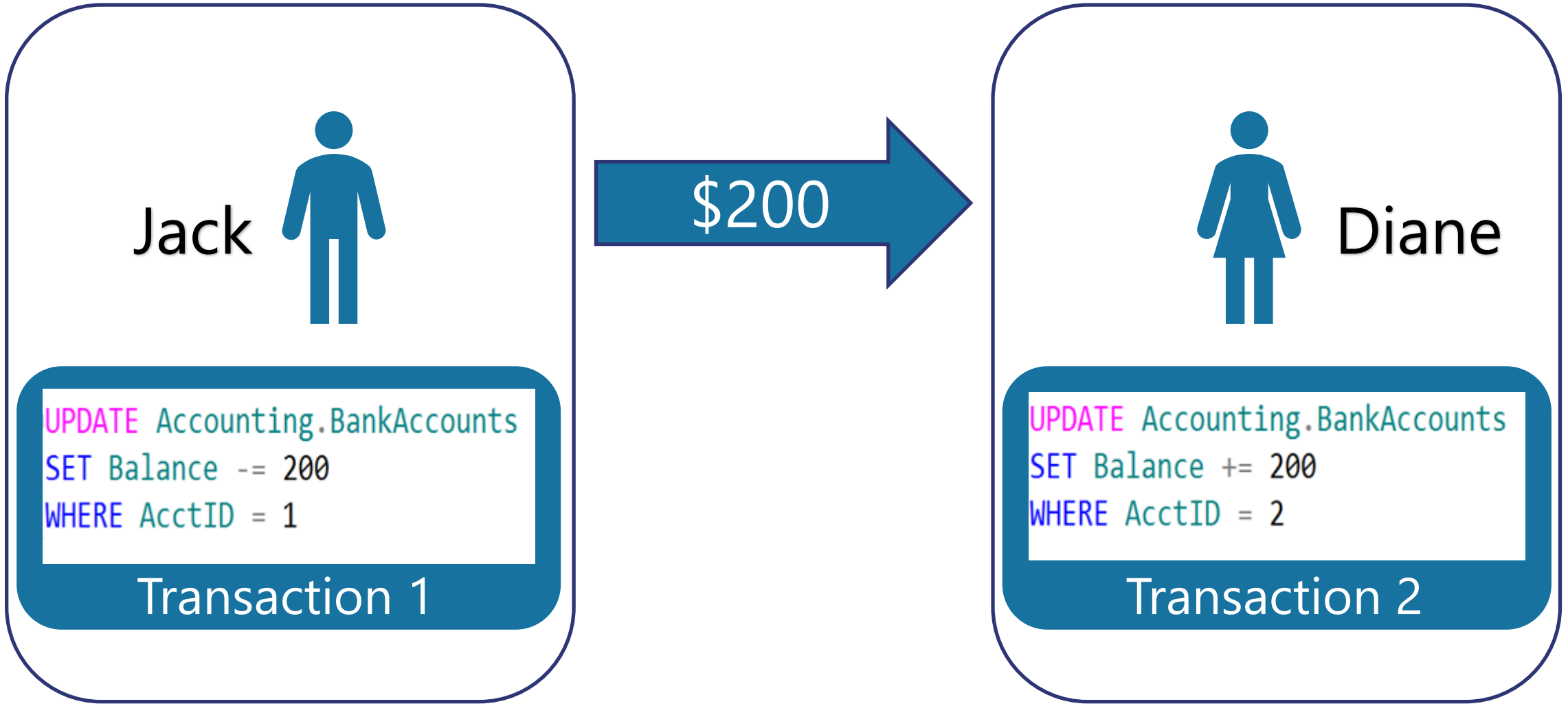
- This might allow part of the batch to commit.

Database engine operates in autocommit until started an explicit transaction

XACT_ABORT ON converts statement into batch terminating errors

Compilation errors not affected by XACT_ABORT ON

Logical Units of Work – Auto Commit Transactions



Explicit transaction mode

An explicit transaction is one in which you explicitly define both the start and end of the transaction.

Begins with the BEGIN TRANSACTION statement

Transaction can be started with a mark

Can have one or more statements

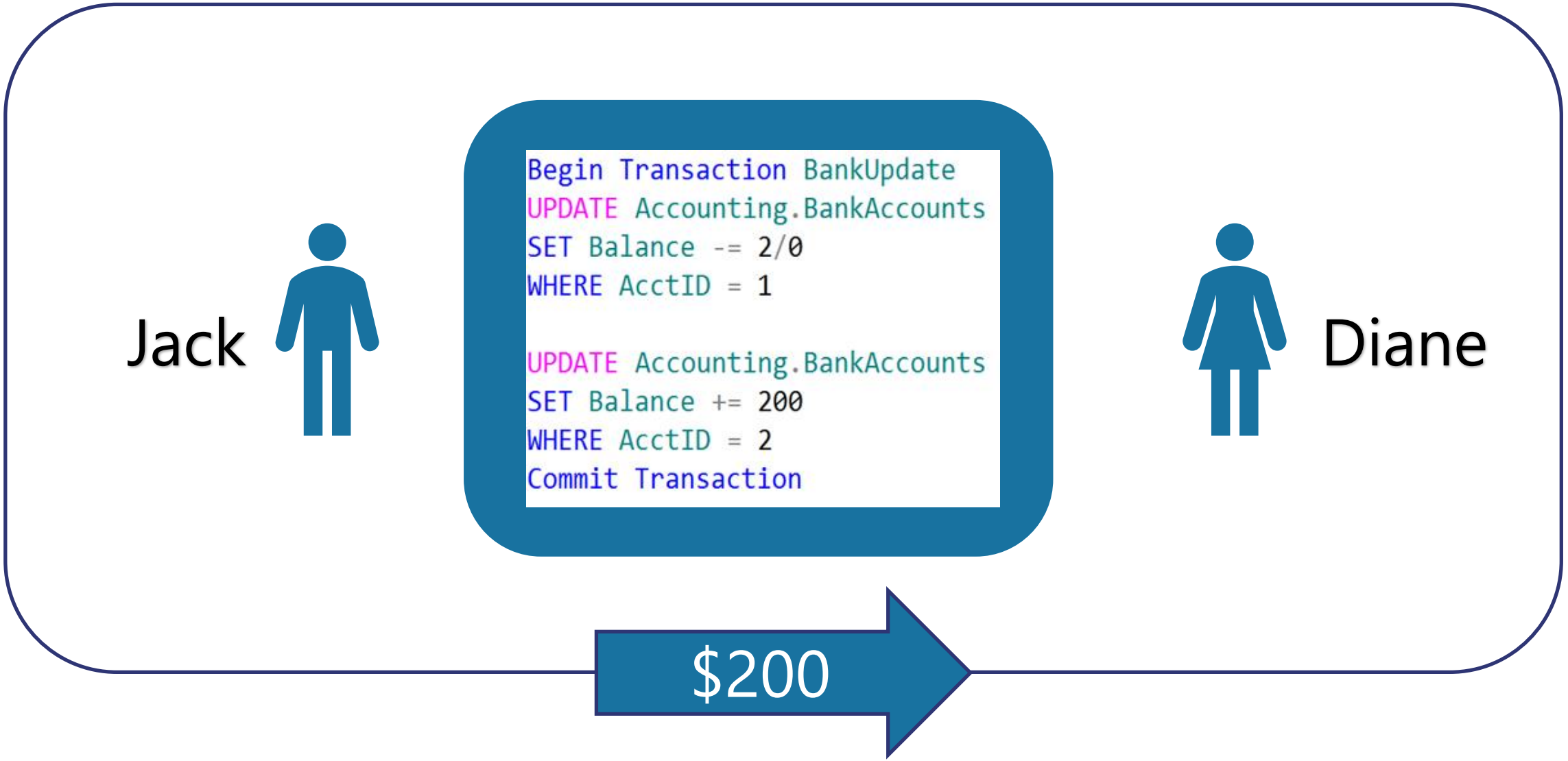
Need to explicitly commit or rollback the transaction

- using COMMIT TRANSACTION or ROLLBACK TRANSACTION

Can also use

- SAVE TRANSACTION <savepoint_name>– to rollback a transaction to named Point
- BEGIN TRANSACTION <transaction_name>WITH MARK ['*description*'] – to specify transaction marked in log

Single Logical Unit of Work – Explicit Transactions



Implicit transaction mode

- Equivalent to an unseen BEGIN TRANSACTION being executed
- Transaction starts automatically once first statement of a batch is received
- SET IMPLICIT_TRANSACTIONS ON used at statement level
- Enabled at Server level by using sp_configure 'user options' , 2
- It can be symptom of severe blocking issues on the server
- Must use commit after SELECTs or DML, otherwise transaction remains open

Considerations for using transactions

Keep transactions as short as possible

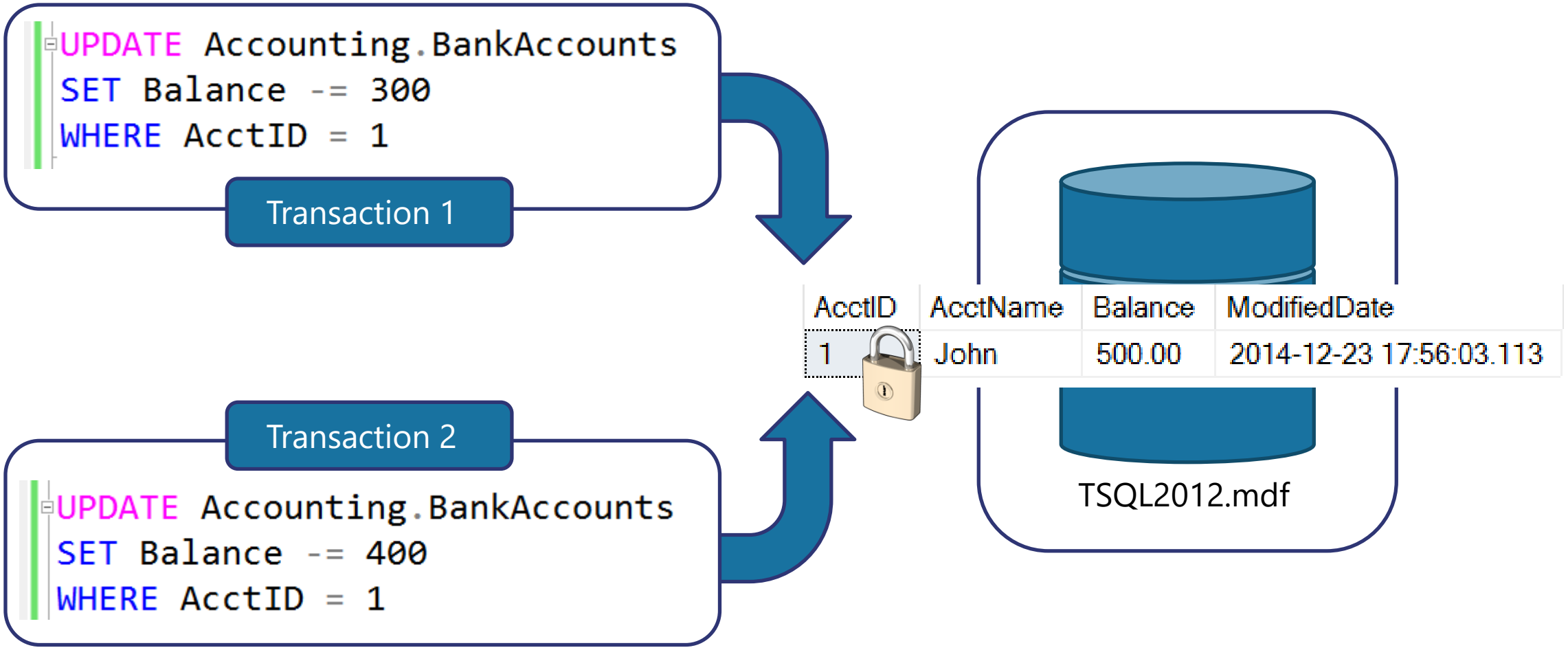
- Do not require user input
- Do not open a transaction while browsing through data
- Access the least amount of data possible
- Do not open the transaction before it is required
- Ensure that appropriate indexing is in place

Try to access resources in the same order

- Wherever possible access resources in the same order to avoid Deadlocks

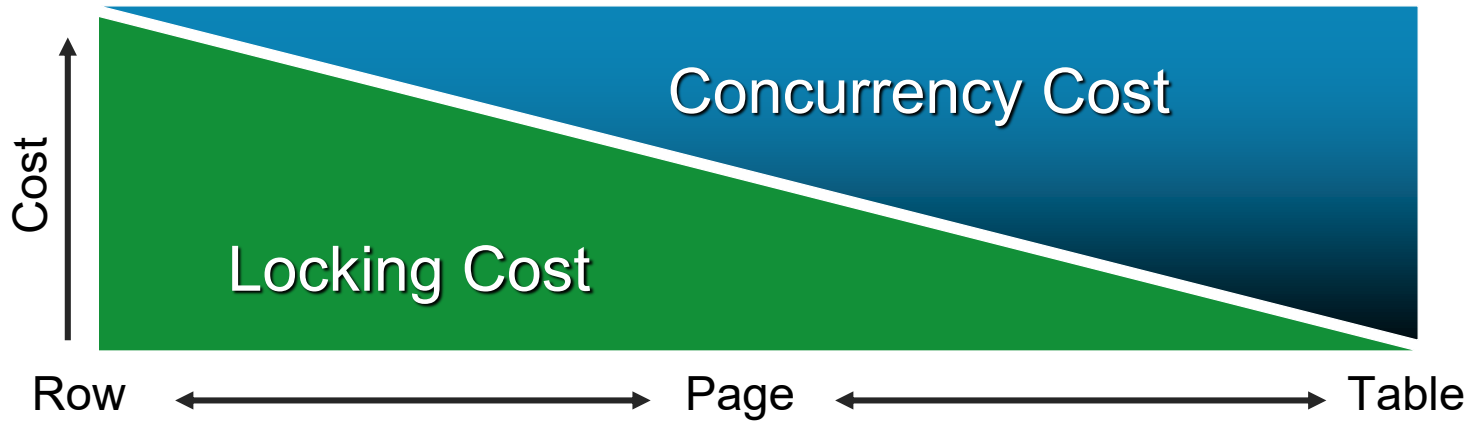
Lesson 2: SQL Server Locking

What is a Lock?



Multi-Granular Locking

- Many items can be locked in SQL Server
  - Databases
  - Schema
  - Objects
- Some objects can be locked at different levels of granularity
- SQL Server will automatically choose the granularity of the lock based on the estimated cost
- Multiple levels of granularity are grouped into a lock hierarchy



Lock Granularity and Hierarchies

Resource	Description
RID	A row identifier used to lock a single row within a heap.
KEY	A row lock within an index used to protect key ranges in serializable transactions.
PAGE	An 8-kilobyte (KB) page in a database, such as data or index pages.
EXTENT	A contiguous group of eight pages, such as data or index pages.
HoBT	A heap or B-tree. A lock protecting heap data pages in a table that does not have a clustered index or the pages of a B-tree index.
TABLE	The entire table, including all data and indexes.
FILE	A database file.
ALLOCATION_UNIT	An allocation unit.
DATABASE	The entire database.

Lock Duration

Mode	Read Committed	Repeatable Read	Serializable	Snapshot
Shared	Held until data read and processed	Held until end of transaction	Held until end of transaction	N/A
Update	Held until data read and processed unless promoted to Exclusive	Held until data read and processed unless promoted to Exclusive	Held until end of transaction unless promoted to Exclusive	Held until data read and processed unless promoted to Exclusive
Exclusive	Held until end of transaction	Held until end of transaction	Held until end of transaction	Held until end of transaction

Lock Mode - Standard

Lock Mode	Description
Schema-Stability (Sch-S)	Used when compiling queries
Schema Modification (Sch-M)	Used when a table data definition language operation (for example, dropping a table) is being performed
Shared (S)	Used for read operations that do not change or update data, such as a SELECT statement
Update (U)	Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and potentially updating resources later
Exclusive (X)	Used for data-modification operations, such as INSERT, UPDATE, or DELETE. Ensures that multiple updates cannot be made to the same resource at the same time

Lock Mode - Special

Lock Mode	Description
Intent Shared (IS)	Have or will request shared lock(s) at a finer level
Intent Update (IU)	Have or will request update lock(s) at a finer level
Intent Exclusive (IX)	Have or will request exclusive lock(s) at a finer level
Shared Intent Update (SIU)	Have shared lock with intention to acquire update lock at a finer level
Shared Intent Exclusive (SIX)	Have shared lock with intention to acquire exclusive lock at a finer level
Update Intent Exclusive (UIX)	Have update lock with intention to acquire exclusive lock at a finer level
Bulk Update (BU)	Used when bulk copying data into a table and either TABLOCK hint is specified or the table lock on bulk load table option is set

Lock hierarchy with intent locks

SQL Server uses intent locks to protect parent-level object in the hierarchy by placing an intent shared (IS) or Intent exclusive (IX) lock.

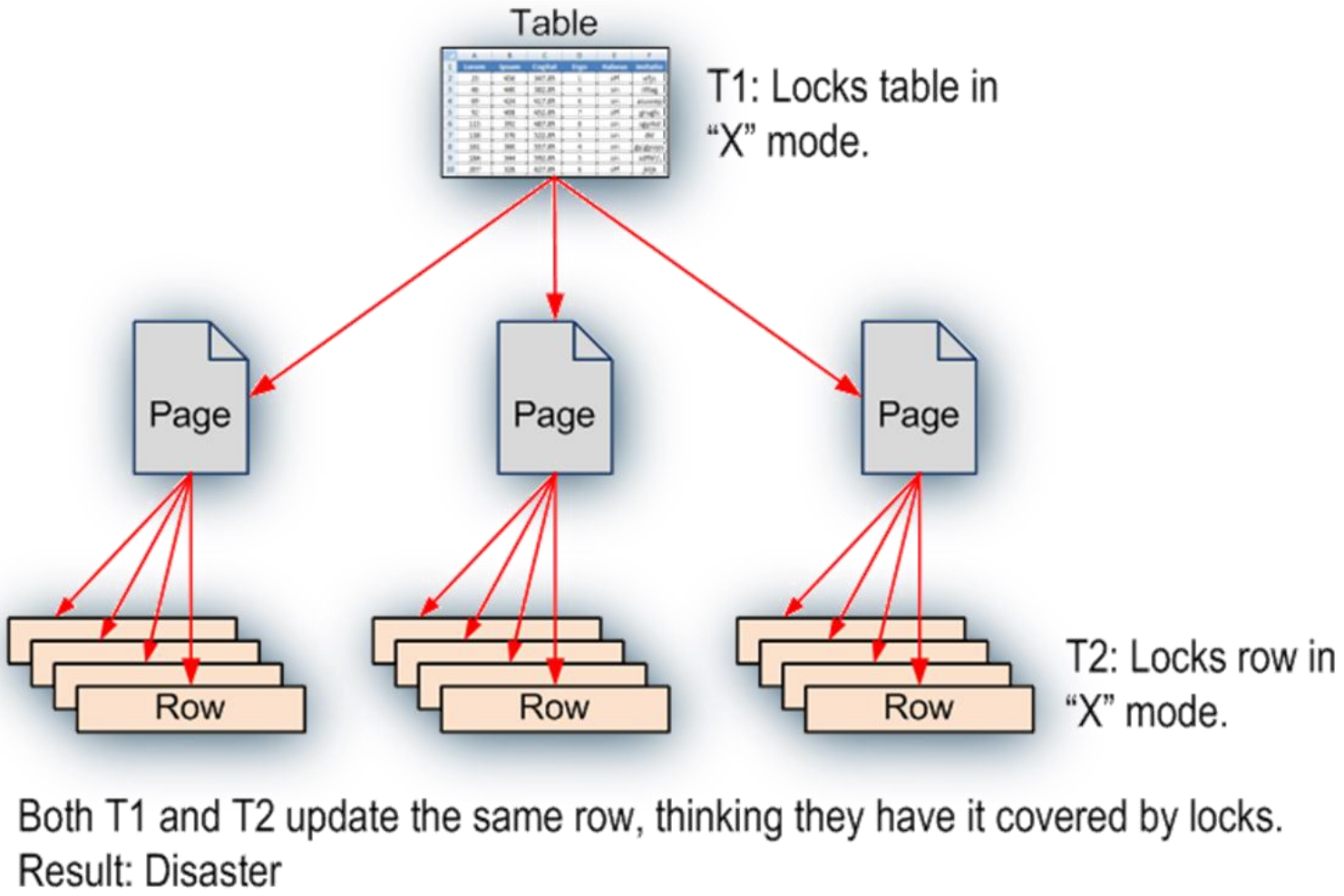
Intent locks are acquired before a lock placed at the lower level.

Intent locks serve two purposes:

- Prevent other transactions from modifying parent-level object
- Improve the efficiency of the SQL Server Database Engine

Establish Lock Hierarchy with Intent Locks

To acquire a fine granular lock, you must acquire intent locks on all the higher levels in the hierarchy



Lock Compatibility

	Existing Granted Mode					
Requested Mode	**IS**	**S**	**U**	**IX**	**SIX**	**X**
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Dynamic locking

Row locking is not always the right choice

- Scanning 100 million rows means 100 million calls to the lock manager

Page, Partition or Table locking can be more efficient

- One Table lock is cheaper and easier to manage than thousands of Row locks

SQL Server chooses lock granularity (Row, Page, Table) at run time based on input from the Query Optimizer

- Least-expensive method is chosen
- Available resources at the time of execution may have an impact
- Incorrect estimates could lead to making the wrong choice

Lock escalation

Lock manager attempts to replace many row or page locks with a single table-level lock.

- One Table lock is faster and easier to manage than thousands of Row locks.
- One Table lock requires less memory than many Row Locks.

It never converts row locks to page locks.

Lock de-escalation never occurs.

Lock Escalation behavior can be controlled.

Server Level	Table Level
Trace Flag 1211 Disables lock escalation due to memory pressure	AUTO | TABLE (Default) | DISABLE

Syntax:

```
ALTER TABLE Table_name  SET ( LOCK_ESCALATION = { AUTO | TABLE | DISABLE } )
```

Locking Hints

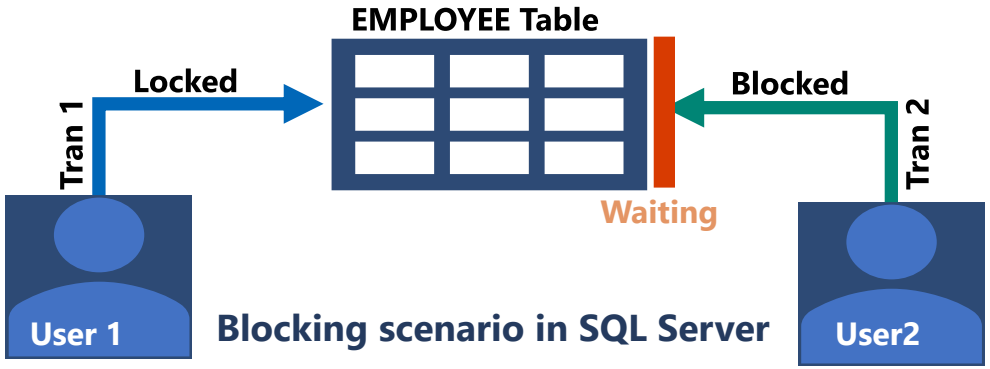
- Override the default behavior of the query optimizer
- Table hints are specified in the **FROM clause** of the DML statement
- Affect only the table or view referenced in that clause
- Locking method can be used at various levels as shown below

Granularity Level hints	ROWLOCK, PAGLOCK and TABLOCK
Isolation LEVEL hints	HOLDLOCK/SERIALIZABLE, NOLOCK/ READUNCOMMITTED, READCOMMITTED,REPEATABLEREAD, READCOMMITTED
UPDLOCK	Use update lock rather than shared lock when reading
XLOCK	Use exclusive lock instead
READPAST	Skips currently locked rows

Lesson 3: Troubleshooting Concurrency Performance

Blocking

Blocking is an unavoidable characteristic of any RDBMS with lock-based concurrency.



Capturing blocking Information

- A Custom SQL scripts using DMVs that monitor locking and blocking
- Use SSMS standard reports i.e. Activity – All blocking transactions
- Extended events - blocked process report

Common Blocking Scenarios and Resolution

- Long running Query
- Orphaned connection
- SPID whose corresponding client application did not fetch all result rows to completion
- Sleeping server SPID that has lost track of the Transaction Nesting Level
- Inappropriate transaction or transaction-isolation level
- SPID that is in a rollback state
- Distributed client/server deadlock

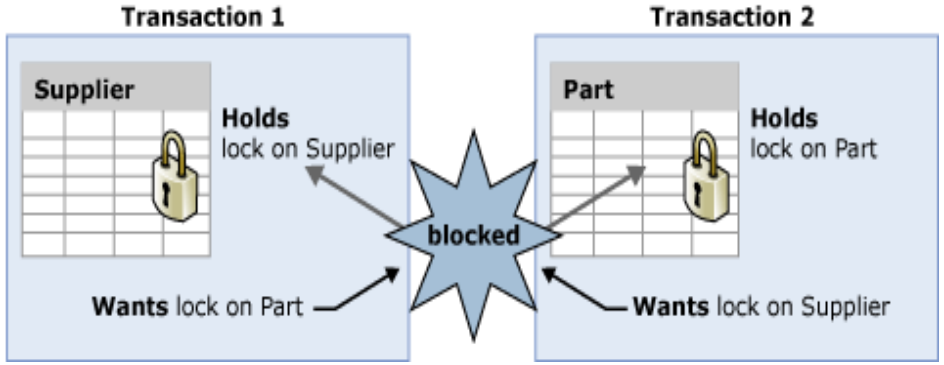
Minimizing Blocking

- Keep transactions short and in one batch.
- Avoid user interaction in a transaction.
- Use proper indexing – The Database Tuning Advisor index analysis.
- Beware of the implicit transactions.
- Reduce the isolation level to lowest possible.
- Locking hint, Index hint, Join hint.
- Roll back when canceling; Roll back on any error or timeout.
- Apply a stress test at maximum projected user load before deployment.

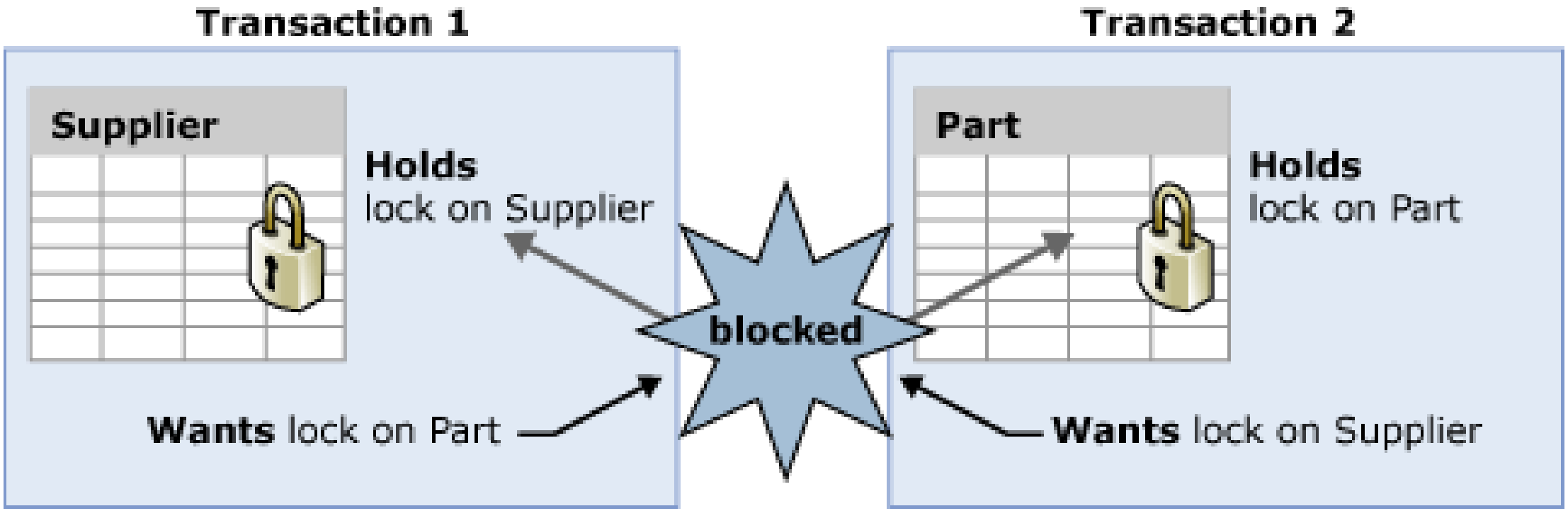
What is a deadlock?

Two or more processes are waiting for one another to obtain locked resources

- Transaction 1 **holds** a lock on the Supplier table and **requires a lock** on the Part table.
- Transaction 2 **holds** a lock on the Part table and **wants a lock** on the Supplier table
- Both tasks cannot continue until a resource is available and the resources cannot be released until a task continues, and therefore a deadlock state exists



What Is a Deadlock?



Detection and Identification

Lock monitor thread periodically perform deadlock detection in SQL Server.

Trace flags 1222,1204.

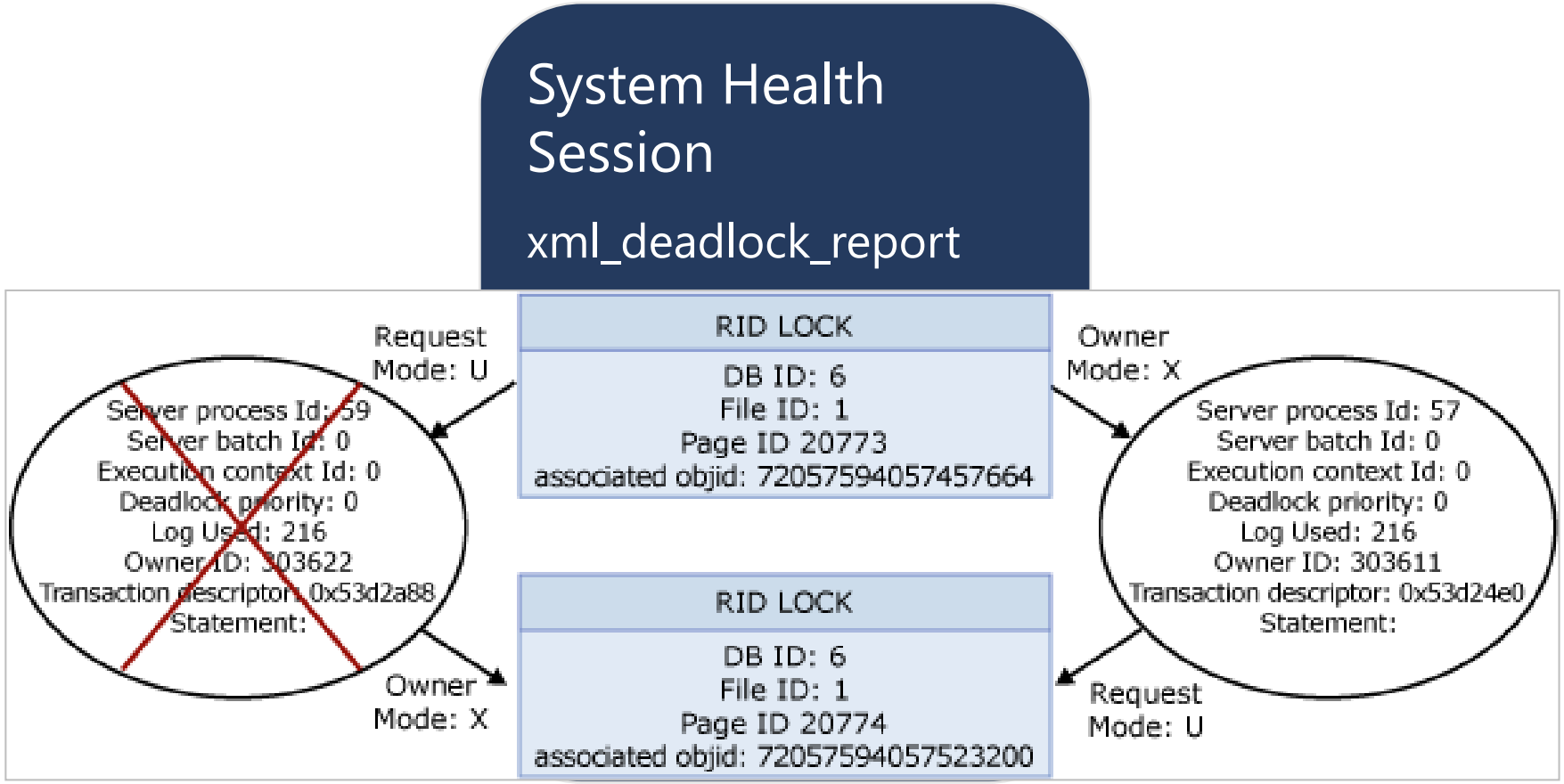
Deadlocks are captured by the system health event session.

When Deadlock occurs, SQL server returns 1205 error code to application.

- Applications should use retry logic.
- Check for 1205 error code to resubmit the transaction.

Deadlock Analysis

Using System Health Xevent



Microsoft

SQL Server Index Structure

Module 4

Learning Units covered in this Module

- Lesson 1: Index Internals
- Lesson 2: Index Strategy
- Lesson 3: Index Monitoring
- Lesson 4: Columnstore Indexes

Lesson 1: Index Internals

What is an Index?

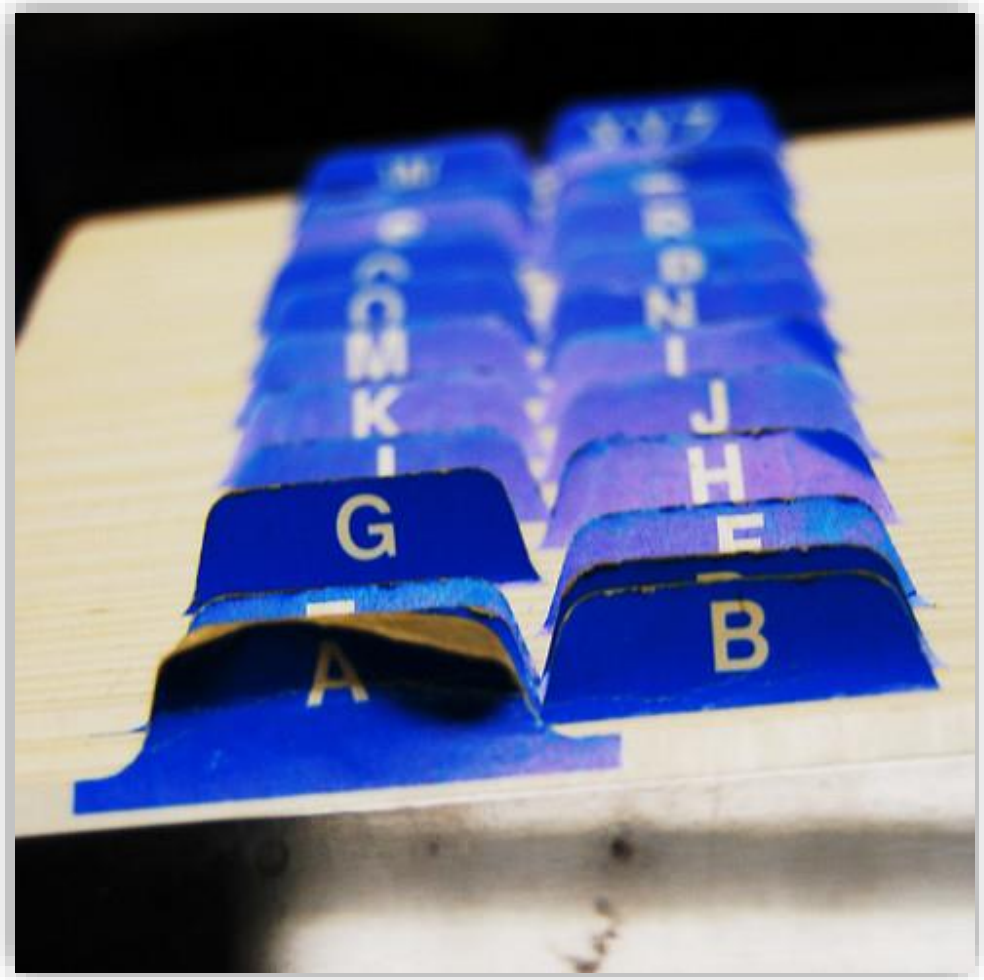
An index is an on-disk structure associated with a table that speeds retrieval of rows.

An index contains keys built from one or more columns in the table.

What is SARGability?

A SARGable item in a search predicate is able to use an index.

Non-SARGable expressions can significantly slow down queries.



How Data is Stored in Data Pages

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.

Heap

AcctID	AcctName	Balance
1	Jack	500.00
2	Diane	750.00
29	Kelli	1250.00
27	Jessica	1005.00
18	Maddison	745.00
22	Bella	445.00

Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.

Clustered Index

AcctID	AcctName	Balance
1	Jack	500.00
2	Diane	750.00
12	Danny	630.00
14	Mayleigh	204.00
15	Molly	790.00
18	Maddison	745.00

Heap

A heap is a table without a clustered index

Unordered masses of data

Can be good for quickly importing large sets of data

Not a good idea for reporting-based data structures

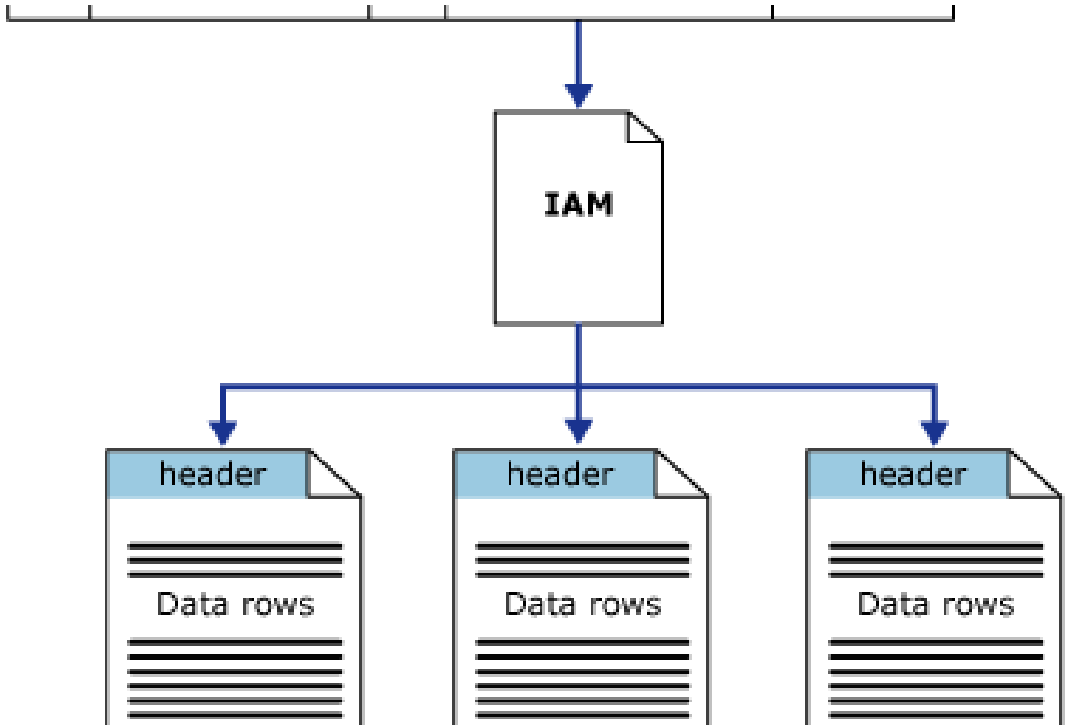
Use the ALTER TABLE…REBUILD command to “rebuild” a heap table

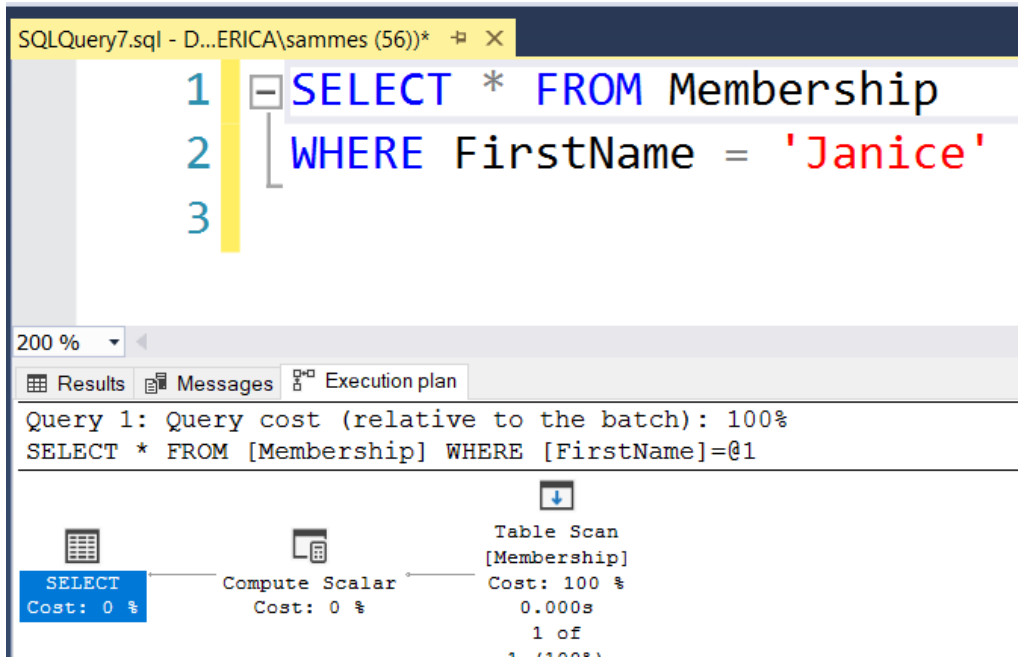
Do not use a heap

- When the data is frequently returned in a sorted order.
- When the data is frequently grouped together.
- When ranges of data are frequently queried from the table.
- When there are no nonclustered indexes and the table is large.

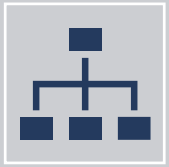
Heap Structures

Heaps have one row in sys.partitions, with index_id = 0 for each partition used by the heap





Clustered Indexes

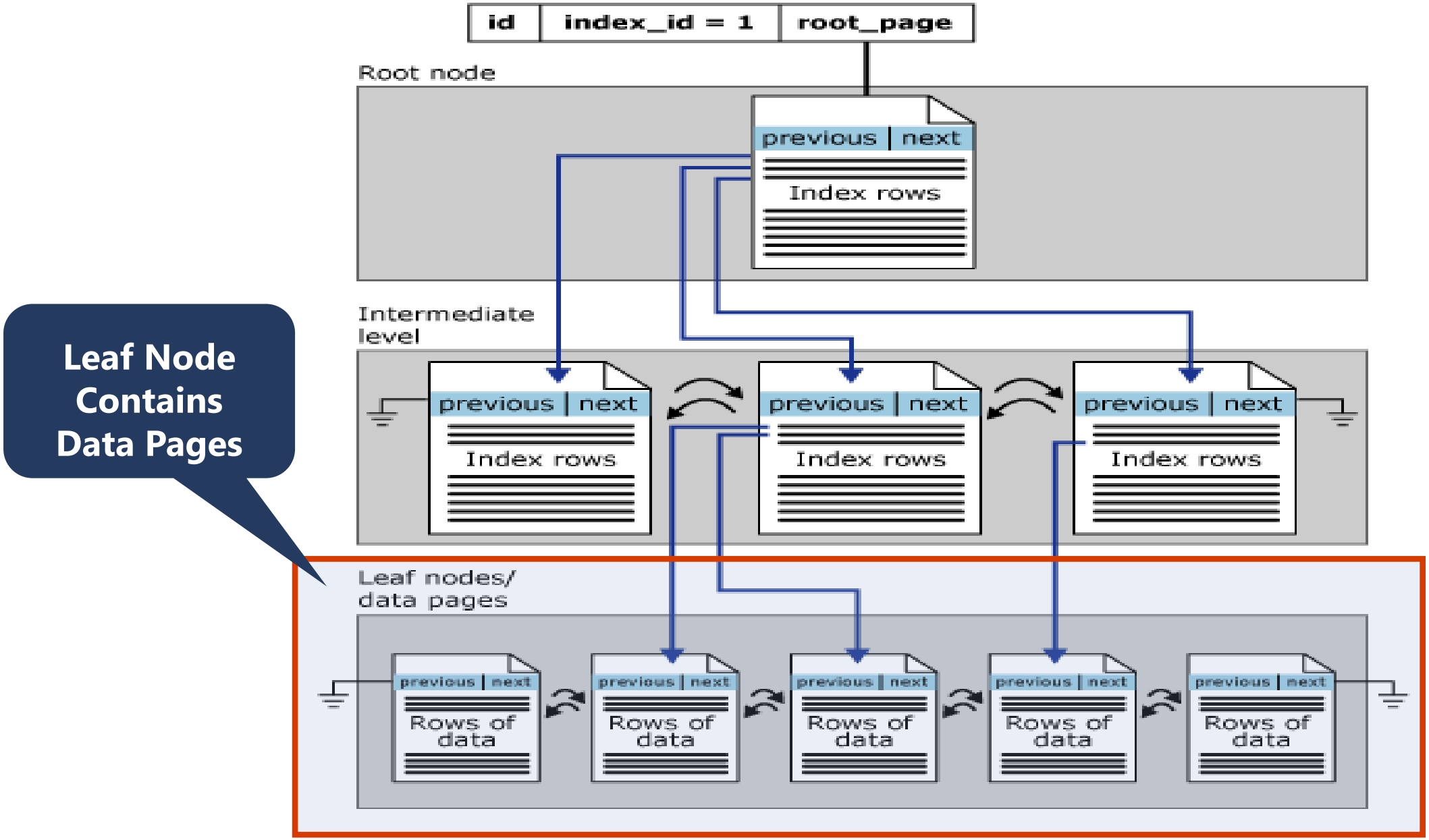


An ordered data structure that is implemented as a Balanced (B) Tree.

When a table is clustered, the leaf level of the index contains ALL data in the table. This means that the clustered index IS the table. This is also why there is only one per table.

The leaf level of the clustered index contains data pages.

Clustered Index Structure



Non-Clustered Indexes

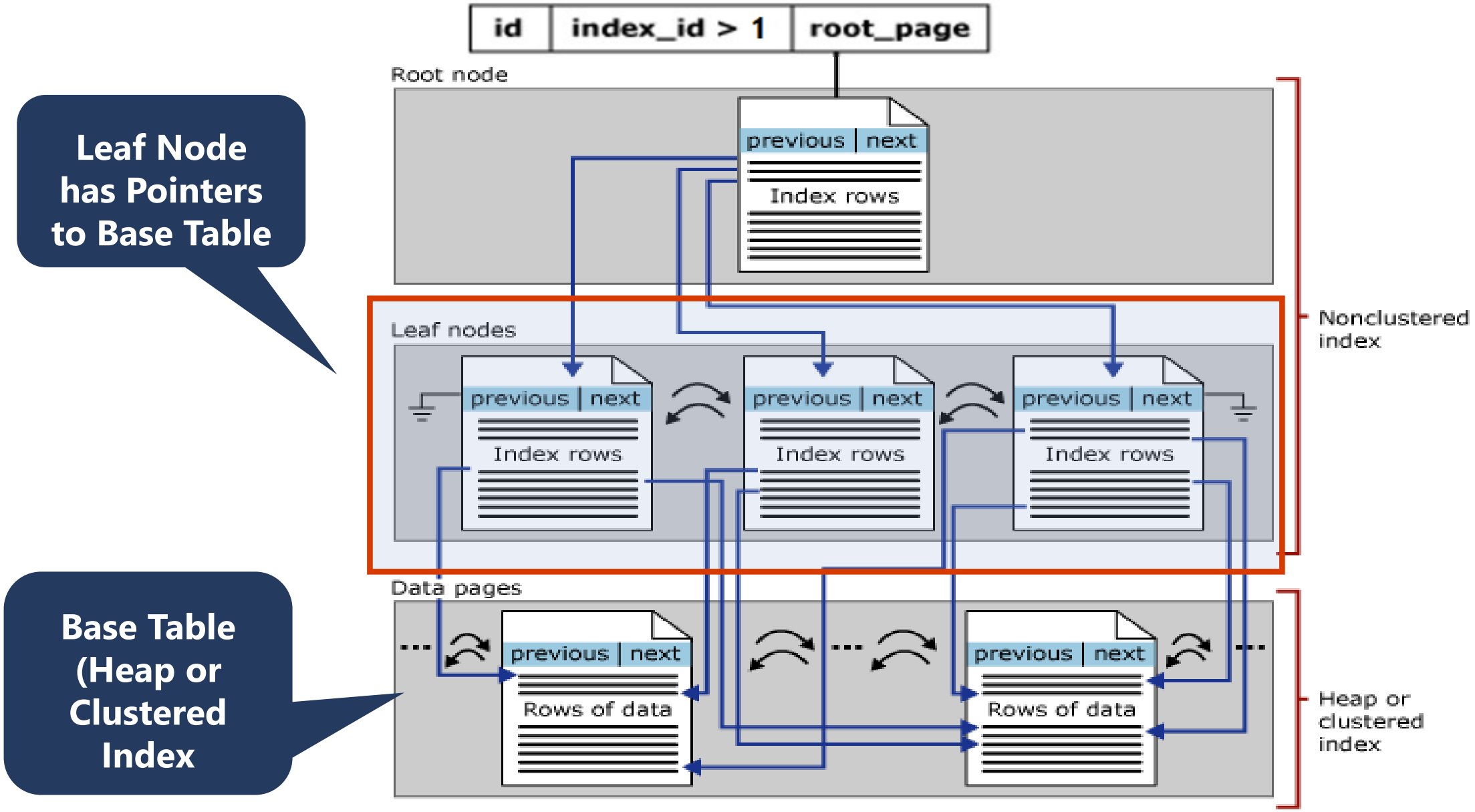
Same B-tree data structure as a clustered index.

It is a separate structure built on top of a Heap or Clustered Index.

Only contains a subset of the columns in the base table

The leaf level contains only the columns defined in the index as well as the clustered key/heap ID that points to the base table structure.

Non-Clustered Index Structure



Clustered vs Nonclustered Indexes

An index is an on-disk structure associated with a table or view that speeds retrieval of rows.

Clustered Indexes

- Defines the order in which data is physically stored in a table.
- Table data can be sorted in only one way.
- Leaf level has data rows stored with index.
- When a table has a clustered index, the object is called clustered table.
- Cluster key is added to nonclustered index (as the pointer), keep it as narrow as possible.

Non-clustered Indexes

- Separate structure from base table.
- Contains a pointer back to base table called:
  - Row ID (RID when base table is HEAP)
  - Key (KEY when base table is Clustered)
- "Skinny" data structure as it contains a subset of base table only.
- To by-pass index key limits (1,700 bytes) ,non-key columns can be added to leaf level.
- As Leaf level contains fewer columns than base table, the non-clustered index uses fewer pages than the corresponding base table.

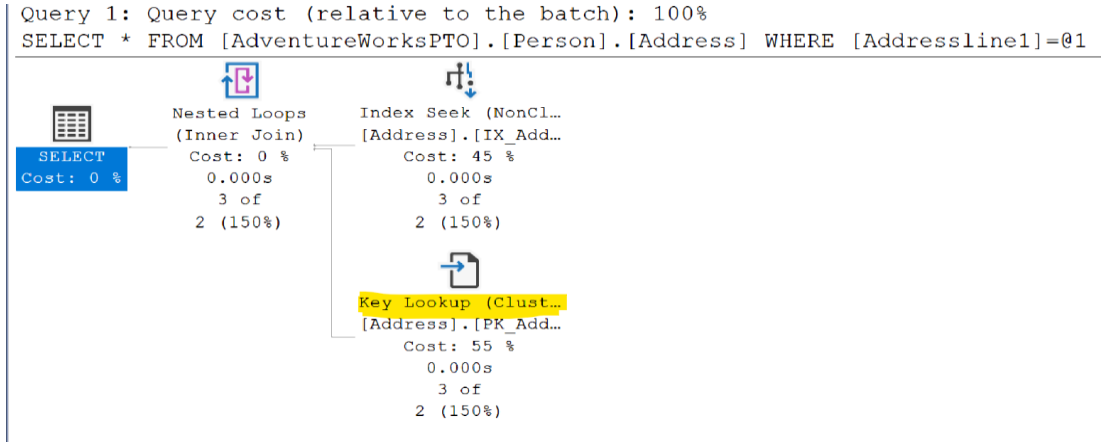
Bookmark Lookup or RID lookup

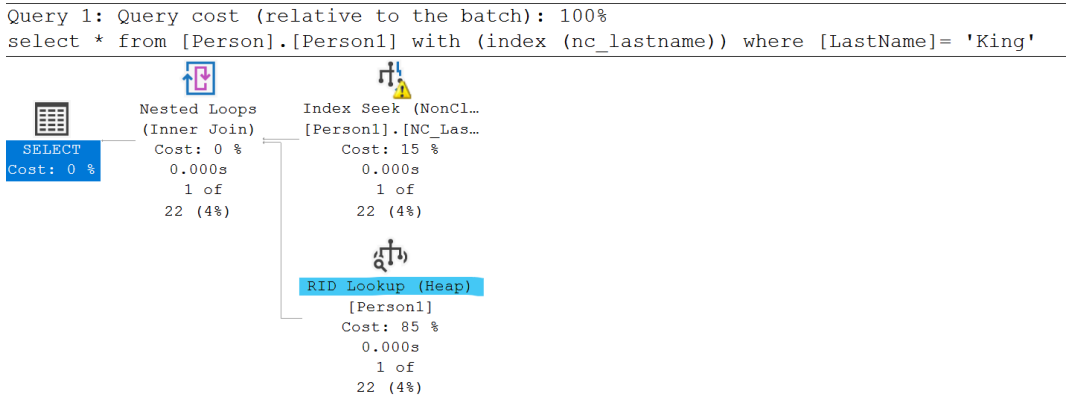
Occurs when SQL uses a nonclustered index to satisfy all or some of a query's predicates, but it doesn't contain all the information needed to cover the query.

Lookup effectively join the nonclustered index back to the clustered index or heap.

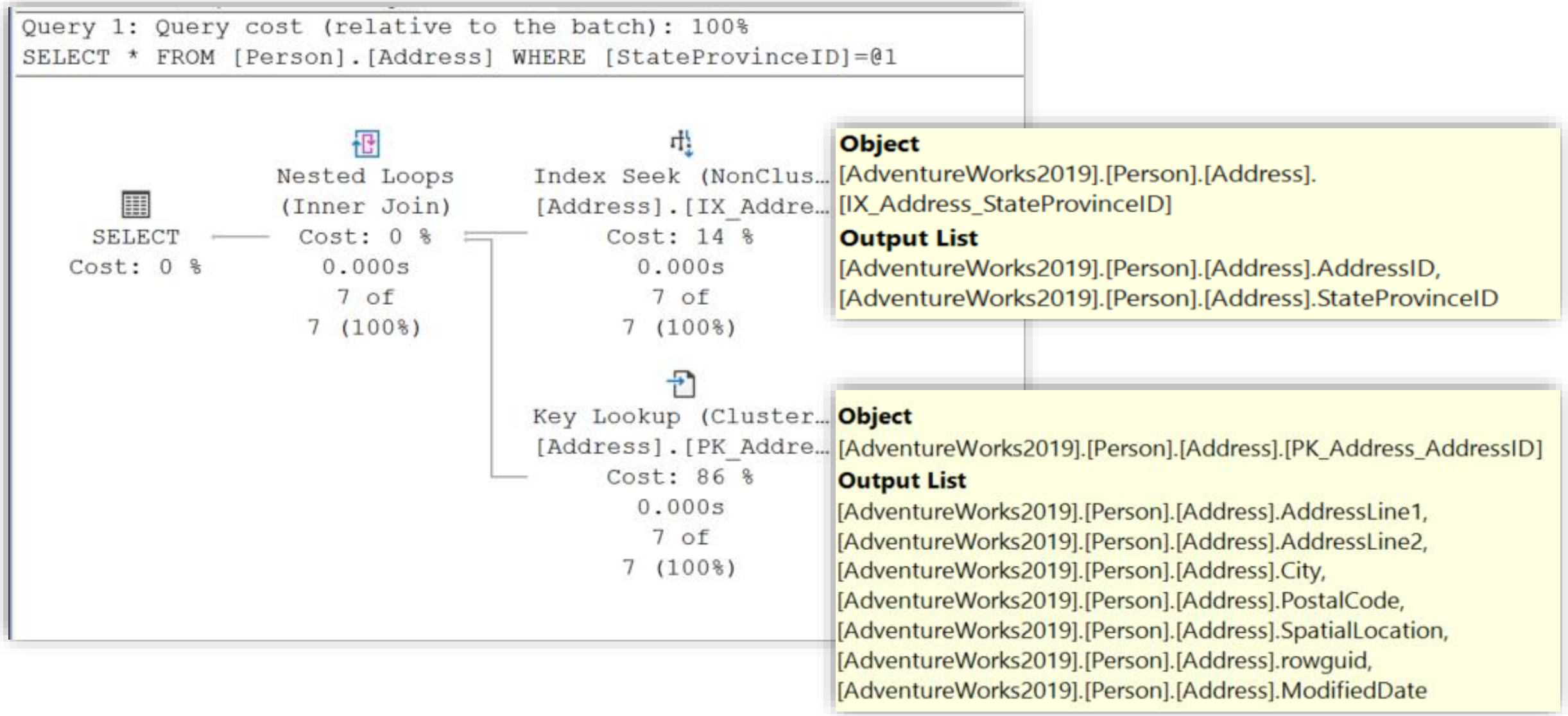
- If table has clustered index, it is called **bookmark lookup** (or key lookup)
- If the table is a heap with a non-clustered index, it is called **RID lookup**

This is an expensive operation.

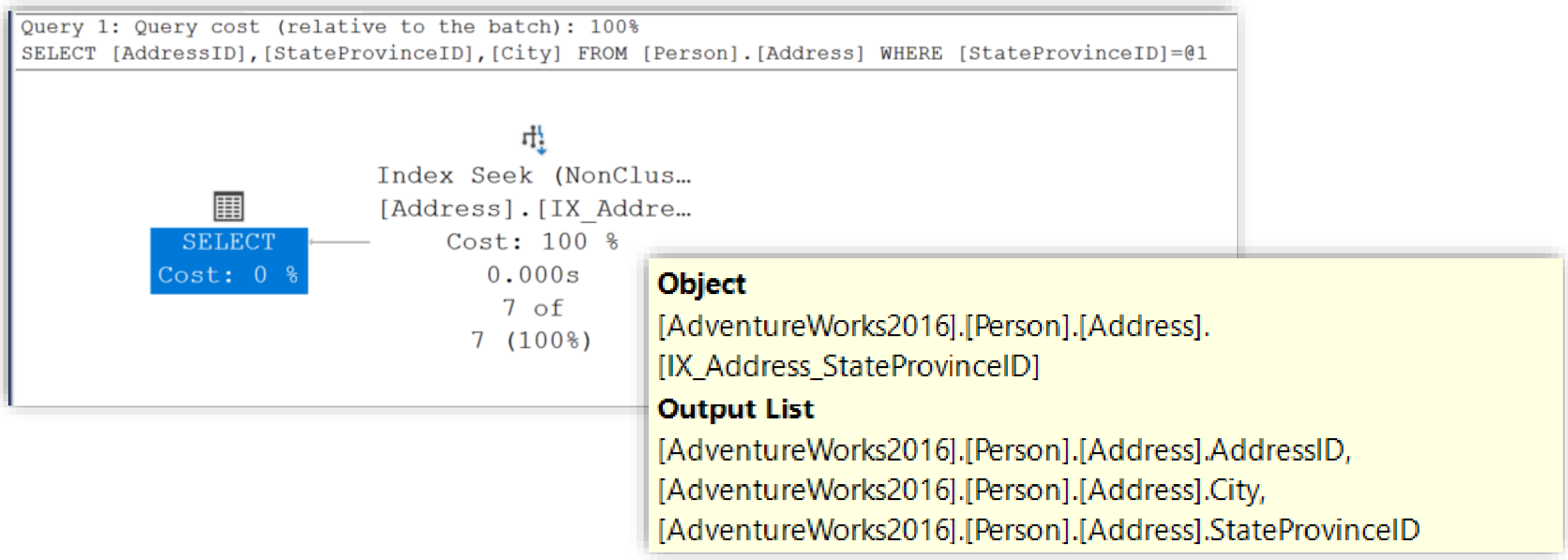




Key Lookup


Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Person].[Address] WHERE [StateProvinceID]=@1
SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
0.000s
7 of
7 (100%)
Index Seek (NonClus…
[Address].[IX_Addre…
Cost: 14 %
0.000s
7 of
7 (100%)
Key Lookup (Cluster…
[Address].[PK_Addre…
Cost: 86 %
0.000s
7 of
7 (100%)
Object
[AdventureWorks2019].[Person].[Address].
[IX_Address_StateProvinceID]
Output List
[AdventureWorks2019].[Person].[Address].AddressID,
[AdventureWorks2019].[Person].[Address].StateProvinceID
Object
[AdventureWorks2019].[Person].[Address].[PK_Address_AddressID]
Output List
[AdventureWorks2019].[Person].[Address].AddressLine1,
[AdventureWorks2019].[Person].[Address].AddressLine2,
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].PostalCode,
[AdventureWorks2019].[Person].[Address].SpatialLocation,
[AdventureWorks2019].[Person].[Address].rowguid,
[AdventureWorks2019].[Person].[Address].ModifiedDate

Non-Clustered Index with Included Column



Included columns for non-clustered indexes

Included as additional non-key columns of data in the leaf level.

Allows for covering more queries.

As query optimizer can locate column values in index (leaf level) hence performance gain is achieved.

Not restricted to a Maximum of 32 key columns and index key size of 1,700 bytes.

(n)varchar(max) can be used, but not (n)text or image data types.

Index search is not permitted on non-key columns.

Lesson 2: Index Strategy

Characteristics of a Good Clustering Key

Narrow

- Use a data type with a small number of bytes to conserve space in tables and indexes

Unique

- To avoid SQL adding a 4-byte uniquifier

Static

- Allows data to stay constant without constant changes which could lead to page splits

Increasing

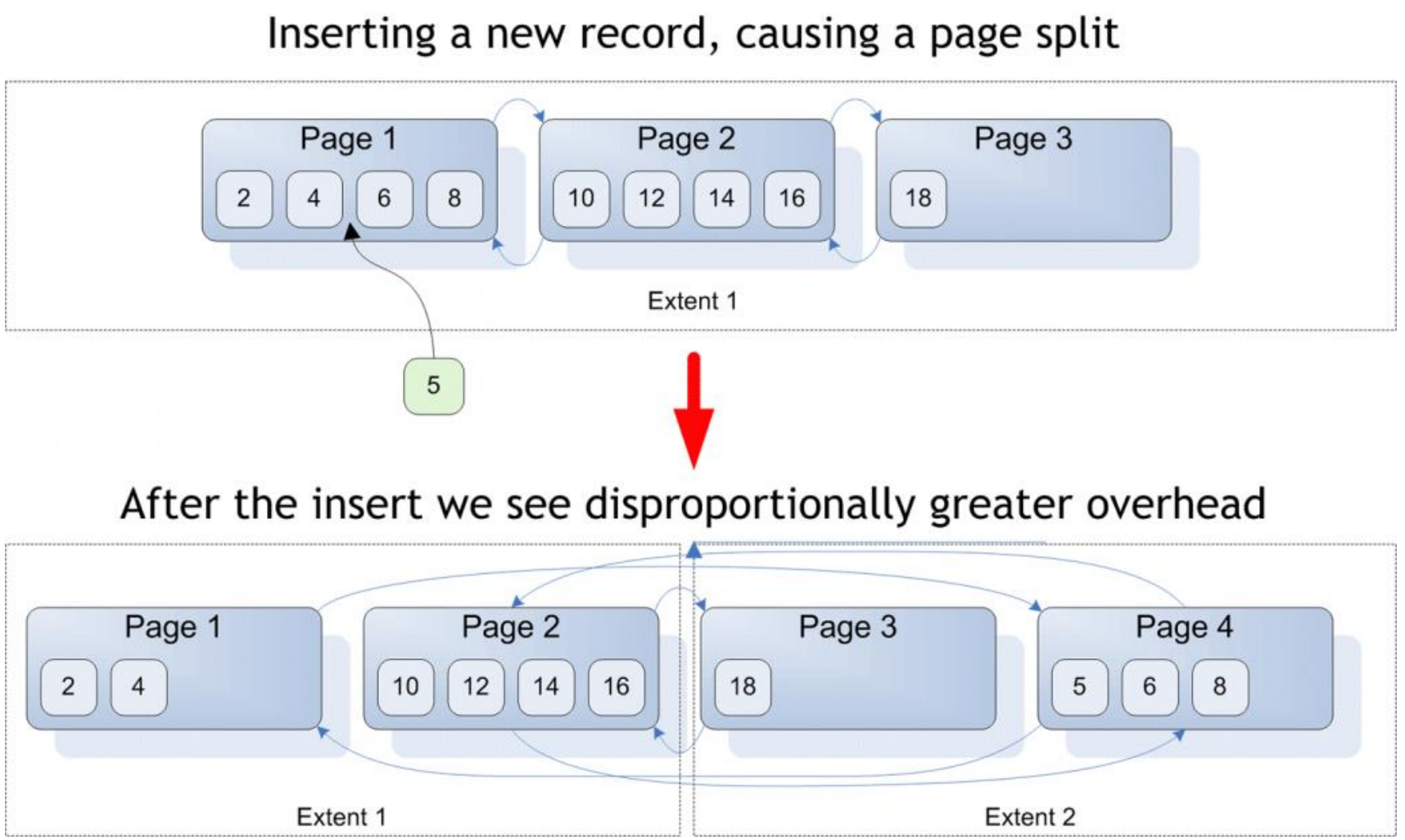
- Allows better write performance and reduces fragmentation issues

Best Practices for Clustered indexes

The primary goal of any indexing strategy is to make queries faster

Best Practice	Seeks, not scans	Avoid key lookups	Get most rows on the index page	Get most rows on the data page	Avoid page splits
Use frequently searched value	X	X			
Use narrow keys			X	X	
Use unique values			X	X	
Use static values					X
Use ascending or descending key					X

Page Splits


Inserting a new record, causing a page split
Page 1
2
4
6
8
Page 2
10
12
14
16
Page 3
18
Extent 1
5
After the insert we see disproportionally greater overhead
Page 1
2
4
Page 2
10
12
14
16
Page 3
18
Page 4
5
6
8
Extent 1
Extent 2

Non-Clustered Index Strategy

Index columns used in WHERE clauses and JOINs.

Keep indexes to a minimum, to minimize impact on DML and log writes.

Two basic approaches: Single column indexes or Multicolumn indexes

Easy to create too many nonclustered indexes.

Specific indexes are appropriate for high-impact queries.

Ideally the focus should be writing queries to use existing indexes, rather than on adding more indexes.

Single Vs Multi-column indexes

Single-column Indexing

- Good choice for columns that are highly selective, or columns referenced often by criteria.
- When filtering data from table with multiple single column indexes, SQL uses the index with higher cardinality column, it may be better to not index the low cardinality column.

Multi-Column Indexing

- Good choice for low cardinality columns.
- Highest cardinality field first, to zoom into a narrow range of the B-Tree as quickly as possible.
- Equality columns before inequality columns.

Multi-Column Indexing Access

Seek only happens if you search for the columns in the specified order.

CREATE INDEX IX1 ON TABLE (PostalCode, StateID, City)

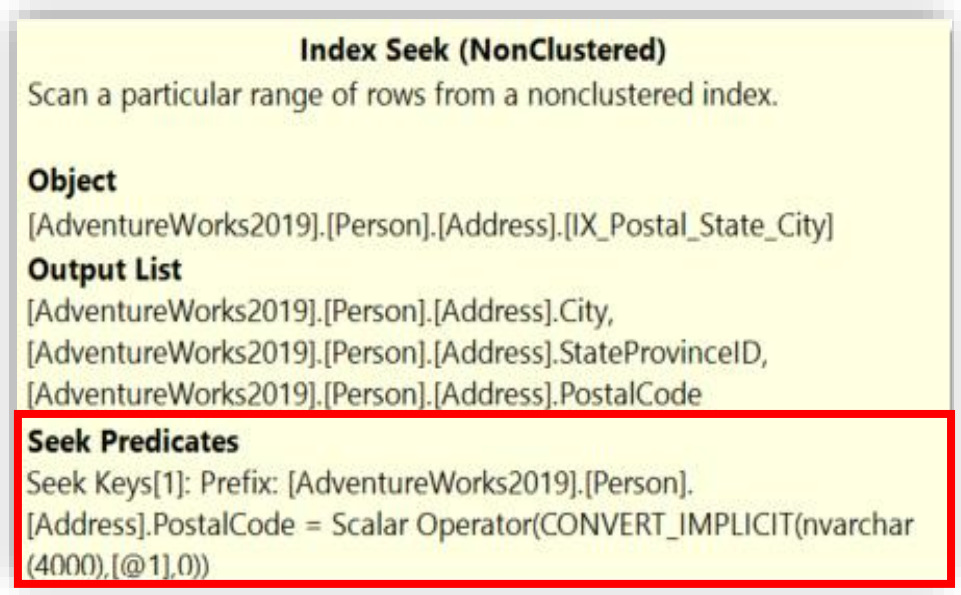
Effect of column in different WHERE clause.

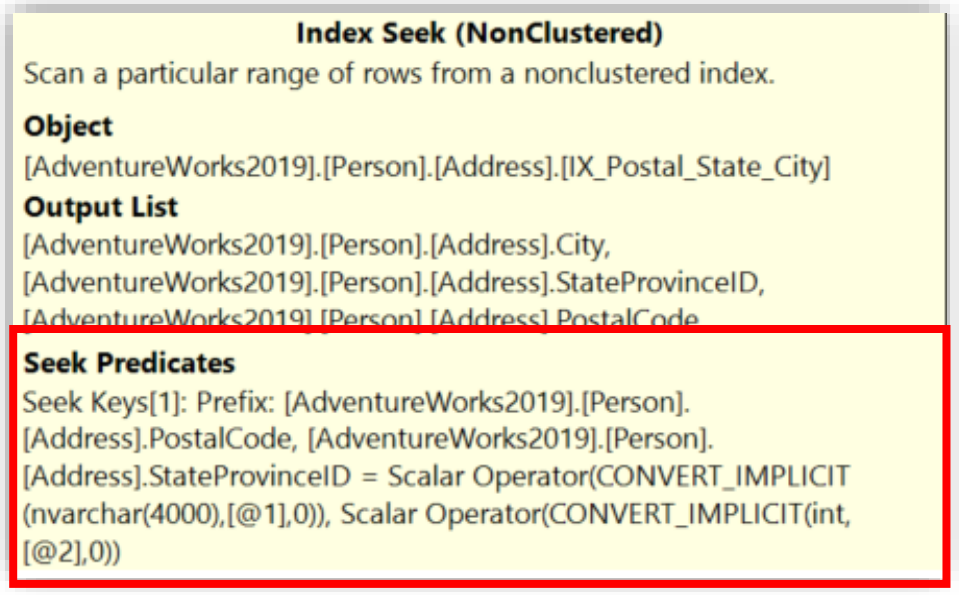
- WHERE PostalCode = 98011 – seek
- WHERE PostalCode = 98011 AND StateID = 79 – seek both
- WHERE PostalCode = 98011 – seek AND City = Bothell -- scan
- WHERE StateID = 79 -- scan

Multi-Column Indexing Access (Seek Predicates)

```
--Single value performs Index Seek.
SELECT City, StateProvinceID,
PostalCode
FROM Person.Address
WHERE PostalCode = '98011'
```

```
--Index Seek on both columns
--Search condition in same order as Index.
SELECT City, StateProvinceID, PostalCode
FROM Person.Address
WHERE PostalCode = '98011' AND
StateProvinceID = 79
```





Multi-Column Indexing Access (Scan Predicates)

```
--Index Seek on first column
--After seek, will scan second column
SELECT City, StateProvinceID, PostalCode
FROM Person.Address
WHERE PostalCode = '98011' AND City =
'Bothell'
```

```
--Search condition not in same order as
Index.
--Performs Index Scan.
SELECT  City, StateProvinceID, PostalCode
FROM Person.Address
WHERE StateProvinceID = 79
```

Index Seek (NonClustered)

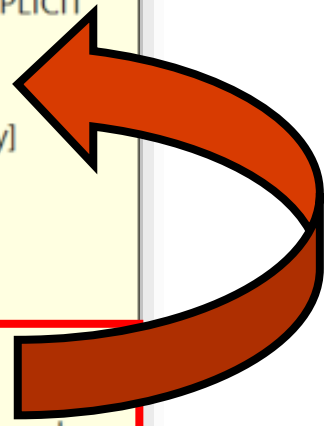
Scan a particular range of rows from a nonclustered index.

Predicate
[AdventureWorks2019].[Person].[Address].[City]=CONVERT_IMPLICIT
(nvarchar(4000),[@2],0)

Object
[AdventureWorks2019].[Person].[Address].[IX_Postal_State_City]

Output List
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].StateProvinceID,
[AdventureWorks2019].[Person].[Address].PostalCode

Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2019].[Person].
[Address].PostalCode = Scalar Operator(CONVERT_IMPLICIT(nvarchar
(4000),[@1],0))



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Predicate
[AdventureWorks2019].[Person].[Address].[StateProvinceID]=(79)

Object
[AdventureWorks2019].[Person].[Address].[IX_Postal_State_City]

Output List
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].StateProvinceID,
[AdventureWorks2019].[Person].[Address].PostalCode

Lesson 3: Index Monitoring and Fragmentation

Why to monitor Index usage

Data access and distribution changes over time.

By reviewing index usage, we can ensure that the indexes are appropriate for the current workload.

Use sys.dm_db_index_usage_stats and sys.dm_db_index_operational_stats to monitor index usage.

Database standard report also monitors Index usage and Physical Statistics.

Clustered Index usage patterns

Pattern	Interpretation
Seeks, but no scans	• May not need maintenance. Seeks are not impacted by fragmentation
Clustered index with High number of scans	• Review missing index recommendations. • Consider changing Clustering key. • Simplify queries with complex WHERE clauses
Clustered index with Low number of seeks and high number of lookups	• May need to change Clustering Key • Look how often bookmark lookups occur.

Non-Clustered Index usage patterns

Pattern	Interpretation
Seeks, but no scans	• May not need maintenance. • Seeks are not impacted by fragmentation
Non-Clustered index with High number of scans	• Consider compression. • Check for non-SARGable queries. • Review missing index recommendations
Index with No/low usage	• May be able to drop or disable index
Non-Clustered Index Used at specific times only	• May be able to drop index, recreate when needed

sys.dm_db_index_usage_stats

- Tracks seeks, scans, and lookups
- Provides date of last access
- Counters are incremented once per query execution
- User and system access (such as index rebuilds) are tallied separately
- Counters are initialized to at SQL Server (MSSQLSERVER) service is restarts

```
SELECT
        Distinct OBJECT_NAME(Indx.OBJECT_ID) Table_Name  ,Indx.name  IndexName
        ,Indx.type_desc IndexType,Iusg.user_seeks  Seeks,Iusg.user_scans  Scans
        ,Iusg.user_lookups  Lookups ,Iusg.user_updates Updates ,Iusg.last_user_seek LastSeek
        ,Iusg.last_user_scan LastScan,Iusg.last_user_lookup LastLookup ,Iusg.last_user_update LastUpdate
FROM
        SYS.INDEXES Indx
        INNER JOIN SYS.DM_DB_INDEX_USAGE_STATS Iusg
                ON Iusg.index_id = Indx.index_id AND Iusg.OBJECT_ID = Indx.OBJECT_ID
        INNER JOIN SYS.DM_DB_PARTITION_STATS PrtSts
                ON PrtSts.object_id=Indx.object_id
WHERE
        OBJECTPROPERTY(Indx.OBJECT_ID,'IsUserTable') = 1
```

sys.dm_db_index_physical_stats

- Returns size and fragmentation information for the data and indexes of the specified table or view in SQL Server.
- Takes database_id, the object_id, the index_id and the partition_number as parameters
- Memory-optimized indexes do not appear in this DMV

```
SELECT OBJECT_SCHEMA_NAME(I.object_id) AS SchemaName, OBJECT_NAME(I.object_id) AS
TableName, I.name, I.Index_ID, IPS.partition_number, IPS.avg_fragmentation_in_percent,
   IPS.page_count, IPS.avg_page_space_used_in_percent
FROM sys.indexes as I
INNER JOIN sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'Limited') as IPS
ON I.object_id = IPS. object_id
AND I.index_id = IPS.index_id
--WHERE IPS.avg_fragmentation_in_percent >30
--AND IPS.page_count >1000
```

Fragmentation

A fragmented table/Index is when some of its data pages point to pages that are not in sequence.

Logical fragmentation

- Occurs when leaf level pages are not physically corresponding to the logical order of the index:
  - Pages are not in the most efficient order for scanning purposes.
- Limits the efficiency of read-ahead scans, but not seeks.

Page density

- How full a page is when a rebuild/reorganization occurs.
- The fuller a page is, the more likely page splits occur when data is modified.
- The less full a page is, the more wasted space in the buffer pool when reading pages.

Page splits

Mechanism to optimize inserts and updates

Occurs when page is full to hold a new or updated row

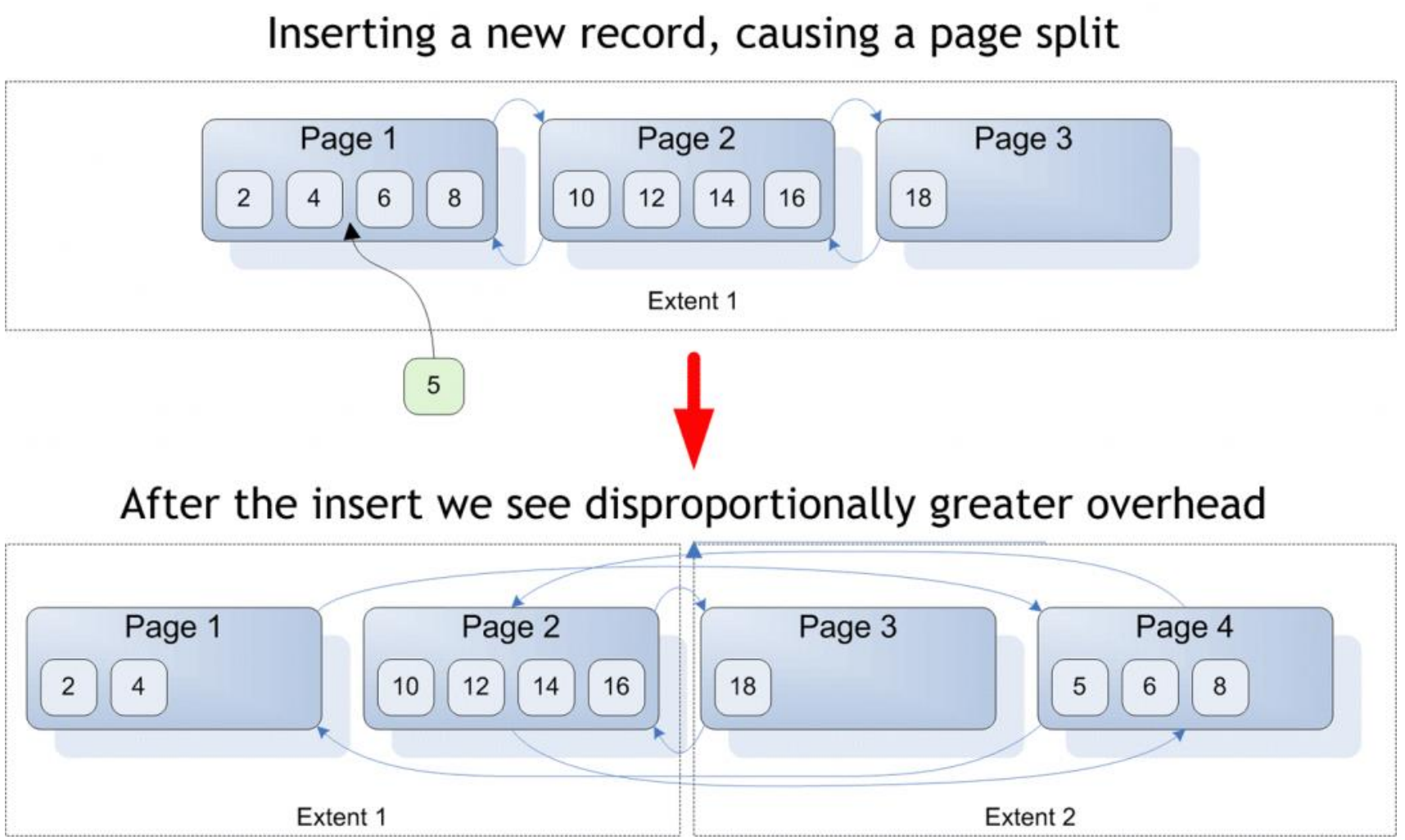
Half the rows are moved to a new page

The new page is added after the last existing page

Cost associated with Page splits

- Fragmentation, which affects scan performance
- Wasted free space, which affects Page Density

Page Splits


Inserting a new record, causing a page split
Page 1
2
4
6
8
Page 2
10
12
14
16
Page 3
18
Extent 1
5
After the insert we see disproportionally greater overhead
Page 1
2
4
Page 2
10
12
14
16
Page 3
18
Page 4
5
6
8
Extent 1
Extent 2

SQL Index Fragmentation and Maintenance options

SSMS -> Index Physical Statistics report

- Will report fragmentation and recommend solution at database level

Custom Solution

- Find fragmentation using Sys.dm_db_index_physical_stats
- Correct fragmentation

Can remove Index fragmentation using Rebuild or Reorganize option

	ALTER INDEX... REBUILD	ALTER INDEX ... REORGANIZE
Removes fragmentation	X	X
Removes free space	X	
Resets fill factor	X	
Updates statistics	X	

Fill Factor

Can address performance issues due to fragmentation.

Use When	Do not use when
• Specifying the amount of free space on a data or index page. • Reducing logical fragmentation.	• Low fragmentation • High seeks and low scans • Index is not scanned

```
ALTER INDEX [pk_bigProduct]
ON [dbo].[bigProduct]
REBUILD WITH (PAD_INDEX = ON, FILLFACTOR = 90)
```

Reducing fragmentation

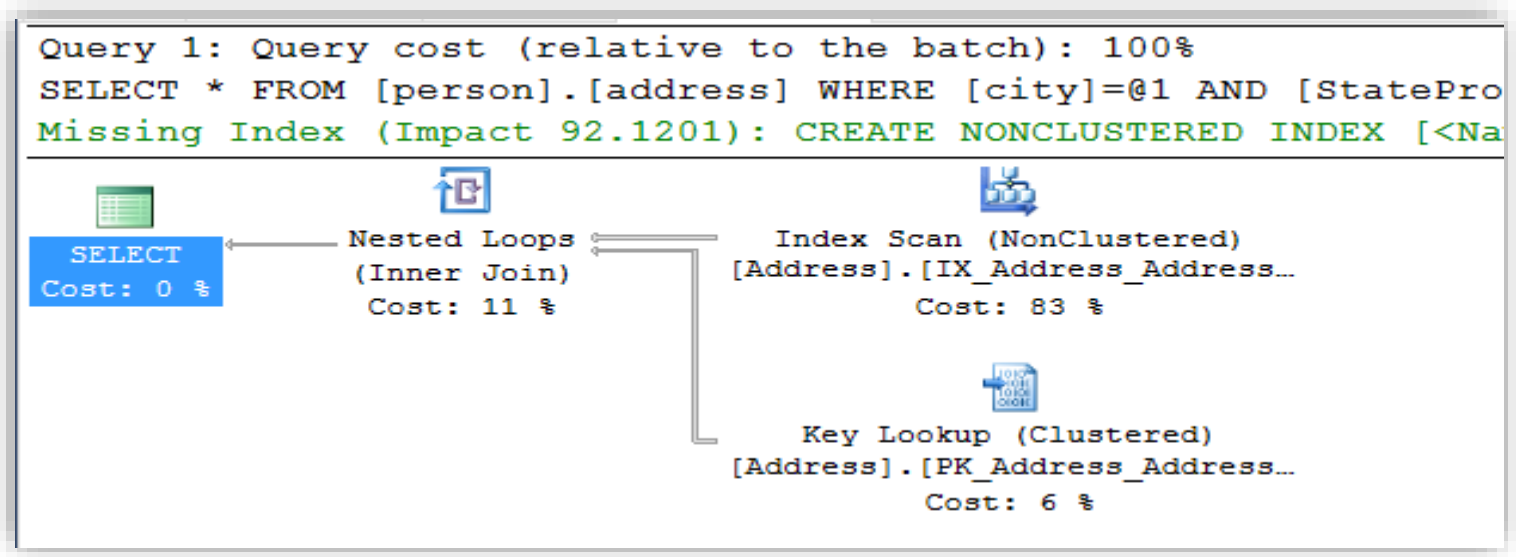
Reducing Fragmentation
• Using an ascending key (not always possible). • Using an appropriate fill factor for the workload. • Update in sets, rather than one at a time. • Do not insert with immediate update.

Missing Index Recommendations

SQL Server identifies indexes that would help a query's performance

The recommendation is included in the query plan

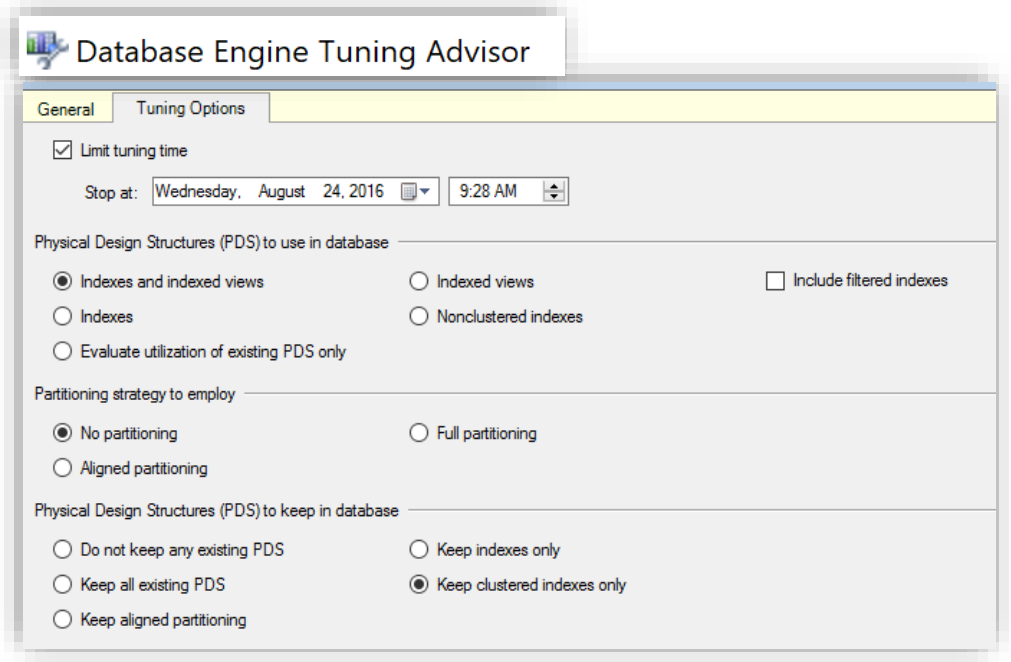
The cost savings are aggregated in DMVs to help identify the most beneficial indexes



Choosing Indexes

Following sources can help you choosing indexes

- Missing Index recommendations
- sys.dm_db_index_usage_stats
- Database Tuning Advisor



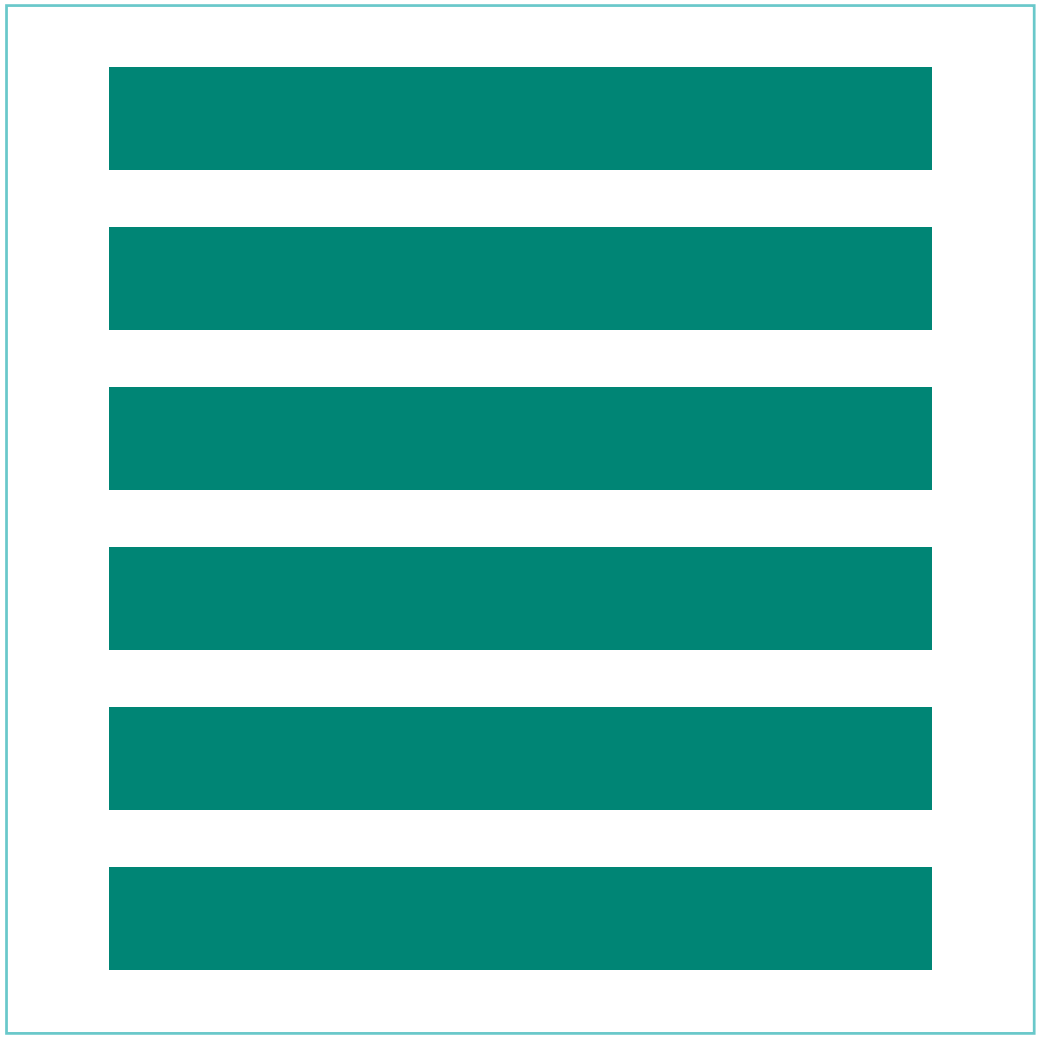
Lesson 4: Columnstore Indexes

Rowstore INDEX design

- Up to 1,000 indexes per table
- Up to 32 columns for index key
- 900 bytes per clustered index
- 1700 bytes per non-clustered index

Regular types of indexes (Clustered / Non-Clustered) wouldn't be able to handle the workload.
It will be impractical to create indexes on all individual columns, or to create all possible column combinations.

RowStore vs ColumnStore



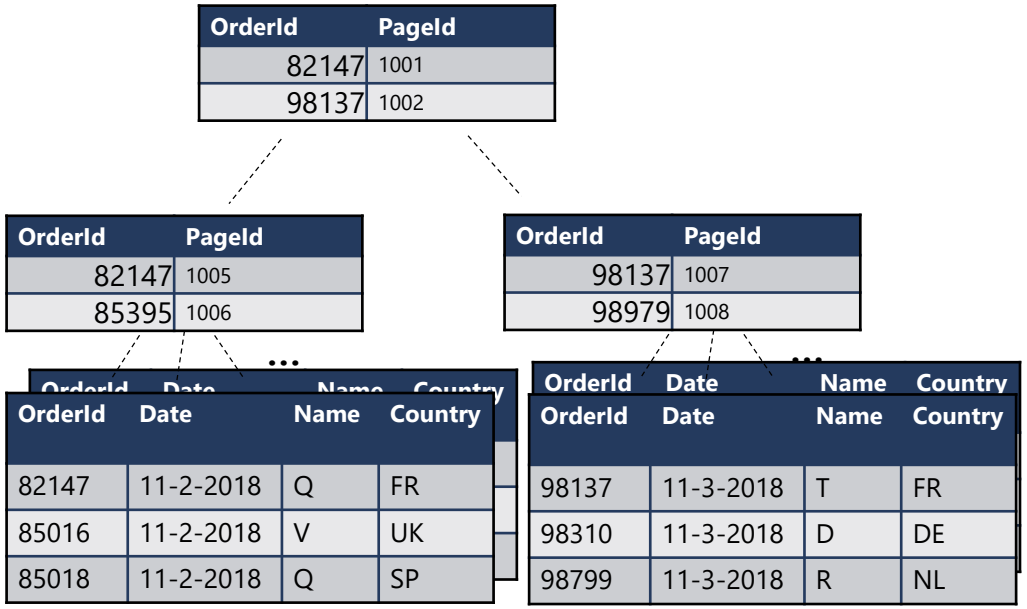
Rowstore vs Columnstore Tables

Logical table structure

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
...	...	...	...

Clustered/Non-clustered rowstore index

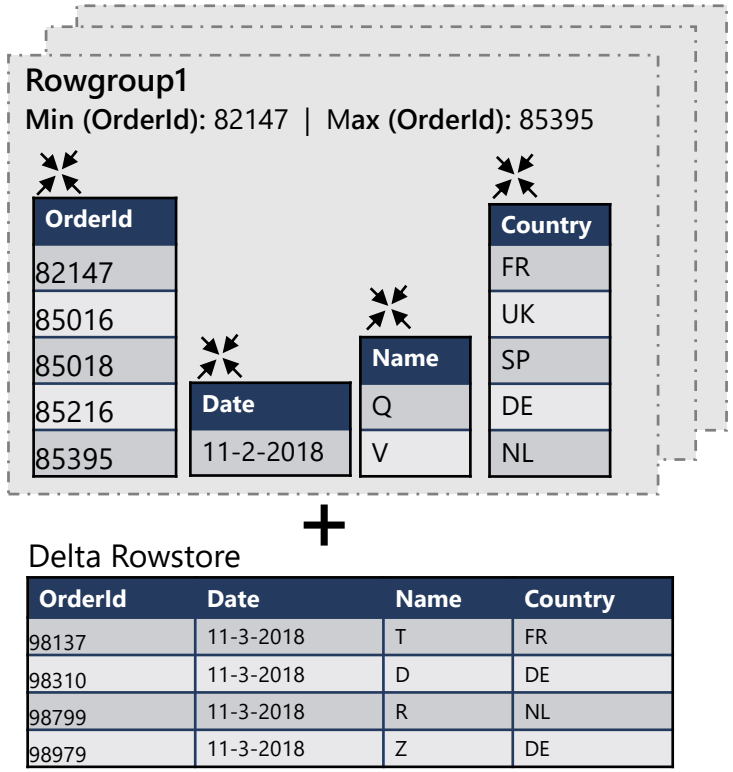
(OrderId)



- Data is stored in a B-tree index structure for performant lookup queries for particular rows.
- Clustered rowstore index: The leaf nodes in the structure store the data values in a row (as pictured above)
- Non-clustered (secondary) rowstore index: The leaf nodes store pointers to the data values, not the values themselves

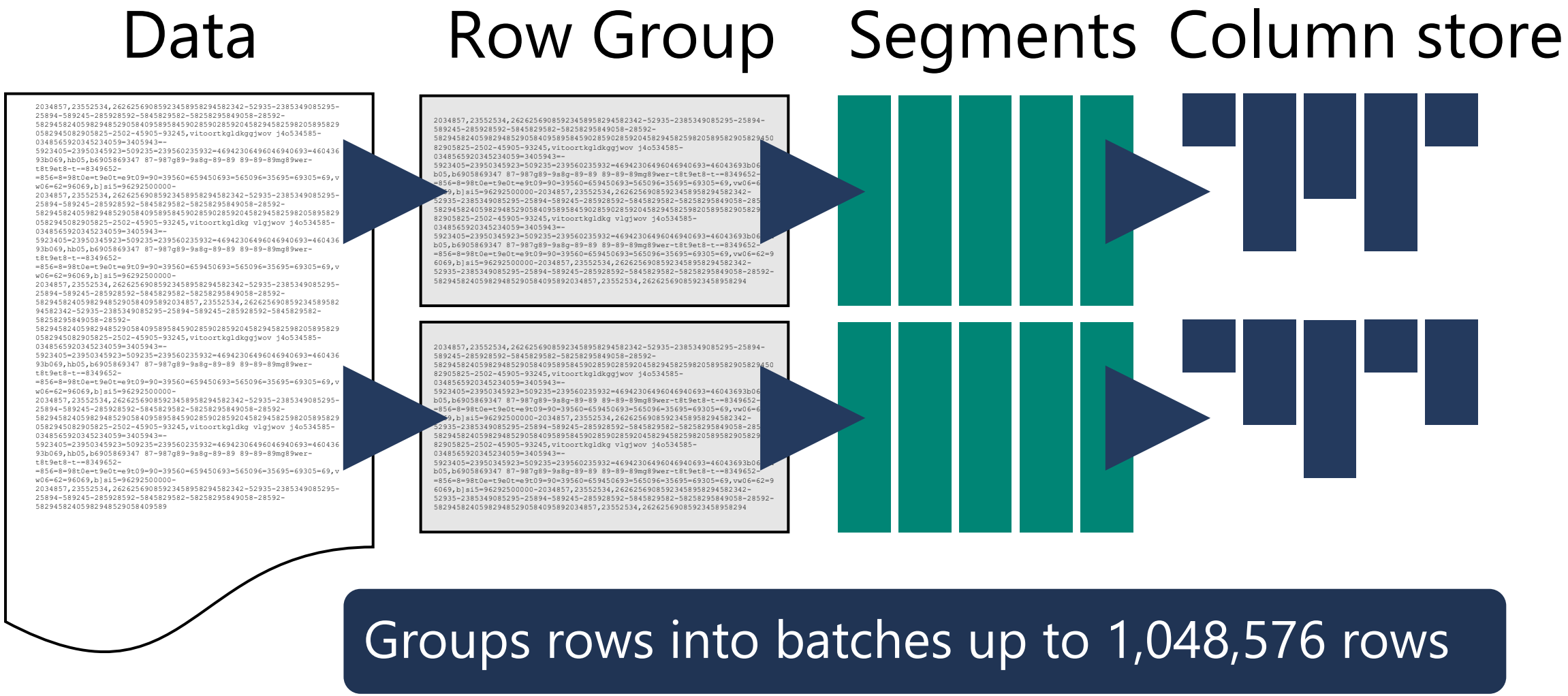
Clustered columnstore index

(OrderId)



- Data stored in compressed columnstore segments after being sliced into groups of rows (rowgroups/micro-partitions) for maximum compression
- Rows are stored in the delta rowstore until the number of rows is large enough to be compressed into a columnstore

Columnstore Index Concept



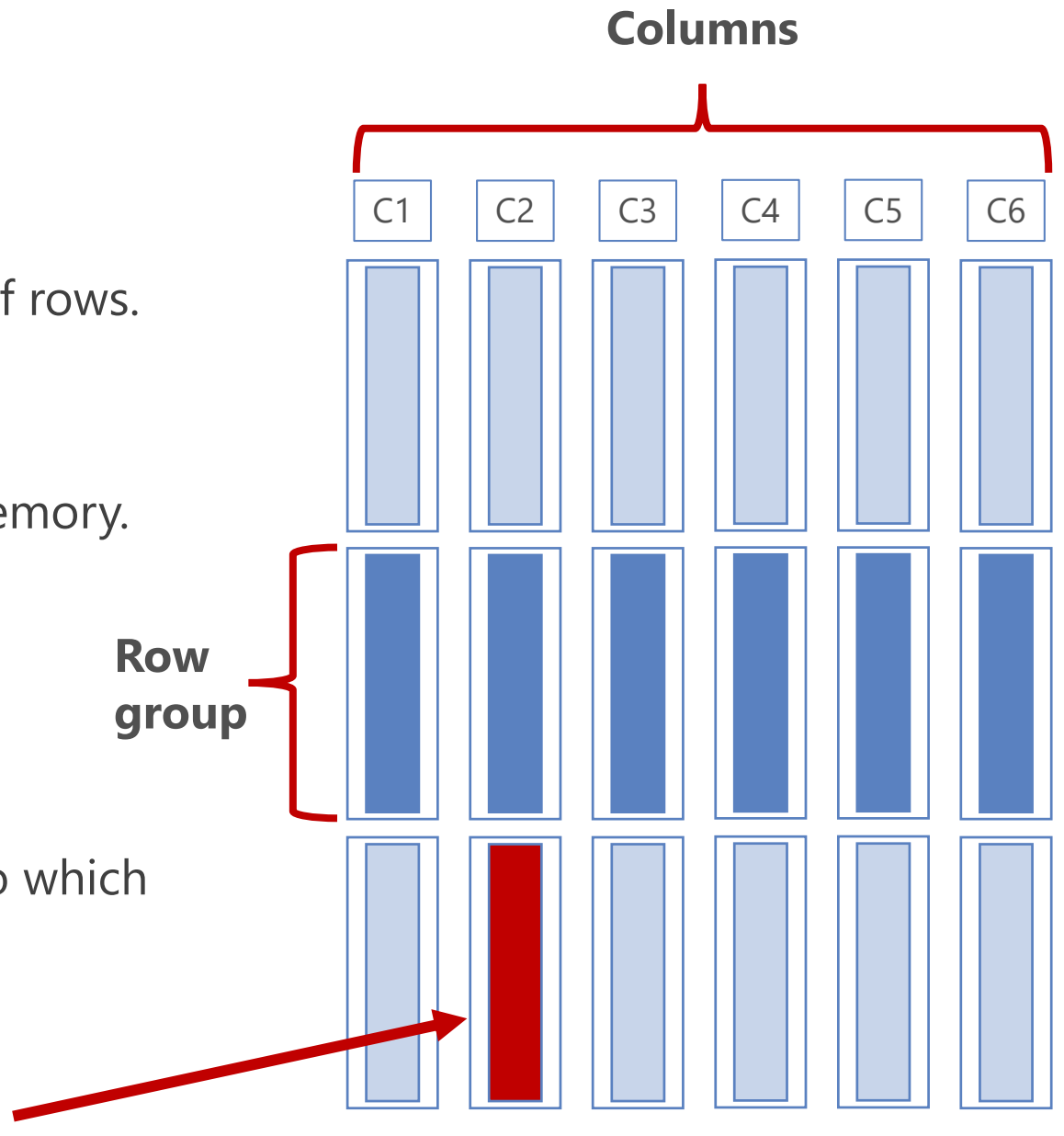
Row Groups & Segments

Segment

- Contains values for one column for a set of rows.
- Segments are compressed.
- Each segment is stored in a separate LOB.
- It is a unit of transfer between disk and memory.

Row Group

- Segments for the same set of rows comprise a row group.
- Position of a value in a column indicates to which row it belongs to.



Columnstore Index Types

Clustered Index

- Use to store fact tables and large dimension tables for data warehousing workloads
- IOT workloads that insert large volumes with minimal updates/deletes
- Average of 10x Compressions
- Covers Entire Table
- May have one or more Nonclustered B-Tree Indexes

Non-Clustered Index

- Use to perform analysis in real time on an OLTP workload.
- Can be created for subset of columns

Columnstore Index Types

SQL Server 2012

- Only Non-Clustered, Non-Updatable Columnstore Indexes.
- Only available in Enterprise Edition.

SQL Server 2014

- Introduced Updatable, Clustered Columnstore Indexes
- Only available in Enterprise Edition.

SQL Server 2016

- Introduced Updatable, Non-Clustered Columnstore Indexes
- Available on Standard Edition. (Service Pack 1)

SQL Server 2019

- Online rebuilds for Clustered Columnstore Indexes.

Delta Store

Page compressed b-tree like a row store

- B-tree on unique integer row ID
- Matches user columns defined in the CCI
- Aligned to underlying CCI partition
- Small bulk loads of less than 102,400 rows, go directly to the deltastore.

Max # rows
delta store
1,048,576

Small DML Against the Column Store

Inserts

- Below threshold? Write to the delta store
- Above threshold? Write as column store

Deletes

- Logical against rows in column store
- Physically against rows in the delta store

Updates

- Converted to a logical delete and an insert

Segment Elimination

Skips large chunks of data to speed up scans

Each partition in a column store index is broken into segments

Each segment has metadata that stores the minimum and maximum value of each column for the segment

The storage engine checks filter conditions against the metadata

If it detects no rows that qualify, it skips the entire segment without reading it from disk

Segment / RowGroup Elimination

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

Rowgroup # 1

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

Rowgroup # 2

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

Segment / RowGroup Elimination

Segment Elimination

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

Rowgroup # 1

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

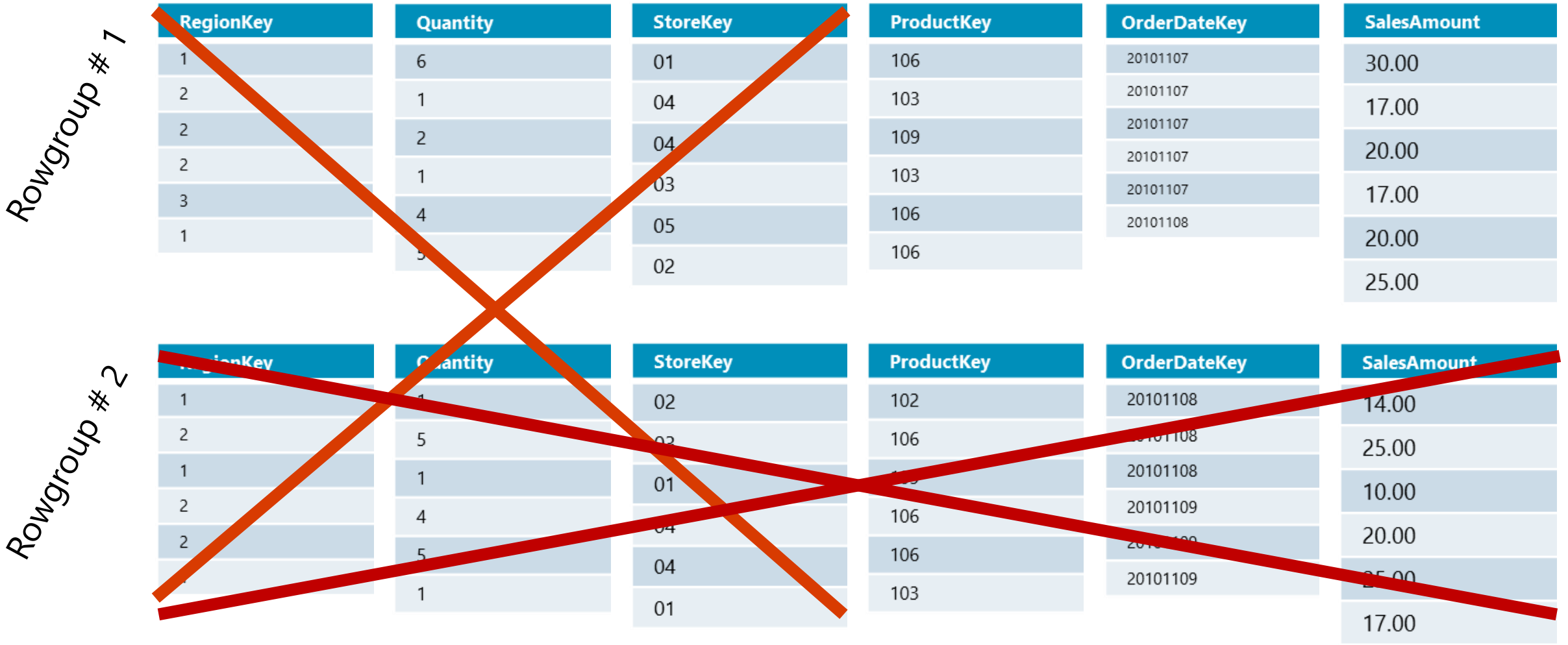
Rowgroup # 2

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

Segment / RowGroup Elimination

Segment Elimination → RowGroup Elimination

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



Segment / RowGroup Elimination

Results

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

Rowgroup # 1

RegionKey
1
2
2
2
3
1

Quantity
6
1
2
1
4
5

StoreKey
01
04
04
03
05
02

ProductKey	OrderDateKey	SalesAmount
106	20101107	30.00
103	20101107	17.00
109	20101107	20.00
103	20101107	17.00
106	20101107	20.00
106	20101108	25.00

Rowgroup # 2

RegionKey
1
2
1
2
2

Quantity
1
5
1
4
5
1

StoreKey
02
03
01
04
04
01

ProductKey	OrderDateKey	SalesAmount
102	20101108	14.00
106	20101108	25.00
109	20101108	10.00
106	20101109	20.00
106	20101109	25.00
103	20101109	17.00

Poor Workloads for CCI

Select * - lose the ability to do column elimination

Point-lookup – easier to find a particular row in a rowstore table

Selecting a range – rowstore can more easily grab that range of rows

A lot of DML – more overhead in CCI.

- UPDATE = DELETE old row, INSERT new row
- DELETE – rows are only logically deleted until an ALTER INDEX REBUILD* is issued

Batch Mode Processing

Process around 1000 rows at a time

As opposed to 1 row at a time with row-based processing

Not all query plan operators can perform batch processing

- nested loops
- merge join

Plan operators are expanded in SQL Server 2016+

Greatly reduced CPU time (7 to 40X)

Microsoft

SQL Server Query Execution and Plans

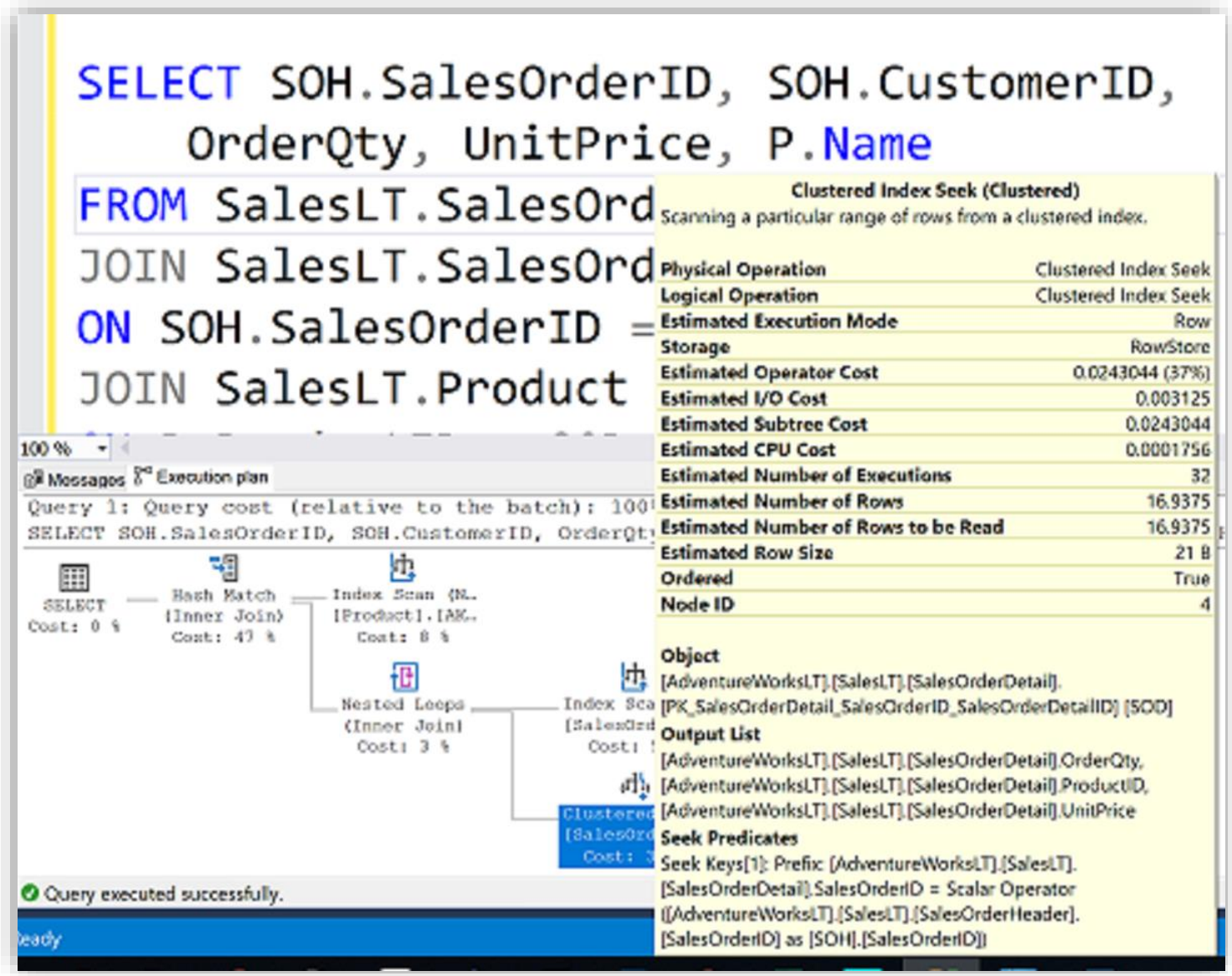
Module 5

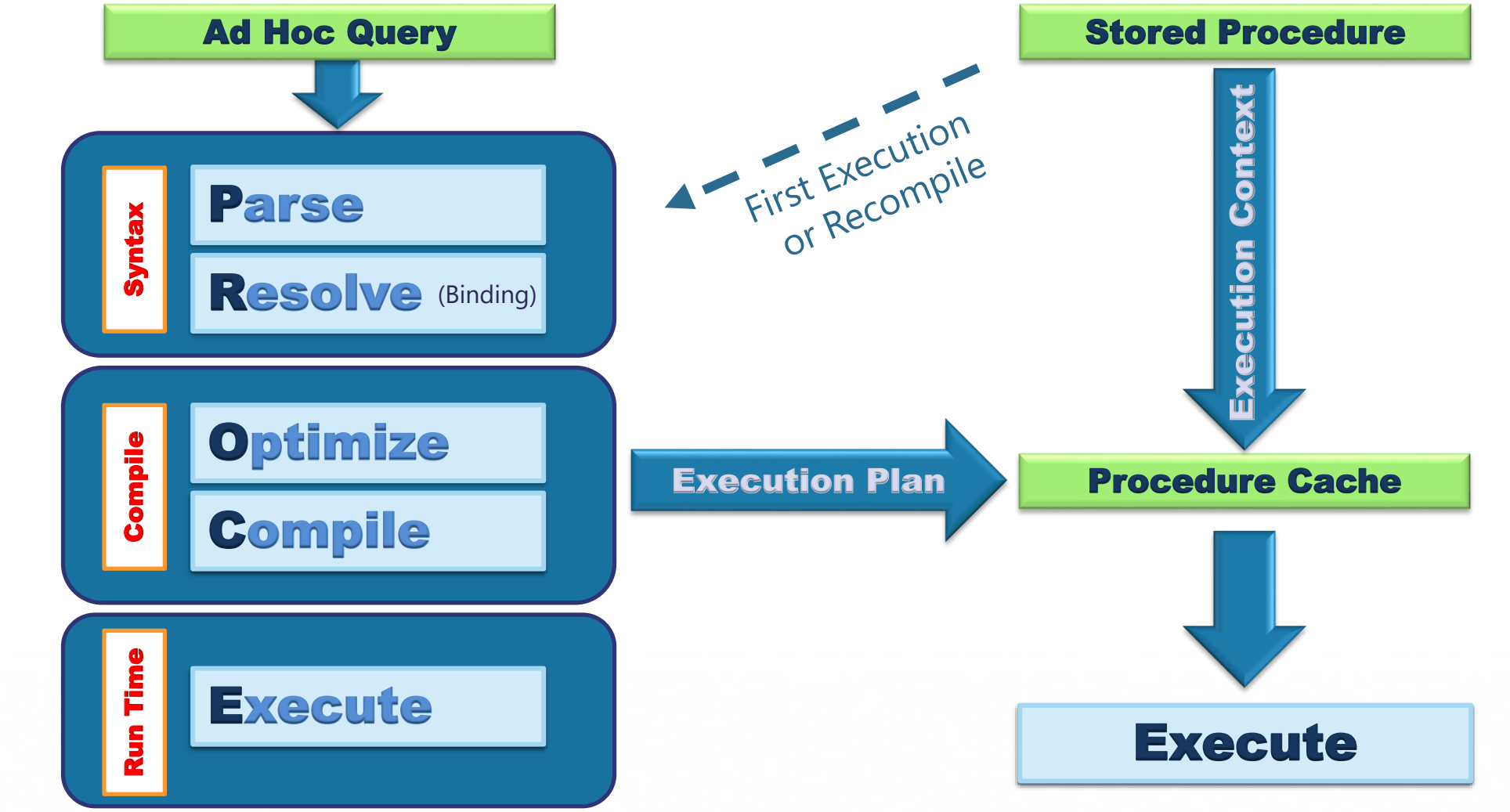
Learning Units covered in this Module

- Lesson 1: SQL Server Query Execution
- Lesson 2: SQL Server Query Plan Analysis
- Lesson 3: Troubleshooting with Query Store

Lesson 1: SQL Server Query Execution

What is an Execution Plan?





SQL

Sets

empid	lastname	firstna...	title	titleofcourt...	birthdate
1	Davis	Sara	CEO	Ms.	1958-12-08 00:00:00.000
2	Funk	Don	Vice President, Sales	Dr.	1962-02-19 00:00:00.000
3	Lew	Judy	Sales Manager	Ms.	1973-08-30 00:00:00.000
4	Peled	Yael	Sales Representative	Mrs.	1947-09-19 00:00:00.000
5	Buck	Sven	Sales Manager	Mr.	1965-03-04 00:00:00.000

How to see the query plan

Graphical execution plan

Estimated Execution Plan (Before Execution)

- The compiled plan.

Actual Execution Plan (After Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.

Live Query Statistics (During Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information during execution progress and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.
- Enables rapid identification of potential bottlenecks.

SQL Server Execution Plan

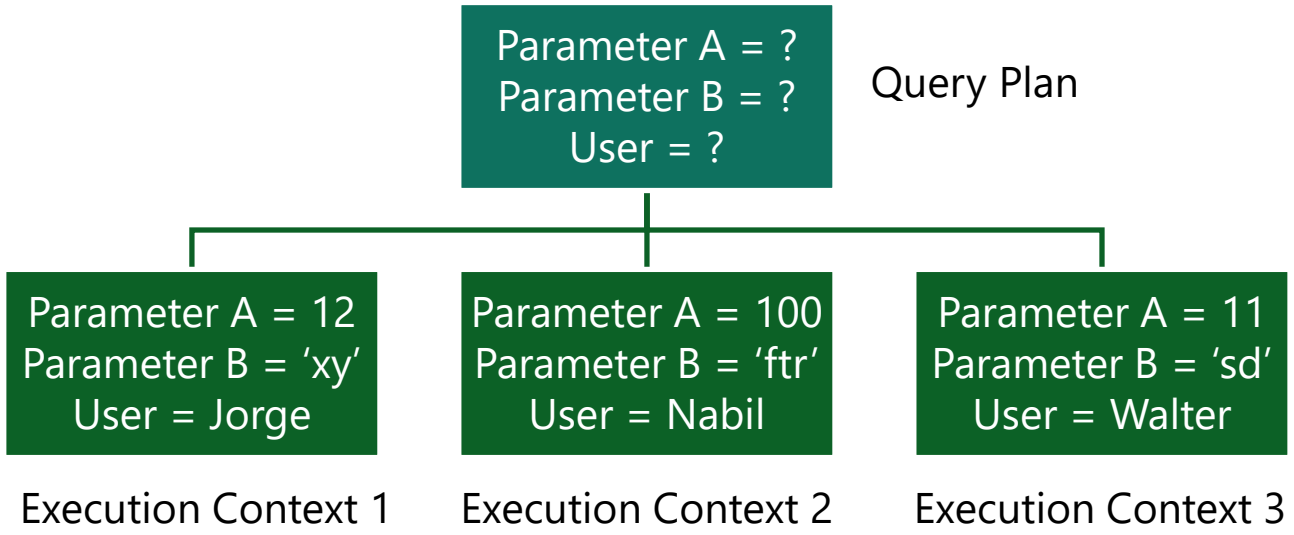
Main components

Compiled Plan (or Query Plan)

Compilation produces a query plan, which is a read-only data structure used by any number of users.

Execution Context

A data structure used to hold information specific to a query execution, such as parameter values.



SQL Server Execution Plan Caching

Overview

Part of the memory pool used to store execution plans – also known as plan cache.

The plan cache has two stores for all compiled plans:

The **Object Plans** cache store (OBJCP)

used for plans related to persisted objects (stored procedures, functions, and triggers).

The **SQL Plans** cache store (SQLCP)

used for plans related to autoparameterized, dynamic, or prepared queries.

SQL Server compilation and execution

Concepts

Compilation

Process of creating a good enough query execution plan, as quickly as possible for a query batch.

Refer to both the compilation of non-DML constructs in SQL statements (control flow, DDL, etc.) and the process of Query Optimization.

Query Execution

Process of executing the plan that is created during query compilation and optimization.

SQL Server Execution Plan Recompilations

Overview

Most recompilations are required either for statement correctness or to obtain potentially faster query execution plan.

The engine detects changes that invalidate execution plan(s) and marks those as not valid. New plan must be recompiled for the next query execution.

Starting with SQL Server 2005, whenever a statement within a batch causes recompilation, only the statement inside the batch that triggers recompilation is recompiled.

SQL Server Execution Plan Recompilations

Recompilation reasons

Table / Index Changes

- Changes made to objects referenced by the query (ALTER TABLE and ALTER VIEW).
- Changing or dropping any indexes used by the execution plan.

Stored Procedures

- Changes made to a single procedure, which would drop all plans for that procedure from the cache (ALTER PROCEDURE).
- Explicit call to sp_recompile.
- Executing a stored procedure using the WITH RECOMPILE option.

Data Volume

- Updates on statistics used by the execution plan
- For tables with triggers, if the number of rows in the inserted or deleted tables grows significantly.

Other

- Large numbers of changes to keys (generated by statements from other users that modify a table referenced by the query).
- Temporary table changes

Lesson 2: SQL Server Query Plan Analysis

How to see the query plan

Graphical execution plan

Estimated Execution Plan (Before Execution)

- The compiled plan.

Actual Execution Plan (After Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.

Live Query Statistics (During Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information during execution progress and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.
- Enables rapid identification of potential bottlenecks.

Contents of an Execution Plan

- Sequence in which the source tables are accessed.
- Methods used to extract data from each table.
- How data is joined
- Use of temporary worktables and sorts
- Estimated rowcount, iterations, and costs from each operator
- Actual rowcount and iterations

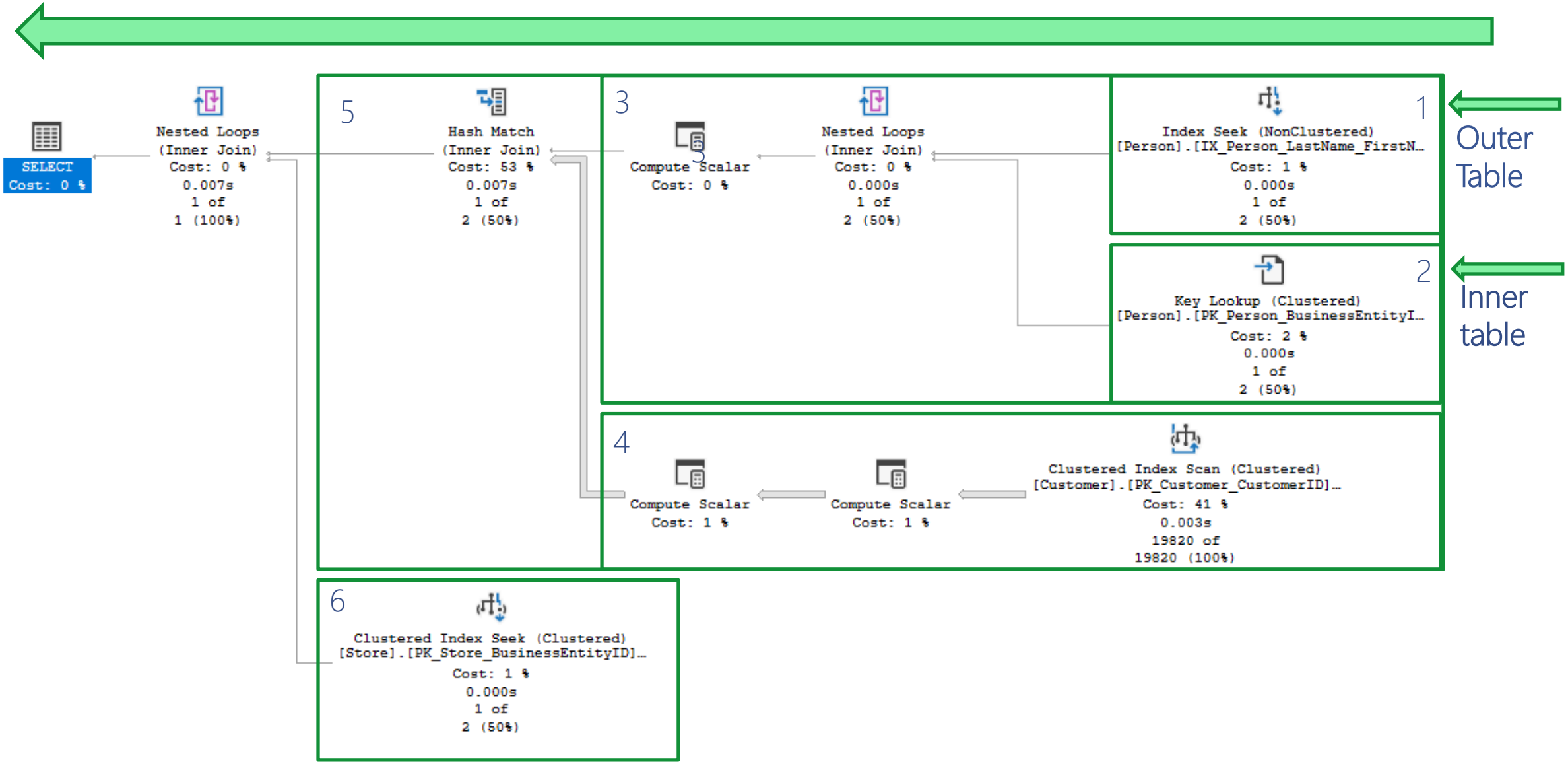
How to see the query plan

Text and XML

	Command	Execute query?	Include estimated row counts & stats (Estimated Query Plan)	Include actual row counts & stats (Actual Query Plan)
Text Plan	SET SHOWPLAN_TEXT ON	No	No	No
	SET SHOWPLAN_ALL ON	No	Yes	No
	SET STATISTICS PROFILE ON	Yes	Yes	Yes
XML Plan	SET SHOWPLAN_XML ON	No	Yes	No
	SET STATISTICS PROFILE XML	Yes	Yes	Yes

SSMS Graphical Plan

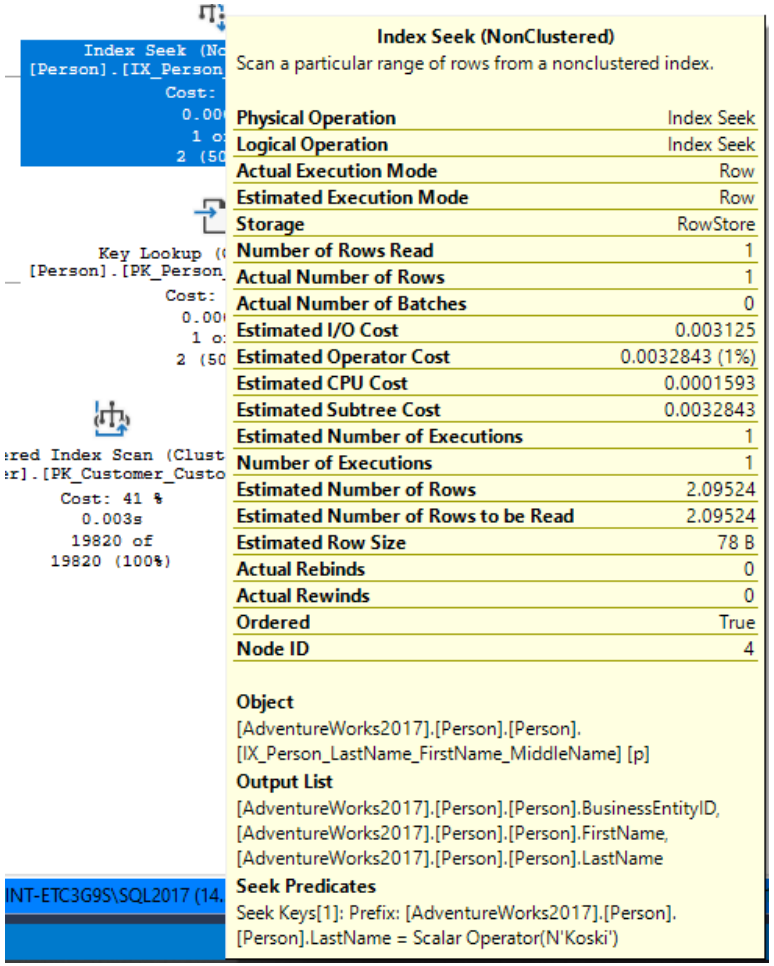
Execution Flow



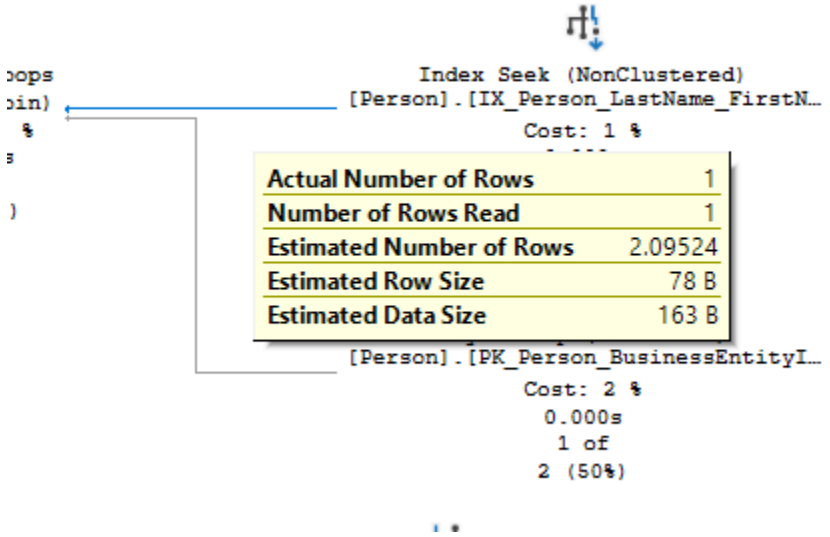
SSMS Graphical Plan

Data flow between the operators and statistical data of each operator

Statistical data for the operator

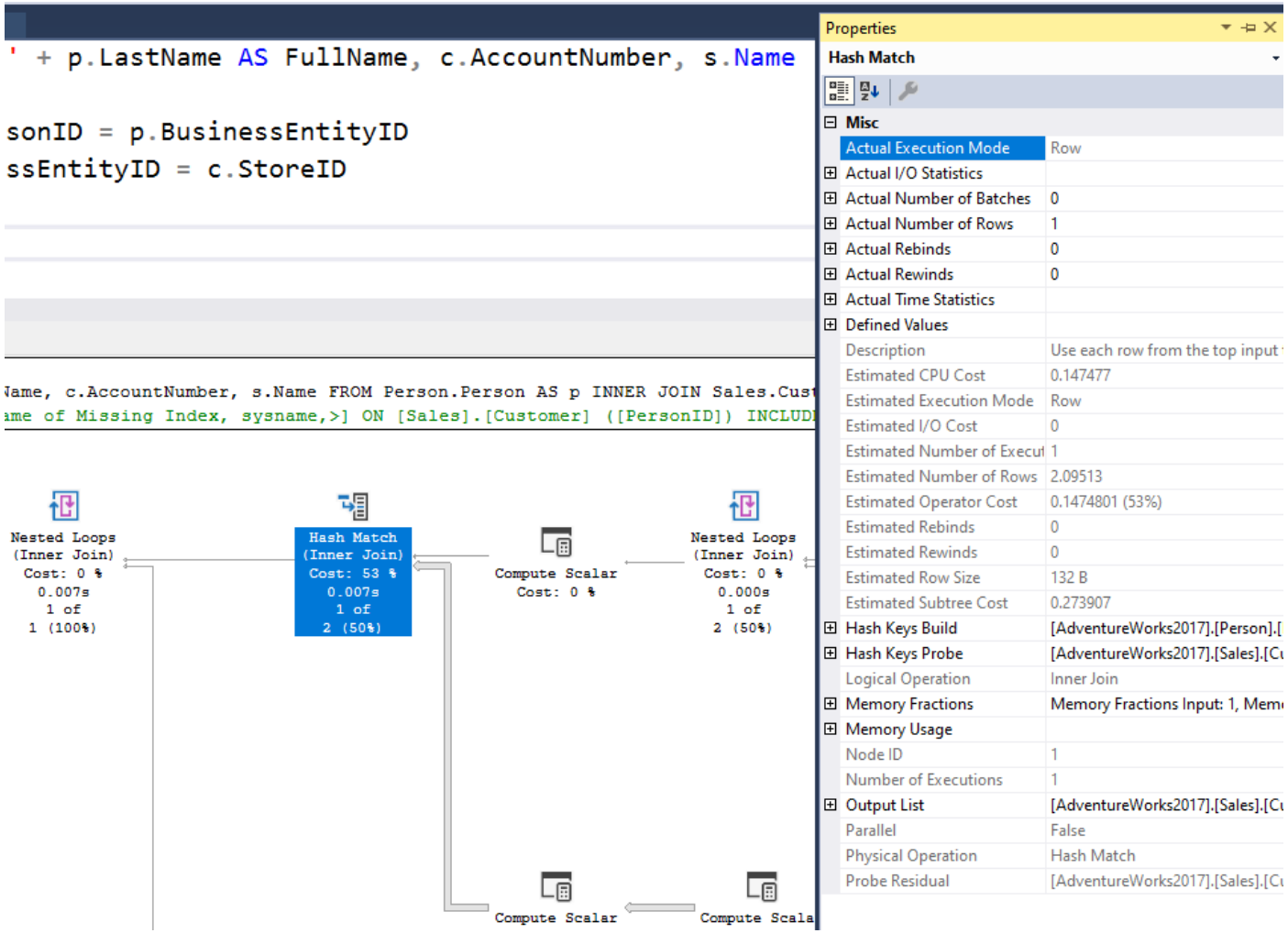


Data flow statistics



SSMS Graphical Plan

Properties sheet



Management Studio Properties sheet includes even more detailed information about each operator and about the overall query plan.

Use the most recent version of Management Studio as every new version display more detailed information about the Query Plan when examining the plan in graphical mode.

SSMS Graphical Plan

Live Query Statistics

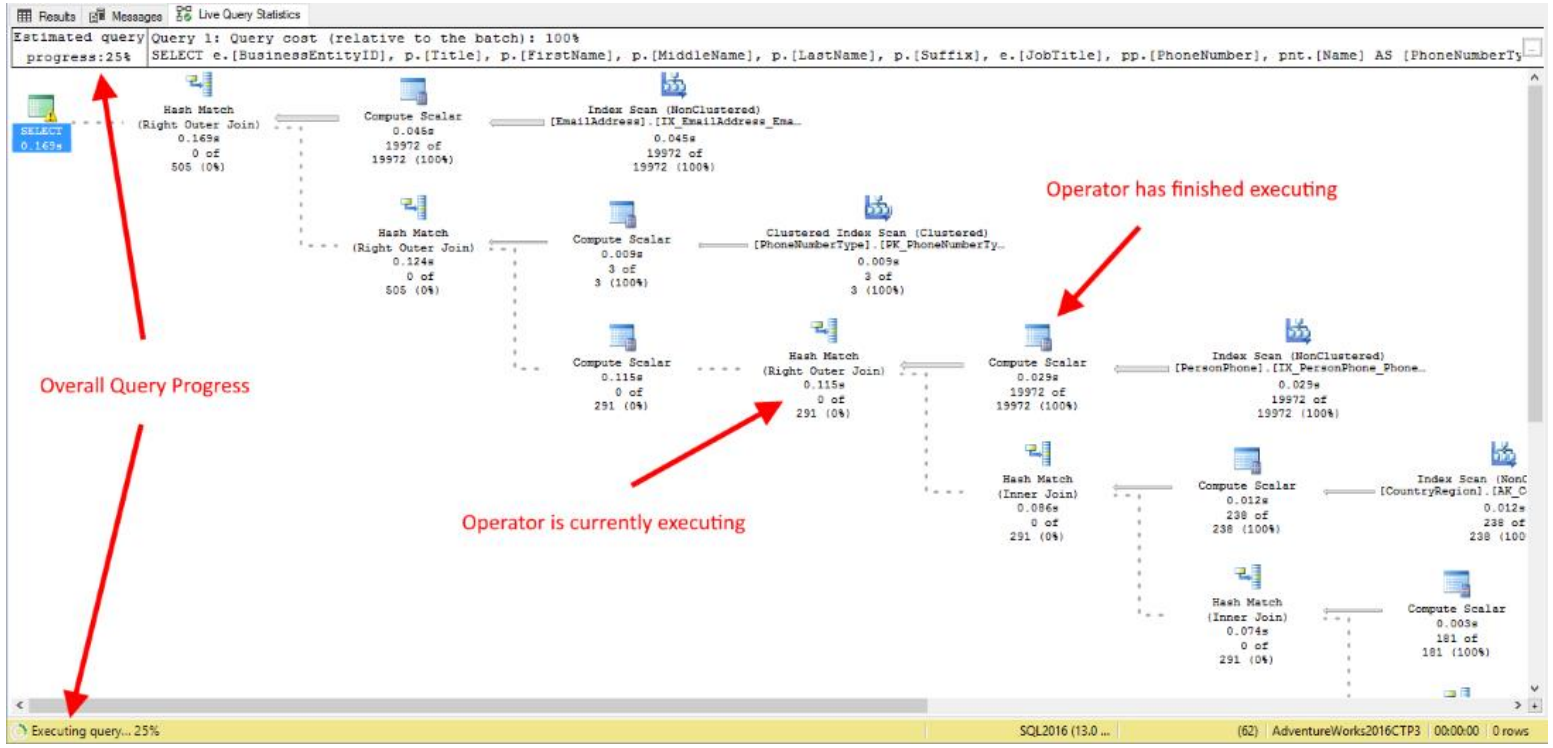
View CPU/memory usage, execution time, and query progress

Allows drill down to live operator-level statistics, such as:

- Number of generated rows
- Elapsed time
- Operator progress
- Live warnings

This feature is primarily intended for troubleshooting purposes.

Using this feature can moderately slow the overall query performance.



Execution Plan

Notable operators

Operators describe how SQL Server executes a query. The query optimizer uses operators to build a query plan to create the result specified in the query.


Table scan


Clustered index scan


Clustered index seek


RID lookup


Key lookup


ColumnStore Index Scan

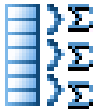

Nonclustered index scan


Nonclustered index seek


Sort


Table spool


Index spool


Stream Aggregation

Execution Plan Table Operators

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.

Table Scan
[BankAccounts]
Cost: 100 %

Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.

Clustered Index Scan (Cluste…
[BankAccounts].[pk_acctID]
Cost: 100 %

Using a WHERE statement on an Index could possibly have the Execution Plan seek the Index instead of scan.

Clustered Index Seek (Cluste…
[BankAccounts].[pk_acctID]
Cost: 100 %

Execution Plan Join Operators (Code)

```
SELECT SOH.SalesOrderID, SOH.CustomerID,
    OrderQty, UnitPrice, P.Name
FROM SalesLT.SalesOrderHeader as SOH
JOIN SalesLT.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN SalesLT.Product AS P
ON P.ProductID = SOD.ProductID
```

Execution Plan Join Operators

A Merge Join is useful if both table inputs are in the same sorted order on the same value.

Merge Join
(Inner Join)
Cost: 39 %

A Hash Match is used when the tables being joined are not in the same sorted order.

Hash Match
(Inner Join)
Cost: 47 %

A Nested Loop is use when a small (outer) table is used to lookup a value in a larger (inner) table.

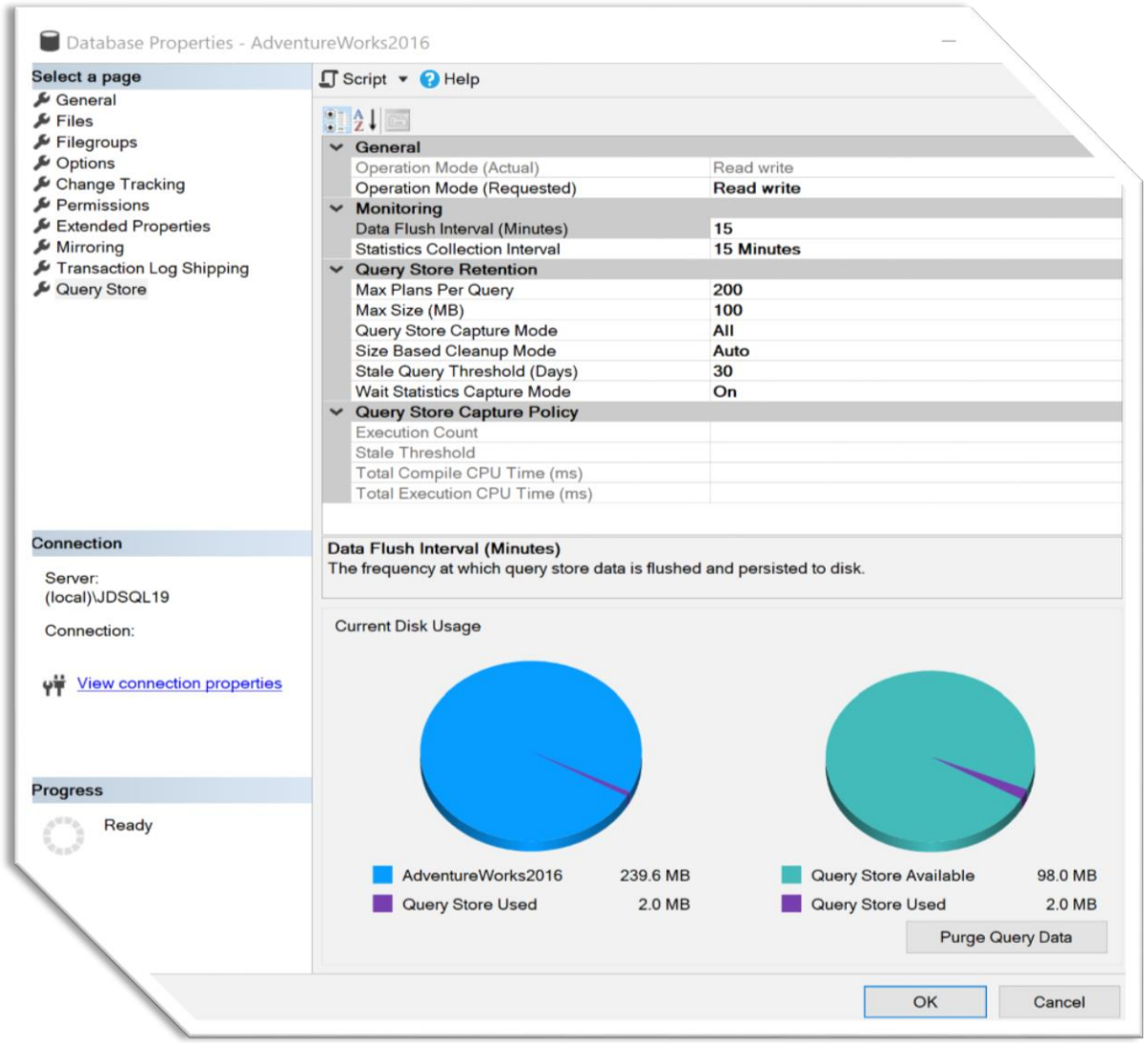
Nested Loops
(Inner Join)
Cost: 3 %

What to look for in the query plan

Warnings	• Information about possible issues with the plan
Top Left Operator	• Overall properties of the plan
Expensive Operators	• Look from most expensive to least expensive
Sort Operators	• Locate why there is a sort operation and is it needed
Data Flow Statistics	• Thicker arrows mean more data is being passed
Nested Loop Operator	• Possible to create index that covers query
Scans vs Seeks	• Not necessarily bad, but could indicate I/O issues
Skewed Estimates	• Statistics could be stale or invalid

Lesson 3: SQL Server Query Store

Introducing the Query Store



Query Store is set at the database level

Cannot be used for Master or TempDB system databases but can be enabled for the Model and MSDB system databases.

The user database stores the data in internal tables that can be accessed by using built-in Query Store views.

SQL Server retains this data until the space allocated to Query Store is full or manually purged.

Why use Query Store?

Before Query Store

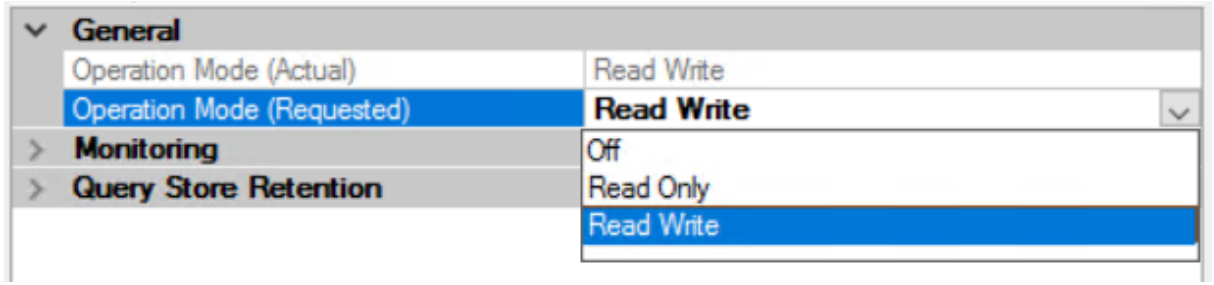
- Requires manual proactive monitoring to identify execution plan problems.
- Only the latest plan was stored in the procedure cache
- Restart caused data to be lost
- Frequent recompiles of procedures or use of DBCC FREEPROCACHE
- No history or aggregated gathering of data available.

With Query Store

- It stores the history of the execution plans for each query
- It establishes a performance baseline for each plan over time
- It identifies queries that may have regressed
- It is possible to force plans quickly and easily
- It works across server restarts, upgrades, and query recompilation

Query Store Operation Modes

Operation Mode can be set under database properties



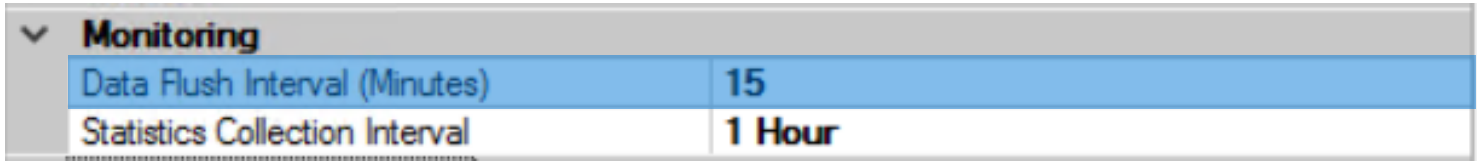
Operation Mode can be enabled two ways using T-SQL. If only using the ON option, the Mode defaults to **Read_Write**

```
ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE = ON;

ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE
(OPERATION_MODE = READ_WRITE);
```

Query Store Monitoring Settings

Data Flush Interval determines the frequency at which data written to the query store is persisted to disk. (Default is **15 Minutes**).




```
ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 1,
DATA_FLUSH_INTERVAL_SECONDS = 60)
```

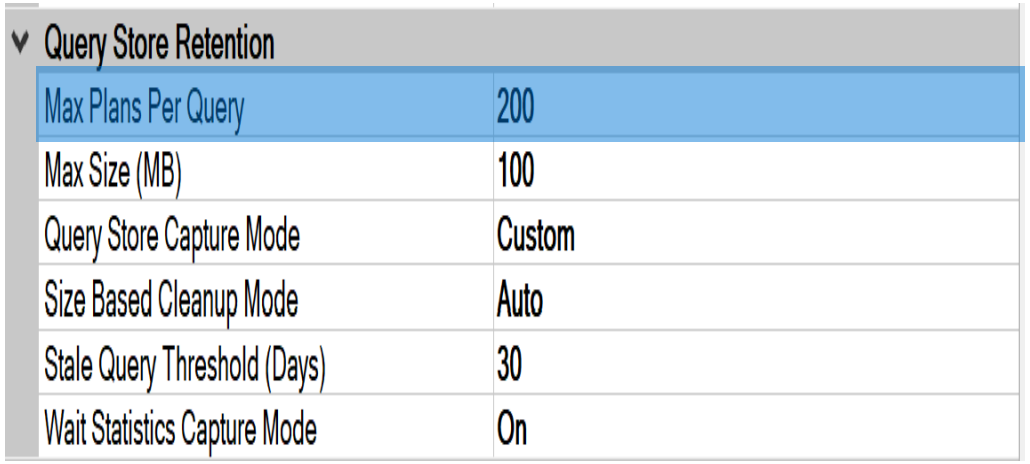
Query Store Monitoring Settings

Statistics Collection Interval determines the time interval at which runtime execution statistics data is aggregated into the query store. Only the values of 1, 5, 10, 15, 60, and 1440 minutes is allowed. (Default is **60**).


```
ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 1,
DATA_FLUSH_INTERVAL_SECONDS = 60)
```

Query Store Retention Settings

Max Plans Per Query is a new retention setting introduced in SQL Server 2017 and is an integer representing the maximum number of plans maintained for each query. (Default is **200**).

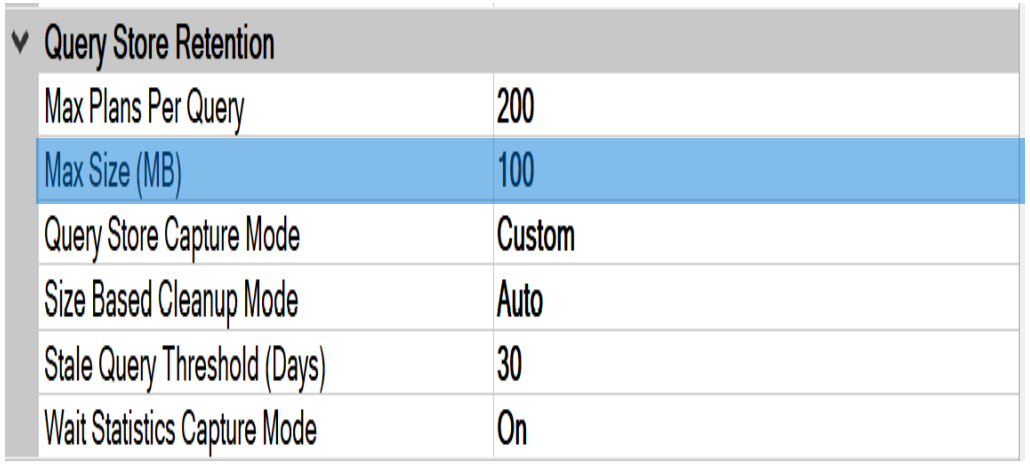


Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Retention Settings

Max Size (MB) configures the maximum storage size for the query store. (Default is **100MB**) When the query store limit is reached, query store changes the state from read-write to read-only.



Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Retention Settings

Query Store Capture Mode determines to capture all the queries (Default is **ALL**), or relevant queries based on execution count and resource consumption (**AUTO**) or stop capturing queries (**NONE**). SQL Server 2019 introduces an additional (**CUSTOM**) setting.

Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Retention Settings

Size Based Cleanup Mode determines whether the cleanup process will be automatically activated when the total amount of data gets close to the maximum size. (Default is **Auto**).

Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Retention Settings

Stale Query Threshold (Days) determines the number of days to retain data in the query store. (Default is **30 days** and Maximum is **367 days**).

Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Retention Settings

Wait Statistics Capture Mode is a new retention setting introduced in SQL Server 2017 that controls if Query Store captures wait statistics information.

(Default = **ON**).

Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 WAIT_STATS_CAPTURE_MODE = ON);
GO
```

Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **EXECUTION COUNT** is the value a query must exceed within the Stale Threshold time period to be captured by the Query Store.

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

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
GO
```

Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Stale Threshold** can be from 1 hour up to 7 days. This setting specifies the time given to exceed the values of the three other settings for a query to be captured.

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

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
GO
```

Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Total Compile CPU Time (ms)** is the value in milliseconds that a query must exceed within the **Stale Threshold** time period to be captured by the Query Store.

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

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
GO
```

Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Total Execution CPU Time (ms)** is the value in milliseconds that a query must exceed within the **Stale Threshold** time period to be captured by the Query Store.

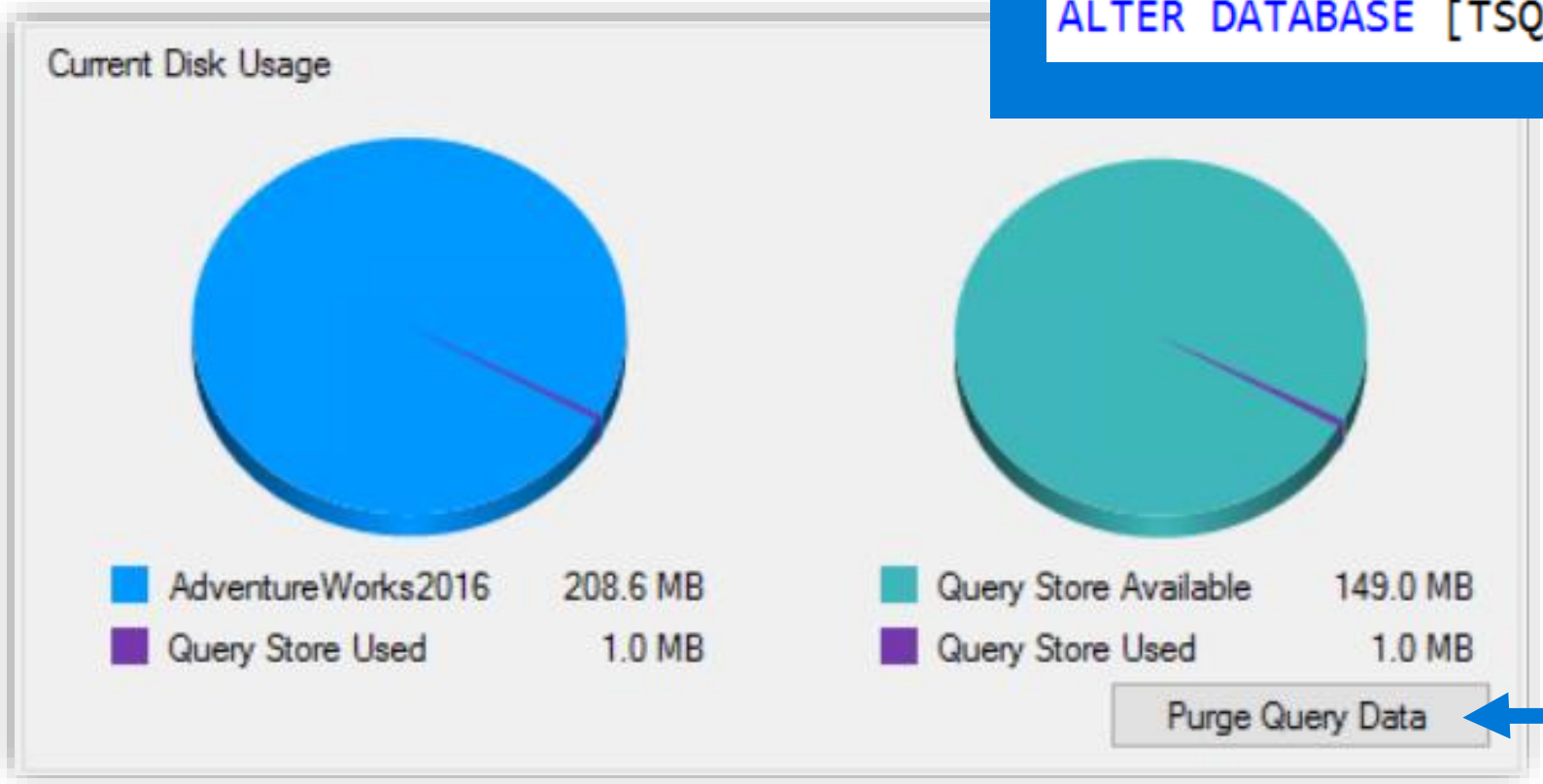
Query Store Capture Policy	
Execution Count	30
Stale Threshold	1 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

```
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
GO
```

Purge Query Data

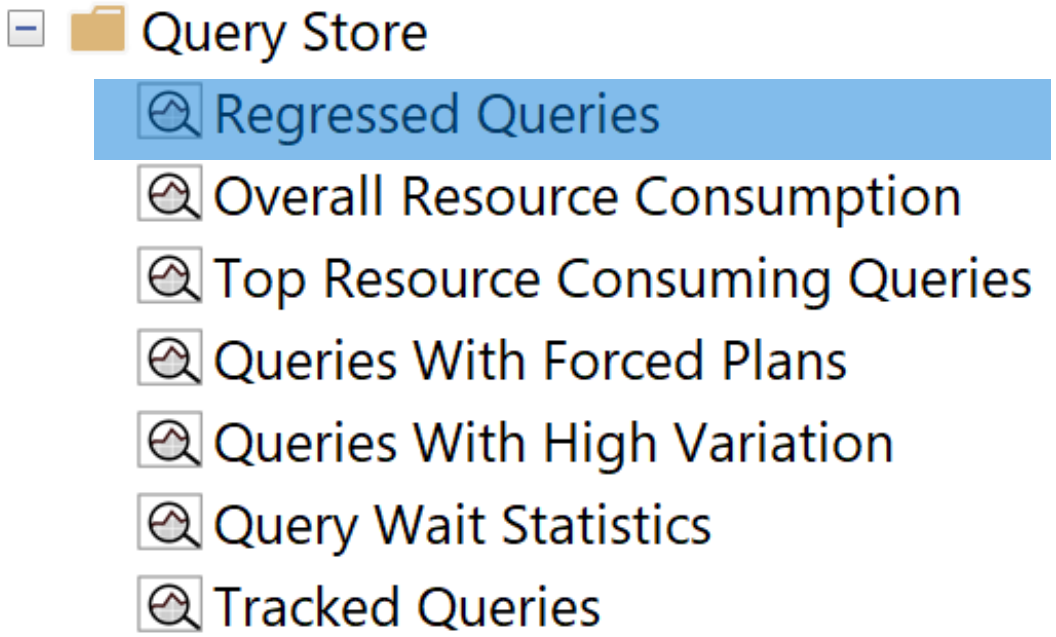
Data can be manually purged from the Query Store.

```
ALTER DATABASE [TSQL] SET QUERY_STORE CLEAR;
```



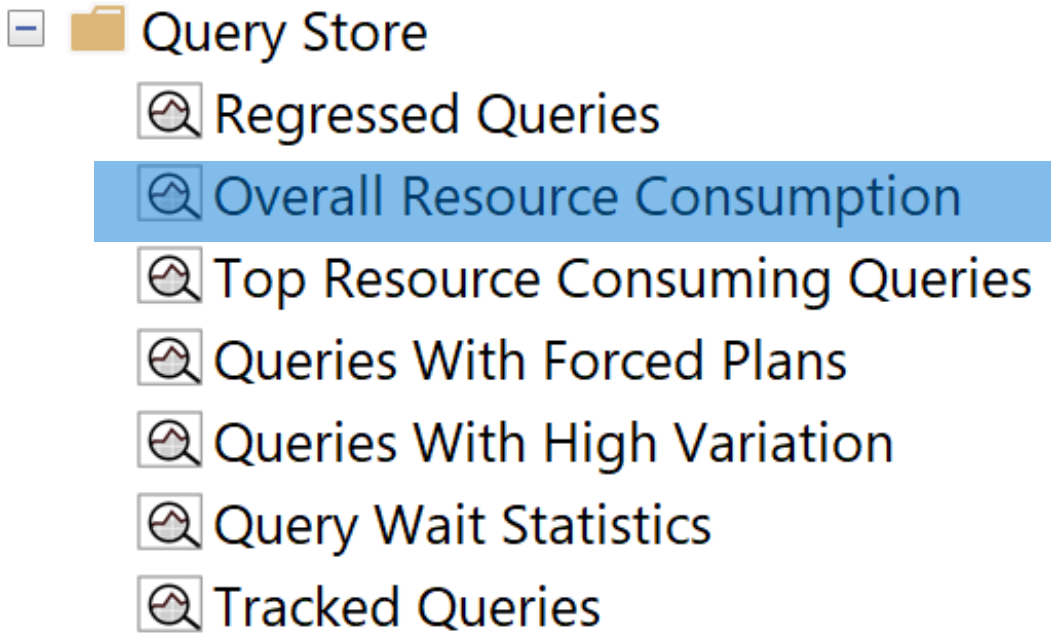
Query Store Reports

Regressed Queries: Use this dashboard to review queries that might have regressed because of execution plan changes



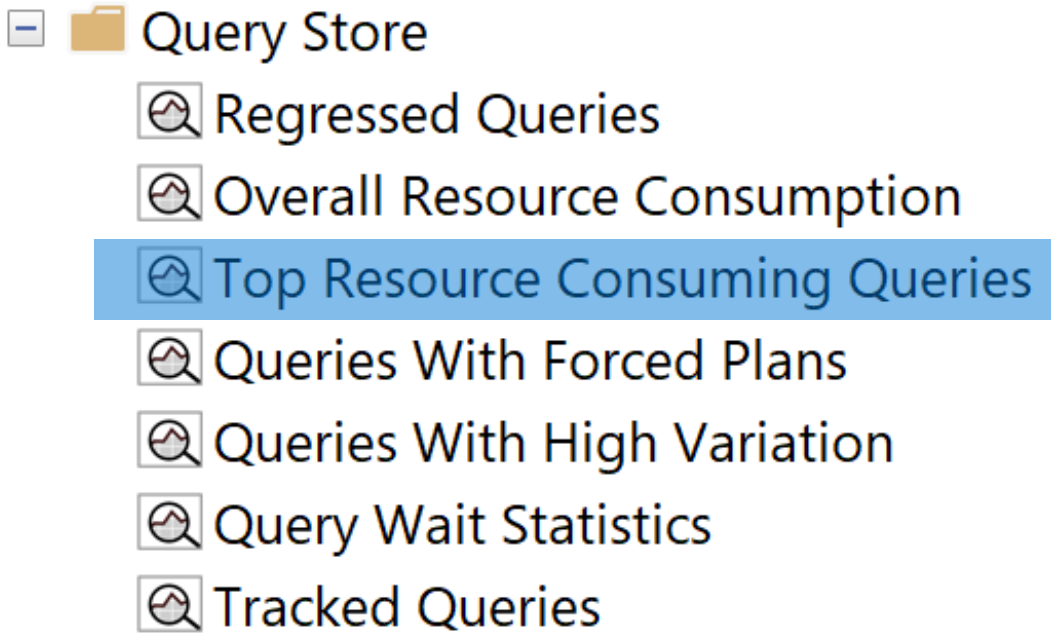
Query Store Reports

Overall Resource Consumption: Use this dashboard to visualize overall resource consumption during the last month in four charts: duration, execution count, CPU time, and logical reads



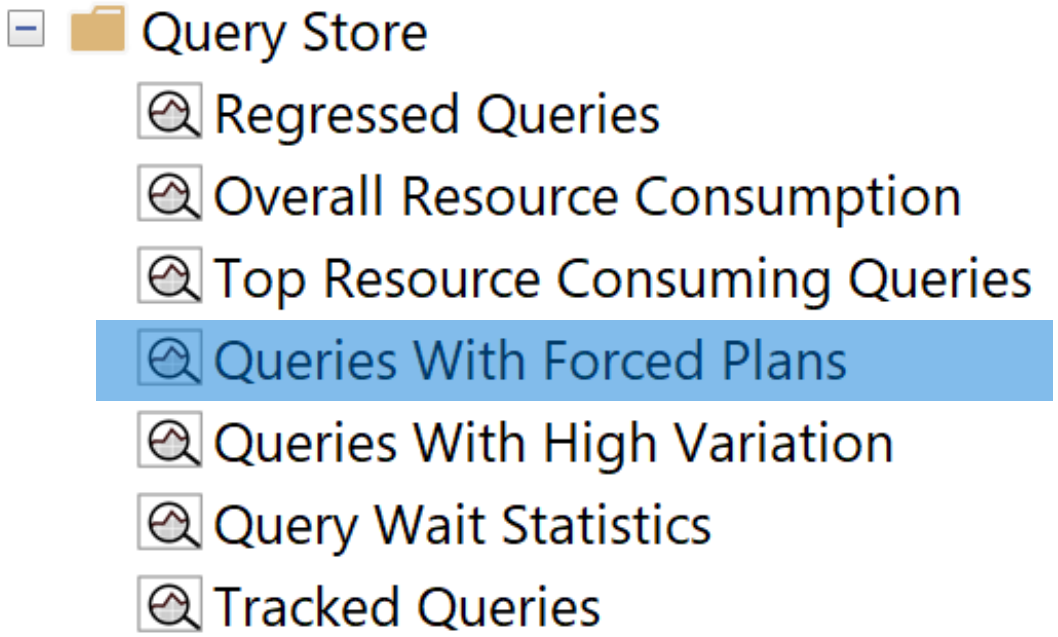
Query Store Reports

Top Resource Consuming Queries: Use this dashboard to review queries in the set of top 25 resource consumers during the last hour



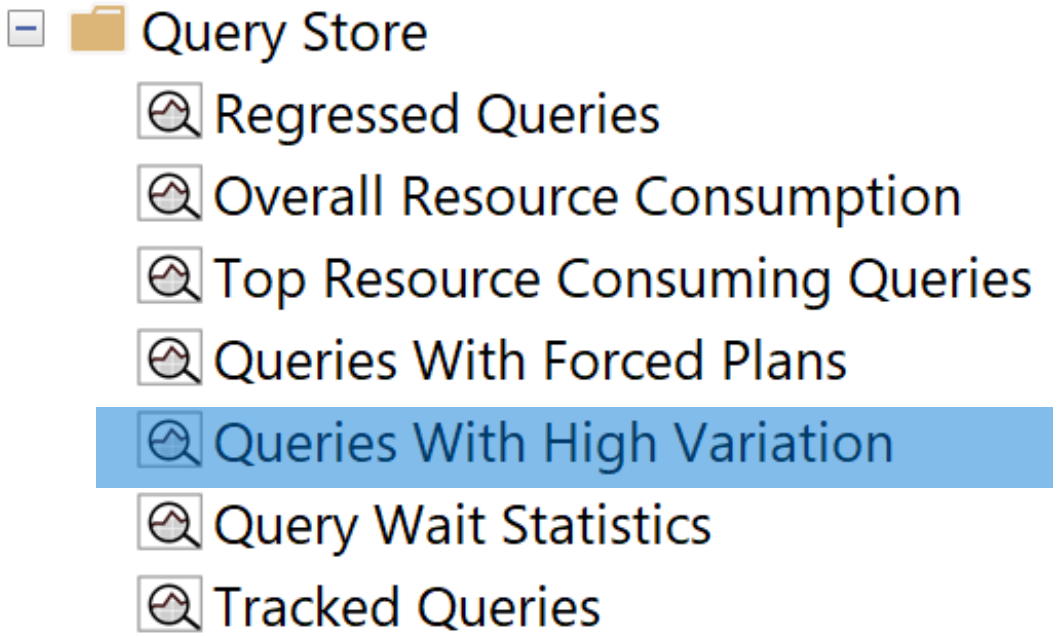
Query Store Reports

Queries With Forced Plans: Used to isolate queries that have been given a forced plan. Requires SQL Server 2016 SP1 or later.



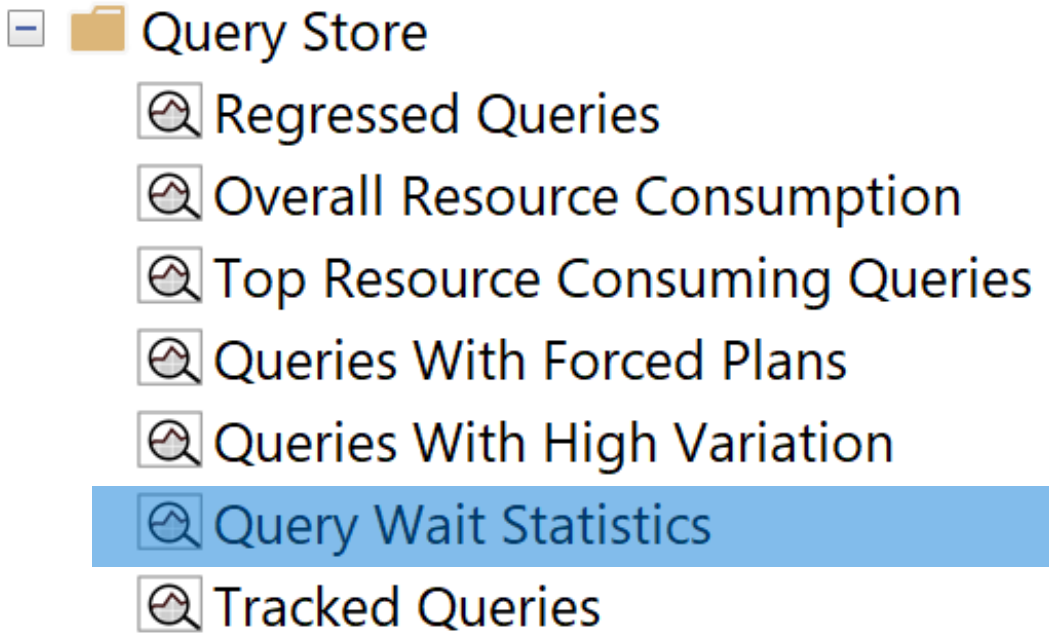
Query Store Reports

Queries With High Variation: Used to locate queries with high variation in query execution. Useful to locate queries with parameterization problems. Requires SQL Server 2016 SP1 or later.



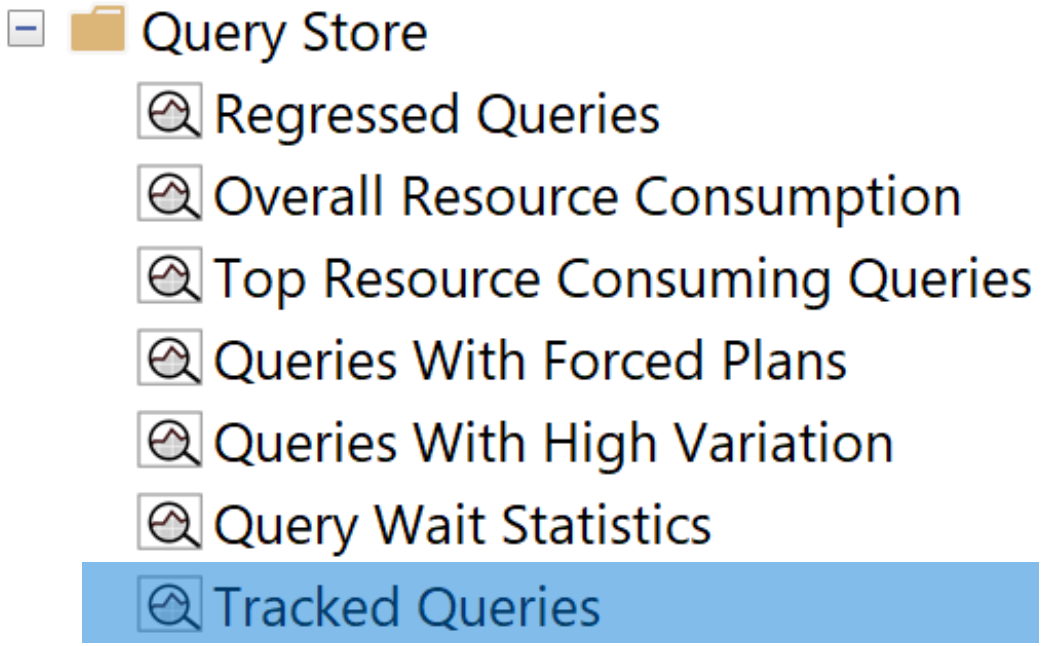
Query Store Reports

Query Wait Statistics shows a bar chart containing the top wait categories in the Query Store. Use the drop down at the top to select an aggregate criteria for the wait time: avg, max, min, std dev, and **total** (default). Requires SQL Server 2017.

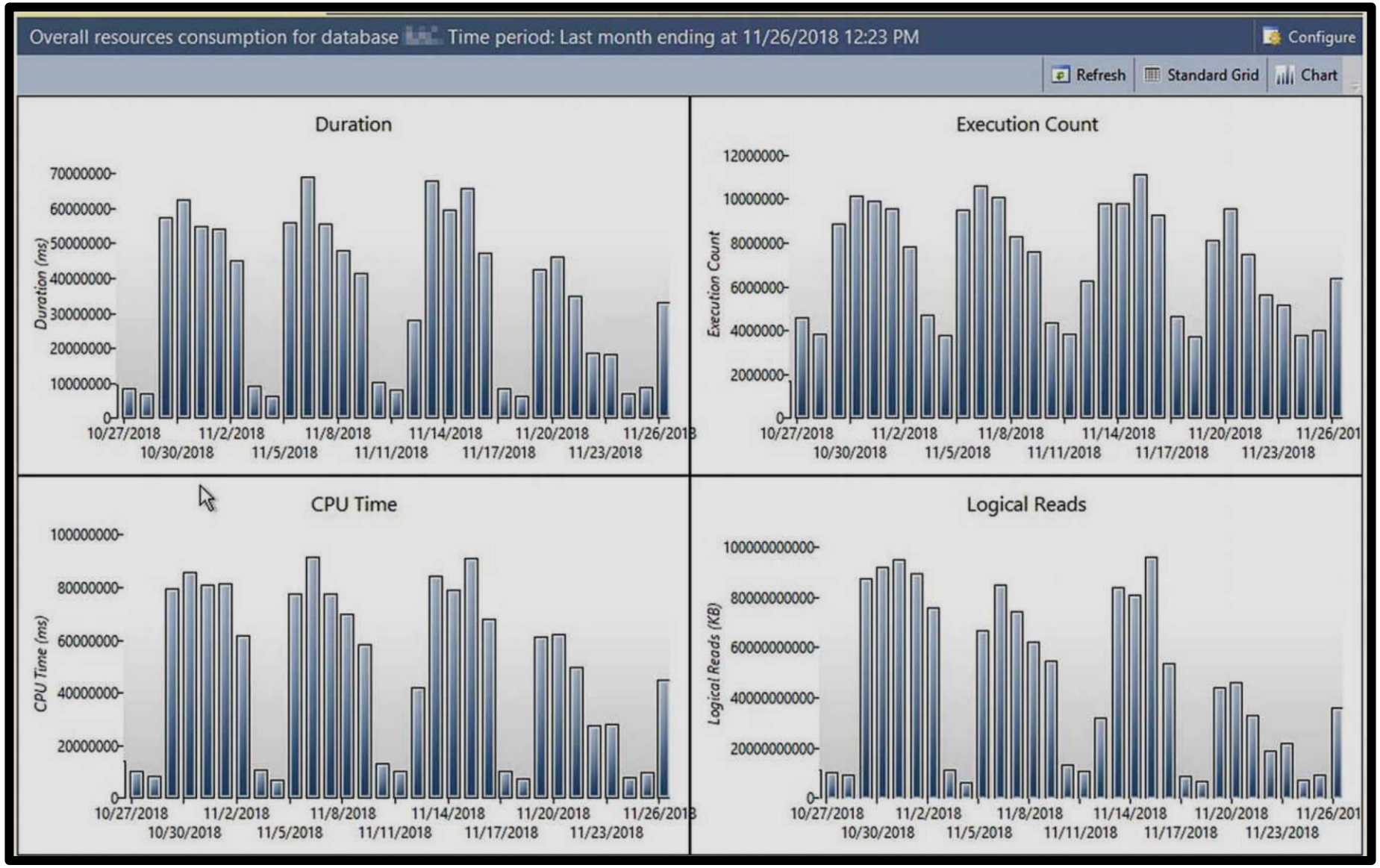


Query Store Reports

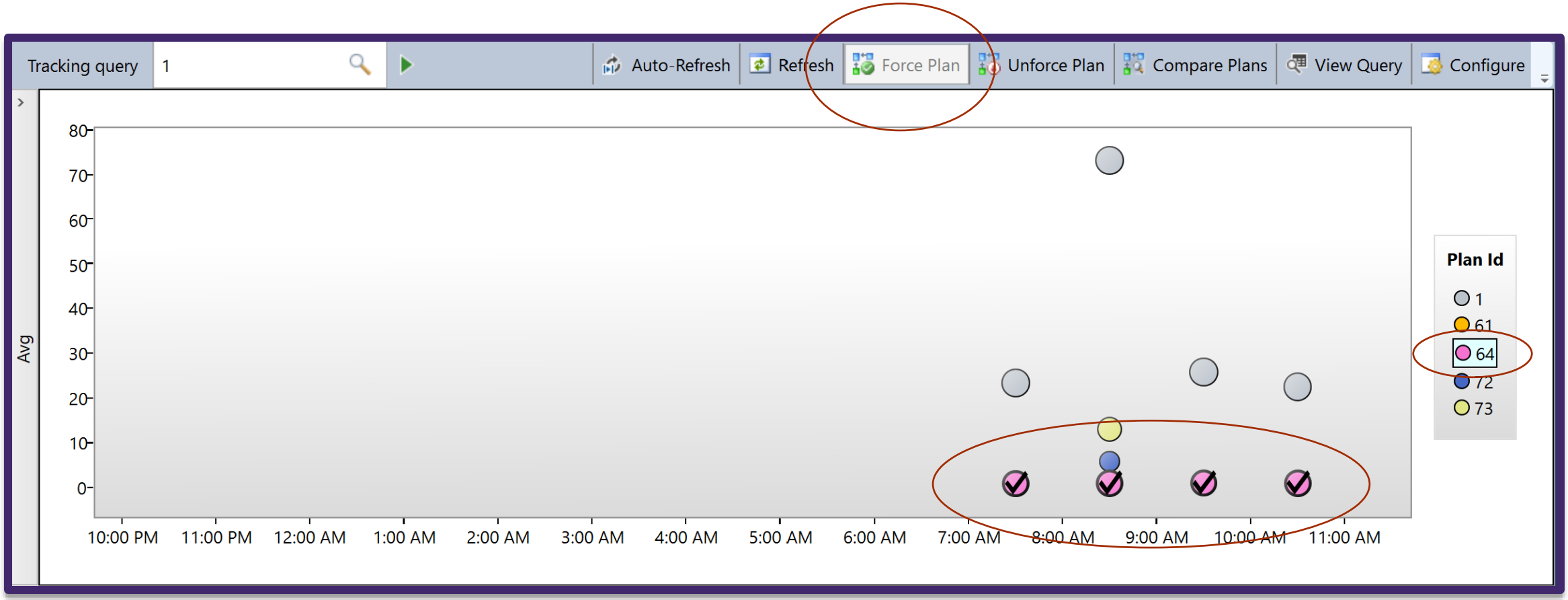
Tracked Queries: Use this dashboard to monitor the execution plans and regression of a specify query



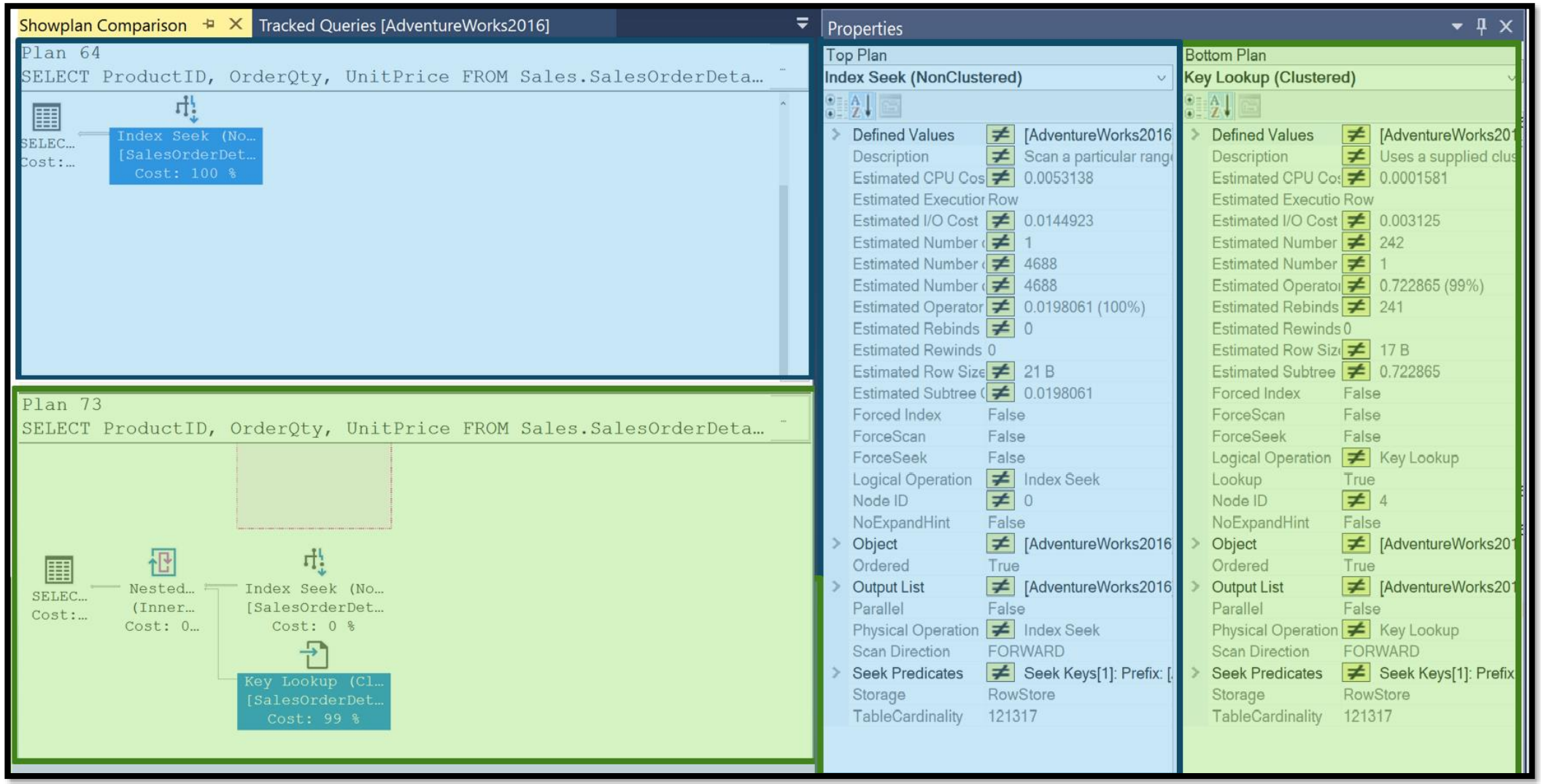
Establishing a Baseline



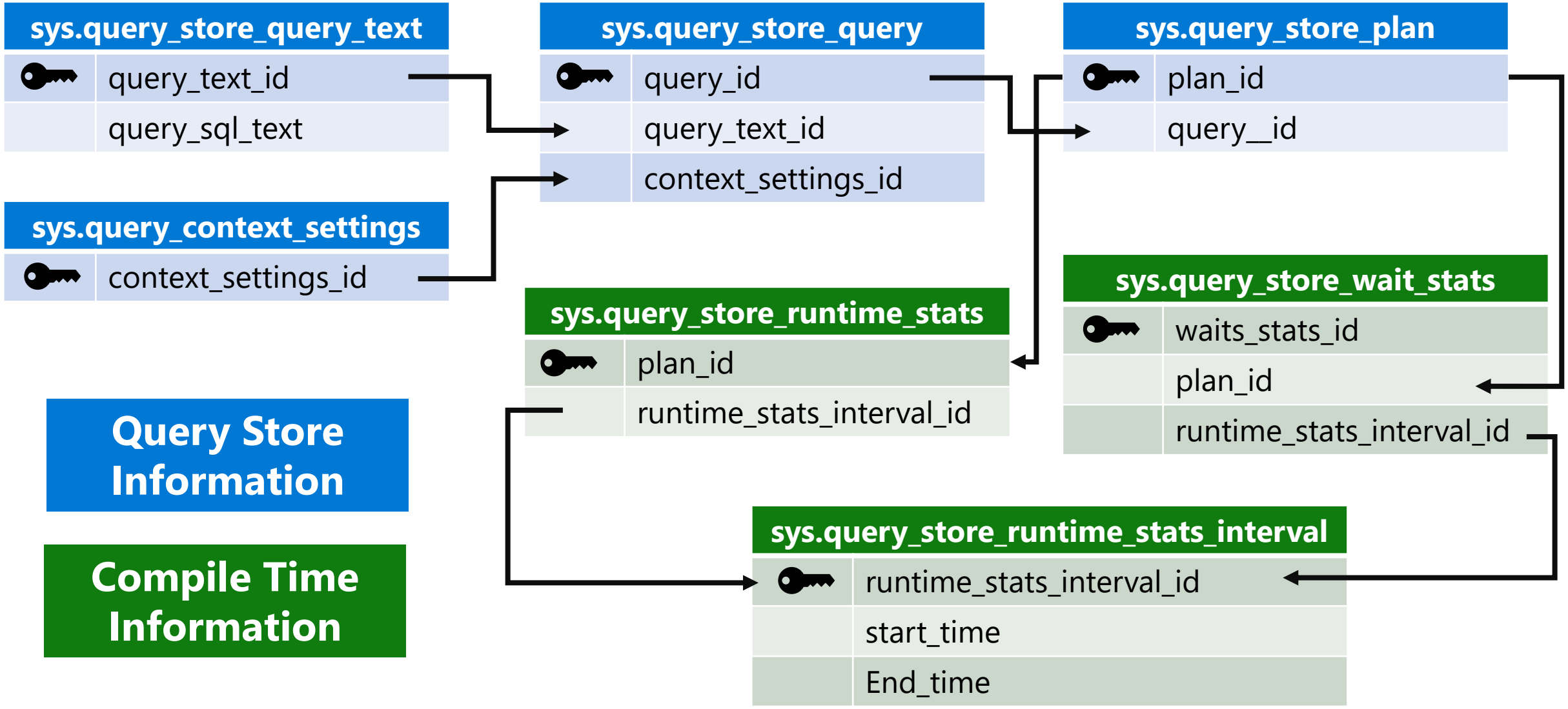
Force Plan


Tracking query
1
Auto-Refresh
Refresh
Force Plan
Unforce Plan
Compare Plans
View Query
Configure
Avg
80
70
60
50
40
30
20
10
0
10:00 PM
11:00 PM
12:00 AM
1:00 AM
2:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM
8:00 AM
9:00 AM
10:00 AM
11:00 AM
Plan Id
1
61
64
72
73

Plan Compare



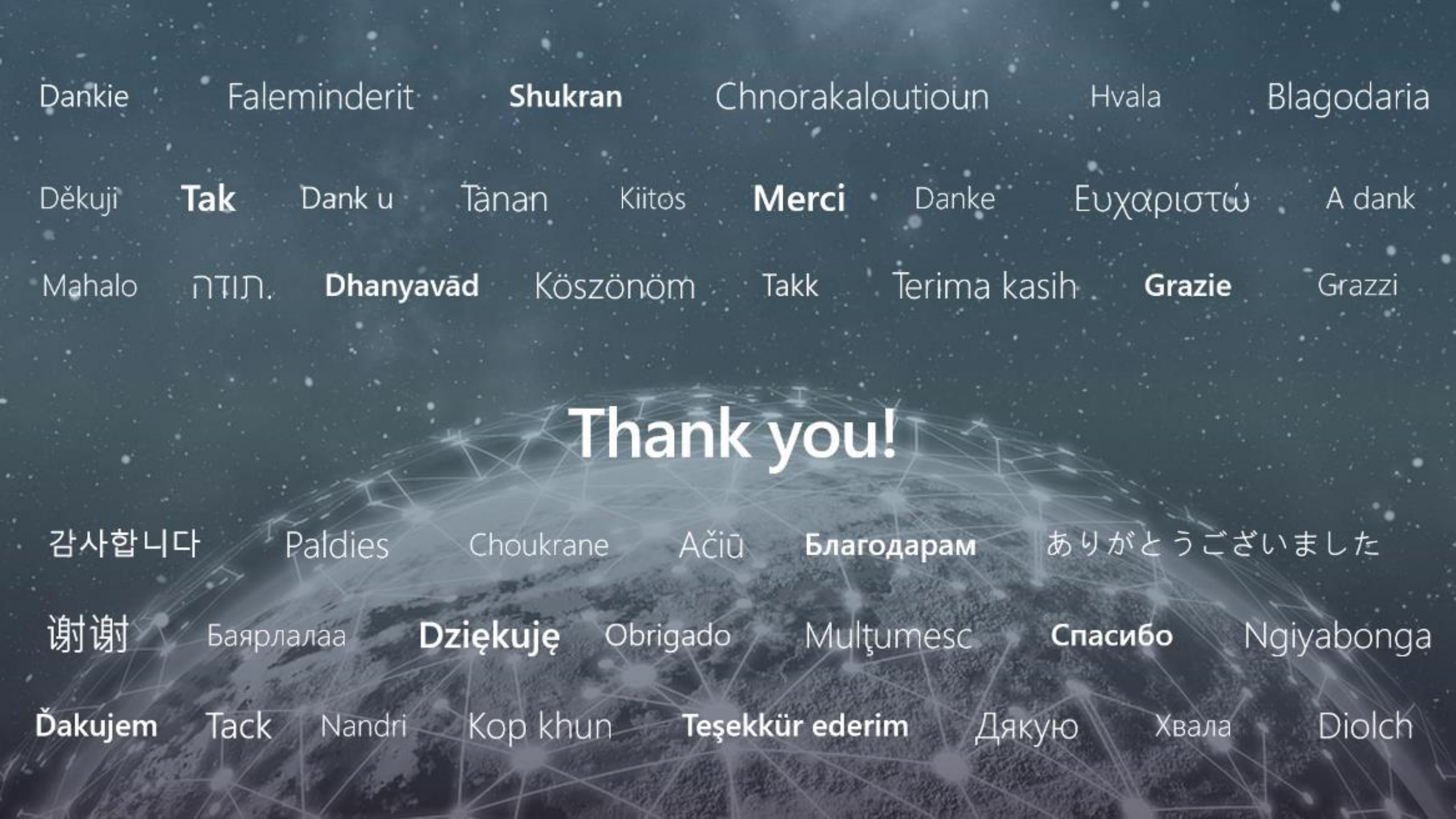
Query Store Catalog Views



Using Query Store Catalog Views

Finding the TOP 10 most frequently executed SQL Server Queries in the Query Store.

```
SELECT TOP 10 t.query_sql_text, q.query_id
FROM sys.query_store_query_text as t
JOIN sys.query_store_query as q
ON t.query_text_id = q.query_text_id
JOIN sys.query_store_plan as p
ON q.query_id = p.query_id
JOIN sys.query_store_runtime_stats as rs
ON p.plan_id = rs.plan_id
WHERE rs.count_executions >1
GROUP BY t.query_sql_text, q.query_id
ORDER BY SUM(rs.count_executions)
```


Dankie
Faleminderit
Shukran
Chnorakaloutioun
Hvala
Blagodaria
Děkuji
Tak
Dank u
Tänan
Kiitos
Merci
Danke
Ευχαριστώ
A dank
Mahalo
תודה.
Dhanyavād
Köszönöm
Takk
Terima kasih
Grazie
Grazzi
Thank you!
감사합니다
Paldies
Choukrane
Ačiū
Благодарам
ありがとうございました
谢谢
Баярлалаа
Dziękuję
Obrigado
Mulţumesc
Спасибо
Ngiyabonga
Ďakujem
Tack
Nandri
Kop khun
Teşekkür ederim
Дякую
Хвала
Diolch