Microsoft

SQL Server Query Execution and Plans

Module 7

Learning Units covered in this Module

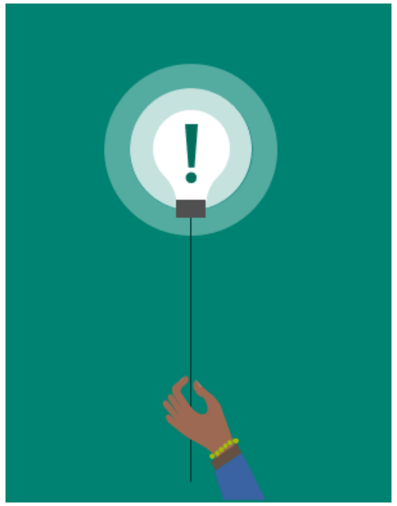
- Lesson 1: SQL Server Query Execution
- Lesson 2: SQL Server Query Plan Analysis
- Lesson 3: SQL Server Intelligent Query Processing

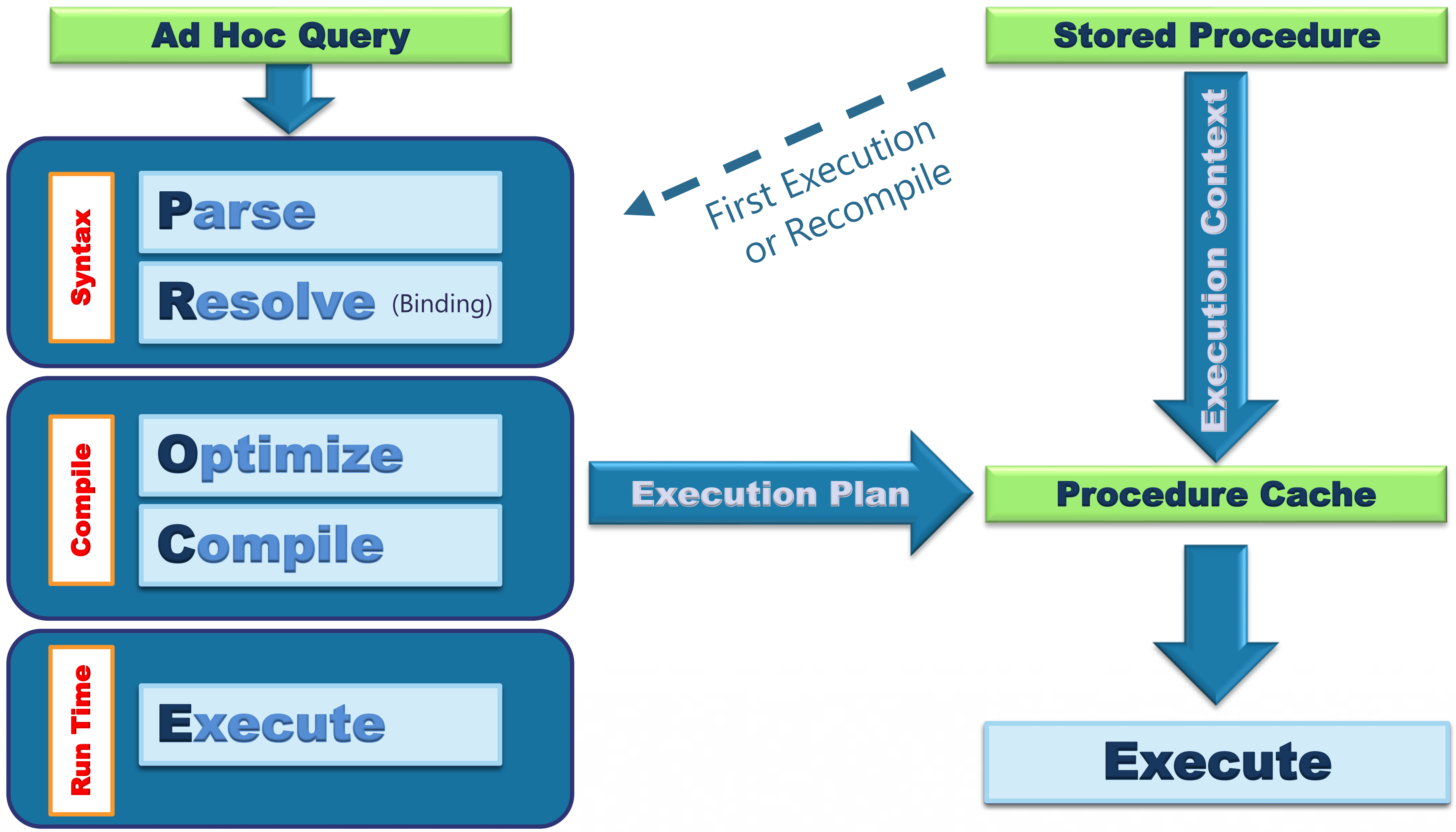
Lesson 1: SQL Server Query Execution

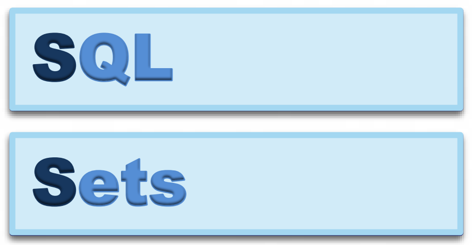
Objectives

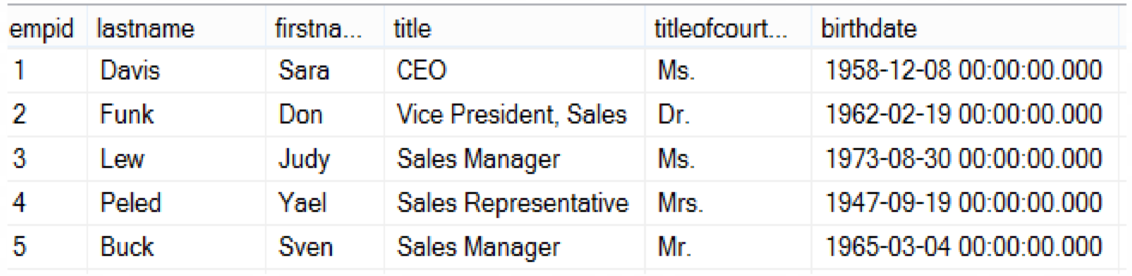
After completing this learning, you will be able to:

- Explain Query Compilation and Optimization Process.
- Explain Query Execution Process.
- Explain Recompilation causes.









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

What does the binding step resolve?

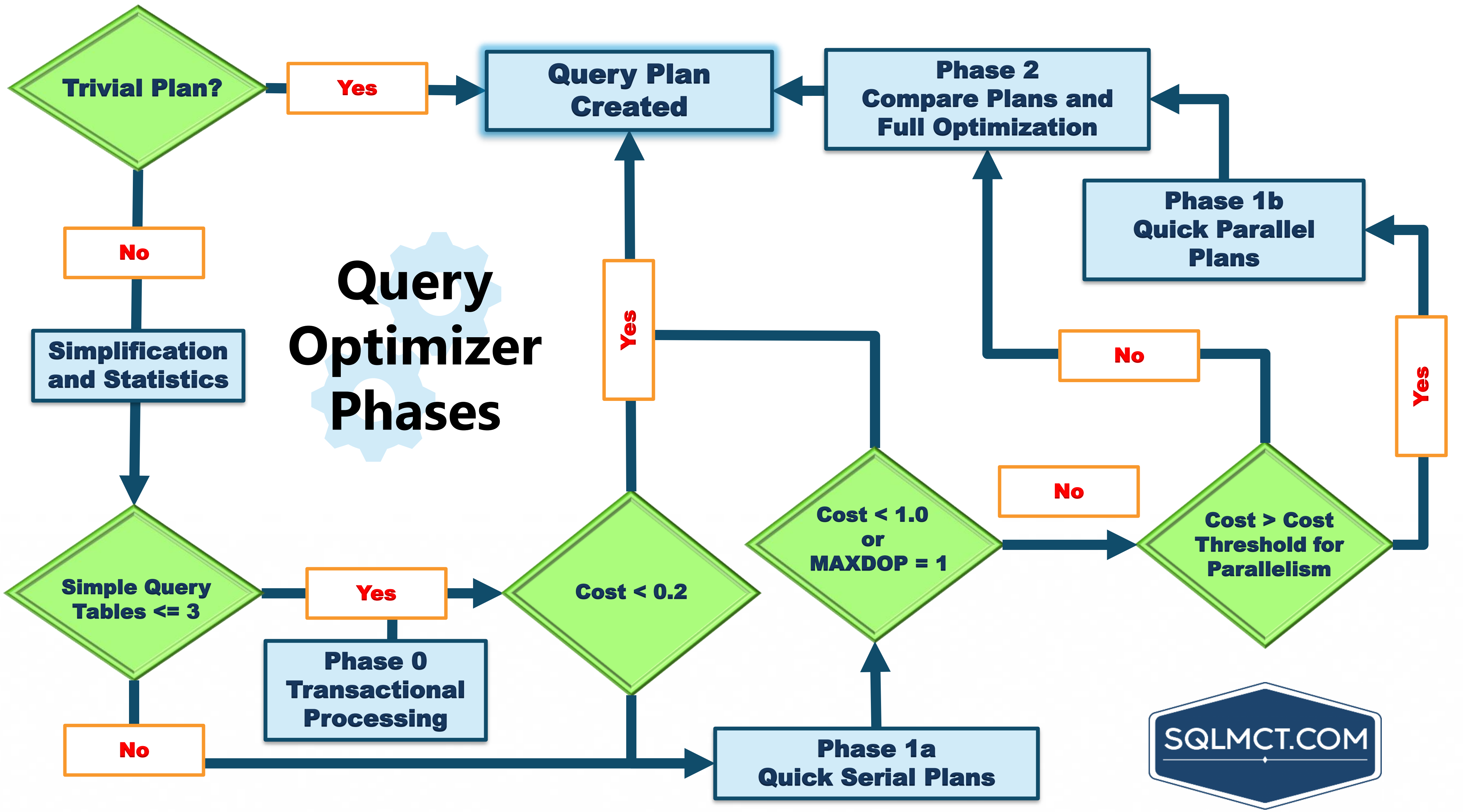
User permissions are checked.

Does a cached plan exist?

Object names (Tables, Views, Columns, etc.) to see if they exist.

Resolve aliases of columns and tables

Data types and if implicit data type conversions are needed.


Trivial Plan?
Yes
Query Plan Created
Phase 2 Compare Plans and Full Optimization
No
Query Optimizer Phases
Yes
Phase 1b Quick Parallel Plans
Simplification and Statistics
No
Yes
No
Cost < 1.0 or MAXDOP = 1
Cost > Cost Threshold for Parallelism
Simple Query Tables <= 3
Yes
Cost < 0.2
Phase 0 Transactional Processing
No
Phase 1a Quick Serial Plans
SQLMCT.COM

Query Simplification phases

Constant Folding: Expressions with constant values are reduced

- **Quantity = 2 + 3** becomes **Quantity = 5**
- **10 < 20** becomes **True**

Contradiction Detection: Removes criteria that doesn’t match table constraints

- **Constraint:** Age > 18
- **Contradiction:** WHERE Age < 18

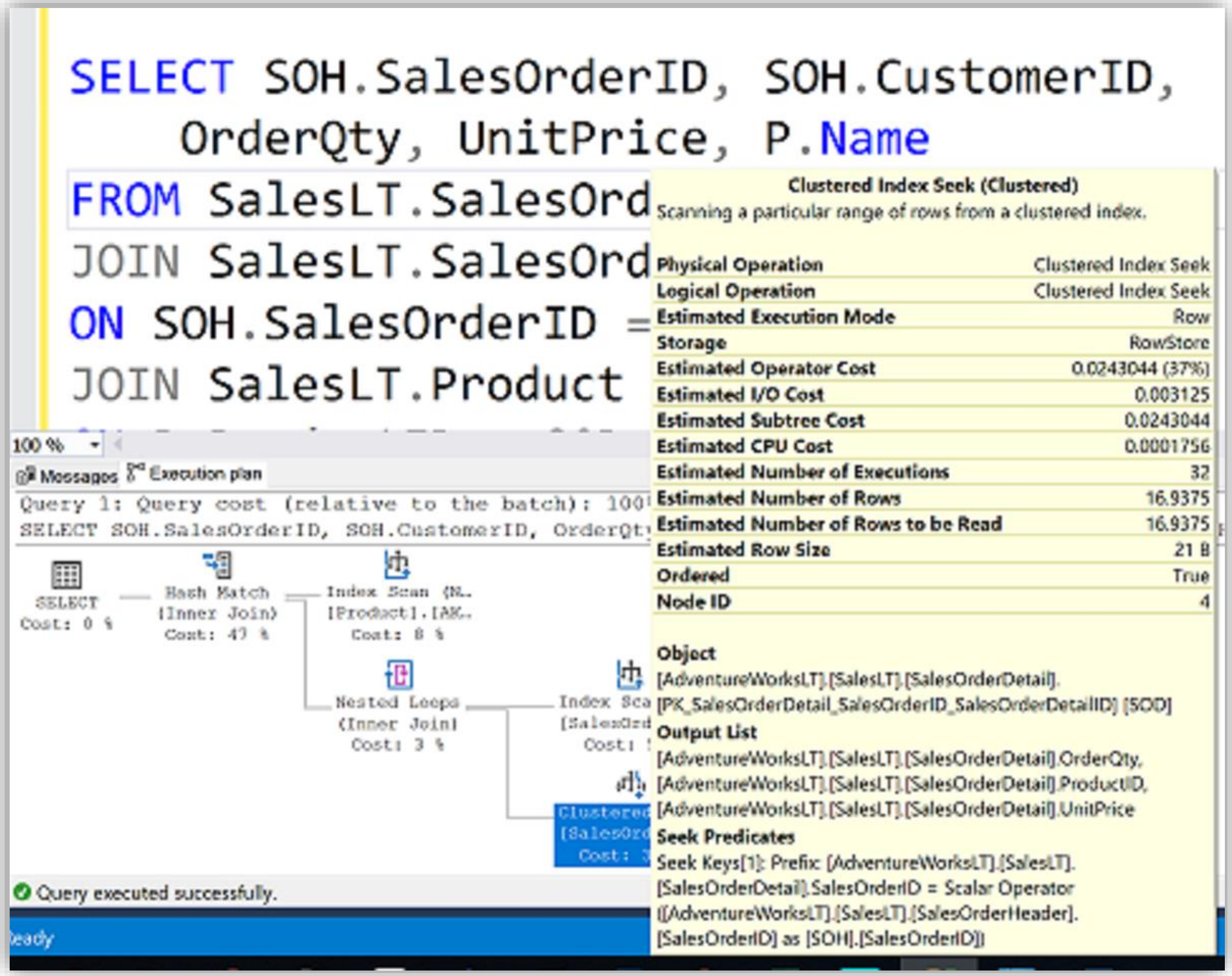
Domain Simplification: Reduces complex ranges to simple ranges

- **Complex range:** ID > 10 and ID < 20 or ID > 30 and < 50
- **Simplified range:** ID > 10 and < 50

Join Simplification: Removes redundant joins that are not necessary

Predicate Pushdown: Perform calculations only on rows returned

What is an Execution Plan?



```
SELECT SOH.SalesOrderID, SOH.CustomerID,
    OrderQty, UnitPrice, P.Name
FROM SalesLT.SalesOrd
JOIN SalesLT.SalesOrd
ON SOH.SalesOrderID =
JOIN SalesLT.Product
```

```
Query 1: Query cost (relative to the batch): 100
SELECT SOH.SalesOrderID, SOH.CustomerID, OrderQt
```

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0243044 (37%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0243044
Estimated CPU Cost	0.0001756
Estimated Number of Executions	32
Estimated Number of Rows	16.9375
Estimated Number of Rows to be Read	16.9375
Estimated Row Size	21 B
Ordered	True
Node ID	4

Object

[AdventureWorksLT].[SalesLT].[SalesOrderDetail].
[PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID] [SOD]

Output List

[AdventureWorksLT].[SalesLT].[SalesOrderDetail].OrderQty,
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].ProductID,
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].UnitPrice

Seek Predicates

Seek Keys[1]: Prefix: [AdventureWorksLT].[SalesLT].
[SalesOrderDetail].SalesOrderID = Scalar Operator
([AdventureWorksLT].[SalesLT].[SalesOrderHeader].
[SalesOrderID] as [SOH].[SalesOrderID])

Query executed successfully.

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

Execution modes

Row mode execution

- Efficient for OLTP scenarios
- Used with traditional tables, where data is stored in row-wise format.
- Operators read all columns from qualifying rows based on predicate, and for each row it retrieves columns needed for the result set

Batch mode execution

- Efficient for Data Warehousing scenarios
- Used to process multiple rows together
- Closely integrated with, and optimized around the ColumnStore storage format
- Operators read only the columns required for the result, from group of rows together.
- Starting SQL Server 2019, Batch mode does not require a ColumnStore index as in the previous versions.

SQL Server Execution Plan

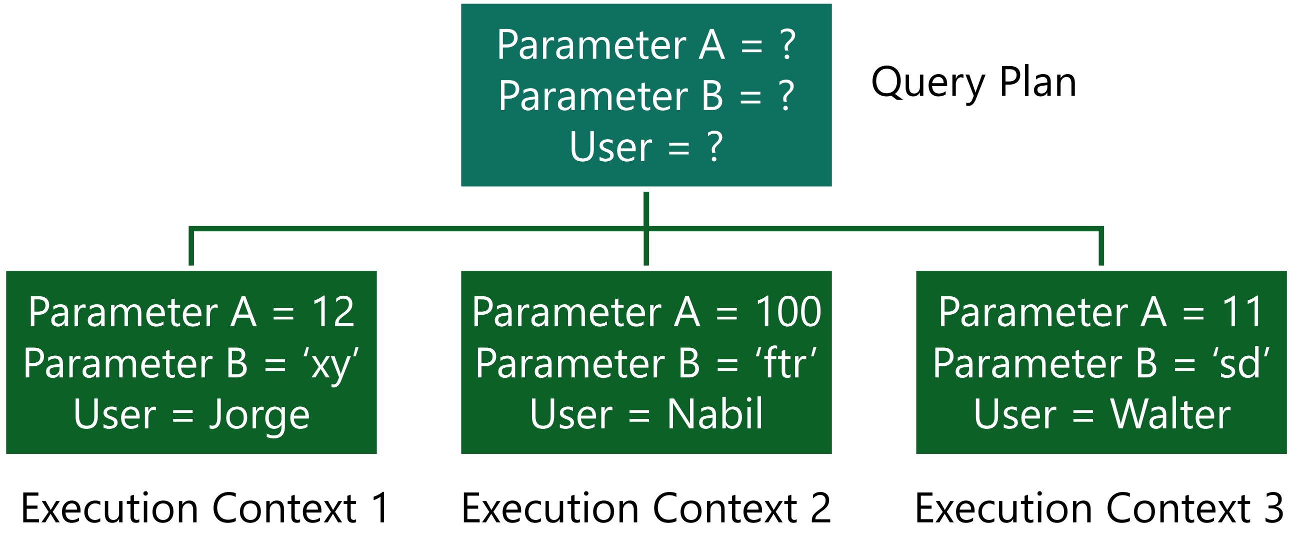
Main components

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	Query Execution
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.	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

Questions?

Knowledge Check

What is meant by SQL Server’s query optimizer being **cost-based**?

When is a query considered for a parallel execution plan?

Will SQL Server evaluate **every** possible query plan in the process of optimization? Why?

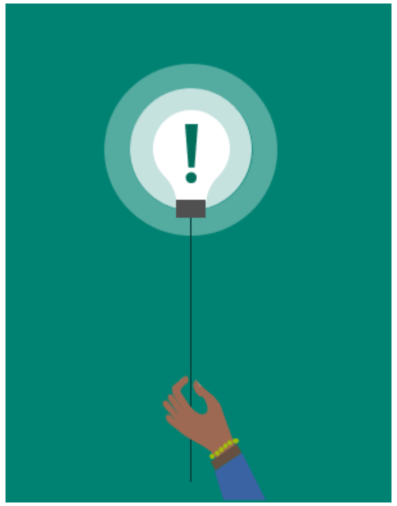
Name two recompilation causes.

Lesson 2: SQL Server Query Plan Analysis

Objectives

After completing this learning, you will be able to:

- Read execution plans.
- Understand logical and physical join operators.
- Describe data access.



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

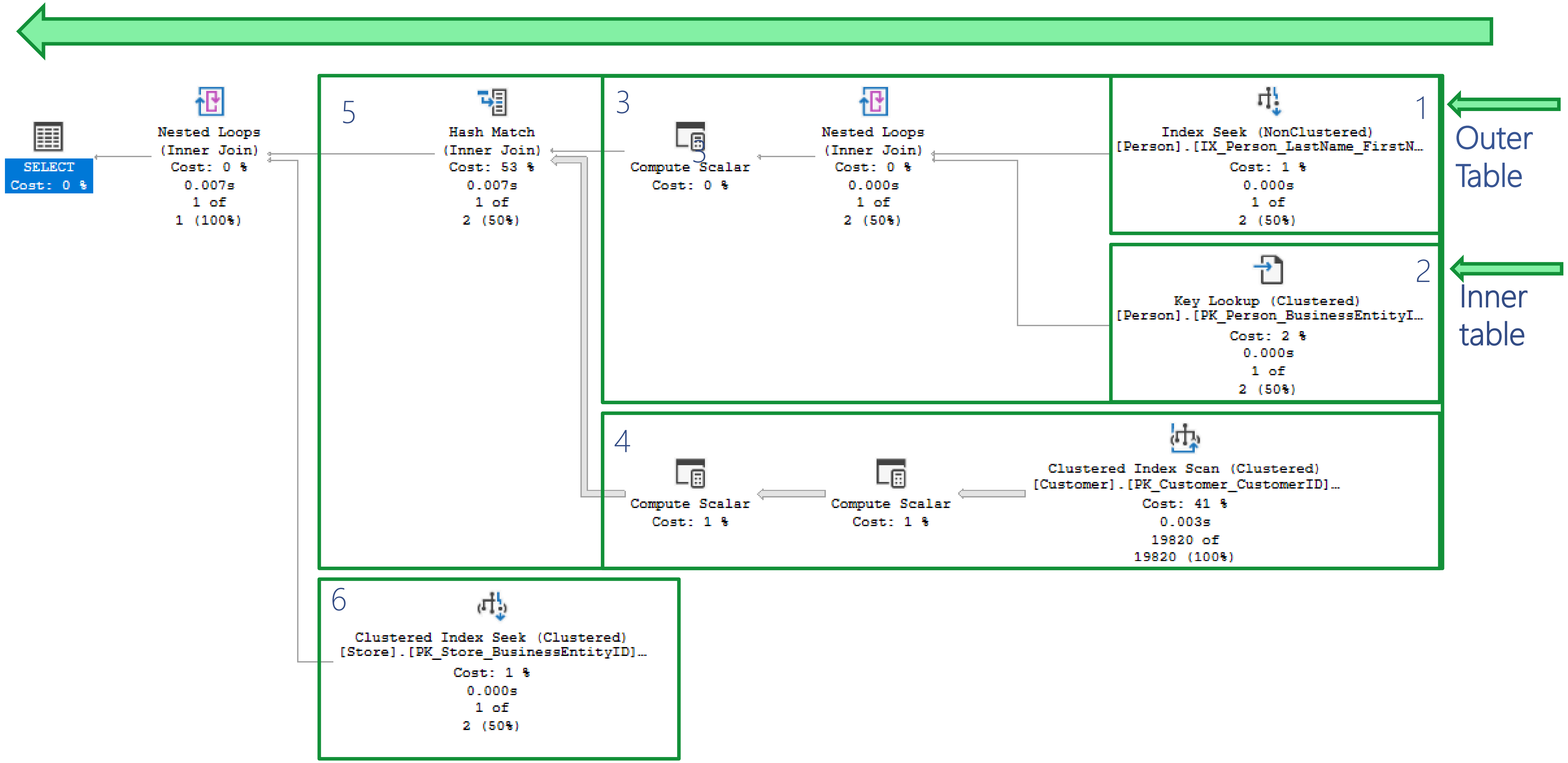
How to see the query plan

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

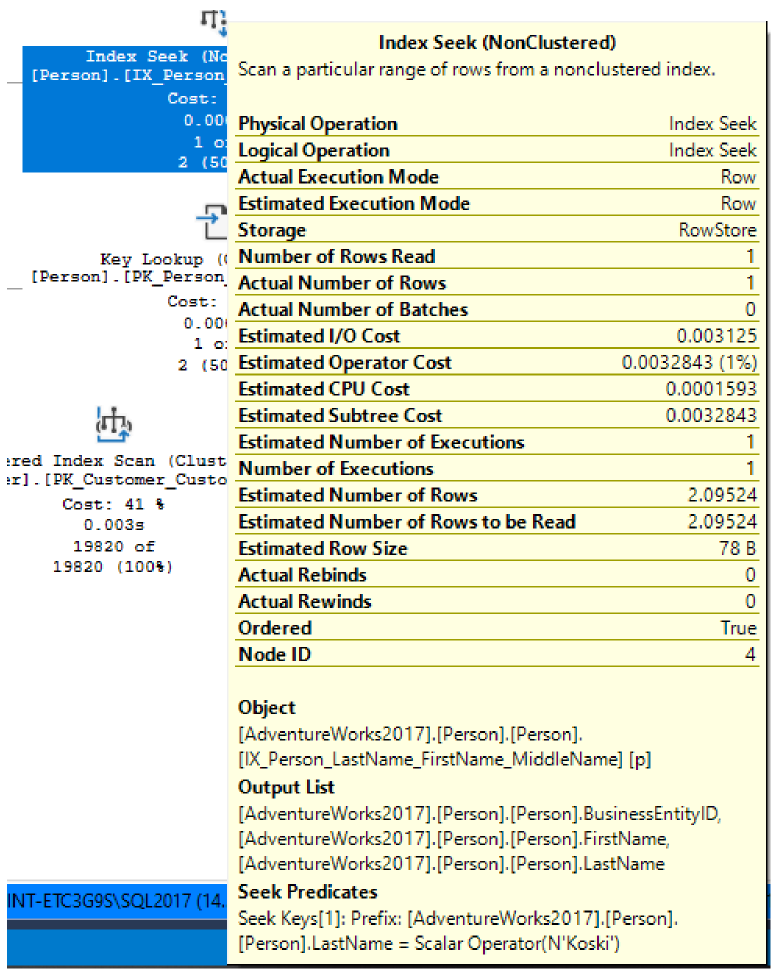
Execution Flow



SSMS Graphical Plan

Data flow between the operators and statistical data of each operator

Statistical data for the operator



Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	1
Actual Number of Rows	1
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032843 (1%)
Estimated CPU Cost	0.0001593
Estimated Subtree Cost	0.0032843
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	2.09524
Estimated Number of Rows to be Read	2.09524
Estimated Row Size	78 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

Object

[AdventureWorks2017].[Person].[Person].
[IX_Person_LastName_FirstName_MiddleName] [p]

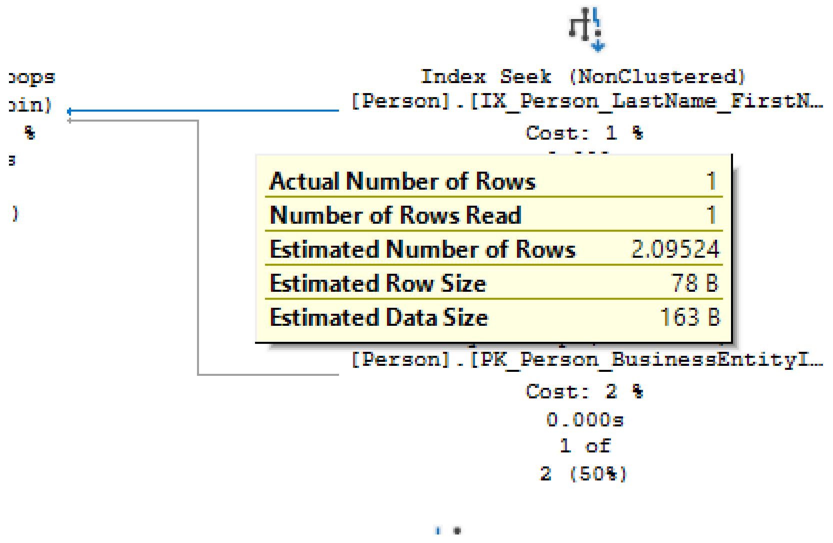
Output List

[AdventureWorks2017].[Person].[Person].BusinessEntityID,
[AdventureWorks2017].[Person].[Person].FirstName,
[AdventureWorks2017].[Person].[Person].LastName

Seek Predicates

Seek Keys[1]: Prefix: [AdventureWorks2017].[Person].
[Person].LastName = Scalar Operator(N'Koski')

Data flow statistics

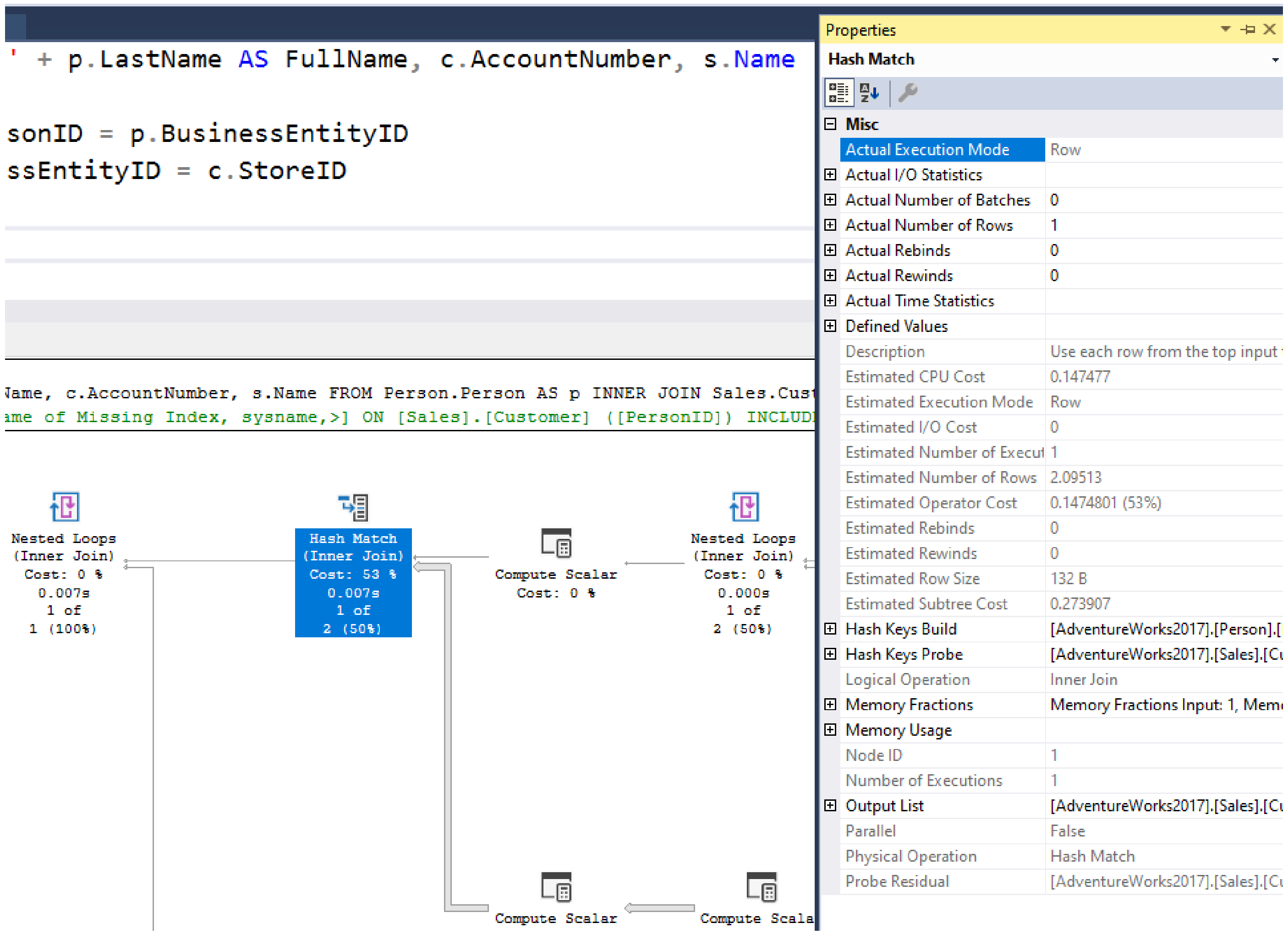


oops
oin)
%
s
)
Index Seek (NonClustered)
[Person].[IX_Person_LastName_FirstN…
Cost: 1 %
[Person].[PK_Person_BusinessEntityI…
Cost: 2 %
0.000s
1 of
2 (50%)

Actual Number of Rows	1
Number of Rows Read	1
Estimated Number of Rows	2.09524
Estimated Row Size	78 B
Estimated Data Size	163 B

SSMS Graphical Plan

Properties sheet



```
' + p.LastName AS FullName, c.AccountNumber, s.Name

sonID = p.BusinessEntityID
ssEntityID = c.StoreID
```

```
Jame, c.AccountNumber, s.Name FROM Person.Person AS p INNER JOIN Sales.Cust
ame of Missing Index, sysname,>] ON [Sales].[Customer] ([PersonID]) INCLUDI
```

Properties

Hash Match

Misc

Actual Execution Mode	Row
⊞ Actual I/O Statistics	
⊞ Actual Number of Batches	0
⊞ Actual Number of Rows	1
⊞ Actual Rebinds	0
⊞ Actual Rewinds	0
⊞ Actual Time Statistics	
⊞ Defined Values	
Description	Use each row from the top input
Estimated CPU Cost	0.147477
Estimated Execution Mode	Row
Estimated I/O Cost	0
Estimated Number of Execut	1
Estimated Number of Rows	2.09513
Estimated Operator Cost	0.1474801 (53%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	132 B
Estimated Subtree Cost	0.273907
⊞ Hash Keys Build	[AdventureWorks2017].[Person].[
⊞ Hash Keys Probe	[AdventureWorks2017].[Sales].[Cu
Logical Operation	Inner Join
⊞ Memory Fractions	Memory Fractions Input: 1, Mem
⊞ Memory Usage	
Node ID	1
Number of Executions	1
⊞ Output List	[AdventureWorks2017].[Sales].[Cu
Parallel	False
Physical Operation	Hash Match
Probe Residual	[AdventureWorks2017].[Sales].[Cu

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

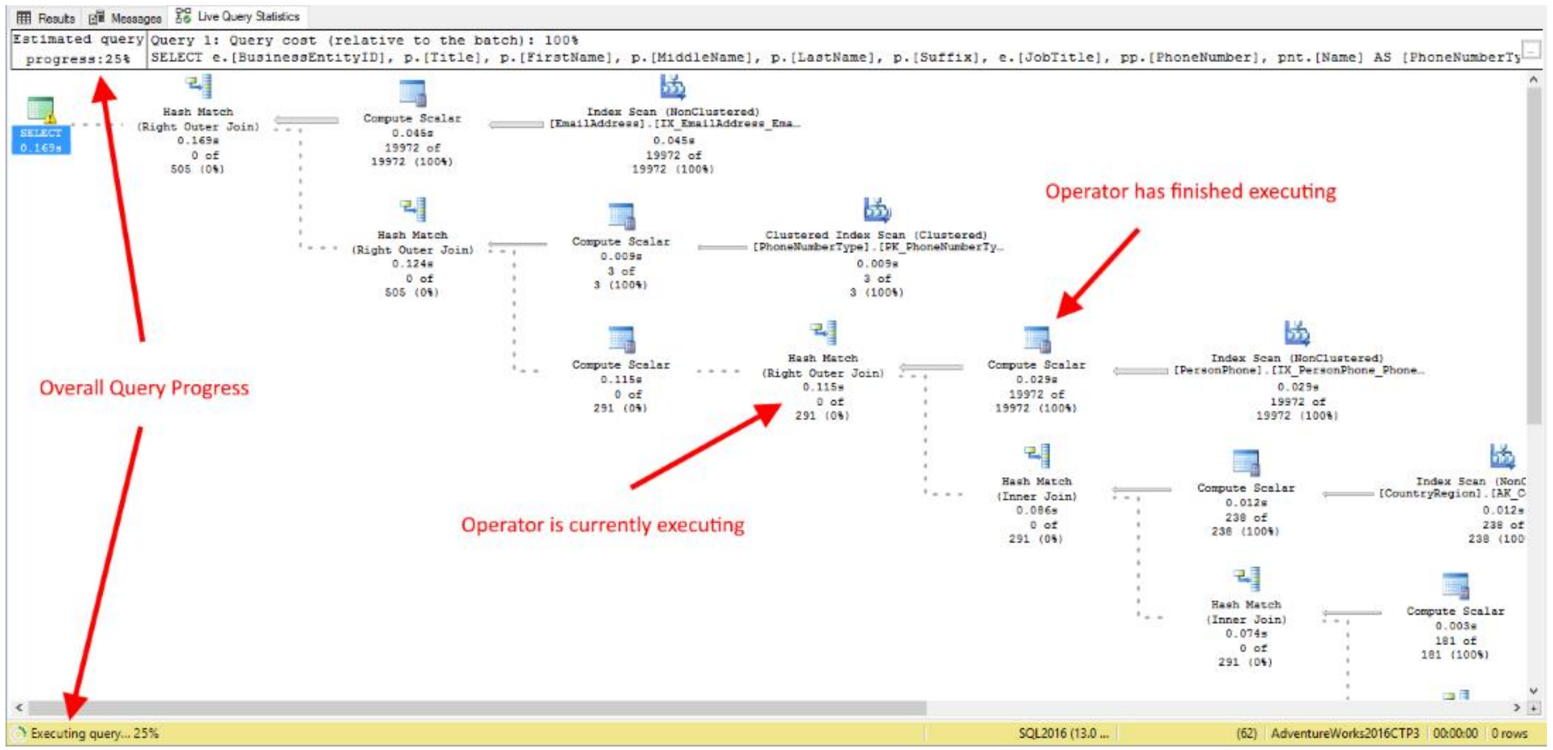
View CPU/memory usage, execution time, and query progress

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.


Results
Messages
Live Query Statistics
SELECT 0.169s
Hash Match (Right Outer Join) 0.169s 0 of 505 (0%)
Compute Scalar 0.046s 19972 of 19972 (100%)
Index Scan (NonClustered) [EmailAddress].[IX_EmailAddress_Ema… 0.045s 19972 of 19972 (100%)
Hash Match (Right Outer Join) 0.124s 0 of 505 (0%)
Compute Scalar 0.009s 3 of 3 (100%)
Clustered Index Scan (Clustered) [PhoneNumberType].[PK_PhoneNumberTy… 0.009s 3 of 3 (100%)
Compute Scalar 0.115s 0 of 291 (0%)
Hash Match (Right Outer Join) 0.115s 0 of 291 (0%)
Compute Scalar 0.029s 19972 of 19972 (100%)
Index Scan (NonClustered) [PersonPhone].[IX_PersonPhone_Phone… 0.029s 19972 of 19972 (100%)
Hash Match (Inner Join) 0.086s 0 of 291 (0%)
Compute Scalar 0.012s 238 of 238 (100%)
Index Scan (NonC [CountryRegion].[AK_C 0.012s 238 of 238 (100
Hash Match (Inner Join) 0.074s 0 of 291 (0%)
Compute Scalar 0.003s 181 of 181 (100%)
Operator has finished executing
Overall Query Progress
Operator is currently executing

```
Estimated query progress:25%
Query 1: Query cost (relative to the batch): 100%
SELECT e.[BusinessEntityID], p.[Title], p.[FirstName], p.[MiddleName], p.[LastName], p.[Suffix], e.[JobTitle], pp.[PhoneNumber], pnt.[Name] AS [PhoneNumberTy
```

```
Executing query... 25%    SQL2016 (13.0 ...    (62)    AdventureWorks2016CTP3    00:00:00    0 rows
```

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.

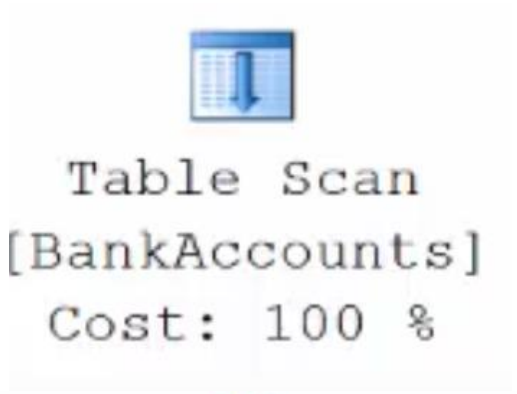
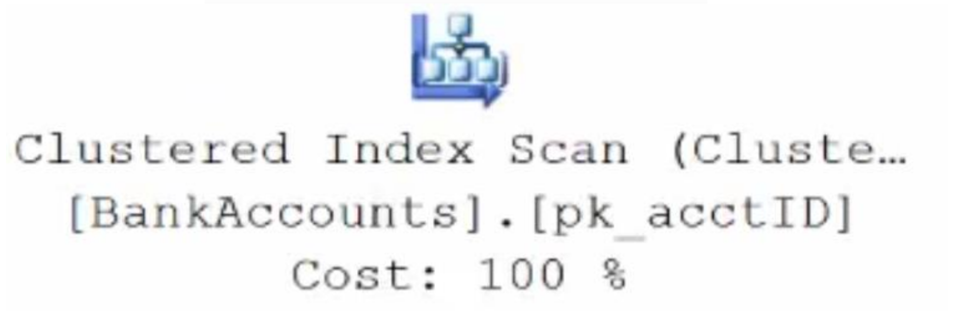


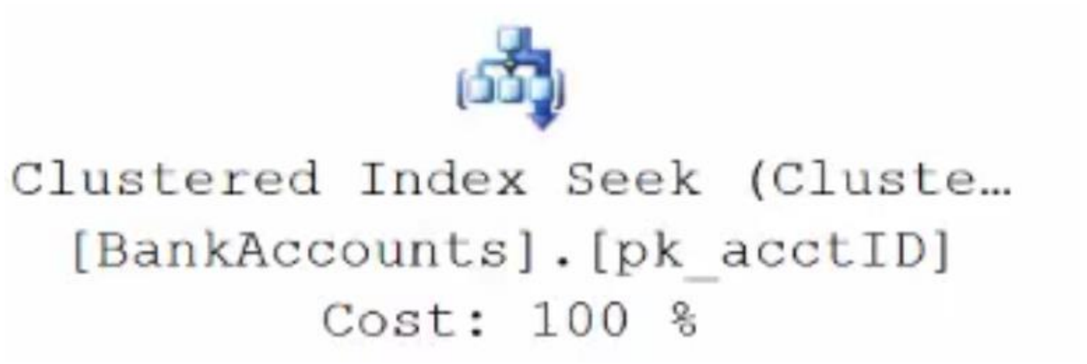
Table Scan
[BankAccounts]
Cost: 100 %

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



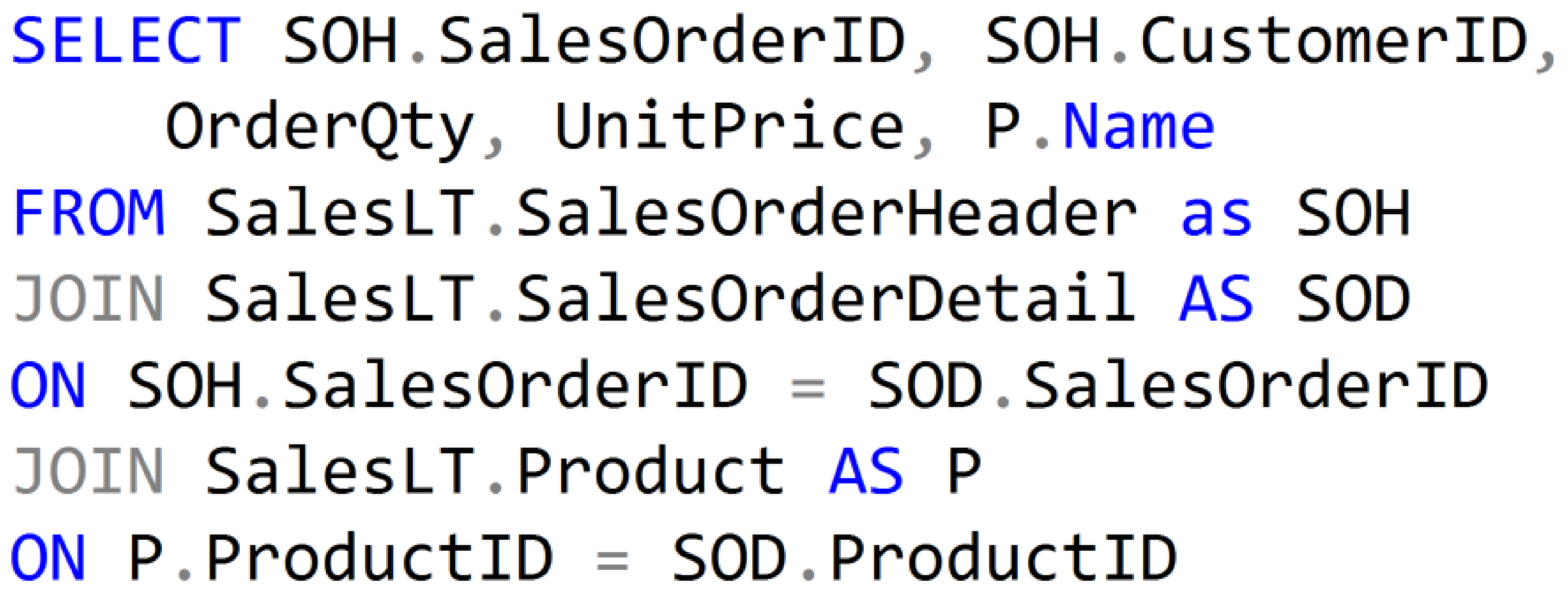
Clustered Index Scan (Cluste…
[BankAccounts].[pk_acctID]
Cost: 100 %

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 %

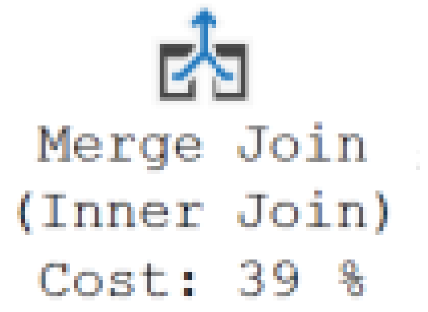
Execution Plan Join Operators (Code)



```sql
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
```

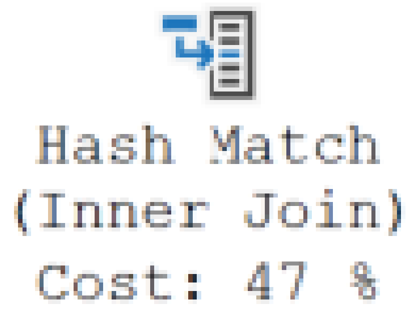
Execution Plan Join Operators

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



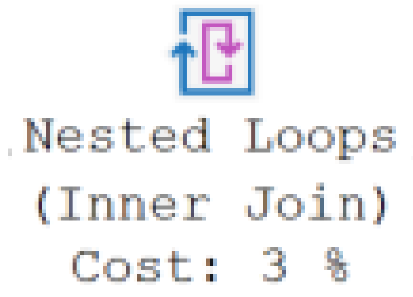
```
  Merge Join
(Inner Join)
 Cost: 39 %
```

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
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

Demonstration

Query Plan Analysis

- Use the graphical execution plan and IO statistics to tune a query.
- Explore Live Query Statistics.

Query Plan Analysis

- Identifying and tuning high cost operators in the query plan

LAB

Questions?

Knowledge Check

What are the physical join operators?

What is a method to eliminate a lookup?

Is it recommended to eliminate all lookups operators?

Under what circumstances would a table scan be more efficient than an index seek on a non-clustered, non-covering index?

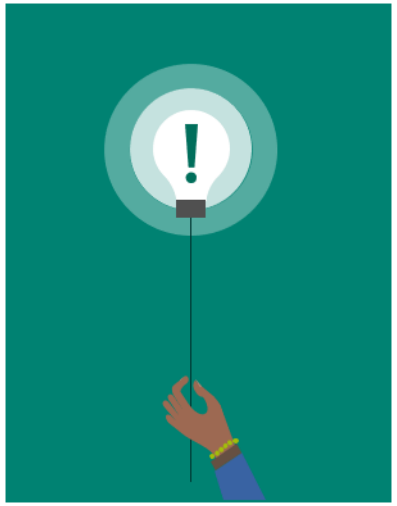
Is a Clustered Index Scan more efficient than a Table Scan?

Lesson 3: SQL Server Intelligent Query Processing

Objectives

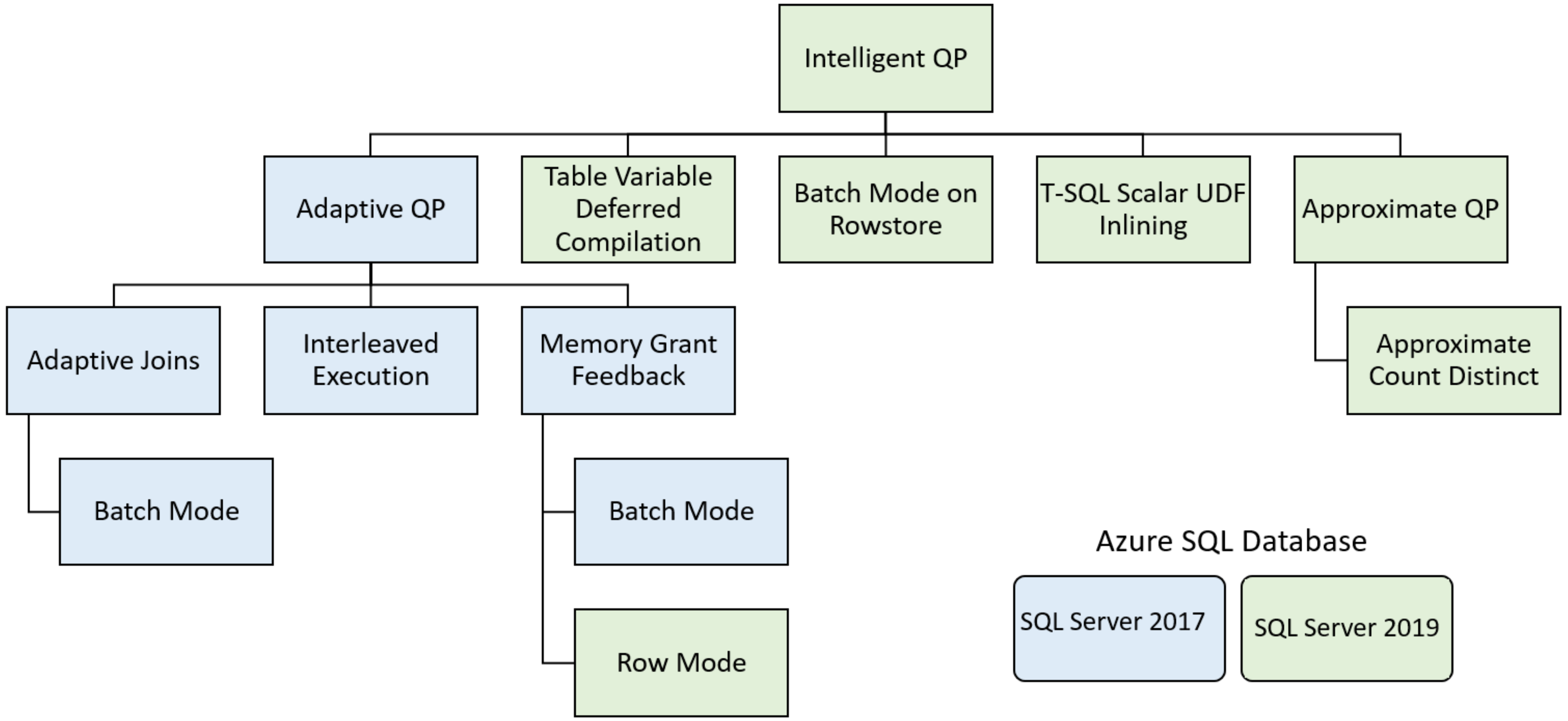
After completing this learning, you will be able to:

- Understand the Intelligent query processing features.
- Enable/disable Intelligent query processing features.



Intelligent query processing

Features family



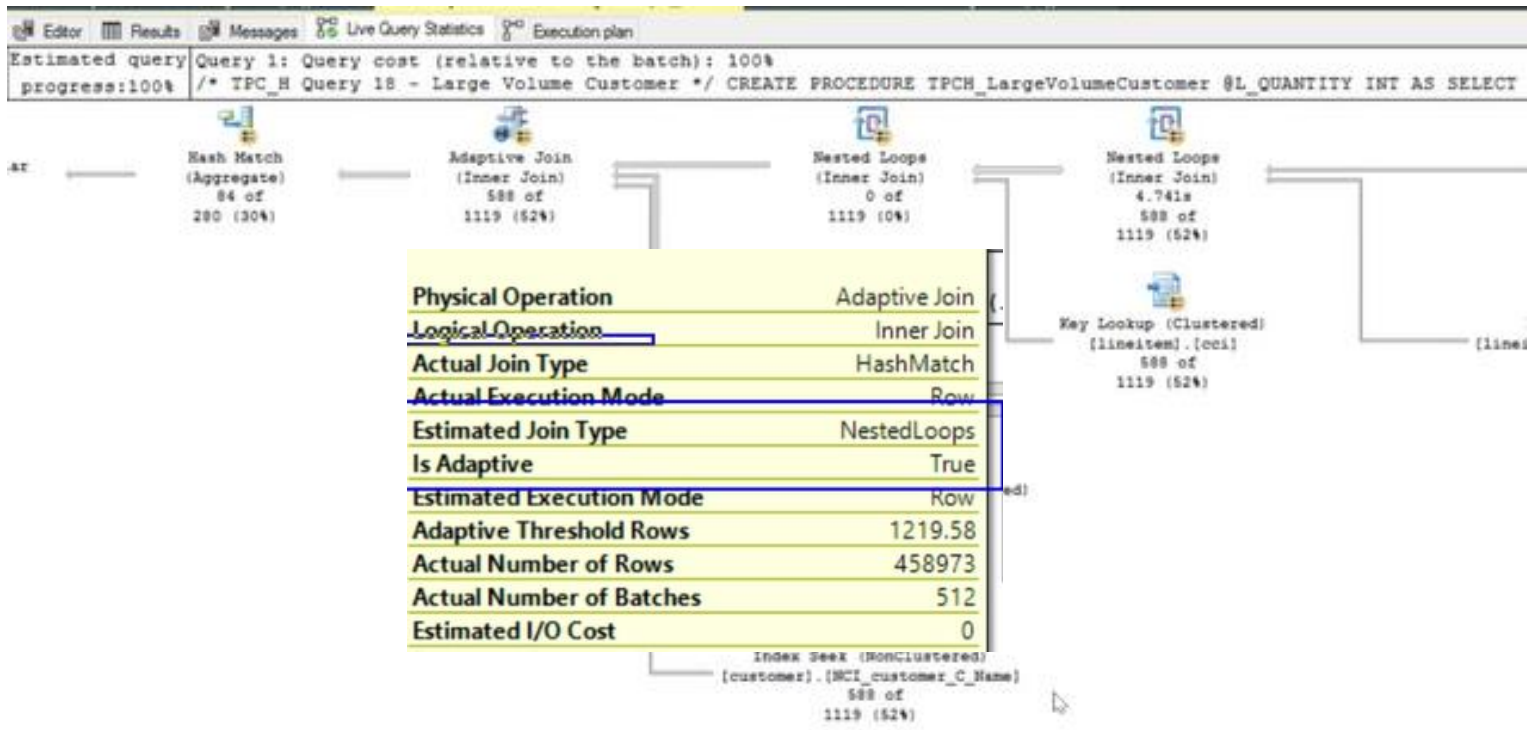
Adaptive joins (Batch mode)

Adaptive join introduction

The choice of a Hash Join or Nested Loops join method is deferred until after the first input has been scanned.

A threshold is used to decide when to switch to a Nested Loops plan, allowing a query plan to use a better join strategy during execution without to be recompiled.

Workloads with frequent oscillations between small and large join input scans will benefit most from this feature.



```
Estimated query   Query 1: Query cost (relative to the batch): 100%
 progress:100%    /* TPC_H Query 18 - Large Volume Customer */ CREATE PROCEDURE TPCH_LargeVolumeCustomer @L_QUANTITY INT AS SELECT
```

```
Hash Match (Aggregate)          84 of 280 (30%)
Adaptive Join (Inner Join)      588 of 1119 (52%)
Nested Loops (Inner Join)       0 of 1119 (0%)
Nested Loops (Inner Join)       4.741s  588 of 1119 (52%)
Key Lookup (Clustered) [lineitem].[cci]   588 of 1119 (52%)
Index Seek (NonClustered) [customer].[NCI_customer_C_Name]   588 of 1119 (52%)
```

Physical Operation	Adaptive Join
Logical Operation	Inner Join
Actual Join Type	HashMatch
Actual Execution Mode	Row
Estimated Join Type	NestedLoops
Is Adaptive	True
Estimated Execution Mode	Row
Adaptive Threshold Rows	1219.58
Actual Number of Rows	458973
Actual Number of Batches	512
Estimated I/O Cost	0

Adaptive joins (Batch mode)

Enabling/Disabling Adaptive joins

- Enabled by default in Compatibility level 140 or higher.
- To disable change compatibility level to 130 or lower (not recommended).
- You can enable/disable it at the database level in

  SQL Server 2017.

```
ALTER DATABASE SCOPED CONFIGURATION SET DISABLE_BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

  Azure SQL Database, SQL Server 2019 and higher

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

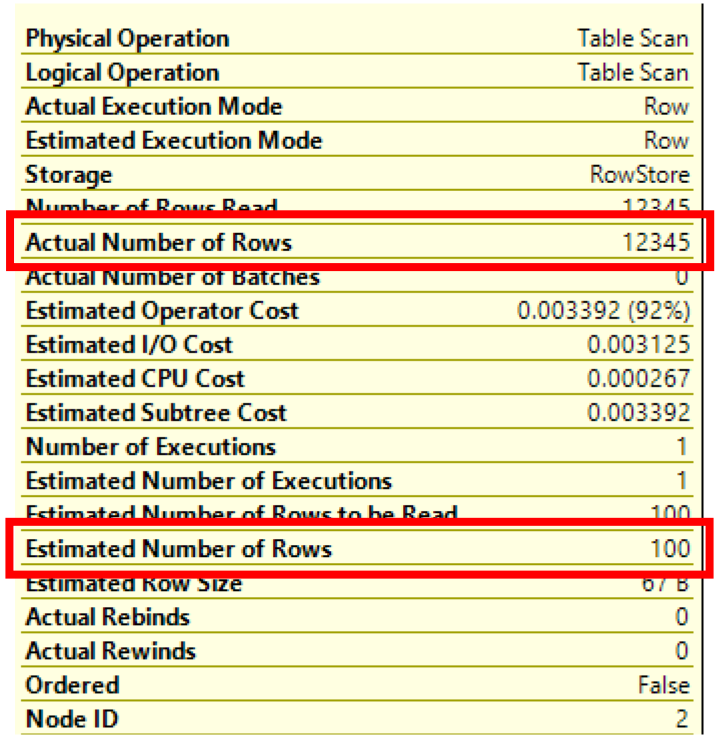
- You can disable it at statement scope if necessary.

```
<statement>
OPTION (USE HINT('DISABLE_BATCH_MODE_ADAPTIVE_JOINS'));
```

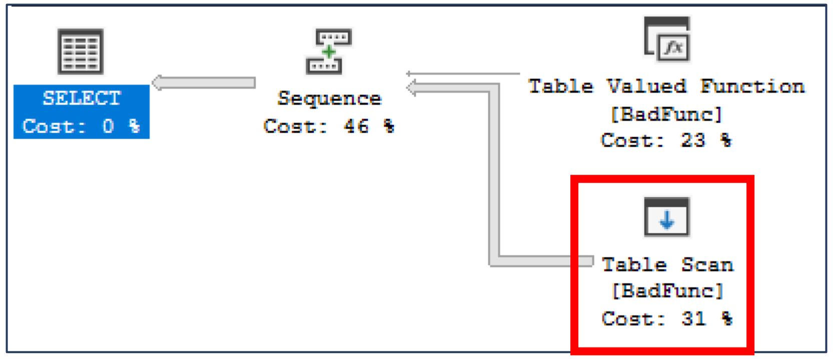
Interleaved Execution

Overview

Compatibility Level 120/130

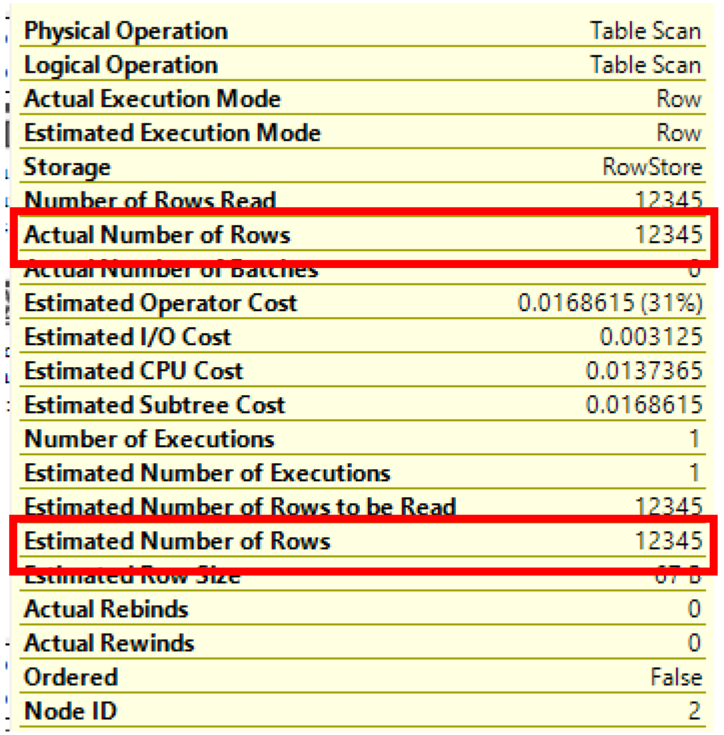


Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	12345
Actual Number of Rows	12345
Actual Number of Batches	0
Estimated Operator Cost	0.003392 (92%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.000267
Estimated Subtree Cost	0.003392
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	100
Estimated Number of Rows	100
Estimated Row Size	67 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2



During optimization if SQL Server encounter a read-only multi-statement table-valued function (MSTVF), it will pause optimization, execute the applicable subtree, capture accurate cardinality estimates, and then resume optimization for downstream operations.

Compatibility Level 140 or higher



Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	12345
Actual Number of Rows	12345
Actual Number of Batches	0
Estimated Operator Cost	0.0168615 (31%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0137365
Estimated Subtree Cost	0.0168615
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	12345
Estimated Number of Rows	12345
Estimated Row Size	67 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Interleaved Execution

Enabling/Disabling Interleaved Execution

- Enabled by default in Compatibility level 140 or higher.
- To disable change compatibility level to 130 or lower (not recommended).
- You can enable/disable it at the database level

  SQL Server 2017.

```
ALTER DATABASE SCOPED CONFIGURATION SET DISABLE_INTERLEAVED_EXECUTION_TVF = ON|OFF;
```

  Azure SQL Database, SQL Server 2019 and higher

```
ALTER DATABASE SCOPED CONFIGURATION SET INTERLEAVED_EXECUTION_TVF = ON|OFF;
```

- You can disable it at statement scope if necessary.

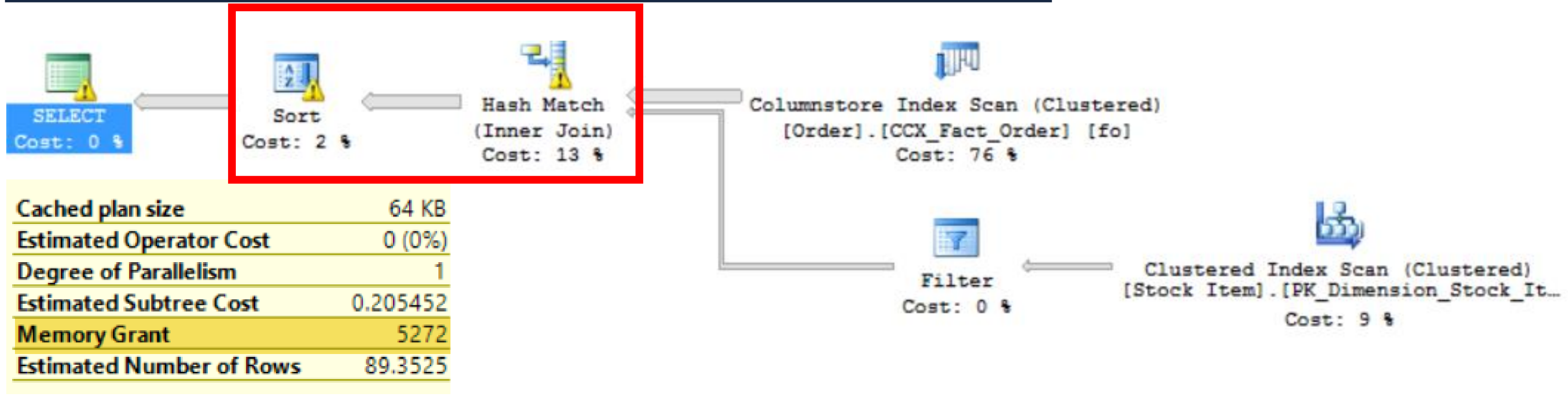
```
<statement>
OPTION (USE HINT('DISABLE_INTERLEAVED_EXECUTION_TVF'));
```

Memory Grant Feedback (Batch Mode)

By addressing repeating workloads, it recalculates the actual memory required for a query and then updates the grant value for the cached plan.

For parameter-sensitive plans, memory grant feedback will disable itself on a query if it has unstable memory requirements.

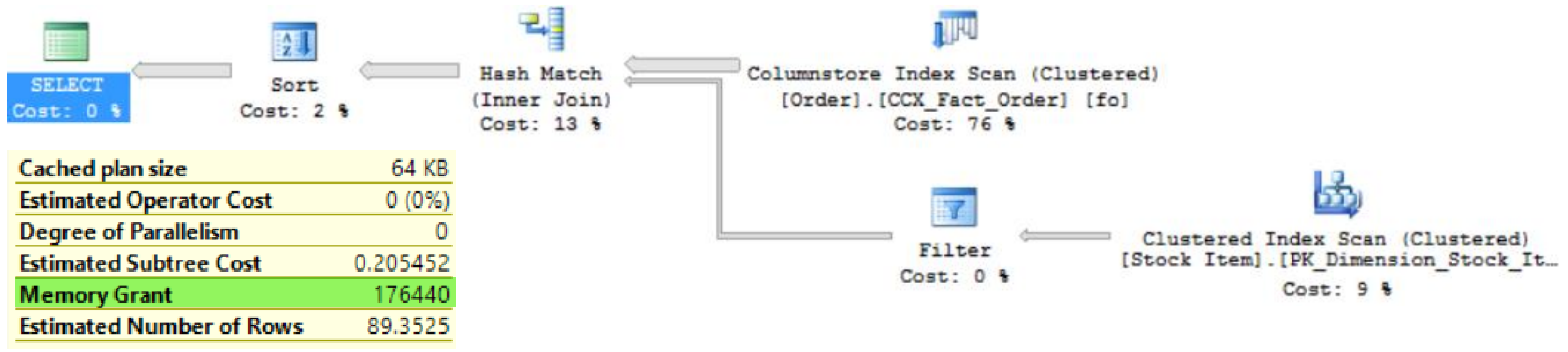
First Execution (Spills detected; feedback generated)



SELECT
Cost: 0 %
Sort
Cost: 2 %
Hash Match
(Inner Join)
Cost: 13 %
Columnstore Index Scan (Clustered)
[Order].[CCX_Fact_Order] [fo]
Cost: 76 %
Filter
Cost: 0 %
Clustered Index Scan (Clustered)
[Stock Item].[PK_Dimension_Stock_It…
Cost: 9 %

Cached plan size	64 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	0.205452
Memory Grant	5272
Estimated Number of Rows	89.3525

Second Execution (Memory grant adjusted)



SELECT
Cost: 0 %
Sort
Cost: 2 %
Hash Match
(Inner Join)
Cost: 13 %
Columnstore Index Scan (Clustered)
[Order].[CCX_Fact_Order] [fo]
Cost: 76 %
Filter
Cost: 0 %
Clustered Index Scan (Clustered)
[Stock Item].[PK_Dimension_Stock_It…
Cost: 9 %

Cached plan size	64 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	0
Estimated Subtree Cost	0.205452
Memory Grant	176440
Estimated Number of Rows	89.3525

Memory Grant Feedback (Batch Mode)

Enabling/Disabling Memory Grant Feedback (Batch Mode)

- Enabled by default in Compatibility level 140 or higher.
- To disable change compatibility level to 130 or lower (not recommended).
- You can enable/disable it at the database level

  SQL Server 2017.

```
ALTER DATABASE SCOPED CONFIGURATION SET DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK=ON|OFF;
```

  Azure SQL Database, SQL Server 2019 and higher

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_MEMORY_GRANT_FEEDBACK = ON|OFF;
```

- You can disable it at statement scope if necessary.

```
<statement>
OPTION (USE HINT ('DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK'))
```

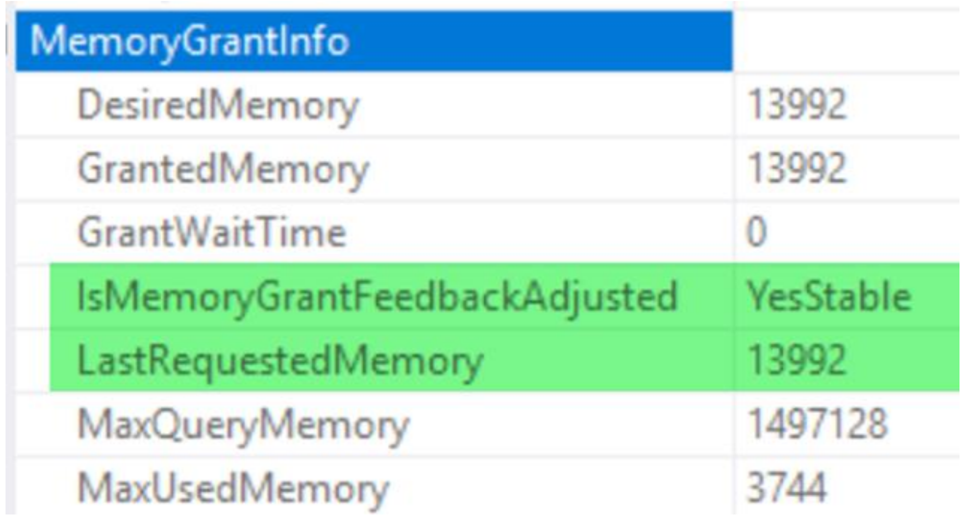
Memory Grant Feedback (Row Mode)

Overview

Expands on the batch mode memory grant feedback feature by also adjusting memory grant sizes for row mode operators.

Two new query plan attributes will be shown for actual post-execution plans:

- IsMemoryGrantFeedbackAdjusted
- LastRequestedMemory



MemoryGrantInfo	
DesiredMemory	13992
GrantedMemory	13992
GrantWaitTime	0
IsMemoryGrantFeedbackAdjusted	YesStable
LastRequestedMemory	13992
MaxQueryMemory	1497128
MaxUsedMemory	3744

Memory Grant Feedback (Row Mode)

Enabling/Disabling Memory Grant Feedback (Batch Mode)

- Enabled by default in Compatibility level 150 or higher.
- To disable change compatibility level to 140 or lower (not recommended).
- You can enable/disable it at the database level.

```
ALTER DATABASE SCOPED CONFIGURATION SET ROW_MODE_MEMORY_GRANT_FEEDBACK = ON|OFF;
```

- You can disable it at statement scope if necessary.

```
<statement>
OPTION (USE HINT ('DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK'));
```

Table Variable Deferred Compilation

Overview

Compilation of a statement that references a table variable is deferred until the first actual execution.

Table variable deferred compilation doesn't change any other characteristics of table variables.

Table variable deferred compilation **doesn't increase recompilation frequency**.

Cached plan is generated based on the initial deferred compilation table variable row count instead of the original one-row guess.

Performance may not be improved by this feature if the table variable row count varies significantly across executions.

Table Variable Deferred Compilation

Enabling/Disabling Table Variable Deferred Compilation

- Enabled by default in Compatibility level 150 or higher.
- To disable change compatibility level to 140 or lower (not recommended).
- You can enable/disable it at the database level.

```
ALTER DATABASE SCOPED CONFIGURATION SET DEFERRED_COMPILATION_TV = ON|OFF;
```

- You can disable it at statement scope if necessary.

```
<table variable declaration>

<data inserted into table variable >

<statement that uses the table variable>
OPTION (USE HINT('DISABLE_DEFERRED_COMPILATION_TV'));
```

Approximate Count Distinct

Overview

Designed for use in big data scenarios and is optimized for the following conditions:

- Access of data sets that are millions of rows or higher and
- Aggregation of a column or columns that have many distinct values

APPROX_COUNT_DISTINCT

It returns the approximate number of unique non-null values in a group.

It is designed to provide aggregations across large data sets where responsiveness is more critical than absolute precision.

Its implementation guarantees up to a 2% error rate within a 97% probability.

requires less memory than an exhaustive COUNT DISTINCT operation so it is less likely to spill memory to disk compared to COUNT DISTINCT.

Batch Mode on RowStore

Overview

Even if a query does not access any tables with ColumnStore indexes, the query processor, using heuristics, will decide whether to consider batch mode.

If batch mode on rowstore is used, you see the actual run mode as **batch mode** in the query plan.

CPU consumption Reduced

No gain in I/O consumption unless data is already in cache

Performance improvement only possible if enough memory to cache all data

Batch Mode on Rowstore

Enabling/Disabling Batch Mode on Rowstore

- Enabled by default in Compatibility level 150 or higher.
- To disable change compatibility level to 140 or lower (not recommended).
- You can enable/disable it at the database level.

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

- You can disable it at statement scope if necessary.

```
<statement>
OPTION(RECOMPILE, USE HINT('DISALLOW_BATCH_MODE'));
```

- Enable it at statement scope (even with the feature disabled at database level).

```
<statement>
OPTION(RECOMPILE, USE HINT('ALLOW_BATCH_MODE'));
```

Scalar UDF Inlining

T-SQL Scalar UDF performance issues

Iterative invocation: Invoked once per qualifying row. Repeated context switching – and even worse for UDFs that execute SQL queries in their definition.

Lack of costing: Scalar operators are not costed (realistically).

Interpreted execution: Each statement itself is compiled, and the compiled plan is cached. No cross-statement optimizations are carried out.

Serial execution: SQL Server does not allow intra-query parallelism in queries that invoke UDFs.

Scalar UDF Inlining

Overview

The goal is to improve performance of queries that invoke T-SQL scalar UDFs, where UDF execution is the main bottleneck.

A scalar T-SQL UDF can not always be inline as conditions apply.

Scalar UDFs are automatically transformed into scalar expressions or scalar subqueries that are substituted in the calling query in place of the UDF operator.

These expressions and subqueries are then optimized. As a result, the query plan will no longer have a user-defined function operator, but its effects will be observed in the plan, like views or inline TVFs.

Scalar UDF Inlining

Enabling/Disabling Scalar UDF Inlining

- Enabled by default in Compatibility level 150 or higher.
- To disable change compatibility level to 140 or lower (not recommended).
- You can enable/disable it at the database level.

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

- You can disable it at statement scope if necessary.

```
<statement>
OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'));
```

Demonstration

Intelligent query processing

- Interleaved Execution
- Batch Mode on RowStore
- Memory Grant Feedback (Row Mode)

Intelligent query processing

- Observing Batch-Mode Memory Grant Feedback
- Using APPROX_COUNT_DISTINCT to improve performance
- Observing Table Variable Deferred Compilation
- Observing Scalar UDF Inlining

LAB

Questions?

Knowledge Check

Is it possible to disable Intelligent Query Processing features?

On queries not using Interleaved execution for MSTVFs. How many rows are estimated for a MSTVFs?

Does table variable deferred compilation increase the recompilation frequency?

On queries not using table variable deferred compilation. How many rows are estimated for a table variable?

What is the minimum compatibility level that supports Batch mode on rowstore?

Microsoft