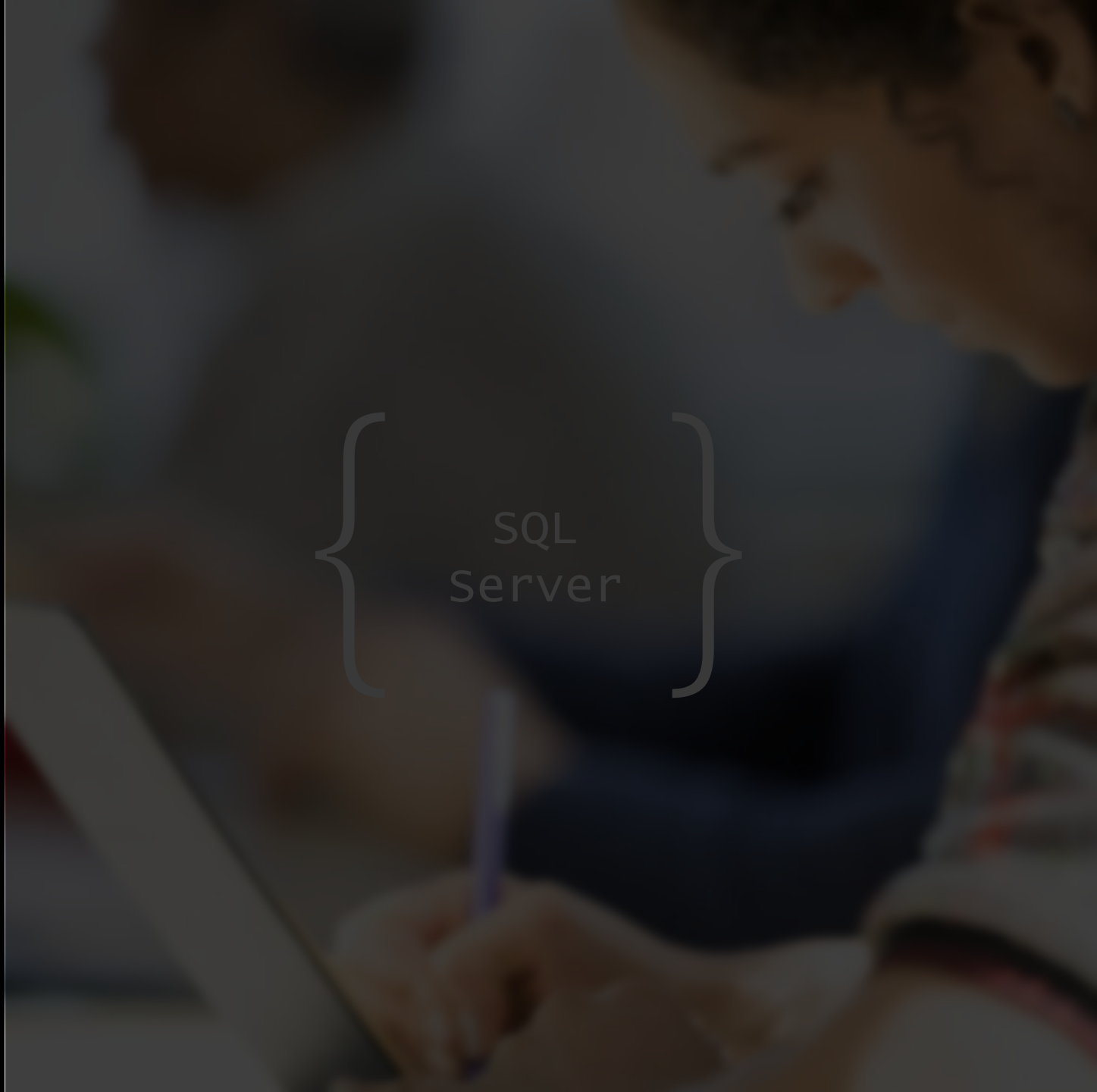
Writing T-SQL for Performance



John Deardurff

Senior Cloud Solution Architect (Global Delivery)
Microsoft Certified Trainer (2000 - Current)
MVP: Data Platform (2016 – 2019)

Email: John.Deardurff@Microsoft.com
Website: www.SQLMCT.com
GitHub: github.com\SQLMCT
LinkedIn: /in/johndeardurff/

MOST VALUABLE PROFESSIONAL Microsoft ALUMNI

Microsoft CERTIFIED TRAINER MCT

Microsoft CERTIFIED POWER BI DATA ANALYST ASSOCIATE

Microsoft CERTIFIED AZURE DATABASE ADMINISTRATOR ASSOCIATE

Microsoft CERTIFIED AZURE DATA ENGINEER ASSOCIATE

SQLskills SQL Server 2019 BLACKBELT

Agenda

Lesson 1: Writing Selective Queries

- Logical Query Processing
- Quick Query Writing Tips
- Selecting Specific Columns

Lesson 2: SARGable Expressions

- What is SARGability?
- Non-SARGable expressions
- Multiple Column Searches

Get Ready to Participate

Close email and Update Status

Make it interactive

Check all equipment

Be Respectful!

Be Kind!

We are all here to learn!

Lesson 1: Writing Selective Queries

What does this lesson cover?

- Logical Query Processing
- Query Writing Tips
- Selecting Specific Columns
- SET STATISTICS IO and TIME

Logical Query Processing

The order in which a query is written is not the order in which it is processed.

Order	Keywords	Expression	Purpose
5	SELECT	<select list>	Specifies which columns to return.
1	FROM	<table source>	Defines the table(s) to query
2	WHERE	<search condition>	Filters out the records to return
3	GROUP BY	<group by list>	Arranges records into groups
4	HAVING	<search condition>	Filters down the groups
6	ORDER BY	<order by list>	Sorts the returned records

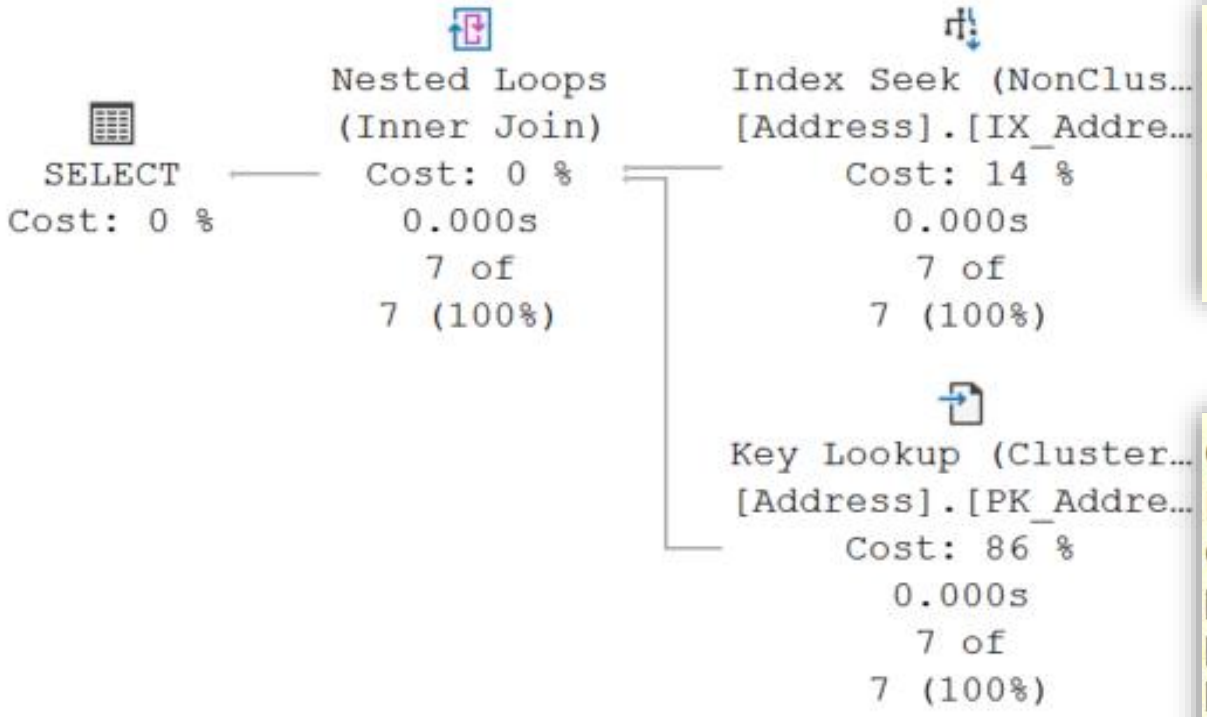
The earlier in this processing order you can eliminate data, the better.

Quick Query Writing Tips

Tip	Reason
Avoid using SELECT (*)	Only retrieve columns you need to reduce data returned
Use SARGable expressions	Provide good Search Arguments that can use existing indexes.
Use JOINS efficiently	Using the correct JOIN type reduces the amount of data returned.
Keep Transactions Short	Long-running transactions can lock resources and degrade performance.
Use of NOLOCK	Avoid using Table Hints if possible. However, the NOLOCK table hint can help improve report-based queries.
Use of Common Table Expressions	Common Table Expressions (CTEs) can be useful; if written properly.

Avoid Using SELECT *

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Person].[Address] WHERE [StateProvinceID]=@1
```



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

Do we need these columns?

SET STATISTICS IO

```sql
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.
```

Demonstration

SELECT * and Key Lookups

Lesson 2: SARGable Expressions

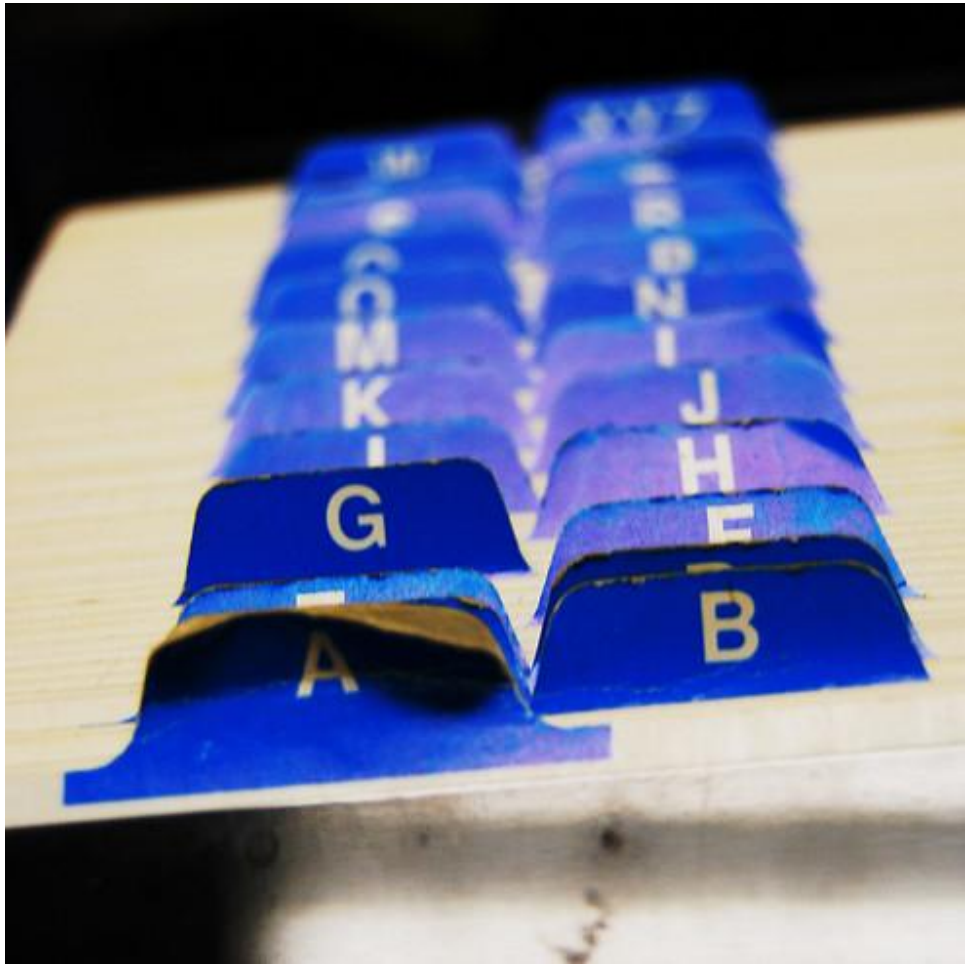
What does this lesson cover?

- What is SARGability?
- Non-SARGable expressions
- Multiple Column Searches

What is SARGability?

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

Non-SARGable expressions can significantly slow down queries.



Non-SARGable Expressions

Avoid leading wildcard with LIKE operator

WHERE Employee.FirstName LIKE '%L%' is non-SARGable
WHERE Employee.FirstName LIKE ‘L%' is SARGable

Avoid using Scalar Functions

WHERE ABS(ProductID) = 771
WHERE UPPER(City) = “London”
WHERE UPPER(surname) = “SMITHS”

Avoid Calculations or Expressions

WHERE Sales.Price + Sales.Tax > 100
WHERE Sales.Price * (1 + Sales.TaxRate) > 100

Avoid using NOT in your searches

WHERE Employee.Firstname NOT = ‘Smith’
WHERE Employee.Firstname NOT IN (‘Smith’, ‘Jones’)

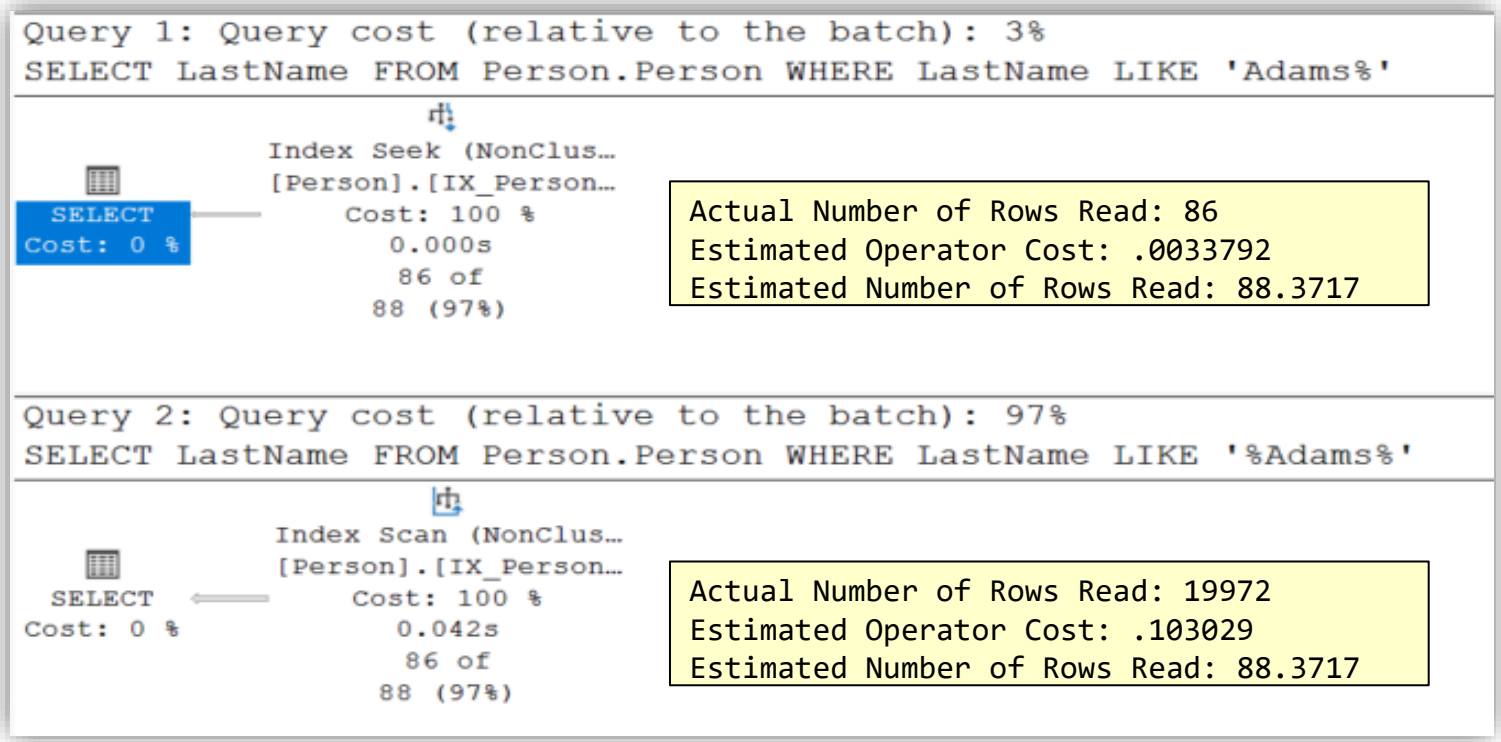
Non-SARGable Expressions – Leading Wildcards

```sql
-- SARGable
SELECT LastName FROM Person.Person
WHERE LastName LIKE 'Adams%’

-- NON-SARGable
SELECT LastName FROM Person.Person
WHERE LastName LIKE '%Adams%'
```

```
(86 rows affected)
Table 'Person'. Scan count 1, logical reads 4

(86 rows affected)
Table 'Person'. Scan count 1, logical reads 111
```



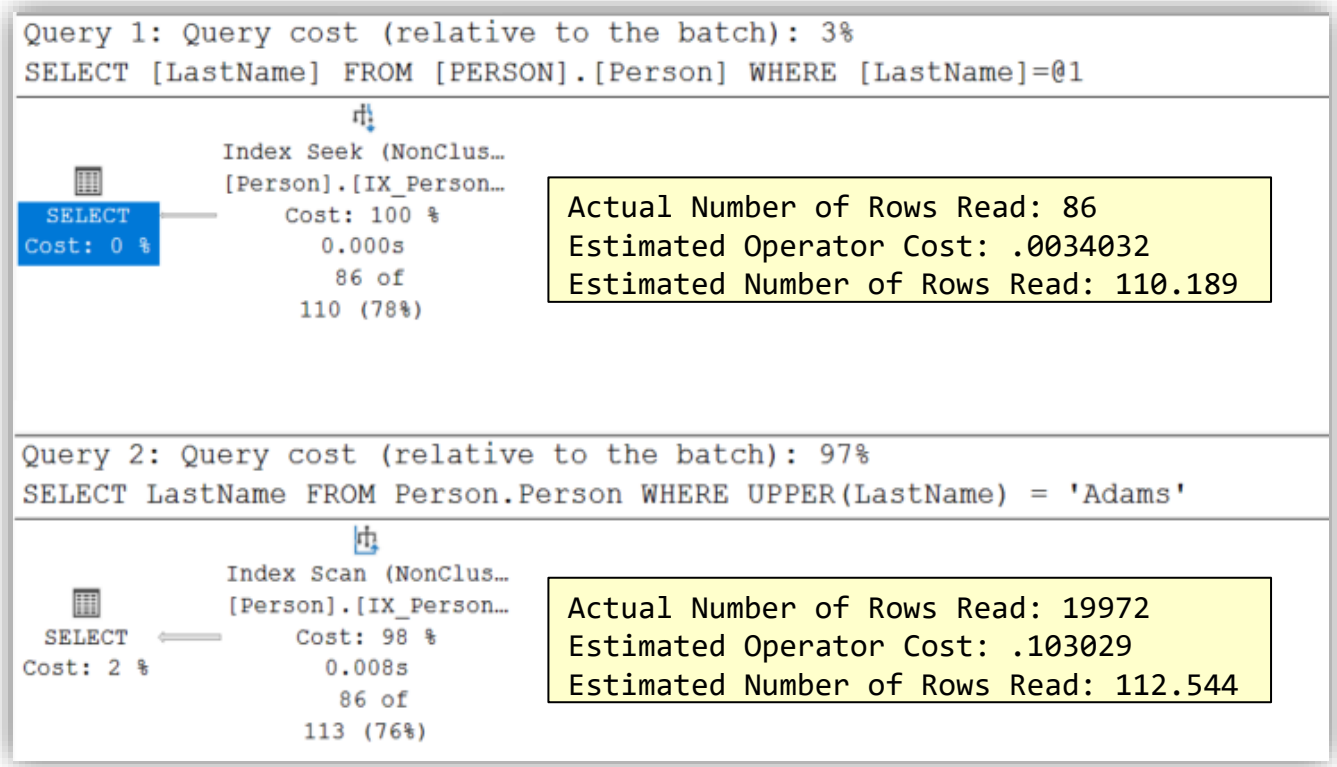
Non-SARGable Expressions – Scalar Functions (Characters)

```sql
-- SARGable
SELECT LastName FROM Person.Person
WHERE LastName = 'Adams’

-- NON-SARGable
SELECT LastName FROM Person.Person
WHERE UPPER(LastName) = 'Adams'
```

```
(86 rows affected)
Table 'Person'. Scan count 1, logical reads 4

(86 rows affected)
Table 'Person'. Scan count 1, logical reads 111
```



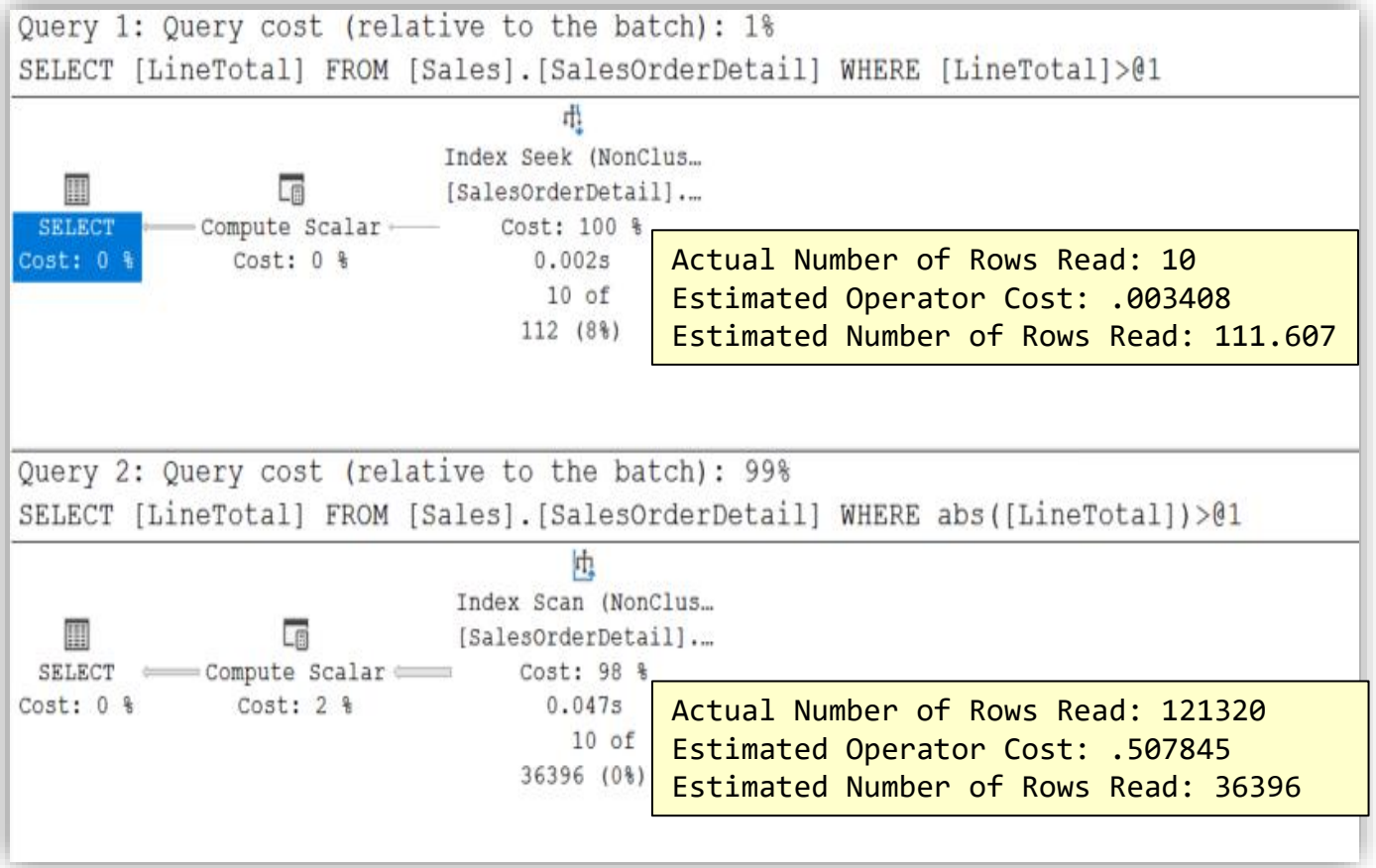
Non-SARGable Expressions – Scalar Functions (Integers)

```sql
-- SARGable
SELECT LineTotal FROM
Sales.SalesOrderDetail
WHERE LineTotal > 23000

-- NON-SARGable
SELECT LineTotal FROM
Sales.SalesOrderDetail
WHERE ABS(LineTotal) > 23000
```

```
(10 rows affected)
Table 'SalesOrderDetail'. Scan count 1, logical reads 3

(10 rows affected)
Table 'SalesOrderDetail'. Scan count 1, logical reads 512
```



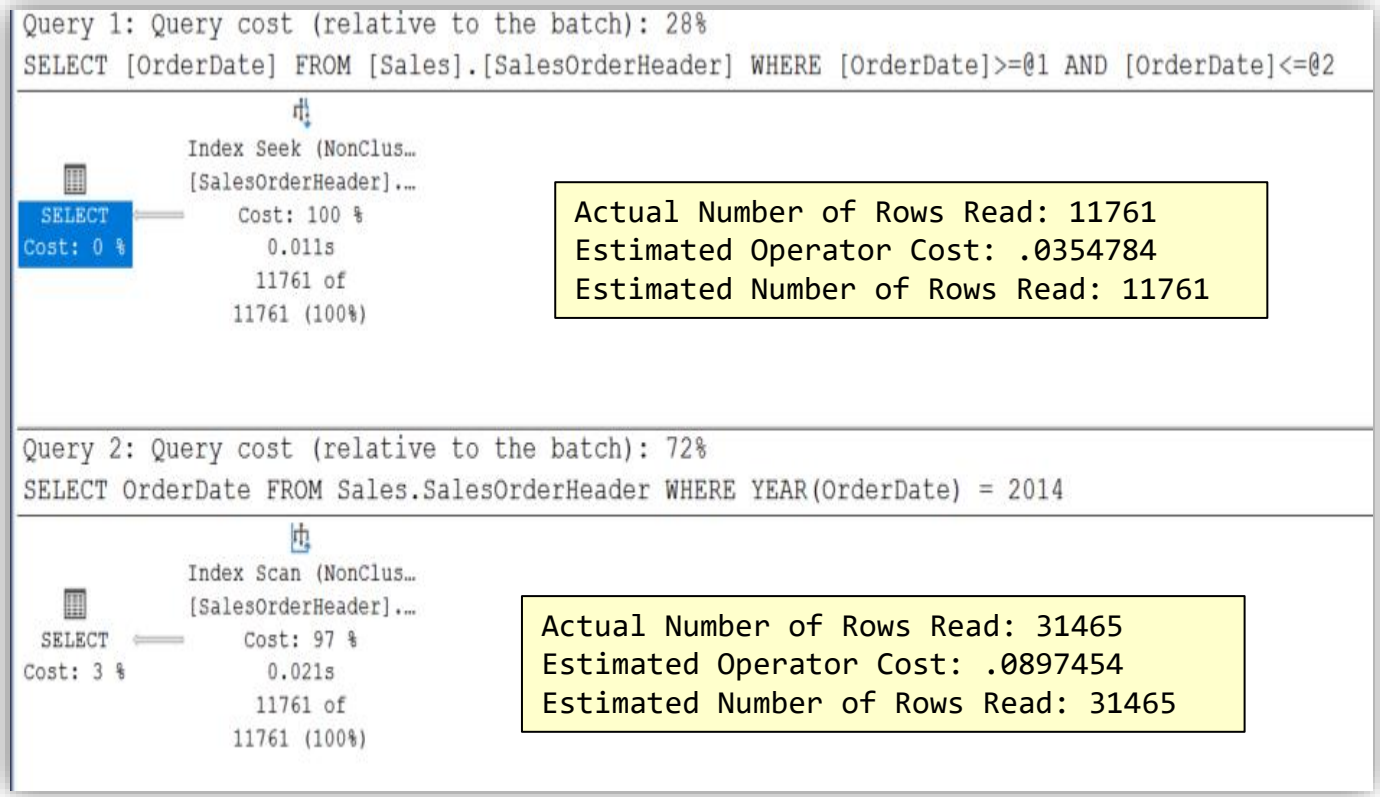
Non-SARGable Expressions – Scalar Functions (DateTime)

```sql
-- SARGable
SELECT OrderDate FROM
Sales.SalesOrderHeader
WHERE OrderDate Between '2014/01/01'
AND '2014/12/31'

-- NON-SARGable
SELECT OrderDate FROM
Sales.SalesOrderHeader
WHERE YEAR(OrderDate) = 2014
```

```
(11761 rows affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 30

(11761 rows affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 73
```

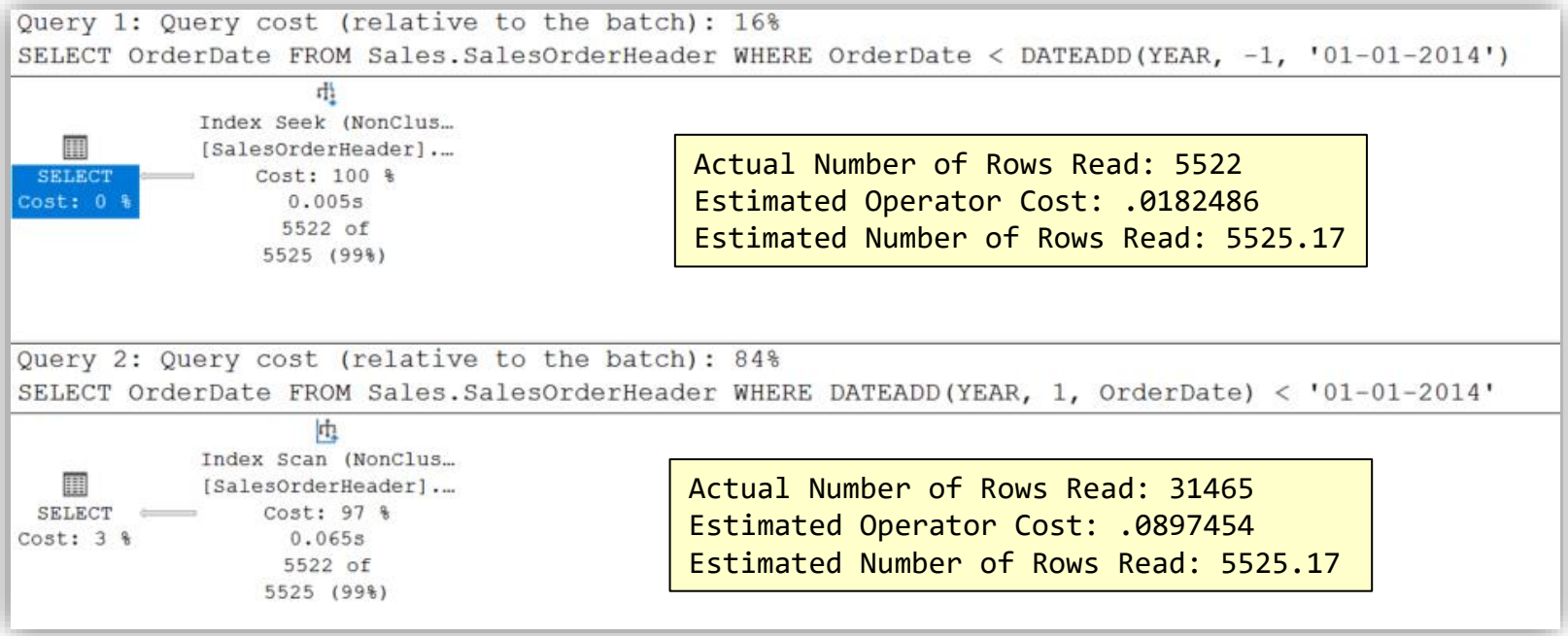


DateTime Comparisons

```sql
-- SARGable
SELECT OrderDate FROM Sales.SalesOrderHeader
WHERE OrderDate < DATEADD(YEAR, -1, '01-01-2014')

-- NON-SARGable
SELECT OrderDate FROM Sales.SalesOrderHeader
WHERE DATEADD(YEAR, 1,OrderDate) < '01-01-2014'
```

```
(5522 rows affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 15
(5522 rows affected)
Table 'SalesOrderHeader'. Scan count 1, logical reads 73
```



Demonstration

Writing SARGable Expressions

Multi-Column Searches

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)

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

```sql
--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
```

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

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 Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

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, [AdventureWorks2019].[Person].[Address].StateProvinceID = Scalar Operator(CONVERT_IMPLICIT(nvarchar(4000),[@1],0)), Scalar Operator(CONVERT_IMPLICIT(int,[@2],0))

Multi-Column Indexing Access (Scan Predicates)

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

```sql
--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

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