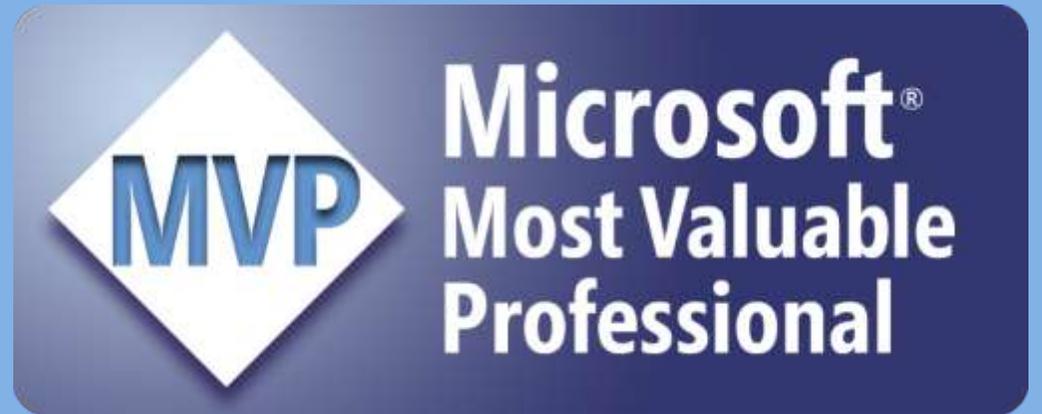


# Execution Plans: The Secret to Query Tuning Success

Jes Borland,  
Senior SQL  
Engineer



# Tech on Tap

<http://techontap.org>

Love technology and beer? So do we!

Our next event - How Cloudy is Your Organization?  
- is Saturday, May 16 at Stone Cellar Brewpub in  
Appleton

# FoxPASS

*The SQL Server community for NE WI!*

<http://fox.sqlpass.org>

We meet the first Wednesday of each month at 5:30 pm at Omni Resources (on College Ave near the mall) in Appleton

# What is an execution plan?



```
SELECT SOH.SalesOrderID,  
SOH.OrderDate, PROD.Name,  
SOD.OrderQty  
FROM Sales.SalesOrderHeader SOH  
    INNER JOIN Sales.SalesOrderDetail  
SOD ON SOD.SalesOrderID =  
SOH.SalesOrderID  
    INNER JOIN Production.Product PROD  
ON PROD.ProductID = SOD.ProductID  
ORDER BY SOH.SalesOrderID
```

## Ingredients

- 1 SalesOrderHeader Clustered Index Scan (PK\_SalesOrderHeader\_SalesOrderID)
- 1 SalesOrderDetail Clustered Index Scan (PK\_SalesOrderDetail\_SalesOrderID\_SalesOrderDetailID)
- 1 Product Nonclustered Index Scan (AK\_Product\_Name)
- 1 Hash Match
- 1 Merge
- 1 Sort
- 1 SELECT

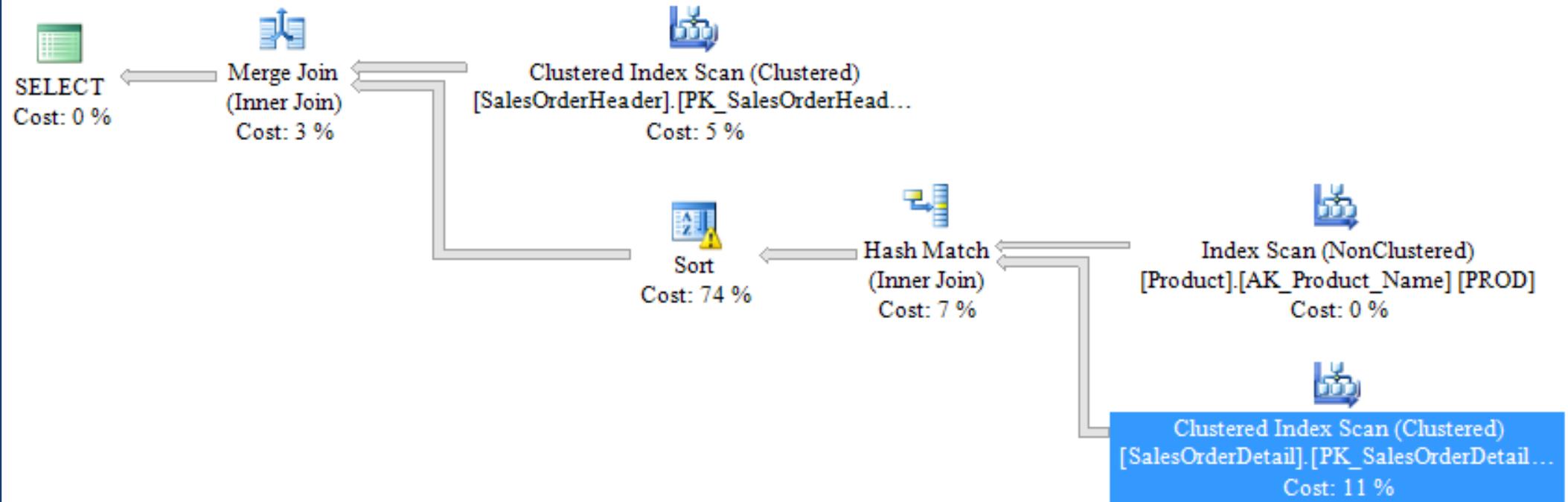
## Directions

Scan the SalesOrderHeader clustered index. Scan the Product non-clustered index. Scan the SalesOrderDetail non-clustered index. Join the output of Product and SalesOrderDetail with a Hash Match. Sort the results. Join the output of SalesOrderheader with the Hash Match output using a Merge.

Return the correct results to the SELECT operator.

Query 1: Query cost (relative to the batch): 100%

SELECT SOH.SalesOrderID, SOH.OrderDate, PROD.Name, SOD.OrderQty FROM Sales.SalesOrderHeader SOH INNER JOIN S...

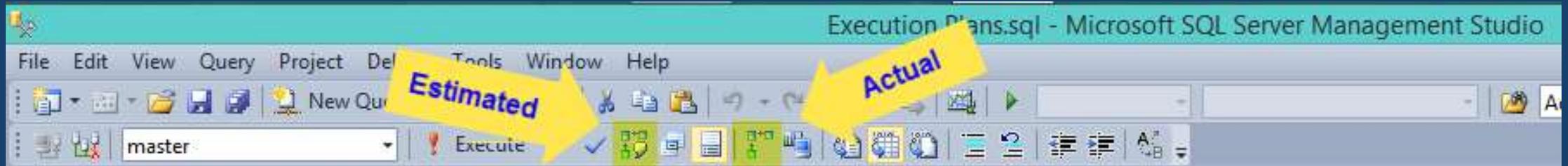


# Execution plans are generated when a query is first executed

- Generally, they get placed in memory for re-use
- The cache stores information such as CPU, memory, and I/O as well

# How you can view execution plans

- SQL Server Management Studio (SSMS)
  - For a currently executing query
    - SHOWPLAN XML – Estimated execution plan
    - Include Estimated execution plan
    - Include Actual Execution Plan
    - Use Extended Events to capture
  - For a previously-executed query
    - Plan cache



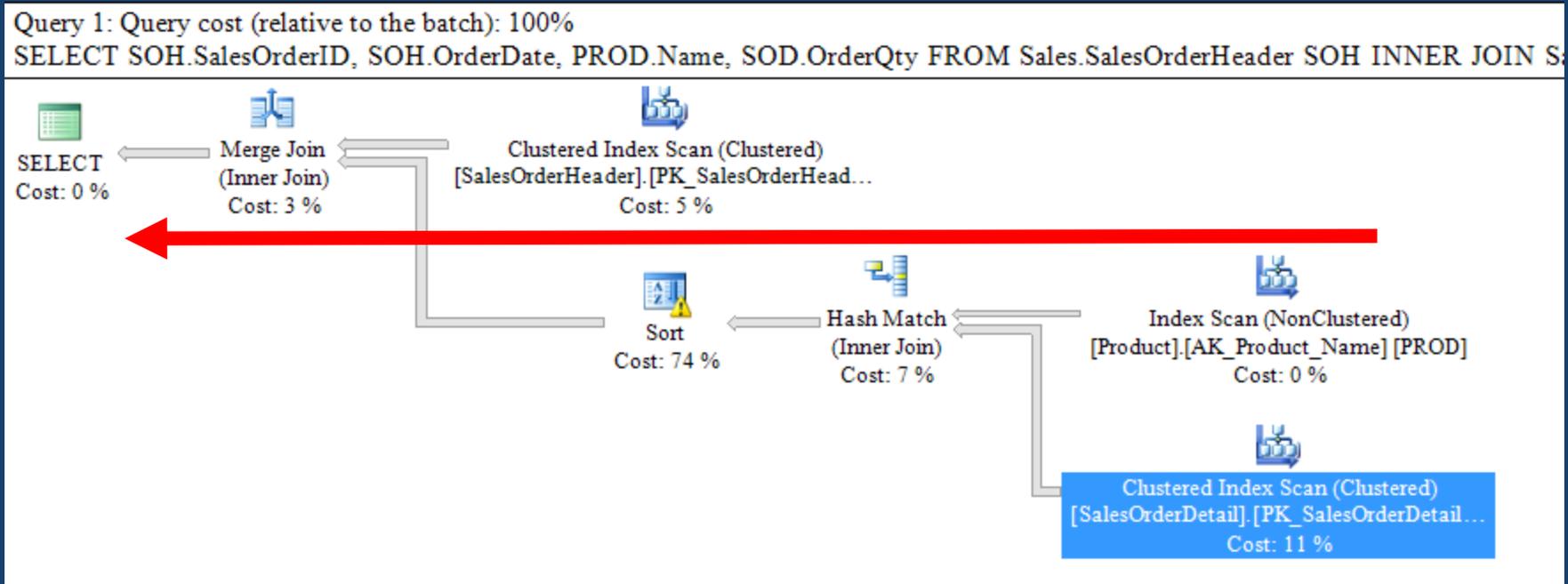
# Proper permissions required

- Permission to execute the query
- SHOWPLAN permission granted in the database
  - <http://tinyurl.com/nn3j8mk>

# How we read execution plans

- Right to left
- Hover over operators for more information
- Use Properties for even more information
- Final operator gives us some extra goodies
- Lines show us how many rows were returned

# Right to left



# Hover over operators

the batch): 100%  
SOH.OrderDate, PROD.Name, SOD.OrderQty FROM Sales.SalesOrderHeader SOH INNER

The screenshot shows a query plan in SQL Server Enterprise Manager. A tooltip is displayed over a 'Clustered Index Scan (Clustered)' operator. The tooltip provides detailed information about the scan operation, including its physical and logical operations, execution modes, storage requirements, and various cost estimates. The object being scanned is identified as [AdventureWorks2012].[Sales].[SalesOrderHeader].[PK\_SalesOrderHeader\_SalesOrderID] [SOH]. The output list shows columns: [Sales].[SalesOrderHeader].SalesOrderID, [AdventureWorks2012].[Sales].[SalesOrderHeader].OrderDate.

Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	31465
Actual Number of Batches	0
Estimated I/O Cost	0.509792
Estimated Operator Cost	0.54456 (5%)
Estimated Subtree Cost	0.54456
Estimated CPU Cost	0.0347685
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	31465
Estimated Row Size	19 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	1
<b>Object</b>	
[AdventureWorks2012].[Sales].[SalesOrderHeader].[PK_SalesOrderHeader_SalesOrderID] [SOH]	
<b>Output List</b>	
[AdventureWorks2012].[Sales].[SalesOrderHeader].SalesOrderID, [AdventureWorks2012].[Sales].[SalesOrderHeader].OrderDate	

# Use Properties

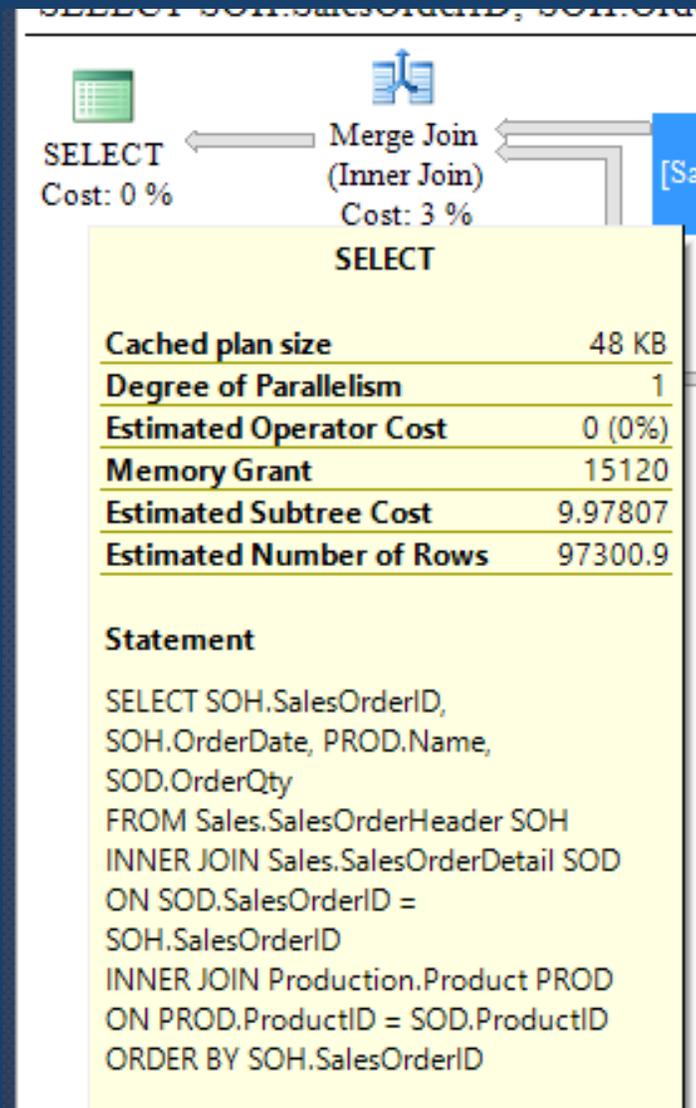
The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Execution plan' tab is active, showing a query plan for a query involving SalesOrderHeader, Product, and SalesOrderDetail tables. The plan includes a 'Clustered Index Scan (Clustered)' operator (highlighted in blue), a 'Hash Match (Inner Join)' operator, an 'Index Scan (NonClustered)' operator, and another 'Clustered Index Scan (Clustered)' operator. A red arrow points from the highlighted operator to the 'Properties' window on the right.

The 'Properties' window shows the following details for the 'Clustered Index Scan (Clustered)' operator:

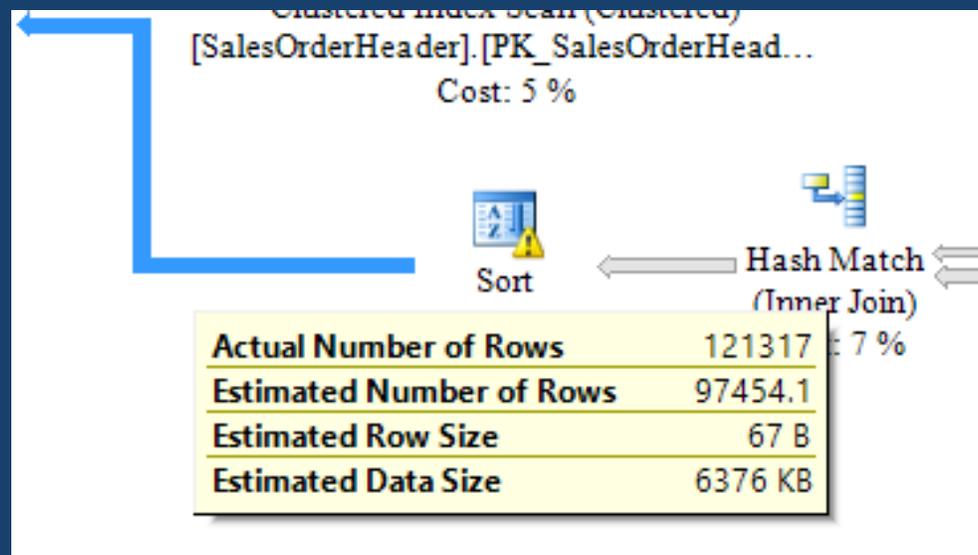
Clustered Index Scan (Clustered)	
Misc	
Actual Execution Mode	Row
Actual Number of Batches	0
Actual Number of Rows	31465
Actual Rebinds	0
Actual Rewinds	0
Defined Values	[AdventureWorks2012].[Sales].[SalesC...
Description	Scanning a clustered index, entirely o...
Estimated CPU Cost	0.0347685
Estimated Execution Mode	Row
Estimated I/O Cost	0.509792
Estimated Number of Executions	1
Estimated Number of Rows	31465
Estimated Operator Cost	0.54456 (5%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	19 B
Estimated Subtree Cost	0.54456
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Clustered Index Scan
Node ID	1
NoExpandHint	False

Actual Execution Mode  
Actual Execution Mode

# Final operator



# Read ~~between~~ the lines

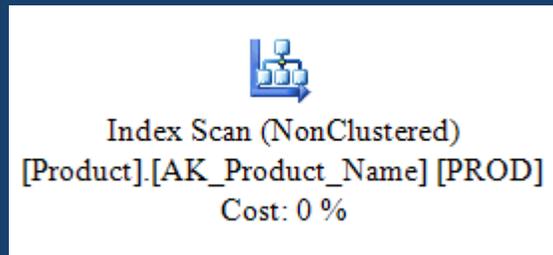
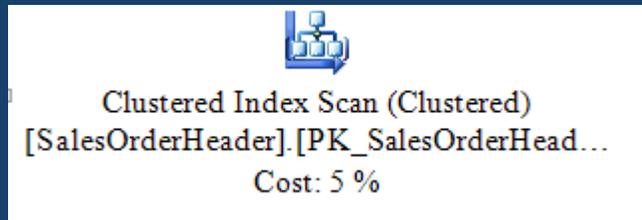
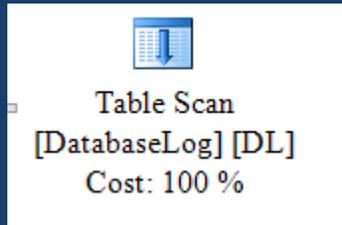


Let's take a look



# Common operators

# The Scan



- Reading all of the rows in a table or index
- If the query has a WHERE clause against that table or index, only the rows matching the predicate are *returned*
- *Can be expensive*

# The Seek



Clustered Index Seek (Clustered)  
[SalesOrderHeader].[PK\_SalesOrderHead...  
Cost: 27 %



Index Seek (NonClustered)  
[SalesOrderDetail].[IX\_SalesOrderDetail\_...  
Cost: 1 %

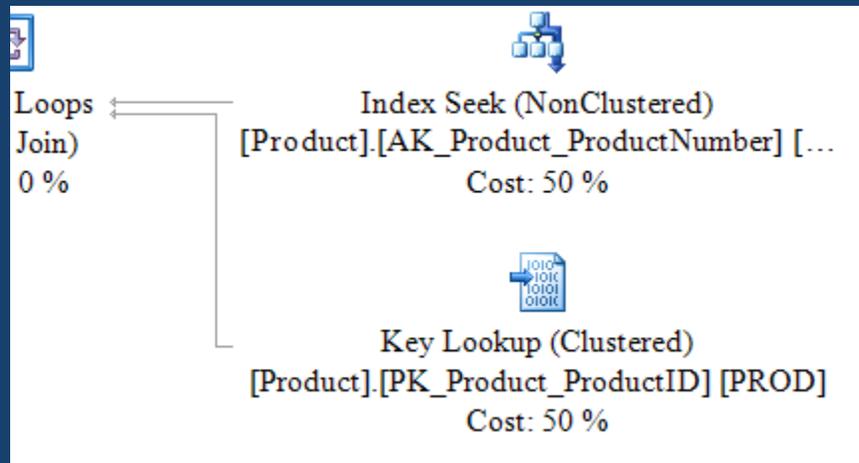
- Reads only a portion of the clustered or non-clustered index
- This is determined by the “seek predicate”

# Seek predicates

```
SELECT SOH.SalesOrderID,  
SOH.OrderDate, PROD.Name  
FROM Sales.SalesOrderHeader SOH  
INNER JOIN Sales.SalesOrderDetail  
SOD ON SOD.SalesOrderID =  
SOH.SalesOrderID  
INNER JOIN Production.Product PROD  
ON PROD.ProductID = SOD.ProductID  
WHERE SOD.ProductID = 764
```

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
Actual Number of Rows	440
Actual Number of Batches	0
Estimated Operator Cost	0.003766 (1%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.000641
Estimated Subtree Cost	0.003766
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	440
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4
<b>Object</b>	
[AdventureWorks2012].[Sales].[SalesOrderDetail]. [IX_SalesOrderDetail_ProductID] [SOD]	
<b>Output List</b>	
[AdventureWorks2012].[Sales]. [SalesOrderDetail].SalesOrderID	
<b>Seek Predicates</b>	
Seek Keys[1]: Prefix: [AdventureWorks2012].[Sales]. [SalesOrderDetail].ProductID = Scalar Operator((764))	

# The Lookup



- The WHERE clause can be evaluated using a non-clustered index
- Other columns returned in other statements (SELECT) are not in the non-clustered index structure
- SQL Server uses the clustered index key stored on the non-clustered index page to get the other columns from the clustered index

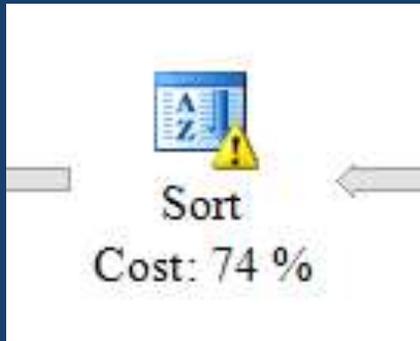
# Lookup

```
SELECT PROD.Name, PROD.ListPrice  
FROM Production.Product PROD  
WHERE PROD.ProductNumber='FR-R92R-  
44'
```

The screenshot displays the execution plan for the query. The 'Key Lookup (Clustered)' operator is highlighted with a red box. Below it, a table of statistics is shown, with the 'Output List' section also highlighted in red. The 'Output List' shows the columns 'Name' and 'ListPrice' from the 'Production.Product' table. The 'Seek Keys' section at the bottom shows the predicate: '([AdventureWorks2012].[Production].[Product].ProductID = Scalar Operator ([AdventureWorks2012].[Production].[Product].ProductID] as [PROD].[ProductID])'.

Key Lookup (Clustered)	
Uses a supplied clustering key to lookup on a table that has a clustered index.	
Physical Operation	Key Lookup
Logical Operation	Key Lookup
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	1
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032831 (50%)
Estimated CPU Cost	0.0001581
Estimated Subtree Cost	0.0032831
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	69 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3
<b>Object</b>	
[AdventureWorks2012].[Production].[Product]. [PK_Product_ProductID].[PROD]	
<b>Output List</b>	
[AdventureWorks2012].[Production].[Product].Name, [AdventureWorks2012].[Production].[Product].ListPrice	
<b>Seek Keys</b>	
Seek Keys[1]: Prefix: [AdventureWorks2012]. [Production].[Product].ProductID = Scalar Operator ([AdventureWorks2012].[Production].[Product]. ProductID] as [PROD].[ProductID])	

# The Sort



- Data needs to be sorted due to an ORDER BY or GROUP BY
- Typically very expensive!

# Joins

# The Logical Join

- INNER JOIN
- OUTER JOIN
- LEFT JOIN
- RIGHT JOIN

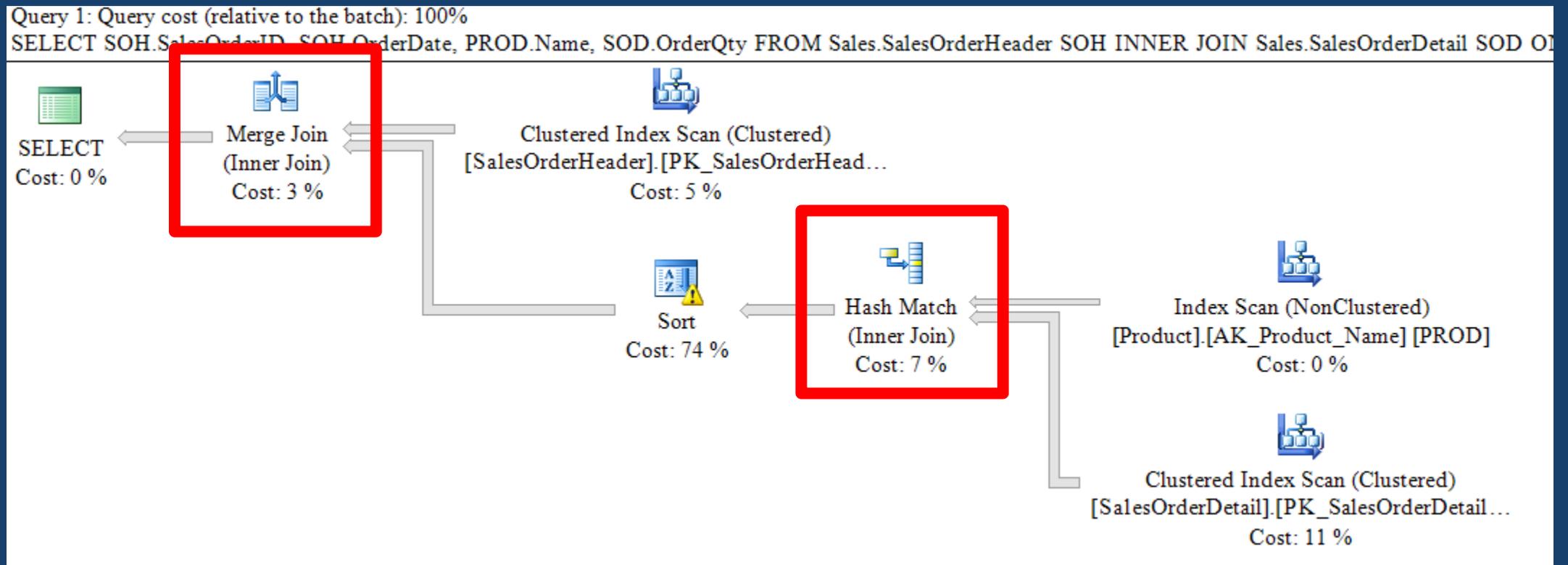
# INNER JOIN

```
SELECT SOH.SalesOrderID, SOH.OrderDate,  
PROD.Name, SOD.OrderQty  
FROM Sales.SalesOrderHeader SOH  
INNER JOIN Sales.SalesOrderDetail SOD ON  
    SOD.SalesOrderID = SOH.SalesOrderID  
INNER JOIN Production.Product PROD ON  
PROD.ProductID = SOD.ProductID  
ORDER BY SOH.SalesOrderID
```

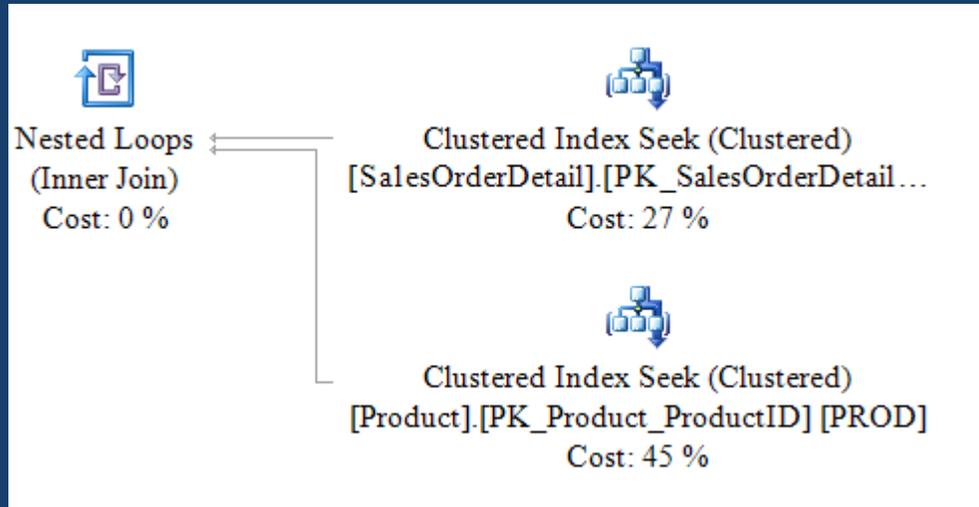
# The Physical Join

- Nested loops
- Merge
- Hash

# This is the execution plan of the INNER JOIN query



# Nested Loops



- Best for small data sets
- Outer (top) and inner (bottom) inputs
- Compare every row in inner loop to outer loop



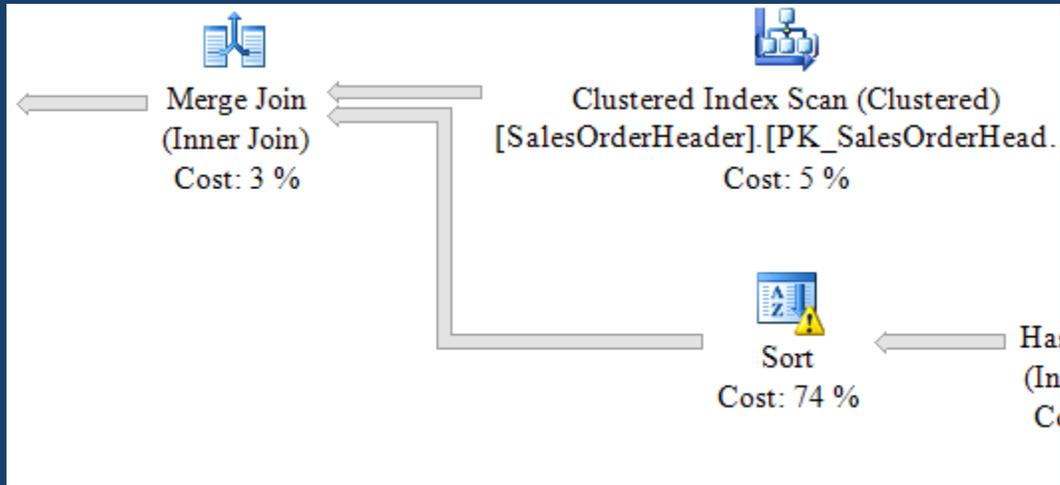
Photo: Aka Hige <https://www.flickr.com/photos/akahige/1044518492/in/photolist-2AiqXw-8peEYs-gsUJq-53AgoC-8t1YnK-28zraz-bWzyjD-bWzuYH-9MkCBU-9MhP5a-9MkCd7-oDUaRR-oWjgSF-oDQ246-oEifhQ-oWL6a3-oWL5xb-c7UTKL-2tGgh-oDPa24-oUuvGC-oWotJi-oW7NLe-oW7N2D-pNzVr2-p9gsCk-q5QCex-q69h1G-p9dL6o-pNBVMs-q698ud-pNzGsg-q5Qobx-q5ZYx4-pNzBoP-oWmA3m-oDQ3sZ-oE2itv-oDPWbS-oDPRQC-oUhfCd-oW3twR-oEjmrT-oEj98b-oULtAm-oWwPHB-oEiFSY-oWLxA9-oEiE35-oEiiMK/>

sunshinecity <https://www.flickr.com/photos/sunshinecity/455880294/in/photolist-Ghvk5-8hWCYQ-siZBM-5LexoC-3bQ9o-qzXsi-w6vki-5LevK5-5Lagb2-75F9J-5Lagwv-5Lestu-5LevpN-8xYKsp-5LabCT-bzC3Dx-6j3DXv-6fJnht-8xYKL6-8y2NoW-5WVfPh-51CWGy-6vm3Mg-5Lerd5-55py8R-6gV2i5-5LerZU-6PmiNj-vr6uX-czhtWQ-7r91yj-5V59yU-siZBN-4RzTQj-bijRqT-dJ85Xp-6iSfjp-zBuHk-aEvev1-akQJb2-6iWpy7-5LewWj-5Lev3j-5LaeKn-5Leuju-5LetQS-5LetuU-5Lad9x-5LesLC-9rbGRt>



ALL RIGHTS RESERVED © 2015

# Merge



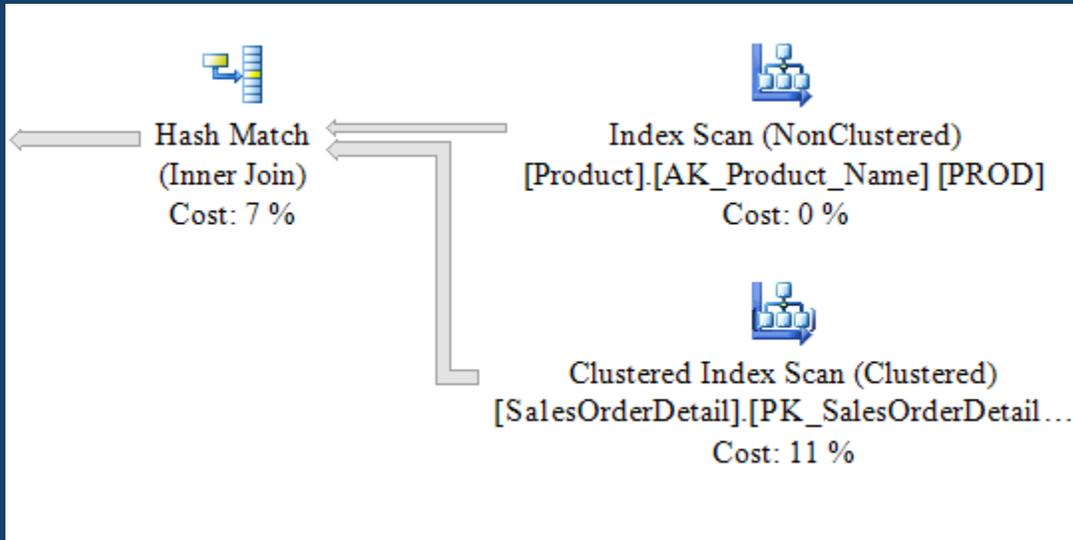
- Both inputs must be sorted in the same order
- Each row is compared



Photos: Becka Spence <https://www.flickr.com/photos/nevrlndtink/1920763302/in/photolist-7JHijY-6qbipm-6qbid7-6qbhQw-6qbhAd-6q798t-3VJq1E-85i1X5-nGdiLA-iTgJrA-iTeFbc-pk979P-caMKg5-bYXuxd-bpJYsn-5LYb4h-aJZj32-c93xaQ-c93wXj-du1XBt-q3HPiz-2Air87-2Air49-2AiqXw-8peEYs-gsUJq-53AgoC-8t1YnK-28zraz-bWzyjD-bWzuYH-9MkCBU-9MhP5a-9MkCd7-asMJZp-oWjePn-oWjfv2-oDUaRR-oWjgSF-oDQ246-oEifhQ-oWL6a3-oWL5xb-c7UTKL-2tGgh-oDPa24-oUuvGC-oWotli-oW7NLe-oW7N2D>

Micky\*\* <https://www.flickr.com/photos/emzee/278221145/in/photolist-qzXsi-w6vki-5LevK5-5Lagb2-75F9J-5Lagww-5Lestu-5LevpN-8xYKsp-5LabCT-bzC3Dx-6j3DXv-6fJnht-8xYKL6-8y2NoW-5WVfPpH-51CWGy-6vm3Mg-5Lerd5-55py8R-6gV2i5-5LerZU-6PmiNj-vr6uX-czhtWQ-7r91yj-5V59yU-siZBN-4RzTQj-bjJRqT-dJ85Xp-6iSjpo-zBuHk-aEvev1-akQlb2-6iWpy7-5LewWj-5Lev3j-5LaeKn-5Leuju-5LetQS-5LetuU-5Lad9x-5LesLC-9rbGRT-6KAFky-p3gag6-6wwjpw-7oDn6M-5czFkp>

# Hash



- Best for large data sets
- Hash tables are built in memory – build and probe
- Values from probe compared to build



Photos: Justin Ennis <https://www.flickr.com/photos/averain/4039312969/in/photolist-79Wx24-6eL9Qd-tF4Y-5zPpQL-bYXxwY-au4JnJ-2Ae4vR-qZmJ9g-8derqL-qZvKzt-bUUh5i-7JHijY-6qbipm-6qbid7-6qbhQw-6qbhAd-6q798t-3VJq1E-85i1X5-nGdiLA-iTgJrA-iTeFbc-pk979P-caMKg5-bYXuxd-bpJYsn-5LYb4h-ajZj32-c93xaQ-c93wXj-du1XBt-q3HPiz-2Air87-2Air49-2AiXw-8peEYs-gsUJq-53AgoC-8t1YnK-28zraz-bWzyjD-bWzuYH-9MkCBU-9MhP5a-9MkCd7-asMJZp-oWjePn-oWjfv2-oDUaRR-oWjgSF>

HISHAM BINSUWAIF <https://www.flickr.com/photos/4444/340569115/in/photolist-w6vki-9tVrew-5LevK5-5Lagb2-75F9J-5Lagwv-5Lestu-5LevpN-8xYKsp-5LabCT-bzC3Dx-6j3DXv-6fJnht-8xYKL6-8y2NoW-5WVFpH-51CWGy-6vm3Mg-5Lerd5-55py8R-6gV2i5-5LerZU-6PmiNj-vr6uX-czhtWQ-7r91yj-5V59yU-sizBN-4RzTQj-bijRqT-7kyAtD-dJ85Xp-6iSfjp-zBuHk-aEew1-akQJb2-6iWpy7-5LewWj-5Lev3j-5LaeKn-5Leuju-5LetQS-5LetuU-5Lad9x-5LesLC-9rbGRt-6KAFkY-p3gag6-6wvjpw-7oDn6M>

Execution plans are trying to help you!



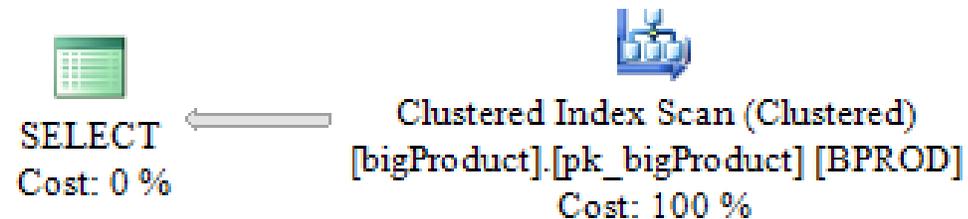
# Missing indexes

```
SELECT Name, Color, Size, Weight  
FROM bigProduct BPROD  
WHERE Class = 'L'
```

Query 1: Query cost (relative to the batch): 100%

```
SELECT [Name],[Color],[Size],[Weight] FROM [bigProduct] [BPROD] WHERE [Class]=@1
```

```
Missing Index (Impact 81.8037): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
```

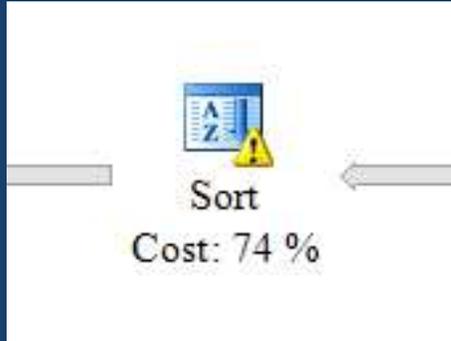


```
/*  
Missing Index Details from Execution Plans.sql  
The Query Processor estimates that implementing the following index could improve the query  
cost by 81.8037%.  
*/
```

```
/*  
USE [AdventureWorks2012]  
GO  
CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]  
ON [dbo].[bigProduct] ([Class])  

```

# Warnings



Sort	
Sort the input.	
	Physical Operation Sort
	Logical Operation Sort
	Actual Execution Mode Row
	Estimated Execution Mode Row
Cost:	Actual Number of Rows 121317
	Actual Number of Batches 0
	Estimated Operator Cost 7.42445 (74%)
	Estimated I/O Cost 0.0112613
	Estimated CPU Cost 7.41318
So:	Estimated Subtree Cost 9.14745
Cost:	Estimated Number of Executions 1
	Number of Executions 1
	Estimated Number of Rows 97454.1
	Estimated Row Size 67 B
	Actual Rebinds 1
	Actual Rewinds 0
	Node ID 2
<b>Output List</b>	
[AdventureWorks2012].[Sales].	
[SalesOrderDetail].SalesOrderID,	
[AdventureWorks2012].[Sales].	
[SalesOrderDetail].OrderQty, [AdventureWorks2012].	
[Production].[Product].Name	
<b>Warnings</b>	
Operator used tempdb to spill data during execution with spill level 1	
<b>Order By</b>	
[AdventureWorks2012].[Sales].	
[SalesOrderDetail].SalesOrderID Ascending	

# Missing Index DMVs

- There are four missing index DMVs
- Combine them to find missing index requests & impact
- <http://blog.sqlauthority.com/2011/01/03/sql-server-2008-missing-index-script-download/>

What the execution plan  
*won't* tell you



# Multiple missing indexes

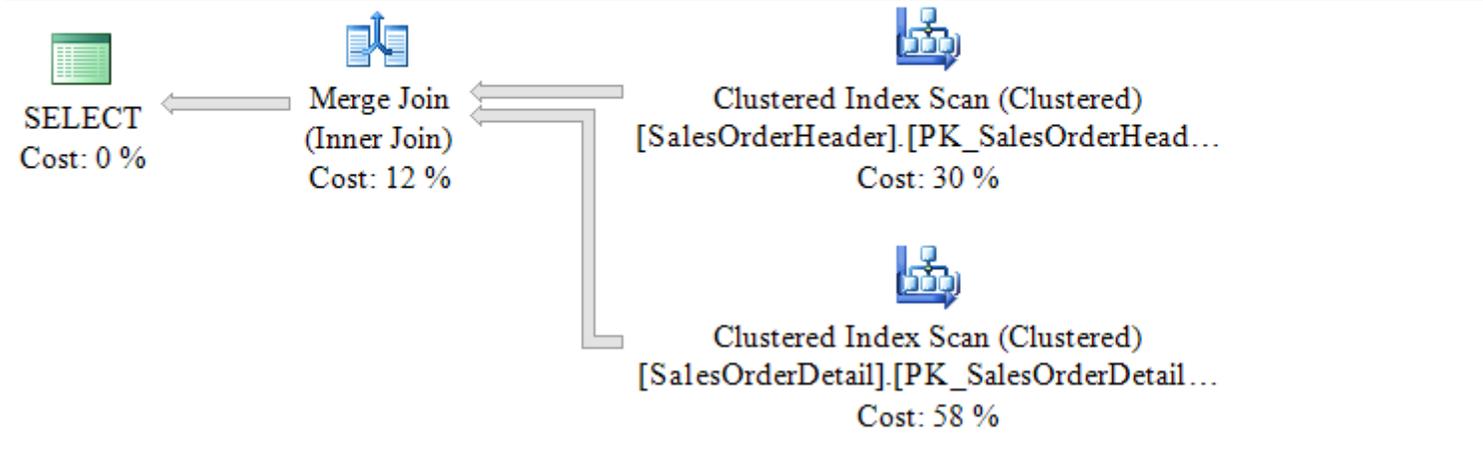
# A query with two predicates

```
SELECT SOH.SalesOrderID,  
SOD.SalesOrderDetailID, SOH.Status,  
SOD.UnitPrice  
FROM Sales.SalesOrderHeader SOH  
INNER JOIN Sales.SalesOrderDetail SOD ON  
SOD.SalesOrderID = SOH.SalesOrderID  
WHERE SOH.Status = 5  
      AND SOD.UnitPrice > 500
```

Query 1: Query cost (relative to the batch): 100%

SELECT SOH.SalesOrderID, SOD.SalesOrderDetailID, SOH.Status, SOD.UnitPrice FROM Sales.SalesOrderHeader

Missing Index (Impact 55.3614): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]



```
/*  
Missing Index Details from Execution Plans.sql  
The Query Processor estimates that implementing the following index could improve the query cost by  
55.3614%.  
*/  
  
/*  
USE [AdventureWorks2012]  
GO  
CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]  
ON [Sales].[SalesOrderDetail] ([UnitPrice])  
INCLUDE ([SalesOrderID],[SalesOrderDetailID])  
GO  
*/
```

# Dig into the XML...

```
9      <MissingIndexes>
10      <MissingIndexGroup Impact="55.3614">
11      <MissingIndex Database="[AdventureWorks2012]" Schema="[Sales]"
12      Table="[SalesOrderDetail]">
13      <ColumnGroup Usage="INEQUALITY">
14      <Column Name="[UnitPrice]" ColumnId="7" />
15      </ColumnGroup>
16      <ColumnGroup Usage="INCLUDE">
17      <Column Name="[SalesOrderID]" ColumnId="1" />
18      <Column Name="[SalesOrderDetailID]" ColumnId="2" />
19      </ColumnGroup>
20      </MissingIndex>
21      </MissingIndexGroup>
22      <MissingIndexGroup Impact="28.3467">
23      <MissingIndex Database="[AdventureWorks2012]" Schema="[Sales]"
24      Table="[SalesOrderHeader]">
25      <ColumnGroup Usage="EQUALITY">
26      <Column Name="[Status]" ColumnId="6" />
27      </ColumnGroup>
28      <ColumnGroup Usage="INCLUDE">
29      <Column Name="[SalesOrderID]" ColumnId="1" />
30      </ColumnGroup>
31      </MissingIndex>
32      </MissingIndexGroup>
33      </MissingIndexes>
```

# Solutions

- Use the missing index DMVs to find all missing indexes
- Review the XML
- Use a (free!) tool called SQL Sentry Plan Explorer to view the execution plan

tmpA603.xml - LENOVO...2 (LENOVOT440P\Jes)\* x

Command Text Results

Text Data

```
SELECT SOH.SalesOrderID, SOD.SalesOrderDe
FROM Sales.SalesOrderHeader SOH
INNER JOIN Sales.SalesOrderDetail SOD ON
```

Text Data Plan XML Plan/Query Info

Plan Diagram

```

graph TD
    S(SELECT) -- 33,264 --> MJ(Merge Join Inner Join)
    CIH(Clustered Index Scan SalesOrderHeader) -- 30,490 --> MJ
    MJ -- 33,264 --> CID(Clustered Index Scan SalesOrderDetail)
  
```

### Missing Indexes

```

/*
Missing Index Details from C:\Users\Jes\AppData\Local\Temp\planexplorer\session\b41e716b-1ce5-4490-a3ac-42c195923a05\
The Query Processor estimates that implementing the following index could improve the query cost by 55.3614%.

WARNING: This is only an estimate, and the Query Processor is making this recommendation based solely upon analysis
of this specific query. It has not considered the resulting index size, or its workload-wide impact, including its
impact on INSERT, UPDATE, DELETE performance. These factors should be taken into account before creating this index.
*/
USE [AdventureWorks2012]
GO
CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
ON [Sales].[SalesOrderDetail] ([UnitPrice])
INCLUDE ([SalesOrderID],[SalesOrderDetailID])
GO
-----

/*
Missing Index Details from C:\Users\Jes\AppData\Local\Temp\planexplorer\session\b41e716b-1ce5-4490-a3ac-42c195923a05\
The Query Processor estimates that implementing the following index could improve the query cost by 28.3467%.

WARNING: This is only an estimate, and the Query Processor is making this recommendation based solely upon analysis
of this specific query. It has not considered the resulting index size, or its workload-wide impact, including its
impact on INSERT, UPDATE, DELETE performance. These factors should be taken into account before creating this index.
*/
USE [AdventureWorks2012]
GO
CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
ON [Sales].[SalesOrderHeader] ([Status])
INCLUDE ([SalesOrderID])
GO

```

How many indexes are affected?

# INSERTS, UPDATES, DELETES

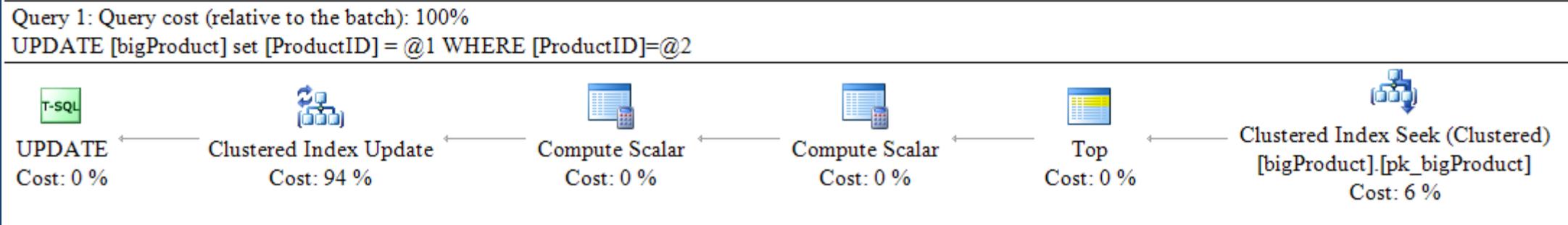
Clustered index

Non-clustered indexes

Column Name	Column Type
ProductID (PK)	int, not null
Name	nvarchar(80), null
ProductNumber	nvarchar(56), null
MakeFlag	Flag(bit), not null
FinishedGoodsFlag	Flag(bit), not null
Color	nvarchar(15), null
SafetyStockLevel	smallint, not null
ReorderPoint	smallint, not null
StandardCost	money, not null
ListPrice	money, not null
Size	nvarchar(5), null
SizeUnitMeasureCode	nchar(3), null
WeightUnitMeasureCode	nchar(3), null
Weight	decimal(8,2), null
DaysToManufacture	int, not null
ProductLine	nchar(2), null
Class	nchar(2), null
Style	nchar(2), null
ProductSubcategoryID	int, null
ProductModelID	int, null
SellStartDate	datetime, not null
SellEndDate	datetime, null
DiscontinuedDate	datetime, null

# A simple UPDATE

```
UPDATE bigProduct
SET ProductID = 1005
WHERE ProductID = 1004
```



# Look at Properties of the Clustered Index Update

SQLQuery6.sql Execution Plans.sql\* x

Editor Messages Execution plan

Query 1: Query cost (relative to the batch): 100%  
UPDATE [bigProduct] set [ProductID] = @1 WHERE [ProductID]=@2

T-SQL  
UPDATE  
Cost: 0 %

Clustered Index Update  
Cost: 94 %

Compute Scalar  
Cost: 0 %

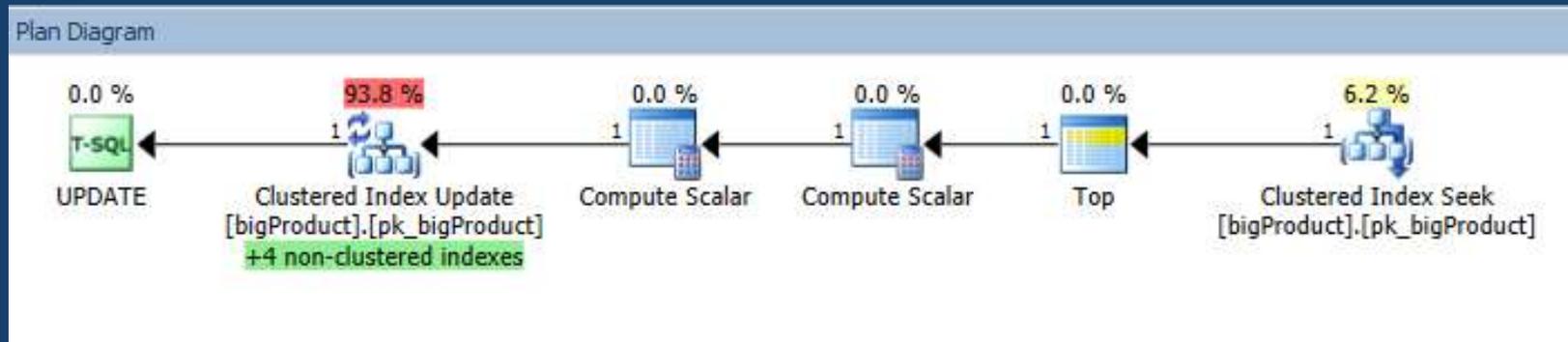
Compute Scalar  
Cost: 0 %

Properties

### Clustered Index Update

DMLRequestSort	False
Estimated CPU Cost	0.000005
Estimated Execution Mode	Row
Estimated I/O Cost	0.05
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Operator Cost	0.050005 (94%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	9 B
Estimated Subtree Cost	0.0532883
Logical Operation	Update
Node ID	1
Number of Executions	1
Object	[AdventureWorks2012].[dbo].[bigProduct].[pk_bigProduct], [AdventureWorks2012].[dbo].[bigProduct].[pk_bigProduct]
▶ [1]	[AdventureWorks2012].[dbo].[bigProduct].[pk_bigProduct]
▶ [2]	[AdventureWorks2012].[dbo].[bigProduct].[IX_bigProduct_Color]
▶ [3]	[AdventureWorks2012].[dbo].[bigProduct].[IX_bigProduct_Name]
▶ [4]	[AdventureWorks2012].[dbo].[bigProduct].[IX_bigProduct_Size]
▶ [5]	[AdventureWorks2012].[dbo].[bigProduct].[IX_bigProduct_Weight]
▶ Output List	

# In SQL Sentry Plan Explorer



# Resources



ALL RIGHTS RESERVED © 2015

# SQL Sentry Plan Explorer

- <http://sqlsentry.net>
- FREE! (But check out the Pro version - extra awesome!)

# Books and videos

- Don't Fear the Execution Plan  
[https://www.youtube.com/watch?v=l-jjgZ51\\_Sw](https://www.youtube.com/watch?v=l-jjgZ51_Sw)
- Spotting Trouble – and Help! – in Execution Plans  
<https://www.youtube.com/watch?v=9lwpULbFK4E>
- What the Execution Plan Doesn't Tell You  
<https://www.youtube.com/watch?v=mbaLmVEmXbl>
- SQL Server Execution Plans, Second Edition, by Grant Fritchey  
<https://www.simple-talk.com/books/sql-books/sql-server-execution-plans,-second-edition,-by-grant-fritchey/>
- Can You Dig It? Plan Cache Series by Jason Strate  
<http://www.jasonstrate.com/2011/02/can-you-dig-it-plan-cache-series/>

# Contact me

- [jborland@concurrency.com](mailto:jborland@concurrency.com)
- @grrl\_geek
- <http://lessthandot.com>