Introduction to Execution Plans

Kevan Riley

#372 | EXETER 2015

Who is this guy?

- Kevan Riley
  - ✉ kevan.riley@rileywaterhouse.co.uk
  - 🐦 @kevriley
- Independent freelance SQL Server
- 15+ years SQL Server
- Moderator at ask.sqlservercentral.com

Microsoft CERTIFIED Solutions Expert
Data Platform

? ASK SQL Server Central

redgate
Friend of Redgate 2015

Agenda

- What are Execution Plans?
- Why are they important?
- How to view an Execution Plan
- How to read Graphical Execution Plans
- Execution Plan Operators
- Performance Tuning

Execution Plans

- Query optimizer
  - Cost based optimizer
  - Cardinality Estimator
  - Generate execution plans
  - Evaluate least cost based on Statistics
  - Estimated execution plan -> Plan cache
- Storage engine -> Actual execution plan

Execution Plans

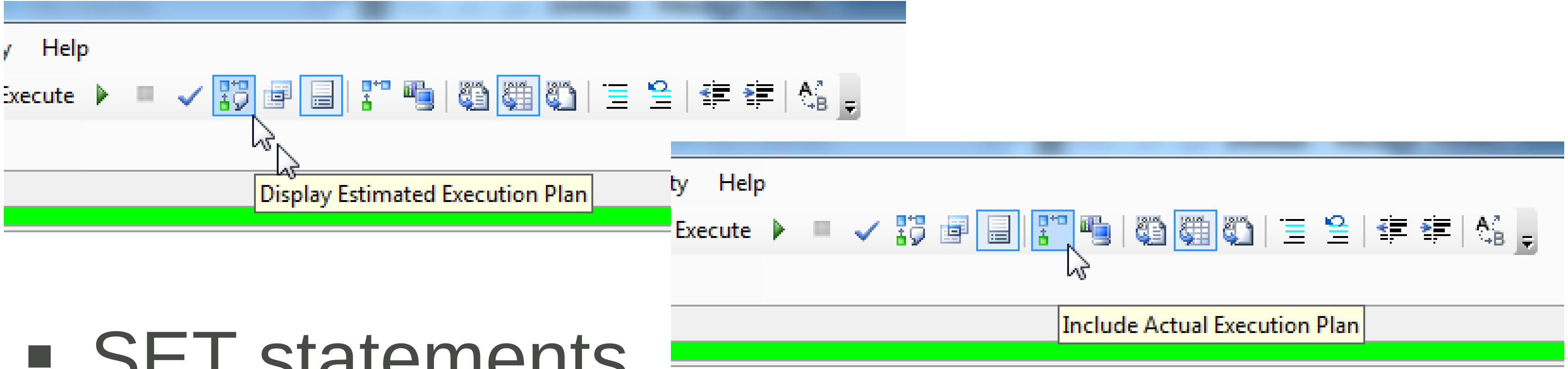
- Optimizer finds ‘best’ plan = least cost plan in the shortest time
- Plan cache enables plan reuse
    - Performance gain
- Plan cache aged out / cleared

Statistics

- Data about your data
- Statistics on columns and indexes
  - Selectivity
  - Uniqueness
  - Distribution
- Automatically created and maintained by default
- Dave Morrison – “Statistics, Estimation & Plan Caching - The Big Three” next session in this room!

View Execution Plans

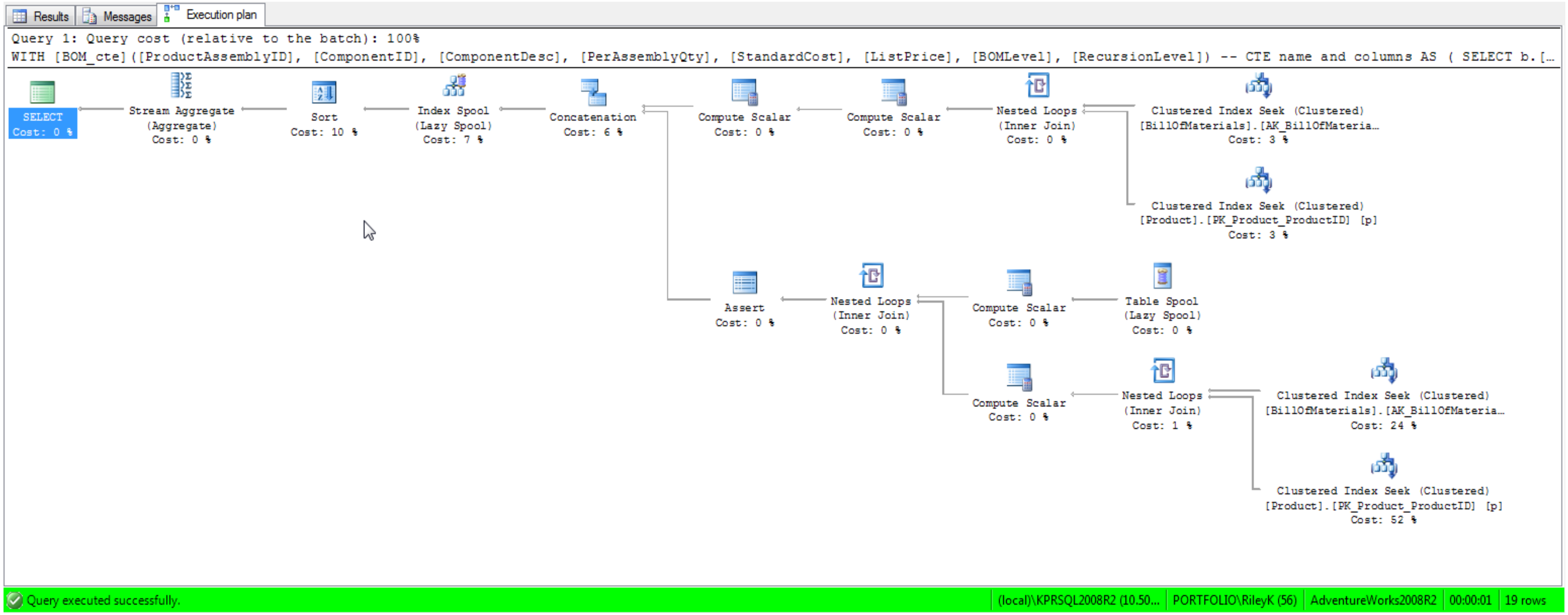
- SQL Server Management Studio (SSMS)


Help
Execute
Display Estimated Execution Plan
Help
Execute
Include Actual Execution Plan

- SET statements
- DMVs (plan cache)
- Extended Events (SQL2008+ only)
- SQL Profiler

View Execution Plans - Graphical

- Estimated
  - Ctrl-L
  - Menu bar
- Actual
  - Ctrl-M
  - Menu Bar



Results
Messages
Execution plan

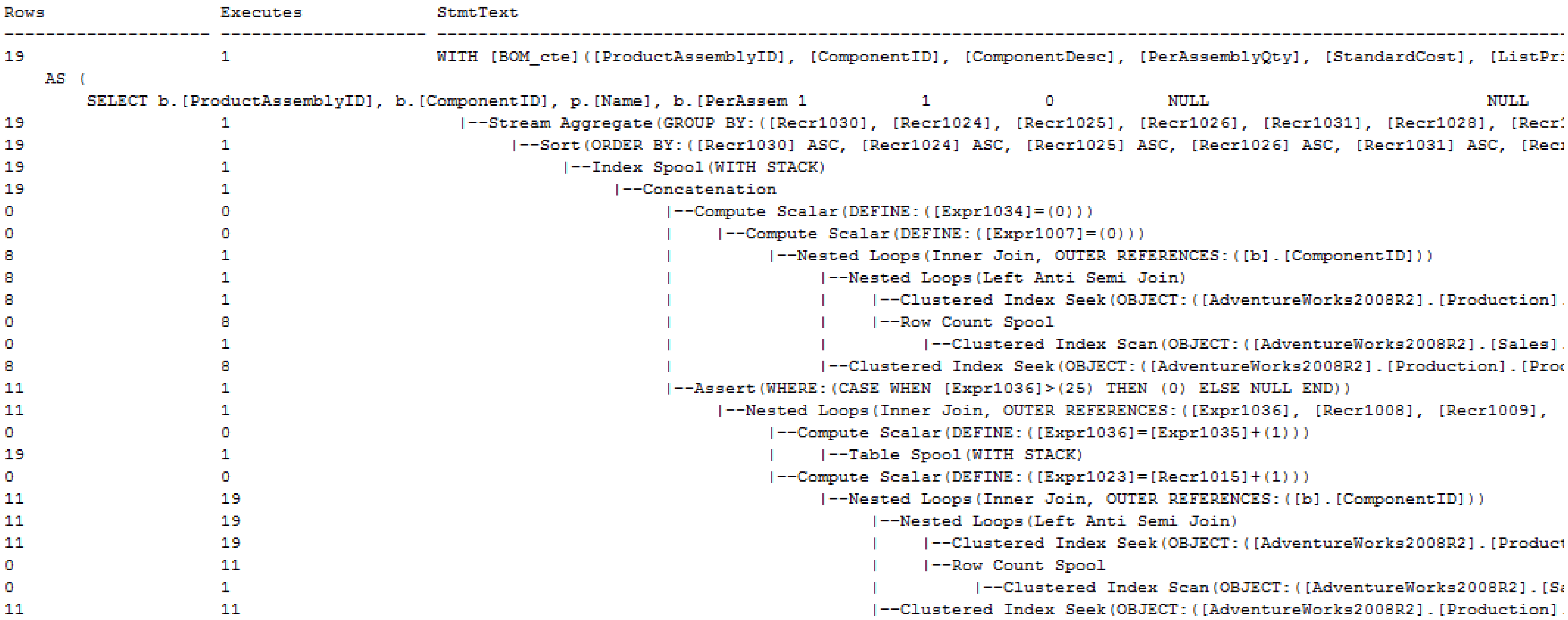
```
Query 1: Query cost (relative to the batch): 100%
WITH [BOM_cte]([ProductAssemblyID], [ComponentID], [ComponentDesc], [PerAssemblyQty], [StandardCost], [ListPrice], [BOMLevel], [RecursionLevel]) -- CTE name and columns AS ( SELECT b.[…
```

SELECT Cost: 0 %
Stream Aggregate (Aggregate) Cost: 0 %
Sort Cost: 10 %
Index Spool (Lazy Spool) Cost: 7 %
Concatenation Cost: 6 %
Compute Scalar Cost: 0 %
Compute Scalar Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Clustered Index Seek (Clustered) [BillOfMaterials].[AK_BillOfMateria… Cost: 3 %
Clustered Index Seek (Clustered) [Product].[PK_Product_ProductID] [p] Cost: 3 %
Assert Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Table Spool (Lazy Spool) Cost: 0 %
Compute Scalar Cost: 0 %
Nested Loops (Inner Join) Cost: 1 %
Clustered Index Seek (Clustered) [BillOfMaterials].[AK_BillOfMateria… Cost: 24 %
Clustered Index Seek (Clustered) [Product].[PK_Product_ProductID] [p] Cost: 52 %

Query executed successfully. | (local)\KPRSQL2008R2 (10.50... | PORTFOLIO\RileyK (56) | AdventureWorks2008R2 | 00:00:01 | 19 rows

View Execution Plans - Text

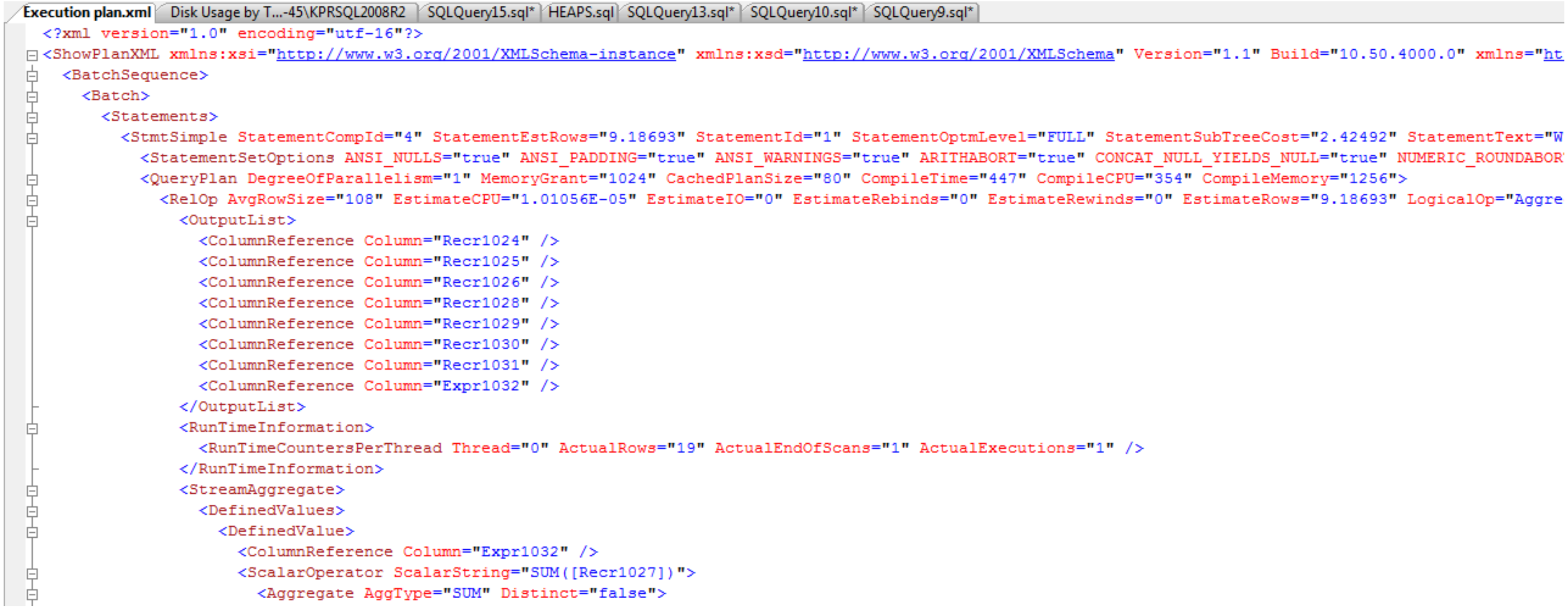
- Estimated
  - SET SHOWPLAN_ALL ON
  - SET SHOWPLAN_TEXT ON
- Actual
  - SET STATISTICS PROFILE ON



```
Rows                 Executes             StmtText
-------------------- -------------------- -------------------------------------------------------------------------------------------------------------
19                   1                    WITH [BOM_cte]([ProductAssemblyID], [ComponentID], [ComponentDesc], [PerAssemblyQty], [StandardCost], [ListPr
    AS (
        SELECT b.[ProductAssemblyID], b.[ComponentID], p.[Name], b.[PerAssem 1           1           0           NULL                           NULL
19                   1                      |--Stream Aggregate(GROUP BY:([Recr1030], [Recr1024], [Recr1025], [Recr1026], [Recr1031], [Recr1028], [Recr
19                   1                           |--Sort(ORDER BY:([Recr1030] ASC, [Recr1024] ASC, [Recr1025] ASC, [Recr1026] ASC, [Recr1031] ASC, [Rec
19                   1                                |--Index Spool(WITH STACK)
19                   1                                     |--Concatenation
0                    0                                          |--Compute Scalar(DEFINE:([Expr1034]=(0)))
0                    0                                          |    |--Compute Scalar(DEFINE:([Expr1007]=(0)))
8                    1                                          |         |--Nested Loops(Inner Join, OUTER REFERENCES:([b].[ComponentID]))
8                    1                                          |              |--Nested Loops(Left Anti Semi Join)
8                    1                                          |              |    |--Clustered Index Seek(OBJECT:([AdventureWorks2008R2].[Production].
0                    8                                          |              |    |--Row Count Spool
0                    1                                          |              |         |--Clustered Index Scan(OBJECT:([AdventureWorks2008R2].[Sales].
8                    8                                          |              |--Clustered Index Seek(OBJECT:([AdventureWorks2008R2].[Production].[Proc
11                   1                                          |--Assert(WHERE:(CASE WHEN [Expr1036]>(25) THEN (0) ELSE NULL END))
11                   1                                               |--Nested Loops(Inner Join, OUTER REFERENCES:([Expr1036], [Recr1008], [Recr1009],
0                    0                                                    |--Compute Scalar(DEFINE:([Expr1036]=[Expr1035]+(1)))
19                   1                                                    |    |--Table Spool(WITH STACK)
0                    0                                                    |--Compute Scalar(DEFINE:([Expr1023]=[Recr1015]+(1)))
11                   19                                                        |--Nested Loops(Inner Join, OUTER REFERENCES:([b].[ComponentID]))
11                   19                                                             |--Nested Loops(Left Anti Semi Join)
11                   19                                                             |    |--Clustered Index Seek(OBJECT:([AdventureWorks2008R2].[Product
0                    11                                                             |    |--Row Count Spool
0                    1                                                              |         |--Clustered Index Scan(OBJECT:([AdventureWorks2008R2].[Sa
11                   11                                                             |--Clustered Index Seek(OBJECT:([AdventureWorks2008R2].[Production].
```

View Execution Plans - XML

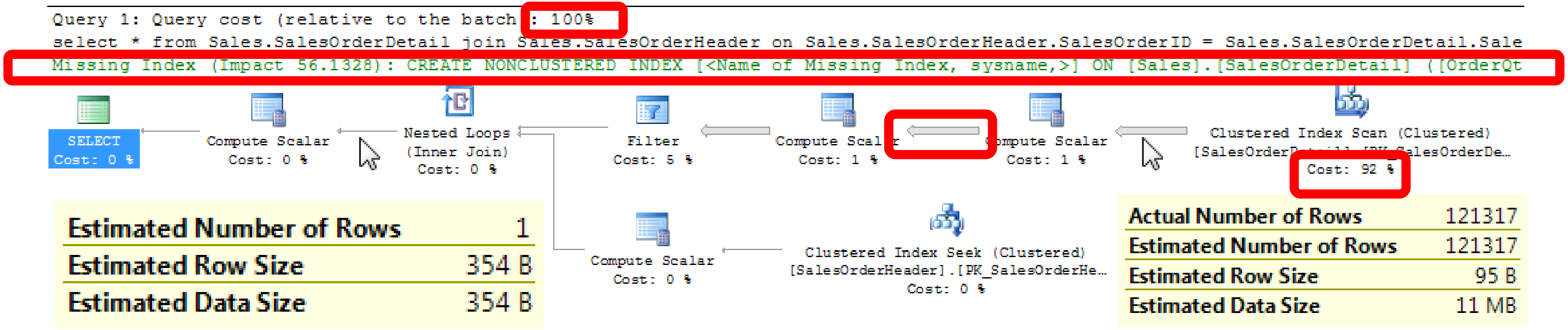
- Estimated
  - SET SHOWPLAN_XML ON
- Actual
  - SET STATISTICS XML ON



```
<?xml version="1.0" encoding="utf-16"?>
<ShowPlanXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1.1" Build="10.50.4000.0" xmlns="ht
  <BatchSequence>
    <Batch>
      <Statements>
        <StmtSimple StatementCompId="4" StatementEstRows="9.18693" StatementId="1" StatementOptmLevel="FULL" StatementSubTreeCost="2.42492" StatementText="W
          <StatementSetOptions ANSI_NULLS="true" ANSI_PADDING="true" ANSI_WARNINGS="true" ARITHABORT="true" CONCAT_NULL_YIELDS_NULL="true" NUMERIC_ROUNDABOR
          <QueryPlan DegreeOfParallelism="1" MemoryGrant="1024" CachedPlanSize="80" CompileTime="447" CompileCPU="354" CompileMemory="1256">
            <RelOp AvgRowSize="108" EstimateCPU="1.01056E-05" EstimateIO="0" EstimateRebinds="0" EstimateRewinds="0" EstimateRows="9.18693" LogicalOp="Aggre
              <OutputList>
                <ColumnReference Column="Recr1024" />
                <ColumnReference Column="Recr1025" />
                <ColumnReference Column="Recr1026" />
                <ColumnReference Column="Recr1028" />
                <ColumnReference Column="Recr1029" />
                <ColumnReference Column="Recr1030" />
                <ColumnReference Column="Recr1031" />
                <ColumnReference Column="Expr1032" />
              </OutputList>
              <RunTimeInformation>
                <RunTimeCountersPerThread Thread="0" ActualRows="19" ActualEndOfScans="1" ActualExecutions="1" />
              </RunTimeInformation>
              <StreamAggregate>
                <DefinedValues>
                  <DefinedValue>
                    <ColumnReference Column="Expr1032" />
                    <ScalarOperator ScalarString="SUM([Recr1027])">
                      <Aggregate AggType="SUM" Distinct="false">
```

How to read Execution Plans

- Right to left ?



```
Query 1: Query cost (relative to the batch : 100%
select * from Sales.SalesOrderDetail join Sales.SalesOrderHeader on Sales.SalesOrderHeader.SalesOrderID = Sales.SalesOrderDetail.Sale
Missing Index (Impact 56.1328): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [Sales].[SalesOrderDetail] ([OrderQt
```

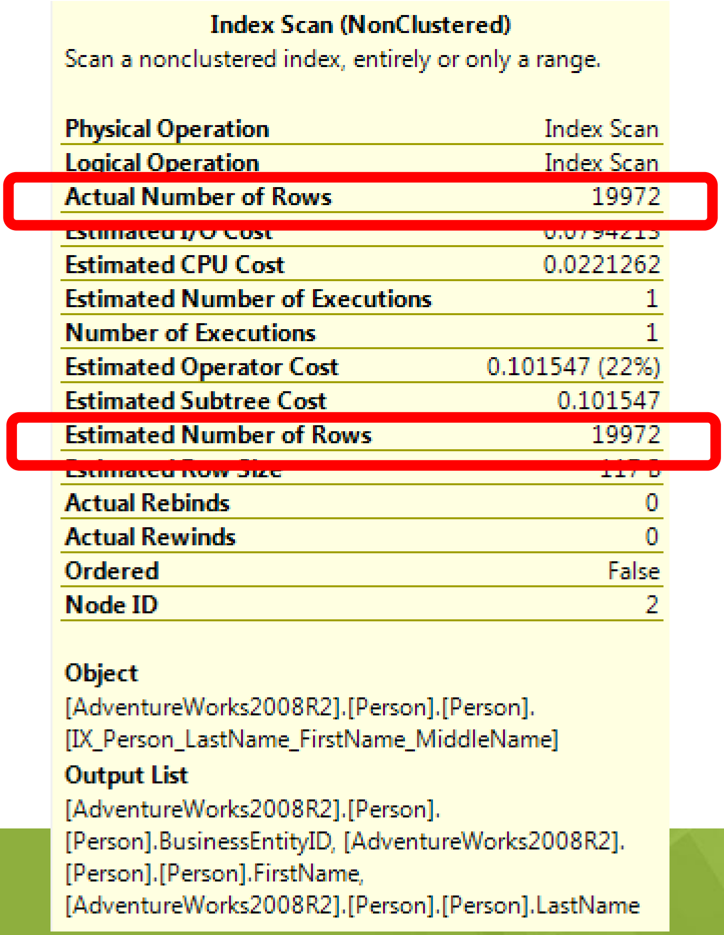
Estimated Number of Rows	1
Estimated Row Size	354 B
Estimated Data Size	354 B

Actual Number of Rows	121317
Estimated Number of Rows	121317
Estimated Row Size	95 B
Estimated Data Size	11 MB

- Batch cost
- Operator costs
- Arrows
- Missing Indexes

How to read Execution Plans

- Properties



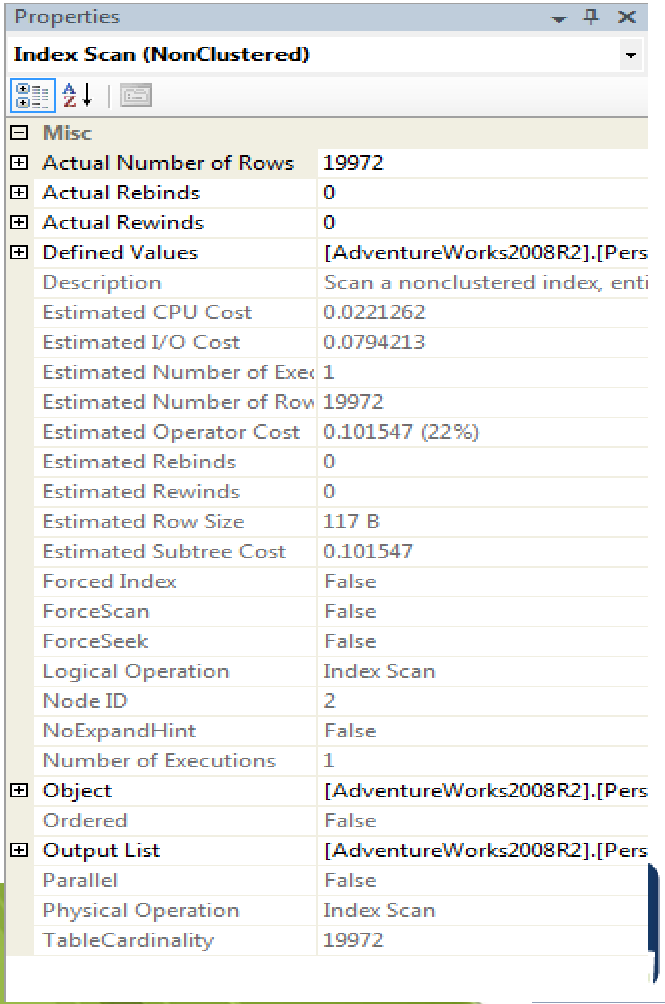
Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Number of Rows	19972
Estimated I/O Cost	0.0794213
Estimated CPU Cost	0.0221262
Estimated Number of Executions	1
Number of Executions	1
Estimated Operator Cost	0.101547 (22%)
Estimated Subtree Cost	0.101547
Estimated Number of Rows	19972
Estimated Row Size	117 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Object
[AdventureWorks2008R2].[Person].[Person].
[IX_Person_LastName_FirstName_MiddleName]

Output List
[AdventureWorks2008R2].[Person].
[Person].BusinessEntityID, [AdventureWorks2008R2].
[Person].[Person].FirstName,
[AdventureWorks2008R2].[Person].[Person].LastName



Properties

Index Scan (NonClustered)

Misc	
Actual Number of Rows	19972
Actual Rebinds	0
Actual Rewinds	0
Defined Values	[AdventureWorks2008R2].[Pers
Description	Scan a nonclustered index, enti
Estimated CPU Cost	0.0221262
Estimated I/O Cost	0.0794213
Estimated Number of Exec	1
Estimated Number of Row	19972
Estimated Operator Cost	0.101547 (22%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	117 B
Estimated Subtree Cost	0.101547
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Index Scan
Node ID	2
NoExpandHint	False
Number of Executions	1
Object	[AdventureWorks2008R2].[Pers
Ordered	False
Output List	[AdventureWorks2008R2].[Pers
Parallel	False
Physical Operation	Index Scan
TableCardinality	19972

Operators

- Data retrieval
  - Table Scan
  - Clustered Index Scan
  - Clustered Index Seek
  - Non-clustered Index Scan
  - Non-clustered Index Seek
  - Key Lookup
  - RID Lookup
- Join (loop/merge/hash)
- Sort

Examples given tested against:
✓ AdventureWorks2008R2
✓ AdventureWorks2012
✓ AdventureWorks2014

Data Operations – Table Scans

```
select *
from Production.ProductProductPhoto
```

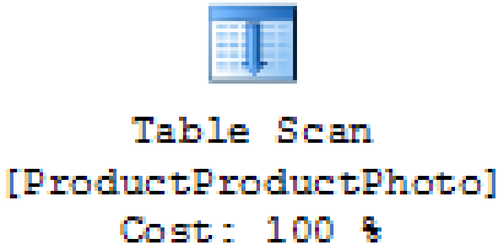
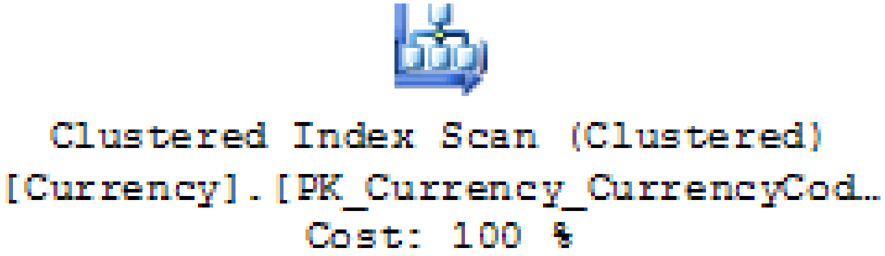


Table Scan
[ProductProductPhoto]
Cost: 100 %

- Table Scan – Heap
  - All (or majority) of rows
  - No useful indexes
  - Small tables
  - Table variables / CTE
- Issue?
  - Large number of rows

Data Operations – Index Scans

```
select *
from Sales.Currency
```

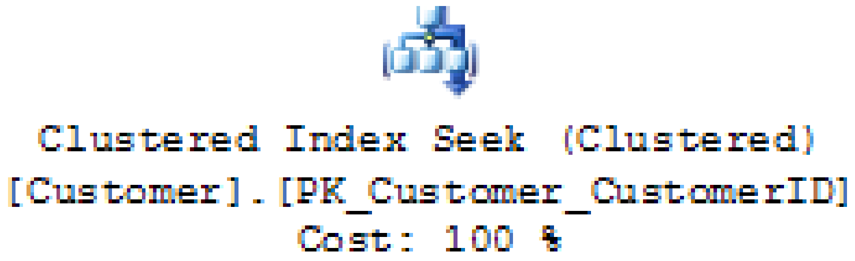


Clustered Index Scan (Clustered)
[Currency].[PK_Currency_CurrencyCod…
Cost: 100 %

- Clustered Index Scans
  - Same as a table scan !
- Issue?
  - Large range of data is been selected
  - Maybe benefit from different index
  - Stale statistics

Data Operations – Index Seeks

```
select CustomerID, PersonId
from Sales.Customer
where CustomerID < 10
```

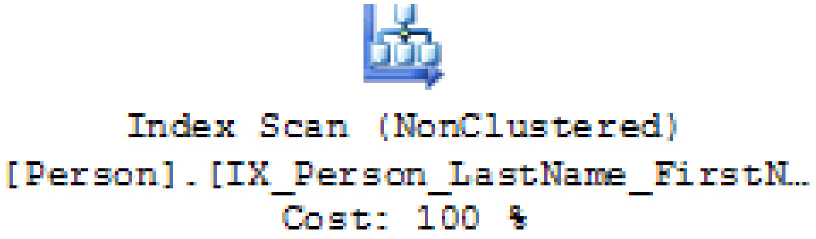


Clustered Index Seek (Clustered)
[Customer].[PK_Customer_CustomerID]
Cost: 100 %

- Clustered Index Seek
  - ‘Gold’ standard
  - All data available at leaf level

Data Operations – Index Scans

```
select BusinessEntityID,
     FirstName, LastName
from Person.Person
```

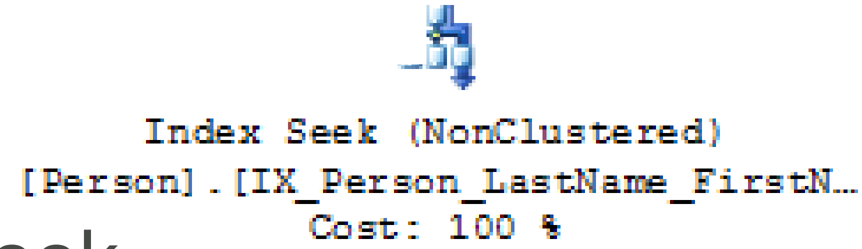


Index Scan (NonClustered)
[Person].[IX_Person_LastName_FirstN…
Cost: 100 %

- Non-Clustered Index Scan
  - No optimal index for the query
  - Dataset returned represents most of the table
- Issue?
  - Refine WHERE clause
  - Lookups (coming later….)

Data Operations – Index Seeks

```sql
select BusinessEntityID, FirstName, LastName
from Person.Person
where LastName = 'Riley'
```



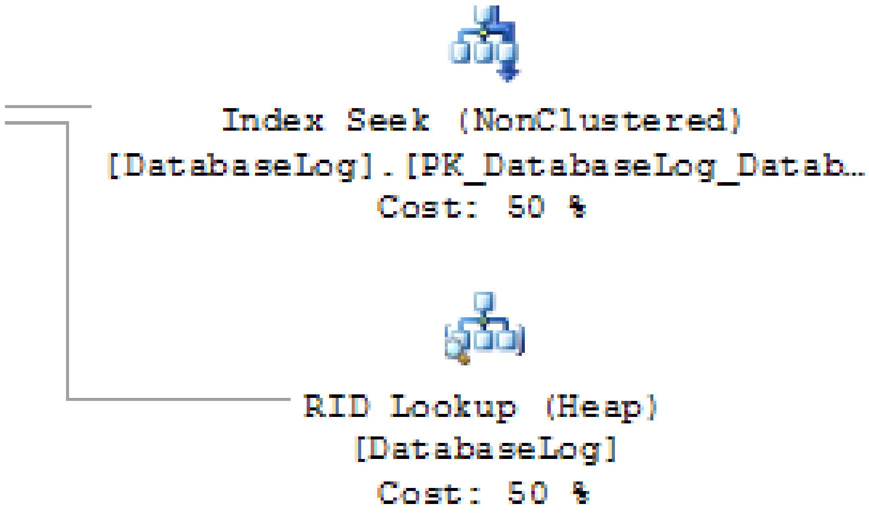
```
Index Seek (NonClustered)
[Person].[IX_Person_LastName_FirstN…
Cost: 100 %
```

- Non-Clustered Index Seek
  - Similar to CIX seek but only index fields available
- Issue?
  - May cause Lookups

Data Operations - Lookups

- Only on non-clustered index operations
- RID Lookup (Heap)

```sql
select  DatabaseLogID, DatabaseUser
from dbo.DatabaseLog
where DatabaseLogID = 100
```



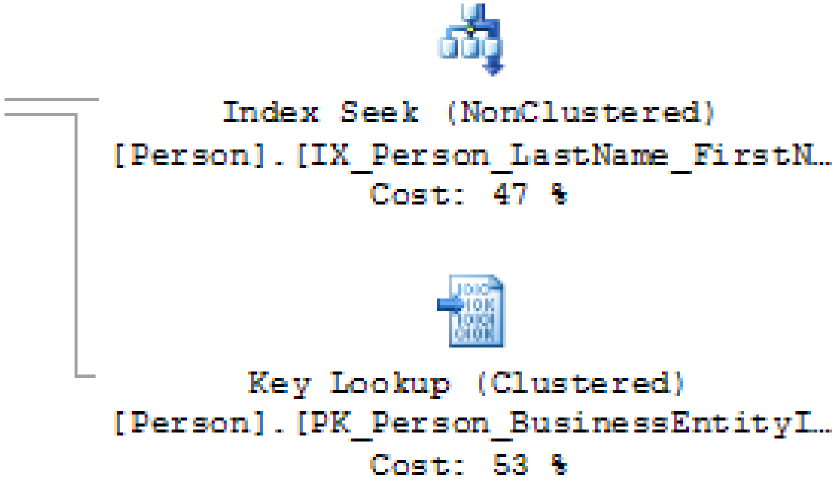
Index Seek (NonClustered)
[DatabaseLog].[PK_DatabaseLog_Datab…
Cost: 50 %

RID Lookup (Heap)
[DatabaseLog]
Cost: 50 %

Data Operations - Lookups

- Only on non-clustered index operations
- Key Lookup

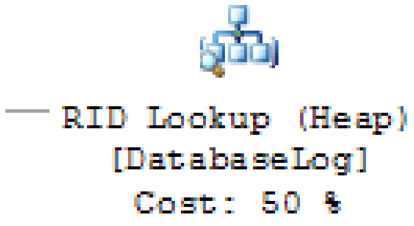
```
select BusinessEntityID, FirstName, LastName, PersonType
from Person.Person
where LastName = 'Riley'
```



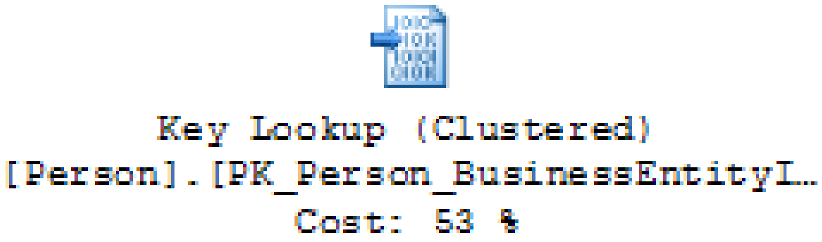
Index Seek (NonClustered)
[Person].[IX_Person_LastName_FirstN…
Cost: 47 %

Key Lookup (Clustered)
[Person].[PK_Person_BusinessEntityI…
Cost: 53 %

Data Operations - Lookups



RID Lookup (Heap)
[DatabaseLog]
Cost: 50 %



Key Lookup (Clustered)
[Person].[PK_Person_BusinessEntityI…
Cost: 53 %

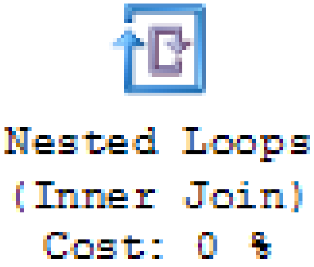
- Eliminating lookups
  - Review column selection
  - Covering indexes (key fields or INCLUDEs)
- Exceptions
  - Select every column from the table

Join Operations

- Three main logical join operators
  - Nested Loop join
  - Merge join
  - Hash join

Join Operation – Nested Loop

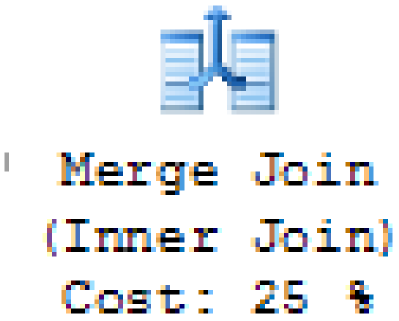
- Very efficient operation
- Takes an outer data set and compares one row at a time to inner data set
- Most effective when
  - Outer input is small
  - Inner (small or large)is indexed



Nested Loops
(Inner Join)
Cost: 0 %

Join Operations - Merge

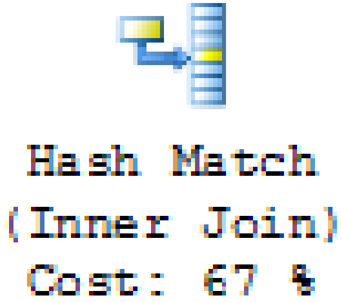
- Sorted data makes this a very efficient operation
- Addition of Sort operator makes it less efficient
- Otherwise Hash Join



Merge Join
(Inner Join)
Cost: 25 %

Join Operations – Hash Join

- Large, unsorted data
- Efficient where no useable indexes



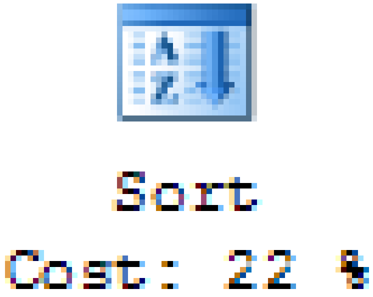
Hash Match
(Inner Join)
Cost: 67 %

Join Operations - Summary

- No ‘ideal’ join – depends on data
- Join operators even if there is no JOIN statement
- Resource-wise Nested Loop is ‘best’
- Merge requires sorted data – can be forced

Sort

- Orders the data
  - ORDER BY
  - Ranking functions
    - ROW_NUMBER()
    - RANK()
    - DENSE_RANK()
    - NTILE()
  - Merge Join
- Blocking Operator
- Can be expensive



Sort
Cost: 22 %

Performance considerations

- Seek vs. Scan
  - Missing indexes
  - Statistics
  - Functions on indexed columns non-Sargable
- Actual rows vs. Estimated Rows
  - Statistics
  - Table variables (consider temp tables)
  - User defined functions
    - Scalar functions
    - Multi-statement table valued functions
    - Consider re-writing to inline table valued functions

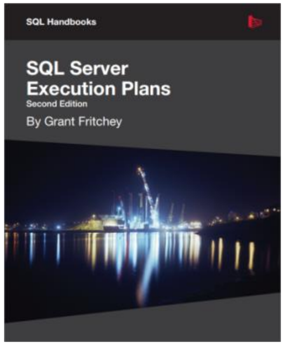
Performance considerations

- Sort operations expensive
  - Statistics are wrong – spill to tempdb/disk
- Correct join operators
  - Statistics
- Review Lookups
- Did I mention statistics?

Further Resources

- SQLSentry Plan Explorer FREE!

- SQL Server Execution Plans – Grant Fritchey
  @GFritchey | www.scarydba.com



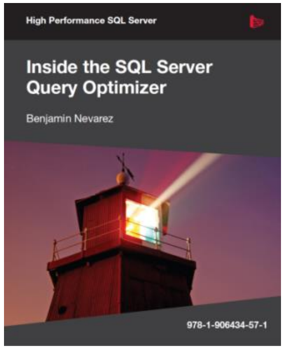
SQL Handbooks

SQL Server Execution Plans

Second Edition

By Grant Fritchey

- Inside the SQL Server Query Optimizer – Benjamin Nevarez
  @BenjaminNevarez | www.benjaminnevarez.com



High Performance SQL Server

Inside the SQL Server Query Optimizer

Benjamin Nevarez

978-1-906434-57-1

Takeaway

- Execution plan is your window
- Get familiar
- Statistics!

Thanks for listening

kevan.riley@rileywaterhouse.co.uk

www.rileywaterhouse.co.uk

@kevriley

Today is brought to you by

redgate

ingeniously simple

and in association with Microsoft

Please visit our sponsors

SanDisk®

BIG BANG DATA CO

DELL Software

PYRAMID ANALYTICS

SQLSENTRY®

Micron®

Prodata
SQL Centre of Excellence

Thanks for listening

kevan.riley@rileywaterhouse.co.uk
www.rileywaterhouse.co.uk
@kevriley