

Execution Plan Cost Formulas

Joe Chang

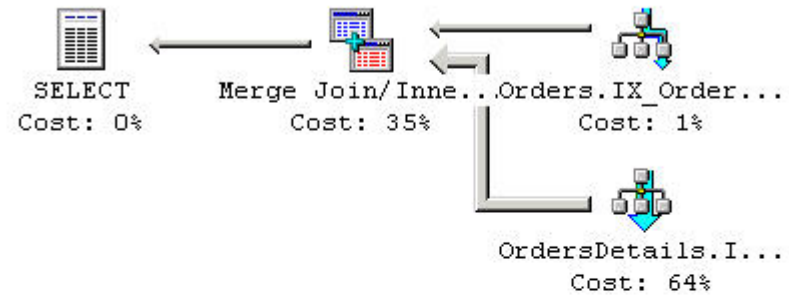
Performance Tuning Corporation

jchang@perftuning.com

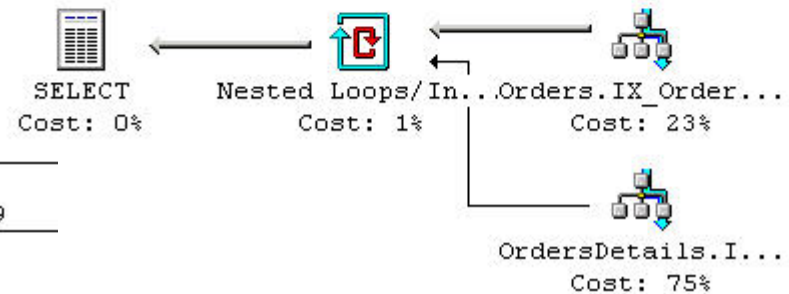


**SELECT xx FROM Orders o
INNER JOIN OrdersDetails od ON
od.OrderID = o.OrderID
WHERE o.CustomerID = 10900**

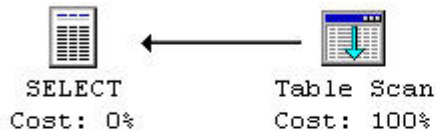
Query 1: Query cost (relative to the batch): 96.45%
Query text: SELECT od.OrderID, od.OrderItemID, od.ProductID



Query 2: Query cost (relative to the batch): 3.55%
Query text: SELECT od.OrderID, od.OrderItemID, od.ProductID



Query 1: Query cost (relative to the batch): 47.95%
Query text: SELECT * FROM M2N WHERE ID2 BETWEEN 1 AND 89



Query 2: Query cost (relative to the batch): 52.05%
Query text: SELECT * FROM M2N WHERE ID2 BETWEEN 1 AND 88



**SELECT xx
FROM M2N
WHERE ID2 BETWEEN 1 AND 88**

Clustered Index Scan

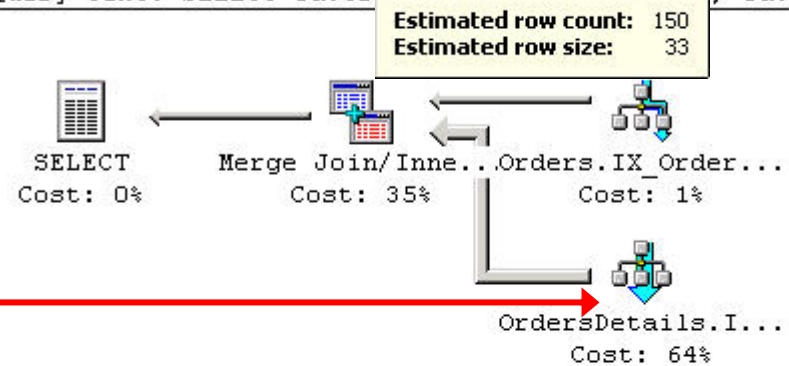
Scanning a clustered index, entirely or only a range.

Physical operation:	Clustered Index Scan
Logical operation:	Clustered Index Scan
Estimated row count:	125,000
Estimated row size:	42
Estimated I/O cost:	0.355
Estimated CPU cost:	0.137
Estimated number of executes:	1.0
Estimated cost:	0.492935(64%)
Estimated subtree cost:	0.492

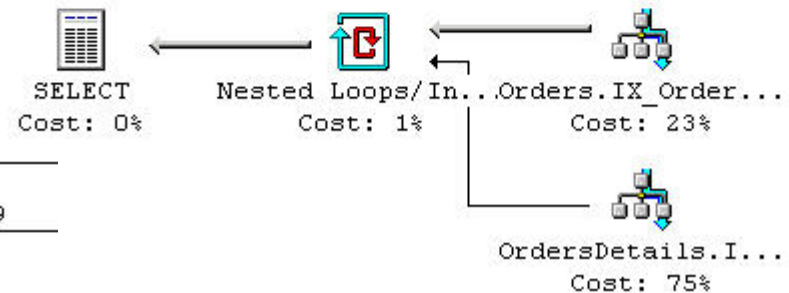
Argument:

OBJECT:([Nile].[dbo].[OrdersDetails].[IX_OD_OrderId] AS [od]), ORDERED FORWARD

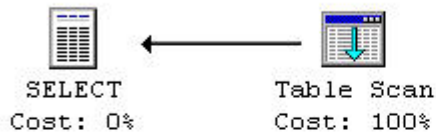
Query 1: Query cost (relative to the batch): 96.45%
Query text: SELECT od.OrderID, od.ProductID



Query 2: Query cost (relative to the batch): 3.55%
Query text: SELECT od.OrderID, od.OrderItemID, od.ProductID



Query 1: Query cost (relative to the batch): 47.95%
Query text: SELECT * FROM M2N WHERE ID2 BETWEEN 1 AND 89



Query 2: Query cost (relative to the batch): 52.05%
Query text: SELECT * FROM M2N WHERE ID2 BETWEEN 1 AND 88



Index Seek

Scanning a particular range of rows from a non-clustered index.

Physical operation:	Index Seek
Logical operation:	Index Seek
Estimated row count:	87
Estimated row size:	19
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000176
Estimated number of executes:	1.0
Estimated cost:	0.006504(1%)
Estimated subtree cost:	0.00650

Argument:

OBJECT:([pubs].[dbo].[M2N].[IX_M2N_ID2]), SEEK:([M2N].[ID2] >= Convert([@1]) AND [M2N].[ID2] <= Convert([@2])) ORDERED FORWARD

Objective

- Why SQL Server uses a given execution plan for a specific query
 - Cost Model & Statistics
- When the execution plan changes
- Performance testing
 - How to get same results in test as actual use
- Size and cardinality



Topics

- **Component Operation Cost Model**
 - Cost Formulas for basic operations
- **Dependencies**
 - Rows & Pages involved - **yes**
 - Index depth – **no**, Locks level – **no**
 - WHERE conditions – yes/no



Execution Plan

- Query
 - One or more possible execution plans
- Plan component operations:
 - index seek, table scan, join, sort, aggregate
- Component operation cost formula
 - Depends on # of pages and rows involved
- Distribution statistics on tables & indexes
 - Estimate number of pages and rows involved



Component Operation Costs

- **Basic Component Operations**

- Index Seek, Table/Index Scan, Joins,



- Aggregates, Compute Scalar, Sort, etc



- **Each component operation has a cost formula**

- Function of number of row or pages involved



Execution Plan Information

- **Query Analyzer**

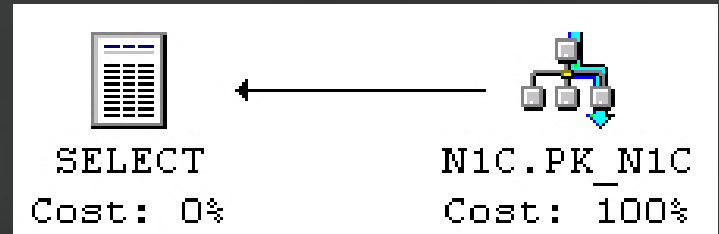
- Display Estimated Execution Plan
- Show Execution Plan

- **SET Options**

- SHOWPLAN_ALL, SHOWPLAN_TEXT
- STATISTICS PROFILE

- **Profiler**

- Events-Performance, Data Col-Binary Data



Display Estimated Execution Plan

Clustered Index Seek

Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek	
Logical operation:	Clustered Index Seek	
Estimated row count:	1	← row count
Estimated row size:	21	
Estimated I/O cost:	0.00632	← I/O Cost
Estimated CPU cost:	0.000080	← CPU Cost
Estimated number of executes:	1.0	← # of executes
Estimated cost:	0.006408(100%)	← cost
Estimated subtree cost:	0.00640	

Argument:

OBJECT:([pubs],[dbo],[N1C],[PK_N1C]), SEEK:([N1C].[ID]=Convert([@1])) ORDERED FORWARD



Show Execution Plan

Clustered Index Seek

Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek	
Logical operation:	Clustered Index Seek	
Row count:	1	←
Estimated row size:	21	
I/O cost:	0.00632	
CPU cost:	0.000080	
Number of executes:	1	
Cost:	0.006408(100%)	
Subtree cost:	0.00640	
Estimated row count:	1	←

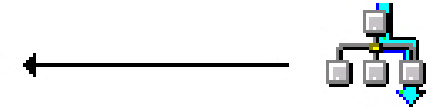
Argument:

```
OBJECT:([pubs].[dbo].[N1C].[PK_N1C]), SEEK:([N1C].[ID  
]=Convert([@1])) ORDERED FORWARD
```

Index Seek - 1 row



SELECT
Cost: 0%



N1C.PK_N1C
Cost: 100%

```
SELECT xx
FROM N1C
WHERE ID = @ID
```

Clustered Index Seek

Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	1.0
Estimated cost:	0.006408(100%)
Estimated subtree cost:	0.00640

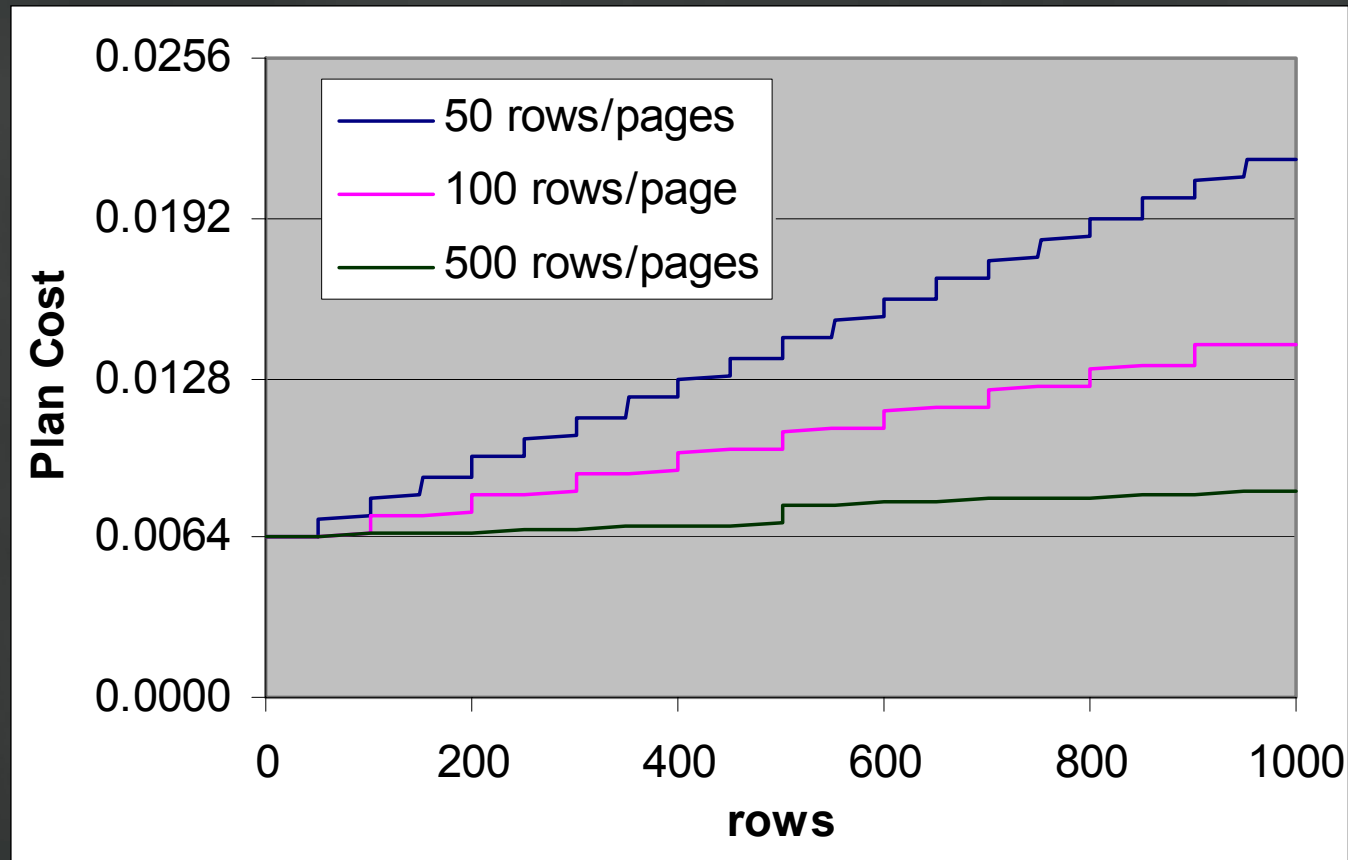
Argument:

OBJECT:([pubs].[dbo].[N1C].[PK_N1C]), SEEK:([N1C].[ID]=Convert([@1])) ORDERED FORWARD

	I/O	CPU	Total
≤ 1GB	0.006328500	0.0000796	0.006408100
> 1GB	0.003203425	0.0000796	0.003283025



Index Seek Cost Formula



Multiple rows, \leq 1GB

I/O: 0.00632850 + 0.00074074 per additional page (leaf level)

CPU: 0.00007960 + 0.00000110 per additional row

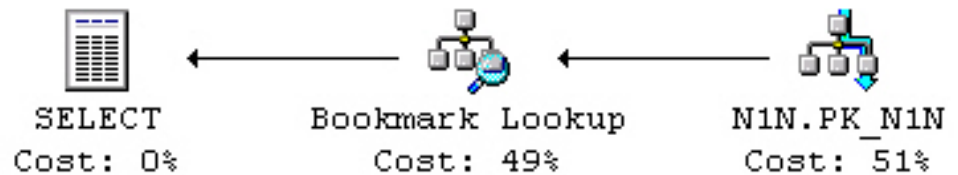
Total: 0.00640810 + 0.00074074 / add. page + 0.00000111 / add. row



Bookmark Lookup

```
SELECT xx  
FROM N1N  
WHERE ID = @ID
```

Same cost for
bookmark lookup
on Heap and
Clustered Index



Bookmark Lookup

Use a Bookmark (RID or Clustering Key) to look up the corresponding row in the Table or Clustered Index.

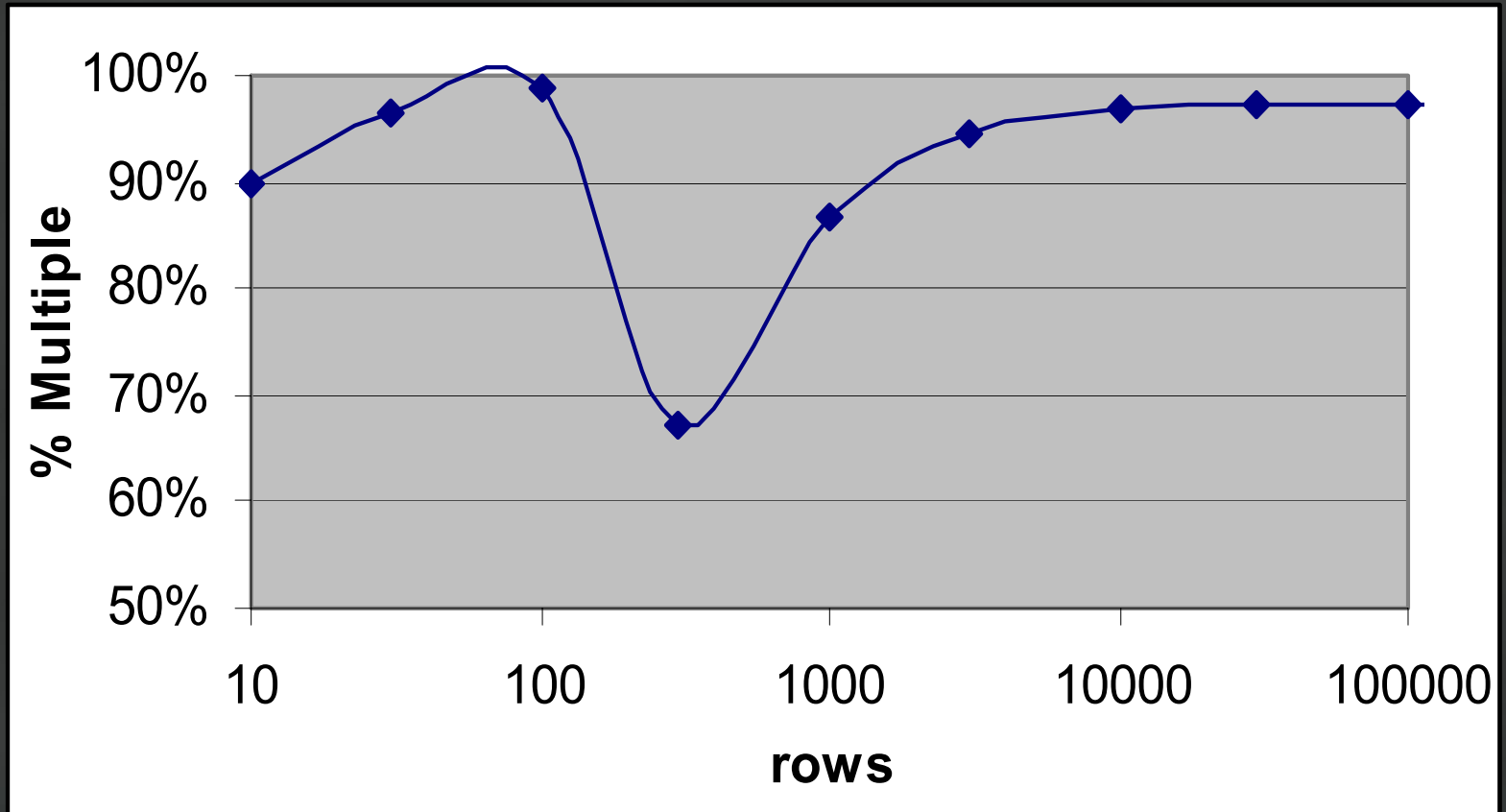
Physical operation:	Bookmark Lookup
Logical operation:	Bookmark Lookup
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.00625
Estimated CPU cost:	0.000001
Estimated number of executes:	1.0
Estimated cost:	0.006251(49%)
Estimated subtree cost:	0.0126

Argument:

BOOKMARK:([Bmk1000]), OBJECT:([pubs].[dbo].[N1N])

	I/O	CPU	Total
≤ 1GB	0.0062500	0.0000011	0.0062511
> 1GB	0.0031249	0.0000011	0.0031260

Bookmark Lookup Cost Formula



I/O: multiple of 0.0062500 ($\leq 1\text{GB}$), 0.0031249 ($> 1\text{GB}$)

CPU: 0.0000011 per row

Total: multiple of I/O + (# of rows) \times 0.0000011

Example: 500,000 rows, 99 rows / page

Table Scan



SELECT
Cost: 0%



Table Scan
Cost: 100%

```
SELECT xx  
FROM N1H  
WHERE ID = @ID
```

Table Scan

Scan rows from a table.

Physical operation:	Table Scan
Logical operation:	Table Scan
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.0375
Estimated CPU cost:	0.000430
Estimated number of executes:	1.0
Estimated cost:	0.038009(100%)
Estimated subtree cost:	0.0380

Argument:

OBJECT:([pubs].[dbo].[N1H]), WHERE:([N1H].[ID]=Convert([@1
]))

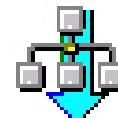
I/O: 0.0375785 + 0.0007407/page

CPU: 0.0000785 + 0.0000011/row

Index Scan



SELECT
Cost: 0%



N1C.PK_N1C
Cost: 100%

Clustered Index Scan

Scanning a clustered index, entirely or only a range.

Physical operation:	Clustered Index Scan
Logical operation:	Clustered Index Scan
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.0375
Estimated CPU cost:	0.000430
Estimated number of executes:	1.0
Estimated cost:	0.038009(100%)
Estimated subtree cost:	0.0380

Argument:

OBJECT:([pubs].[dbo].[N1C].[PK_N1C]), WHERE:([N1C].[Value]=[@1])



Bookmark versus Scan

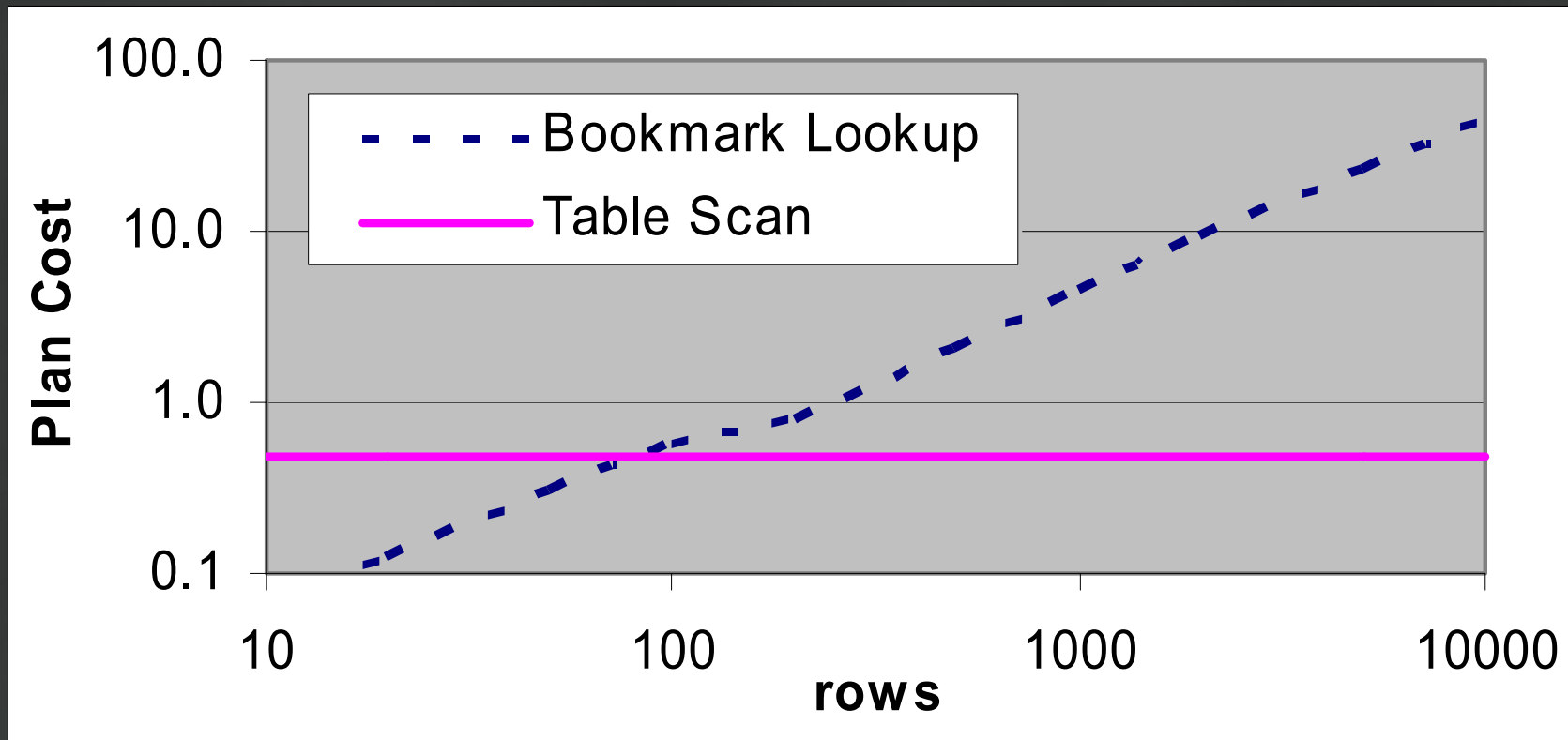
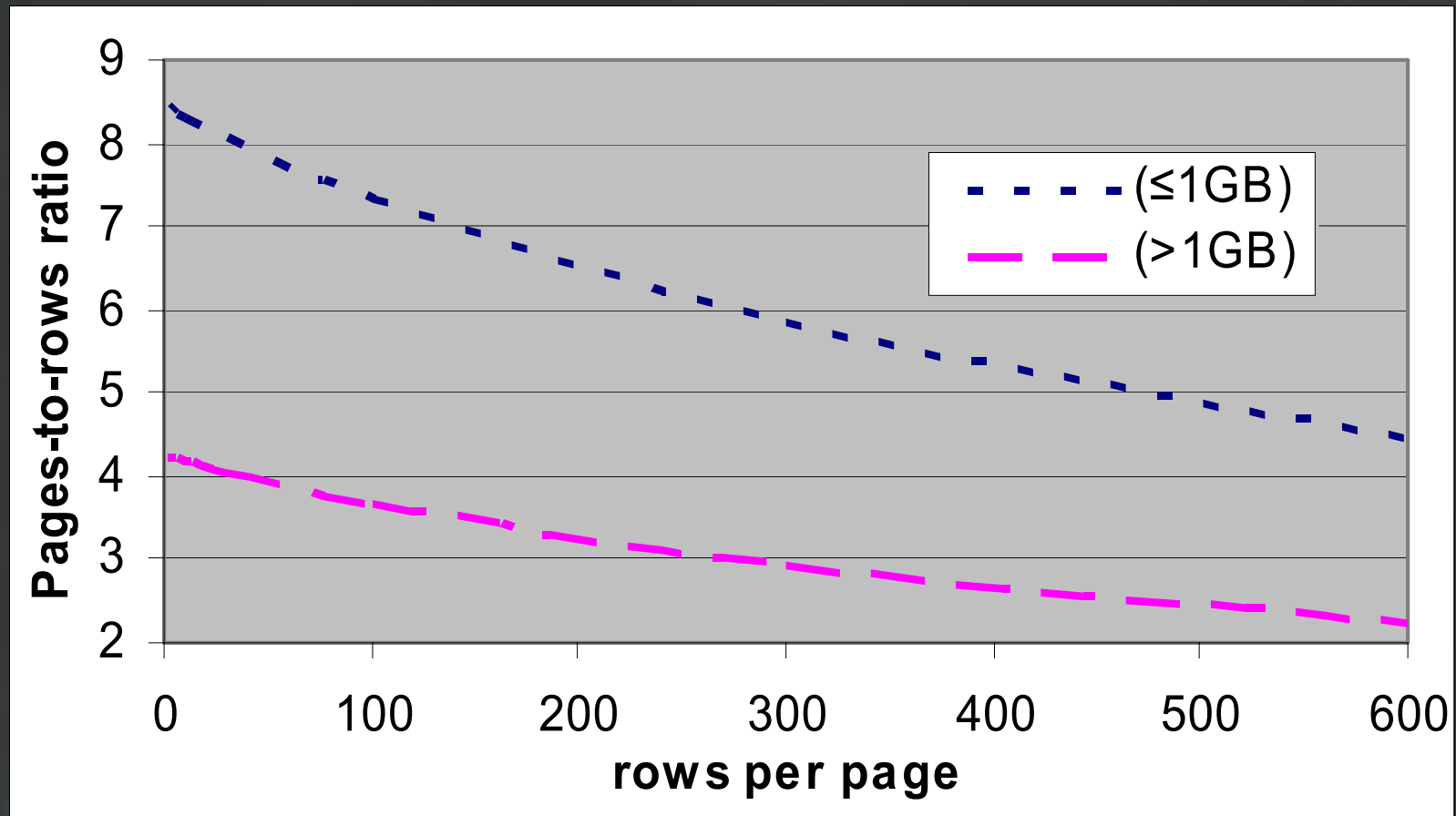


Table scan cost for 50,000 row, 506 pages
Index Seek and Bookmark Lookup cost for $\leq 1\text{GB}$

$\leq 1\text{GB}$



Bookmark-Table Scan Crossover



Rows $\sim 5 + \text{Pages} \div (\text{CF} \times (\text{Pages-to-rows ratio}))$ ($\leq 1\text{GB}$)

$\sim 11 + \text{Pages} \div (\text{CF} \times (\text{Pages-to-rows ratio}))$ ($> 1\text{GB}$)

1 Bookmark Lookup costs ~ 7 ($\leq 1\text{GB}$) or 3.5 ($> 1\text{GB}$)
times more than a scan of 1 page



Bookmark & Table Scan Costs

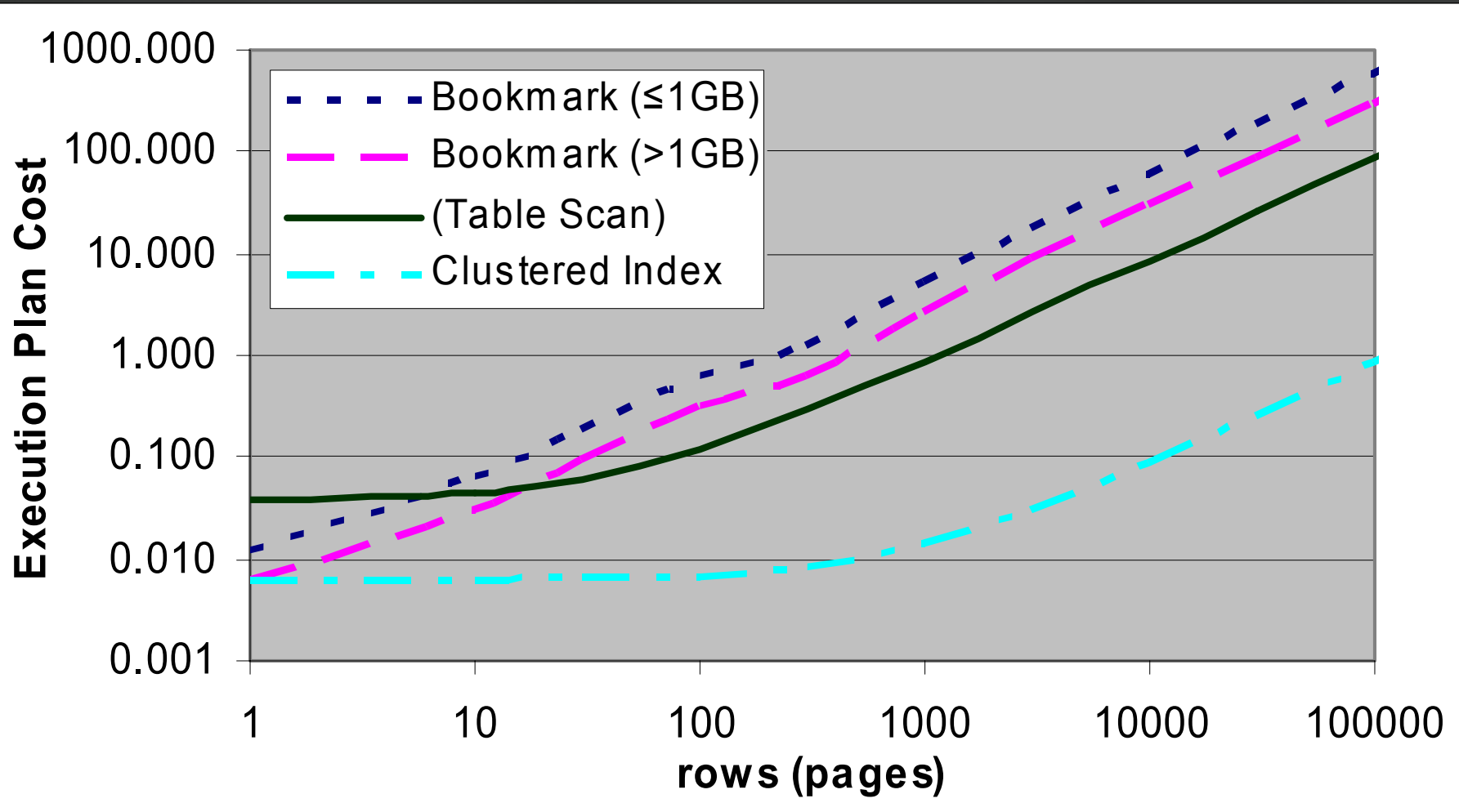


Table scan: cost per page, others: cost per row

Aggregates



Aggregate:
MIN & MAX

```
SELECT MIN(x)
FROM M2C
WHERE
GroupID = 1
```

Stream Aggregate/Aggregate

Computing summary values for groups of rows in a suitably sorted stream.

Physical operation:	Stream Aggregate
Logical operation:	Aggregate
Estimated row count:	1
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000050
Estimated number of executes:	1.0
Estimated cost:	0.000050(1%)
Estimated subtree cost:	0.00700

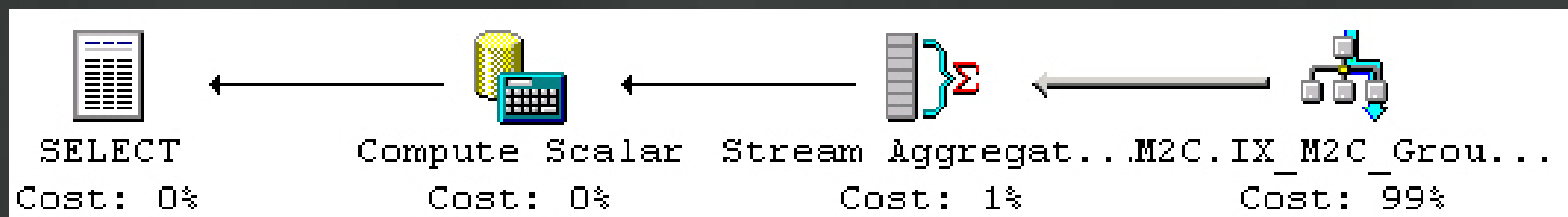
Argument:

[Expr1012]=Count(*), [Expr1013]=SUM([M2C].[randDecimal])

For single row result I/O: None CPU: 0.0000001/row



Compute Scalar



Compute Scalar


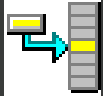

Computing new values from existing values in a row.

Physical operation:	Compute Scalar
Logical operation:	Compute Scalar
Estimated row count:	1
Estimated row size:	11
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000050
Estimated number of executes:	1.0
Estimated subtree cost:	0.00774

Argument:

DEFINE:([Expr1002]=Convert([Expr1003]))

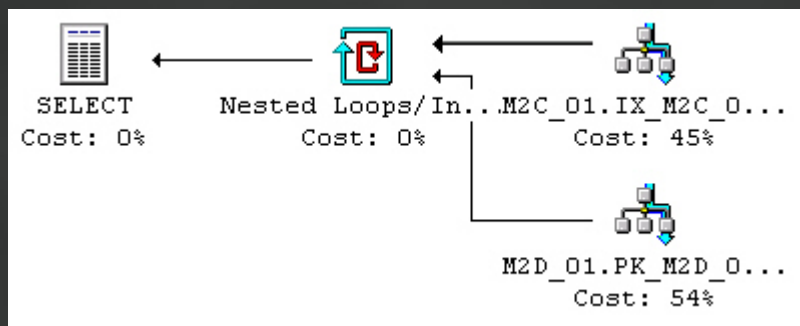
Loop, Hash and Merge Joins

- SQL Server supports 3 types of joins
 - Loop , Hash , Merge 
- Hash join subtypes
 - In memory, Grace, Recursive
 - Different settings for SQL Batch & RPC
- Merge join
 - one-to-many
 - many-to-many



Loop Joins

```
SELECT M2C_01.ID, N1C.Value
FROM M2C_01 INNER JOIN M2D_01
ON M2D_01.ID = M2C_01.ID2
WHERE M2C_01.Group1 = @Group1
```



Outer Source (Top Input)

Index Seek
Scanning a particular range of rows from a non-clustered index.

Physical operation:	Index Seek
Logical operation:	Index Seek
Estimated row count:	10
Estimated row size:	38
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000090
Estimated number of executes:	1.0
Estimated cost:	0.006418(45%)
Estimated subtree cost:	0.00641

Argument:
OBJECT:([pubs],[dbo],[M2C_01],[IX_M2C_01_GroupID]), SEEK:([M2C_01].[GroupID]=1) ORDERED FORWARD

Inner Source (Bottom Input)

Clustered Index Seek
Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	42
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	10.0
Estimated cost:	0.007697(54%)
Estimated subtree cost:	0.00769

Argument:
OBJECT:([pubs],[dbo],[M2D_01],[PK_M2D_01]), SEEK:([M2D_01].[ID]=[M2C_01].[ID]) ORDERED FORWARD

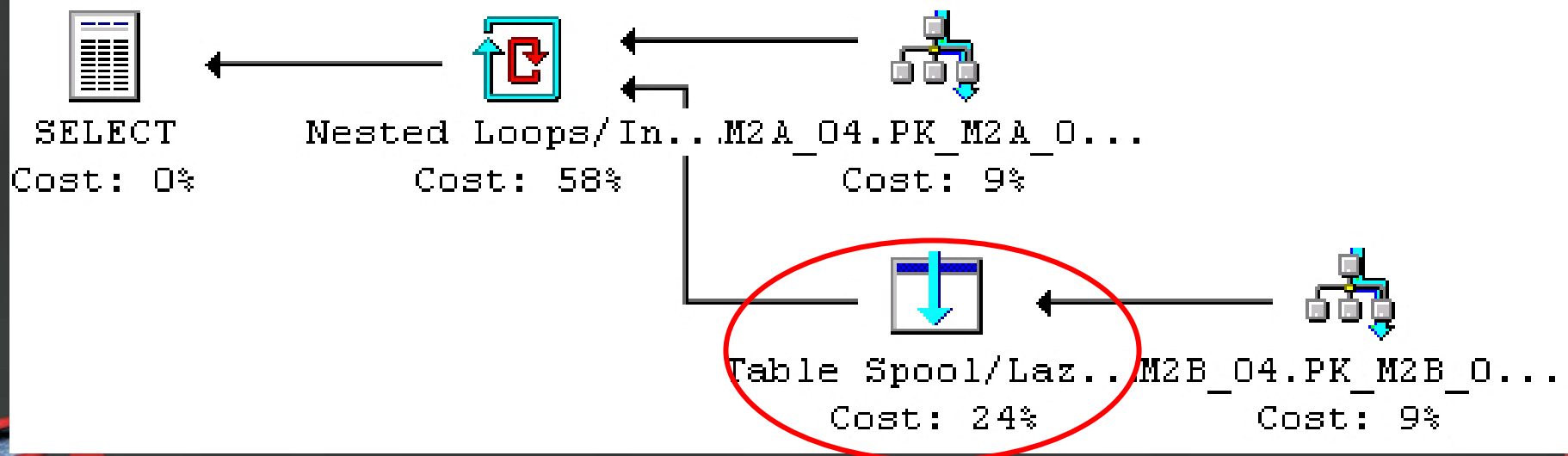
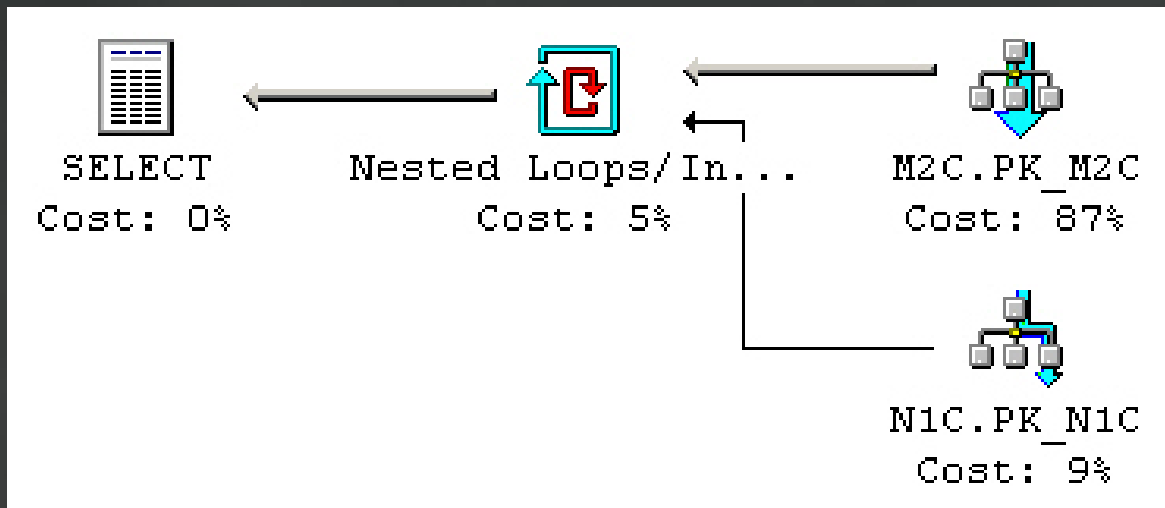
Join

Nested Loops/Inner Join
For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.

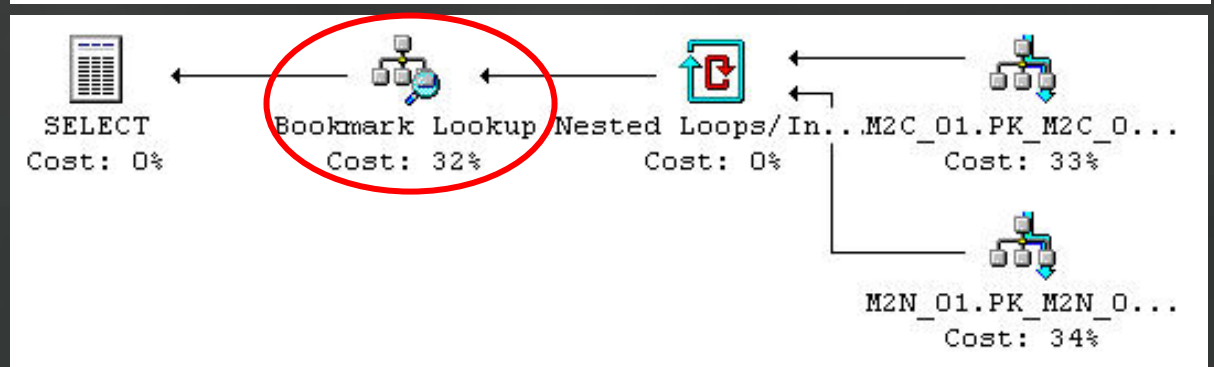
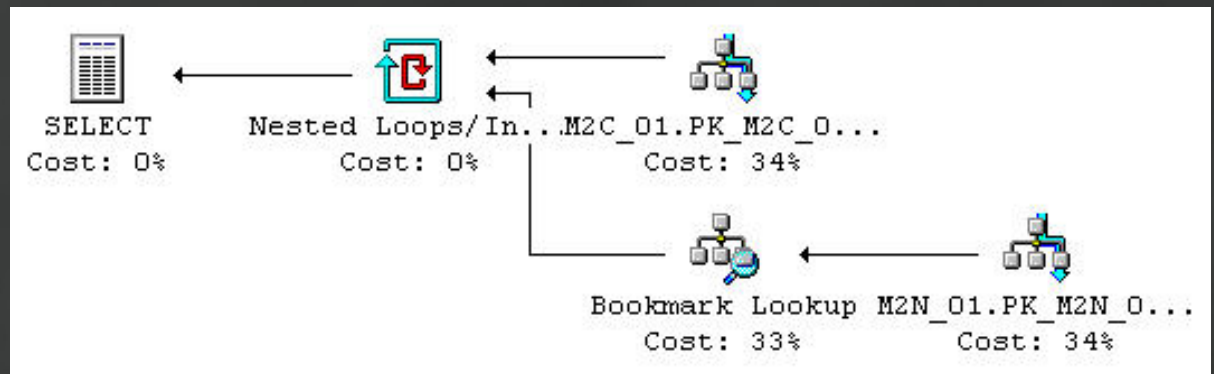
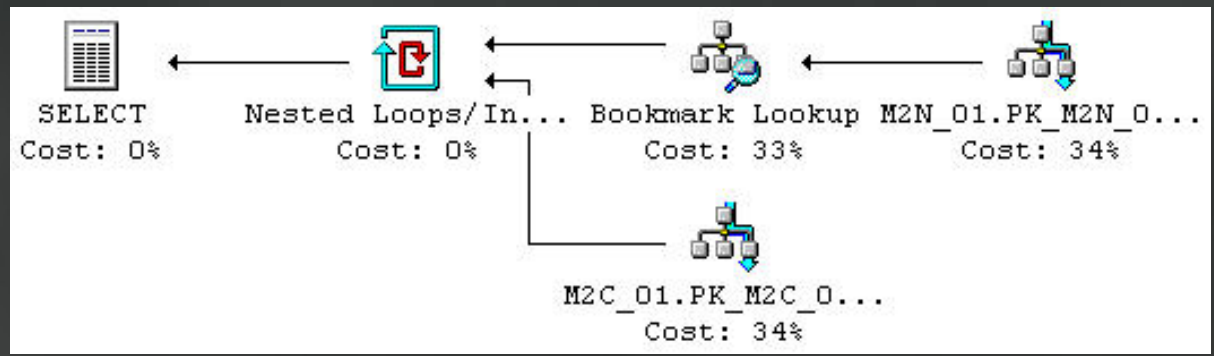
Physical operation:	Nested Loops
Logical operation:	Inner Join
Estimated row count:	9
Estimated row size:	72
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000042
Estimated number of executes:	1.0
Estimated cost:	0.000042(0%)
Estimated subtree cost:	0.0141

Argument:
OUTER REFERENCES:([M2C_01].[ID])

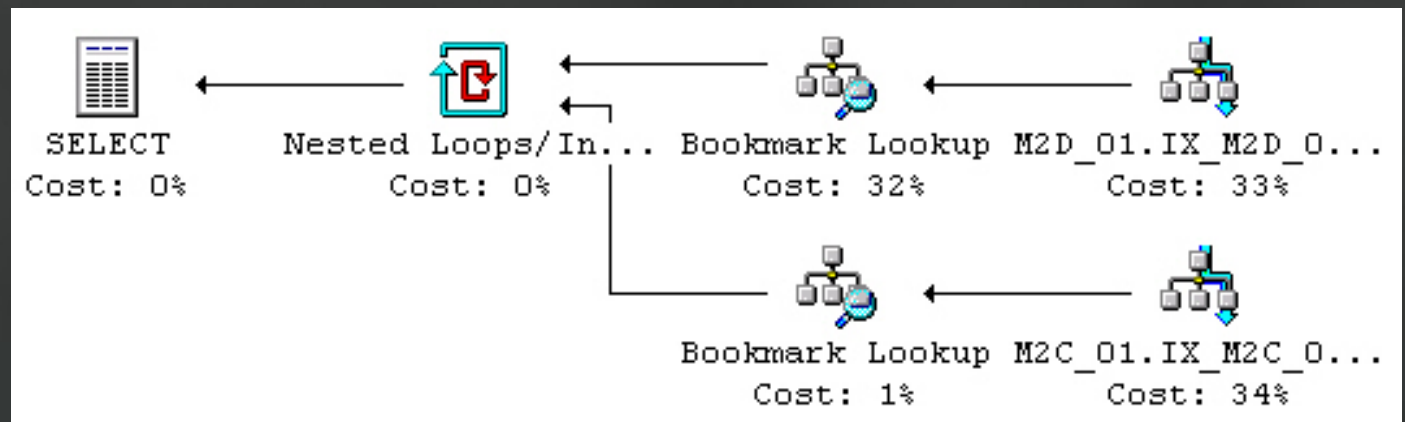
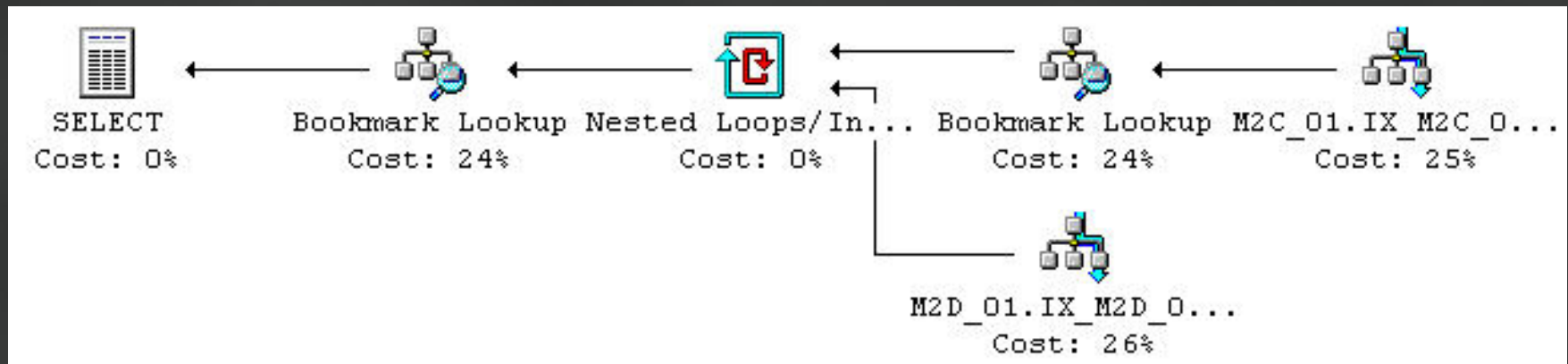
Loop Join with Scan, Spool



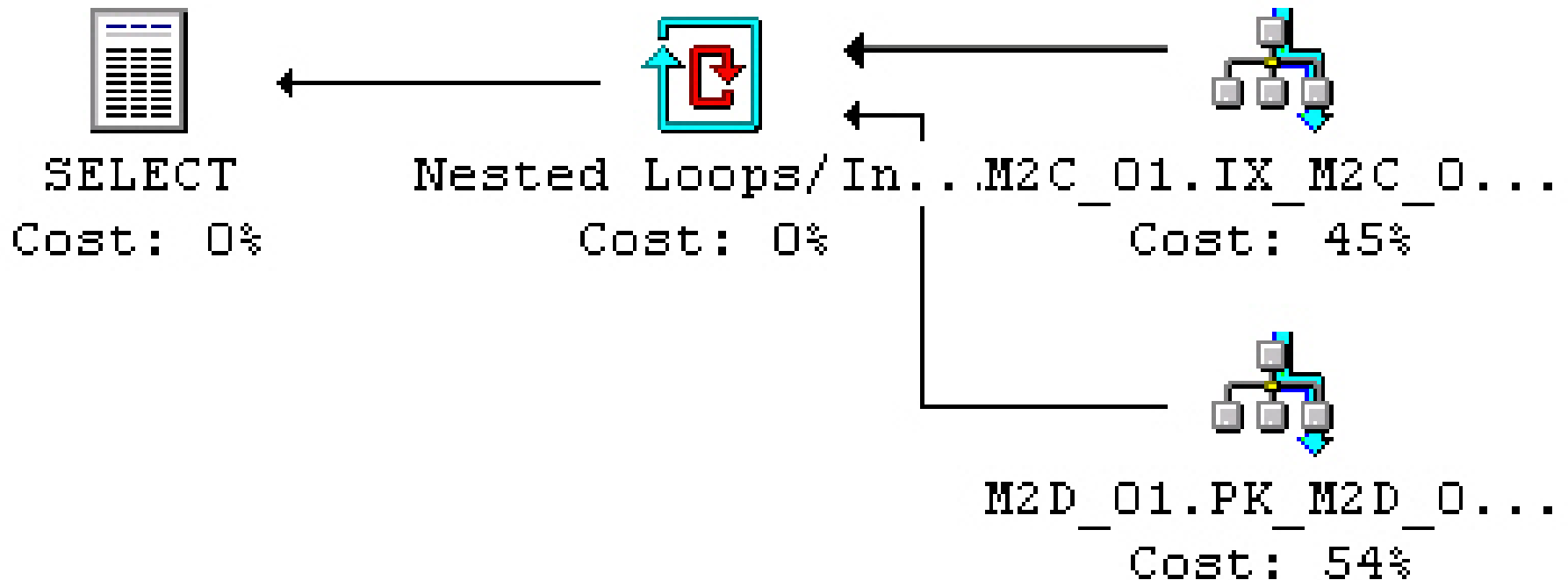
Loop Joins with Bookmark Lookups



Joins with 2 Bookmark Lookups



Loop Join Cost



Complete Loop Join Cost

= Outer Source Cost + Inner Source Cost + Join Cost



Loop Join – Outer Source

Index Seek

Scanning a particular range of rows from a non-clustered index.

Physical operation:	Index Seek
Logical operation:	Index Seek
Estimated row count:	10
Estimated row size:	38
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000090
Estimated number of executes:	1.0
Estimated cost:	0.006418(45%)
Estimated subtree cost:	0.00641

— 1 or more rows

— 1 execute

Argument:

OBJECT:([pubs].[dbo].[M2C_01].[IX_M2C_01_GroupID]), SEEK:
([M2C_01].[GroupID]=1) ORDERED FORWARD



Loop Join – Inner Source

Clustered Index Seek

Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	42
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	10.0
Estimated cost:	0.007697(54%)
Estimated subtree cost:	0.00769

Argument:

OBJECT:([pubs].[dbo].[M2D_01].[PK_M2D_01]), SEEK:([M2D_01].[ID]
]=[M2C_01].[ID]) ORDERED FORWARD

row count is expected matches per row for each row from outer source (rounded down)

I/O and CPU cost is for 1 execute

Number of executes is row count from outer source

Cost is for all executes



Loop Join, cont

Nested Loops/Inner Join

For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.

Physical operation:	Nested Loops
Logical operation:	Inner Join
Estimated row count:	9
Estimated row size:	72
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000042
Estimated number of executes:	1.0
Estimated cost:	0.000042(0%)
Estimated subtree cost:	0.0141

Argument:

OUTER REFERENCES:([M2C_01].[ID])

I/O Cost: 0
CPU Cost
0.00000418
per row



Loop Join Cost Formula

Total cost = Outer Input + Inner Input + Nested Loops cost

Outer input costs: Same as index seek (+ bookmark loop) or table scan costs

Inner input costs: Same for first execute only, more complex formula for multiple executes

Nested Loop/Inner Join costs: 0.00000418 per row



Loop Join Examples

SARG on Outer Source Only

```
SELECT ...  
FROM M2C  
INNER JOIN N1C ON N1C.ID = M2C.CodeID  
WHERE M2C.GroupID = @GroupID
```

OS Index on SARG
IS Index on join condition

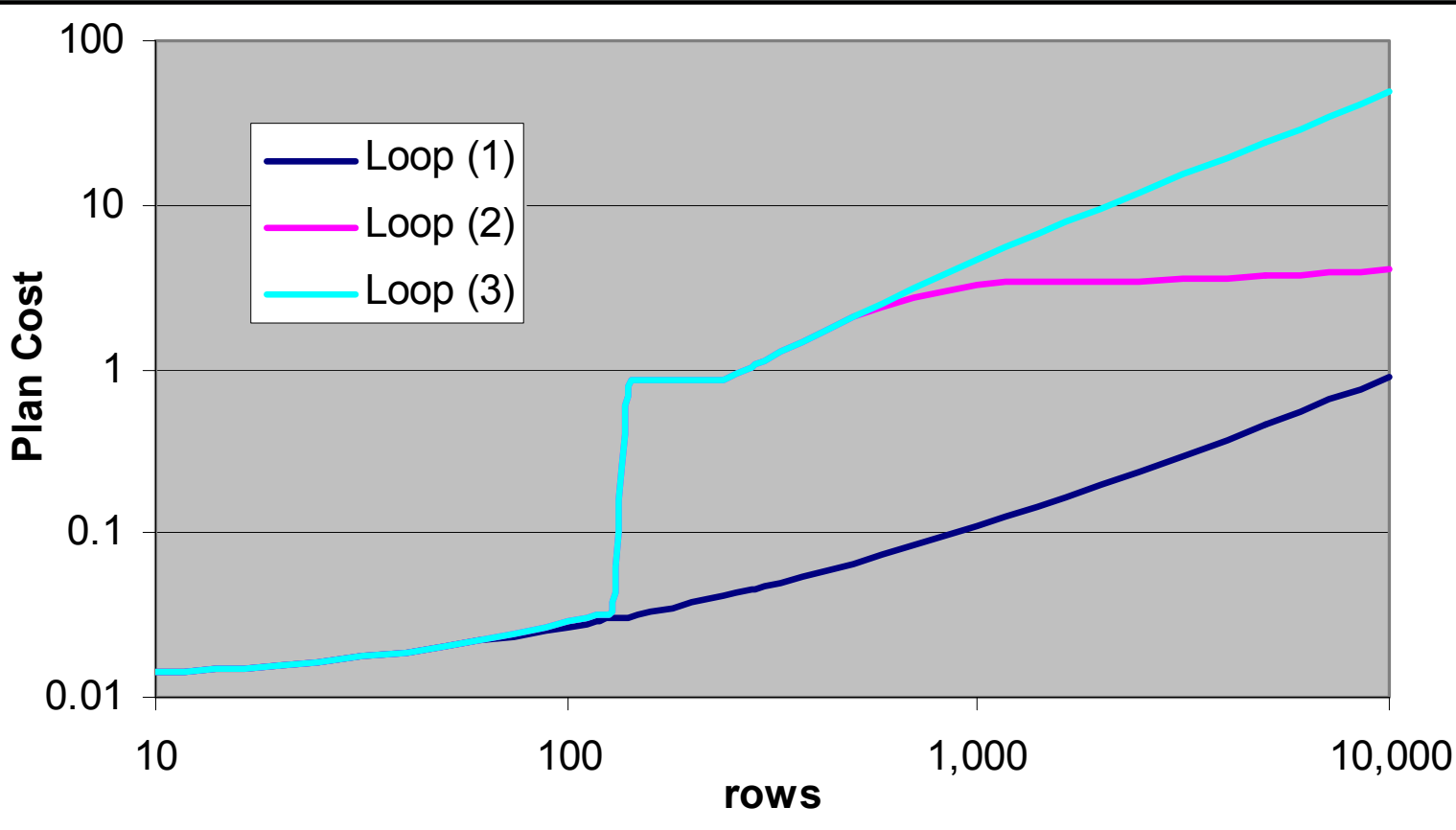
SARG on both sources

```
SELECT ...  
FROM M2C  
INNER JOIN M2D ON M2D.ID = M2C.ID2  
WHERE M2C.GroupID = @GroupID AND M2D.GroupID = @GroupID
```

OS Index on SARG
IS Index on SARG followed by join condition



Loop Join Costs

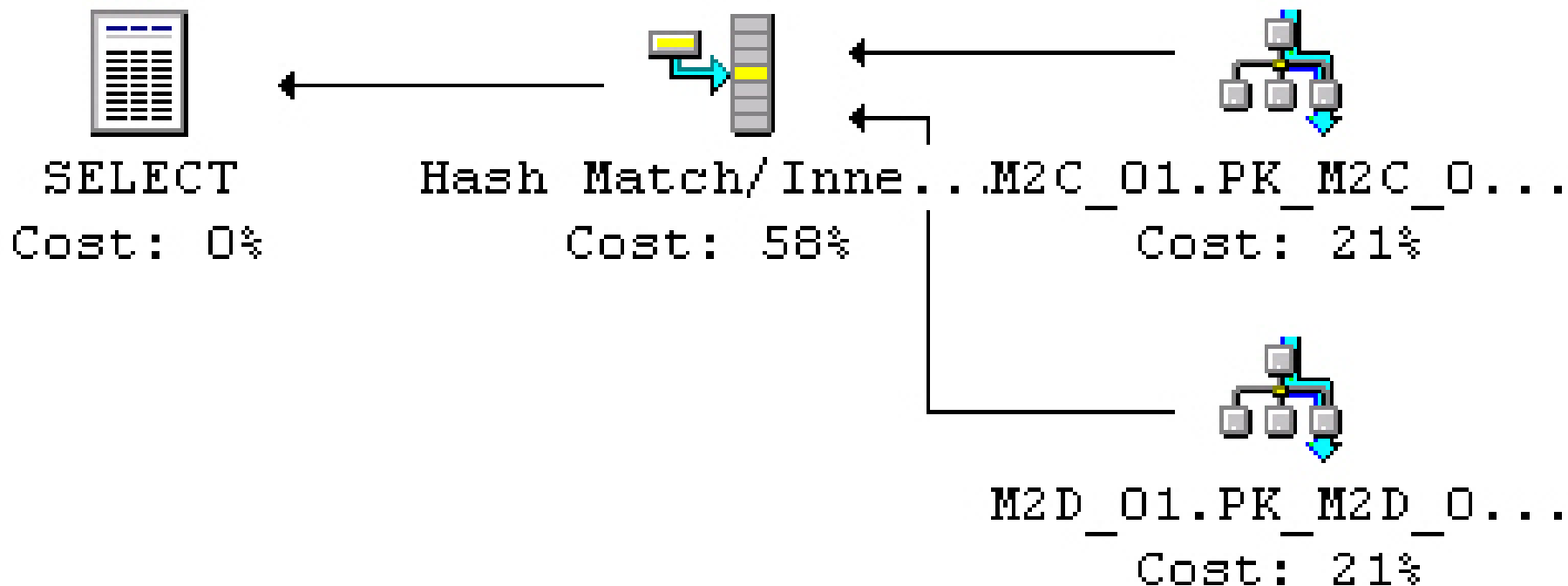


(1) or SARG
on both
sources
and IS
effectively
small

- (1) SARG on OS, IS small table
- (2) SARG on OS, IS not small
- (3) SARG on OS & IS and IS not small



Hash Join



```
SELECT * FROM M2C m INNER HASH JOIN M2D n ON n.ID = m.ID  
WHERE m.GroupID = @Group1 AND n.GroupID = @Group2
```

Hash Join Cost = Outer Source + Inner Source + Hash Match



Hash Join Cost

Outer Source & Inner Source
are index seek or scan

1 execute, 1 or more rows

1:1 match

CPU: 0.01777000

+ ~0.00001885 per row

1:n match

+0.00000523

to 0.00000531 per row in
I.S.

Hash Match/Inner Join

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

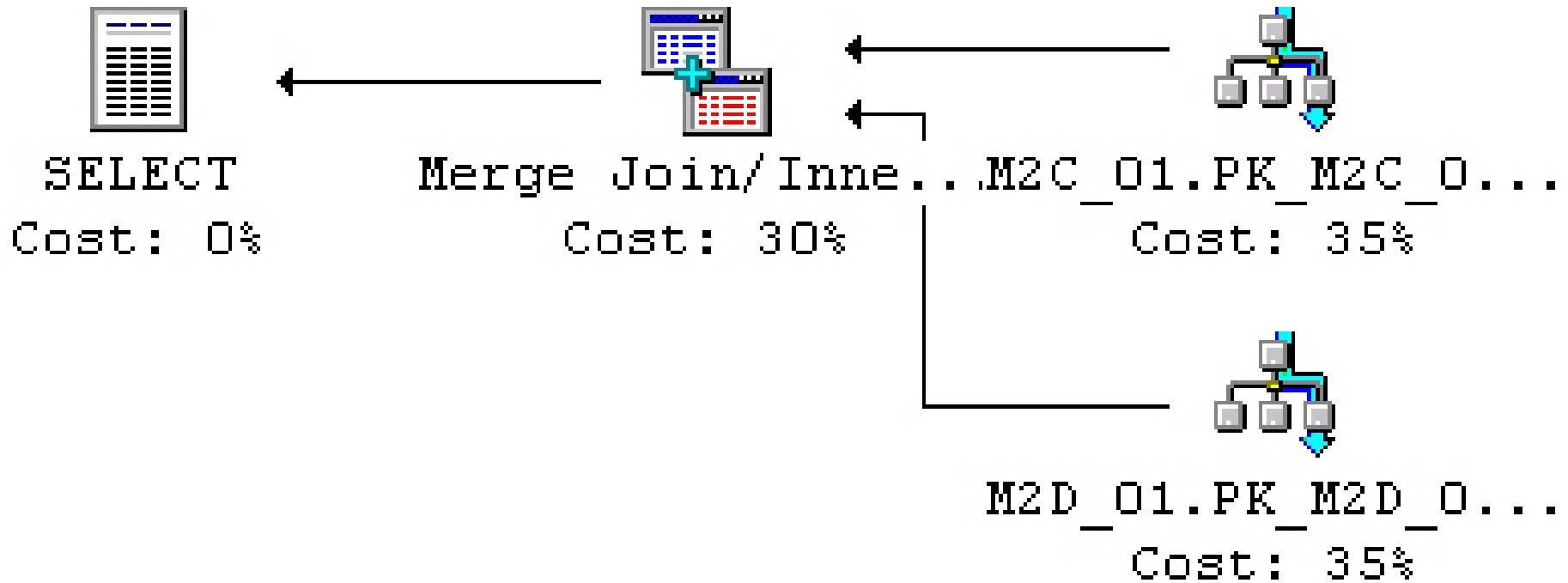
Physical operation:	Hash Match
Logical operation:	Inner Join
Row count:	10
Estimated row size:	25
I/O cost:	0.000000
CPU cost:	0.0179
Number of executes:	1
Cost:	0.017942(58%)
Subtree cost:	0.0307
Estimated row count:	9

Argument:

HASH:([m].[ID])=([n].[ID])



Merge Join



```
SELECT xx FROM M2C m INNER MERGE JOIN M2D n ON n.ID = m.ID  
WHERE m.GroupID = @Group1 AND n.GroupID = @Group2
```

Merge Join Cost = Outer Source + Inner Source + Merge cost

Merge Join Cost

Cost

CPU: 0.0056046

+ 0.00000446/row

Discrepancy:

0.0000030

1:n

additional rows:

+0.000002370 / row

Merge Join/Inner Join

Matching rows from two suitably sorted input tables exploiting their sort order.

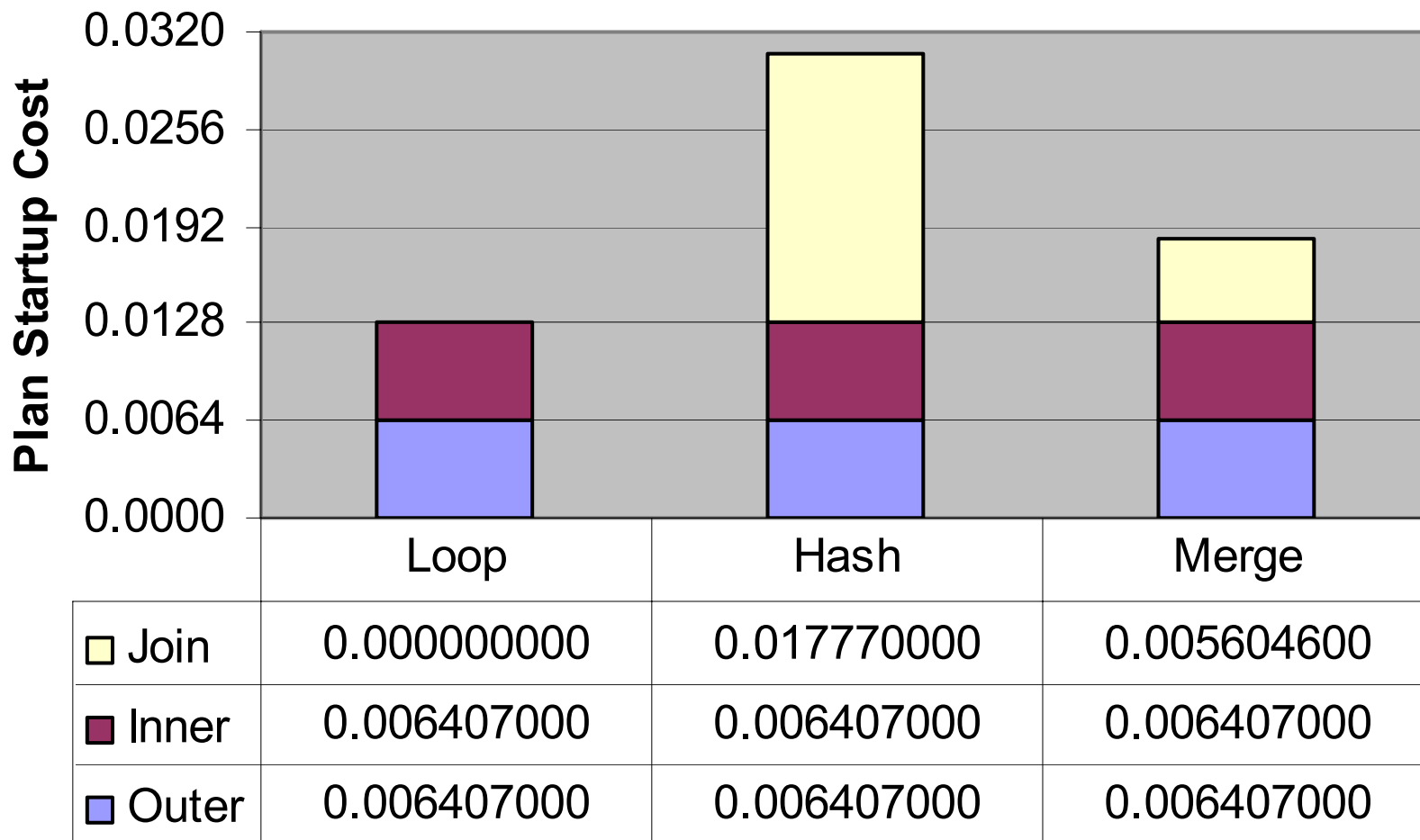
Physical operation:	Merge Join
Logical operation:	Inner Join
Row count:	10
Estimated row size:	47
I/O cost:	0.000000
CPU cost:	0.00564
Number of executes:	1
Cost:	0.005648(31%)
Subtree cost:	0.0184
Estimated row count:	9

Argument:

MERGE:([m].[ID])=([n].[ID]), RESIDUAL:([m].[ID]=
[n].[ID])



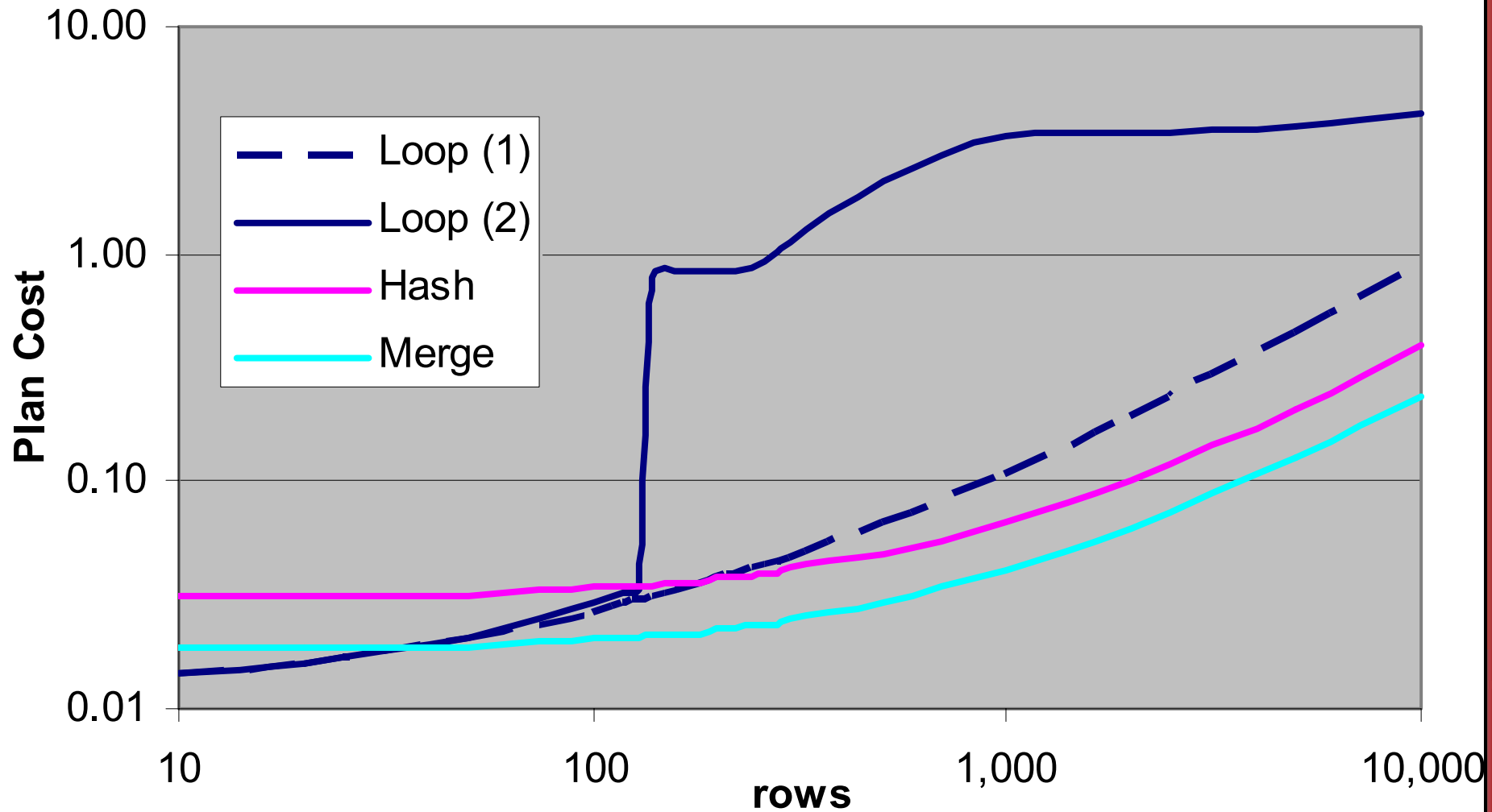
Joins – Base cost



Base cost excludes cost per row

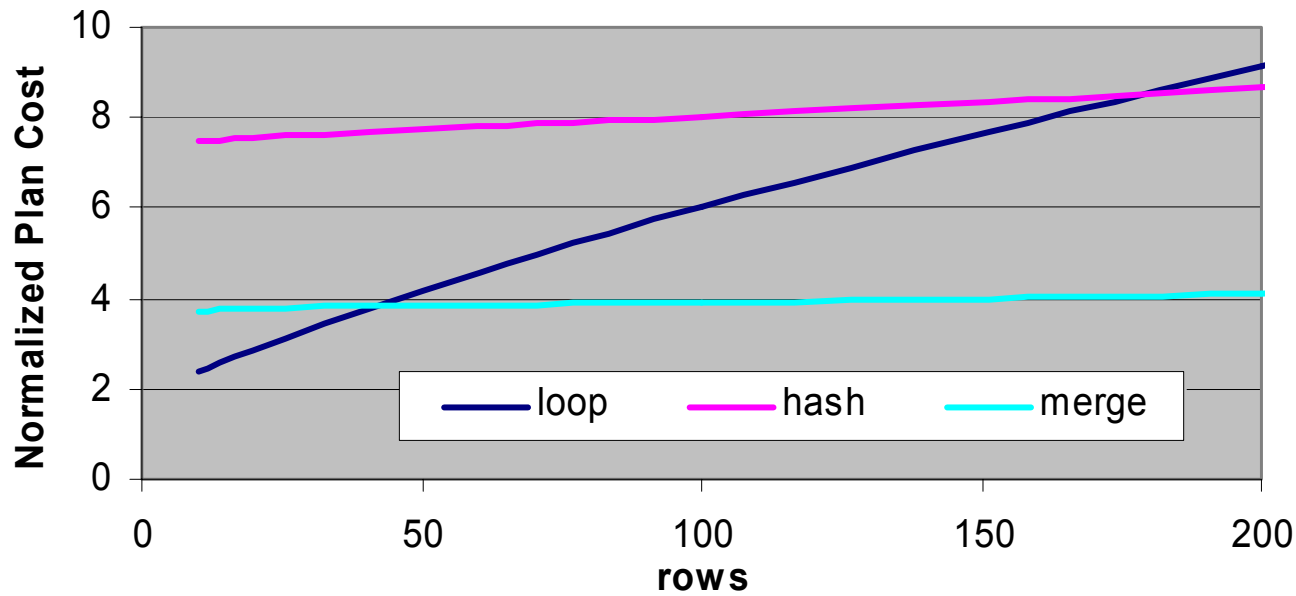
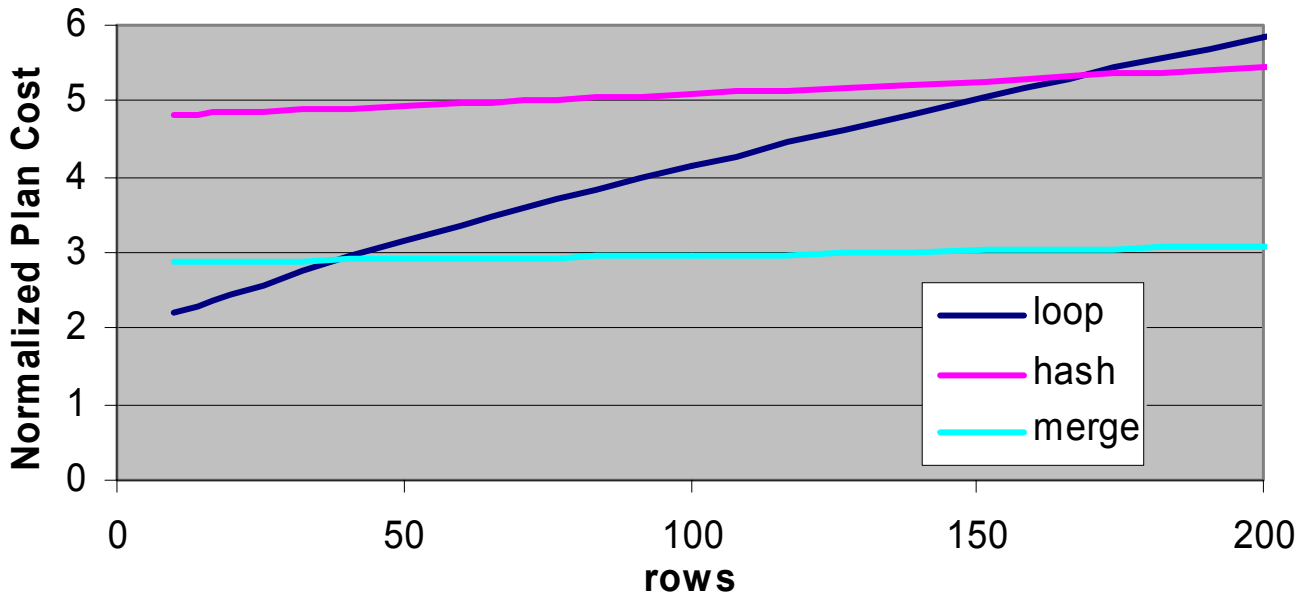


Loop, Hash and Merge Join Costs



Case 2 Loop join

1:1 (\leq 1GB)



Many-to-Many Merge

Merge Join/Inner Join

Matching rows from two suitably sorted input tables exploiting their sort order.

Physical operation:	Merge Join
Logical operation:	Inner Join
Estimated row count:	3
Estimated row size:	51
Estimated I/O cost:	0.00125
Estimated CPU cost:	0.00582
Estimated number of executes:	1.0
Estimated cost:	0.007079(36%)
Estimated subtree cost:	0.0199

Argument:

MANY-TO-MANY MERGE:([m].[ID2])=([n].[ID]), RESIDUAL:([m].[ID2]=[n].[ID])

I/O: 0.000310471 per row

CPU: 0.0056046 + 0.00004908 per row



Merge with Sort

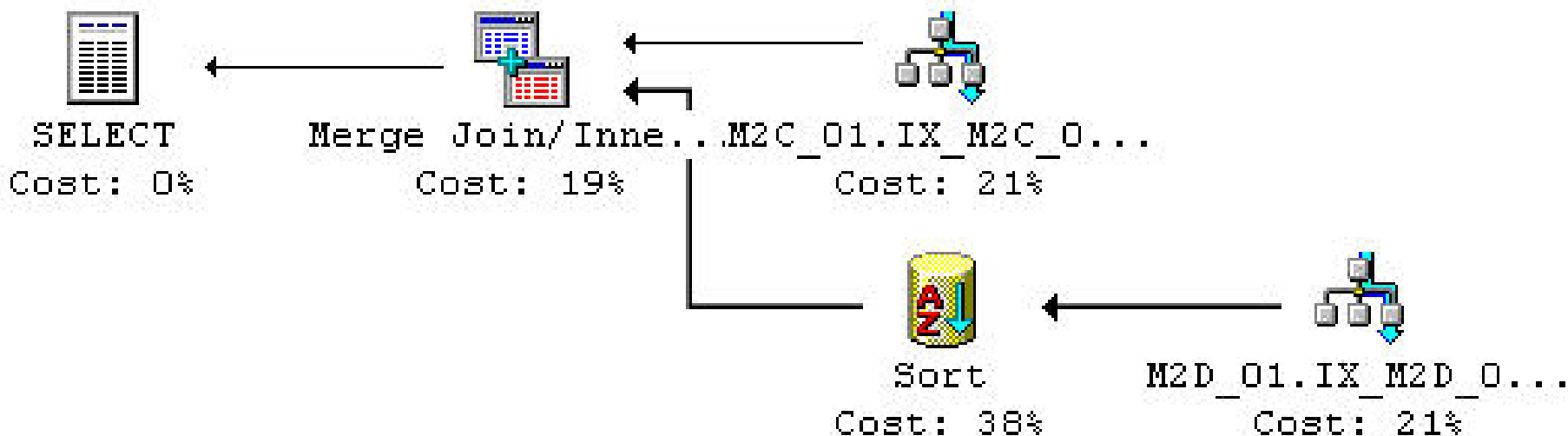
Sort

Sorting the input.

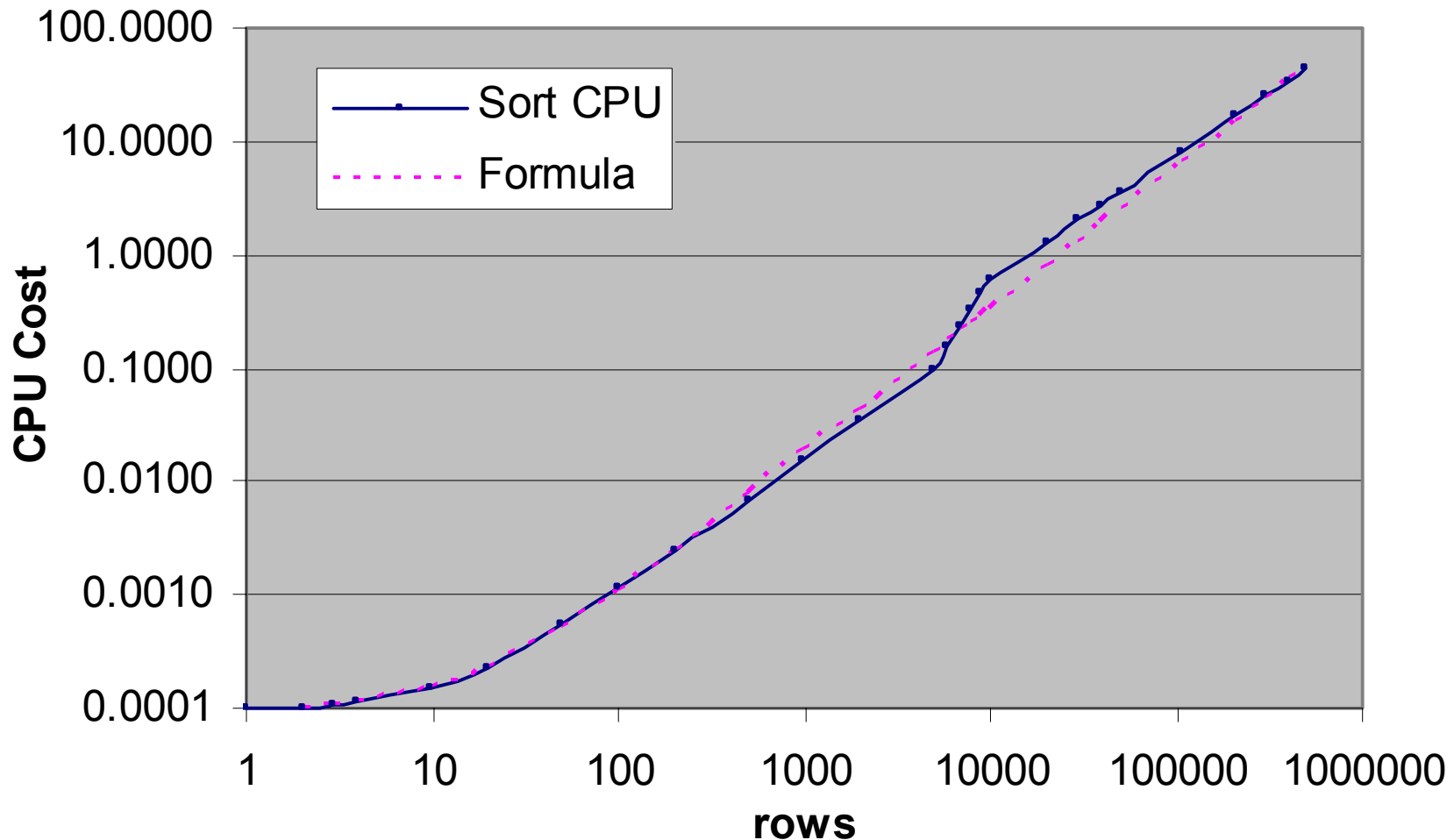
Physical operation:	Sort
Logical operation:	Sort
Estimated row count:	10
Estimated row size:	16
Estimated I/O cost:	0.0112
Estimated CPU cost:	0.000152
Estimated number of executes:	1.0
Estimated cost:	0.011413(38%)
Estimated subtree cost:	0.0178

Argument:

ORDER BY:([n].[ID] ASC)



Sort Cost cont.

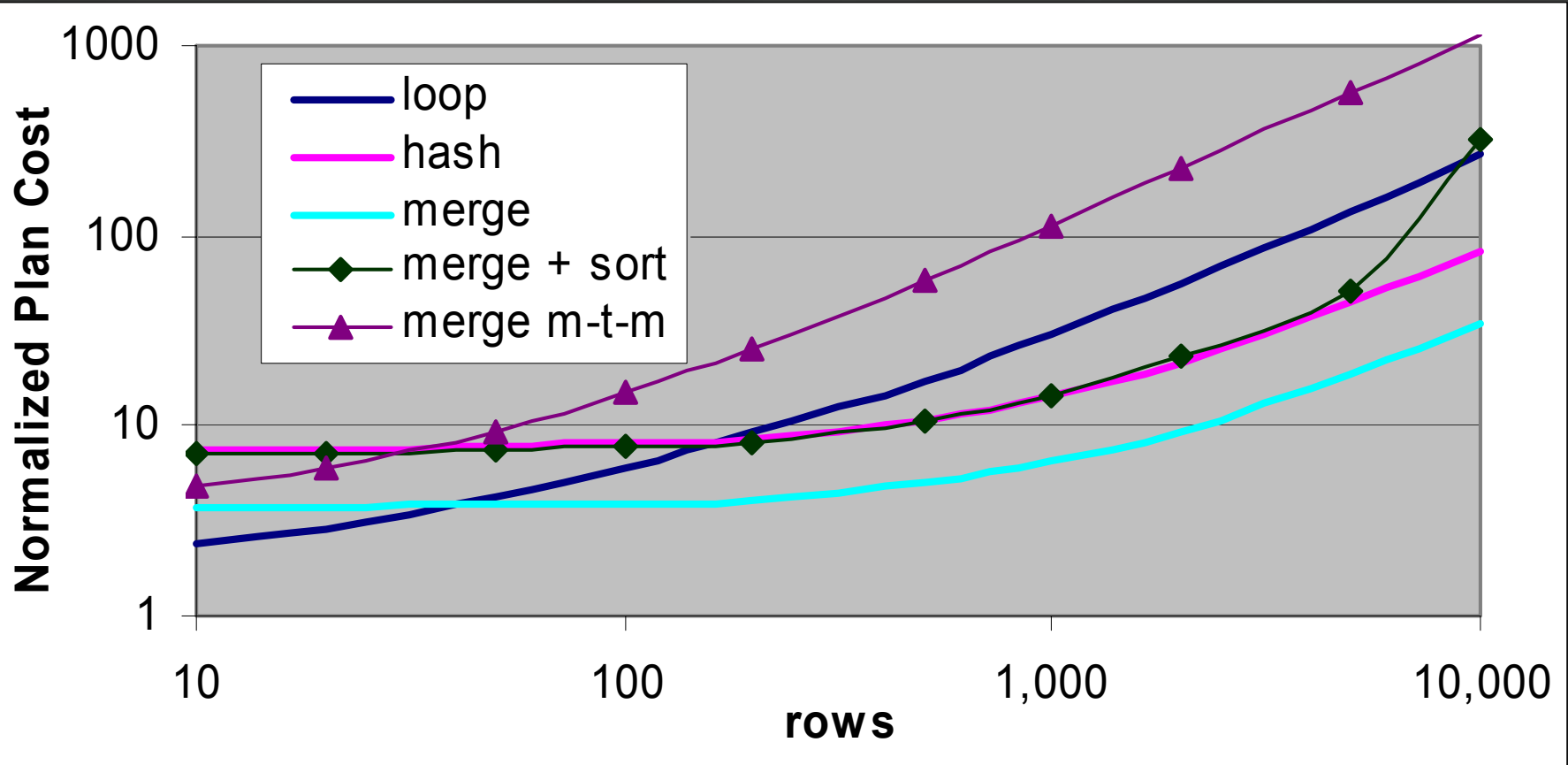


I/O: 0.011261261

CPU: $0.000100079 + 0.00000305849 * (\text{rows} - 1)^{1.26}$



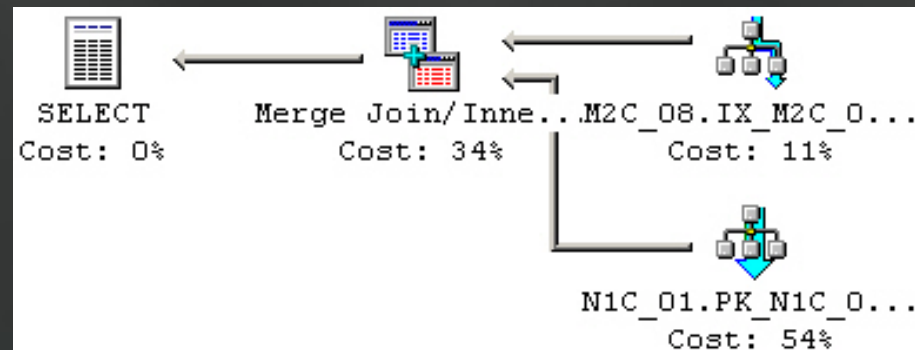
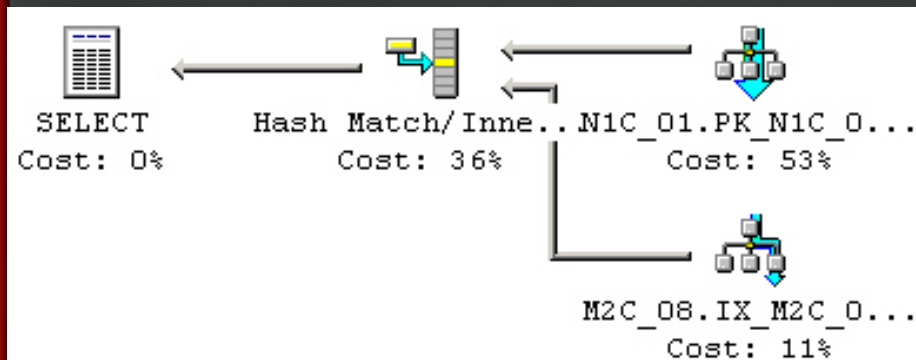
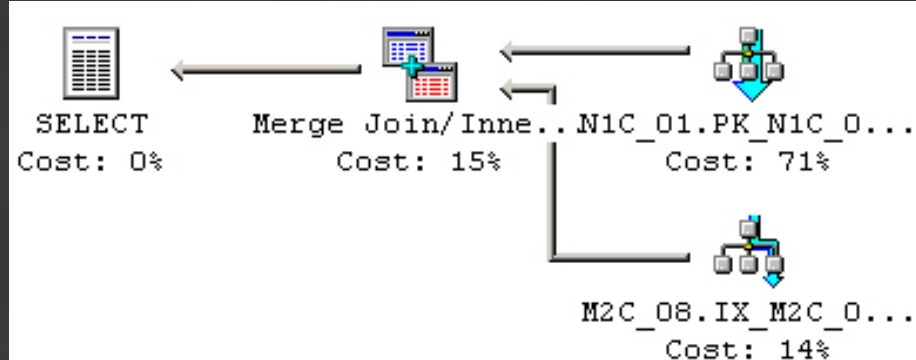
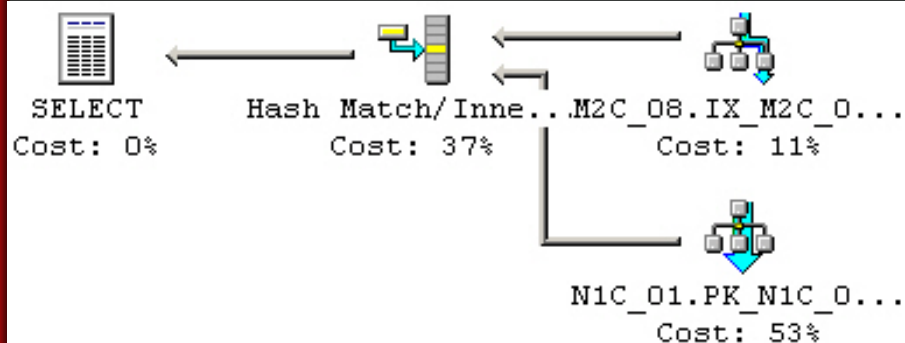
Join Costs Compared



Merge + Sort slightly less expensive than hash join at lower row counts



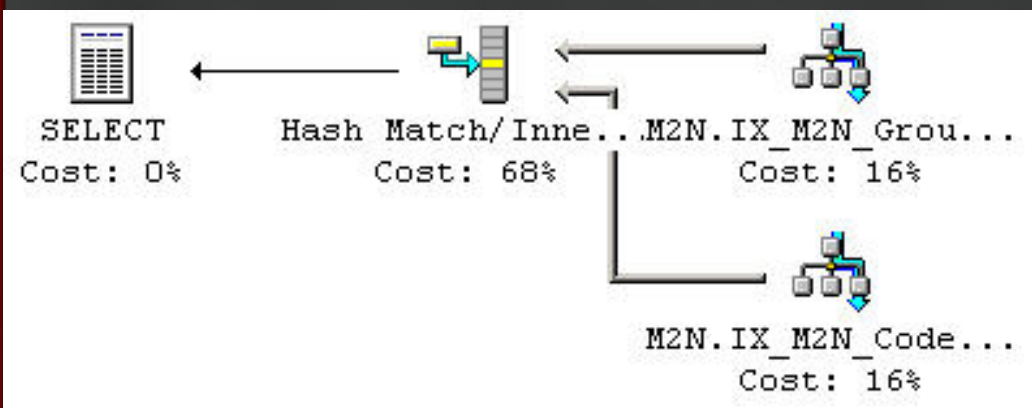
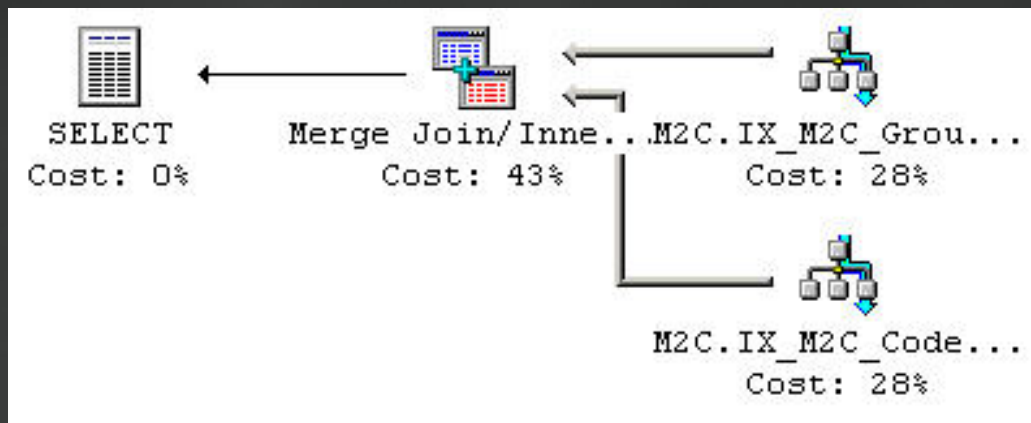
More Hash & Merge Joins



Index Intersection

SELECT xx FROM M2C WHERE GroupID = @Group AND CodeID = @Code

Table M2x,
Index on GroupID
Index on CodeID



SELECT xx FROM M2C a INNER JOIN M2C b ON b.ID = a.ID

WHERE a.GroupID = @Group AND b.CodeID = @Code

Merge Join cost formula different than previously discussed

Execution Plan Costs

	I/O	CPU	Total
Index Seek			
≤ 1GB	0.006328500	0.0000796	0.006408100
> 1GB	0.003203425	0.0000796	0.003283025
Additional page	0.00074074/p		
Additional rows		0.00000110/r	
Bookmark Lookup			
≤ 1GB	0.0062500	0.0000011	0.0062511
> 1GB	0.0031249	0.0000011	0.0031260
Table Scan			
Base	0.0375785	0.0000785	
Additional page	0.00074074/p		
Additional row		0.0000011/r	



Logical IO count

Example: Index Depth 2, rows per page: 100

I/O per additional row

Bookmark Lookup (Heap) 1

Bookmark Lookup (Clustered) 2

Loop Join (IS) 2

Very little relation between IO count and plan cost for different component operations

IO count comparisons more relevant for similar operations

I/O per addition 100 rows

Index Seek 1

Hash & Merge join 2



Accurate Performance Testing

- Execution Plan - match
 - Raw size of DB not as important:
 - 1M customers actual, 10K test
 - Cardinality more important
 - 1 Customer – 10 orders – 10 order items per order
- Statistics & actual data queried
 - Statistics could be accurate but actual queries favors different distribution



Plan Cost Formula Summary

- Plan costs do not include RPC cost
- Plan costs are a model
- Index seek independent of index depth
- Bookmark L/U independent of table org.
- Conditions do not influence cost
- Costs are not influenced by lock hints
- Populate test DB with accurate cardinality
- No claims made about actual query costs



Links

www.sql-server-performance.com/joe_chang.asp

www.perftuning.com

SQL Server Quantitative Performance Analysis

Server System Architecture

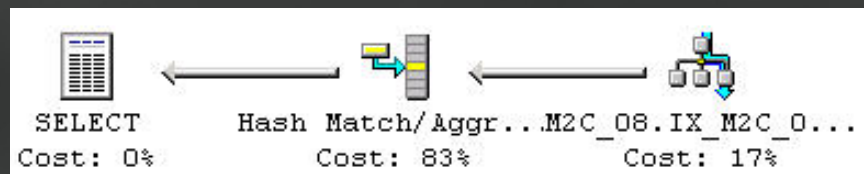
Processor Performance

Direct Connect Gigabit Networking

jchang@perftuning.com, jchang6@yahoo.com



Aggregates



Stream Aggregate/Aggregate

Computing summary values for groups of rows in a suitably sorted stream.

Physical operation:	Stream Aggregate
Logical operation:	Aggregate
Estimated row count:	2,000
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.0149
Estimated number of executes:	1.0
Estimated cost:	0.014900(56%)
Estimated subtree cost:	0.0264

Argument:

GROUP BY:([M2C_08].[ID2]) [Expr1002]=SUM([M2C_08].[randDecimal])

Hash Match/Aggregate

Insert each input row into a hash table, grouping on the GROUP BY columns and computing aggregate expressions.

Physical operation:	Hash Match
Logical operation:	Aggregate
Estimated row count:	2,000
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.0554
Estimated number of executes:	1.0
Estimated cost:	0.055452(83%)
Estimated subtree cost:	0.0670

Argument:

HASH:([M2C_08].[ID2]) [Expr1002]=SUM([M2C_08].[randDecimal])

CPU

Cost per result row: 0.000007450/row

CPU Cost per result row:

0.01777 + 0.0000188

