



BRENT OZAR
UNLIMITED®

T-SQL Anti-patterns

Our Agenda

Low Functioning Functions

Nesting Habits of the Wild Query

Low Functioning Functions

Scalar Functions

A function that returns a **single** value

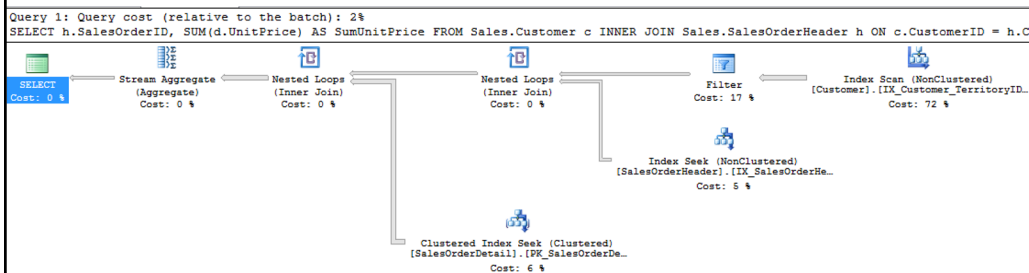
What could go wrong?

```
SELECT  h.SalesOrderID,  
        SUM(d.UnitPrice) AS SumUnitPrice  
FROM    Sales.Customer c  
        INNER JOIN Sales.SalesOrderHeader h  
            ON c.CustomerID = h.CustomerID  
        INNER JOIN Sales.SalesOrderDetail d  
            ON h.SalesOrderID = d.SalesOrderID  
WHERE   dbo.CheckPermissions(c.CustomerID) = 1  
GROUP BY h.SalesOrderID;
```

What could go wrong?

```
SELECT  h.SalesOrderID,  
        SUM(d.UnitPrice) AS SumUnitPrice  
FROM    Sales.Customer c  
        INNER JOIN Sales.SalesOrderHeader h  
        ON c.CustomerID = h.CustomerID  
        INNER JOIN Sales.SalesOrderDetail d  
        ON h.SalesOrderID = d.SalesOrderID  
WHERE   dbo.CheckPermissions(c.CustomerID) = 1  
GROUP BY h.SalesOrderID;
```

What could go wrong?



What could go wrong?

Table 'SalesOrderDetail'.
Scan count 861, logical reads 2765

Table 'SalesOrderHeader'.
Scan count 121, logical reads 243

Table 'Customer'.
Scan count 1, logical reads 37



© SQLintersection. All rights reserved.
<http://www.SQLintersection.com>

What could go wrong?

Table 'SalesOrderDetail'.
Scan count 861, logical reads 2765

Table 'SalesOrderHeader'.
Scan count 121, logical reads 243

Table 'Customer'.
Scan count 1, logical



Wait, what?

CheckPermissions forces row by row execution.

- Increases access to underlying tables.
- No easy way to tell without in-depth investigation.

This is going to be hard to find

- Average CPU per execution is 75 microseconds.
- 6 reads per execution.

Identifying the Problem: Interactive Means

Service Side Trace

- Requires the problem to be observed.
- This issue won't show up in most traces: short duration, low reads.

Extended Events

- Finer grained control.
- Same problems as a trace.

Identifying the Problem: Historical Means

Plan Cache

- Historical information.
- Lightweight querying.
- We don't have to prepare in advance to capture problems.

I like this approach

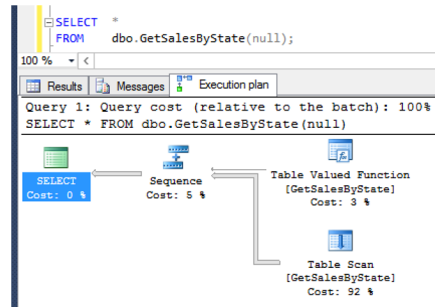
- It's lightweight.
- It lets me find problems I didn't think to watch for.

Demo!

What about TVFs?

A function
that returns
a **tabular** value

A TVF in Action!



A TVF in Action!

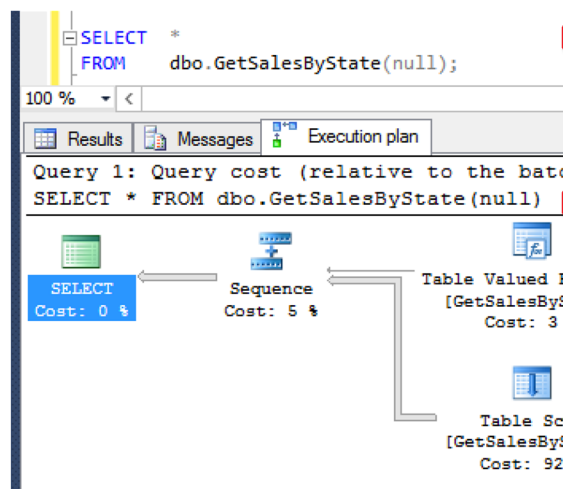


Table Scan	
Scan rows from a table.	
Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	6669
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.003392 (92%)
Estimated CPU Cost	0.000267
Estimated Subtree Cost	0.003392
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	100
Estimated Row Size	87 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3
Object	
[AdventureWorks2012_CS].[dbo].[GetSalesByState]	
Output List	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].MonthFirstDate,	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].StateProvinceName,	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].TotalSales	

A TVF in Action! Now with different parameters

```
SELECT *  
FROM dbo.GetSalesByState('Washington')
```

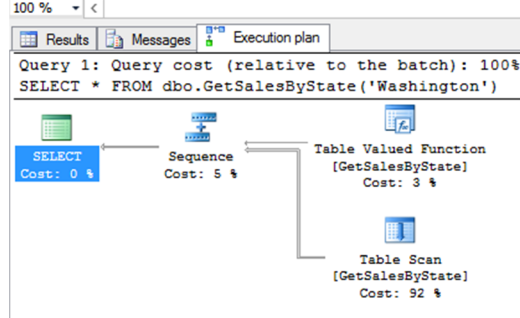


Table Scan	
Scan rows from a table.	
Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	37
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.003392 (92%)
Estimated CPU Cost	0.000267
Estimated Subtree Cost	0.003392
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	100
Estimated Row Size	87 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3
Object	
[AdventureWorks2012_CS].[dbo].[GetSalesByState]	
Output List	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].MonthFirstDate,	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].StateProvinceName,	
[AdventureWorks2012_CS].[dbo].[GetSalesByState].TotalSales	

TVFs Give Bad Estimates

Regardless of parameter, we get the same estimate.

If you only have a few rows, don't worry.

If you have a lot of rows:

- Dump rows into temporary tables
- Re-write the TVF to inline

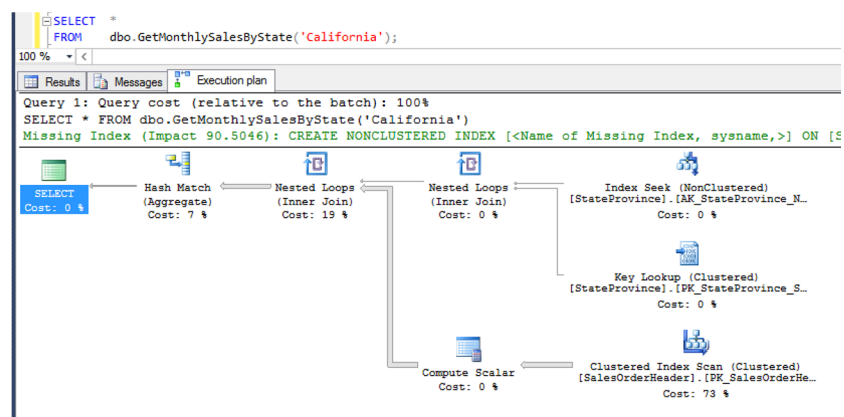
An Inline TVF

A single statement TVF can be inlined by the optimizer.

When inlined:

- The body of the TVF is re-written into the query.
- We get good estimates.
- Things can run faster.
- Everyone wins, yay!

An Inlined TVF



That's better

Nested Loops	
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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	6224
Actual Number of Batches	0
Estimated Operator Cost	0.1409626 (19%)
Estimated I/O Cost	0
Estimated CPU Cost	0.131524
Estimated Subtree Cost	0.69524
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	3074.52
Estimated Row Size	77 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	1
Predicate	
[AdventureWorks2012_CS].[Person].[StateProvince].[TerritoryID] as [sp].[TerritoryID] = [AdventureWorks2012_CS].[Sales].[SalesOrderHeader].[TerritoryID] as [soh].[TerritoryID]	
Output List	
[AdventureWorks2012_CS].[Sales].[SalesOrderHeader].SubTotal, [AdventureWorks2012_CS].[Person].[StateProvince].Name, Expr1003	

You can't even
count on the bad
TVF estimates to
be right.

Check this out!

A simple string splitting trick...

```
DECLARE @numbers AS VARCHAR(MAX) =
    '1,2,3,4,5,6';

DECLARE @words AS VARCHAR(MAX) =
    'one,two,three,four,five,six';

DECLARE @spanish AS VARCHAR(MAX) =
    'uno,dos,tres,quatro,cinco,seis';

SELECT  soh.PurchaseOrderNumber ,
        sod.CarrierTrackingNumber ,
        w.element AS EnglishQty ,
        s.element AS SpanishQty
FROM    Sales.SalesOrderHeader AS soh
        JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID =
        sod.SalesOrderID
        JOIN dbo.fn_split(@numbers) AS o ON sod.OrderQty = o.element
        JOIN dbo.fn_split(@words) AS w ON o.pos = w.pos
        JOIN dbo.fn_split(@spanish) AS s ON w.pos = s.pos ;
```

A Slight Estimation Problem

- It turns out that SQL Server doesn't always make great guesses.
- In this limited case, it doesn't matter. In big queries, this can create terrible execution plans

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, bytes on disk 0
 Table 'Worktable'. Scan count 3, logical reads 346483, physical reads 0, read-ahead reads 0, bytes on disk 0
 Table 'nums'. Scan count 5, logical reads 14, physical reads 0, read-ahead reads 0, bytes on disk 0
 Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead reads 0, bytes on disk 0

Nested Loops	
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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	6
Actual Number of Batches	0
Estimated Operator Cost	69690951.2 (92%)
Estimated I/O Cost	0
Estimated CPU Cost	31732900
Estimated Subtree Cost	72466000
Estimated Number of Executions	1
Estimated Number of Rows	1518320000000
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

```
(1)=[AdventureWorks2012].[dbo].[nums].[n]-len(replace
(substring([@words],(1),[AdventureWorks2012].[dbo].[nums].[n]),',',CONVERT_IMPLICIT(varchar(max),'0')))+(1))
```

Output List

```
[AdventureWorks2012].[dbo].[nums].n,
[AdventureWorks2012].[dbo].[nums].n,
[AdventureWorks2012].[dbo].[nums].n
```

Moving Toward More Accurate Estimates

- Avoid joining to TVFs wherever possible
- Load data into temp tables
 - Better estimated and actual plans
 - Reduced CPU and I/O

```
CREATE TABLE #words
(
    pos INT ,
    ordinal VARCHAR(10) ,
    english VARCHAR(50) ,
    spanish VARCHAR(50)
);

-- Step one
INSERT INTO #words
( pos ,
  ordinal
)
SELECT pos ,
       element
FROM   dbo.fn_split(@numbers)

-- Step two
UPDATE #words
SET    english = element
FROM   fn_split(@words) AS w
WHERE  w.pos = #words.pos

-- Step three
UPDATE #words
SET    spanish = element
FROM   fn_split(@spanish) AS s
WHERE  s.pos = #words.pos ;
```

Functions in SQL Server

Pros:

- Encapsulate logic
- Re-usable

Cons:

- Provide inaccurate estimates
- Hide I/O requirements
- Obscure the cost of operations

The Catch All

Raise your hand if
you've seen a query
where *every*
parameter was
optional.

COALESCE

```
SELECT *  
FROM dbo.BestViewEver  
WHERE col_a = COALESCE(@a, col_a)  
      AND col_b = COALESCE(@b, col_b)  
      AND col_c = COALESCE(@c, col_c)
```

**But I'll just use an
OR, you say.**

IS NULL and OR

```
SELECT *  
FROM dbo.BestViewEver  
WHERE (@a IS NULL OR col_a = @a)  
      AND (@b IS NULL OR col_b = @b)  
      AND (@c IS NULL or col_c = @c)
```

It's all the same!

SQL Server is bad at short circuiting logical comparison.

- Both comparisons will be executed every time.
- No, you can't trick it.

The worst part:

SQL Server won't use indexes.

SARGability

The example is said to be non-SARG-able.

That's a fancy way of saying it won't use an index.

Basically, by hiding a column value we've messed with optimization.

Other losing patterns

WHERE LEFT(LastName,3) = ...

WHERE UPPER(string) = ...

WHERE LastName LIKE '%smith%'

WHERE YEAR(MyDateField) = '1973'

WHERE VARCHAR = @NVARCHAR

Learn more:

BrentOzar.com/go/sargable

What can we do?

Use dynamic SQL

Build a string with only the parameters we need.

Run that through sp_executesql

Yes, this makes development harder, but it makes SQL Server faster.

Re-write queries

Find creative ways to re-write queries.

Move functions to variables and parameters.

Find logical equivalents.

- `YEAR(col) = 1973` becomes
`col >= '1973-01-01 AND col < '1974-01-01'`

Nesting Habits of the Wild Query

Correlated Subqueries

What is a correlated subquery?

A subquery that uses
values from an outer query.

Problems?

Extra reads and CPU time

- SQL Server may run the subquery once per row in the outer query.

Sudden onset

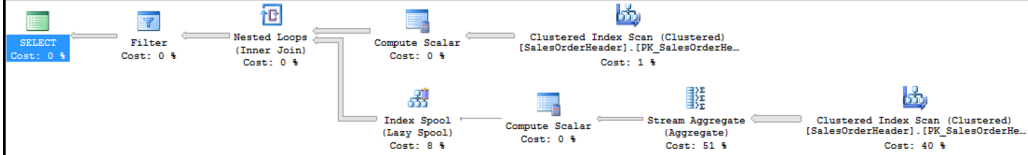
- Great in development, terrible in production.
- Difficult to identify without careful observation.

Our awful query

```
SELECT  SalesOrderID, SubTotal,
        OrderYear, OrderMonth
FROM    ( SELECT  SalesOrderID, SubTotal,
                YEAR(OrderDate) AS OrderYear,
                MONTH(OrderDate) AS OrderMonth
        FROM    Sales.SalesOrderHeader
        ) AS x
WHERE   x.SubTotal >
        ( SELECT  AVG(soh.SubTotal)
        FROM    Sales.SalesOrderHeader soh
        WHERE   YEAR(soh.OrderDate) =
                x.OrderYear
                AND MONTH(soh.OrderDate) =
                x.OrderMonth
        )
```

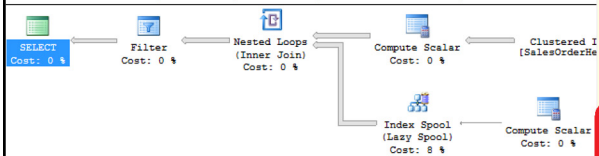
What's going on here?

Query 1: Query cost (relative to the batch): 100%
SELECT SalesOrderID, SubTotal, OrderYear, OrderMonth FROM (SELECT SalesOrderID, SubTotal, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth FROM SalesOrderHeader)



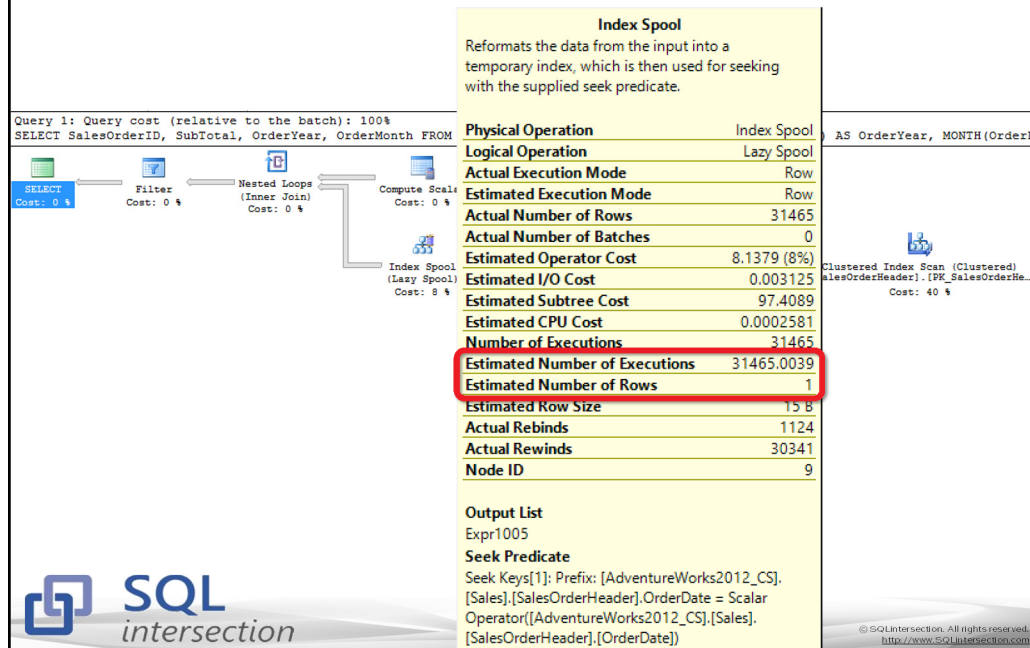
Scan SalesOrderHeader

Query 1: Query cost (relative to the batch): 100%
SELECT SalesOrderID, SubTotal, OrderYear, OrderMonth FROM (SELECT SalesOrderID, SubTotal, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth FROM SalesOrderHeader)



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 (1%)
Estimated CPU Cost	0.0347685
Estimated Subtree Cost	0.54456
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	31465
Estimated Row Size	278
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3
Object	
[AdventureWorks2012_CS].[Sales].[SalesOrderHeader].	
[PK_SalesOrderHeader_SalesOrderID]	
Output List	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].SalesOrderID,	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].OrderDate, [AdventureWorks2012_CS].	
[Sales].[SalesOrderHeader].SubTotal	

What's wrong here?



What's wrong here?

Table 'Worktable'.
Scan count 32589, logical reads 69792

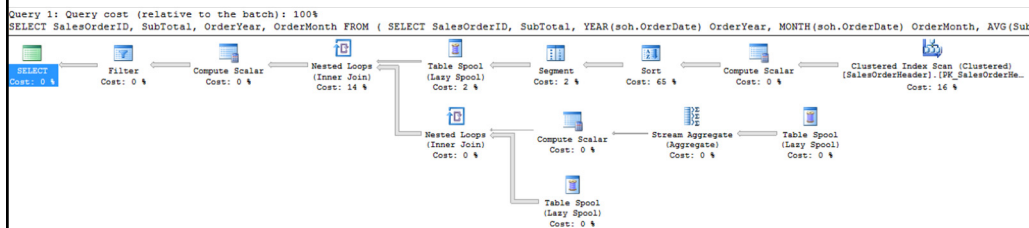
Table 'SalesOrderHeader'.
Scan count 1125, logical reads 775125

We read **6,759,336KB** to perform this query

A re-write!

```
SELECT SalesOrderID, SubTotal,
       OrderYear, OrderMonth
FROM   (
        SELECT SalesOrderID,
               SubTotal,
               YEAR(soh.OrderDate) OrderYear,
               MONTH(soh.OrderDate) OrderMonth,
               AVG(SubTotal) OVER
               (PARTITION BY YEAR(soh.OrderDate),
                           MONTH(soh.OrderDate)
                ) AS AverageSale
        FROM   Sales.SalesOrderHeader soh
        ) AS x
WHERE  SubTotal > AverageSale ;
```

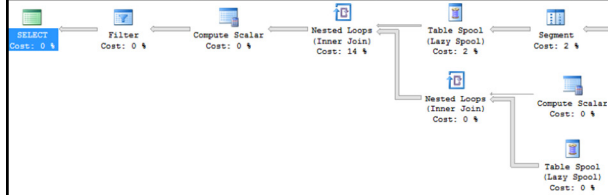
What happens now?



What happens now?

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

SELECT SalesOrderID, SubTotal, OrderYear, OrderMonth FROM (SELECT SalesOrderID, SubTotal, YEAR

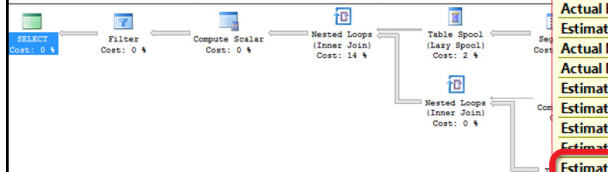


Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	31465
Actual Number of Batches	0
Estimated I/O Cost	0.509792
Estimated Operator Cost	0.54456 (16%)
Estimated CPU Cost	0.0347685
Estimated Subtree Cost	0.54456
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	31465
Estimated Row Size	27 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	7
Object	
[AdventureWorks2012_CS].[Sales].[SalesOrderHeader].	
[PK_SalesOrderHeader_SalesOrderID] [soh]	
Output List	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].SalesOrderID,	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].OrderDate, [AdventureWorks2012_CS].	
[Sales].[SalesOrderHeader].SubTotal	

Everything runs once!

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

SELECT SalesOrderID, SubTotal, OrderYear, OrderMonth FROM (SELECT SalesOrderID, SubTo



Segment	
Segment.	
Physical Operation	Segment
Logical Operation	Segment
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	31465
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	0.05939 (2%)
Estimated CPU Cost	0.0593869
Estimated Subtree Cost	2.77627
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	31465
Estimated Row Size	35 B
Actual Rebinds	0
Actual Rewinds	0
Segment Column	Segment1006
Node ID	4
Output List	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].SalesOrderID,	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].OrderDate,	
[AdventureWorks2012_CS].[Sales].	
[SalesOrderHeader].SubTotal, Expr1001, Expr1002,	
Segment1006	
Group By	
Expr1001, Expr1002	

What's going on here?

Table 'Worktable'.

Scan count 3, logical reads 64141

Table 'SalesOrderHeader'.

Scan count 1, logical reads 689

We've reduced the amount of I/O to **518,640KB**

Simple Rewrites Save Time

By re-writing the query slightly differently:

- SQL Server produces a better plan.
- We read the table only once.
- The code is, arguably, more maintainable.

Most importantly:

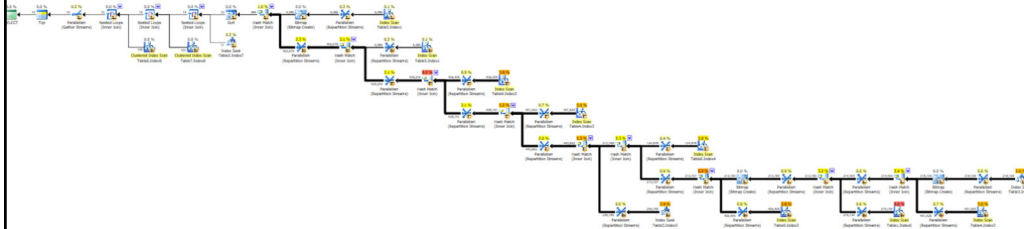
- We eliminate multiple executions of one code fragment.

Nested Views

It's so simple!

```
SELECT TOP 10 *  
FROM [dbo].[ILookReallySimple]  
WHERE Column10 = @Column10Value  
      OR Column11 = @Column11Value  
ORDER BY ADateColumn DESC
```

... What?



The problem with nested views

It's not directly a problem with nested views.

It's a problem of query complexity.

- 2 tables – 2 ways to join
- 3 tables – 9 ways to join
- 4 tables – 64 ways to join
- ...
- 10 tables – 1,000,000,000 ways to join

This is just the total number of possible join combinations and doesn't include optimizer smarts

At some point the optimizer will give up and return the best plan it has **right now**.

Finding Exhausted Optimizers

Optimization Level	FULL
Parameter List	Column14, Column13
Physical Operation	
QueryHash	0xF71AAAB5149FC515
QueryPlanHash	0x4D9C241222B6733D
Reason For Early Termination C	Time Out

Getting Rid of Nested Views

Break out the query.

Re-write using temp tables.

- Insert into temp table from view.
- Join between temp tables.
- May appear more complex, almost always faster.

Only use the tables you need.

- Track down the columns you need.
- Re-write your query to avoid views.
- May not always be possible, depending on permissions.

Why no working example?

We didn't produce an example in AdventureWorks that did the wrong thing.

In fact, in trying to "do the wrong thing" we made our queries faster.

Want to prove it out for yourself?

- Open up SSMS and get cracking!
- Look in your own code base.
- This is really time consuming if you're starting from scratch.

What Can We Do?

Patterns and Anti-Patterns

Understand the data access patterns of your code.

Write code for convenience.

Tune for performance.