Breaking Bad Habits:

Solutions for Common Query Antipatterns

Jeff Iannucci

Who in the world is Jeff Iannucci?

StraightPath Solutions

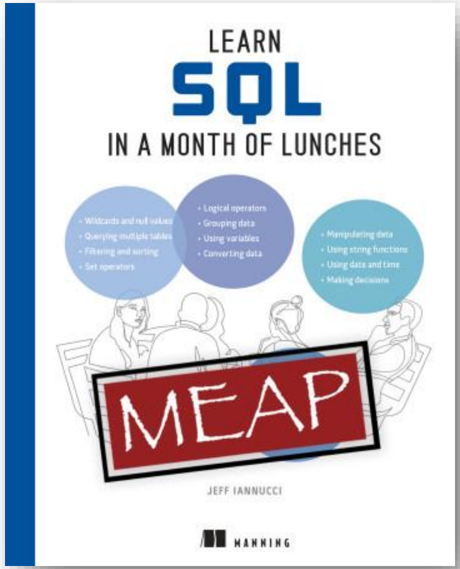
Consultant at Straight Path Solutions

PLURALSIGHT

Content Author at Pluralsight

MANNING

Author of “Learn SQL in a Month of Lunches”



Session Goals

WHAT are some T-SQL antipatterns?

WHY are they antipatterns?

HOW can we correct them?

WARNING!

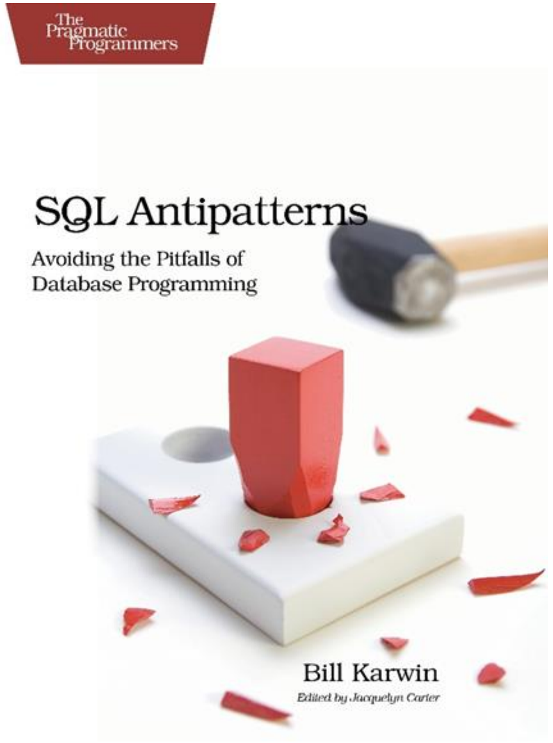
Some index discussion!

Execution plans ahead!

...and more dog pictures!

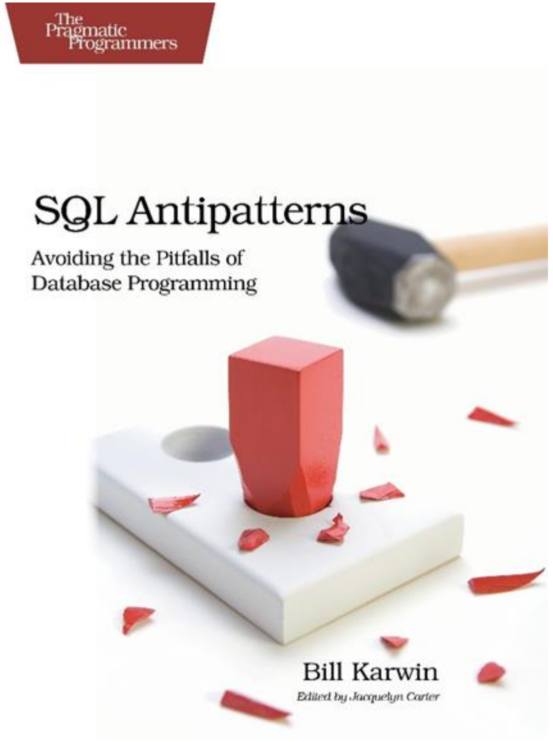
(Image courtesy of designerpoint via pixabay)

What is an antipattern?


The Pragmatic Programmers
SQL Antipatterns
Avoiding the Pitfalls of Database Programming
Bill Karwin
Edited by Jacquelyn Carter

"A technique
that is intended
to solve a problem but
often leads to other
problems."
-Bill Karwin

What is an antipattern?


The Pragmatic Programmers
SQL Antipatterns
Avoiding the Pitfalls of Database Programming
Bill Karwin
Edited by Jacquelyn Carter

1. Scenario
2. Example
3. Name
4. Reasons
5. Solution

About the antipatterns we will discuss

They are all common T-SQL solutions.

They all return a correct result set.

…but they perform unnecessary work.

Who is using antipatterns?

“Boss”

ADVENTURE WORKS cycles

“Junior”

Selection antipatterns

(Image courtesy of TheDigitalWay via pixabay)

Boss request #1

Hey!

Let’s find anyone

Named “Barker”

Because maybe

We are related!

Junior's query #1

```
SELECT *
FROM Person.Person
WHERE LastName = 'Barker'
```

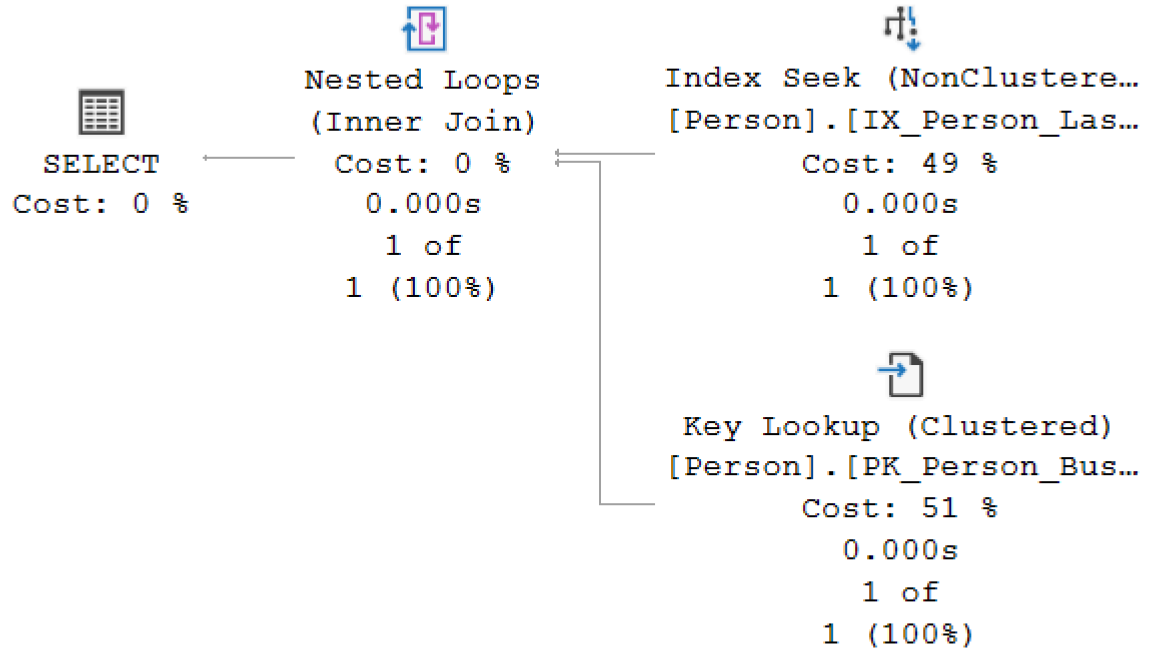
What’s wrong with “SELECT *”?

Read more data

Use more memory

Take more CPU time

What about that Execution Plan?



What determines an Execution Plan?

Your query

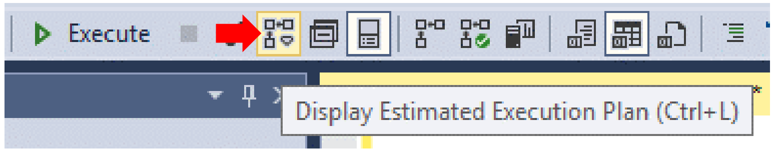
Indexes

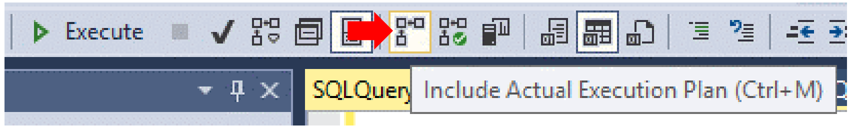
Statistics

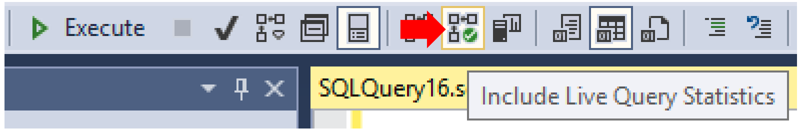
Optimizer Rules

Server/Database Settings

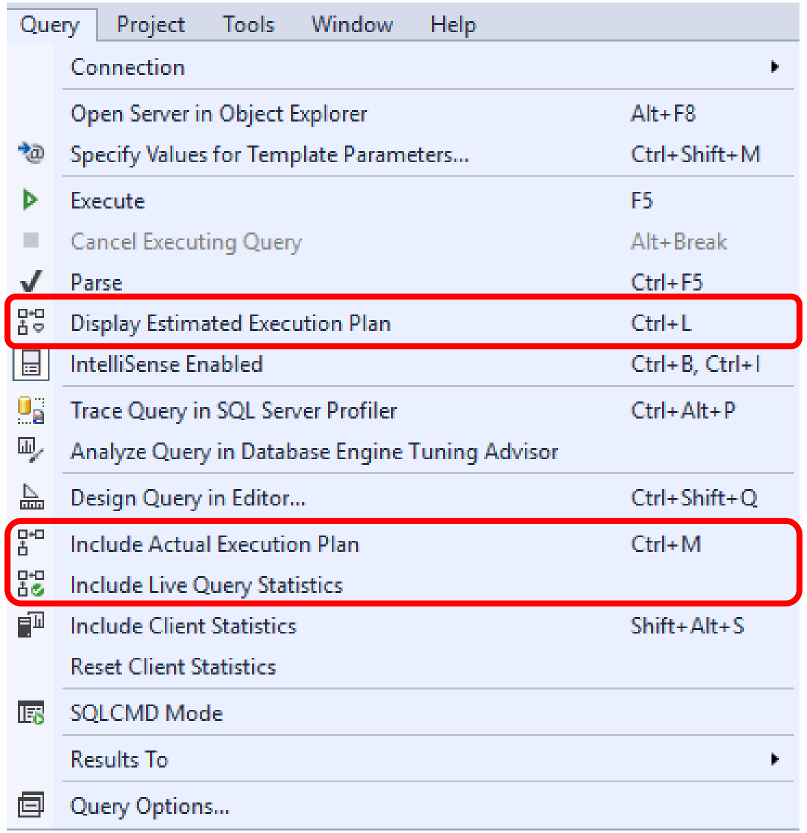
How to see an execution plan







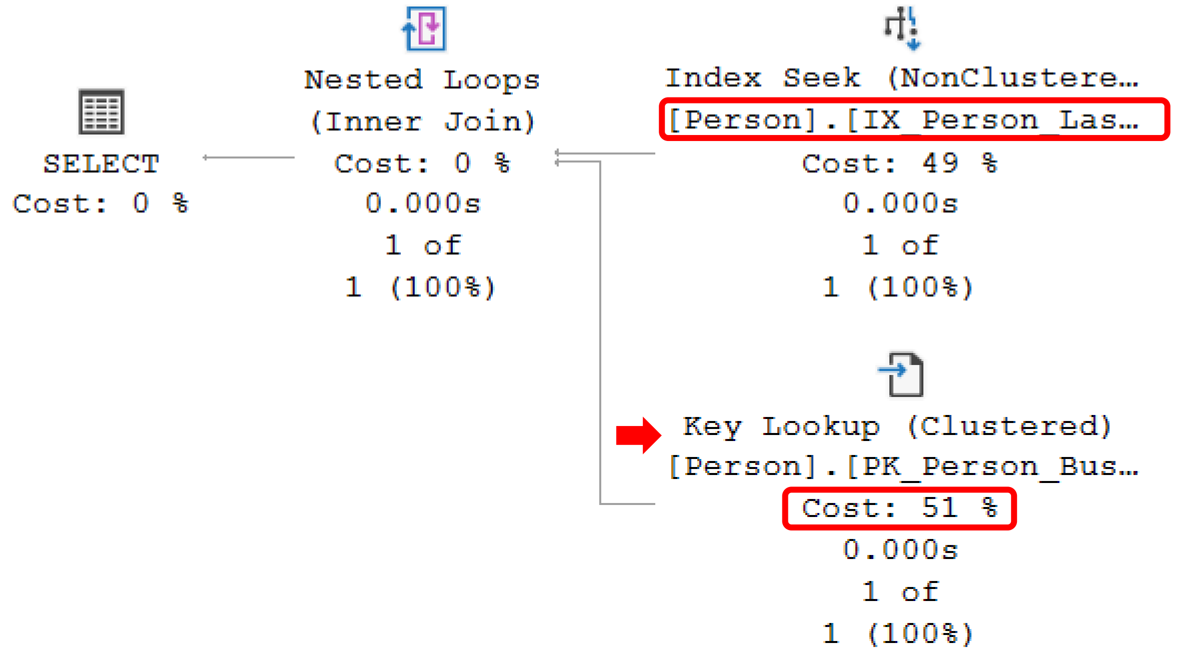
```
SET SHOWPLAN_TEXT ON;
```



Query Project Tools Window Help

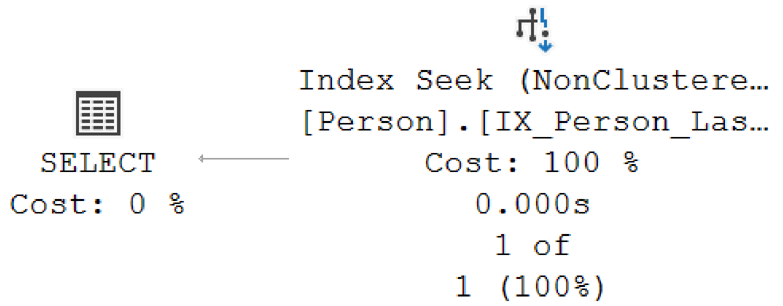
Connection	
Open Server in Object Explorer	Alt+F8
Specify Values for Template Parameters...	Ctrl+Shift+M
Execute	F5
Cancel Executing Query	Alt+Break
Parse	Ctrl+F5
Display Estimated Execution Plan	Ctrl+L
IntelliSense Enabled	Ctrl+B, Ctrl+I
Trace Query in SQL Server Profiler	Ctrl+Alt+P
Analyze Query in Database Engine Tuning Advisor	
Design Query in Editor...	Ctrl+Shift+Q
Include Actual Execution Plan	Ctrl+M
Include Live Query Statistics	
Include Client Statistics	Shift+Alt+S
Reset Client Statistics	
SQLCMD Mode	
Results To	
Query Options...	

Antipattern: Unnatural SELECTion



A better way

```
SELECT FirstName, LastName
FROM Person.Person
WHERE LastName = 'Barker'
```


```
                      Index Seek (NonClustere…
                      [Person].[IX_Person_Las…
SELECT    ←                Cost: 100 %
Cost: 0 %                    0.000s
                              1 of
                            1 (100%)
```

How can we measure the difference?

```
SET STATISTICS IO ON
```

```
Table 'Person'. Scan count 1, logical reads 3821, physical reads 3, read-ahead reads 3866,
                   lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Let’s compare the logical reads

```
SET STATISTICS IO ON
```

```
SELECT *
FROM Person.Person
WHERE LastName = 'Barker'
```

```
logical reads 5
```

```
SELECT FirstName, LastName
FROM Person.Person
WHERE LastName = 'Barker'
```

```
logical reads 2
```

Boss request #2

Hey!

I just had a killer workout

Chasing a yellow car

It looked magnificent!

I want a list of orders

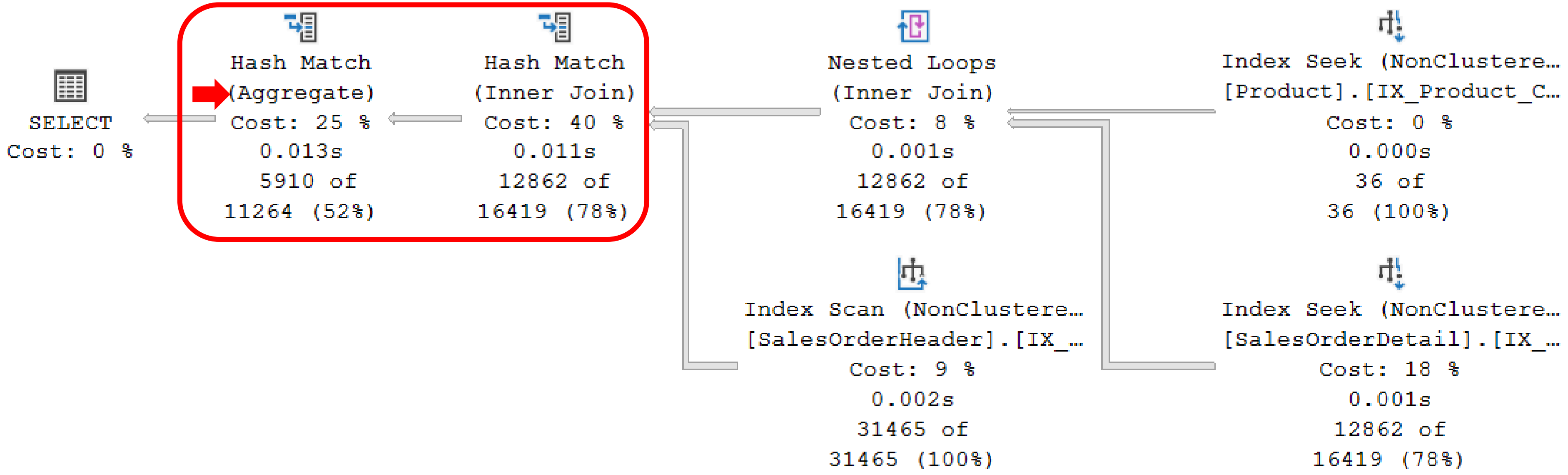
With any yellow products

ASAP!

Junior’s query #2

```
SELECT DISTINCT soh.SalesOrderID
FROM Sales.SalesOrderHeader soh
INNER JOIN Sales.SalesOrderDetail sod
 ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN Production.Product pd
 ON sod.ProductID = pd.ProductID
WHERE pd.Color = 'Yellow'
```

Antipattern: DISTINCT disadvantage

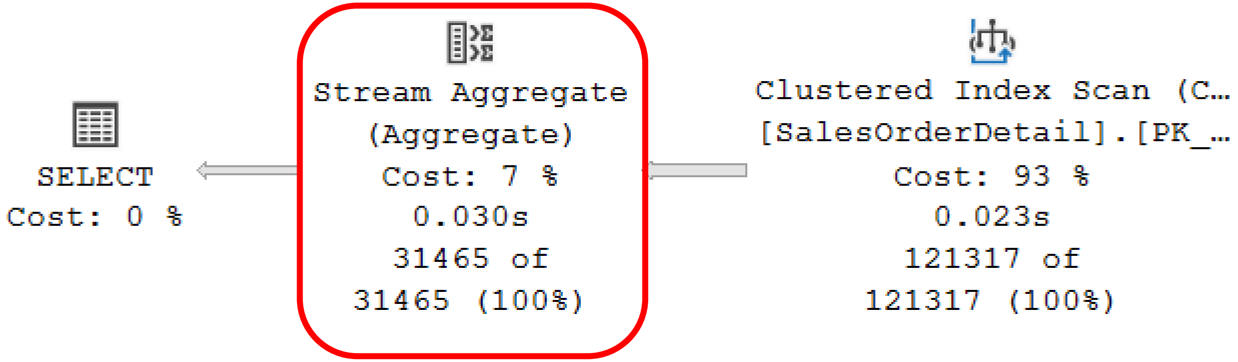


```
SELECT
Cost: 0 %

Hash Match
(Aggregate)
Cost: 25 %
0.013s
5910 of
11264 (52%)

Hash Match
(Inner Join)
Cost: 40 %
0.011s
12862 of
16419 (78%)

Nested Loops
(Inner Join)
Cost: 8 %
0.001s
12862 of
16419 (78%)

Index Seek (NonClustere…
[Product].[IX_Product_C…
Cost: 0 %
0.000s
36 of
36 (100%)

Index Scan (NonClustere…
[SalesOrderHeader].[IX_…
Cost: 9 %
0.002s
31465 of
31465 (100%)

Index Seek (NonClustere…
[SalesOrderDetail].[IX_…
Cost: 18 %
0.001s
12862 of
16419 (78%)
```

DISTINCT = GROUP BY

```
SELECT DISTINCT SalesOrderID
FROM Sales.SalesOrderDetail
```

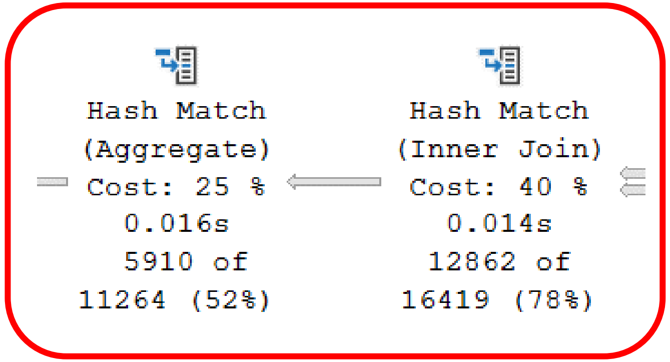
```
SELECT SalesOrderID
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
```



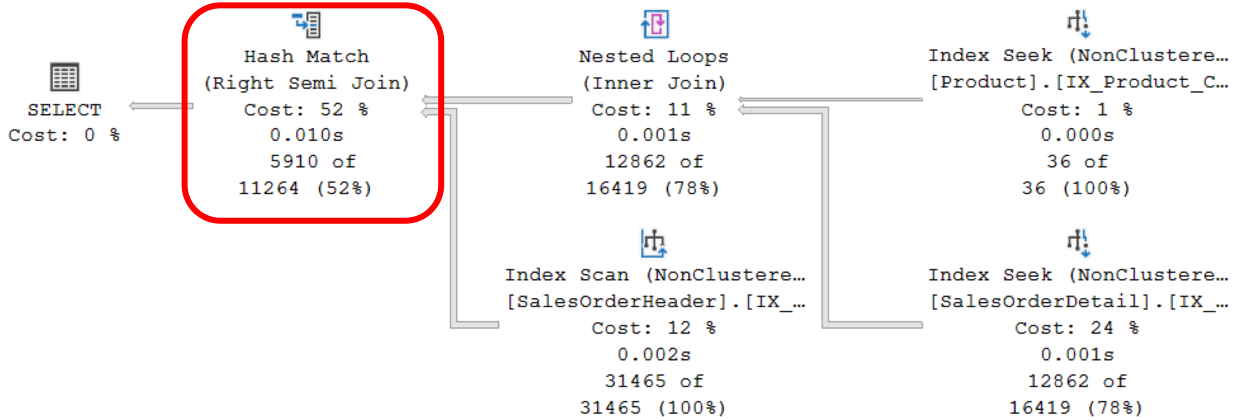
```
SELECT          Stream Aggregate      Clustered Index Scan (C…
Cost: 0 %       (Aggregate)           [SalesOrderDetail].[PK_…
                Cost: 7 %             Cost: 93 %
                0.030s                0.023s
                31465 of              121317 of
                31465 (100%)          121317 (100%)
```

A better way

```sql
SELECT soh.SalesOrderID
FROM Sales.SalesOrderHeader soh
WHERE soh.SalesOrderID IN (
    SELECT sod.SalesOrderID
    FROM Sales.SalesOrderDetail sod
    INNER JOIN Production.Product pd
     ON sod.ProductID = pd.ProductID
    WHERE pd.Color = 'Yellow');
```



```
Hash Match         Hash Match
(Aggregate)        (Inner Join)
Cost: 25 %         Cost: 40 %
0.016s             0.014s
5910 of            12862 of
11264 (52%)        16419 (78%)
```



```
SELECT      Hash Match            Nested Loops      Index Seek (NonClustere...
Cost: 0 %   (Right Semi Join)     (Inner Join)      [Product].[IX_Product_C...
            Cost: 52 %            Cost: 11 %        Cost: 1 %
            0.010s                0.001s            0.000s
            5910 of               12862 of          36 of
            11264 (52%)           16419 (78%)       36 (100%)

                          Index Scan (NonClustere...     Index Seek (NonClustere...
                          [SalesOrderHeader].[IX_...     [SalesOrderDetail].[IX_...
                          Cost: 12 %                     Cost: 24 %
                          0.002s                         0.001s
                          31465 of                       12862 of
                          31465 (100%)                   16419 (78%)
```

What about the logical reads?

```sql
SELECT DISTINCT soh.SalesOrderID
FROM Sales.SalesOrderHeader soh
INNER JOIN Sales.SalesOrderDetail sod
 ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN Production.Product pd
 ON sod.ProductID = pd.ProductID
WHERE pd.Color = 'Yellow';
```

```sql
SELECT soh.SalesOrderID
FROM Sales.SalesOrderHeader soh
WHERE soh.SalesOrderID IN (
    SELECT sod.SalesOrderID
    FROM Sales.SalesOrderDetail sod
    INNER JOIN Production.Product pd
     ON sod.ProductID = pd.ProductID
    WHERE pd.Color = 'Yellow');
```

```
Table 'SalesOrderDetail'
```

```
logical reads 214,
```

```
logical reads 128,
```

Boss request #3

Hey!

Can I get a report

Of how much we have sold

For each product?

Later today I might ask for

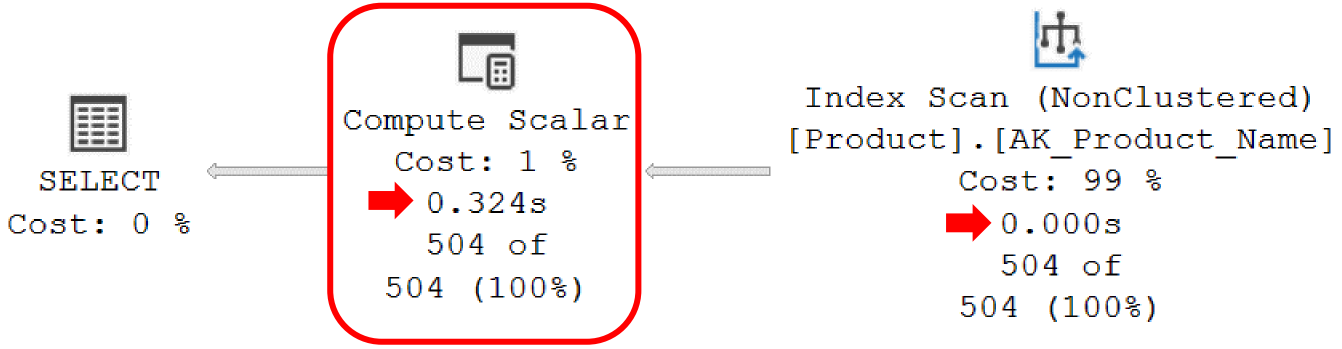
A report for one product.

Or…maybe I won’t.

Junior’s query #3

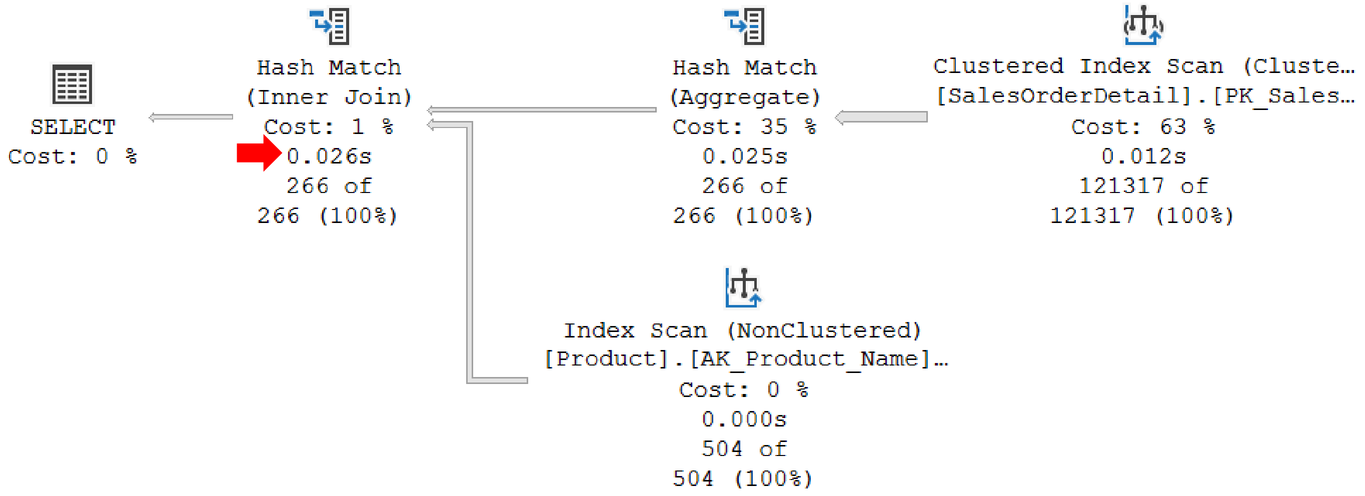
```sql
CREATE OR ALTER FUNCTION dbo.fn_GetSales (@ProductID INT)
    RETURNS INT
AS
    BEGIN
        DECLARE @TotalSold INT;

        SELECT @TotalSold = SUM(OrderQty)
        FROM Sales.SalesOrderDetail
        WHERE ProductID = @ProductID

        RETURN @TotalSold;
    END;
GO

SELECT [Name], dbo.fn_GetSales (ProductID) as TotalSold
FROM Production.Product;
```

Antipattern: Scalar Dysfunction



A better way

```
SELECT p.[Name], SUM(sod.OrderQty) as TotalSold
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
 ON p.ProductID = sod.ProductID
GROUP BY p.[Name]
```

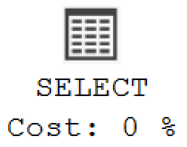


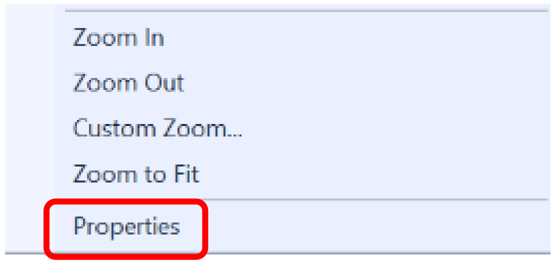
```
SELECT        Hash Match       Hash Match       Clustered Index Scan (Cluste…
Cost: 0 %     (Inner Join)     (Aggregate)      [SalesOrderDetail].[PK_Sales…
              Cost: 1 %        Cost: 35 %       Cost: 63 %
              0.026s           0.025s           0.012s
              266 of           266 of           121317 of
              266 (100%)       266 (100%)       121317 (100%)

                         Index Scan (NonClustered)
                         [Product].[AK_Product_Name]…
                         Cost: 0 %
                         0.000s
                         504 of
                         504 (100%)
```

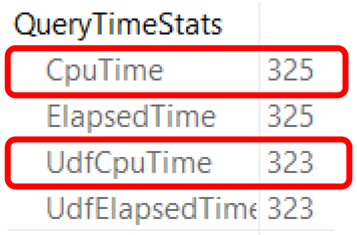
Why the difference in times?

```sql
SELECT [Name], dbo.fn_GetSales (ProductID) as TotalSold
FROM Production.Product;
```

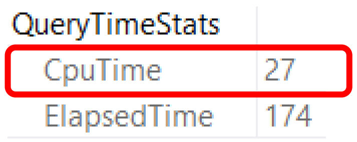
```sql
SELECT p.[Name], SUM(sod.OrderQty) as TotalSold
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
 ON p.ProductID = sod.ProductID
GROUP BY p.[Name]
```





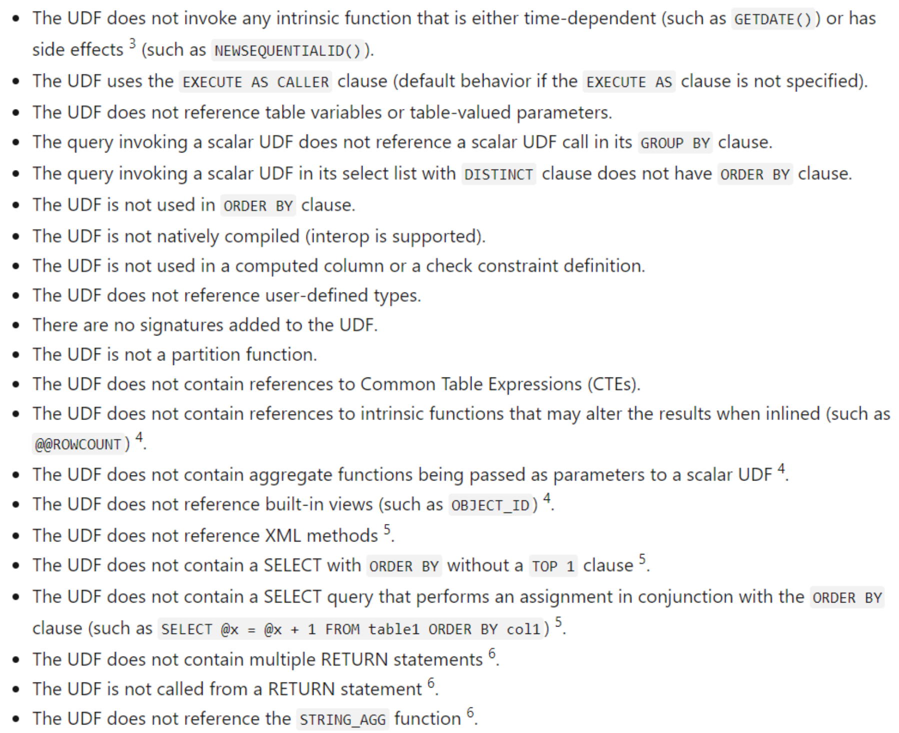


QueryTimeStats	
CpuTime	325
ElapsedTime	325
UdfCpuTime	323
UdfElapsedTime	323



QueryTimeStats	
CpuTime	27
ElapsedTime	174

What about SQL Server 2019 and 2022?



- The UDF does not invoke any intrinsic function that is either time-dependent (such as `GETDATE()`) or has side effects [3] (such as `NEWSEQUENTIALID()`).
- The UDF uses the `EXECUTE AS CALLER` clause (default behavior if the `EXECUTE AS` clause is not specified).
- The UDF does not reference table variables or table-valued parameters.
- The query invoking a scalar UDF does not reference a scalar UDF call in its `GROUP BY` clause.
- The query invoking a scalar UDF in its select list with `DISTINCT` clause does not have `ORDER BY` clause.
- The UDF is not used in `ORDER BY` clause.
- The UDF is not natively compiled (interop is supported).
- The UDF is not used in a computed column or a check constraint definition.
- The UDF does not reference user-defined types.
- There are no signatures added to the UDF.
- The UDF is not a partition function.
- The UDF does not contain references to Common Table Expressions (CTEs).
- The UDF does not contain references to intrinsic functions that may alter the results when inlined (such as `@@ROWCOUNT`) [4].
- The UDF does not contain aggregate functions being passed as parameters to a scalar UDF [4].
- The UDF does not reference built-in views (such as `OBJECT_ID`) [4].
- The UDF does not reference XML methods [5].
- The UDF does not contain a SELECT with `ORDER BY` without a `TOP 1` clause [5].
- The UDF does not contain a SELECT query that performs an assignment in conjunction with the `ORDER BY` clause (such as `SELECT @x = @x + 1 FROM table1 ORDER BY col1`) [5].
- The UDF does not contain multiple RETURN statements [6].
- The UDF is not called from a RETURN statement [6].
- The UDF does not reference the `STRING_AGG` function [6].

https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining?view=sql-server-ver15

Joining antipatterns

(Image courtesy of violeta via pixabay)

Boss request #4

Hey!

Let’s find any US State

With Persons named “Jr”

Who ordered anything black

Because your name

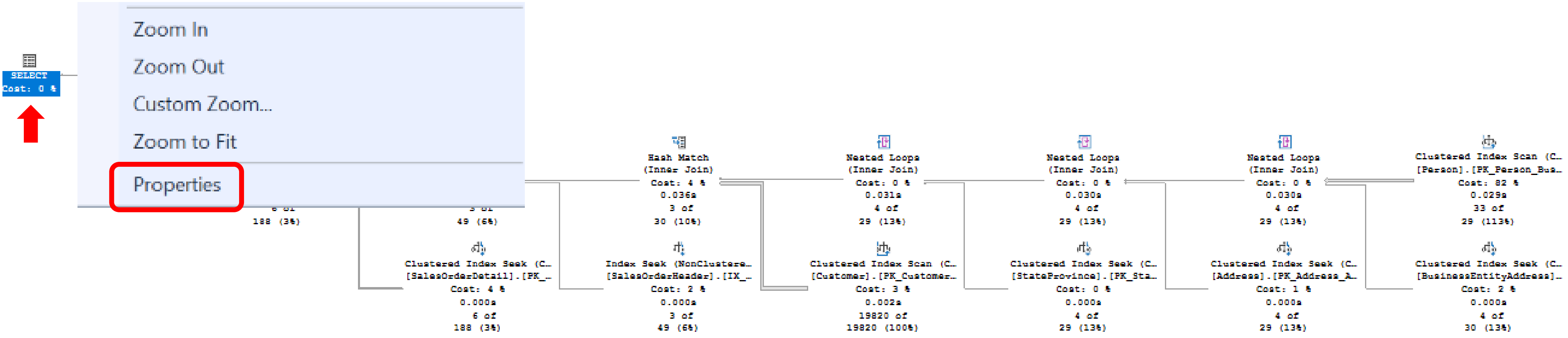
And my color

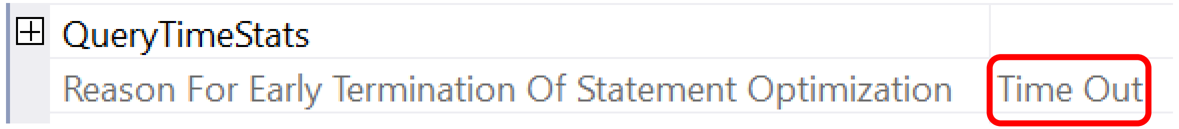
Are awesome!

Junior’s query #4

```
SELECT sp.[Name]
FROM Sales.SalesOrderHeader soh
INNER JOIN Sales.SalesOrderDetail sod
 ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN Production.Product pd
 ON sod.ProductID = pd.ProductID
INNER JOIN Sales.Customer c
 ON soh.CustomerID = c.CustomerID
INNER JOIN Person.Person pr
 ON c.PersonID = pr.BusinessEntityID
INNER JOIN Person.BusinessEntityAddress bea
 ON pr.BusinessEntityID = bea.BusinessEntityID
INNER JOIN Person.Address a
 ON bea.AddressID = a.AddressID
INNER JOIN Person.StateProvince sp
 ON a.StateProvinceID = sp.StateProvinceID
WHERE pr.Suffix = 'Jr.'
 AND pd.Color = 'Black'
```

Antipattern: JOIN-zilla

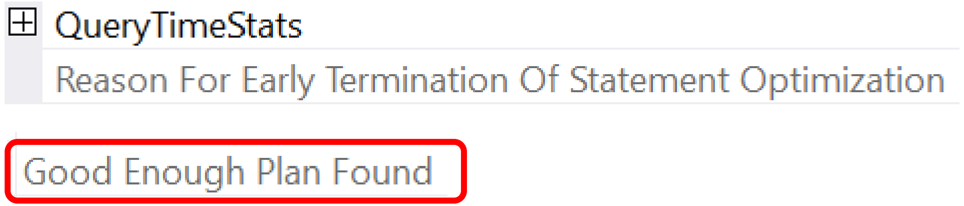




⊞ QueryTimeStats	
Reason For Early Termination Of Statement Optimization	Time Out

A better way

```
SELECT c.CustomerID, sp.[Name] as StateProvince
INTO #JrState
FROM Sales.Customer c
INNER JOIN Person.Person pr
 ON c.PersonID = pr.BusinessEntityID
INNER JOIN Person.BusinessEntityAddress bea
 ON pr.BusinessEntityID = bea.BusinessEntityID
INNER JOIN Person.Address a
 ON bea.AddressID = a.AddressID
INNER JOIN Person.StateProvince sp
 ON a.StateProvinceID = sp.StateProvinceID
WHERE Suffix = 'Jr.'


SELECT ProductID
INTO #Black
FROM Production.Product
WHERE Color = 'Black'

SELECT jr.StateProvince
FROM Sales.SalesOrderHeader soh
INNER JOIN Sales.SalesOrderDetail sod
 ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN #Black black
 ON sod.ProductID = black.ProductID
INNER JOIN #JrState jr
 ON soh.CustomerID = jr.CustomerID
```


QueryTimeStats
Reason For Early Termination Of Statement Optimization
Good Enough Plan Found

What about table variables?

Temporary Table or Table Variable?

Characteristic	#TempTable	@TableVariable
Can be altered	🐶	💩
Can be truncated	🐶	💩
Can be used with SELECT INTO	🐶	💩
Has statistics	🐶	💩
Can participate in a transaction	🐶	💩
Writes only to memory	💩	💩
Avoids writing to the log file	💩	💩
Avoids a recompile in procedure	💩	🐶
Can be passed from a function	💩	🐶

What about Common Table Expressions?

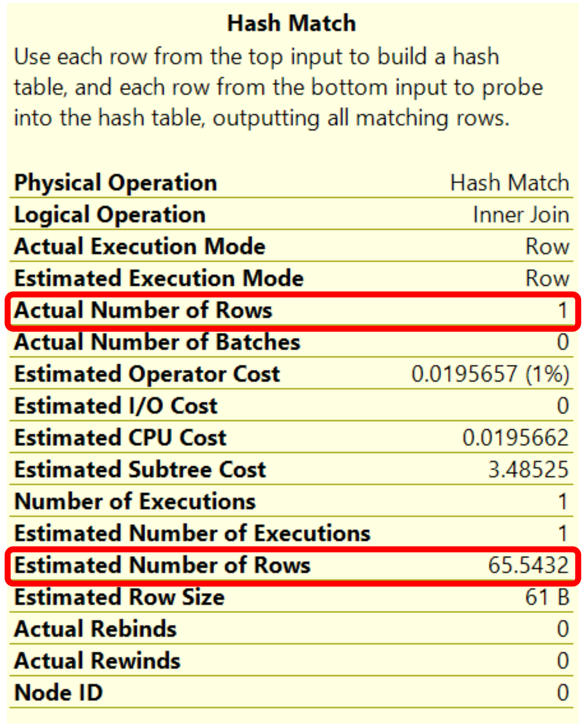
What about Common Table Expressions?

Not a fair comparison

CTEs are not materialized

Can have poor estimates

...and poor execution plans



Hash Match

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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	1
Actual Number of Batches	0
Estimated Operator Cost	0.0195657 (1%)
Estimated I/O Cost	0
Estimated CPU Cost	0.0195662
Estimated Subtree Cost	3.48525
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	65.5432
Estimated Row Size	61 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

If you use CTEs

Use minimal JOINs

Use minimal columns in SELECT

Doing this will keep estimates closer…

…but they still may not be consistent

Boss request #5

Hey!

I just went for another run

Actually, I fetched the mail

And now I am exhausted

But I was wondering

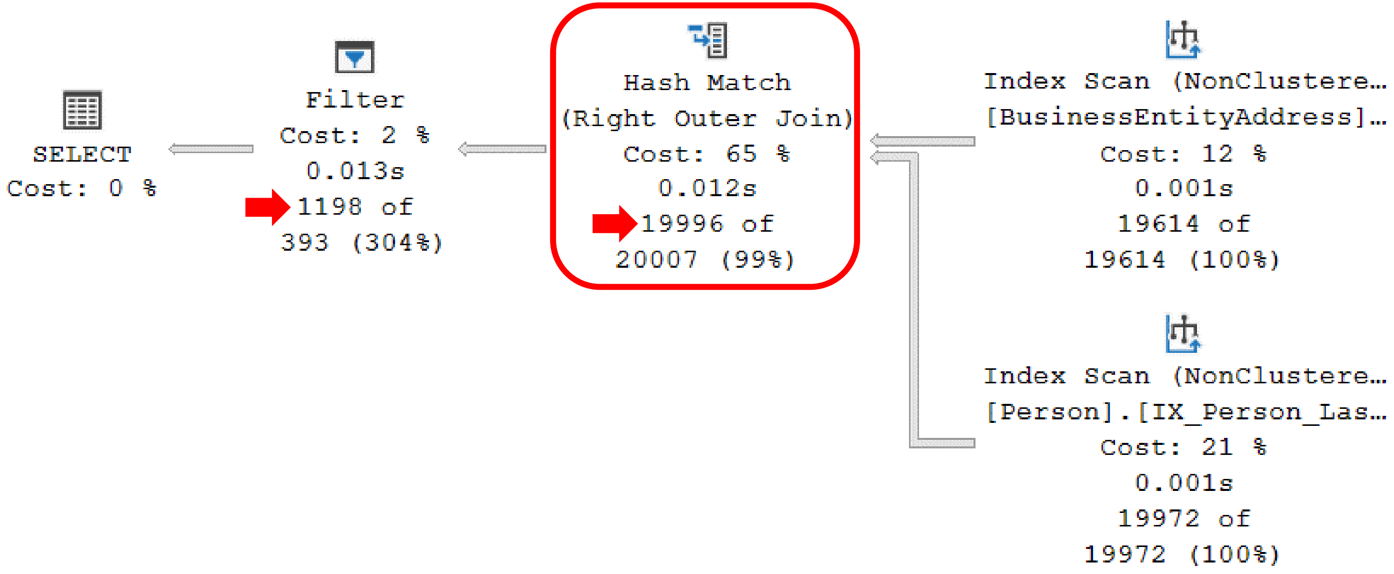
Are any of our customers

Missing a mailing address?

Junior’s query #5

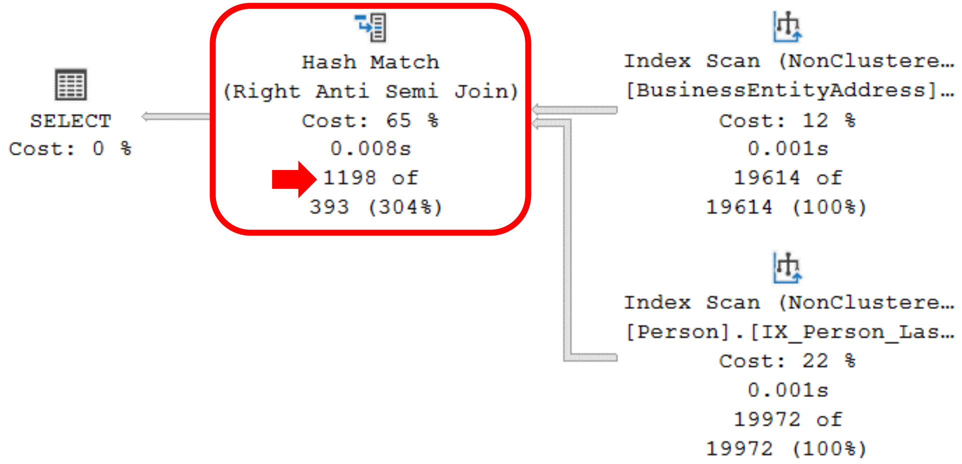
```sql
SELECT p.FirstName, p.LastName
FROM Person.Person p
LEFT OUTER JOIN Person.BusinessEntityAddress a
 ON p.BusinessEntityID = a.BusinessEntityID
WHERE a.BusinessEntityID IS NULL;
```

Antipattern: Avoiding the Semi



A better way

```
SELECT p.FirstName, p.LastName
FROM Person.Person p
WHERE NOT EXISTS (
    SELECT 1
    FROM Person.BusinessEntityAddress a
    WHERE p.BusinessEntityID = a.BusinessEntityID);
```

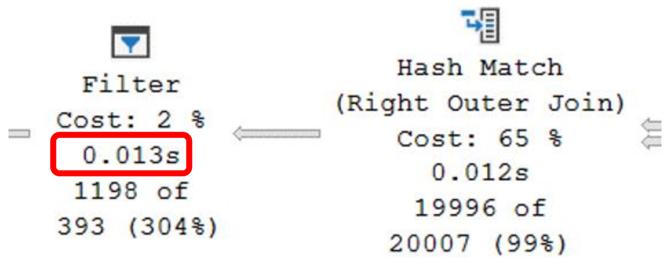


```
SELECT        Hash Match                    Index Scan (NonClustere…
Cost: 0 %     (Right Anti Semi Join)        [BusinessEntityAddress]…
              Cost: 65 %                    Cost: 12 %
              0.008s                        0.001s
              1198 of                       19614 of
              393 (304%)                    19614 (100%)

                                            Index Scan (NonClustere…
                                            [Person].[IX_Person_Las…
                                            Cost: 22 %
                                            0.001s
                                            19972 of
                                            19972 (100%)
```

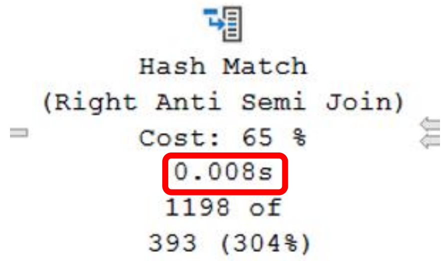
Comparison: Actual Elapsed CPU Time

```sql
SELECT p.FirstName, p.LastName
FROM Person.Person p
LEFT OUTER JOIN Person.BusinessEntityAddress a
 ON p.BusinessEntityID = a.BusinessEntityID
WHERE a.BusinessEntityID IS NULL;
```

```sql
SELECT p.FirstName, p.LastName
FROM Person.Person p
WHERE NOT EXISTS (
    SELECT 1
    FROM Person.BusinessEntityAddress a
    WHERE p.BusinessEntityID = a.BusinessEntityID);
```



```
Filter
Cost: 2 %
0.013s
1198 of
393 (304%)

Hash Match
(Right Outer Join)
Cost: 65 %
0.012s
19996 of
20007 (99%)
```



```
Hash Match
(Right Anti Semi Join)
Cost: 65 %
0.008s
1198 of
393 (304%)
```

Predicate antipatterns

(Image courtesy of Pezibear via pixabay)

Boss request #6

Hey!

I was born in 2012

It is my favorite year

Let’s find how many orders

We had that year

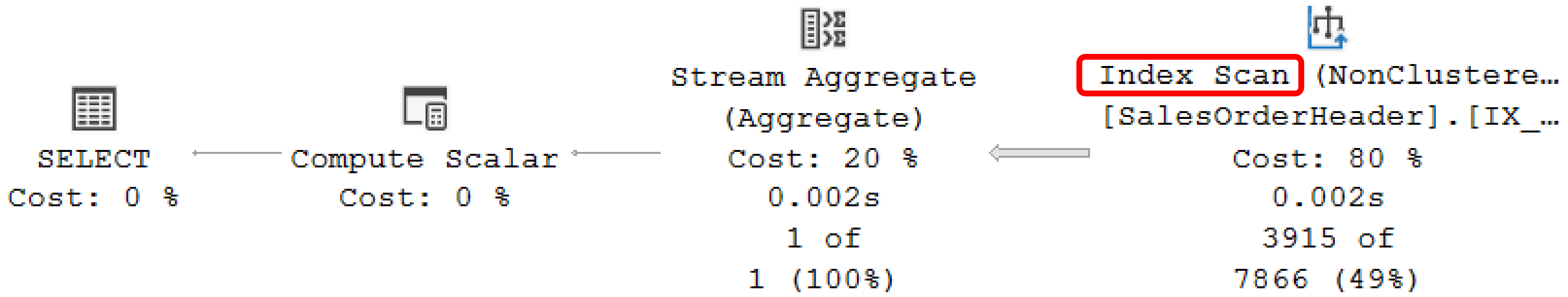
Because

It is my favorite year!

Junior’s query #6

```sql
DECLARE @Year int;

SET @Year = 2012;

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE YEAR(OrderDate) = @Year;
```

Antipattern: Fallacious arguments



This is not “SARGable”

SARGable - Able to efficiently use an index for a search argument

So…what else is not SARGable?

So…what else is not SARGable?

Description	Example
Most functions	`WHERE DATEADD(YEAR, -1, OrderDate) = GETDATE()`
Conversions	`WHERE CAST(OrderDate AS CHAR(10)) = '2016-01-01'`
Operators	`WHERE TotalDue - 1000.00 > 0`
Concatenating columns	`WHERE FirstName + LastName = 'Charles Barkley'`
LIKE with a leading wildcard	`WHERE FirstName LIKE '%Wolf'`
CASE statements	`WHERE 1 = CASE WHEN FirstName = 'Cat' THEN 0`

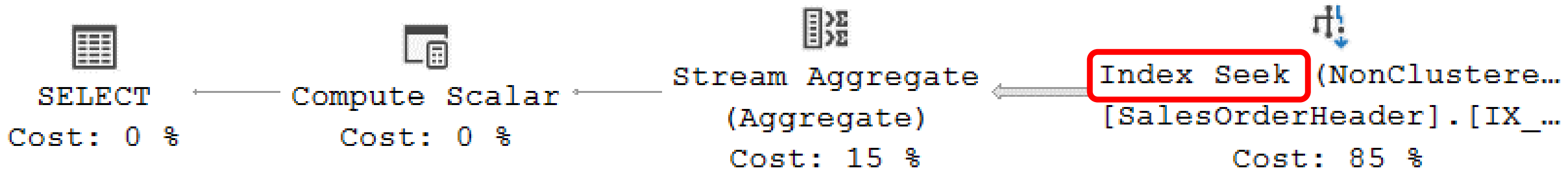
What’s the solution?

Vern Rabe says:

“Embrace Verbosity!”

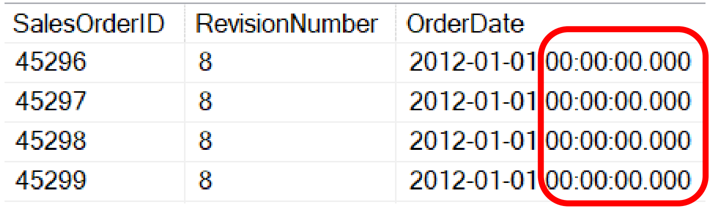
A better way

```
DECLARE @Year int;

SET @Year = 2012;

DECLARE @YearStart datetime, @YearEnd datetime

SELECT @YearStart = CAST(CAST(@Year as VARCHAR(4)) + '-01-01' AS DATETIME)
SELECT @YearEnd = CAST(CAST(@Year as VARCHAR(4)) + '-12-31' AS DATETIME)

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @YearStart AND @YearEnd;
```



But...what about 12-31 after midnight?

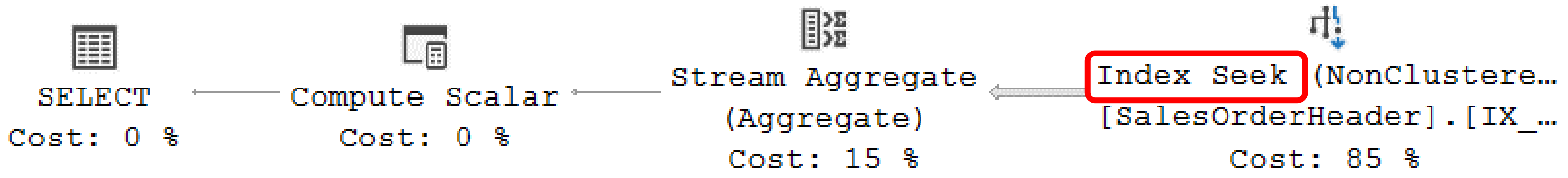
```
DECLARE @Year int;

SET @Year = 2012;

DECLARE @YearStart datetime, @YearEnd datetime

SELECT @YearStart = CAST(CAST(@Year as VARCHAR(4)) + '-01-01' AS DATETIME)
SELECT @YearEnd = CAST(CAST(@Year as VARCHAR(4)) + '-12-31' AS DATETIME)

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN @YearStart AND @YearEnd;
```



SalesOrderID	RevisionNumber	OrderDate
45296	8	2012-01-01 00:00:00.000
45297	8	2012-01-01 00:00:00.000
45298	8	2012-01-01 00:00:00.000
45299	8	2012-01-01 00:00:00.000

An even better way

```
DECLARE @Year int;

SET @Year = 2012;

DECLARE @YearStart datetime

SELECT @YearStart = CAST(CAST(@Year as VARCHAR(4)) + '-01-01' AS DATETIME)

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE OrderDate >= @YearStart
 AND OrderDate < DATEADD(YY, 1, @YearStart);
```


SELECT
Cost: 0 %

Compute Scalar
Cost: 0 %

Stream Aggregate
(Aggregate)
Cost: 15 %

Index Seek (NonClustere…
[SalesOrderHeader].[IX_…
Cost: 85 %

Show me the logical reads!

```
DECLARE @Year int;

SET @Year = 2012;

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE YEAR(OrderDate) = @Year;
```

```
DECLARE @Year int;

SET @Year = 2012;

DECLARE @YearStart datetime

SELECT @YearStart = CAST(CAST(@Year as VARCHAR(4)) + '-01-01' AS DATETIME)

SELECT COUNT(SalesOrderID)
FROM Sales.SalesOrderHeader
WHERE OrderDate >= @YearStart
 AND OrderDate < DATEADD(YY, 1, @YearStart);
```

```
logical reads 73,
```

```
logical reads 12,
```

Boss request #7

Hey!

I need a report

To find any order shipped

On a given day

Or, any unshipped orders

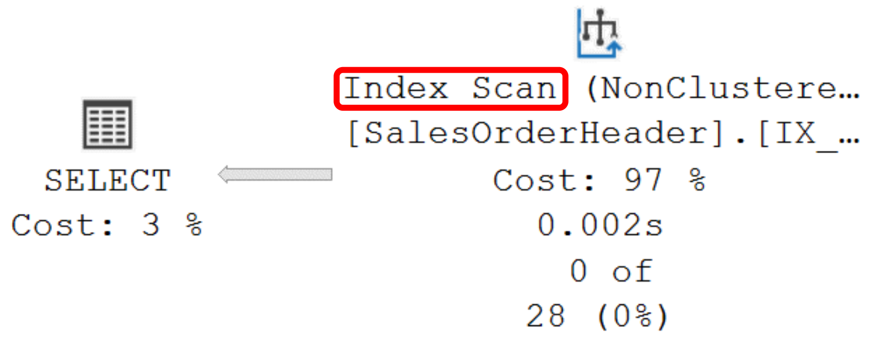
Also, I am tired

And wish to take a nap

Junior’s query #7

```sql
CREATE PROCEDURE usp_GetShippedOrders
    @ShipDate datetime
AS

SELECT SalesOrderID, ShipDate
FROM Sales.SalesOrderHeader
WHERE ISNULL(ShipDate, '19010101')
 = ISNULL(@ShipDate, '19010101');
```

Antipattern: A Lot of Nothing

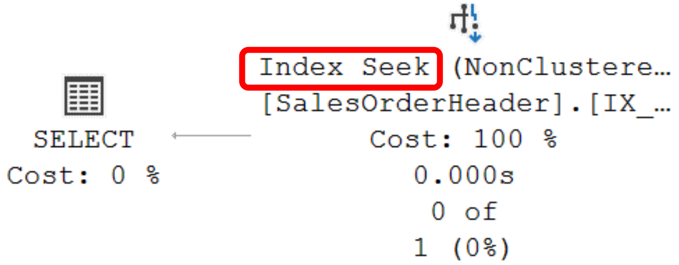


```
                         Index Scan (NonClustere…
                         [SalesOrderHeader].[IX_…
  SELECT                        Cost: 97 %
Cost: 3 %                         0.002s
                                   0 of
                                 28 (0%)
```

Déjà vu - this is also not “SARGable”

A better way

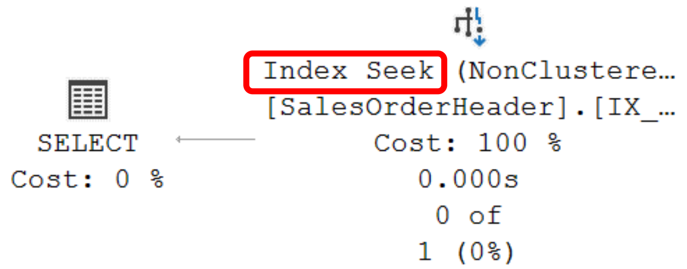
```
CREATE PROCEDURE usp_GetShippedOrders
    @ShipDate datetime
AS

SELECT SalesOrderID, ShipDate
FROM Sales.SalesOrderHeader
FROM Sales.SalesOrderHeader
WHERE ShipDate = @ShipDate
 OR (ShipDate IS NULL AND @ShipDate IS NULL);
```



```
                        Index Seek (NonClustere…
                        [SalesOrderHeader].[IX_…
SELECT                      Cost: 100 %
Cost: 0 %                    0.000s
                              0 of
                             1 (0%)
```

An even better way

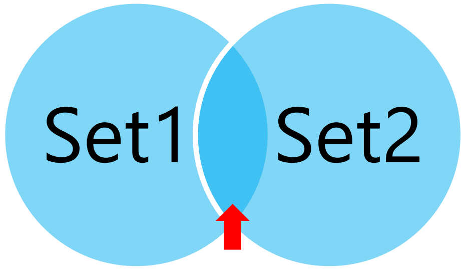
```
CREATE PROCEDURE usp_GetShippedOrders
    @ShipDate datetime
AS

SELECT SalesOrderID, ShipDate
FROM Sales.SalesOrderHeader
WHERE EXISTS (
 SELECT ShipDate INTERSECT SELECT @ShipDate
 );
```



Tell me more about INTERSECT

INNER JOIN – matches data

INTERSECT – finds distinct common values



Rule #1: column name & order must match

Rule #2: column data types must match

How many logical reads this time?

```
CREATE PROCEDURE usp_GetShippedOrders
    @ShipDate datetime
AS

SELECT SalesOrderID, ShipDate
FROM Sales.SalesOrderHeader
WHERE ISNULL(ShipDate, '19010101')
 = ISNULL(@ShipDate, '19010101');
```

```
logical reads 73,
```

```
CREATE PROCEDURE usp_GetShippedOrders
    @ShipDate datetime
AS

SELECT SalesOrderID, ShipDate
FROM Sales.SalesOrderHeader
WHERE EXISTS (
 SELECT ShipDate INTERSECT SELECT @ShipDate
 );
```

```
logical reads 2,
```

Boss request #8

Hey!

We got a charge back

And we might have more

We need to find the SalesID

For an approval code

I will Slack you the code

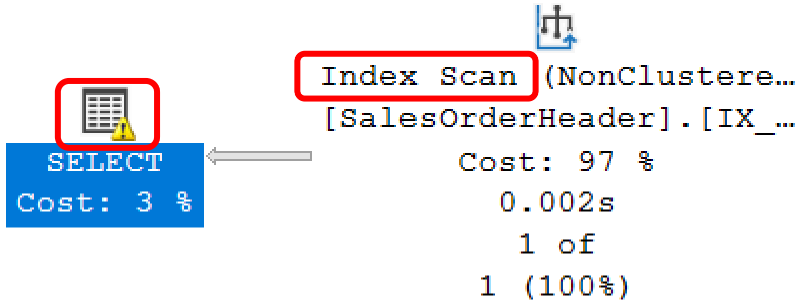
And then take another nap

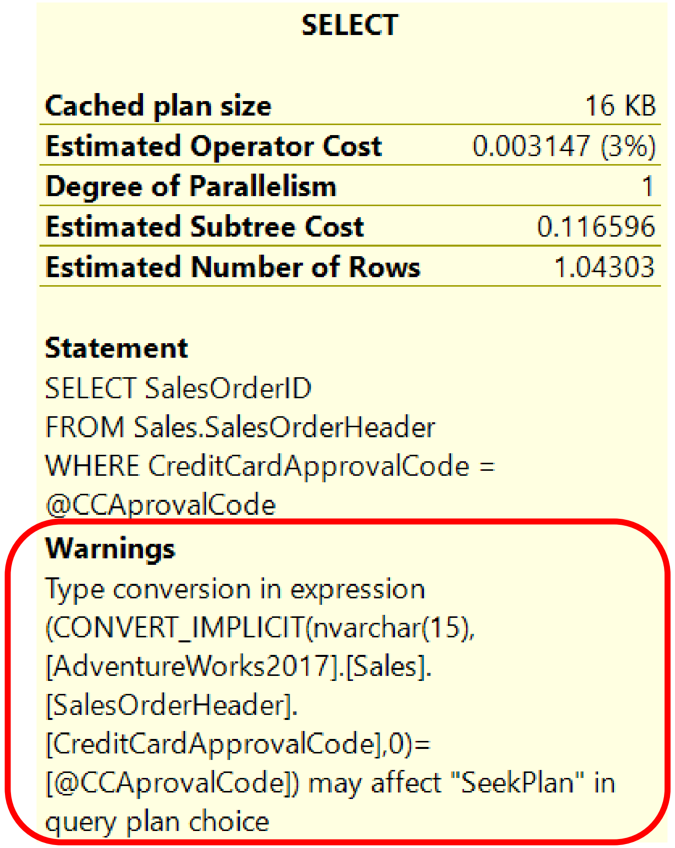
Junior’s query #8

```sql
CREATE PROCEDURE usp_GetShippedOrders
    @CCAprovalCode NVARCHAR(15)
AS

SELECT SalesOrderID
FROM Sales.SalesOrderHeader
WHERE CreditCardApprovalCode = @CCAprovalCode;
```

```
⊟ ▦ Sales.SalesOrderHeader
    ⊟ Columns
        CreditCardApprovalCode (varchar(15), null)
```

Antipattern: Involuntary conversion





SELECT

Cached plan size	16 KB
Estimated Operator Cost	0.003147 (3%)
Degree of Parallelism	1
Estimated Subtree Cost	0.116596
Estimated Number of Rows	1.04303

Statement
SELECT SalesOrderID
FROM Sales.SalesOrderHeader
WHERE CreditCardApprovalCode =
@CCAprovalCode

Warnings
Type conversion in expression
(CONVERT_IMPLICIT(nvarchar(15),
[AdventureWorks2017].[Sales].
[SalesOrderHeader].
[CreditCardApprovalCode],0)=
[@CCAprovalCode]) may affect "SeekPlan" in
query plan choice

What are Implicit Conversions?

Did you say…?

(Image courtesy of BarkPost)

What about Precedence?

Precedence Chart*
DATETIME
SMALLDATETIME
FLOAT
DECIMAL
MONEY
BIGINT
INT
SMALLINT
TINYINT
BIT
NVARCHAR (including MAX)
NCHAR
VARCHAR (including MAX)
CHAR

Data of lower precedence must be implicitly converted to be compared to data of higher precedence

```
@CCAprovalCode NVARCHAR(15)
```

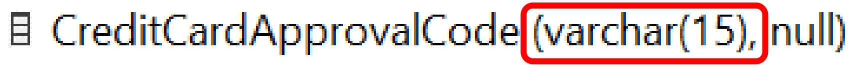
```
CreditCardApprovalCode (varchar(15), null)
```

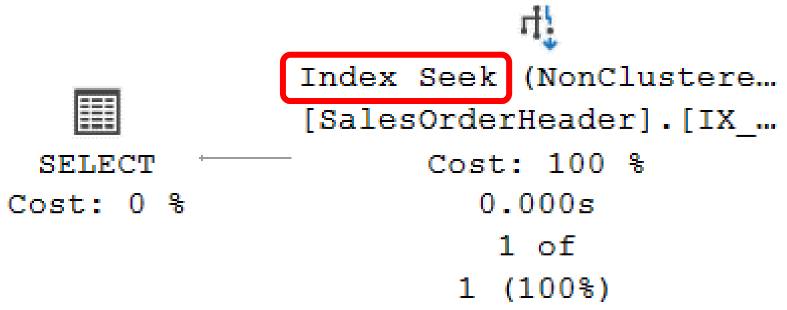
*The Fine Print

Abbreviated chart. Includes only commonly used data types. There are 30 different data types in the full chart, with User-Defined Data Types occupying the highest precedence.

A better way

```
CREATE PROCEDURE usp_GetShippedOrders
    @CCAprovalCode VARCHAR(15)
AS

SELECT SalesOrderID
FROM Sales.SalesOrderHeader
WHERE CreditCardApprovalCode = @CCAprovalCode;
```


CreditCardApprovalCode (varchar(15), null)



“sql authority find implicit conversion”

Pinal Dave says:

```
-- (c) https://blog.sqlauthority.com
SELECT TOP(50) DB_NAME(t.[dbid]) AS [Database Name],
t.text AS [Query Text],
qs.total_worker_time AS [Total Worker Time],
qs.total_worker_time/qs.execution_count AS [Avg Worker Time],
qs.max_worker_time AS [Max Worker Time],
qs.total_elapsed_time/qs.execution_count AS [Avg Elapsed Time],
qs.max_elapsed_time AS [Max Elapsed Time],
qs.total_logical_reads/qs.execution_count AS [Avg Logical Reads],
qs.max_logical_reads AS [Max Logical Reads],
qs.execution_count AS [Execution Count],
qs.creation_time AS [Creation Time],
qp.query_plan AS [Query Plan]
FROM sys.dm_exec_query_stats AS qs WITH (NOLOCK)
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS t
CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
WHERE CAST(query_plan AS NVARCHAR(MAX)) LIKE ('%CONVERT_IMPLICIT%')
 AND t.[dbid] = DB_ID()
ORDER BY qs.total_worker_time DESC OPTION (RECOMPILE);
```

https://blog.sqlauthority.com/2017/01/29/find-all-queries-with-implicit-conversion-in-sql-server-interview-question-of-the-week-107/

“A final ask”

Hey!

I just woke up from a nap

And I wanted to reach out

To briefly touch base

With a final ask

To see if we can circle back

For a helicopter view

Junior’s Summary

Something about Index Seeks

Confusing execution plans

A bunch of dog pictures

Summary

Antipattern	Characteristic	Solution
Unnatural SELECTION	SELECT *	Limit column selection
DISTINCT Disadvantage	SELECT DISTINCT…	WHERE IN (…)
UD Dysfunction	dbo.fn_xxx	Maybe 2019?
JOIN-zilla	Optimization “Time Out”	Break it up
Avoiding the Semi	LEFT JOIN…WHERE NULL	WHERE NOT EXISTS (…)
Fallacious Arguments	Not SARGable	“Embrace verbosity”
A Lot of Nothing	ISNULL() = ISNULL()	EXISTS & INTERSECT
Involuntary Conversion	CONVERT_IMPLICIT	Match all data types

That’s the end. Thank you!

“Seven”

desertdba.com

jeff@desertdba.com

github.com\desertdba

“Daphne”