

The Analyst's Guide To:

Identifying and Fixing Performance Anti-Patterns



@bertwagner



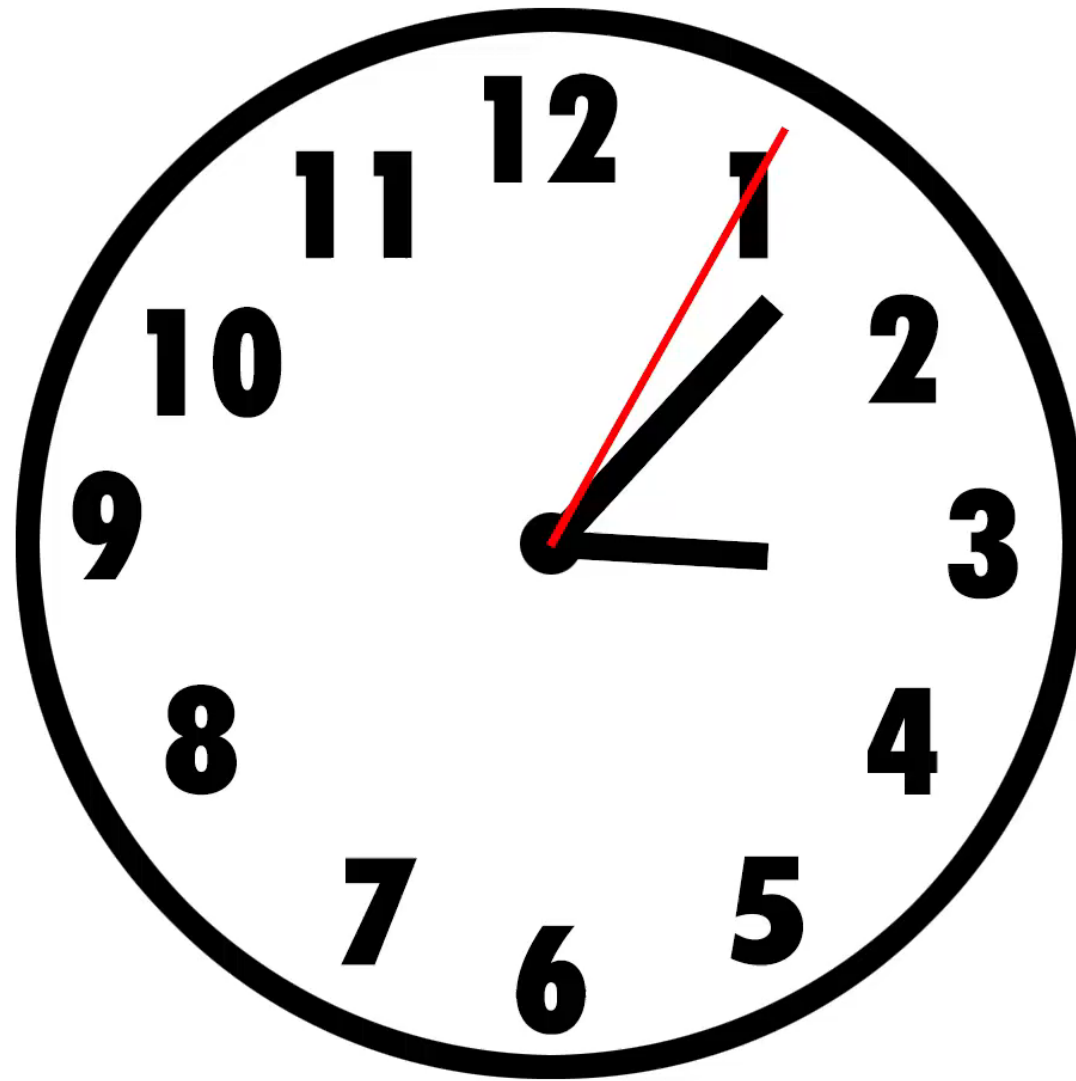
bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

Patterns



FlooringSupplies.co.uk

Anti-Patterns



[Chris Maluszynski / MOMENT](#)



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

5 Performance Anti-Patterns

(and how to fix them)



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

Anti-Pattern #1: Downloading Data

Move the smaller data



EligibleCustomers.csv



```
SELECT  
    ...  
FROM  
    dbo.Customers  
WHERE  
    Id IN (1,2,3,...)
```



@bertwagner



bertwagner.com

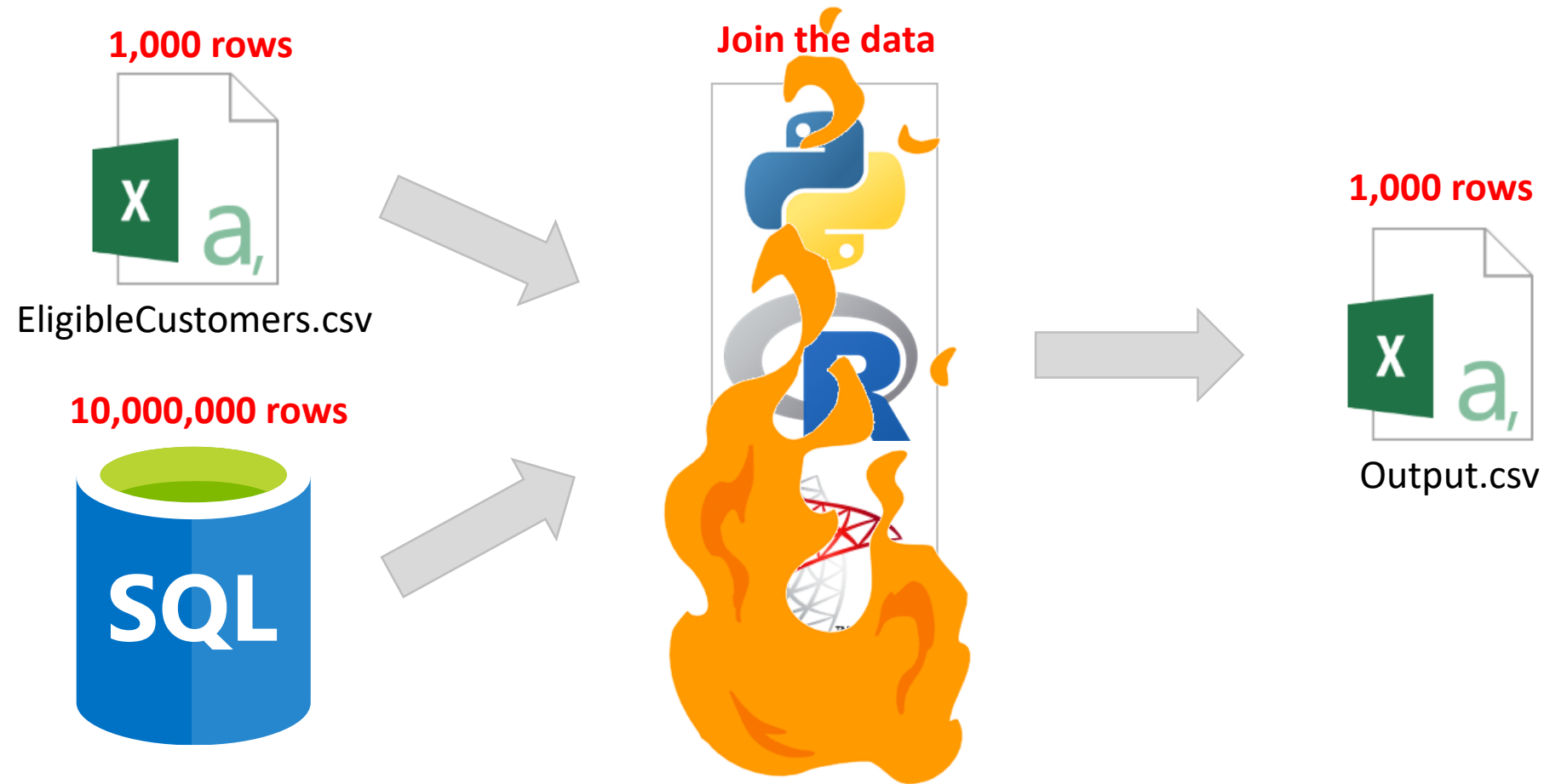


youtube.com/bertwagner

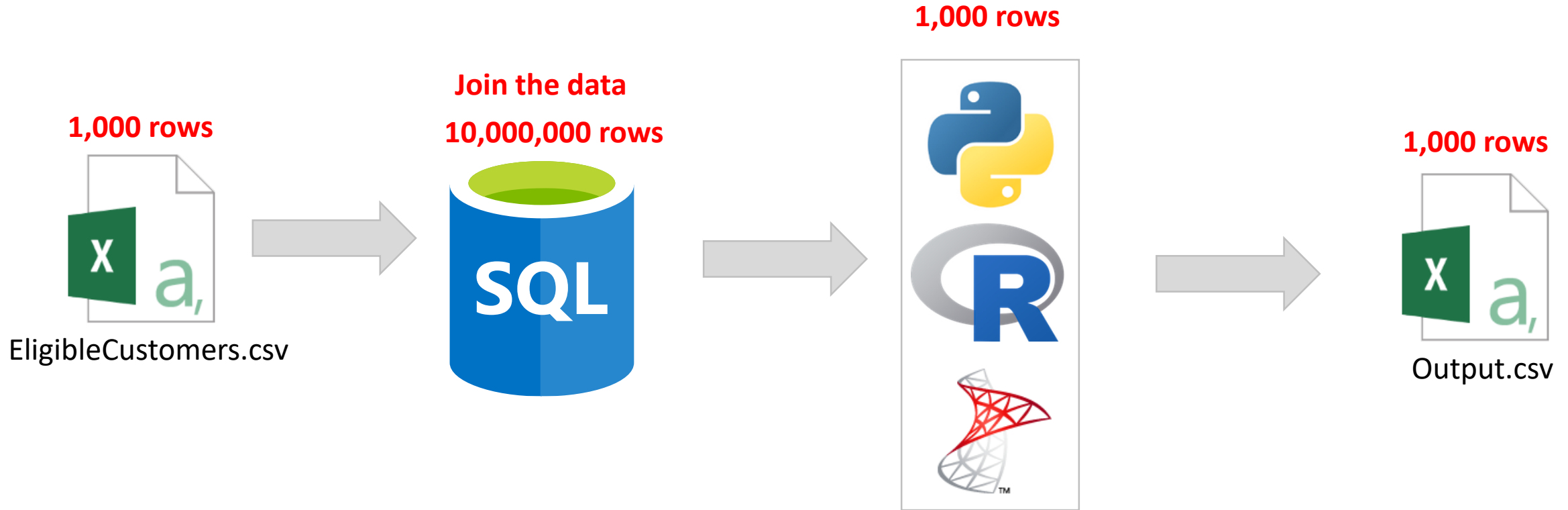


bert@bertwagner.com

Move the smaller data



Move the smaller data



Move the smaller data

```
CREATE TABLE ##EligibleCustomers;
-- bulk insert OR use your tool of choice
BULK INSERT ##EligibleCustomers FROM 'C:\EligibleCustomers.csv'

CREATE CLUSTERED INDEX CL_Id
    ON ##EligibleCustomers (Id);
CREATE NONCLUSTERED INDEX IX_PurchaseDate
    ON ##EligibleCustomers(PurchaseDate) INCLUDE (FullName);

SELECT
    ...
FROM
    dbo.Customers c
    INNER JOIN ##EligibleCustomers ec
        ON c.Id = ec.Id
```

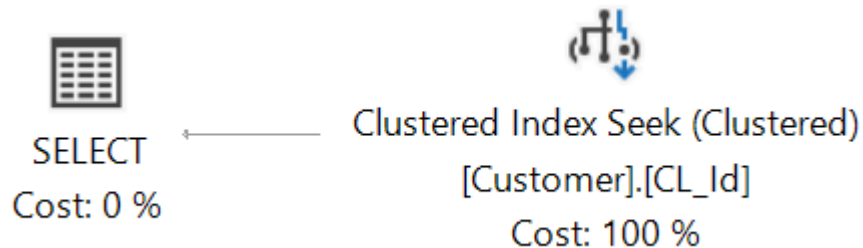


A photograph of two clenched fists, one on the left and one on the right, with metal handcuffs attached to the wrists. The fists are raised and clenched, symbolizing resistance or protest. The background is a solid, muted gray.

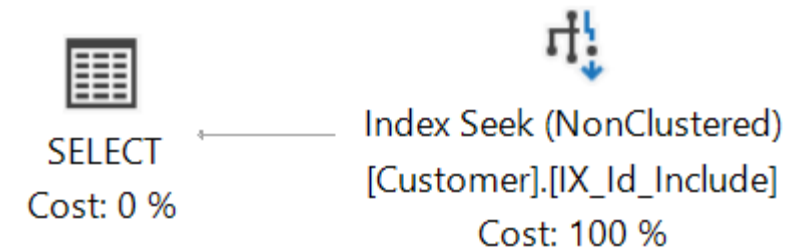
Anti-Pattern #2: Preventing Index Use

SELECT *

```
SELECT
    *
FROM
    dbo.Customer
WHERE
    Id = 123
```



```
SELECT
    Id,
    PurchaseDate
FROM
    dbo.Customer
WHERE
    Id = 123
```



Converting Data on Read

SELECT

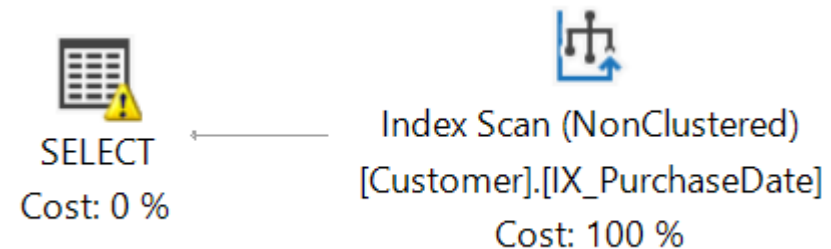
...

FROM

dbo.Customer

WHERE

CAST(PurchaseDate AS CHAR(10)) = '2019-01-01'



SELECT

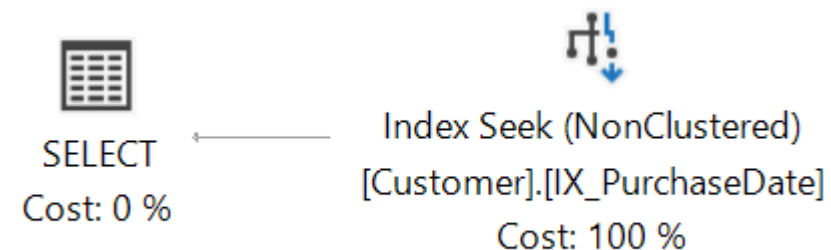
...

FROM

dbo.Customer

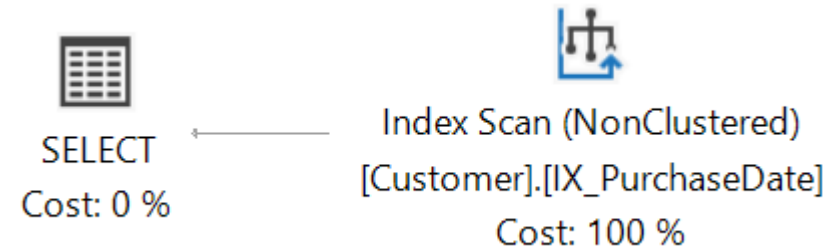
WHERE

PurchaseDate = CAST('2019-01-01' AS DATETIME2)

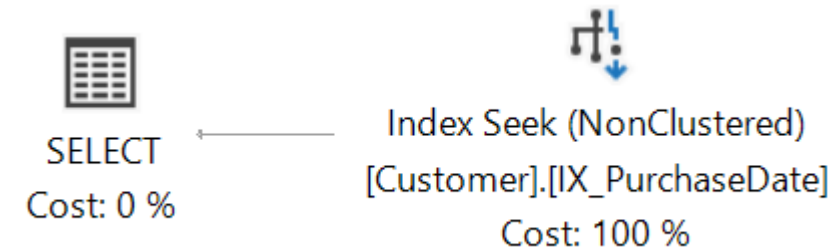


Scalar Functions

```
SELECT
    FullName, Id
FROM
    dbo.Customer
WHERE
    DATEADD(day, 30, PurchaseDate) >= GETDATE()
```



```
SELECT
    FullName, Id
FROM
    dbo.Customer
WHERE
    PurchaseDate >= DATEADD(day, -30, GETDATE())
```



Implicit Conversions

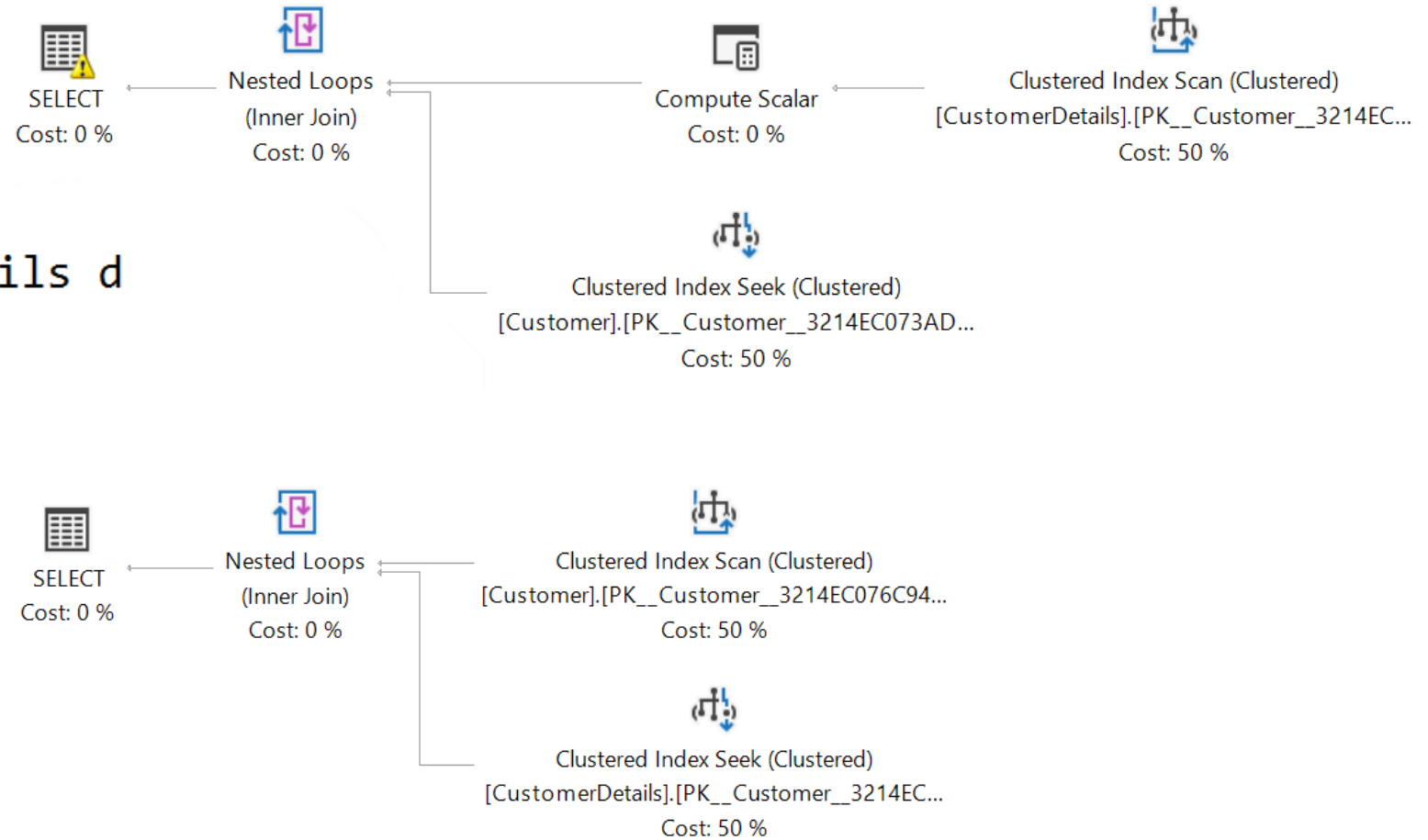
```
CREATE TABLE dbo.Customer (Id int PRIMARY KEY...  
CREATE TABLE dbo.CustomerDetails (Id int PRIMARY KEY... /...
```

SELECT

...

FROM

```
dbo.Customer c  
INNER JOIN dbo.CustomerDetails d  
ON c.Id = d.Id
```

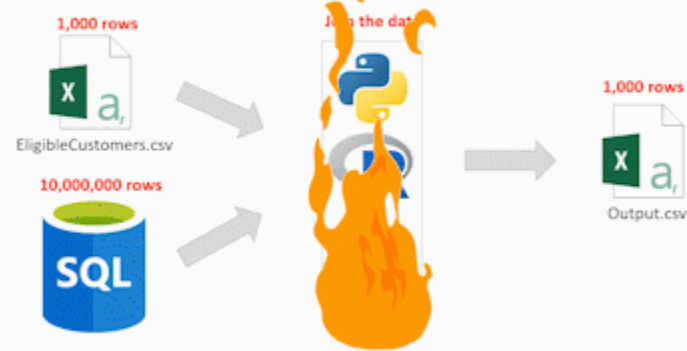




Anti-Pattern #3: Using The Wrong Tool

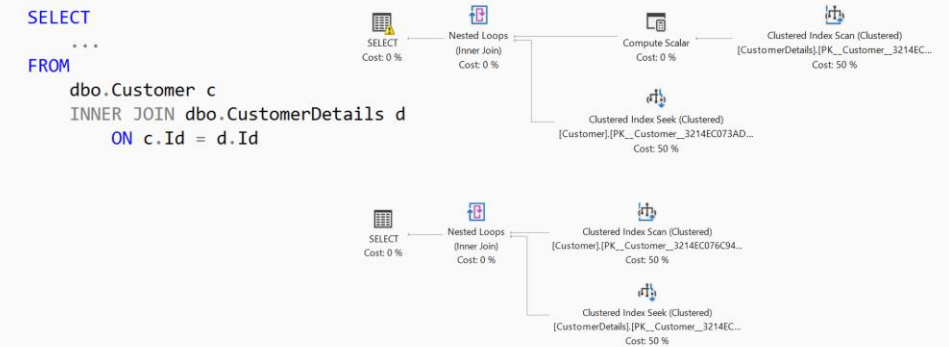
Joining

Move the small data



@bertwagner bertwagner.com youtube.com/bertwagner bert@bertwagner.com

Implicit Conversions



@bertwagner bertwagner.com youtube.com/bertwagner bert@bertwagner.com

Join data where it makes sense to do so:

- Do you need to move a lot of data?
- Are indexes available?
- Do you need to convert datatypes?

Filtering

```
SELECT
    ...
FROM
    dbo.Orders
WHERE
    PurchaseDate = '2019-01-01'
```



Index Seek (NonClustered)
[Orders].[IX_PurchaseDate_Includes]

Rows returned: **5,000**
Rows read: **5,000**

```
for (int i = 0; i < table.Rows.Count; i++)
{
    if (table.Rows[i]["PurchaseDate"].ToString() == "2019-01-01")
    {
        output.Rows.Add(table.Rows[i]);
    }
}
```

Rows returned: **5,000**
Rows read: **10,000,000**



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

Aggregating

```
table = ReadInDataFromDatabase();  
  
var total = table.AsEnumerable()  
    .Sum(x => x.Field<int>("Total") );  
  
var average = table.AsEnumerable()  
    .Average(x => x.Field<int>("Total") );
```

```
SELECT  
    SUM(Total) AS OrderTotal,  
    AVG(Total) AS OrderAverage,  
    ...  
FROM  
    dbo.Orders
```



Columnstore Index Scan (NonClustered)
[Orders].[NCC_Total]



Window and Analytical Functions

```
SELECT
  CustomerId,
  Total,
  SUM(Total) OVER (
    PARTITION BY CustomerId
    ORDER BY PurchaseDate
  ) AS RunningTotal,
  LEAD(Total) OVER (...) AS NextValue,
  LAST_VALUE(Total) OVER (
    ...
    ROWS BETWEEN UNBOUNDED PRECEDING
    AND UNBOUNDED FOLLOWING) AS LastValue
FROM
  dbo.Orders
```

	CustomerId	Total	RunningTotal	NextValue	LastValue
1	1	5	5	8	6
2	1	8	13	3	6
3	1	3	16	6	6
4	1	6	22	NULL	6
5	2	5	5	5	1
6	2	5	10	1	1
7	2	1	11	NULL	1
8	3	2	2	8	2
9	3	8	10	9	2
10	3	9	19	5	2
11	3	5	24	4	2
12	3	4	28	2	2
13	3	2	30	NULL	2



Sorting*

SELECT

CustomerId,
PurchaseDate,
Total

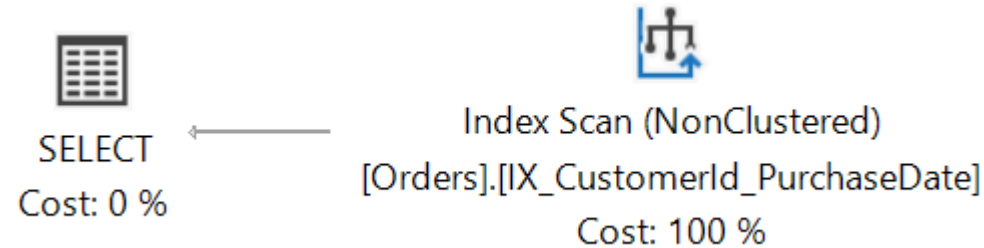
FROM

dbo.Orders

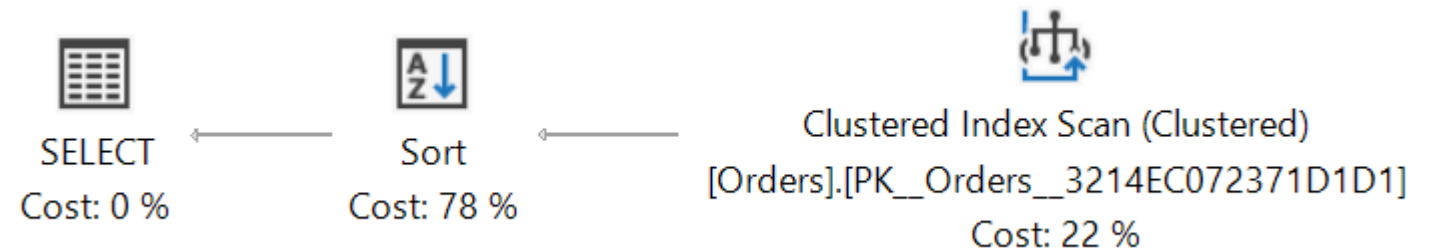
ORDER BY

CustomerId,
PurchaseDate

Might already be sorted if index exists:



Might have to sort if no index found:



Common Table Expressions (CTEs)

```
SELECT
    ...
FROM
    dbo.Customers c
    LEFT JOIN dbo.Orders o
        ON c.Id = o.CustomerId
    INNER JOIN (
        SELECT ...
        FROM dbo.OrderDetails
        WHERE OrderType = 'Online') d
        ON o.Id = d.OrderId
WHERE
    o.PurchaseDate >= DATEADD(month,-1,GETDATE())
    AND c.JoinDate >= DATEADD(day,-365,GETDATE())
```

```
WITH NewCustomers AS (
    SELECT ...
    FROM dbo.Customers
    WHERE c.JoinDate >= DATEADD(day,-365,GETDATE())
), RecentPurchases AS (
    SELECT ...
    FROM dbo.Orders
    WHERE PurchaseDate >= DATEADD(month,-1,GETDATE())
), OnlineOrdersOnly AS (
    SELECT ...
    FROM RecentPurchases o
    INNER JOIN dbo.OrderDetails d
        ON o.Id = d.OrderId
    WHERE d.OrderType = 'Online'
)
SELECT ...
FROM NewCustomers c
INNER JOIN OnlineOrdersOnly o
    ON c.Id = o.CustomerId
```

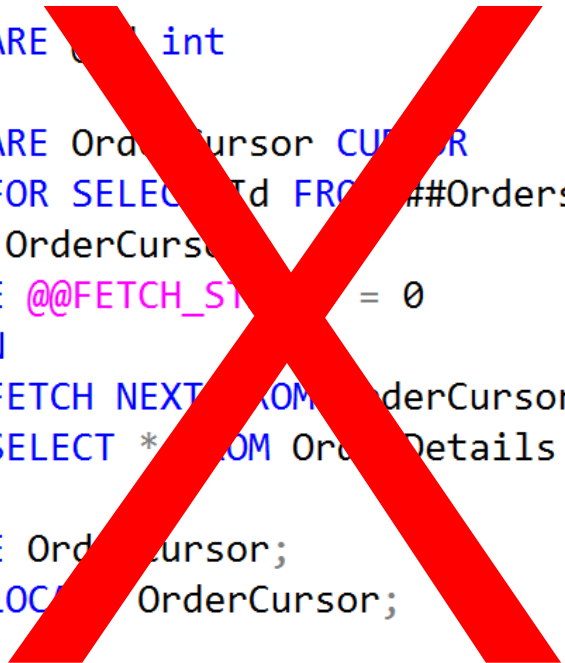




Anti-Pattern #4: Looping

Cursors and WHILE

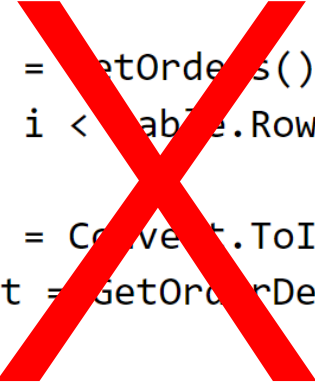
```
DECLARE @Id int  
  
DECLARE OrderCursor CURSOR  
    FOR SELECT Id FROM ##Orders  
OPEN OrderCursor  
WHILE @@FETCH_STATUS = 0  
BEGIN  
    FETCH NEXT FROM OrderCursor INTO @Id;  
    SELECT * FROM OrderDetails WHERE OrderId = @Id  
END  
CLOSE OrderCursor;  
DEALLOCATE OrderCursor;
```



```
SELECT  
    ...  
FROM  
    dbo.Orders o  
    INNER JOIN dbo.OrderDetails d  
        ON o.Id = d.OrderId
```



Eager Lazy Loading



```
DataTable table = GetOrders();  
for (int i = 0; i < table.Rows.Count; i++)  
{  
    var orderId = Convert.ToInt32(table.Rows[i]["Id"]);  
    DataTable dt = GetOrderDetails(orderId);  
}
```

```
SELECT  
    ...  
FROM  
    dbo.Orders o  
    INNER JOIN dbo.OrderDetails d  
        ON o.Id = d.OrderId
```



Correlated Subqueries

```
SELECT
    o.Id,
    (SELECT AVG(Total)
     FROM ##OrderDetails
     WHERE OrderId = o.Id
    ) AS CombinedTotal
FROM
    ##Orders o
```

Table '##OrderDetails'. Scan count 6, logical reads 12...

Table '##Orders'. Scan count 1, logical reads 1...

```
SELECT
    o.Id,
    a.AverageTotal
FROM
    ##Orders o
    INNER JOIN (
        SELECT AVG(Total) AS AverageTotal,
        OrderId
        FROM ##OrderDetails
        GROUP BY OrderId) a
        ON o.Id = a.OrderId
```

Table '##OrderDetails'. Scan count 1, logical reads 2...

Table '##Orders'. Scan count 1, logical reads 1...



A close-up photograph of a hand holding a black and silver ballpoint pen, writing on a piece of white paper. The paper has some faint, handwritten text on it. The background is blurred, showing a wooden desk and a red book. The overall lighting is soft and warm.

Anti-Pattern #5: Writing Queries Once

DISTINCT vs GROUP BY

```
SELECT DISTINCT  
    Col1,  
    Col2  
FROM  
    dbo.Table1
```

```
SELECT  
    Col1,  
    Col2  
FROM  
    dbo.Table1  
GROUP BY  
    Col1,  
    Col2
```



OR vs UNION ALL

```
SELECT
    ...
FROM
    dbo.Table1
WHERE
    Col1 = 'A'
    OR Col2 = 'A'
```

```
SELECT
    ...
FROM
    dbo.Table1
WHERE
    Col1 = 'A'
UNION ALL
SELECT
    ...
FROM
    dbo.Table1
WHERE
    Col2 = 'A'
```



NOT IN vs NOT EXISTS

```
SELECT
    Col1,
    Col2
FROM
    dbo.Table1 t1
WHERE
    Col1 NOT IN (
        SELECT Col1
        FROM dbo.Table2 t2
    )
```

```
SELECT
    Col1,
    Col2
FROM
    dbo.Table1 t1
WHERE
    NOT EXISTS (
        SELECT 1/0
        FROM dbo.Table2 t2
        WHERE t1.Col1 = t2.Col1
    )
```



JOINS vs JOINS

SELECT

...

FROM

dbo.Table1 t1

INNER JOIN dbo.Table2 t2

ON t1.Col1 = t2.Col1

SELECT

...

FROM

dbo.Table1 t1

LEFT JOIN dbo.Table2 t2

ON t1.Col1 = t2.Col1

WHERE

t2.Col1 IS NOT NULL



Correlated Subqueries vs JOINS

```
SELECT
    Col1,
    (SELECT
        Col2
    FROM
        dbo.Table2 t2
    WHERE
        t1.Col1 = t2.Col1
    ) AS Col2
FROM
    dbo.Table1 t1
```

```
SELECT
    t1.Col1,
    t2.Col2
FROM
    dbo.Table1 t1
    LEFT JOIN dbo.Table2 t2
        ON t1.Col1 = t2.Col1
```



Forcing Join Order

SELECT

...

FROM

dbo.Table1 t1

INNER JOIN dbo.Table2 t2

ON t1.Col1 = t2.Col1

INNER JOIN dbo.Table3 t3

ON t2.Col2 = t3.Col2

SELECT

...

FROM

dbo.Table1 t1

INNER JOIN (

-- More rows than your query
-- will ever return

SELECT TOP 2147483647

t2.Col1, ...

FROM

dbo.Table2 t2

INNER JOIN dbo.Table3 t3

ON t2.Col2 = t3.Col2

) t4

ON t1.Col1 = t4.Col1



Break up giant queries

```
WITH InitialData AS (  
    ...  
) , FilteredData AS (  
    ...  
) , JoinedData AS (  
    ...  
)  
SELECT ...  
FROM JoinedData
```

```
WITH InitialData AS (  
    ...  
) , FilteredData AS (  
    ...  
)  
SELECT * INTO #FilteredData FROM FilteredData;  
CREATE CLUSTERED INDEX ON #FilteredData (Col1, Col2);  
  
WITH JoinedData AS (  
    SELECT ... FROM #FilteredData  
    ...  
)  
SELECT ...  
FROM JoinedData
```



Review

1. Move smaller data to where the larger data exists
2. Allow index use
3. Use the right tools for the job
4. Think in sets instead of looping
5. Rewrite your queries



Thank you!

SQL WITH BERT

New Episodes Every
Tuesday



[@bertwagner](https://twitter.com/bertwagner)



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

← New posts and videos
every Tuesday!



@bertwagner



bertwagner.com



youtube.com/bertwagner



bert@bertwagner.com

Appendix

- 19
 - <https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-query-performance?view=sql-server-2017>
 - <http://www.nikoport.com/columnstore/>
- 20
 - <https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-2017>
 - <https://docs.microsoft.com/en-us/sql/t-sql/functions/analytic-functions-transact-sql?view=sql-server-2017>
- 21
 - <https://dba.stackexchange.com/a/158429/168918>
- 28
 - <https://sqlperformance.com/2017/01/t-sql-queries/surprises-assumptions-group-by-distinct>
- 29
 - <https://bertwagner.com/2018/02/20/or-vs-union-all-is-one-better-for-performance/>
- 30
 - <https://sqlperformance.com/2012/12/t-sql-queries/left-anti-semi-join>
- 33
 - <https://bertwagner.com/2017/11/21/does-the-join-order-of-my-tables-matter/>
 - [https://sqlbits.com/Sessions/Event14/Query Tuning Mastery Clash of the Row Goals](https://sqlbits.com/Sessions/Event14/Query_Tuning_Mastery_Clash_of_the_Row_Goals)

