Bad practices caught in the wild

Peter Kruis

Who are you?
You are a SQL developer, but when you handover your code to your senior, you sometimes might hear: “Don’t do this anymore, it’s a bad practice!”. If you want to learn more about those bad practices which I see are being used a lot, this session is for you!

How is this session going to help you?!
*In this session you will learn about some of those "code smells". I will show you an example of the bad practice, explain why it should be avoided and give a possible solution for them. We are going to talk about: Functions in the WHERE clause, Implicit casting, SELECT * and some others. This knowledge will help you make your queries run faster, need less resources and therefore will make your DBA and users happy.*

SQLSATURDAY 2023 Oregon & SW Washington

Thank You Sponsors!

Our sponsors make this event possible. Please thank and support them.

Microsoft

OMNIDATA™

VirtueTech Inc
Solutions Made Simple

Quest

SIOS

silk

COZYROC

redgate

Peter Kruis

https://www.linkedin.com/in/peter-kruis

peterkruis@hotmail.com

What to expect from this session

To expect

- Smelly code examples
- Explanation about why it is bad
- Possible fixes

Not to expect

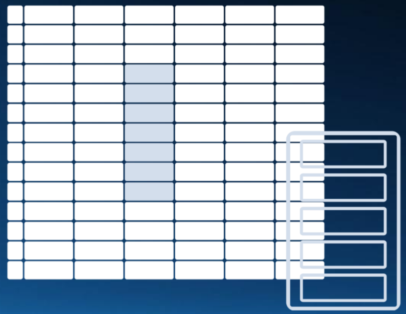
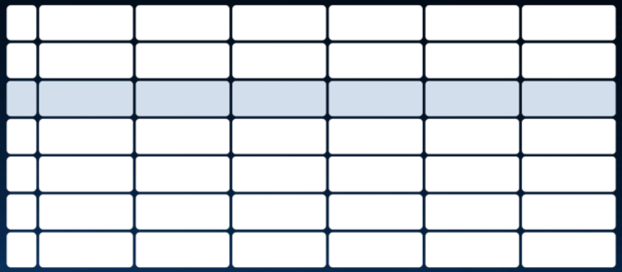
- Environment optimization
  - Server
  - Hardware
  - SQL Configuration
  - Etc.
- Index optimization
- Finding problem queries
  - (Brent Ozar: First Responder Kit)

User Defined Scalar Functions

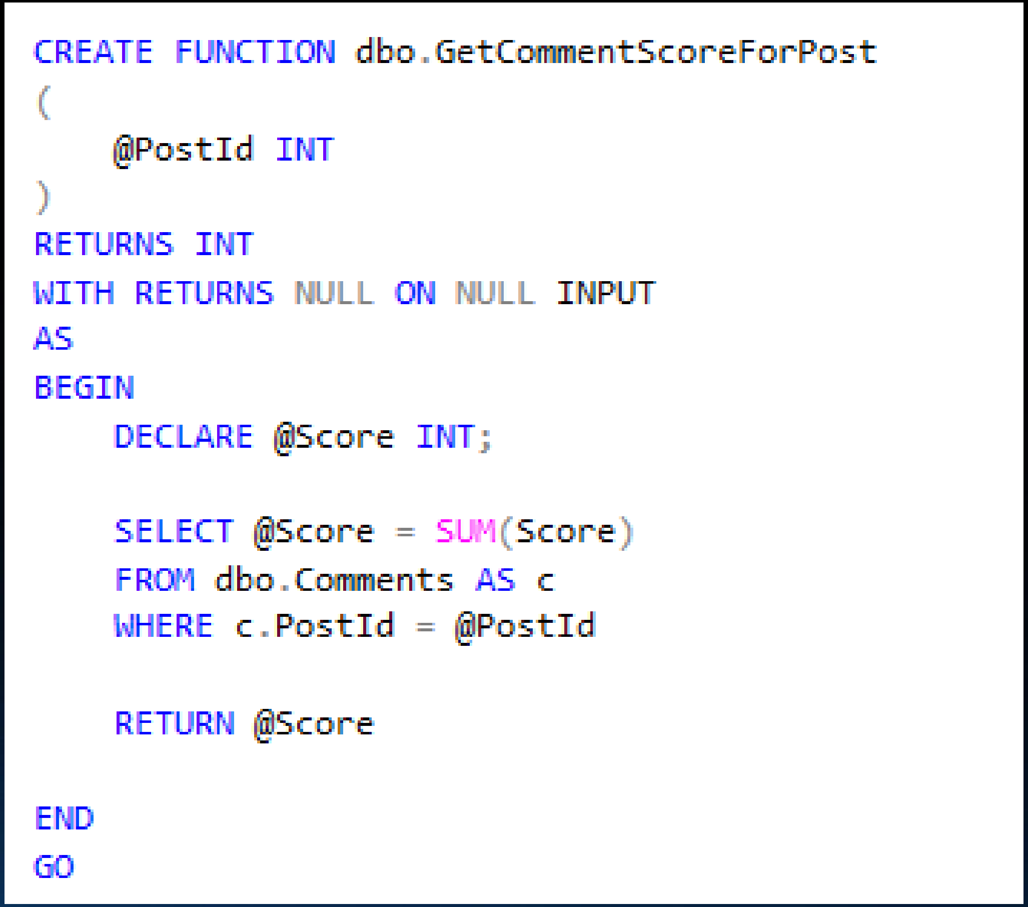
User Defined Scalar Functions takes one or more parameters and returns a single value.

This may help you with simplifying your code, you can reuse scalar functions in many queries; Just like we learned when we are using other programming languages, like .NET, Java, etc..

Sounds great.. Right?

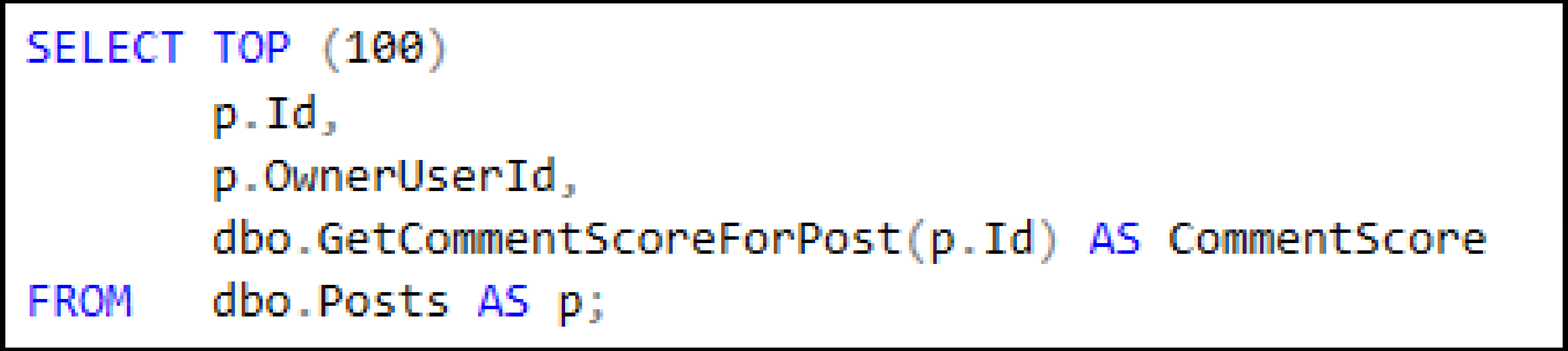



Example

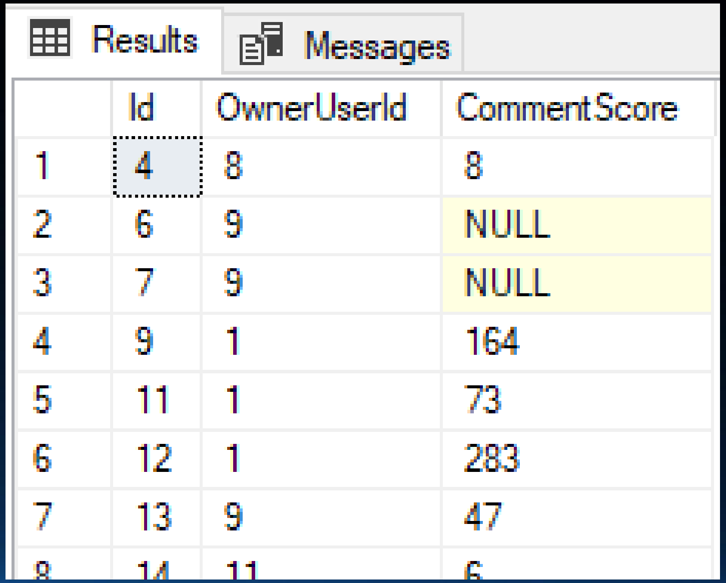


```
CREATE FUNCTION dbo.GetCommentScoreForPost
(
    @PostId INT
)
RETURNS INT
WITH RETURNS NULL ON NULL INPUT
AS
BEGIN
    DECLARE @Score INT;

    SELECT @Score = SUM(Score)
    FROM dbo.Comments AS c
    WHERE c.PostId = @PostId

    RETURN @Score

END
GO
```

Example

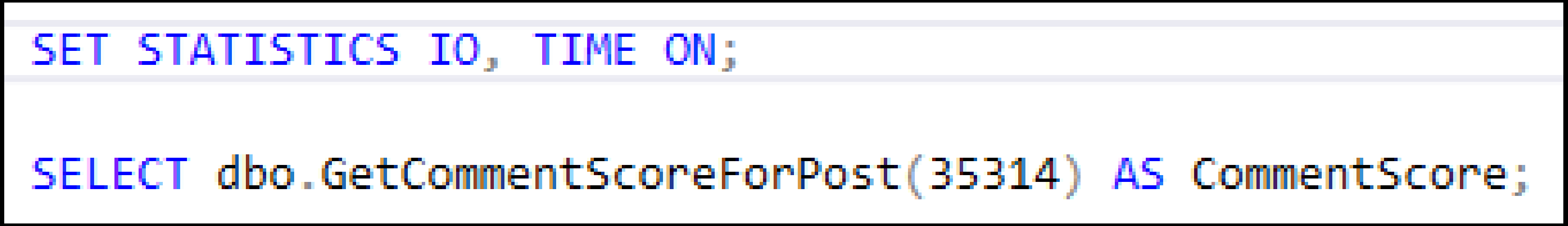


```sql
SELECT TOP (100)
        p.Id,
        p.OwnerUserId,
        dbo.GetCommentScoreForPost(p.Id) AS CommentScore
FROM    dbo.Posts AS p;
```

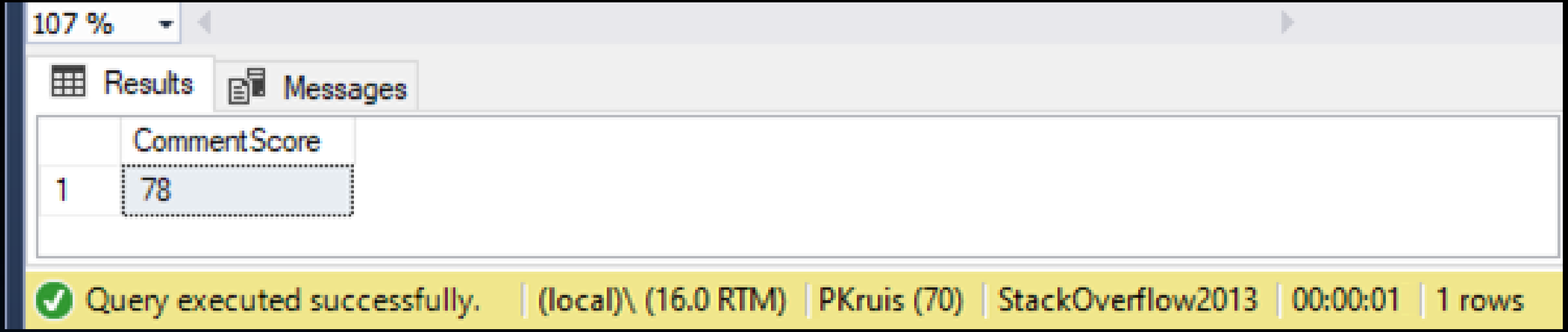


Results | Messages

	Id	OwnerUserId	CommentScore
1	4	8	8
2	6	9	NULL
3	7	9	NULL
4	9	1	164
5	11	1	73
6	12	1	283
7	13	9	47
8	14	11	6

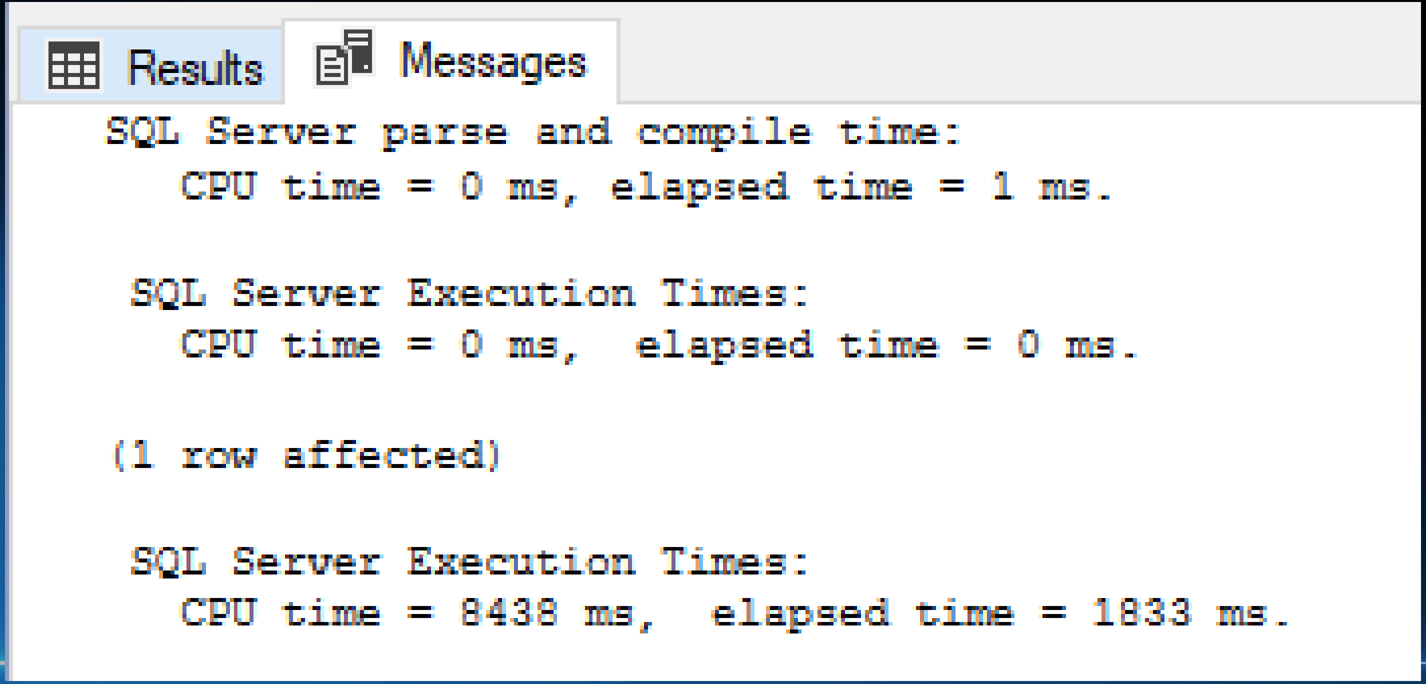


```sql
SET STATISTICS IO, TIME ON;

SELECT dbo.GetCommentScoreForPost(35314) AS CommentScore;
```

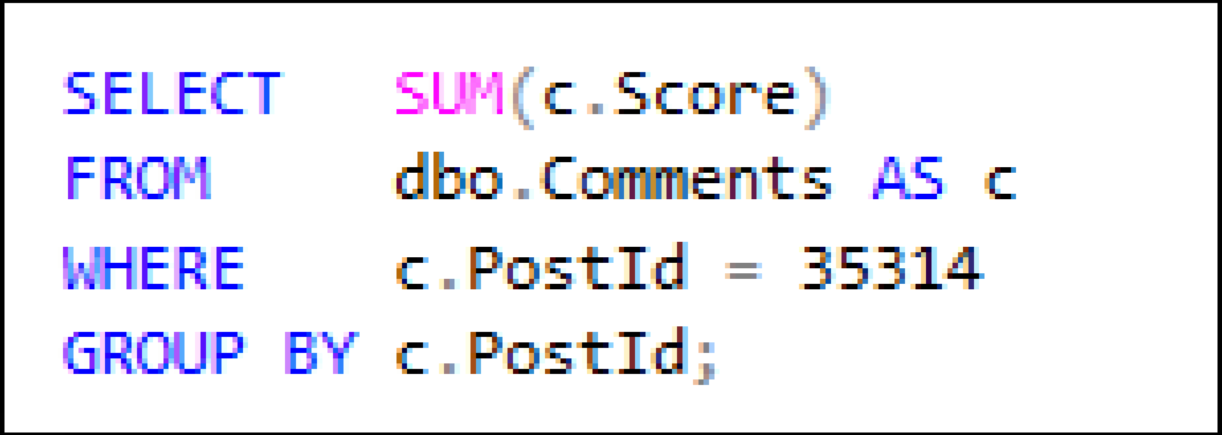


	CommentScore
1	78

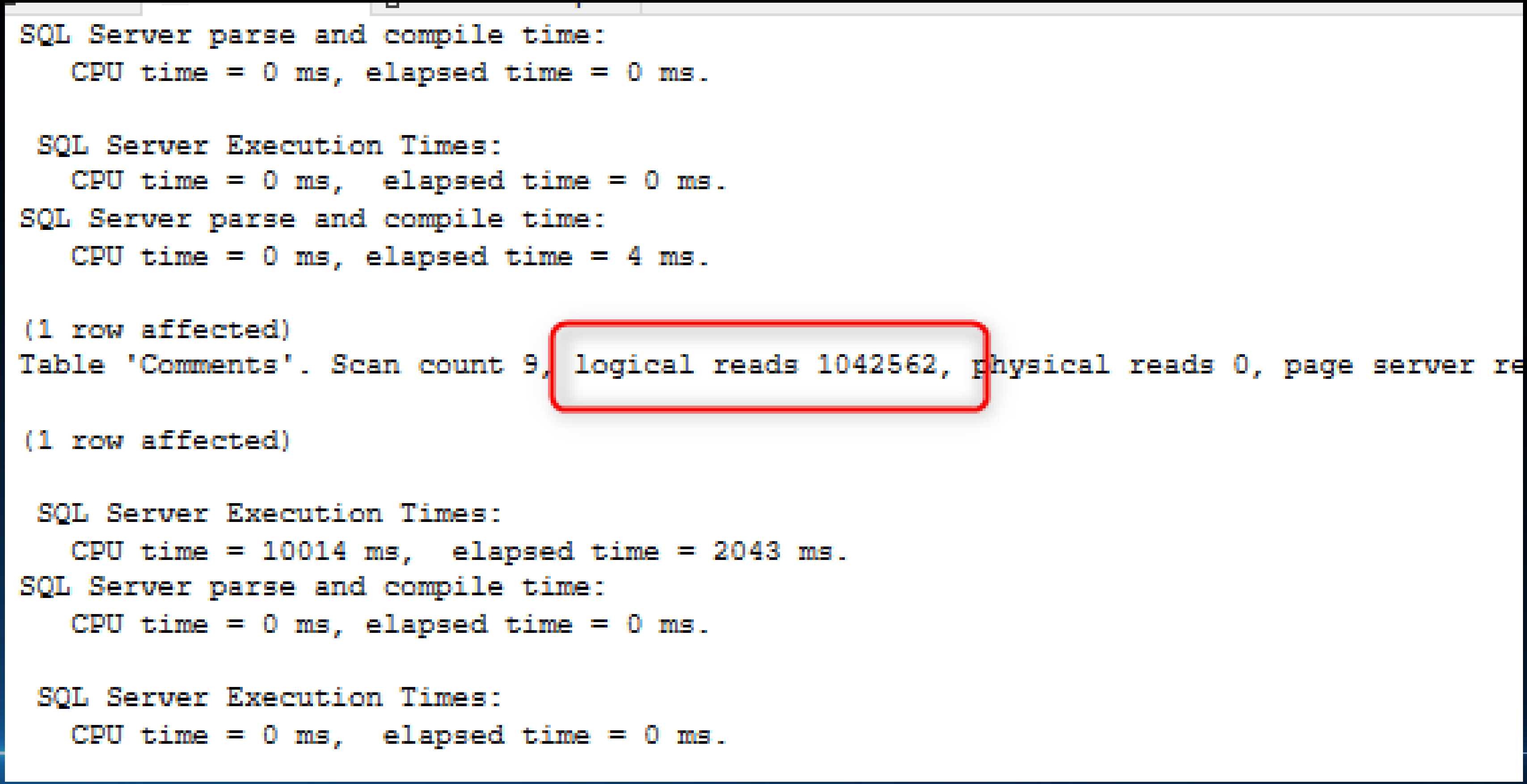
Query executed successfully. | (local)\ (16.0 RTM) | PKruis (70) | StackOverflow2013 | 00:00:01 | 1 rows



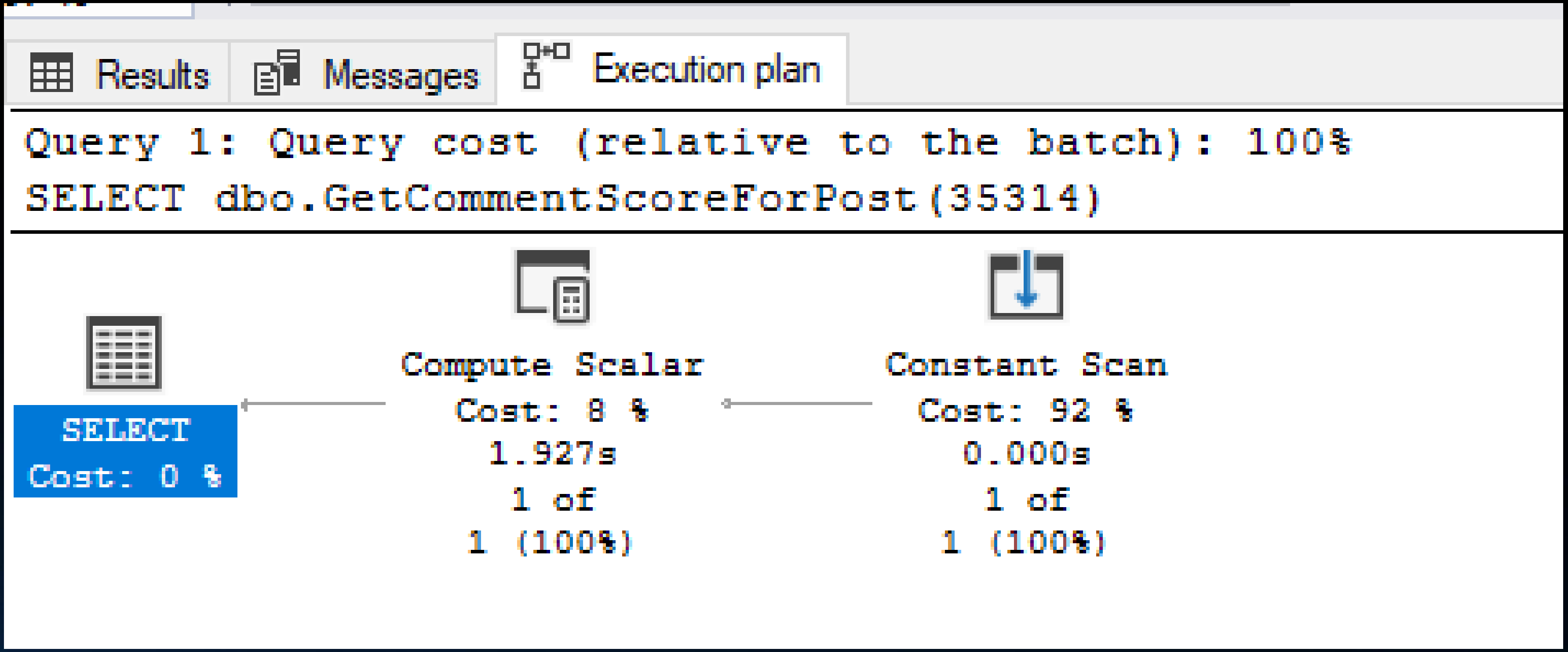
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(1 row affected)

 SQL Server Execution Times:
   CPU time = 8438 ms,  elapsed time = 1833 ms.
```



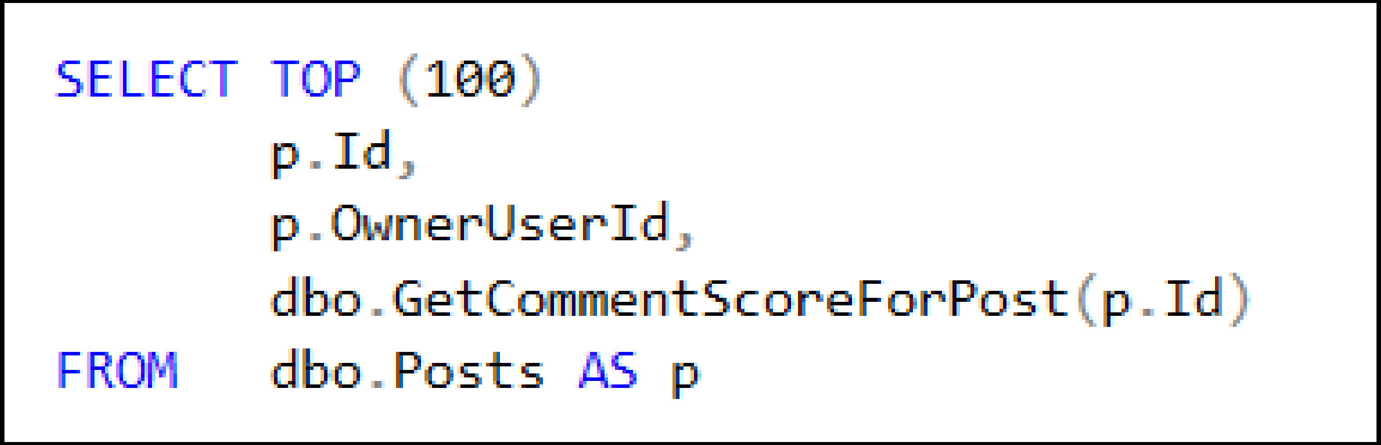
```sql
SELECT   SUM(c.Score)
FROM     dbo.Comments AS c
WHERE    c.PostId = 35314
GROUP BY c.PostId;
```



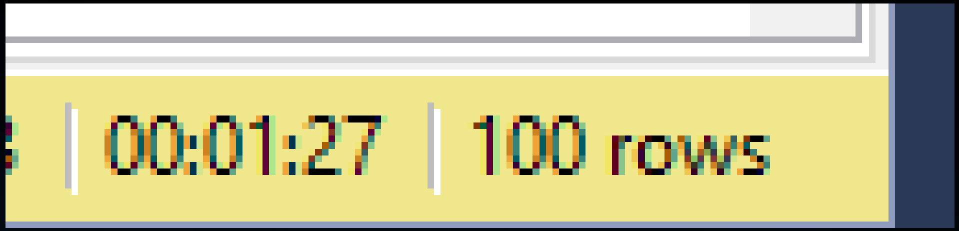
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 4 ms.

(1 row affected)
Table 'Comments'. Scan count 9, logical reads 1042562, physical reads 0, page server re

(1 row affected)

 SQL Server Execution Times:
   CPU time = 10014 ms,  elapsed time = 2043 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```



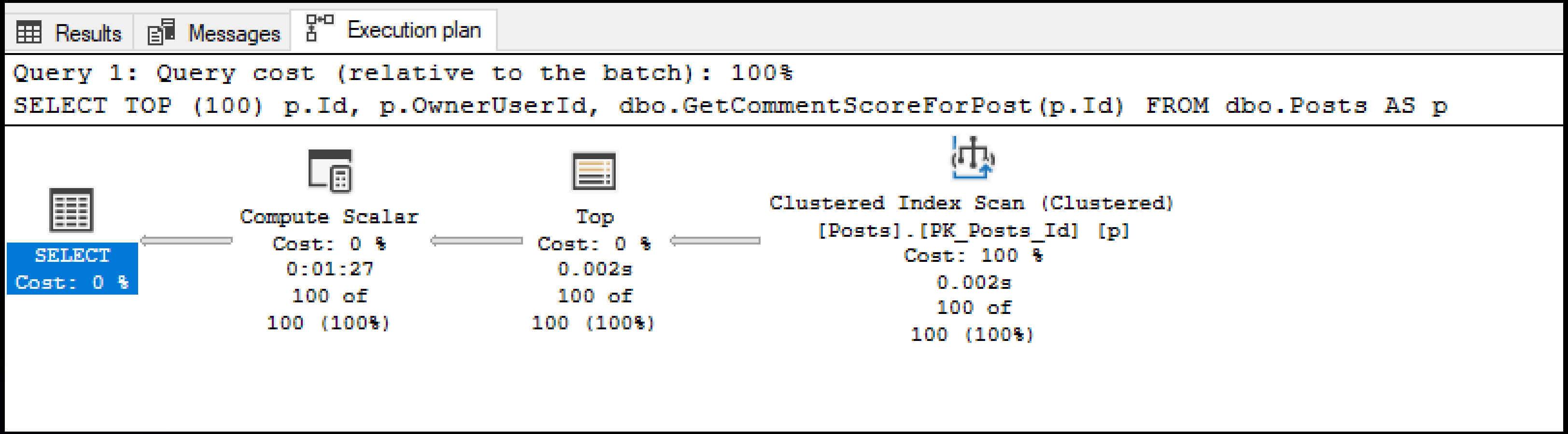
```
Query 1: Query cost (relative to the batch): 100%
SELECT dbo.GetCommentScoreForPost(35314)
```



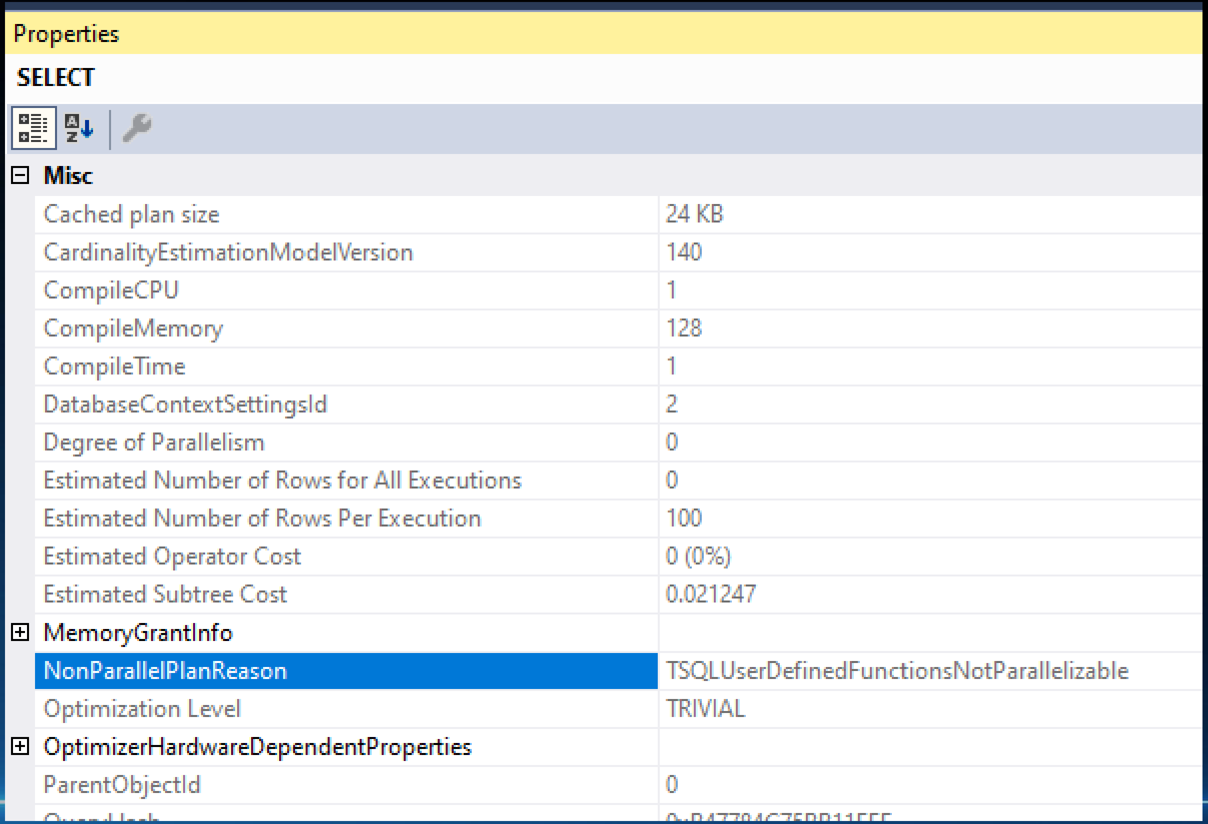
```
SELECT TOP (100)
       p.Id,
       p.OwnerUserId,
       dbo.GetCommentScoreForPost(p.Id)
FROM   dbo.Posts AS p
```



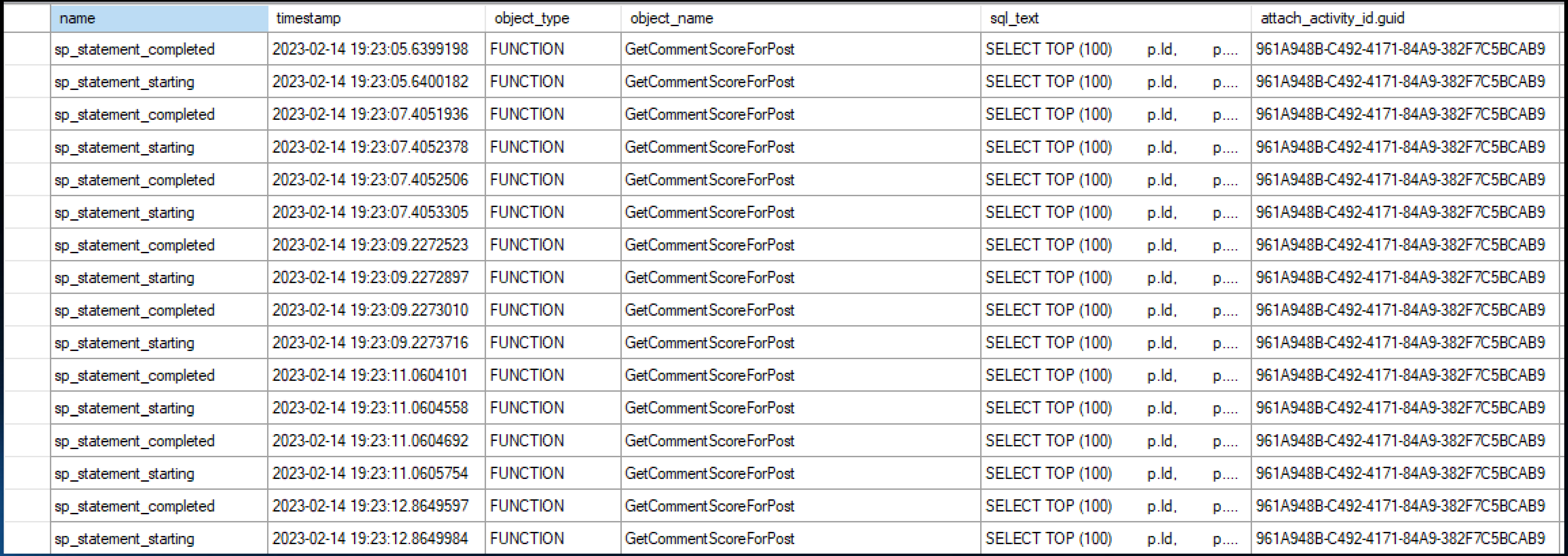
```
00:01:27 | 100 rows
```



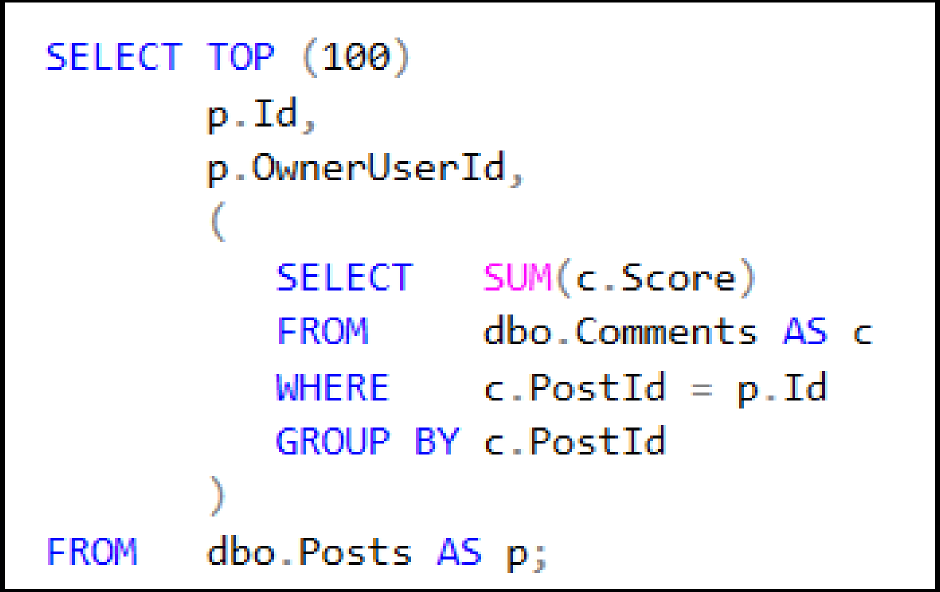
```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP (100) p.Id, p.OwnerUserId, dbo.GetCommentScoreForPost(p.Id) FROM dbo.Posts AS p
```



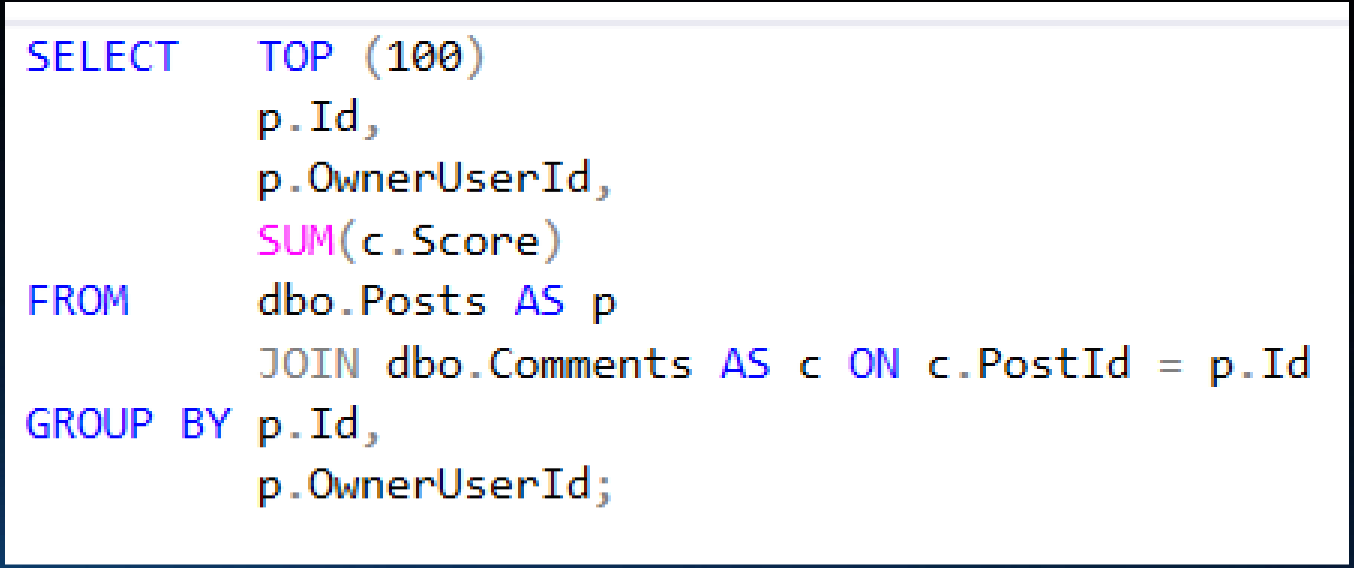
Cached plan size	24 KB
CardinalityEstimationModelVersion	140
CompileCPU	1
CompileMemory	128
CompileTime	1
DatabaseContextSettingsId	2
Degree of Parallelism	0
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	100
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.021247
MemoryGrantInfo	
NonParallelPlanReason	TSQLUserDefinedFunctionsNotParallelizable
Optimization Level	TRIVIAL
OptimizerHardwareDependentProperties	
ParentObjectId	0



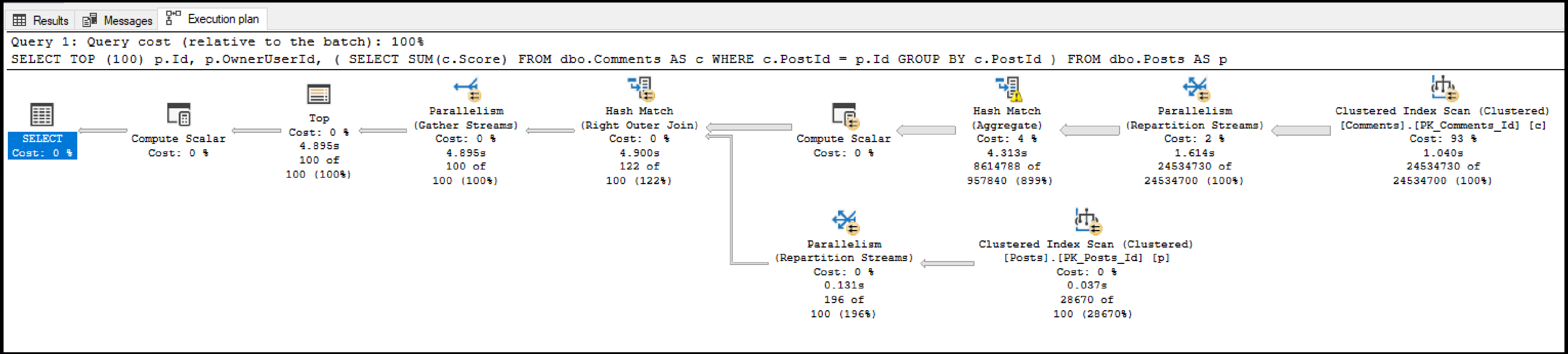
name	timestamp	object_type	object_name	sql_text	attach_activity_id.guid
sp_statement_completed	2023-02-14 19:23:05.6399198	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:05.6400182	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:07.4051936	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:07.4052378	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:07.4052506	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:07.4053305	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:09.2272523	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:09.2272897	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:09.2273010	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:09.2273716	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:11.0604101	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:11.0604558	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:11.0604692	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:11.0605754	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_completed	2023-02-14 19:23:12.8649597	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9
sp_statement_starting	2023-02-14 19:23:12.8649984	FUNCTION	GetCommentScoreForPost	SELECT TOP (100) p.Id, p....	961A948B-C492-4171-84A9-382F7C5BCAB9



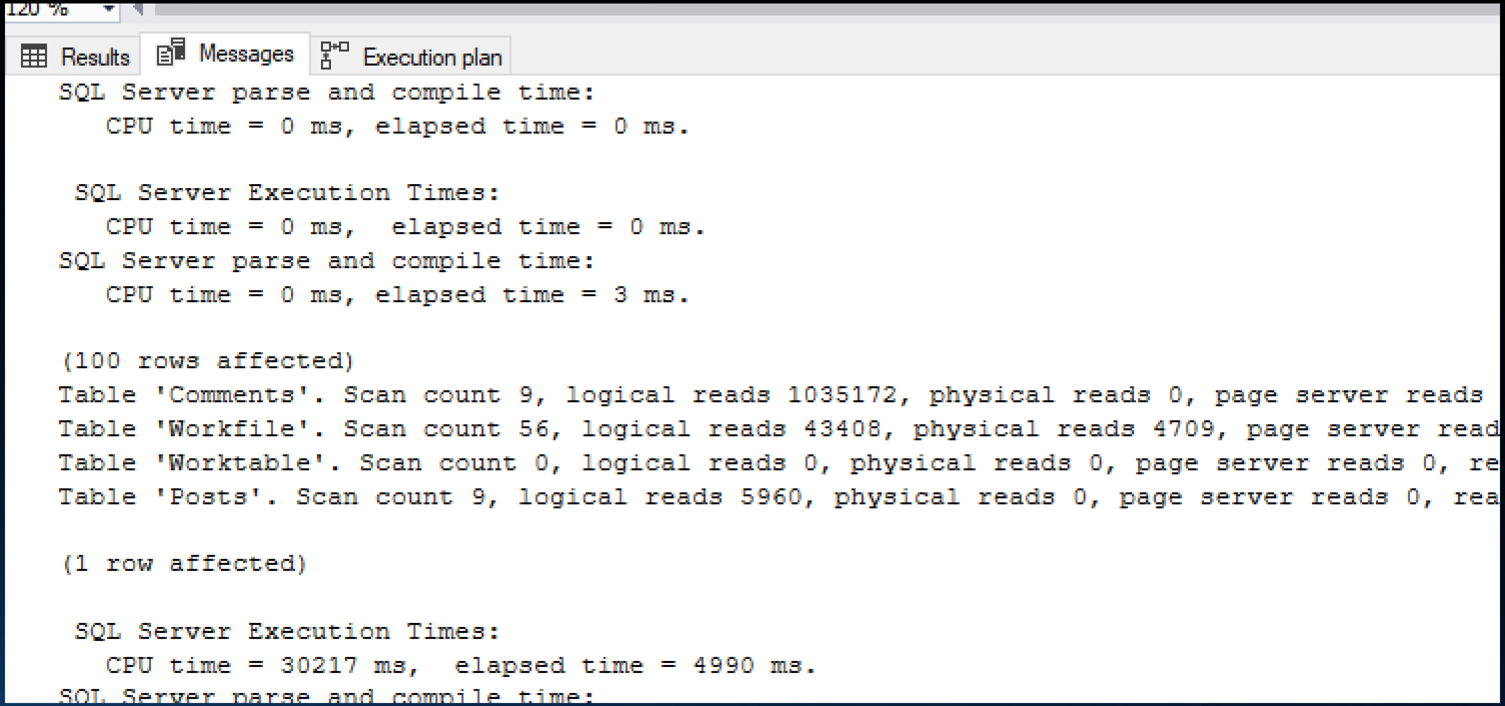
```sql
SELECT TOP (100)
       p.Id,
       p.OwnerUserId,
       (
          SELECT   SUM(c.Score)
          FROM     dbo.Comments AS c
          WHERE    c.PostId = p.Id
          GROUP BY c.PostId
       )
FROM   dbo.Posts AS p;
```



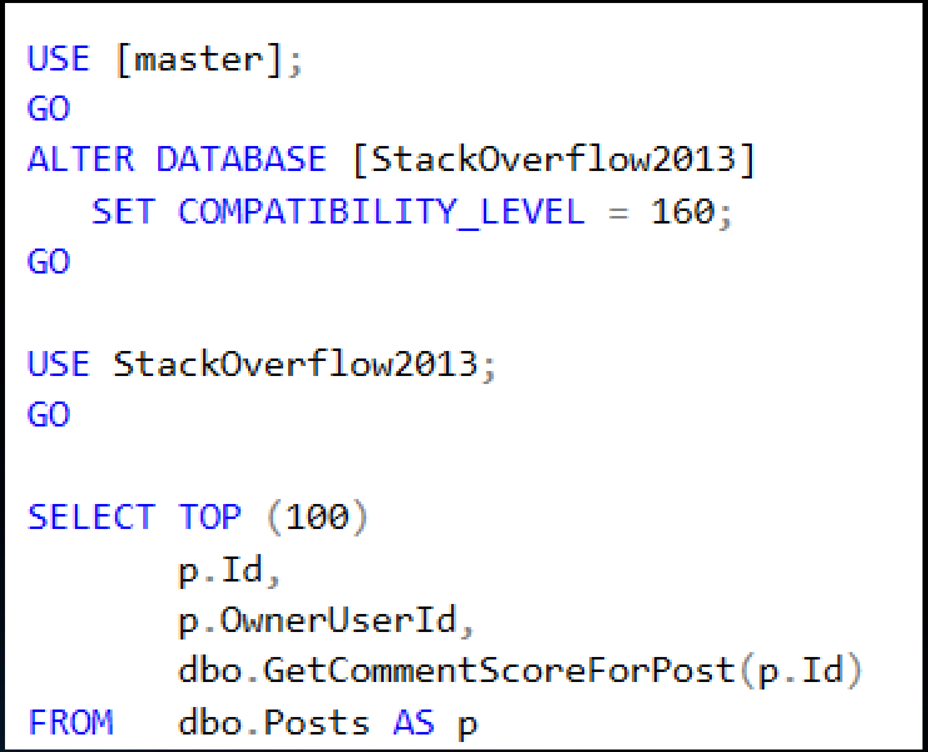
```sql
SELECT   TOP (100)
         p.Id,
         p.OwnerUserId,
         SUM(c.Score)
FROM     dbo.Posts AS p
         JOIN dbo.Comments AS c ON c.PostId = p.Id
GROUP BY p.Id,
         p.OwnerUserId;
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP (100) p.Id, p.OwnerUserId, ( SELECT SUM(c.Score) FROM dbo.Comments AS c WHERE c.PostId = p.Id GROUP BY c.PostId ) FROM dbo.Posts AS p
```

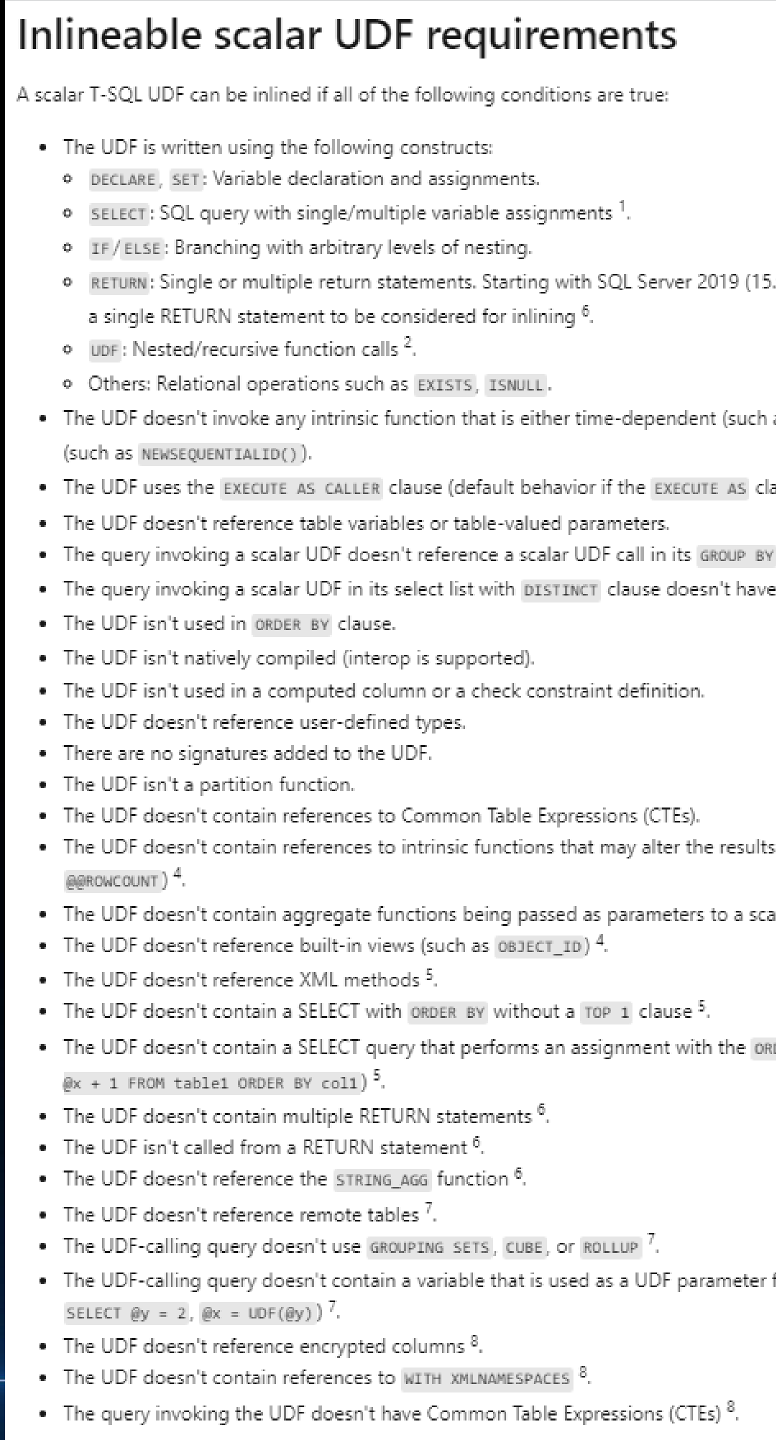


```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 3 ms.

(100 rows affected)
Table 'Comments'. Scan count 9, logical reads 1035172, physical reads 0, page server reads
Table 'Workfile'. Scan count 56, logical reads 43408, physical reads 4709, page server read
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, re
Table 'Posts'. Scan count 9, logical reads 5960, physical reads 0, page server reads 0, rea

(1 row affected)

 SQL Server Execution Times:
   CPU time = 30217 ms,  elapsed time = 4990 ms.
SQL Server parse and compile time:
```



```sql
USE [master];
GO
ALTER DATABASE [StackOverflow2013]
   SET COMPATIBILITY_LEVEL = 160;
GO

USE StackOverflow2013;
GO

SELECT TOP (100)
       p.Id,
       p.OwnerUserId,
       dbo.GetCommentScoreForPost(p.Id)
FROM   dbo.Posts AS p
```

https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining?view=sql-server-ver16#requirements



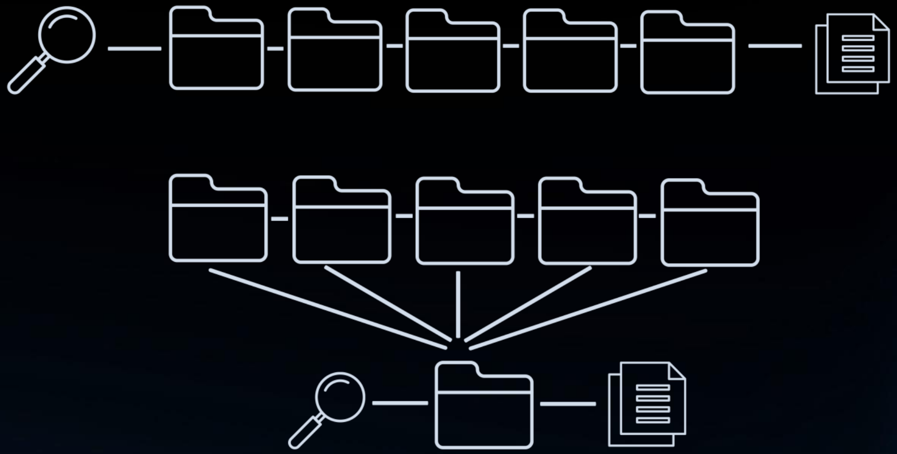
Inlineable scalar UDF requirements

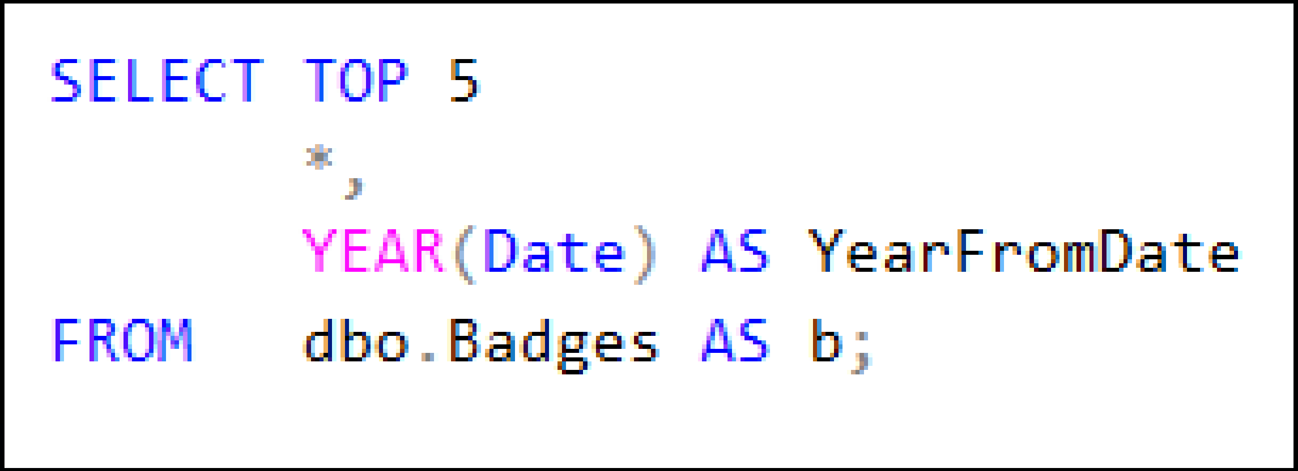
A scalar T-SQL UDF can be inlined if all of the following conditions are true:

- The UDF is written using the following constructs:
  - `DECLARE`, `SET`: Variable declaration and assignments.
  - `SELECT`: SQL query with single/multiple variable assignments [1].
  - `IF`/`ELSE`: Branching with arbitrary levels of nesting.
  - `RETURN`: Single or multiple return statements. Starting with SQL Server 2019 (15.
    a single RETURN statement to be considered for inlining [6].
  - `UDF`: Nested/recursive function calls [2].
  - Others: Relational operations such as `EXISTS`, `ISNULL`.
- The UDF doesn't invoke any intrinsic function that is either time-dependent (such
  (such as `NEWSEQUENTIALID()`).
- The UDF uses the `EXECUTE AS CALLER` clause (default behavior if the `EXECUTE AS` cla
- The UDF doesn't reference table variables or table-valued parameters.
- The query invoking a scalar UDF doesn't reference a scalar UDF call in its `GROUP BY`
- The query invoking a scalar UDF in its select list with `DISTINCT` clause doesn't have
- The UDF isn't used in `ORDER BY` clause.
- The UDF isn't natively compiled (interop is supported).
- The UDF isn't used in a computed column or a check constraint definition.
- The UDF doesn't reference user-defined types.
- There are no signatures added to the UDF.
- The UDF isn't a partition function.
- The UDF doesn't contain references to Common Table Expressions (CTEs).
- The UDF doesn't contain references to intrinsic functions that may alter the results
  `@@ROWCOUNT`) [4].
- The UDF doesn't contain aggregate functions being passed as parameters to a sca
- The UDF doesn't reference built-in views (such as `OBJECT_ID`) [4].
- The UDF doesn't reference XML methods [5].
- The UDF doesn't contain a SELECT with `ORDER BY` without a `TOP 1` clause [5].
- The UDF doesn't contain a SELECT query that performs an assignment with the `ORI`
  `@x + 1 FROM table1 ORDER BY col1`) [5].
- The UDF doesn't contain multiple RETURN statements [6].
- The UDF isn't called from a RETURN statement [6].
- The UDF doesn't reference the `STRING_AGG` function [6].
- The UDF doesn't reference remote tables [7].
- The UDF-calling query doesn't use `GROUPING SETS`, `CUBE`, or `ROLLUP` [7].
- The UDF-calling query doesn't contain a variable that is used as a UDF parameter f
  `SELECT @y = 2`, `@x = UDF(@y)`) [7].
- The UDF doesn't reference encrypted columns [8].
- The UDF doesn't contain references to `WITH XMLNAMESPACES` [8].
- The query invoking the UDF doesn't have Common Table Expressions (CTEs) [8].

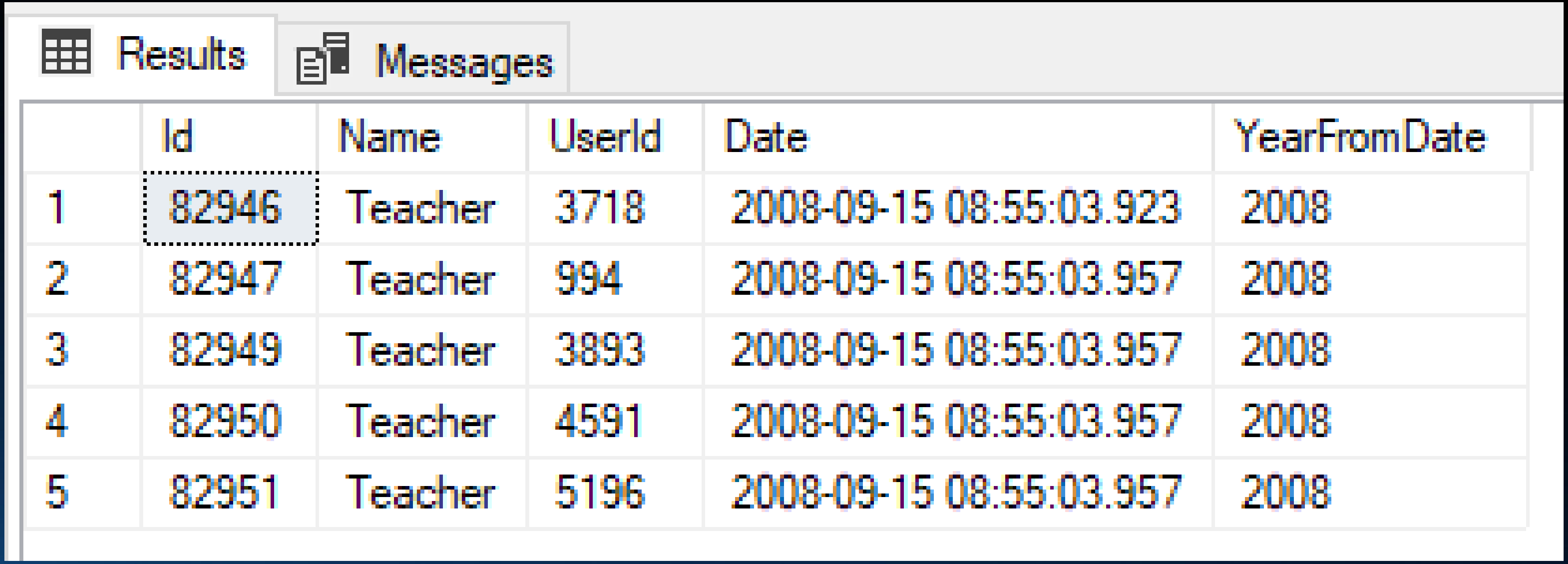
Functions in the WHERE clause

Function	Purpose
CONCAT	Adds two or more strings together
FORMAT	Formats a value with the specified format *(don't use this one at all, let your application format your output)*
LEFT	Extracts a number of characters from a string (starting from left)
CEILING	Returns the smallest integer value that is >= a number
DATEADD	Adds a time/date interval to a date and then returns the date
YEAR	Returns the year part for a specified date
ISNULL	Return a specified value if the expression is NULL, otherwise return the expression

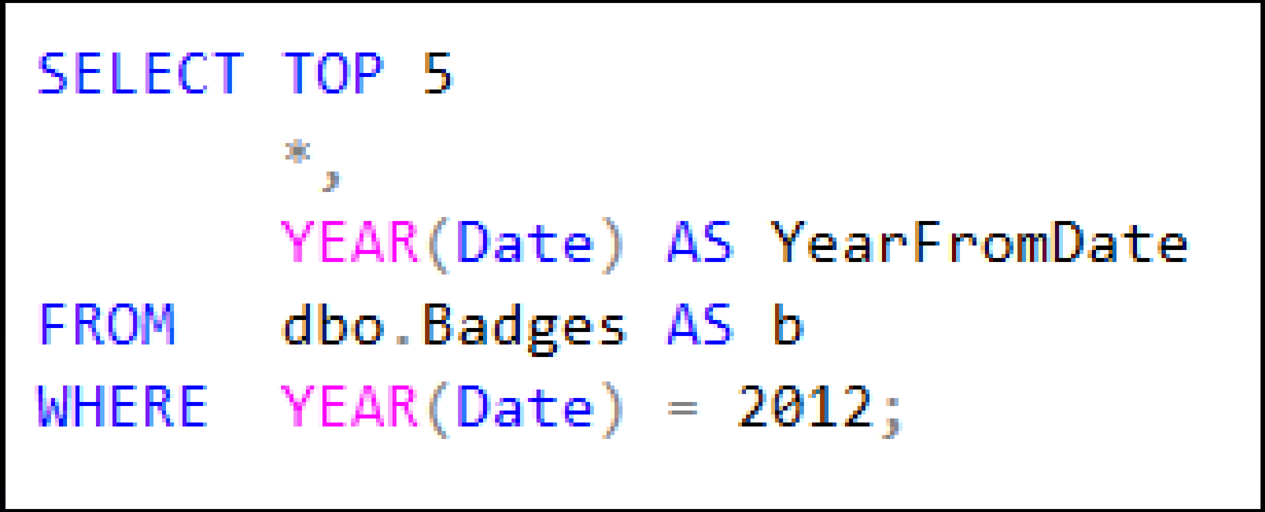




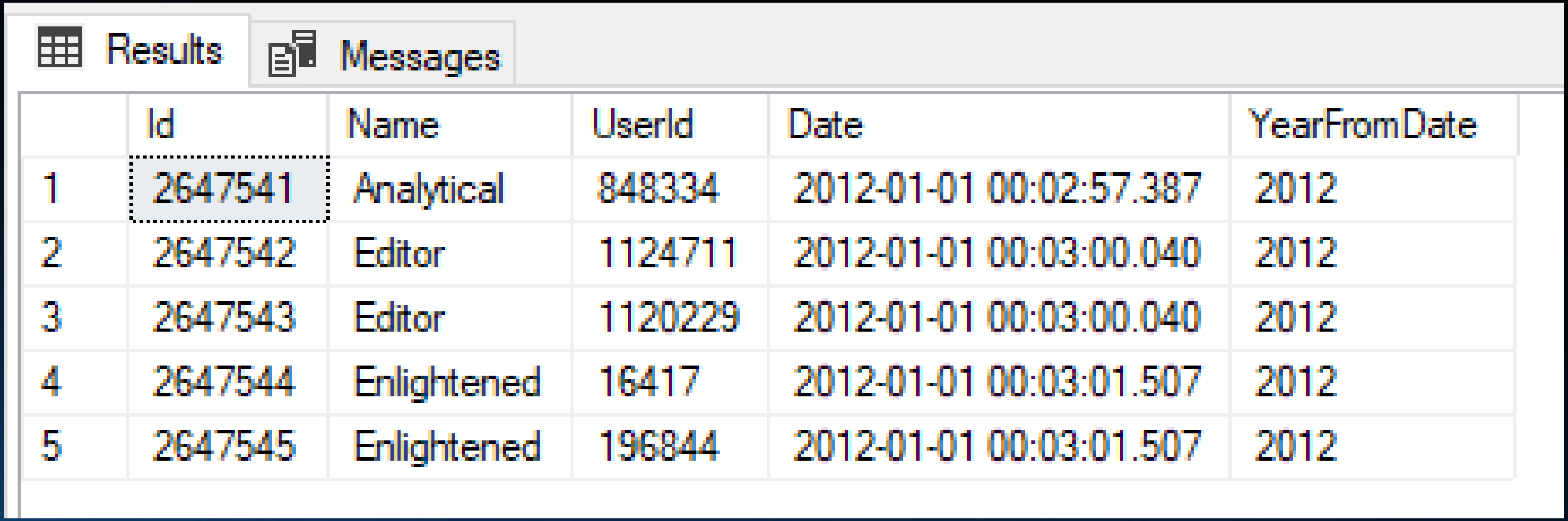
```sql
SELECT TOP 5
        *,
        YEAR(Date) AS YearFromDate
FROM    dbo.Badges AS b;
```



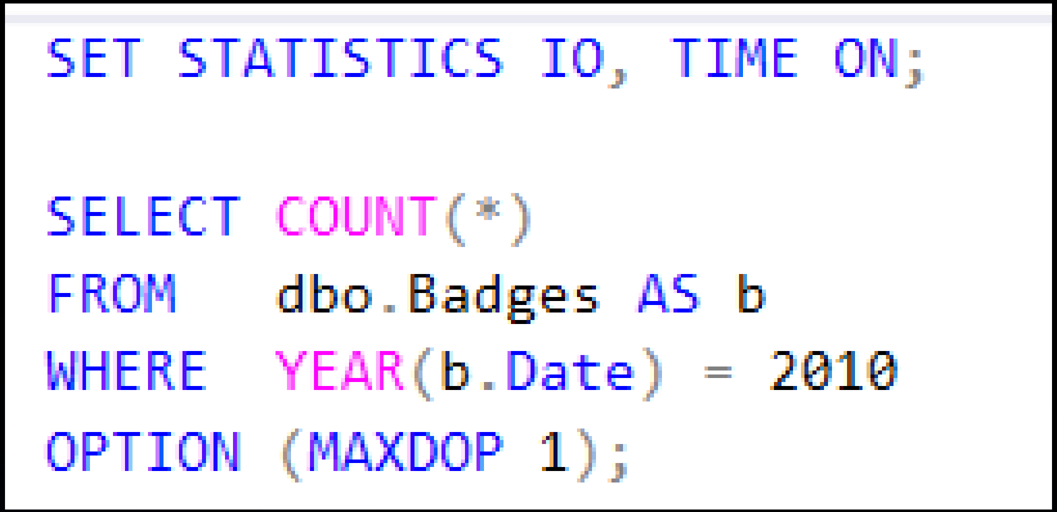
	Id	Name	UserId	Date	YearFromDate
1	82946	Teacher	3718	2008-09-15 08:55:03.923	2008
2	82947	Teacher	994	2008-09-15 08:55:03.957	2008
3	82949	Teacher	3893	2008-09-15 08:55:03.957	2008
4	82950	Teacher	4591	2008-09-15 08:55:03.957	2008
5	82951	Teacher	5196	2008-09-15 08:55:03.957	2008



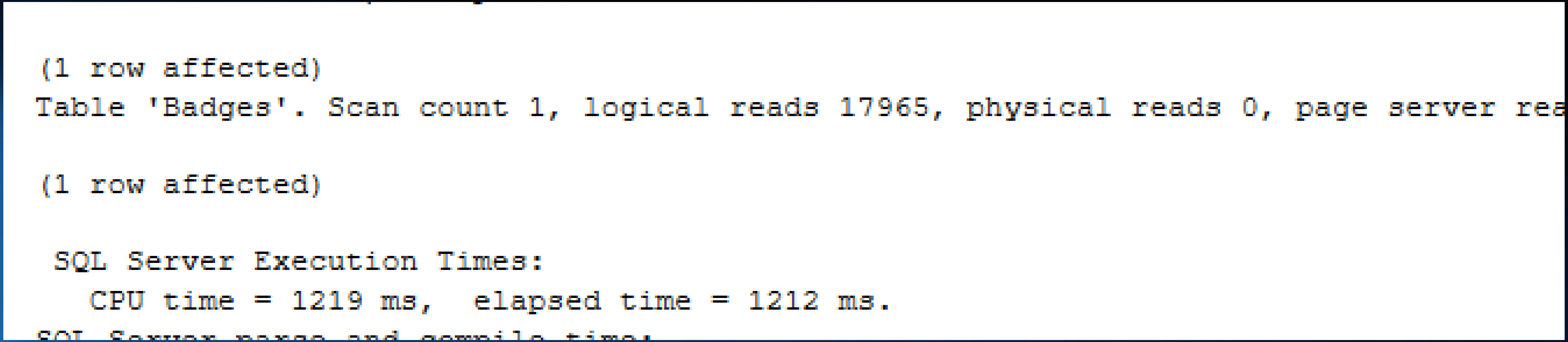
```
SELECT TOP 5
        *,
        YEAR(Date) AS YearFromDate
FROM    dbo.Badges AS b
WHERE   YEAR(Date) = 2012;
```



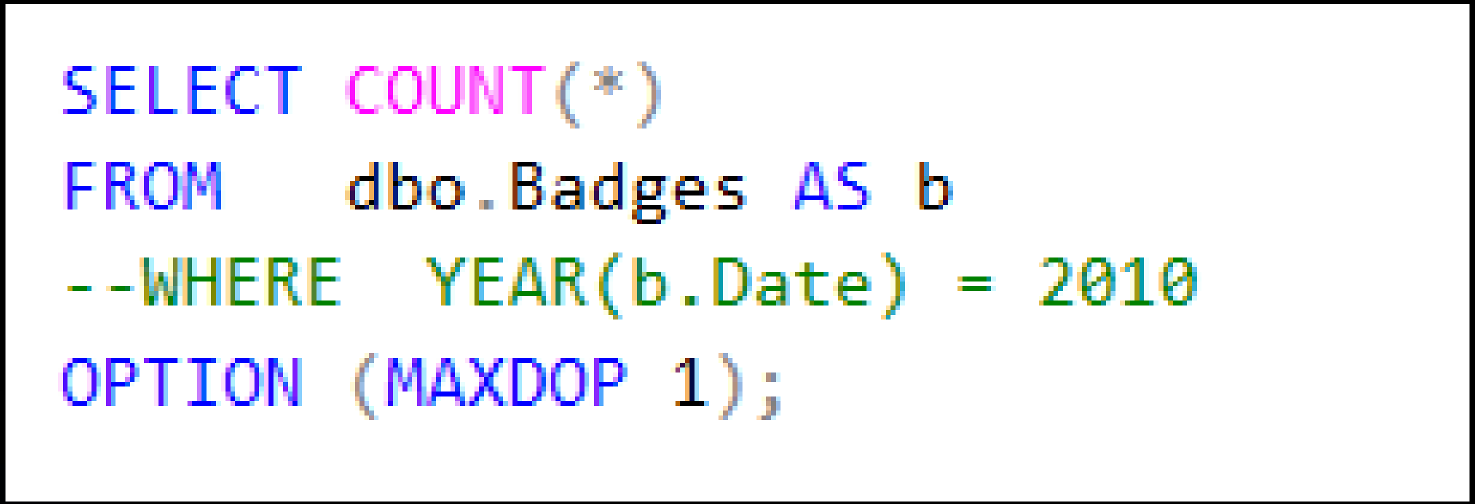
	Id	Name	UserId	Date	YearFromDate
1	2647541	Analytical	848334	2012-01-01 00:02:57.387	2012
2	2647542	Editor	1124711	2012-01-01 00:03:00.040	2012
3	2647543	Editor	1120229	2012-01-01 00:03:00.040	2012
4	2647544	Enlightened	16417	2012-01-01 00:03:01.507	2012
5	2647545	Enlightened	196844	2012-01-01 00:03:01.507	2012



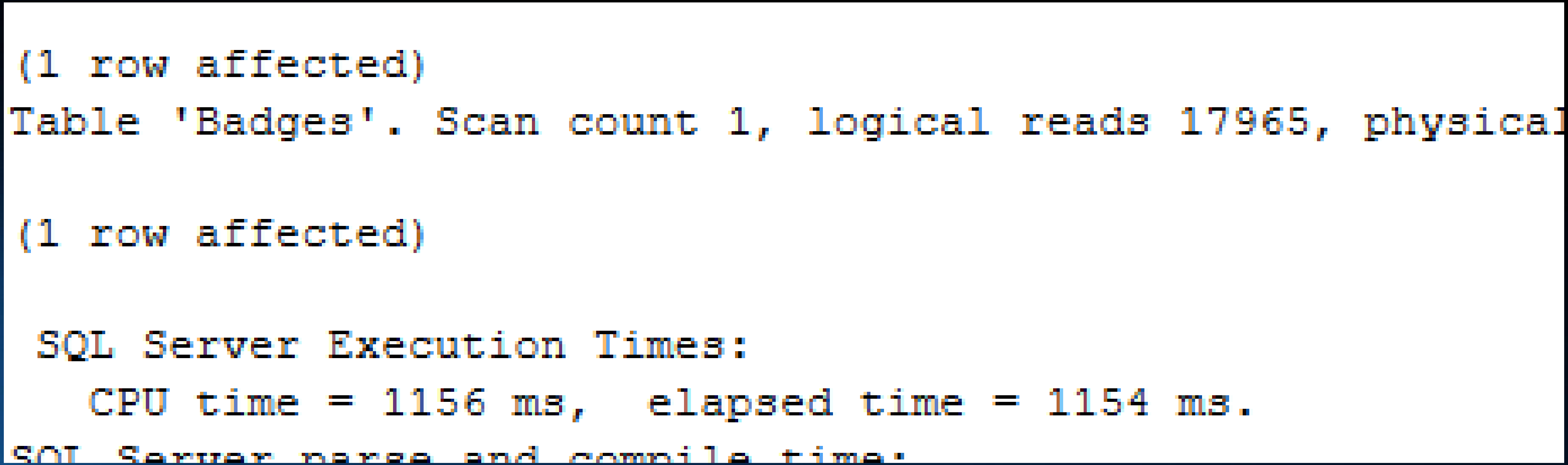
```
SET STATISTICS IO, TIME ON;

SELECT COUNT(*)
FROM   dbo.Badges AS b
WHERE  YEAR(b.Date) = 2010
OPTION (MAXDOP 1);
```



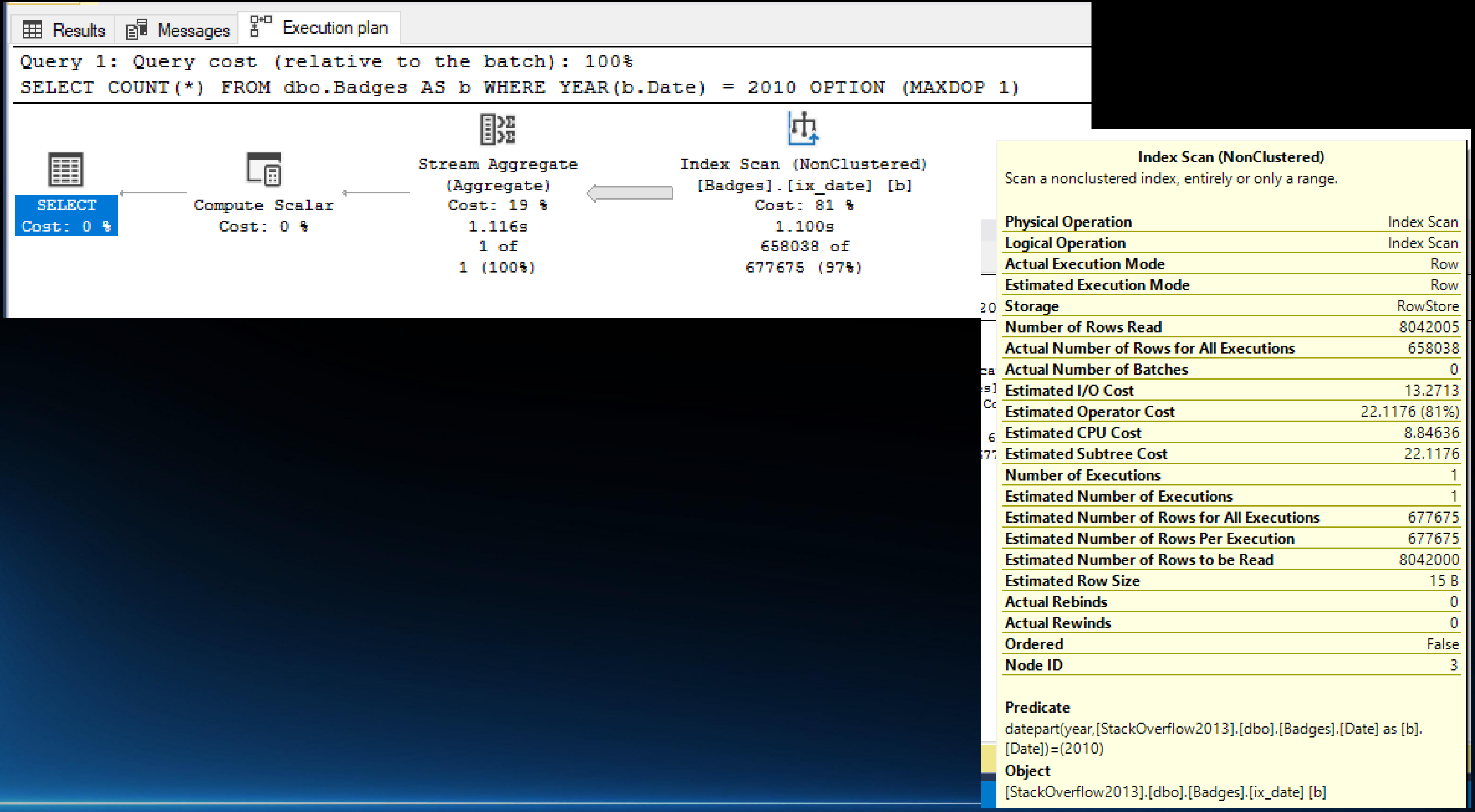
```
(1 row affected)
Table 'Badges'. Scan count 1, logical reads 17965, physical reads 0, page server rea

(1 row affected)

 SQL Server Execution Times:
   CPU time = 1219 ms,  elapsed time = 1212 ms.
```



```sql
SELECT COUNT(*)
FROM   dbo.Badges AS b
--WHERE  YEAR(b.Date) = 2010
OPTION (MAXDOP 1);
```



```
(1 row affected)
Table 'Badges'. Scan count 1, logical reads 17965, physical

(1 row affected)

 SQL Server Execution Times:
   CPU time = 1156 ms,  elapsed time = 1154 ms.
SQL Server parse and compile time:
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT(*) FROM dbo.Badges AS b WHERE YEAR(b.Date) = 2010 OPTION (MAXDOP 1)
```

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

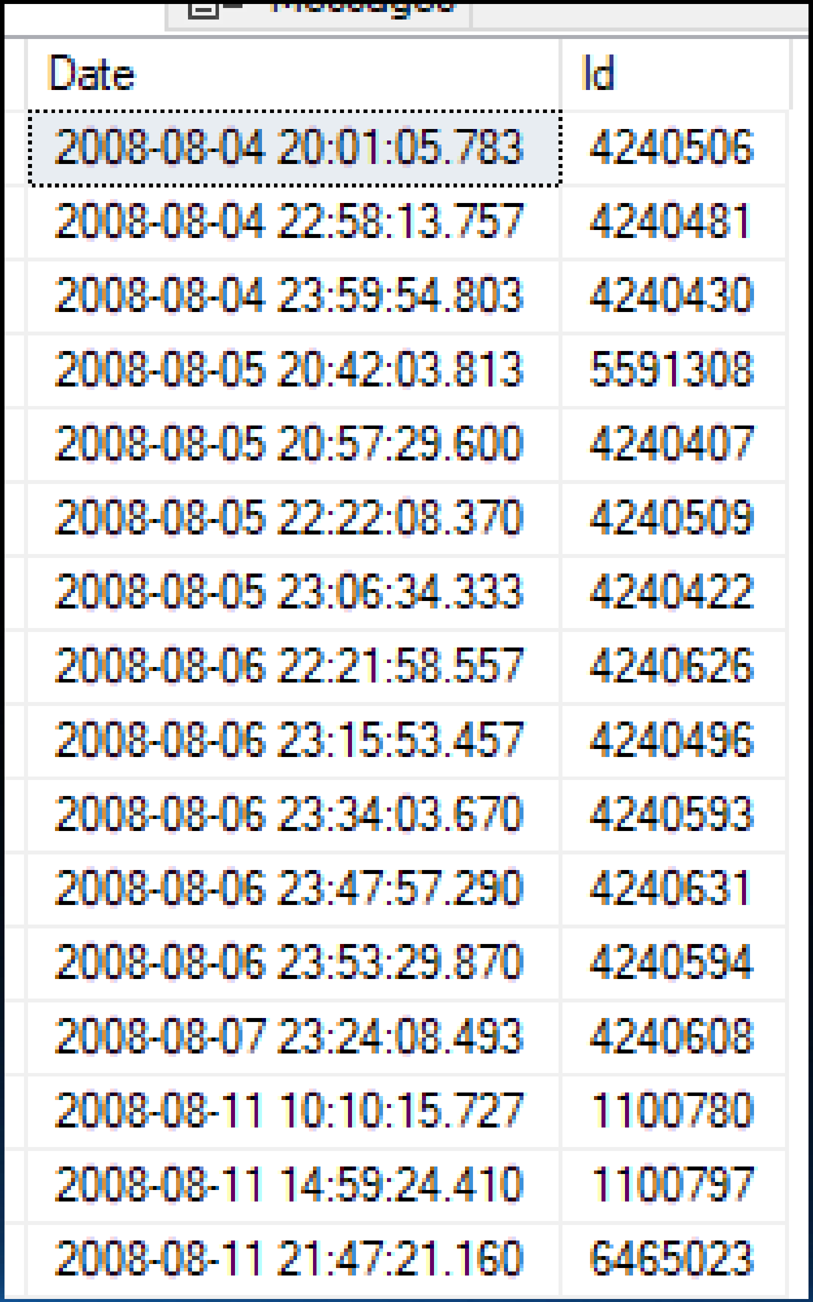
Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	8042005
Actual Number of Rows for All Executions	658038
Actual Number of Batches	0
Estimated I/O Cost	13.2713
Estimated Operator Cost	22.1176 (81%)
Estimated CPU Cost	8.84636
Estimated Subtree Cost	22.1176
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	677675
Estimated Number of Rows Per Execution	677675
Estimated Number of Rows to be Read	8042000
Estimated Row Size	15 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3

Predicate

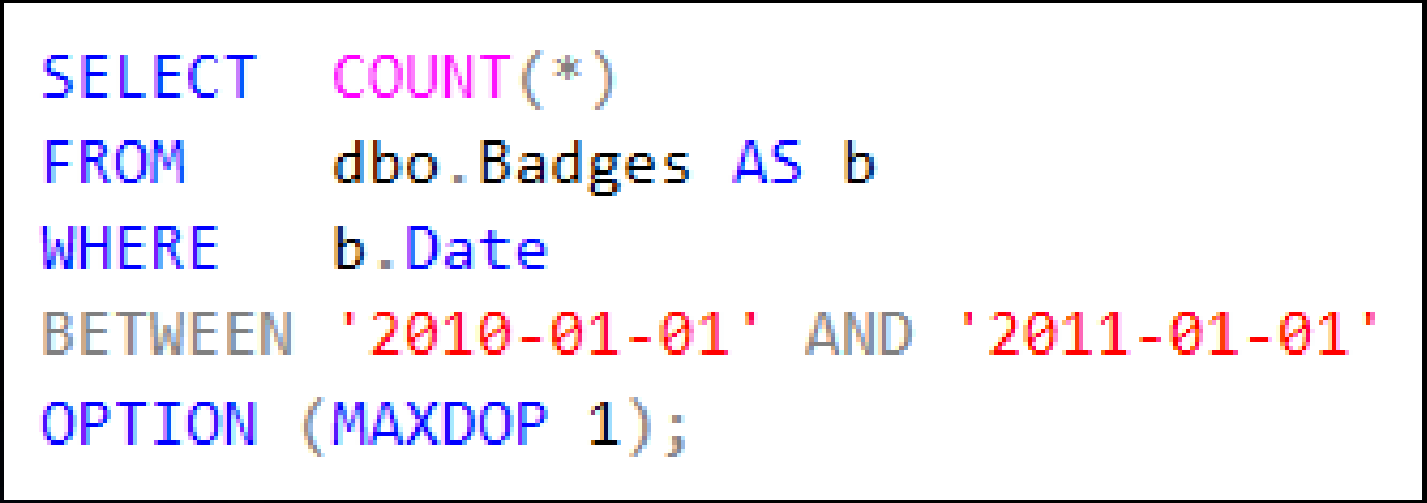
datepart(year,[StackOverflow2013].[dbo].[Badges].[Date] as [b].[Date])=(2010)

Object

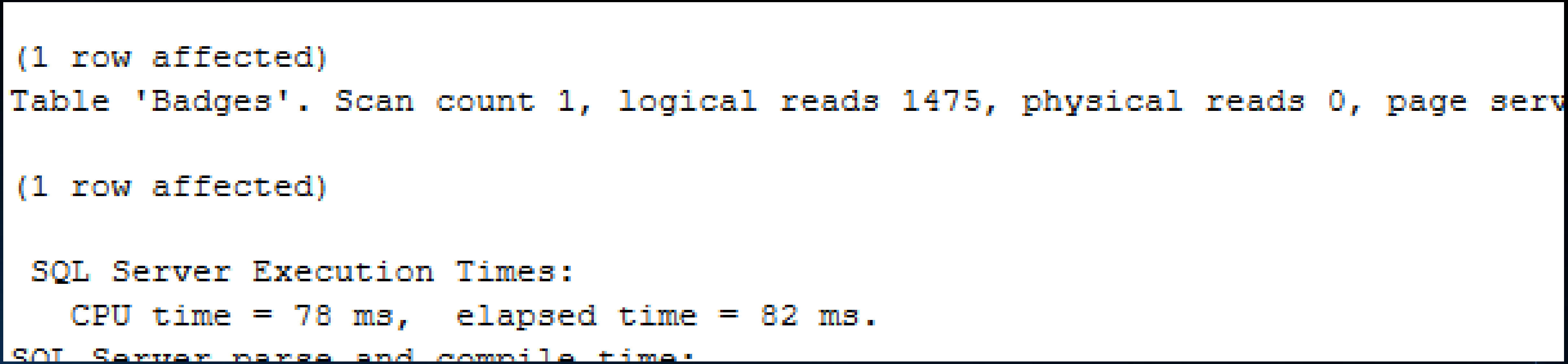
[StackOverflow2013].[dbo].[Badges].[ix_date] [b]



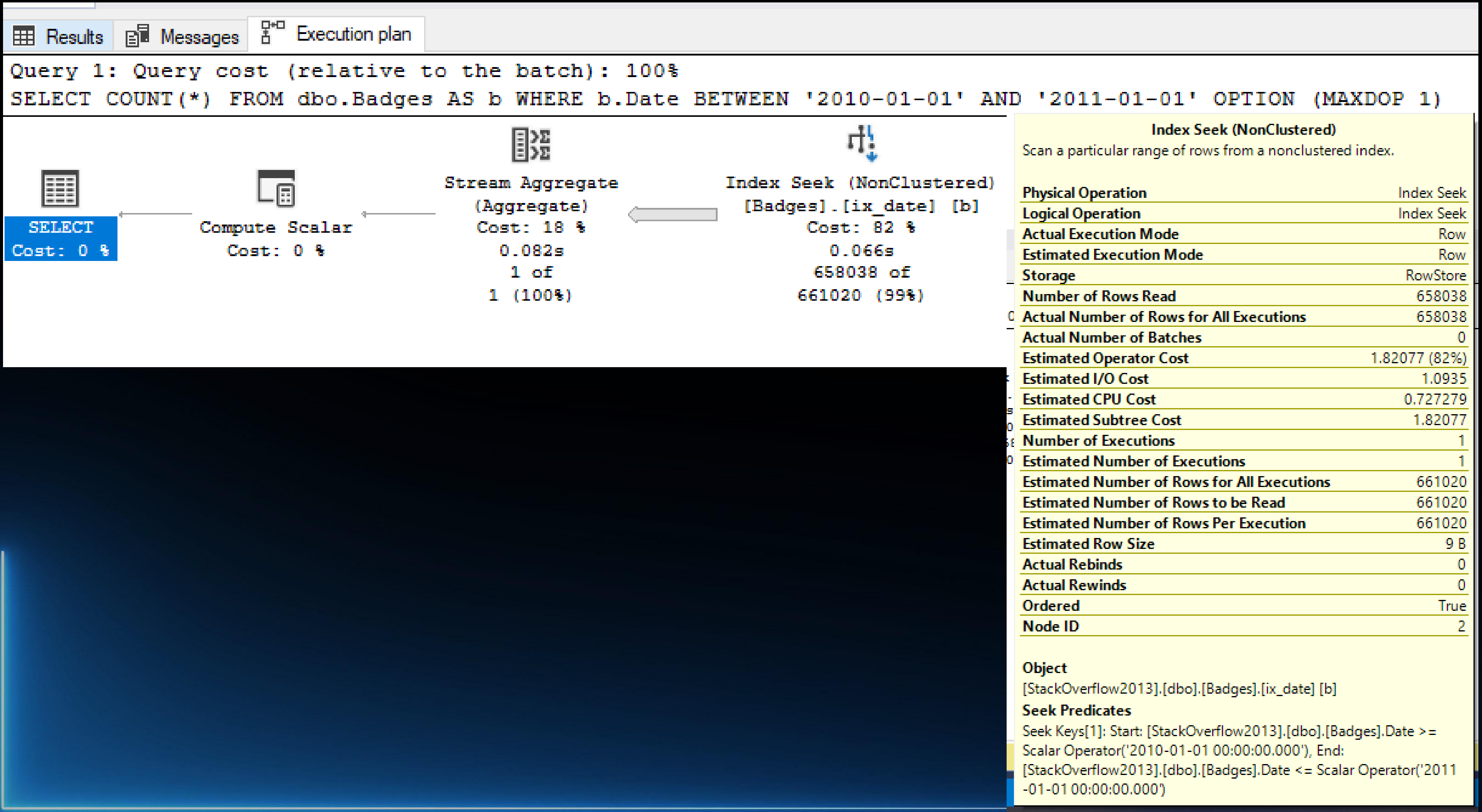
Date	Id
2008-08-04 20:01:05.783	4240506
2008-08-04 22:58:13.757	4240481
2008-08-04 23:59:54.803	4240430
2008-08-05 20:42:03.813	5591308
2008-08-05 20:57:29.600	4240407
2008-08-05 22:22:08.370	4240509
2008-08-05 23:06:34.333	4240422
2008-08-06 22:21:58.557	4240626
2008-08-06 23:15:53.457	4240496
2008-08-06 23:34:03.670	4240593
2008-08-06 23:47:57.290	4240631
2008-08-06 23:53:29.870	4240594
2008-08-07 23:24:08.493	4240608
2008-08-11 10:10:15.727	1100780
2008-08-11 14:59:24.410	1100797
2008-08-11 21:47:21.160	6465023



```sql
SELECT  COUNT(*)
FROM    dbo.Badges AS b
WHERE   b.Date
BETWEEN '2010-01-01' AND '2011-01-01'
OPTION (MAXDOP 1);
```



```
(1 row affected)
Table 'Badges'. Scan count 1, logical reads 1475, physical reads 0, page serv

(1 row affected)

 SQL Server Execution Times:
   CPU time = 78 ms,  elapsed time = 82 ms.
SQL Server parse and compile time:
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT(*) FROM dbo.Badges AS b WHERE b.Date BETWEEN '2010-01-01' AND '2011-01-01' OPTION (MAXDOP 1)
```

```
SELECT        Compute Scalar    Stream Aggregate    Index Seek (NonClustered)
Cost: 0 %     Cost: 0 %         (Aggregate)         [Badges].[ix_date] [b]
                                Cost: 18 %          Cost: 82 %
                                0.082s              0.066s
                                1 of                658038 of
                                1 (100%)            661020 (99%)
```

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

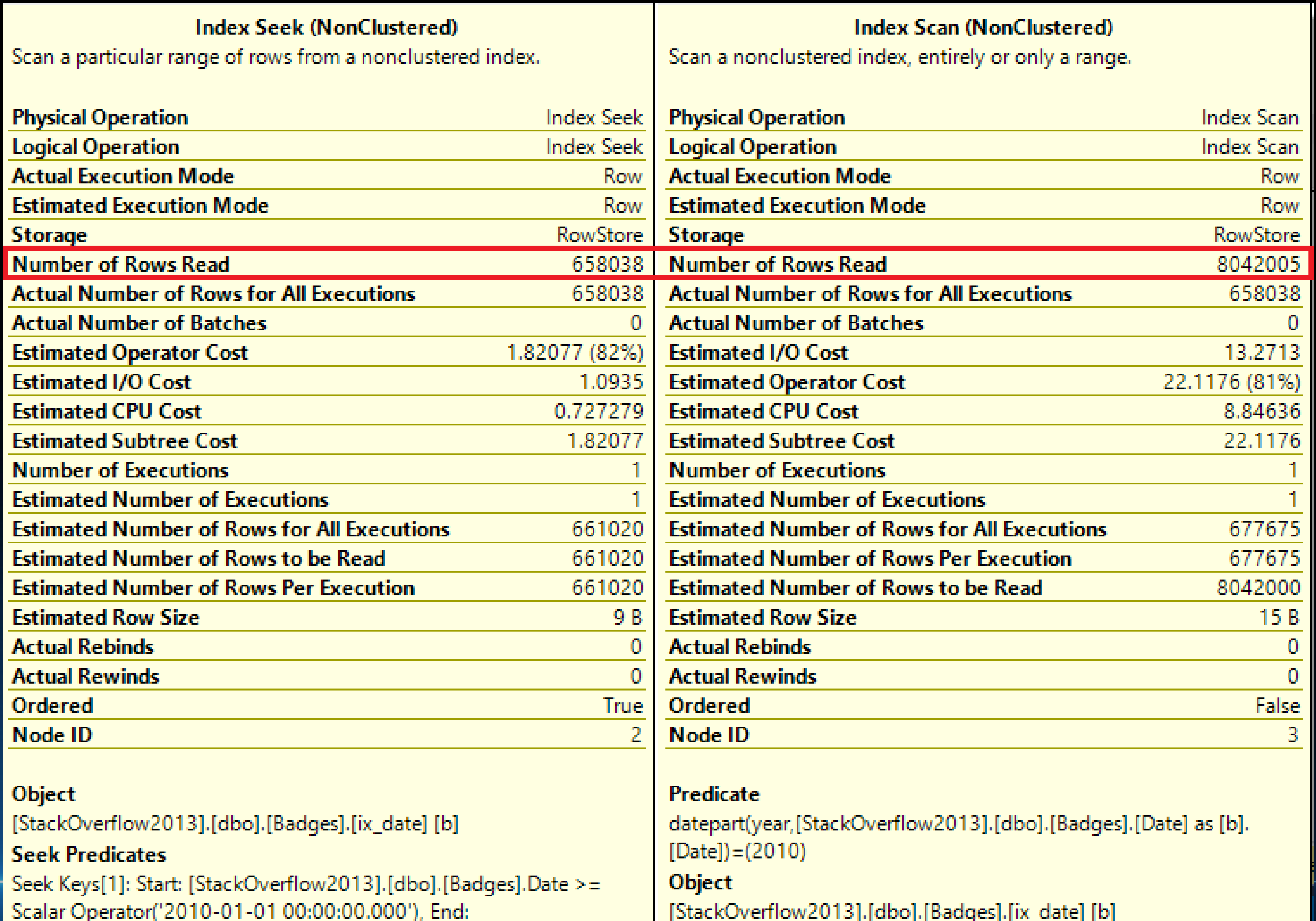
Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	658038
Actual Number of Rows for All Executions	658038
Actual Number of Batches	0
Estimated Operator Cost	1.82077 (82%)
Estimated I/O Cost	1.0935
Estimated CPU Cost	0.727279
Estimated Subtree Cost	1.82077
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	661020
Estimated Number of Rows to be Read	661020
Estimated Number of Rows Per Execution	661020
Estimated Row Size	9 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Object

[StackOverflow2013].[dbo].[Badges].[ix_date] [b]

Seek Predicates

Seek Keys[1]: Start: [StackOverflow2013].[dbo].[Badges].Date >= Scalar Operator('2010-01-01 00:00:00.000'), End: [StackOverflow2013].[dbo].[Badges].Date <= Scalar Operator('2011-01-01 00:00:00.000')



Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	658038
Actual Number of Rows for All Executions	658038
Actual Number of Batches	0
Estimated Operator Cost	1.82077 (82%)
Estimated I/O Cost	1.0935
Estimated CPU Cost	0.727279
Estimated Subtree Cost	1.82077
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	661020
Estimated Number of Rows to be Read	661020
Estimated Number of Rows Per Execution	661020
Estimated Row Size	9 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Object

[StackOverflow2013].[dbo].[Badges].[ix_date] [b]

Seek Predicates

Seek Keys[1]: Start: [StackOverflow2013].[dbo].[Badges].Date >= Scalar Operator('2010-01-01 00:00:00.000'), End:

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	8042005
Actual Number of Rows for All Executions	658038
Actual Number of Batches	0
Estimated I/O Cost	13.2713
Estimated Operator Cost	22.1176 (81%)
Estimated CPU Cost	8.84636
Estimated Subtree Cost	22.1176
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	677675
Estimated Number of Rows Per Execution	677675
Estimated Number of Rows to be Read	8042000
Estimated Row Size	15 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3

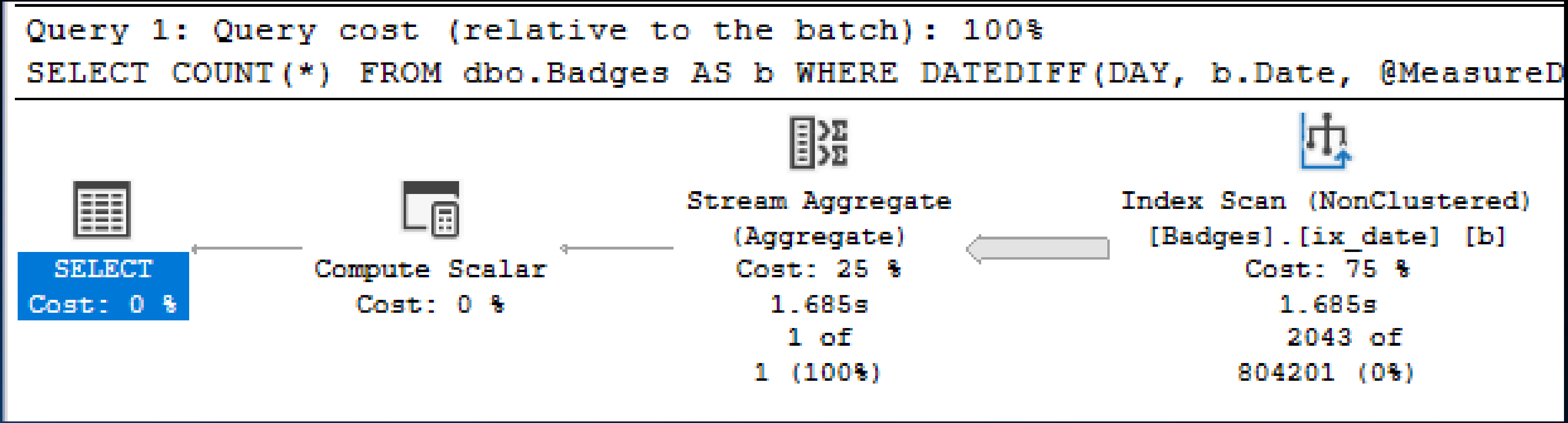
Predicate

datepart(year,[StackOverflow2013].[dbo].[Badges].[Date] as [b].[Date])=(2010)

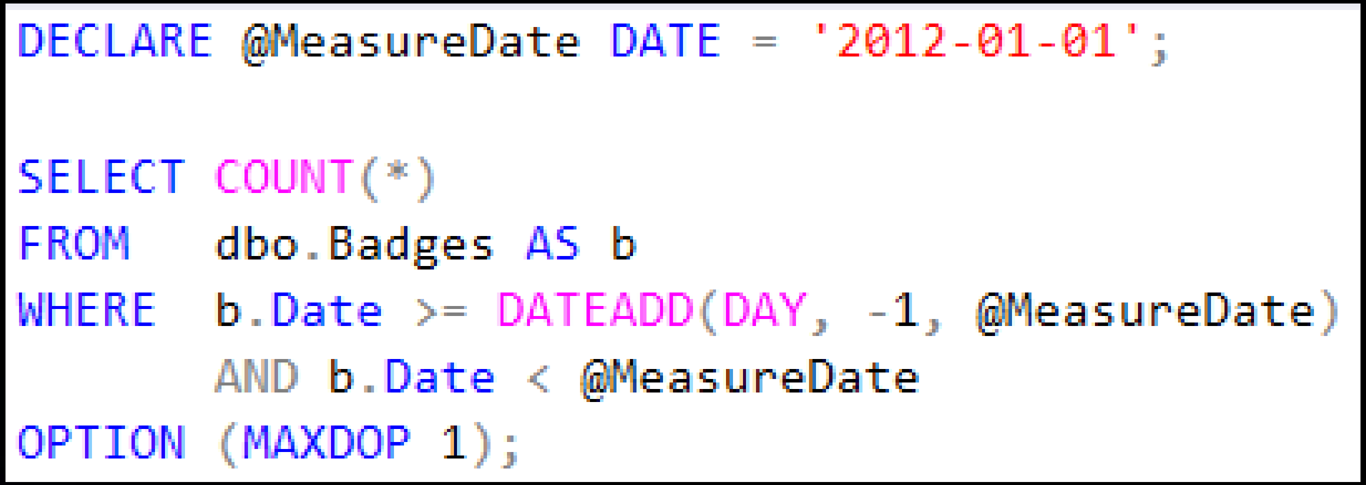
Object

[StackOverflow2013].[dbo].[Badges].[ix_date] [b]

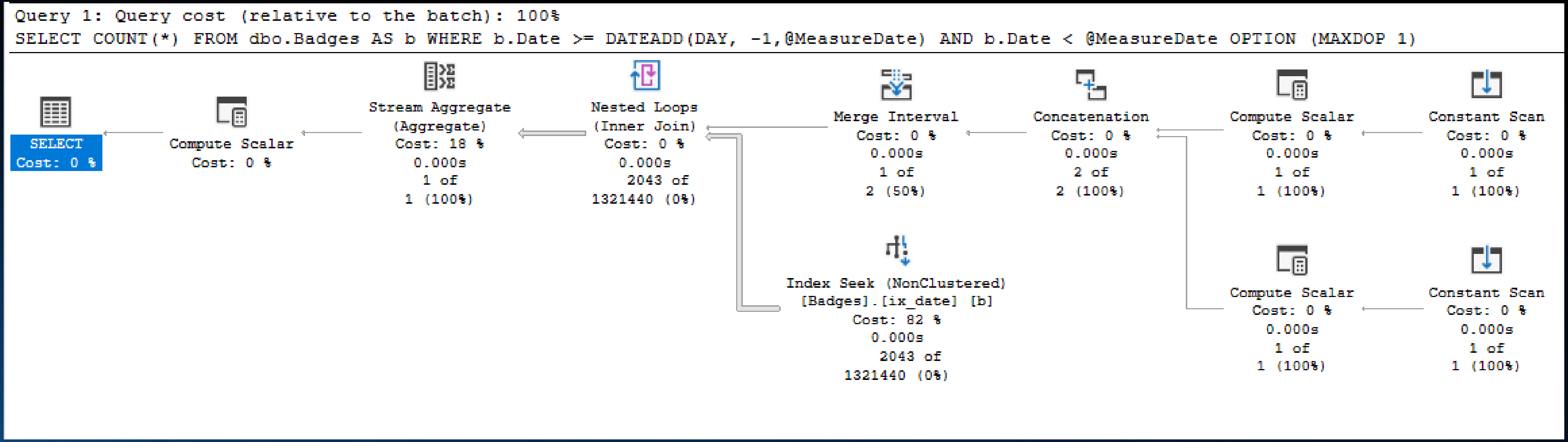
```sql
DECLARE @MeasureDate DATE = '2012-01-01';

SELECT  COUNT(*)
FROM    dbo.Badges AS b
WHERE   DATEDIFF(DAY, b.Date, @MeasureDate) = 1
OPTION (MAXDOP 1);
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT(*) FROM dbo.Badges AS b WHERE DATEDIFF(DAY, b.Date, @MeasureD
```

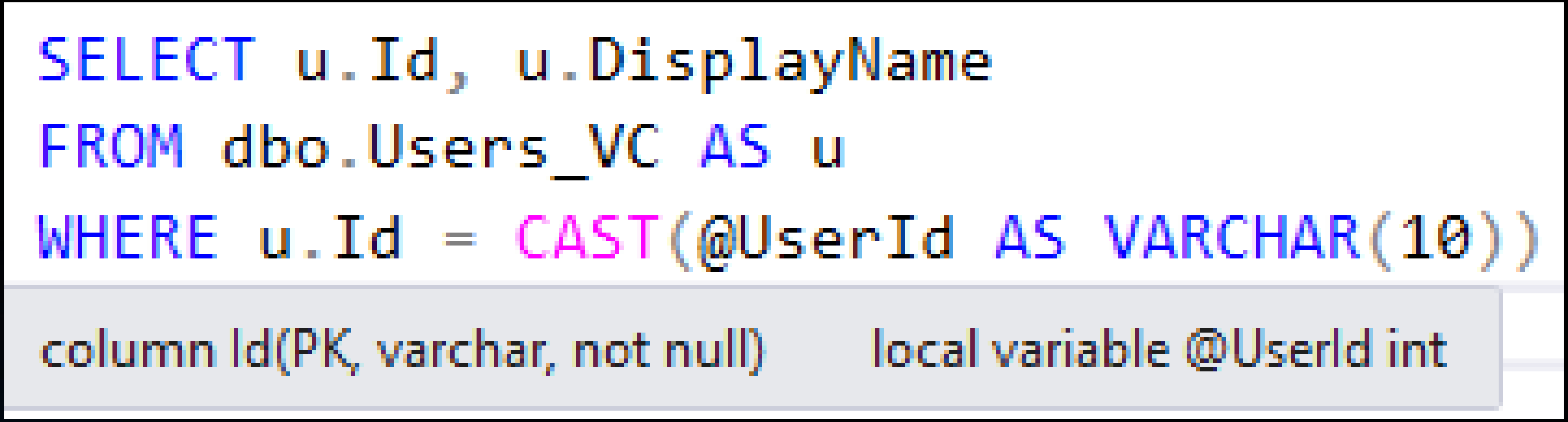


```sql
DECLARE @MeasureDate DATE = '2012-01-01';

SELECT COUNT(*)
FROM   dbo.Badges AS b
WHERE  b.Date >= DATEADD(DAY, -1, @MeasureDate)
       AND b.Date < @MeasureDate
OPTION (MAXDOP 1);
```



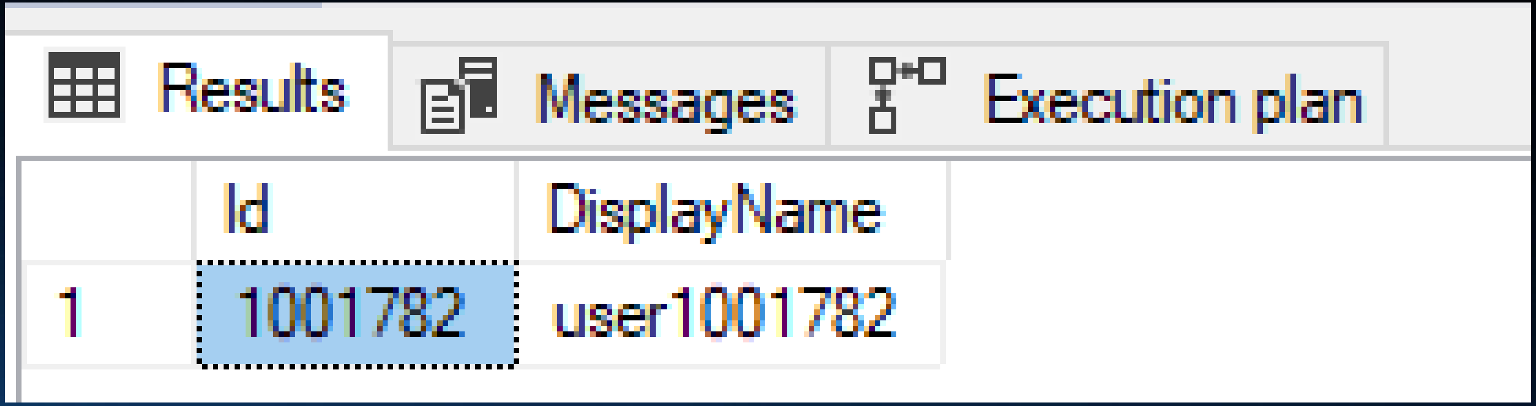
```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT(*) FROM dbo.Badges AS b WHERE b.Date >= DATEADD(DAY, -1,@MeasureDate) AND b.Date < @MeasureDate OPTION (MAXDOP 1)
```

Implicit casting

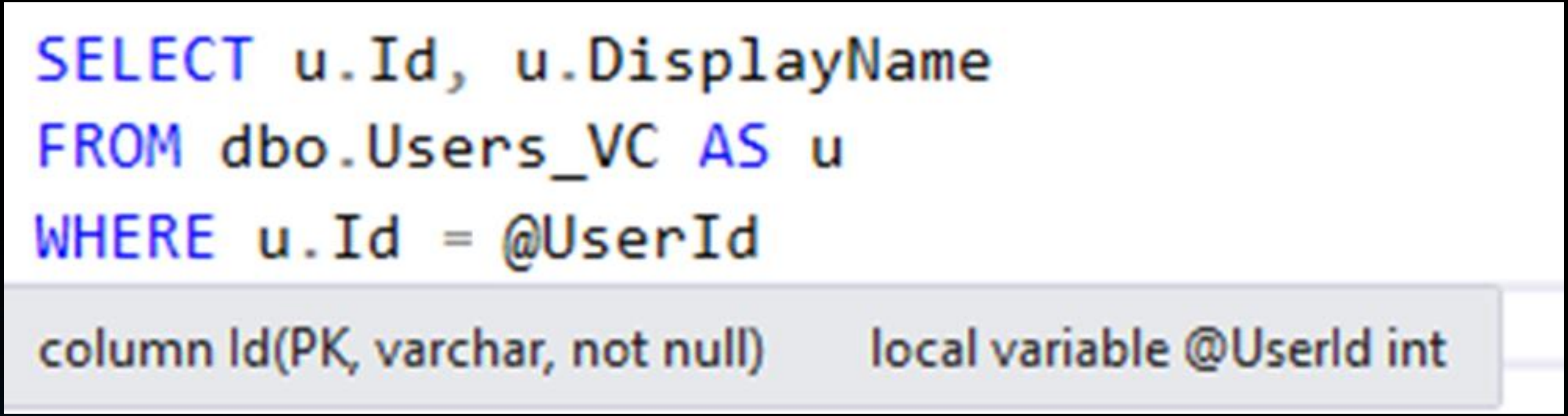


```sql
SELECT u.Id, u.DisplayName
FROM dbo.Users_VC AS u
WHERE u.Id = CAST(@UserId AS VARCHAR(10))
```

column Id(PK, varchar, not null)   local variable @UserId int

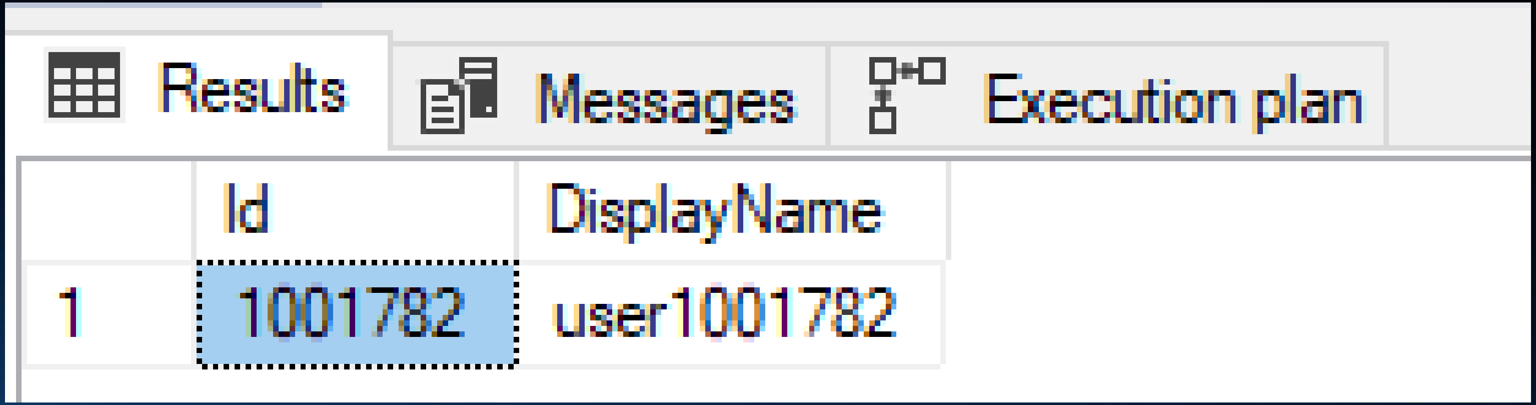


	Id	DisplayName
1	1001782	user1001782

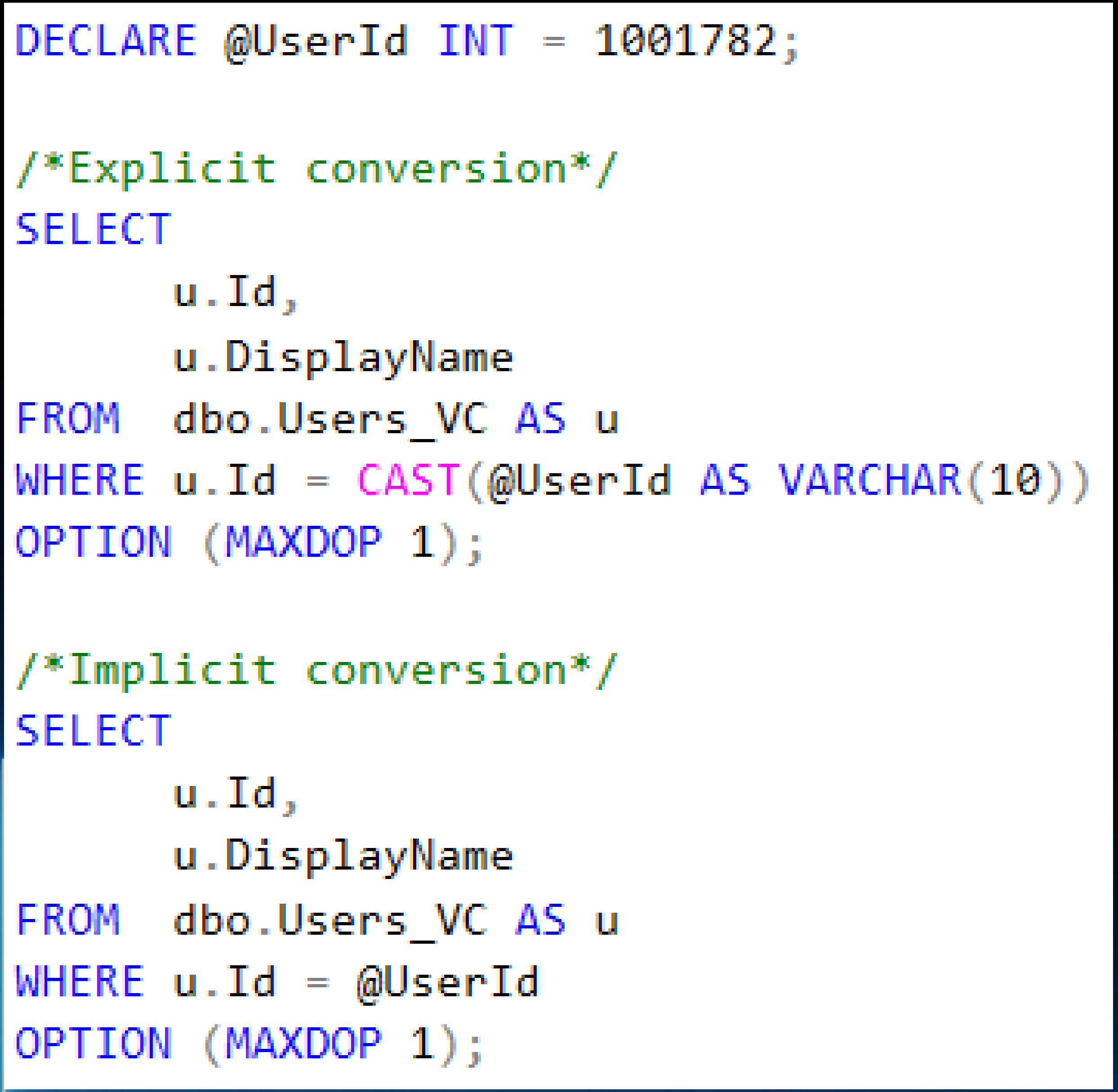


```sql
SELECT u.Id, u.DisplayName
FROM dbo.Users_VC AS u
WHERE u.Id = @UserId
```

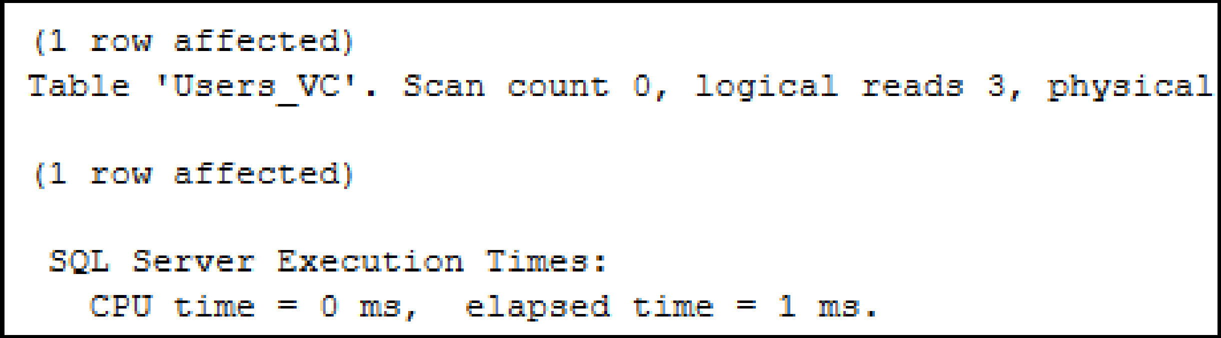
```
column Id(PK, varchar, not null)      local variable @UserId int
```



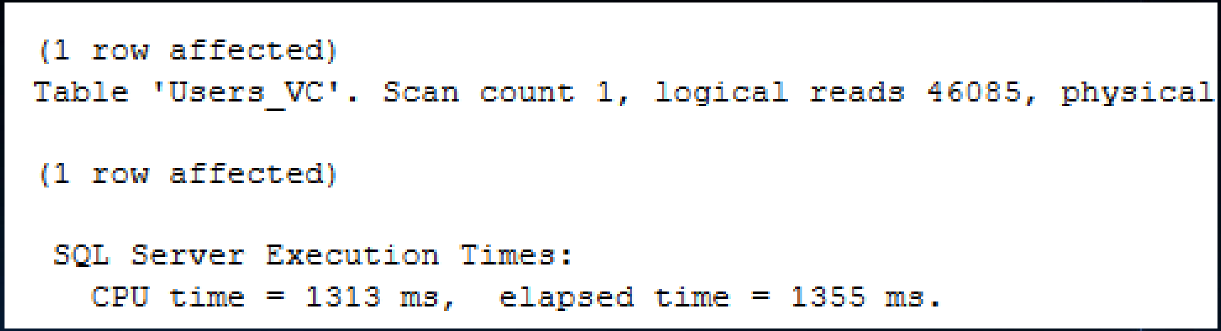
	Id	DisplayName
1	1001782	user1001782



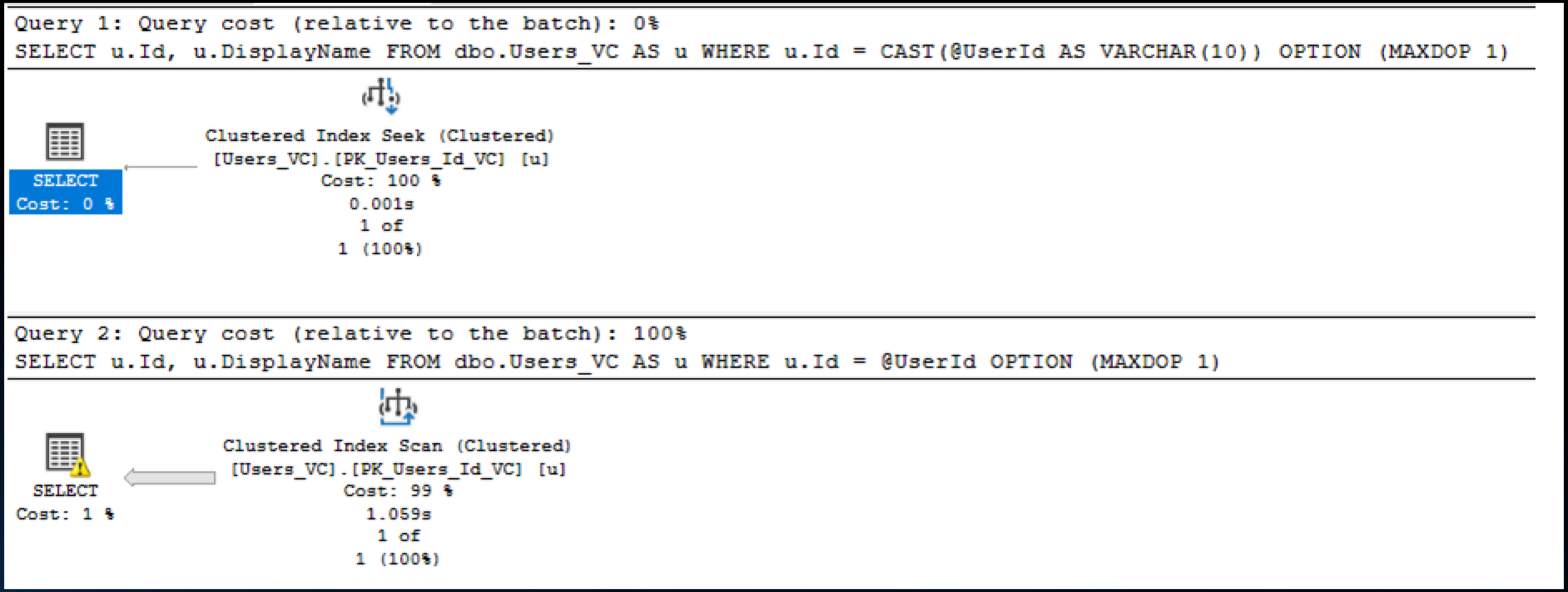
```sql
DECLARE @UserId INT = 1001782;

/*Explicit conversion*/
SELECT
       u.Id,
       u.DisplayName
FROM   dbo.Users_VC AS u
WHERE u.Id = CAST(@UserId AS VARCHAR(10))
OPTION (MAXDOP 1);

/*Implicit conversion*/
SELECT
       u.Id,
       u.DisplayName
FROM   dbo.Users_VC AS u
WHERE u.Id = @UserId
OPTION (MAXDOP 1);
```



```
(1 row affected)
Table 'Users_VC'. Scan count 0, logical reads 3, physical

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
```



```
(1 row affected)
Table 'Users_VC'. Scan count 1, logical reads 46085, physical

(1 row affected)

 SQL Server Execution Times:
   CPU time = 1313 ms,  elapsed time = 1355 ms.
```

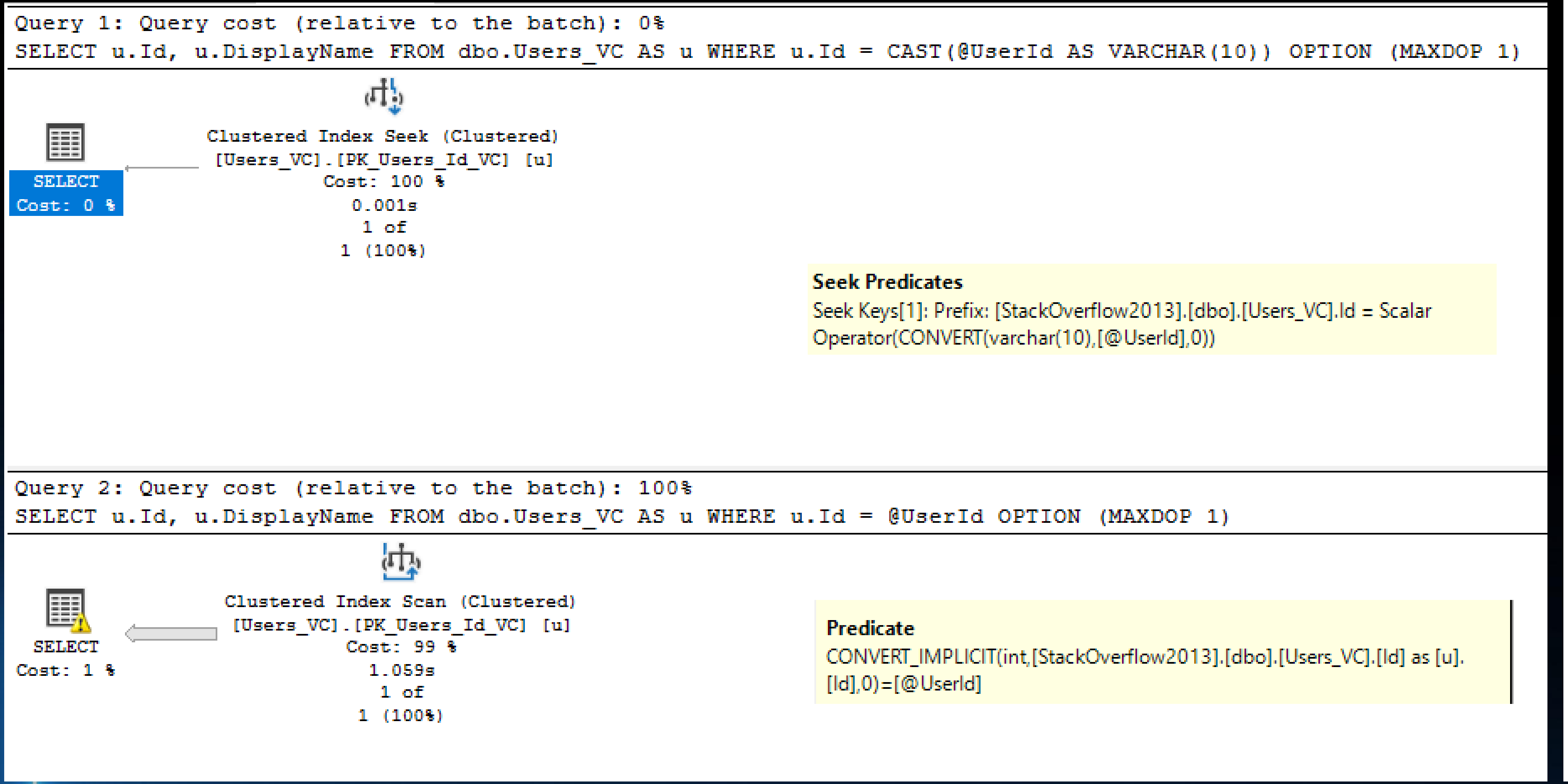


```
Query 1: Query cost (relative to the batch): 0%
SELECT u.Id, u.DisplayName FROM dbo.Users_VC AS u WHERE u.Id = CAST(@UserId AS VARCHAR(10)) OPTION (MAXDOP 1)
```

SELECT
Cost: 0 %
Clustered Index Seek (Clustered)
[Users_VC].[PK_Users_Id_VC] [u]
Cost: 100 %
0.001s
1 of
1 (100%)

```
Query 2: Query cost (relative to the batch): 100%
SELECT u.Id, u.DisplayName FROM dbo.Users_VC AS u WHERE u.Id = @UserId OPTION (MAXDOP 1)
```

SELECT
Cost: 1 %
Clustered Index Scan (Clustered)
[Users_VC].[PK_Users_Id_VC] [u]
Cost: 99 %
1.059s
1 of
1 (100%)



```
Query 1: Query cost (relative to the batch): 0%
SELECT u.Id, u.DisplayName FROM dbo.Users_VC AS u WHERE u.Id = CAST(@UserId AS VARCHAR(10)) OPTION (MAXDOP 1)
```

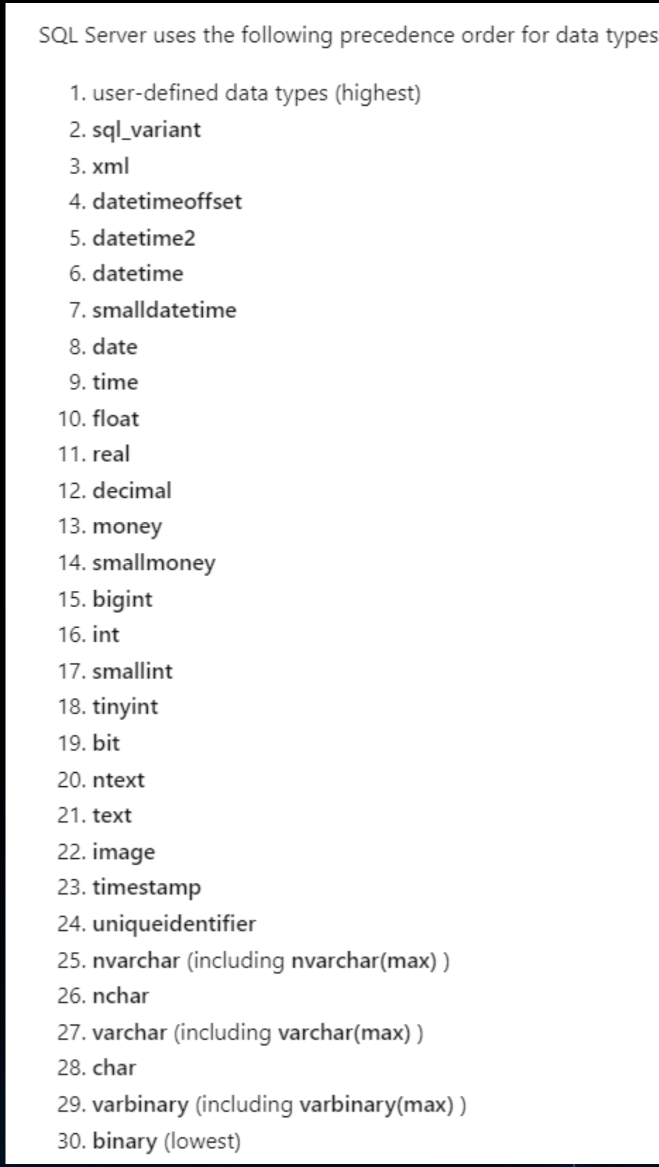
```
SELECT        Clustered Index Seek (Clustered)
Cost: 0 %     [Users_VC].[PK_Users_Id_VC] [u]
                        Cost: 100 %
                          0.001s
                           1 of
                         1 (100%)
```

Seek Predicates
Seek Keys[1]: Prefix: [StackOverflow2013].[dbo].[Users_VC].Id = Scalar Operator(CONVERT(varchar(10),[@UserId],0))

```
Query 2: Query cost (relative to the batch): 100%
SELECT u.Id, u.DisplayName FROM dbo.Users_VC AS u WHERE u.Id = @UserId OPTION (MAXDOP 1)
```

```
SELECT        Clustered Index Scan (Clustered)
Cost: 1 %     [Users_VC].[PK_Users_Id_VC] [u]
                        Cost: 99 %
                          1.059s
                           1 of
                         1 (100%)
```

Predicate
CONVERT_IMPLICIT(int,[StackOverflow2013].[dbo].[Users_VC].[Id] as [u].[Id],0)=[@UserId]



SQL Server uses the following precedence order for data types

1. user-defined data types (highest)
2. sql_variant
3. xml
4. datetimeoffset
5. datetime2
6. datetime
7. smalldatetime
8. date
9. time
10. float
11. real
12. decimal
13. money
14. smallmoney
15. bigint
16. int
17. smallint
18. tinyint
19. bit
20. ntext
21. text
22. image
23. timestamp
24. uniqueidentifier
25. nvarchar (including nvarchar(max))
26. nchar
27. varchar (including varchar(max))
28. char
29. varbinary (including varbinary(max))
30. binary (lowest)

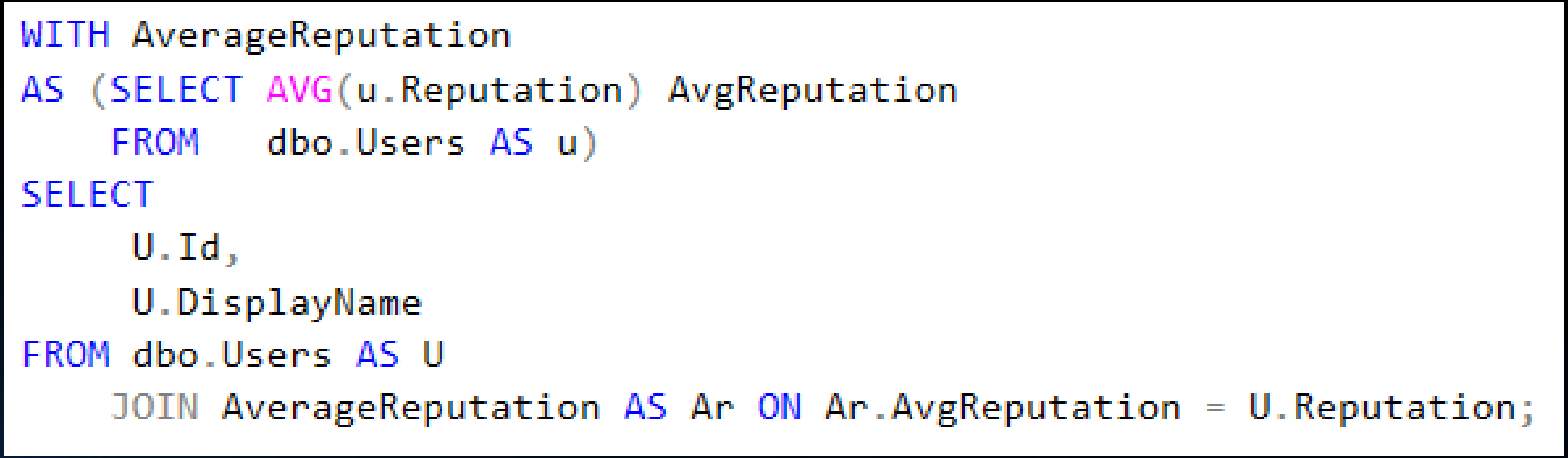
https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-type-precedence-transact-sql?view=sql-server-ver16

How to fix?

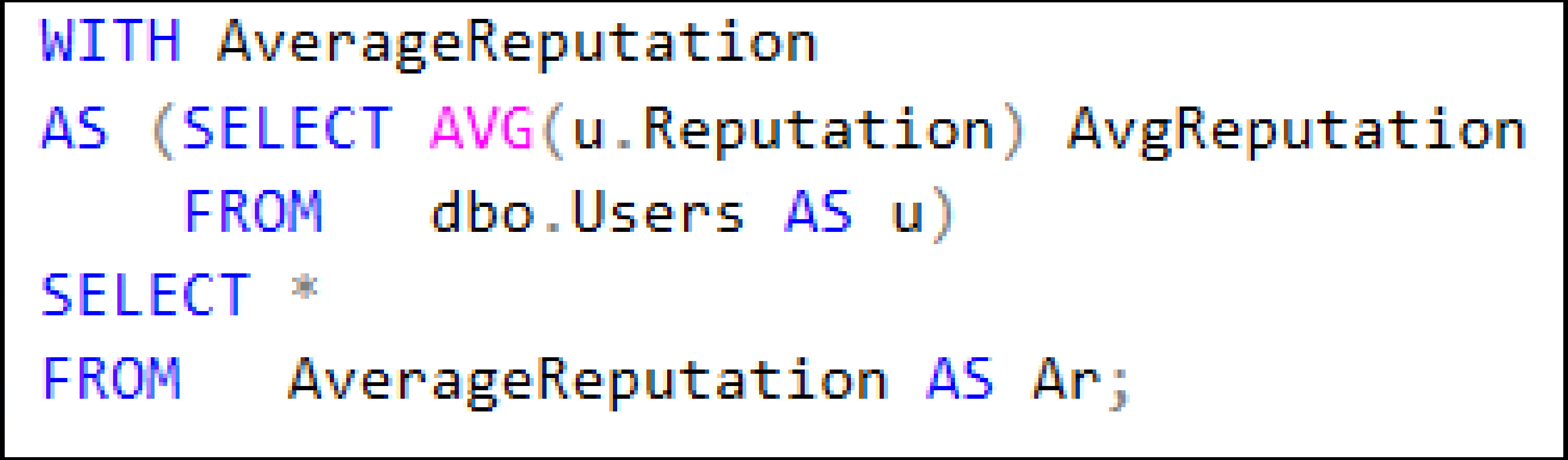
- Make use of Explicit casting
  - On the ‘small size’
- Fix your database types
  - Make sure, that for the same type of data, the same data-types are used

Multiple CTE references

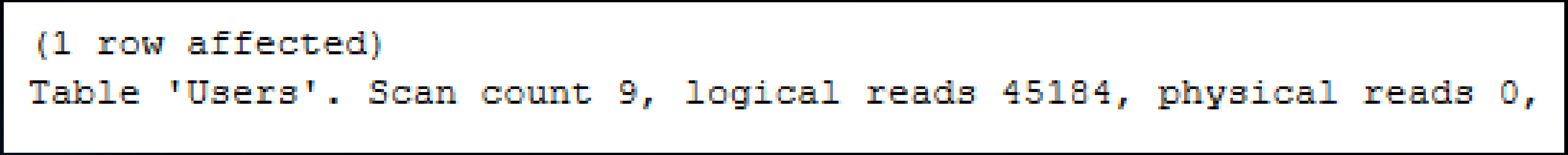
Examples



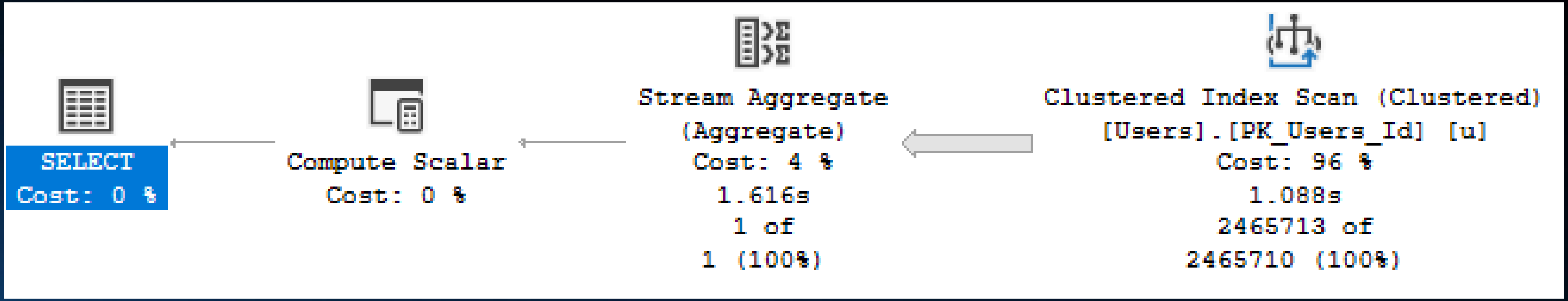
```
WITH AverageReputation
AS (SELECT AVG(u.Reputation) AvgReputation
    FROM   dbo.Users AS u)
SELECT
     U.Id,
     U.DisplayName
FROM dbo.Users AS U
    JOIN AverageReputation AS Ar ON Ar.AvgReputation = U.Reputation;
```

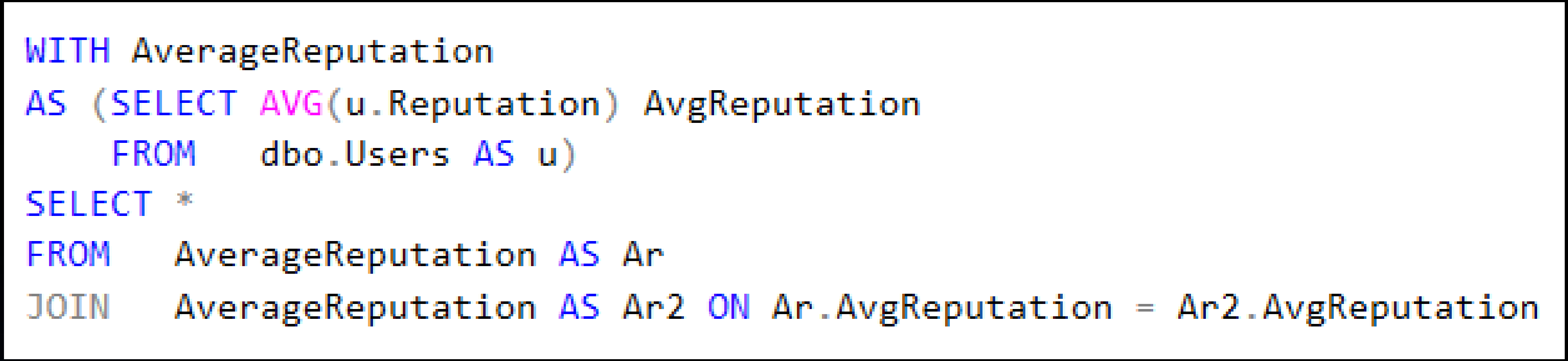


```sql
WITH AverageReputation
AS (SELECT AVG(u.Reputation) AvgReputation
    FROM   dbo.Users AS u)
SELECT *
FROM   AverageReputation AS Ar;
```

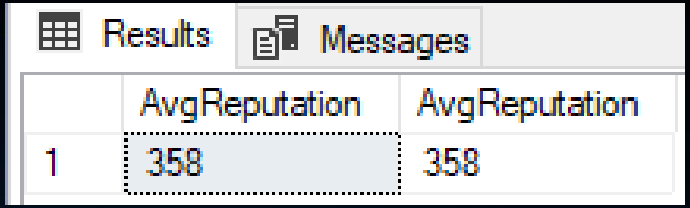


```
(1 row affected)
Table 'Users'. Scan count 9, logical reads 45184, physical reads 0,
```

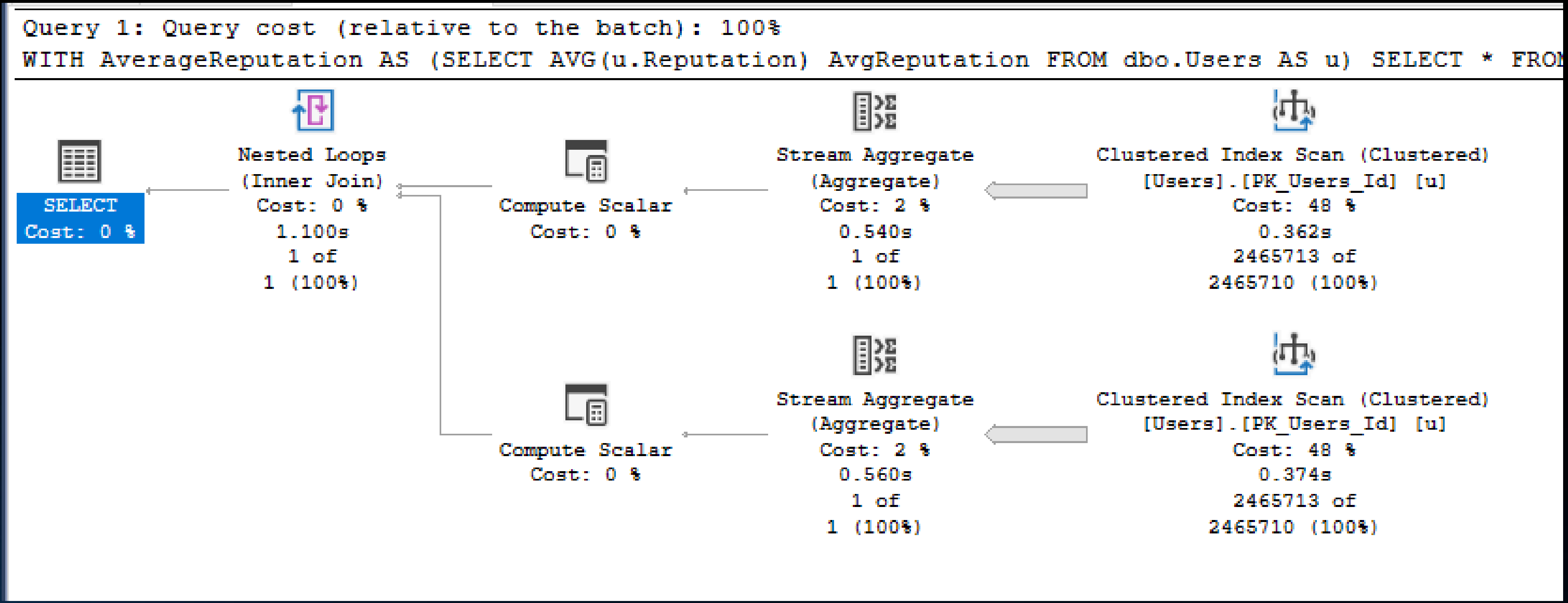


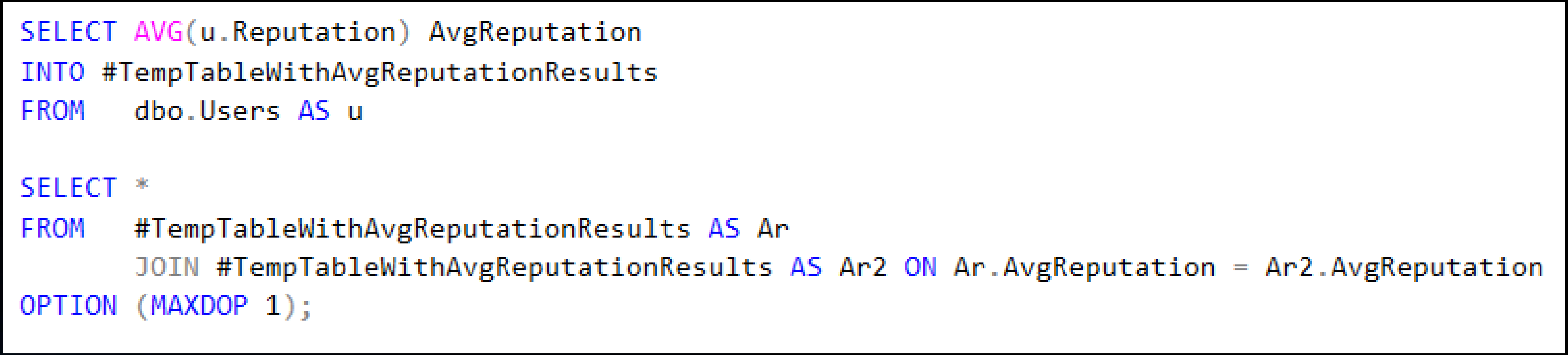


```sql
WITH AverageReputation
AS (SELECT AVG(u.Reputation) AvgReputation
    FROM   dbo.Users AS u)
SELECT *
FROM   AverageReputation AS Ar
JOIN   AverageReputation AS Ar2 ON Ar.AvgReputation = Ar2.AvgReputation
```

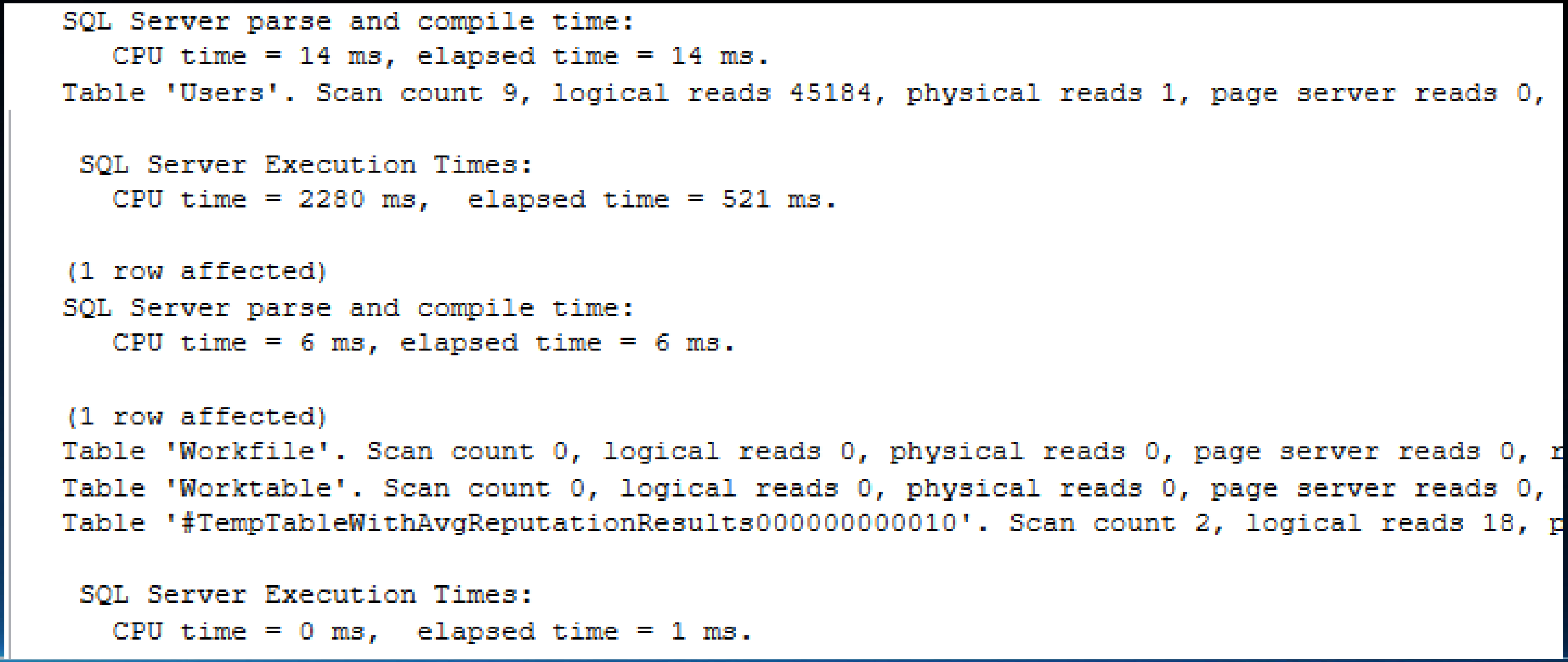


	AvgReputation	AvgReputation
1	358	358


Query 1: Query cost (relative to the batch): 100%
WITH AverageReputation AS (SELECT AVG(u.Reputation) AvgReputation FROM dbo.Users AS u) SELECT * FRO

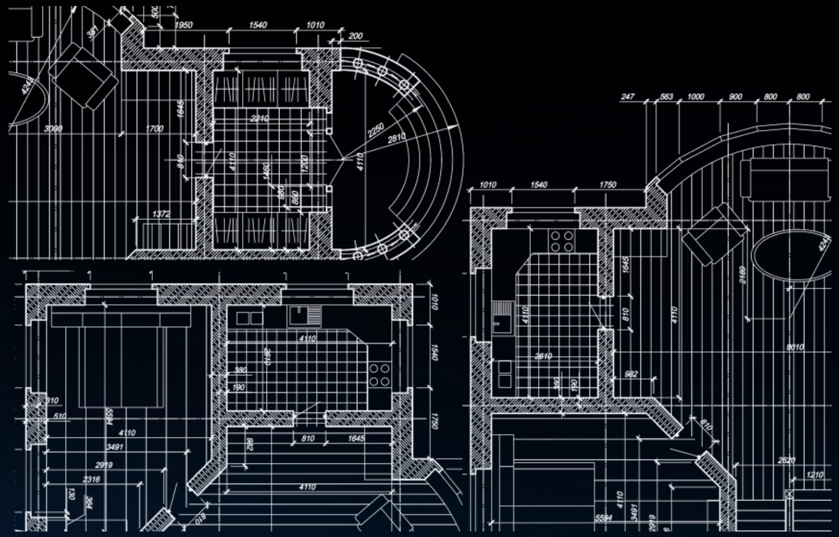


```sql
SELECT AVG(u.Reputation) AvgReputation
INTO #TempTableWithAvgReputationResults
FROM   dbo.Users AS u

SELECT *
FROM   #TempTableWithAvgReputationResults AS Ar
       JOIN #TempTableWithAvgReputationResults AS Ar2 ON Ar.AvgReputation = Ar2.AvgReputation
OPTION (MAXDOP 1);
```



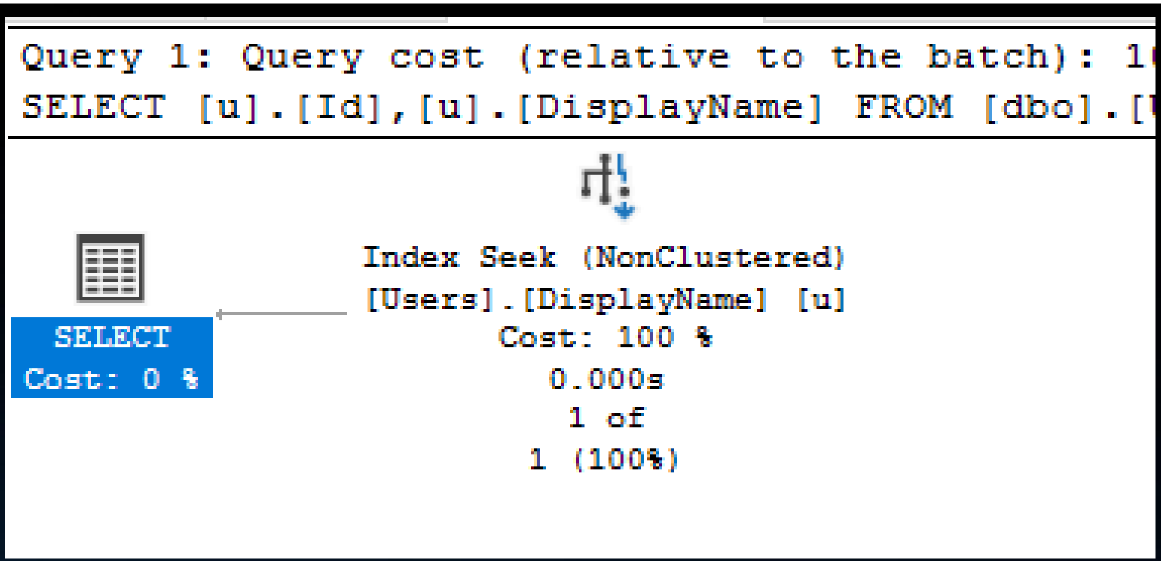
```
SQL Server parse and compile time:
   CPU time = 14 ms, elapsed time = 14 ms.
Table 'Users'. Scan count 9, logical reads 45184, physical reads 1, page server reads 0,

 SQL Server Execution Times:
   CPU time = 2280 ms,  elapsed time = 521 ms.

(1 row affected)
SQL Server parse and compile time:
   CPU time = 6 ms, elapsed time = 6 ms.

(1 row affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, r
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0,
Table '#TempTableWithAvgReputationResults000000000010'. Scan count 2, logical reads 18, p

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
```

SELECT *



```sql
CREATE NONCLUSTERED INDEX [DisplayName] ON [dbo].[Users] ([DisplayName] ASC)
```

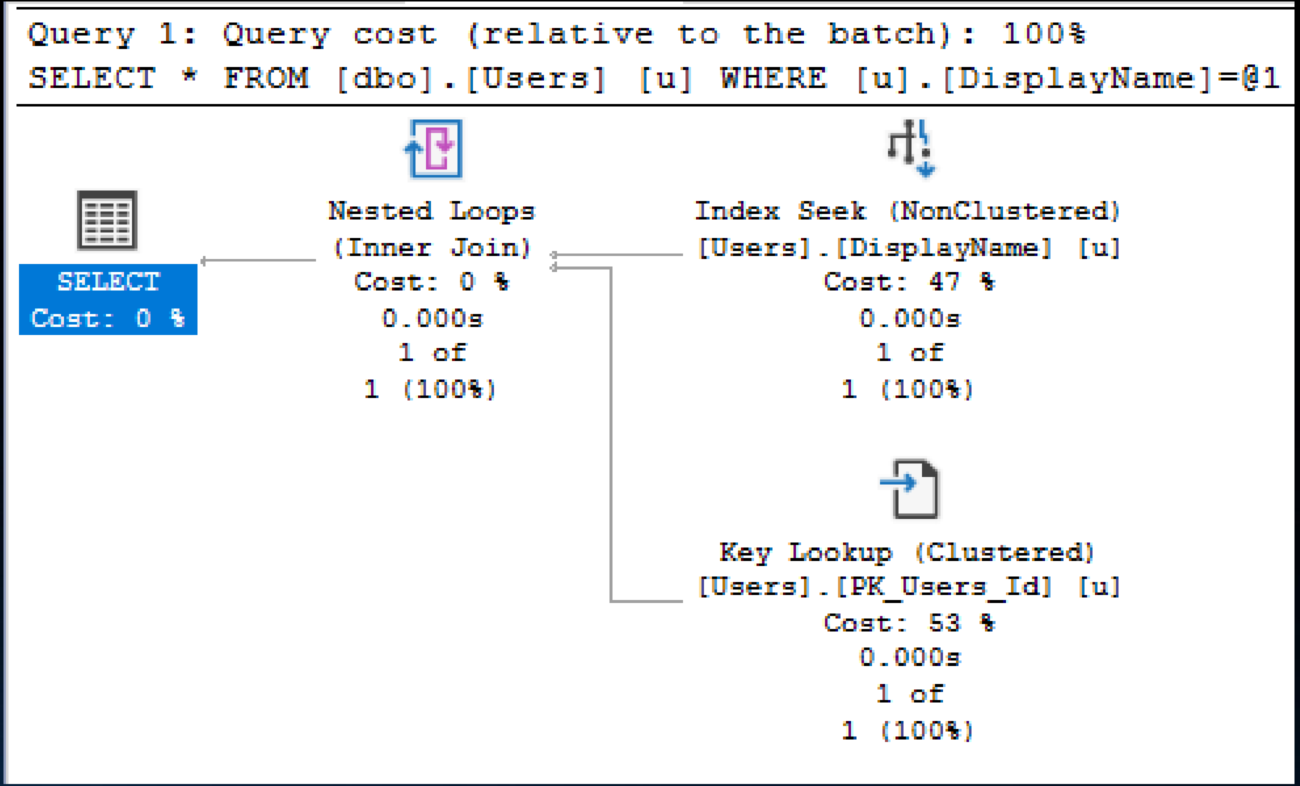
```sql
SELECT u.Id, u.DisplayName
FROM dbo.Users AS u
WHERE u.DisplayName = 'Peter Radocchia'
```



```
Query 1: Query cost (relative to the batch): 1
SELECT [u].[Id],[u].[DisplayName] FROM [dbo].[
```

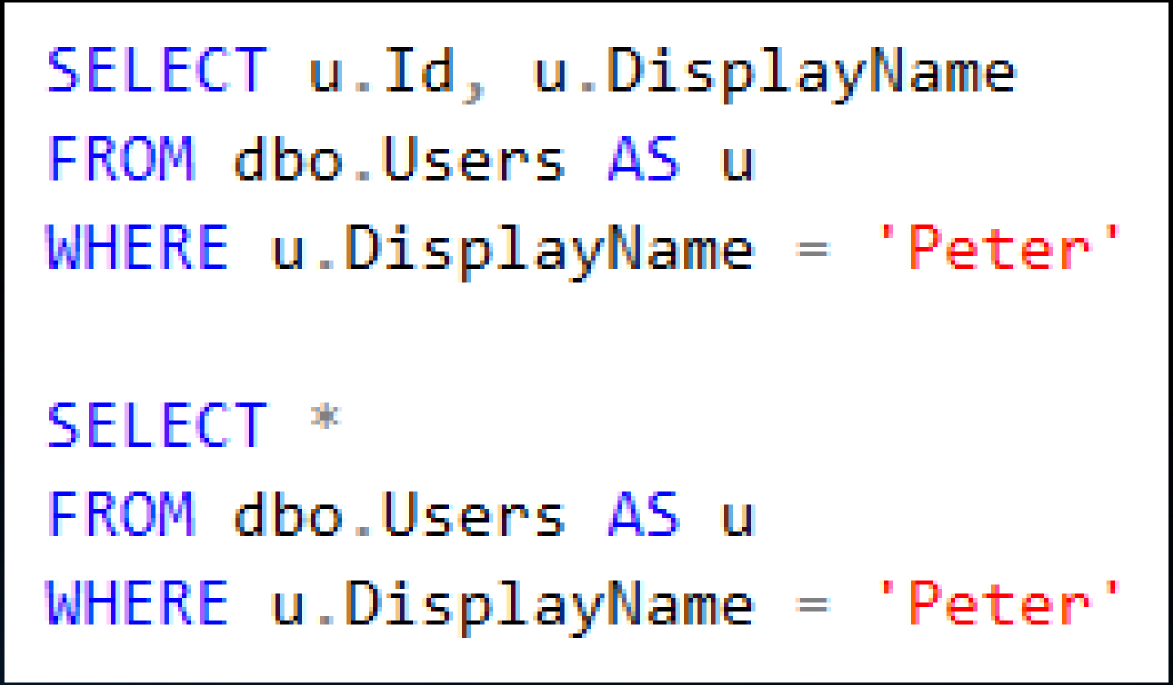
```
(1 row affected)
Table 'Users'. Scan count 1, logical reads 3, physi

(1 row affected)
```

```sql
SELECT *
FROM dbo.Users AS u
WHERE u.DisplayName = 'Peter Radocchia'
```

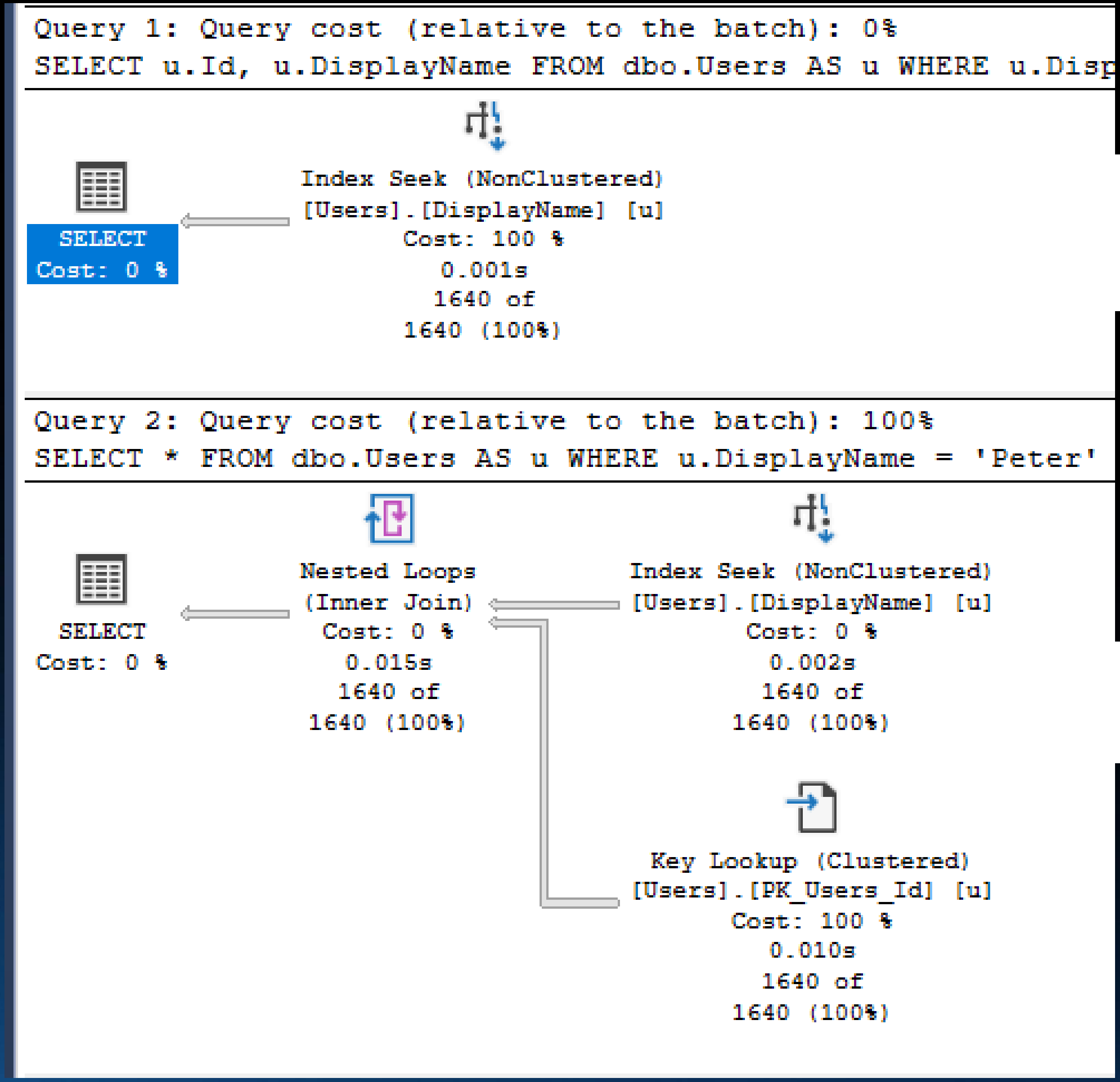


```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Users] [u] WHERE [u].[DisplayName]=@1
```

```
(1 row affected)
Table 'Users'. Scan count 1, logical reads 6,
```

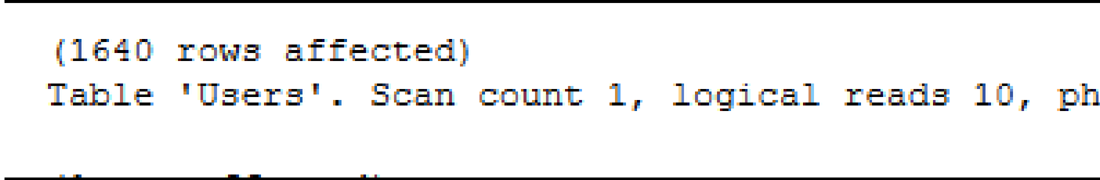


```sql
SELECT u.Id, u.DisplayName
FROM dbo.Users AS u
WHERE u.DisplayName = 'Peter'

SELECT *
FROM dbo.Users AS u
WHERE u.DisplayName = 'Peter'
```

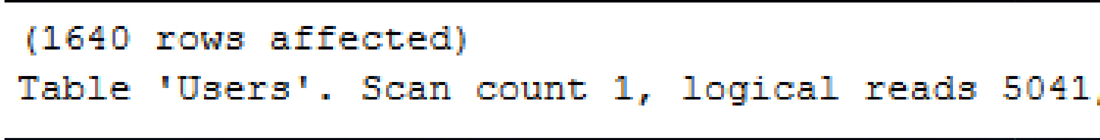


```
Query 1: Query cost (relative to the batch): 0%
SELECT u.Id, u.DisplayName FROM dbo.Users AS u WHERE u.Disp
```

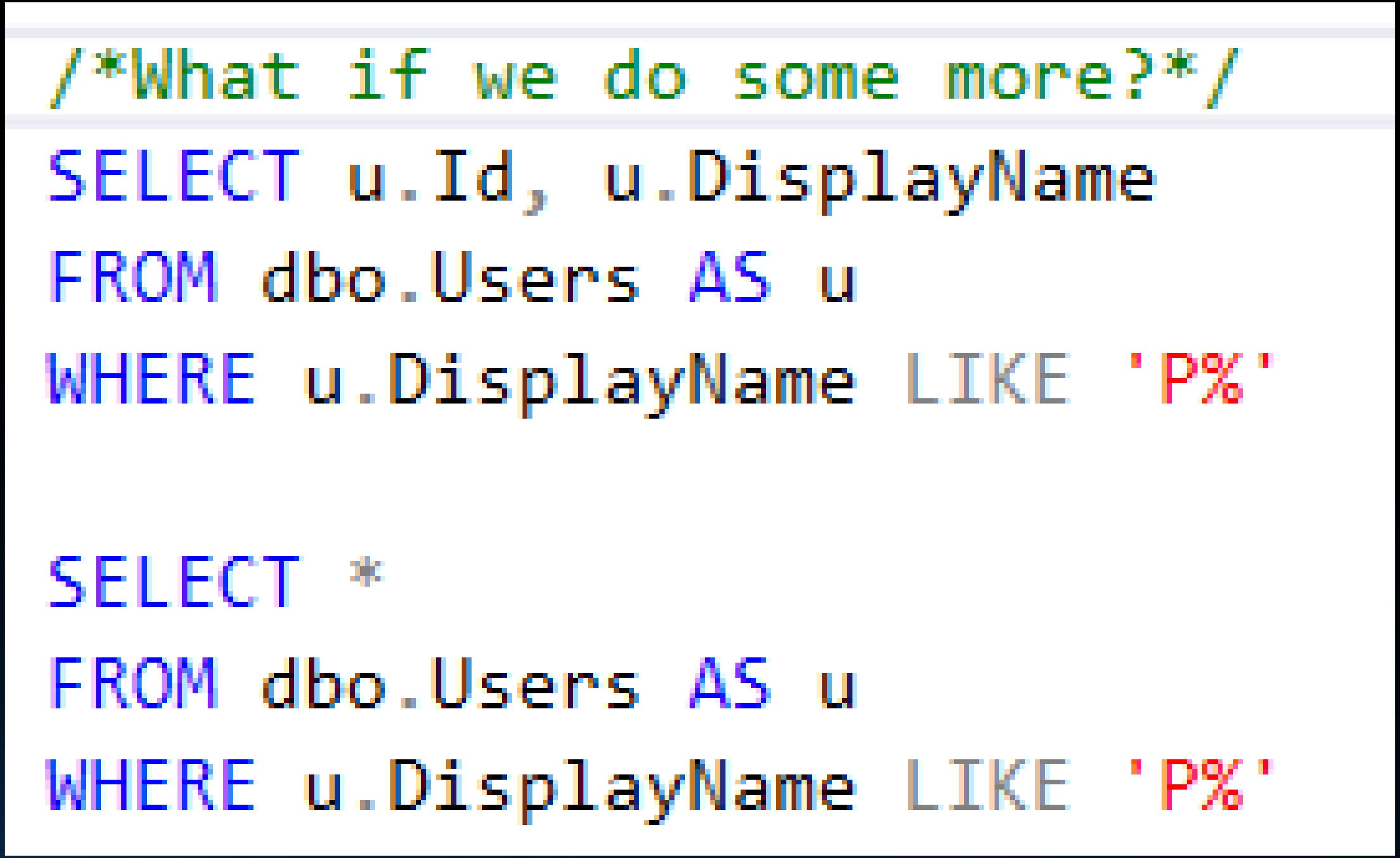
```
Query 2: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users AS u WHERE u.DisplayName = 'Peter'
```



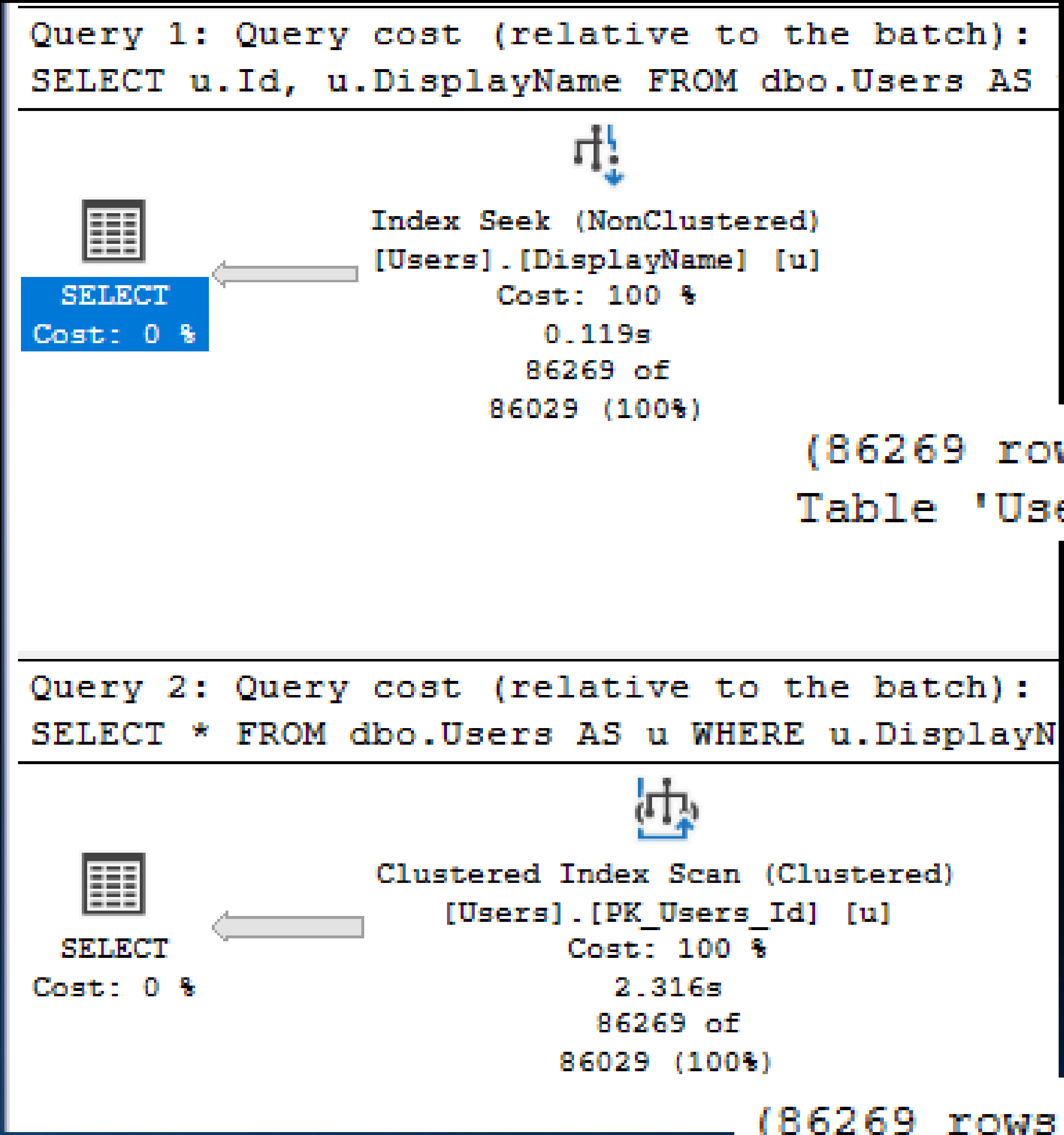
```
(1640 rows affected)
Table 'Users'. Scan count 1, logical reads 10, ph
```



```
(1640 rows affected)
Table 'Users'. Scan count 1, logical reads 5041,
```



```sql
/*What if we do some more?*/
SELECT u.Id, u.DisplayName
FROM dbo.Users AS u
WHERE u.DisplayName LIKE 'P%'

SELECT *
FROM dbo.Users AS u
WHERE u.DisplayName LIKE 'P%'
```



```
Query 1: Query cost (relative to the batch):
SELECT u.Id, u.DisplayName FROM dbo.Users AS
```

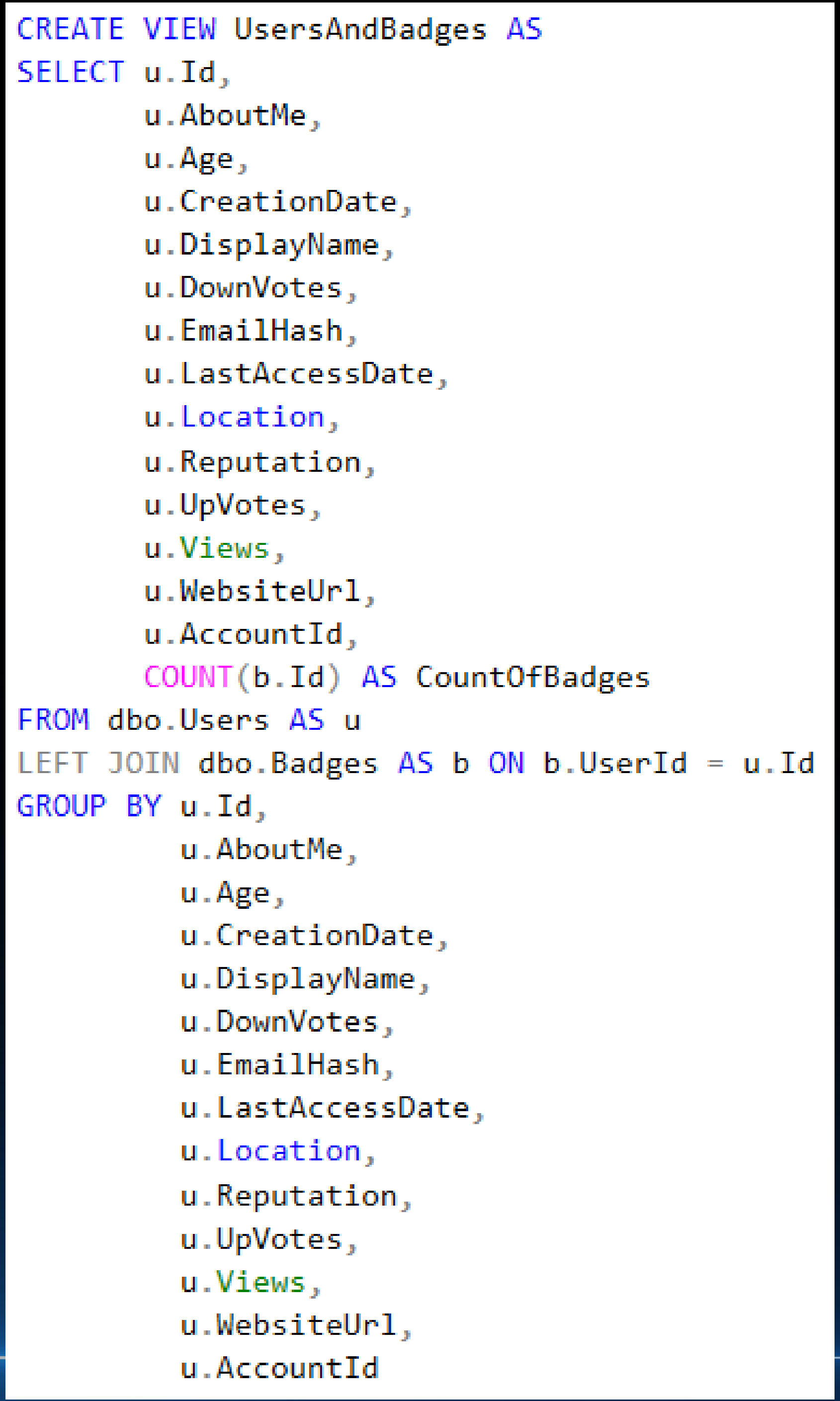
Index Seek (NonClustered)
[Users].[DisplayName] [u]
Cost: 100 %
0.119s
86269 of
86029 (100%)
SELECT
Cost: 0 %

```
(86269 rows affected)
Table 'Users'. Scan count 1, logical reads 355,
```

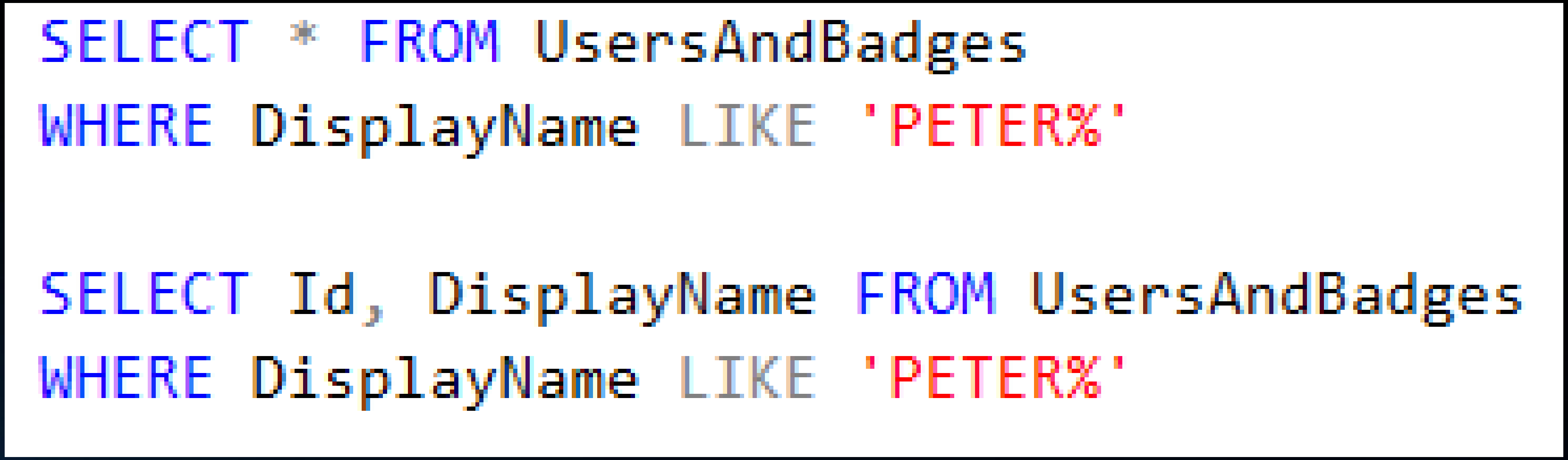
```
Query 2: Query cost (relative to the batch):
SELECT * FROM dbo.Users AS u WHERE u.DisplayN
```

Clustered Index Scan (Clustered)
[Users].[PK_Users_Id] [u]
Cost: 100 %
2.316s
86269 of
86029 (100%)
SELECT
Cost: 0 %

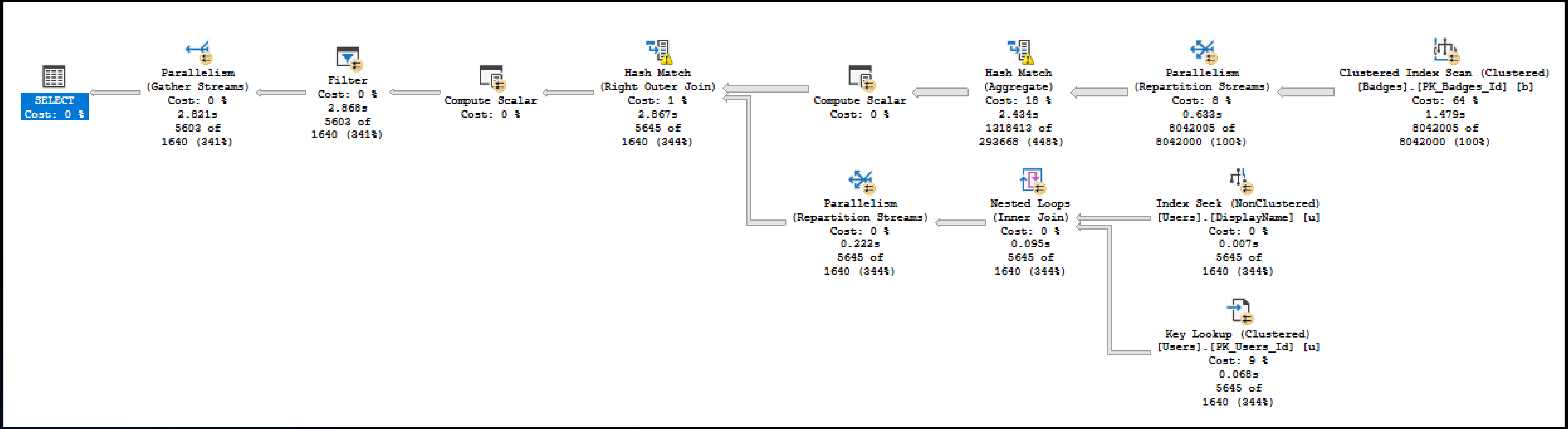
```
(86269 rows affected)
Table 'Users'. Scan count 1, logical reads 44530,
```

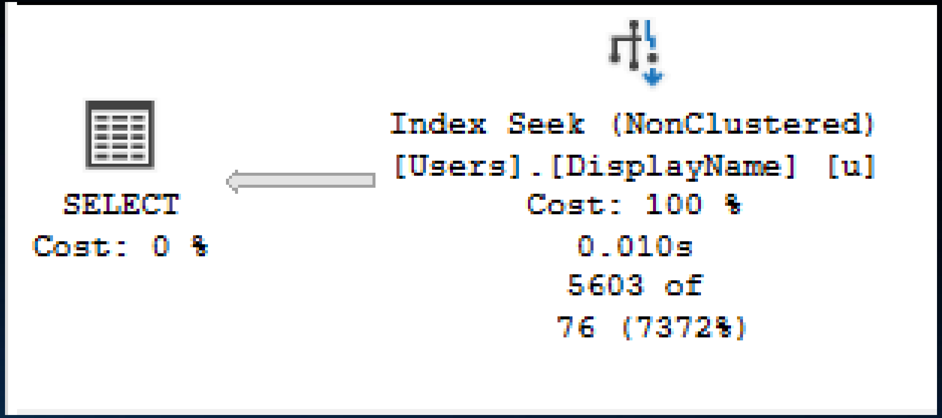


```sql
CREATE VIEW UsersAndBadges AS
SELECT u.Id,
       u.AboutMe,
       u.Age,
       u.CreationDate,
       u.DisplayName,
       u.DownVotes,
       u.EmailHash,
       u.LastAccessDate,
       u.Location,
       u.Reputation,
       u.UpVotes,
       u.Views,
       u.WebsiteUrl,
       u.AccountId,
       COUNT(b.Id) AS CountOfBadges
FROM dbo.Users AS u
LEFT JOIN dbo.Badges AS b ON b.UserId = u.Id
GROUP BY u.Id,
         u.AboutMe,
         u.Age,
         u.CreationDate,
         u.DisplayName,
         u.DownVotes,
         u.EmailHash,
         u.LastAccessDate,
         u.Location,
         u.Reputation,
         u.UpVotes,
         u.Views,
         u.WebsiteUrl,
         u.AccountId
```

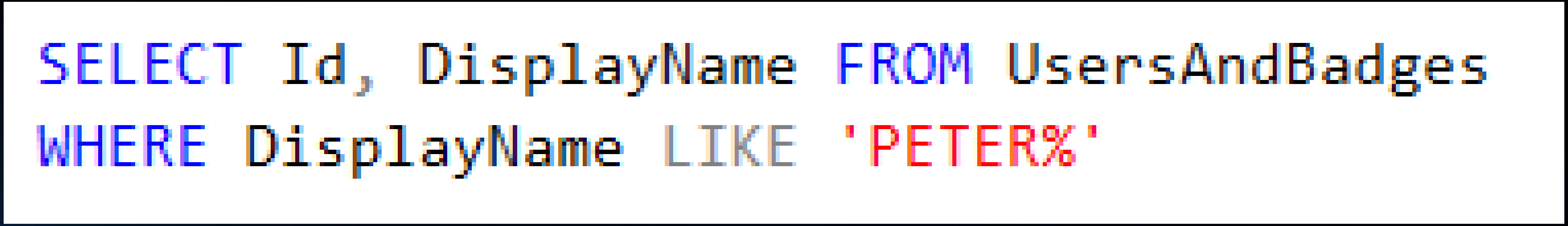


```sql
SELECT * FROM UsersAndBadges
WHERE DisplayName LIKE 'PETER%'

SELECT Id, DisplayName FROM UsersAndBadges
WHERE DisplayName LIKE 'PETER%'
```


SELECT
Cost: 0 %
Parallelism
(Gather Streams)
Cost: 0 %
2.821s
5603 of
1640 (341%)
Filter
Cost: 0 %
2.868s
5603 of
1640 (341%)
Compute Scalar
Cost: 0 %
Hash Match
(Right Outer Join)
Cost: 1 %
2.867s
5645 of
1640 (344%)
Compute Scalar
Cost: 0 %
Hash Match
(Aggregate)
Cost: 18 %
2.434s
1318413 of
293668 (448%)
Parallelism
(Repartition Streams)
Cost: 8 %
0.633s
8042005 of
8042000 (100%)
Clustered Index Scan (Clustered)
[Badges].[PK_Badges_Id] [b]
Cost: 64 %
1.479s
8042005 of
8042000 (100%)
Parallelism
(Repartition Streams)
Cost: 0 %
0.222s
5645 of
1640 (344%)
Nested Loops
(Inner Join)
Cost: 0 %
0.095s
5645 of
1640 (344%)
Index Seek (NonClustered)
[Users].[DisplayName] [u]
Cost: 0 %
0.007s
5645 of
1640 (344%)
Key Lookup (Clustered)
[Users].[PK_Users_Id] [u]
Cost: 9 %
0.068s
5645 of
1640 (344%)


SELECT
Cost: 0 %
Index Seek (NonClustered)
[Users].[DisplayName] [u]
Cost: 100 %
0.010s
5603 of
76 (7372%)

Possible fixes..

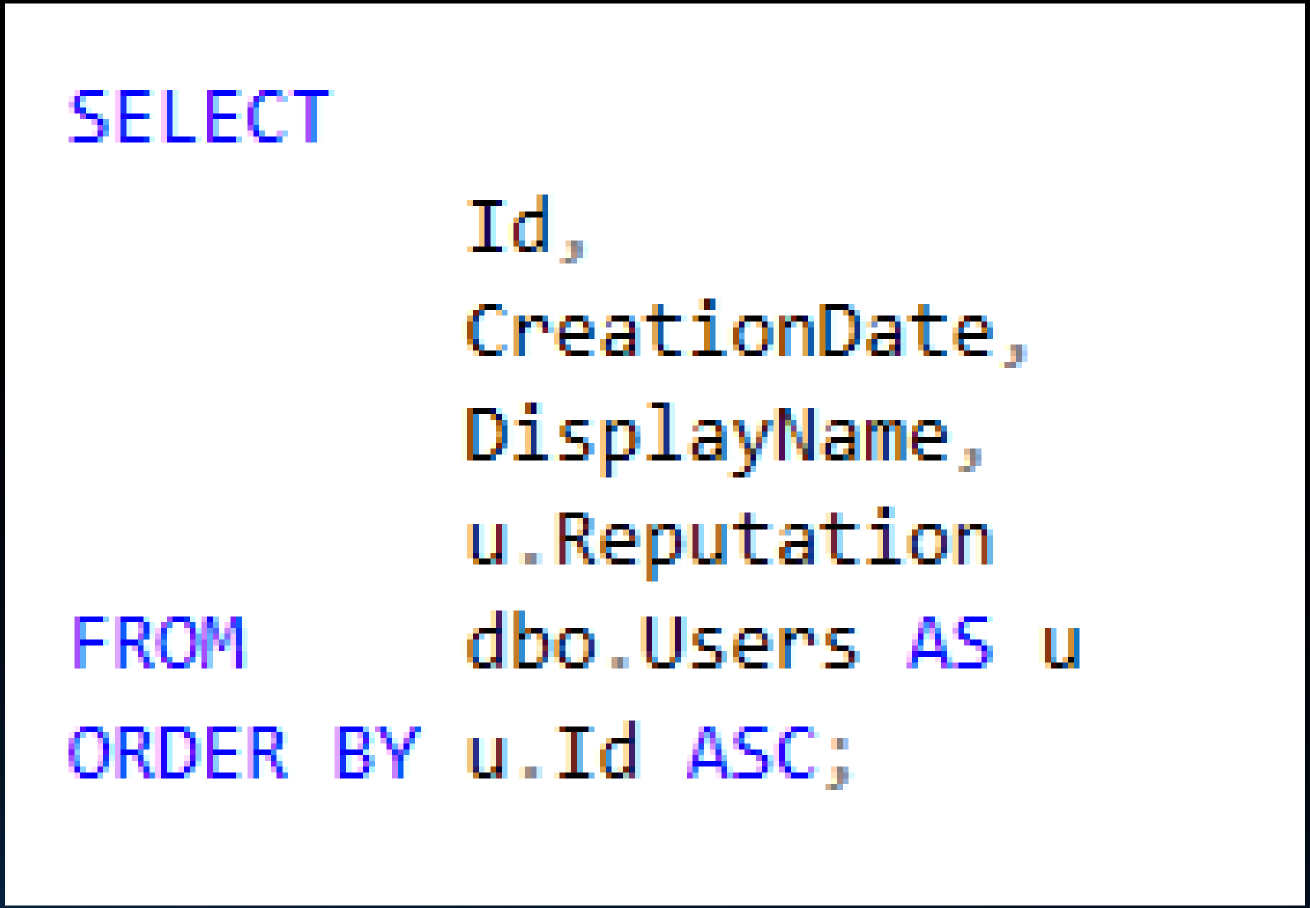


```sql
SELECT Id, DisplayName FROM UsersAndBadges
WHERE DisplayName LIKE 'PETER%'
```

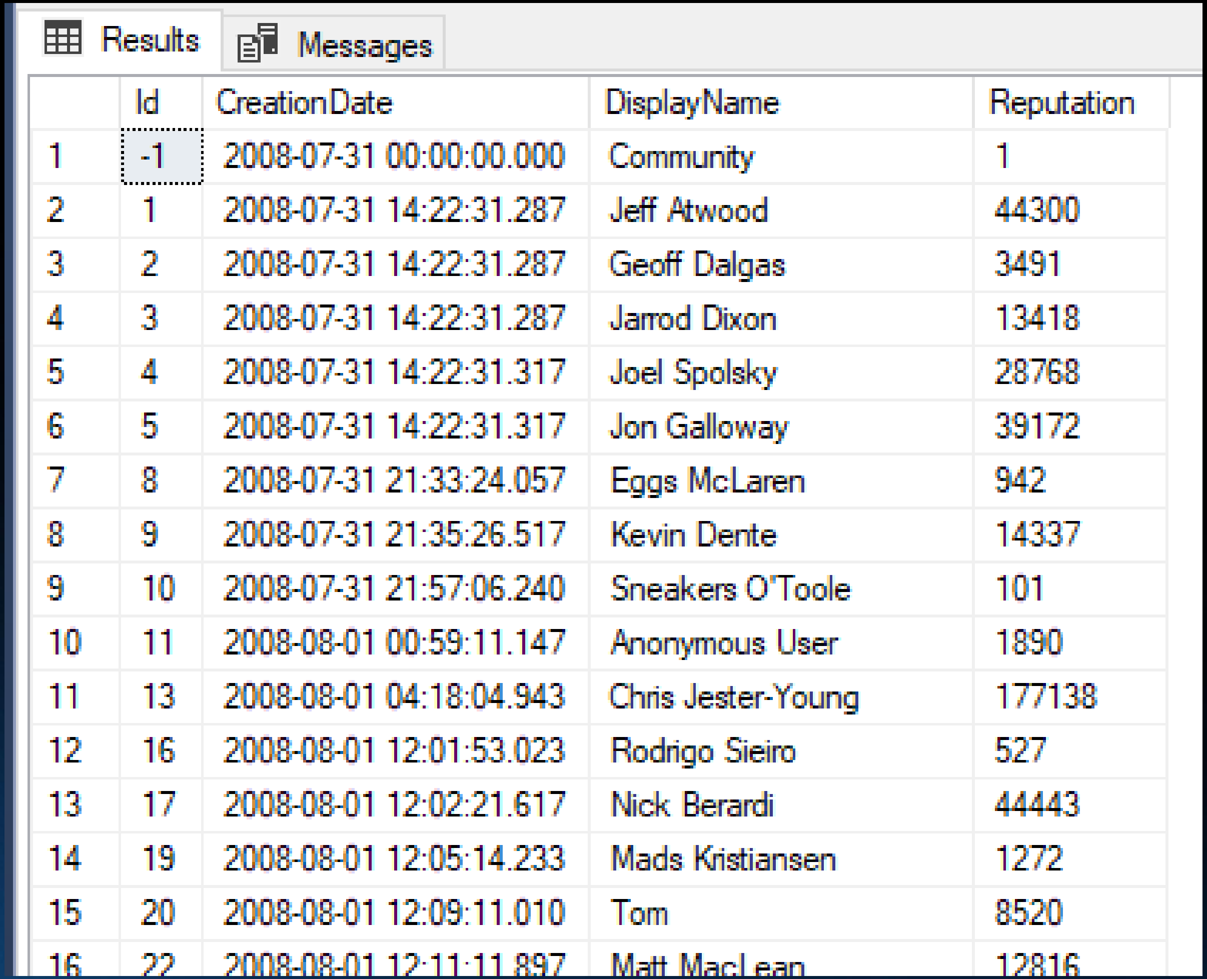
Let’s return all the rows!

In some cases, we may need to display a large amount of data in a grid or generate a report based on query results.

However, in a normal OLTP load, it may not be necessary to return a large number of rows, such as 300k, is it?

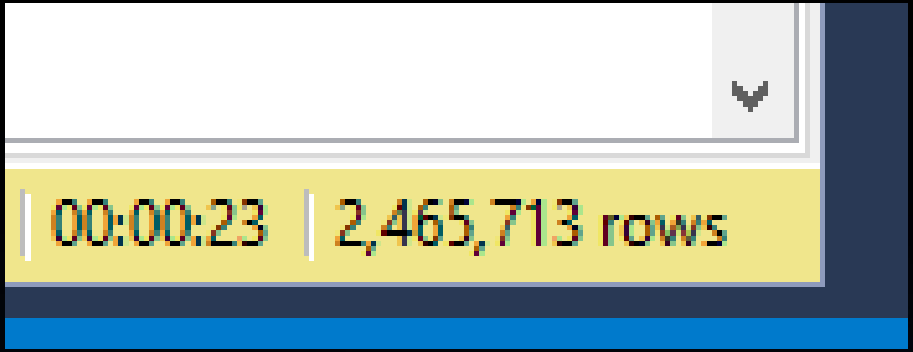


```
SELECT
            Id,
            CreationDate,
            DisplayName,
            u.Reputation
FROM        dbo.Users AS u
ORDER BY u.Id ASC;
```

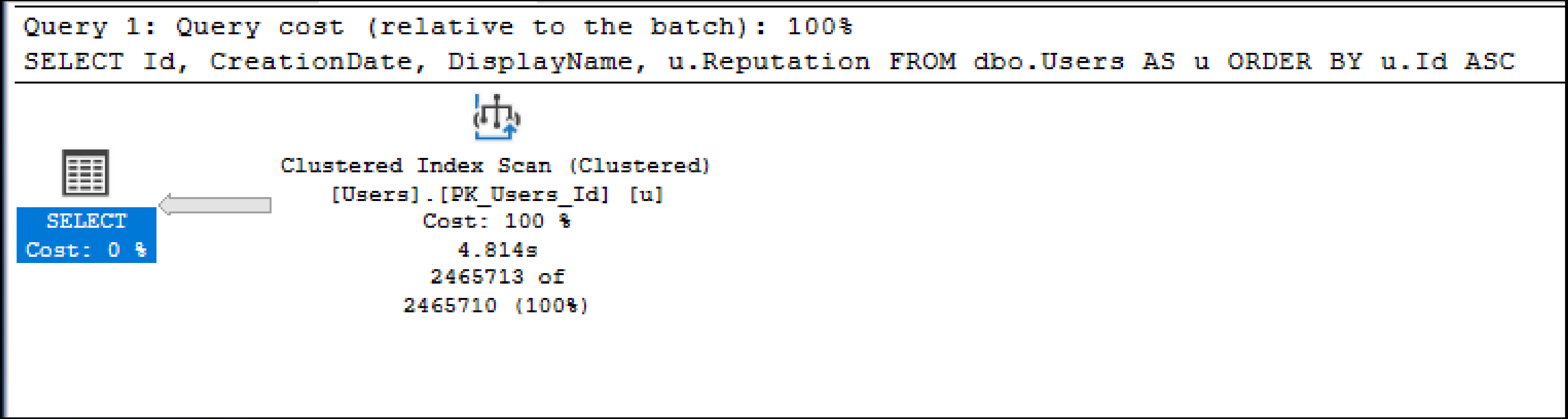


Results | Messages

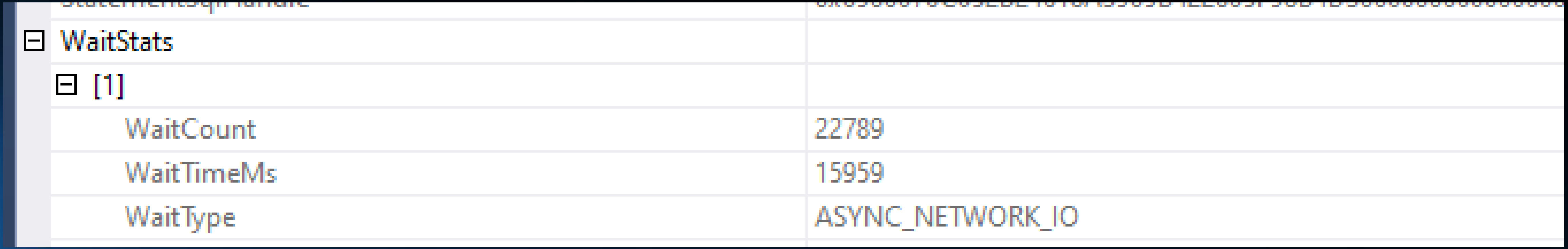
	Id	CreationDate	DisplayName	Reputation
1	-1	2008-07-31 00:00:00.000	Community	1
2	1	2008-07-31 14:22:31.287	Jeff Atwood	44300
3	2	2008-07-31 14:22:31.287	Geoff Dalgas	3491
4	3	2008-07-31 14:22:31.287	Jarrod Dixon	13418
5	4	2008-07-31 14:22:31.317	Joel Spolsky	28768
6	5	2008-07-31 14:22:31.317	Jon Galloway	39172
7	8	2008-07-31 21:33:24.057	Eggs McLaren	942
8	9	2008-07-31 21:35:26.517	Kevin Dente	14337
9	10	2008-07-31 21:57:06.240	Sneakers O'Toole	101
10	11	2008-08-01 00:59:11.147	Anonymous User	1890
11	13	2008-08-01 04:18:04.943	Chris Jester-Young	177138
12	16	2008-08-01 12:01:53.023	Rodrigo Sieiro	527
13	17	2008-08-01 12:02:21.617	Nick Berardi	44443
14	19	2008-08-01 12:05:14.233	Mads Kristiansen	1272
15	20	2008-08-01 12:09:11.010	Tom	8520
16	22	2008-08-01 12:11:11.897	Matt MacLean	12816



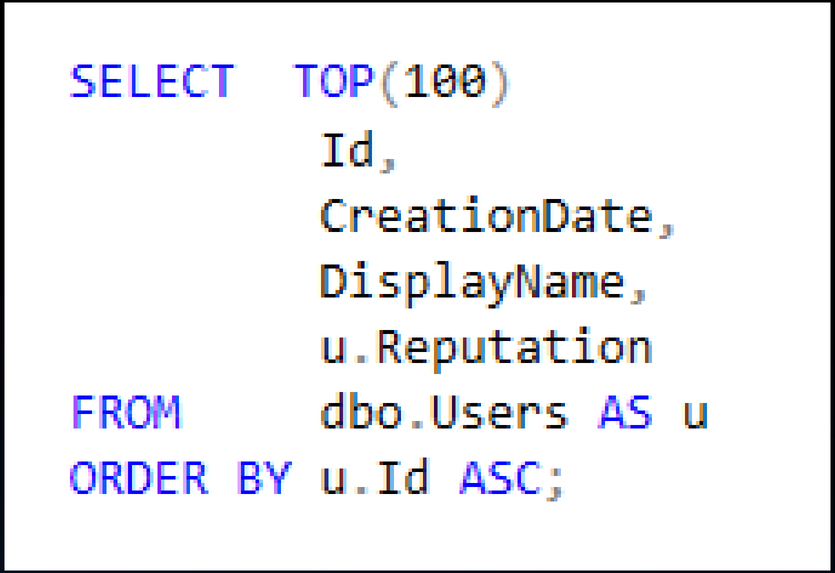
00:00:23 | 2,465,713 rows



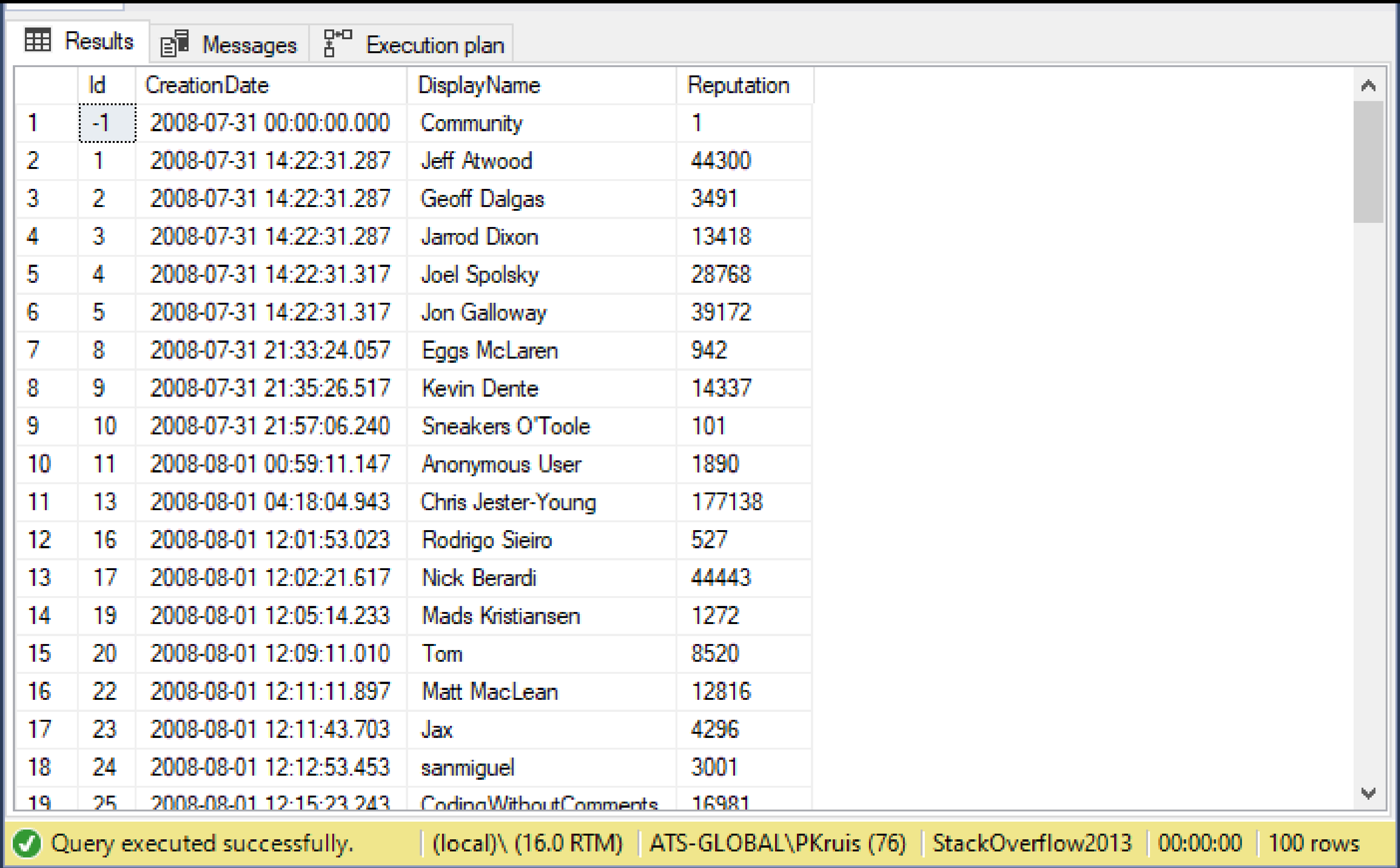
```
Query 1: Query cost (relative to the batch): 100%
SELECT Id, CreationDate, DisplayName, u.Reputation FROM dbo.Users AS u ORDER BY u.Id ASC
```



WaitStats	
[1]	
WaitCount	22789
WaitTimeMs	15959
WaitType	ASYNC_NETWORK_IO

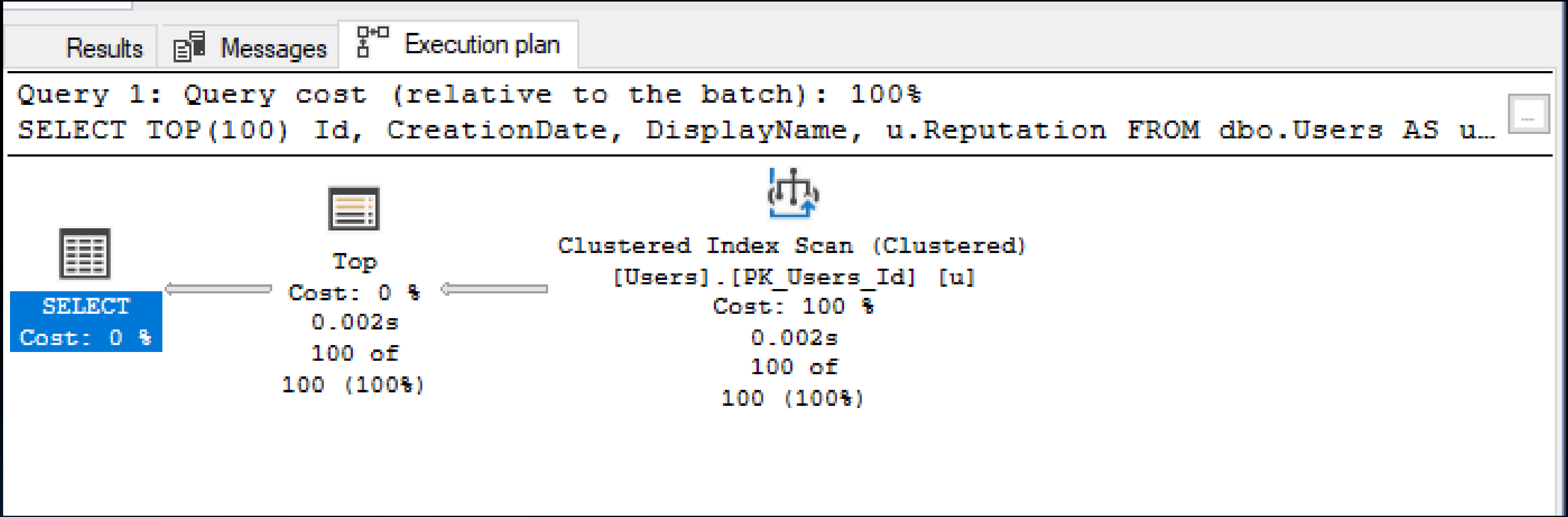


```sql
SELECT  TOP(100)
         Id,
         CreationDate,
         DisplayName,
         u.Reputation
FROM     dbo.Users AS u
ORDER BY u.Id ASC;
```

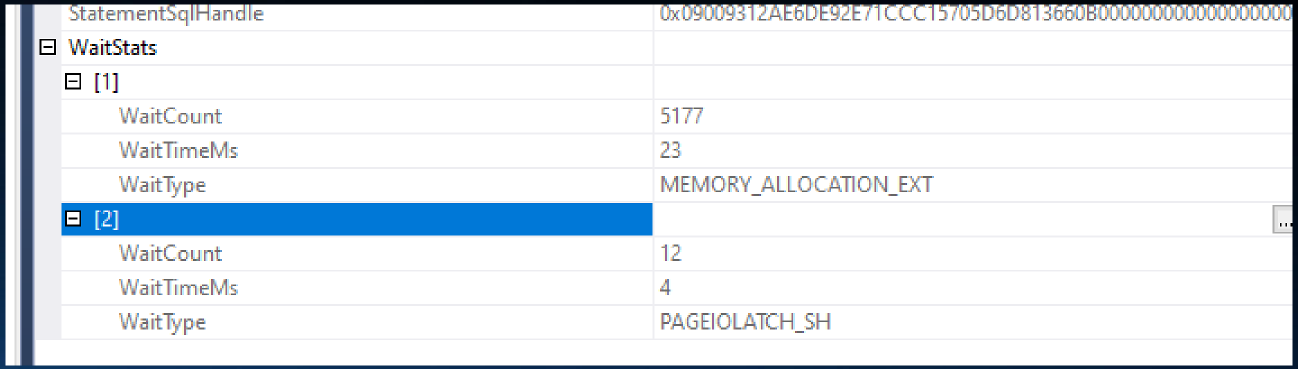


	Id	CreationDate	DisplayName	Reputation
1	-1	2008-07-31 00:00:00.000	Community	1
2	1	2008-07-31 14:22:31.287	Jeff Atwood	44300
3	2	2008-07-31 14:22:31.287	Geoff Dalgas	3491
4	3	2008-07-31 14:22:31.287	Jarrod Dixon	13418
5	4	2008-07-31 14:22:31.317	Joel Spolsky	28768
6	5	2008-07-31 14:22:31.317	Jon Galloway	39172
7	8	2008-07-31 21:33:24.057	Eggs McLaren	942
8	9	2008-07-31 21:35:26.517	Kevin Dente	14337
9	10	2008-07-31 21:57:06.240	Sneakers O'Toole	101
10	11	2008-08-01 00:59:11.147	Anonymous User	1890
11	13	2008-08-01 04:18:04.943	Chris Jester-Young	177138
12	16	2008-08-01 12:01:53.023	Rodrigo Sieiro	527
13	17	2008-08-01 12:02:21.617	Nick Berardi	44443
14	19	2008-08-01 12:05:14.233	Mads Kristiansen	1272
15	20	2008-08-01 12:09:11.010	Tom	8520
16	22	2008-08-01 12:11:11.897	Matt MacLean	12816
17	23	2008-08-01 12:11:43.703	Jax	4296
18	24	2008-08-01 12:12:53.453	sanmiguel	3001
19	25	2008-08-01 12:15:23.243	CodingWithoutComments	16981

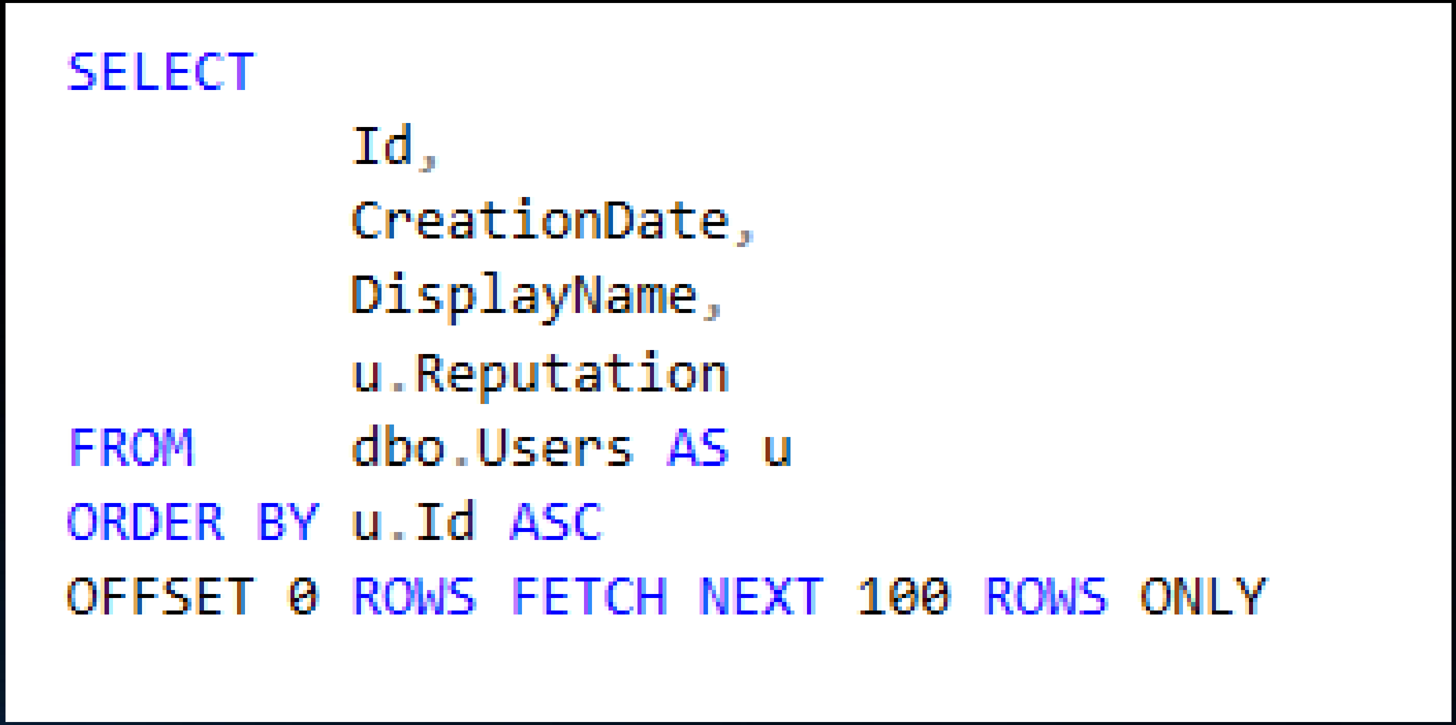
```
Query executed successfully.  | (local)\ (16.0 RTM) | ATS-GLOBAL\PKruis (76) | StackOverflow2013 | 00:00:00 | 100 rows
```



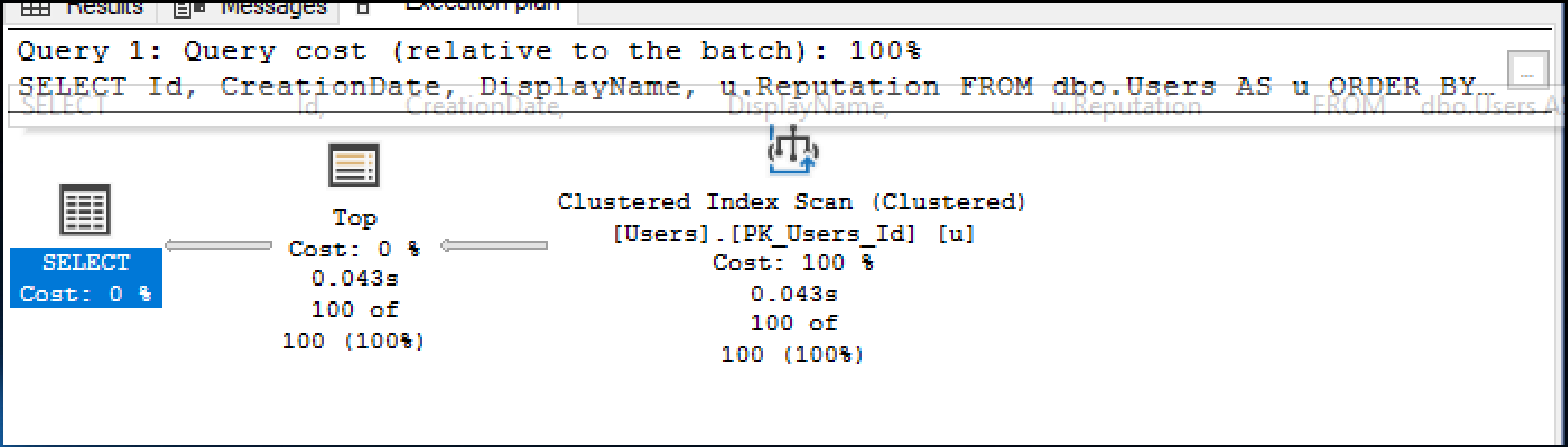
```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP(100) Id, CreationDate, DisplayName, u.Reputation FROM dbo.Users AS u…
```



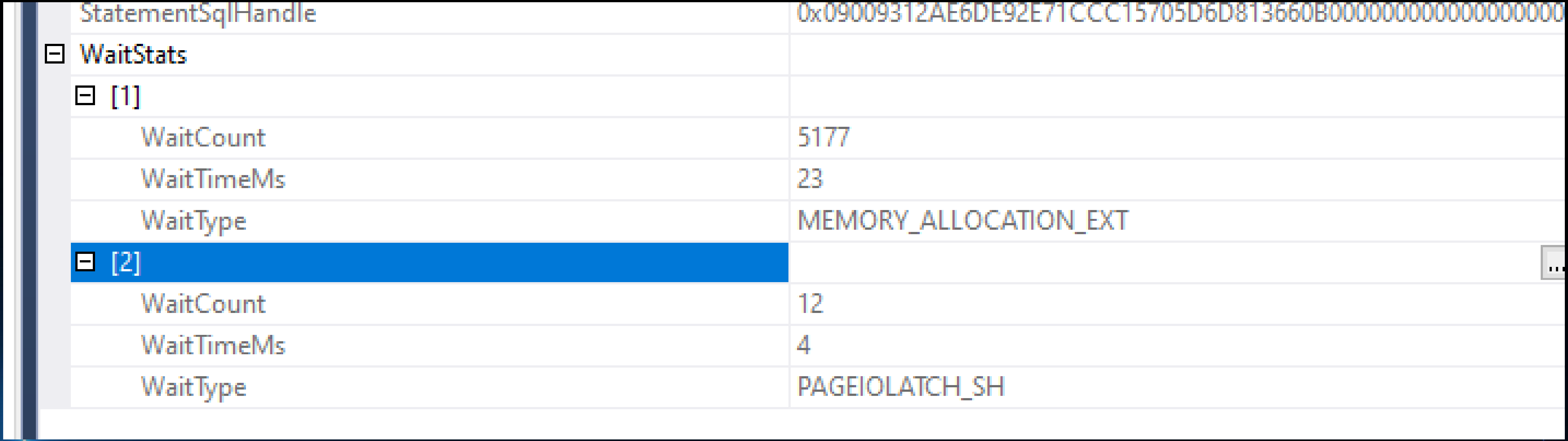
StatementSqlHandle	0x09009312AE6DE92E71CCC15705D6D813660B0000000000000000
⊟ WaitStats	
⊟ [1]	
WaitCount	5177
WaitTimeMs	23
WaitType	MEMORY_ALLOCATION_EXT
⊟ [2]	
WaitCount	12
WaitTimeMs	4
WaitType	PAGEIOLATCH_SH



```sql
SELECT
            Id,
            CreationDate,
            DisplayName,
            u.Reputation
FROM        dbo.Users AS u
ORDER BY u.Id ASC
OFFSET 0 ROWS FETCH NEXT 100 ROWS ONLY
```

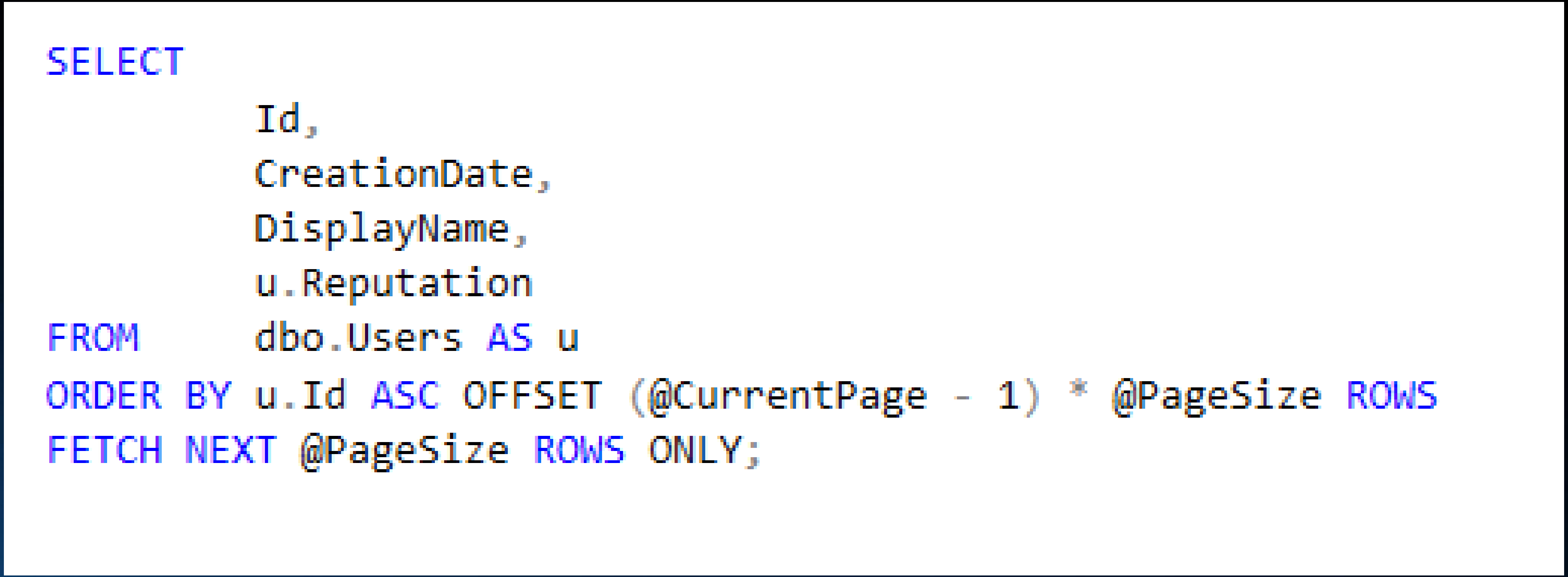


```
Query 1: Query cost (relative to the batch): 100%
SELECT Id, CreationDate, DisplayName, u.Reputation FROM dbo.Users AS u ORDER BY…
```



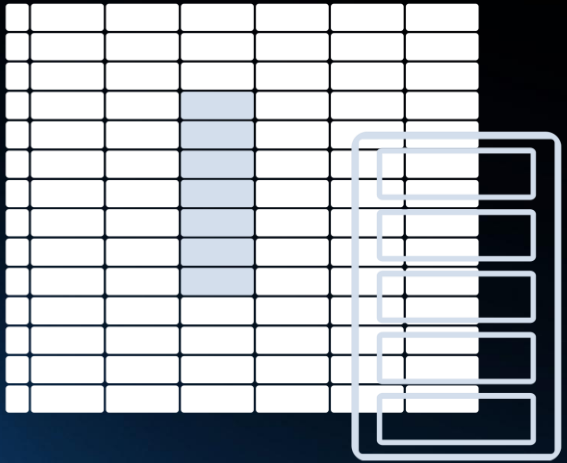
```
StatementSqlHandle   0x09009312AE6DE92E71CCC15705D6D813660B000000000000000000
⊟ WaitStats
  ⊟ [1]
      WaitCount      5177
      WaitTimeMs     23
      WaitType       MEMORY_ALLOCATION_EXT
  ⊟ [2]
      WaitCount      12
      WaitTimeMs     4
      WaitType       PAGEIOLATCH_SH
```

Possible fixes..

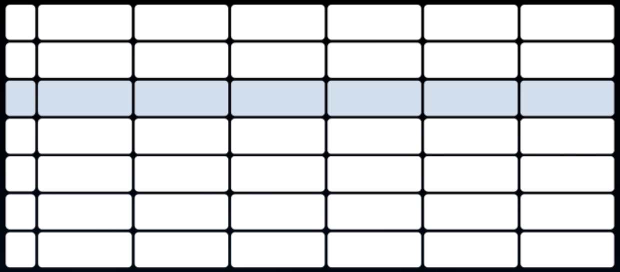


```sql
SELECT
         Id,
         CreationDate,
         DisplayName,
         u.Reputation
FROM     dbo.Users AS u
ORDER BY u.Id ASC OFFSET (@CurrentPage - 1) * @PageSize ROWS
FETCH NEXT @PageSize ROWS ONLY;
```

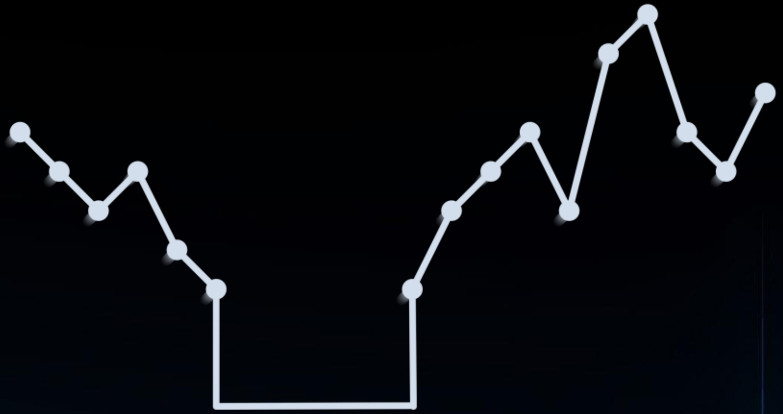
Recap: User Defined Scalar Functions



Not going parallel

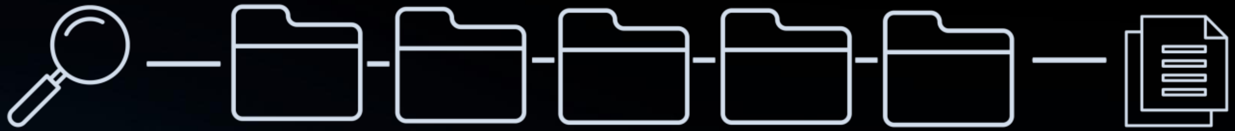


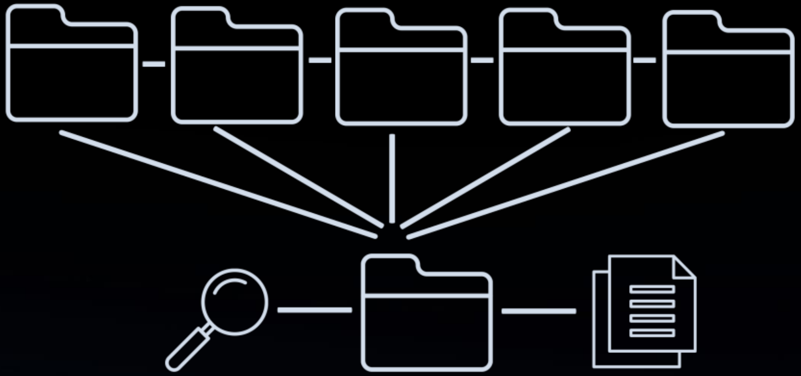
Row by row processing



Missing statistics

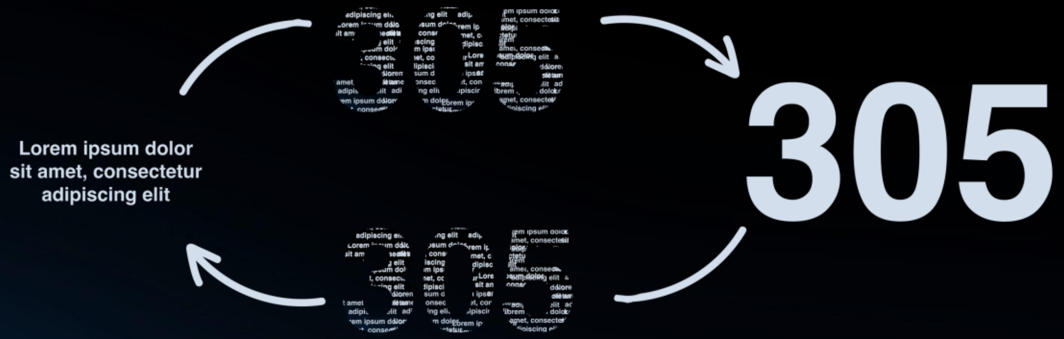
Recap: Functions in the WHERE clause

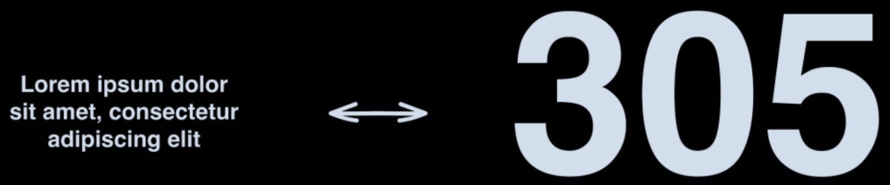




Scanning through everything vs Seeking the right data

Recap: Implicit casting





Make sure you cast the right size

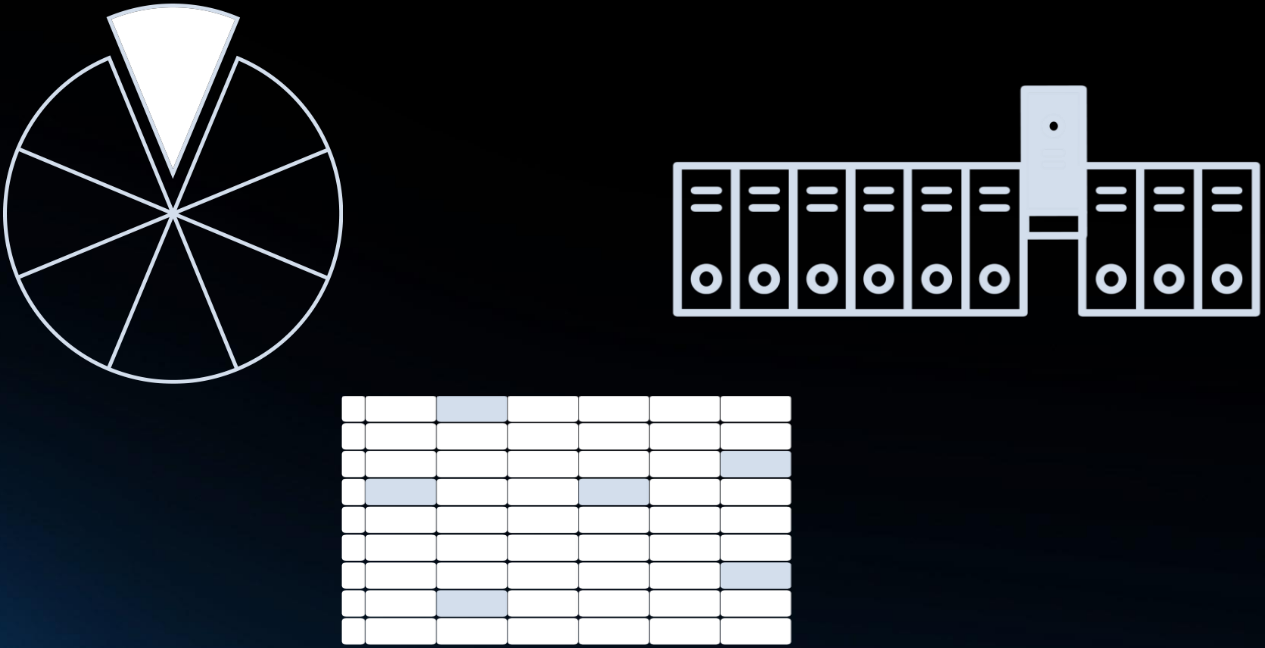
or better..

Fix your data types

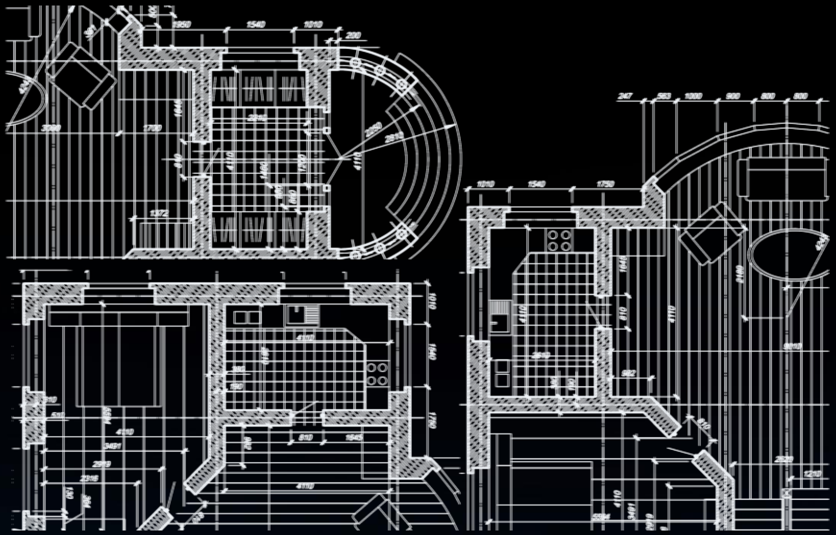
Recap: CTE

Each time a CTE is references, it is being queried again. You might want to save your ‘inbetween’ results to a temp table to gain performance.

Recap: SELECT *

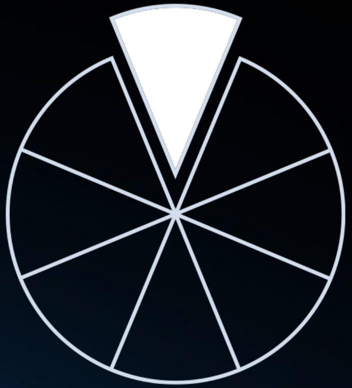


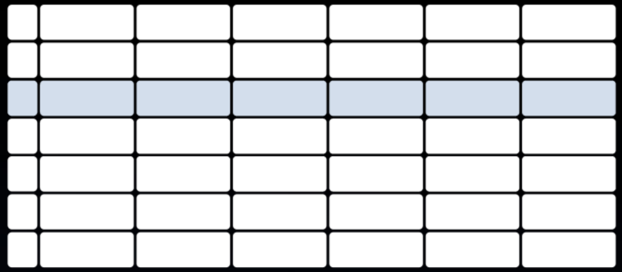
Only select the part of the data you need.



Optimize query plans

Recap: Return all the rows





Think if we really need al the rows, or that a subset will work

Implement pagination

Bad practices caught in the wild

Peter Kruis

https://www.linkedin.com/in/peter-kruis

peterkruis@hotmail.com