INTELIGENTNO PROCESIRANJE U SQL SERVERU 2019

MILOŠ RADIVOJEVIĆ

DATA PLATFORM MVP

26. DECEMBAR 2020

SCALAR UDF INLINING

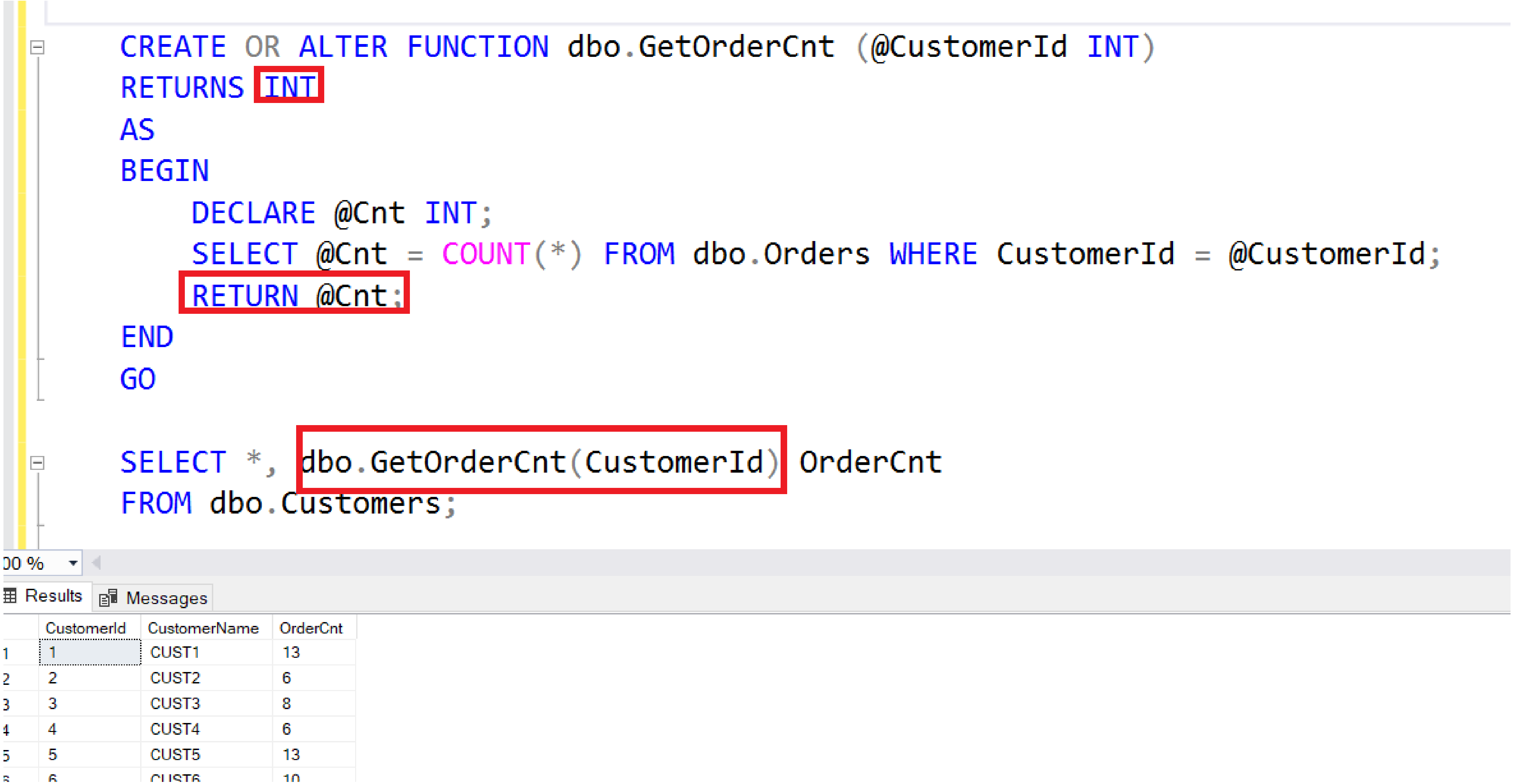
FUNCTIONS

- Code reuse, encapsulation and modularity
- Complex business rules or computations
- Single place change
- Written once, invoke from many modules
- Reduce network traffic

USER-DEFINED FUNCTIONS IN SQL SERVER

- Scalar functions
- Inline table-valued functions 👍
- Multi-statement table-valued functions – MSTVFs

SCALAR FUNCTIONS

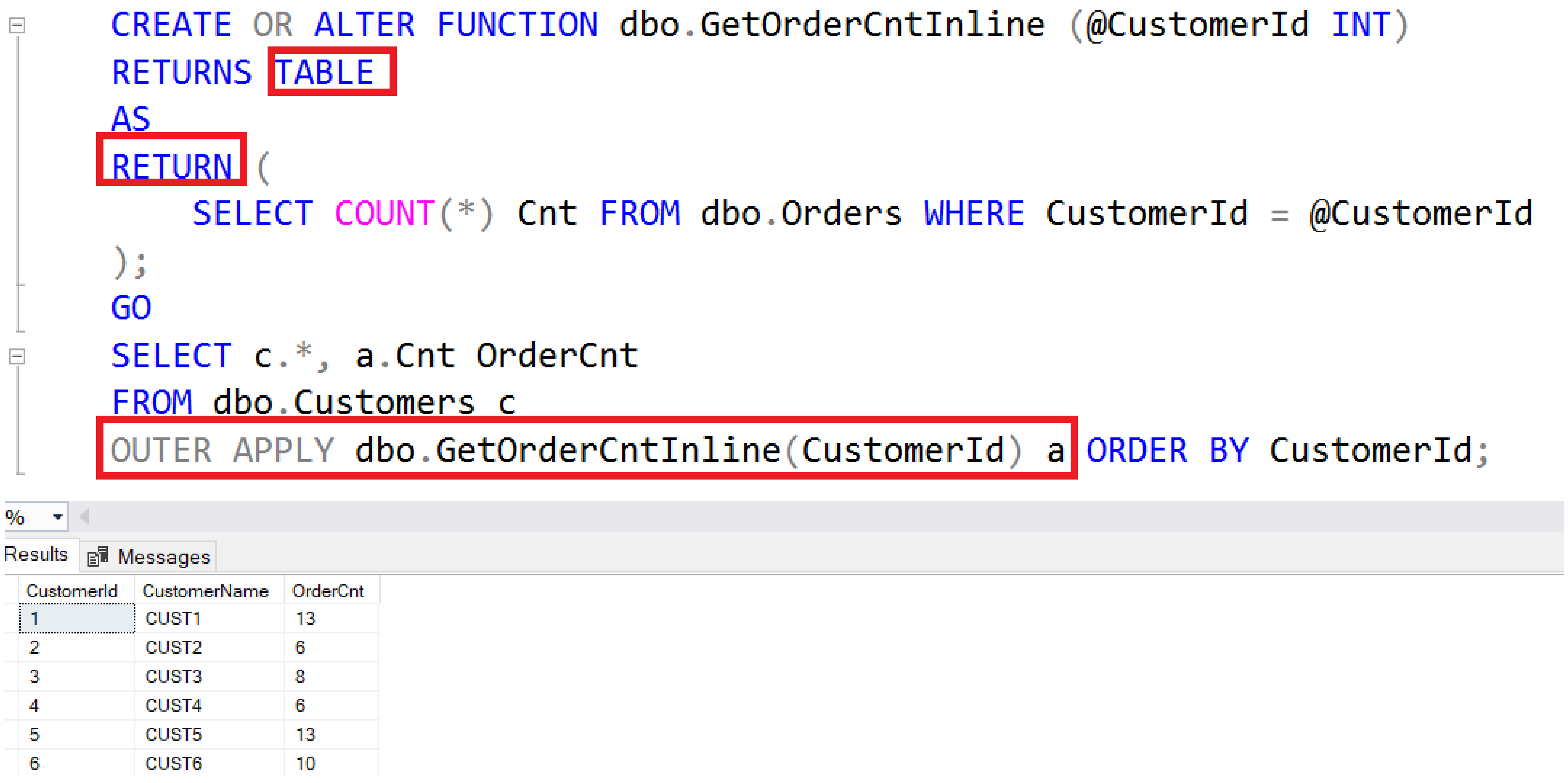


```
CREATE OR ALTER FUNCTION dbo.GetOrderCnt (@CustomerId INT)
RETURNS INT
AS
BEGIN
    DECLARE @Cnt INT;
    SELECT @Cnt = COUNT(*) FROM dbo.Orders WHERE CustomerId = @CustomerId;
    RETURN @Cnt;
END
GO

SELECT *, dbo.GetOrderCnt(CustomerId) OrderCnt
FROM dbo.Customers;
```

	CustomerId	CustomerName	OrderCnt
1	1	CUST1	13
2	2	CUST2	6
3	3	CUST3	8
4	4	CUST4	6
5	5	CUST5	13
6	6	CUST6	10

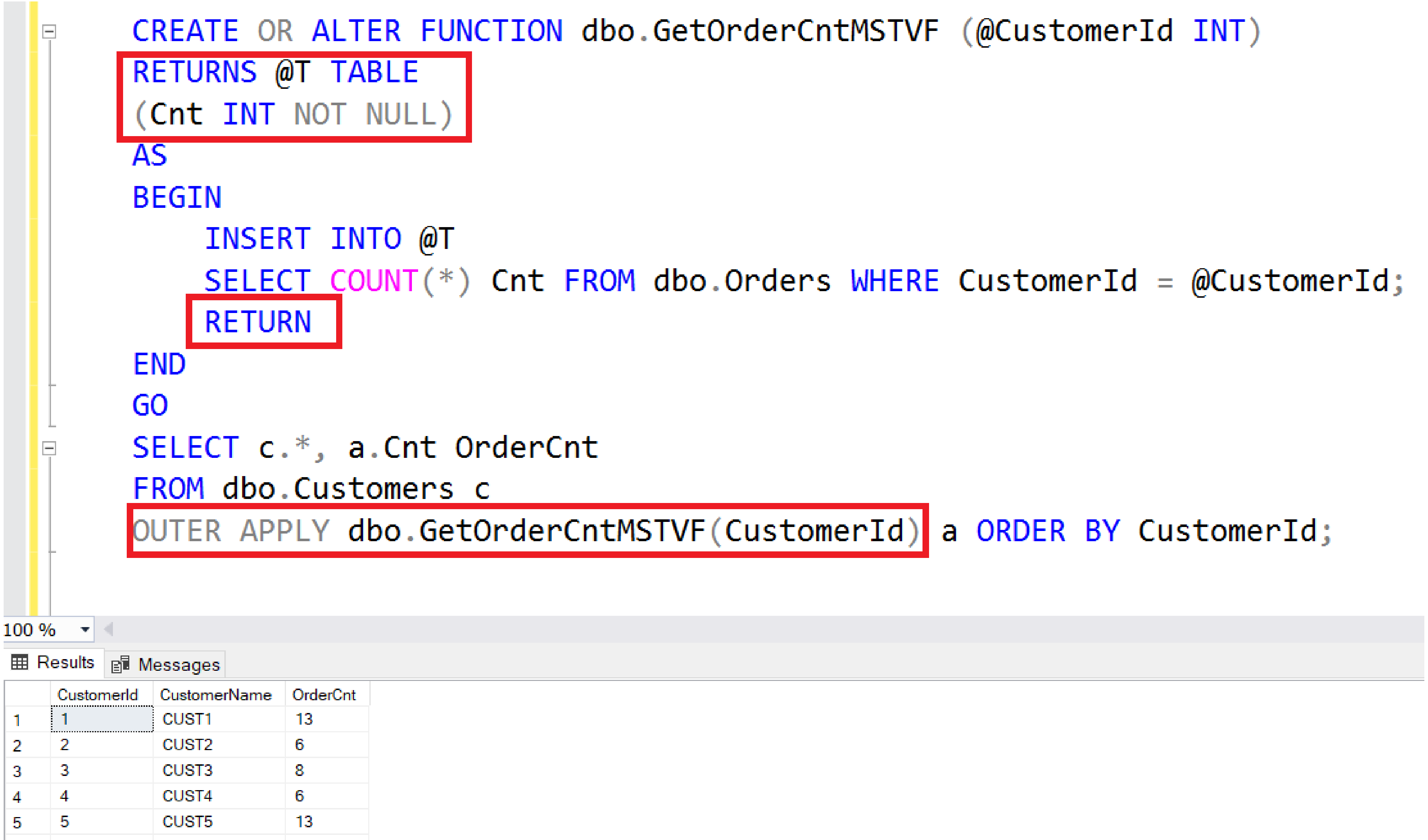
INLINE TABLE-VALUED FUNCTIONS



```
CREATE OR ALTER FUNCTION dbo.GetOrderCntInline (@CustomerId INT)
RETURNS TABLE
AS
RETURN (
    SELECT COUNT(*) Cnt FROM dbo.Orders WHERE CustomerId = @CustomerId
);
GO
SELECT c.*, a.Cnt OrderCnt
FROM dbo.Customers c
OUTER APPLY dbo.GetOrderCntInline(CustomerId) a ORDER BY CustomerId;
```

CustomerId	CustomerName	OrderCnt
1	CUST1	13
2	CUST2	6
3	CUST3	8
4	CUST4	6
5	CUST5	13
6	CUST6	10

MULTI-STATEMENT TABLE-VALUED FUNCTIONS



```
CREATE OR ALTER FUNCTION dbo.GetOrderCntMSTVF (@CustomerId INT)
RETURNS @T TABLE
(Cnt INT NOT NULL)
AS
BEGIN
    INSERT INTO @T
    SELECT COUNT(*) Cnt FROM dbo.Orders WHERE CustomerId = @CustomerId;
    RETURN
END
GO
SELECT c.*, a.Cnt OrderCnt
FROM dbo.Customers c
OUTER APPLY dbo.GetOrderCntMSTVF(CustomerId) a ORDER BY CustomerId;
```

	CustomerId	CustomerName	OrderCnt
1	1	CUST1	13
2	2	CUST2	6
3	3	CUST3	8
4	4	CUST4	6
5	5	CUST5	13

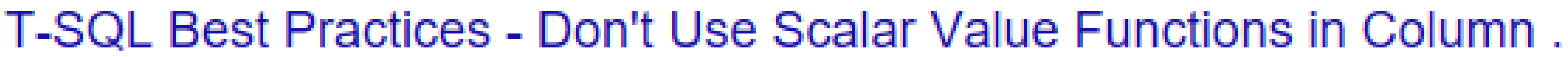
BUT…

SCALAR UDFS IN SQL SERVER

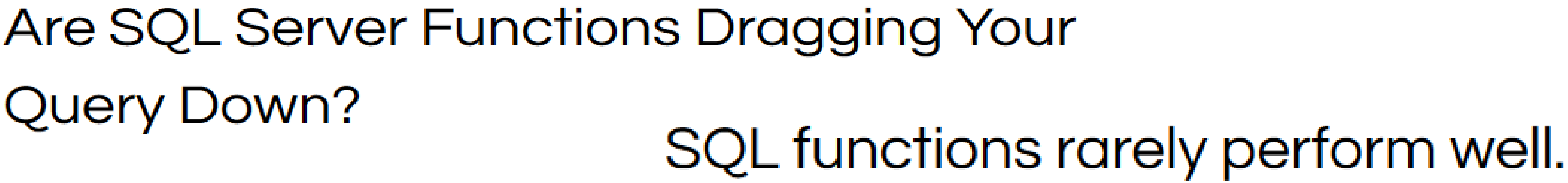
Why do SQL Server Scalar-valued functions get slower?

Refactor SQL Server scalar UDF to inline TVF to improve performance

Why SQL Server scalar functions are bad?



T-SQL Best Practices - Don't Use Scalar Value Functions in Column .



Are SQL Server Functions Dragging Your Query Down?

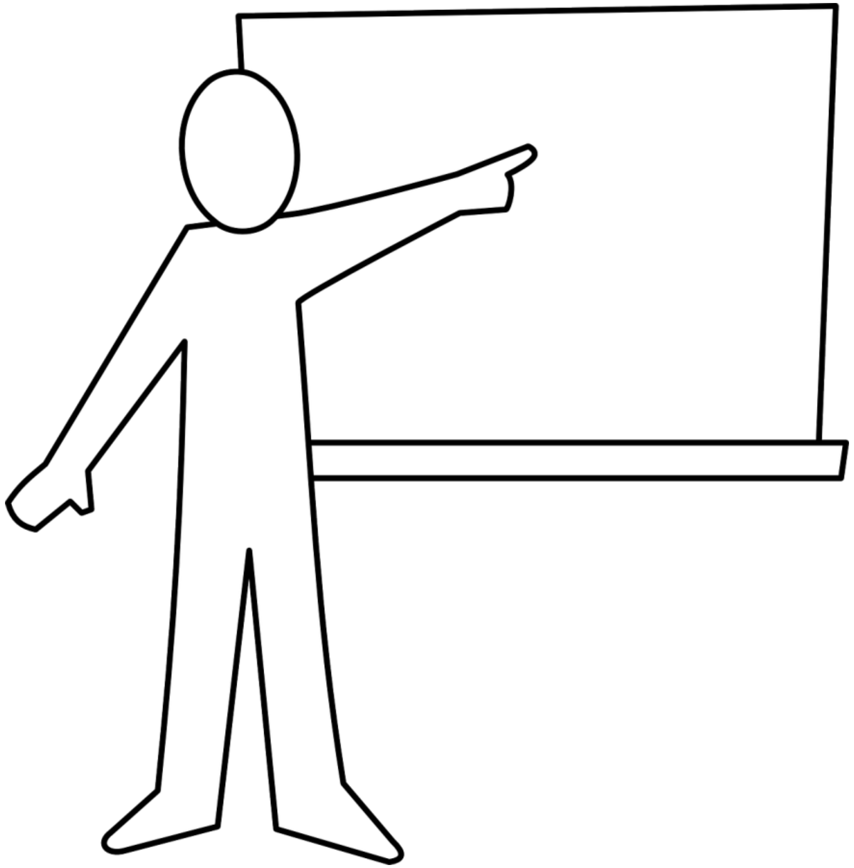
SQL functions rarely perform well.

SCALAR UDFS IN SQL SERVER

- They are very slow

- Iterative invocation
  - Overhead for invoking function – once per row
  - No cross-statement optimization
- Only serial execution plans possible

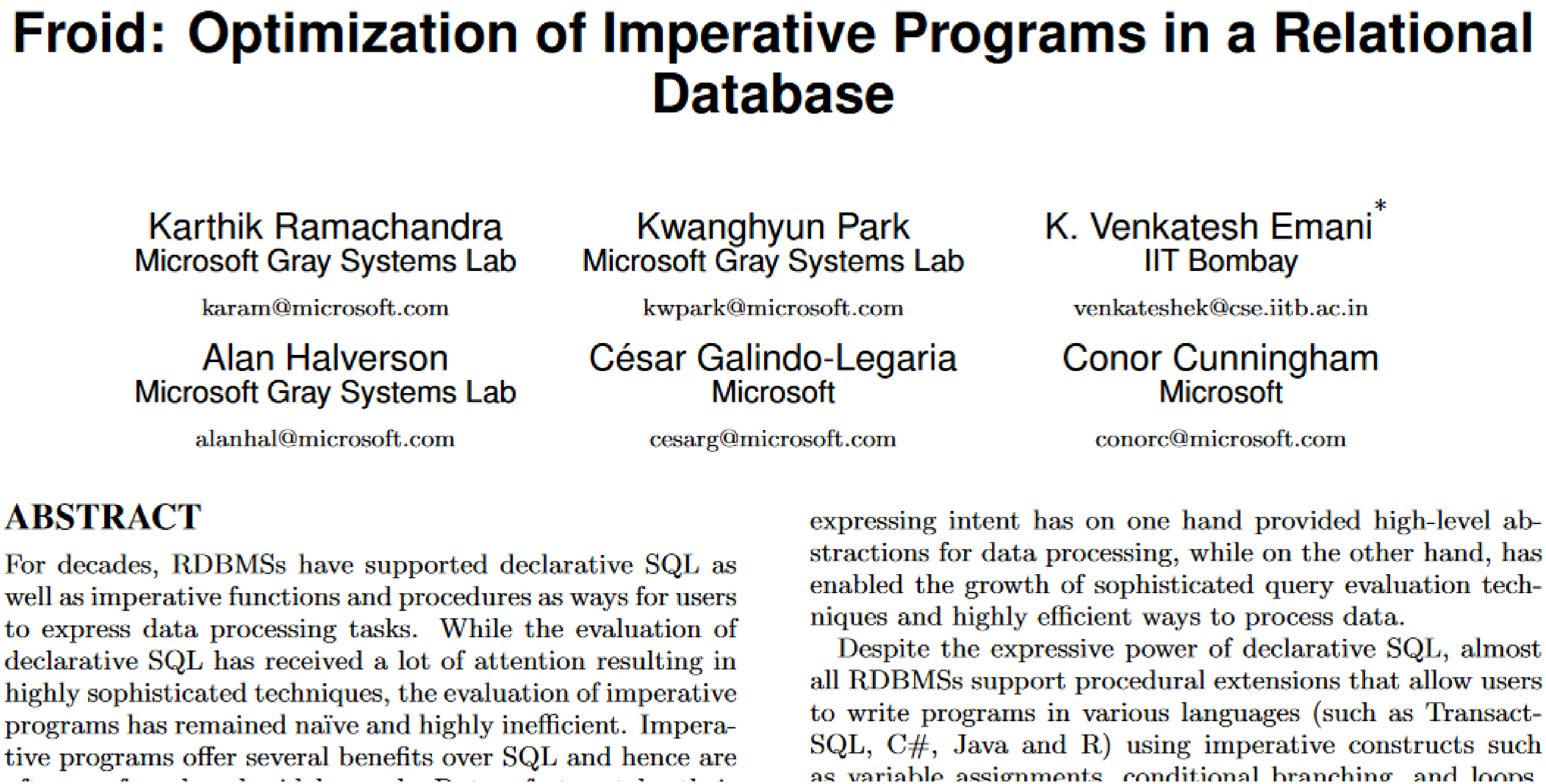
WARUM SIND SKALARFUNKTIONEN LANGSAM?



DEMO

FROID FRAMEWORK

http://www.vldb.org/pvldb/vol11/p432-ramachandra.pdf



Froid: Optimization of Imperative Programs in a Relational Database

Karthik Ramachandra
Microsoft Gray Systems Lab
karam@microsoft.com

Kwanghyun Park
Microsoft Gray Systems Lab
kwpark@microsoft.com

K. Venkatesh Emani*
IIT Bombay
venkateshek@cse.iitb.ac.in

Alan Halverson
Microsoft Gray Systems Lab
alanhal@microsoft.com

César Galindo-Legaria
Microsoft
cesarg@microsoft.com

Conor Cunningham
Microsoft
conorc@microsoft.com

ABSTRACT

For decades, RDBMSs have supported declarative SQL as well as imperative functions and procedures as ways for users to express data processing tasks. While the evaluation of declarative SQL has received a lot of attention resulting in highly sophisticated techniques, the evaluation of imperative programs has remained naïve and highly inefficient. Imperative programs offer several benefits over SQL and hence are

expressing intent has on one hand provided high-level abstractions for data processing, while on the other hand, has enabled the growth of sophisticated query evaluation techniques and highly efficient ways to process data.

Despite the expressive power of declarative SQL, almost all RDBMSs support procedural extensions that allow users to write programs in various languages (such as Transact-SQL, C#, Java and R) using imperative constructs such as variable assignments, conditional branching, and loops.

FROID FRAMEWORK

- Goal – improve queries where scalar UDFs are problem

- Scalar UDF Inlining Feature (Froid framework):
  - transforms imperative scalar UDFs into relational expressions (IF => CASE WHEN)
  - Embeds them in the calling query by using APPLY operator
  - Optimize expressions or subqueries

- Result:
  - Performance improved (more efficient plan)
  - Execution plan could be parallel

FROID TRANSORMATION

```
DECLARE @val VARCHAR(10);
DECLARE @a INT = 2000;

    IF @a > 1000
        SET @val = 'HIGH';
    ELSE IF @a > 500
        SET @val = 'MEDIUM'
    ELSE
        SET @val = 'LOW'

    SELECT @val;
```

```
SELECT q5.v
FROM
(
         (SELECT 2000 AS a) AS q1
         OUTER APPLY
                  (SELECT CASE WHEN q1.a > 1000 THEN 'HIGH' END AS val)
AS q2
         OUTER APPLY
         (SELECT CASE WHEN q1.a > 500 THEN 'HIGH' END AS val) AS q3
         OUTER APPLY
                  (SELECT 'LOW' AS val) AS q4
         OUTER APPLY
                  (SELECT CASE WHEN q2.val IS NOT NULL
            THEN q2.val
            WHEN q3.val IS NOT NULL
            THEN q3.val
            ELSE q4.val
        END v) AS q5
);
```

FROID TRANSORMATION

```
DECLARE @val VARCHAR(10);
DECLARE @a INT = 2000;

    IF @a > 1000
        SET @val = 'HIGH';
    ELSE IF @a > 500
        SET @val = 'MEDIUM'
    ELSE
        SET @val = 'LOW'

    SELECT @val;
```

```
SELECT q5.v
FROM
(
         (SELECT 2000 AS a) AS q1
         OUTER APPLY
                  (SELECT CASE WHEN q1.a > 1000 THEN 'HIGH' END AS val)
AS q2
         OUTER APPLY
         (SELECT CASE WHEN q1.a > 500 THEN 'HIGH' END AS val) AS q3
         OUTER APPLY
                  (SELECT 'LOW' AS val) AS q4
         OUTER APPLY
                  (SELECT CASE WHEN q2.val IS NOT NULL
            THEN q2.val
            WHEN q3.val IS NOT NULL
            THEN q3.val
            ELSE q4.val
        END v) AS q5
);
```

FROID TRANSORMATION

```
DECLARE @val VARCHAR(10);
DECLARE @a INT = 2000;

    IF @a > 1000
        SET @val = 'HIGH';
    ELSE IF @a > 500
        SET @val = 'MEDIUM'
    ELSE
        SET @val = 'LOW'

    SELECT @val;
```

```
SELECT q5.v
FROM
(
         (SELECT 2000 AS a) AS q1
         OUTER APPLY
                  (SELECT CASE WHEN q1.a > 1000 THEN 'HIGH' END AS val)
AS q2
         OUTER APPLY
         (SELECT CASE WHEN q1.a > 500 THEN 'HIGH' END AS val) AS q3
         OUTER APPLY
                  (SELECT 'LOW' AS val) AS q4
         OUTER APPLY
                  (SELECT CASE WHEN q2.val IS NOT NULL
            THEN q2.val
            WHEN q3.val IS NOT NULL
            THEN q3.val
            ELSE q4.val
        END v) AS q5
);
```

FROID TRANSORMATION

```
DECLARE @val VARCHAR(10);
DECLARE @a INT = 2000;

    IF @a > 1000
        SET @val = 'HIGH';
    ELSE IF @a > 500
        SET @val = 'MEDIUM'
    ELSE
        SET @val = 'LOW'

    SELECT @val;
```

```
SELECT q5.v
FROM
(
         (SELECT 2000 AS a) AS q1
         OUTER APPLY
                  (SELECT CASE WHEN q1.a > 1000 THEN 'HIGH' END AS val)
AS q2
         OUTER APPLY
         (SELECT CASE WHEN q1.a > 500 THEN 'HIGH' END AS val) AS q3
         OUTER APPLY
                  (SELECT 'LOW' AS val) AS q4
         OUTER APPLY
                  (SELECT CASE WHEN q2.val IS NOT NULL
            THEN q2.val
            WHEN q3.val IS NOT NULL
            THEN q3.val
            ELSE q4.val
        END v) AS q5
);
```

FROID TRANSORMATION

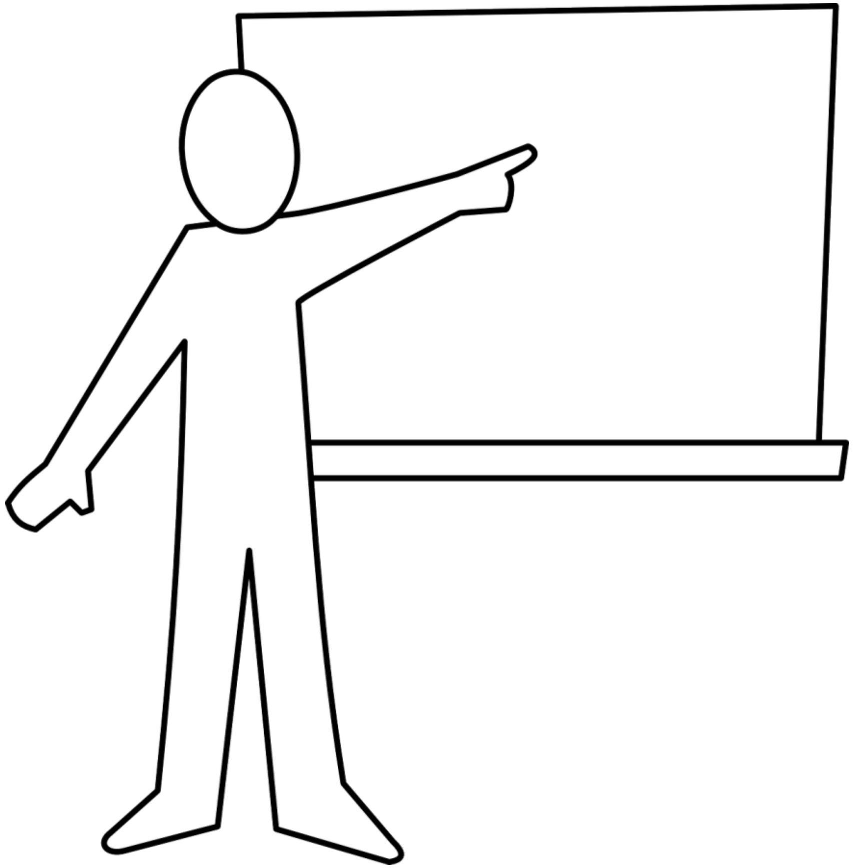
```
DECLARE @val VARCHAR(10);
DECLARE @a INT = 2000;

    IF @a > 1000
        SET @val = 'HIGH';
    ELSE IF @a > 500
        SET @val = 'MEDIUM'
    ELSE
        SET @val = 'LOW'

    SELECT @val;
```

```
SELECT q5.v
FROM
(
    (SELECT 2000 AS a) AS q1
    OUTER APPLY
         (SELECT CASE WHEN q1.a > 1000 THEN 'HIGH' END AS val)
AS q2
    OUTER APPLY
    (SELECT CASE WHEN q1.a > 500 THEN 'HIGH' END AS val) AS q3
    OUTER APPLY
         (SELECT 'LOW' AS val) AS q4
    OUTER APPLY
         (SELECT CASE WHEN q2.val IS NOT NULL
      THEN q2.val
      WHEN q3.val IS NOT NULL
      THEN q3.val
      ELSE q4.val
    END v) AS q5
);
```

SCALAR UDF INLINING



DEMO

SCALAR UDF INLINING

- Not all UDFs can be inlined
  - Check whether a function can be inlined :

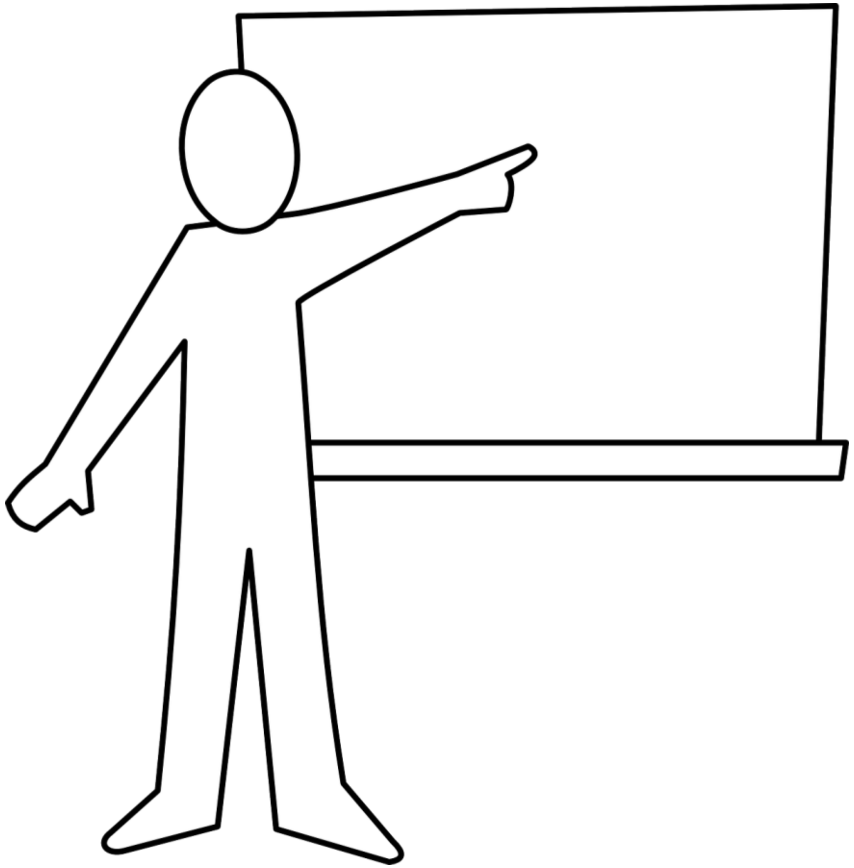
```
SELECT CONCAT(SCHEMA_NAME(o.schema_id),'.',o.name), is_inlineable
FROM sys.sql_modules m INNER JOIN sys.objects o ON o.object_id = m.object_id WHERE o.type = 'FN';
```

- **is_inlineable = 1** does not mean that the function is necessarily inlined
- Decision is made when the query referencing a scalar UDF is compiled

SCALAR UDF INLINING - LIMITATIONS

- UDF does not invoke any intrinsic function that is either time-dependent or has side effects such as GETDATE() or NEWSEQUENTIALID
- The UDF does not reference table variables, table-valued parameters or user-defined types
- UDF is not natively compiled (interop is supported)
- UDF is not used in a computed column or a check constraint definition
- The UDF is not a partition function
- Full list of limitations: https://docs.microsoft.com/de-de/sql/relational-databases/user-defined-functions/scalar-udf-inlining?view=sql-server-ver15

SCALAR UDF INLINING - REGRESSIONS

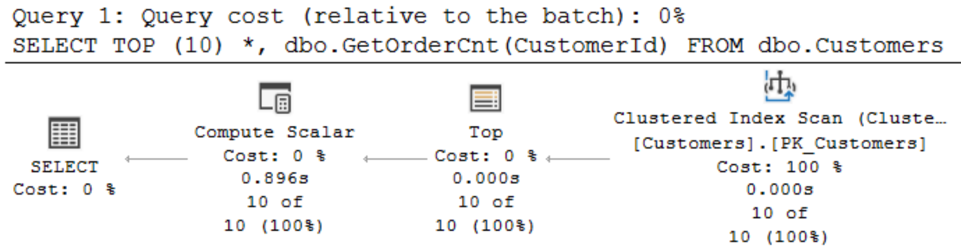


DEMO

REGRESSION?

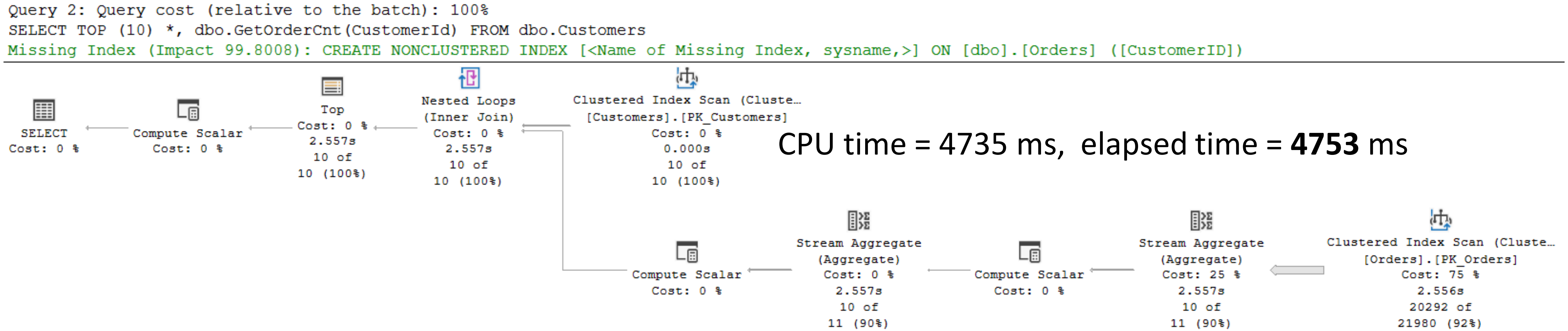
5x

```
ALTER DATABASE TestDb SET COMPATIBILITY_LEVEL = 140;
GO
SELECT TOP (10) * FROM dbo.Customers WHERE dbo.GetOrderCnt(CustomerId) > 25;
GO
ALTER DATABASE TestDb SET COMPATIBILITY_LEVEL = 150;
GO
SELECT TOP (10) * FROM dbo.Customers WHERE dbo.GetOrderCnt(CustomerId) > 25;
```



Query 1: Query cost (relative to the batch): 0%
SELECT TOP (10) *, dbo.GetOrderCnt(CustomerId) FROM dbo.Customers

CPU time = 5922 ms, elapsed time = **873** ms.



Query 2: Query cost (relative to the batch): 100%
SELECT TOP (10) *, dbo.GetOrderCnt(CustomerId) FROM dbo.Customers
Missing Index (Impact 99.8008): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Orders] ([CustomerID])

CPU time = 4735 ms, elapsed time = **4753** ms

REGRESSION?

```
ALTER DATABASE TestDb SET COMPATIBILITY_LEVEL = 140;
GO
SELECT TOP (10) * FROM dbo.Customers WHERE dbo.GetOrderCnt(CustomerId) > 25;
GO
ALTER DATABASE TestDb SET COMPATIBILITY_LEVEL = 150;
GO
SELECT TOP (10) * FROM dbo.Customers WHERE dbo.GetOrderCnt(CustomerId) > 25;
```

Solution

```
SELECT TOP (10) * FROM dbo.Customers WHERE dbo.GetOrderCnt(CustomerId) > 25
OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'));
```

CONFIGURATION

- Enable:

```
ALTER DATABASE current SET COMPATIBILITY_LEVEL = 150;

ALTER DATABASE SCOPED CONFIGURATION SET TSQL_SCALAR_UDF_INLINING = ON;

CREATE OR ALTER FUNCTION dbo.getMaxOrderDate(@CustID INT) RETURNS DATETIME WITH INLINE
= ON
```

- Disable:

```
ALTER DATABASE SCOPED CONFIGURATION SET TSQL_SCALAR_UDF_INLINING = OFF;

OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING’));

CREATE OR ALTER FUNCTION dbo.getMaxOrderDate(@CustID INT) RETURNS DATETIME WITH INLINE
= OFF
```

CONCLUSION

- Very promising feature
  - Improvements with no efforts
  - Part of the Standard Edition

- Many limitations (GETDATE(), table variables…)

- Very useful for small and medium companies (not enough people to rewrite UDFs) 3rd party tools

- BUT
  - As far I know, it is still not available in Azure!
  - A lot of bugs in the meantime!