

# APPLY operator in Action

Andrea Martorana Tusa

@bruco441

*andrea.martoranatusa@gmail.com*



# Sponsors

---



# Organizers

---

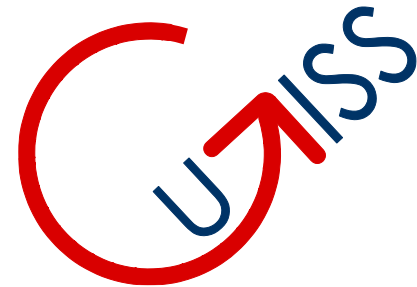


UNIVERSITÀ DEGLI STUDI DI PARMA

ENGAGE  
IT SERVICES



getlatestversion.it



November 26°, 2016



#566 | PARMA 2016

# Andrea Martorana Tusa | @bruco441

---

- Developer working in IT department of **Banco Popolare**. Focused on all the applications involved in Human Resources management
- Main tasks: SQL Server and Oracle development, data warehousing, reporting, BI, Analysis Services, C#, R
- MCTS "SQL Server Developer"
- Speaker at SQL Saturdays, SQL Nexus, PASS Italian Virtual chapter, Community Days
- Author on sqlservercentral.com, UGISS (User Group Italiano SQL Server)

# Agenda

---

- Introducing the APPLY operator
- APPLY vs. JOIN
- CROSS APPLY - OUTER APPLY
- Use cases
- Implicit APPLY
- Conclusions

# Introducing the APPLY operator

---

APPLY is a table operator in T-SQL. Table operators are operators that act on tables provided as an input, applies a logical query processing phase and return a table results.

Table operators in SQL Server are, for example: JOIN, APPLY, PIVOT, UNPIVOT.

APPLY, PIVOT, UNPIVOT are not SQL standard operators. They exist only in SQL Server.

In case of APPLY, the second element invoked by the query is usually a Table Valued Function (TVF), that processes the input and gives a resultset as output.

# Introducing the APPLY operator

---

APPLY is an alternative to JOINS and subqueries that goes beyond the limitations of these two operators.

It was introduced in SQL Server 2005.

The basic purpose of APPLY is to join a table and a function using a *divide and conquer* approach almost like a procedural language.

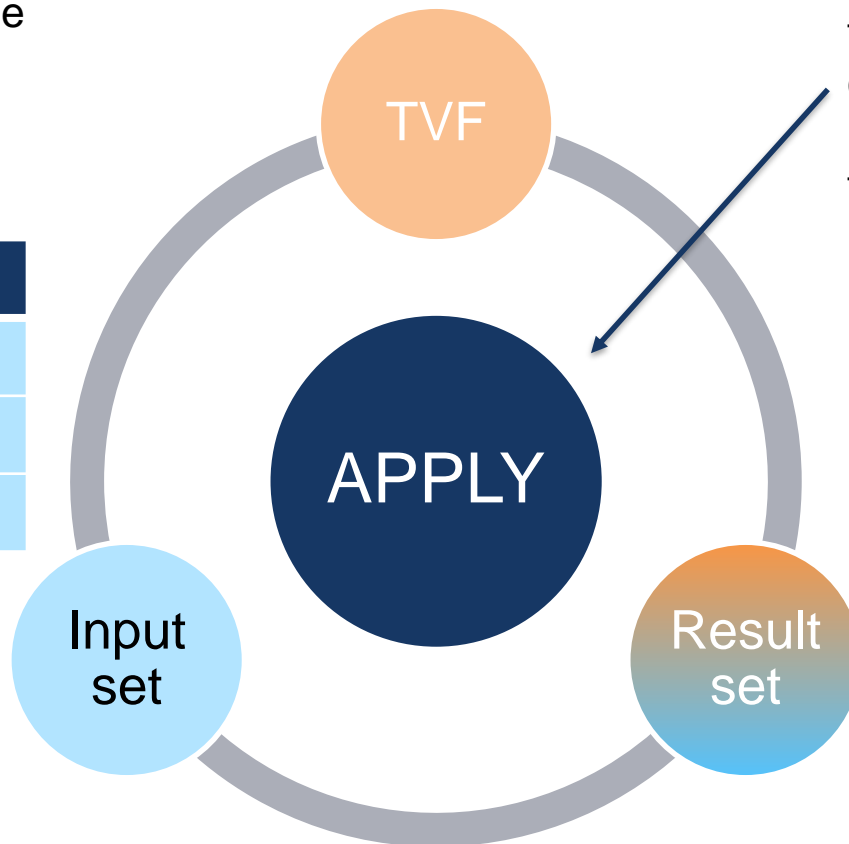
Using APPLY you can simplify expressions that would require a complex SQL standard code to be solved, or can apply a complex business logic to a table in an easy and straightforward way.

# How does APPLY works?

On the left side of the operator there's an input set. Most commonly a table.

CustomerID	Customer
1AC	Ambrosi Claudio
2TL	Tavani Luciano
3FG	Ferrari Giuseppe

Function that retrieves the last 3 orders for every customer.



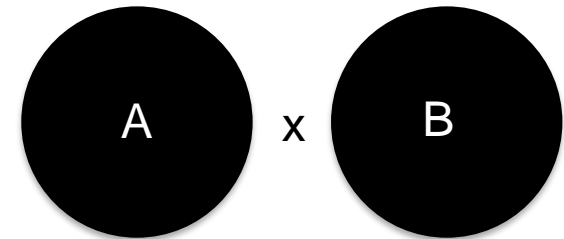
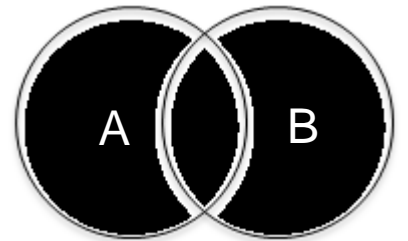
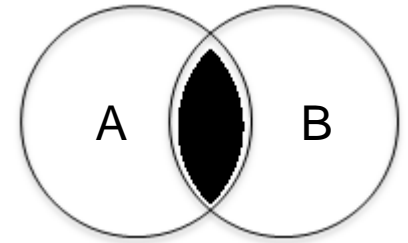
APPLY operator receives the set of input rows and applies it to the table-valued function. APPLY calls the function once for each row from the input and returns a table on each call.

CustID	Customer	LastOrders
1AC	Ambrosi Claudio	24/06/2016 350.000
1AC	Ambrosi Claudio	14/10/2015 67.000
2TL	Tavani Luciano	04/03/2016 120.000
3FG	Ferrari Giuseppe	25/07/2016 43.000
3FG	Ferrari Giuseppe	29/02/2016 87.000
3FG	Ferrari Giuseppe	15/11/2015 144.000



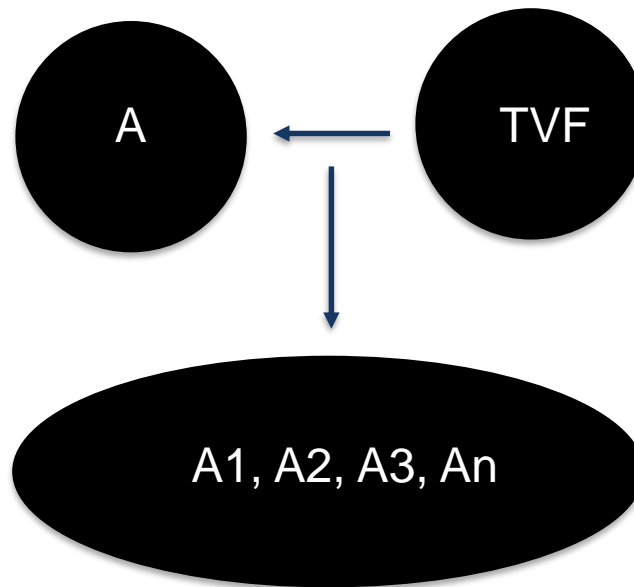
# APPLY vs Join

- INNER JOIN
  - Two input sets. Result: only matching combinations
- OUTER JOIN
  - Two input sets. Result: full rows from one or both datasets depending on the type of join (FULL, LEFT, RIGHT)
- CROSS JOIN
  - Two input sets. Result: all possible combinations of all values in the rows



# APPLY vs Join

- APPLY
  - Left input set, right set evaluated per left row using correlation  
Result: unified results of evaluations



# APPLY vs. JOIN

---

The operator provides two variants: CROSS APPLY and OUTER APPLY.

**CROSS APPLY** can be assimilated to an **INNER JOIN**: it doesn't retrieve any row in the resultset where the function fails to produce a result.

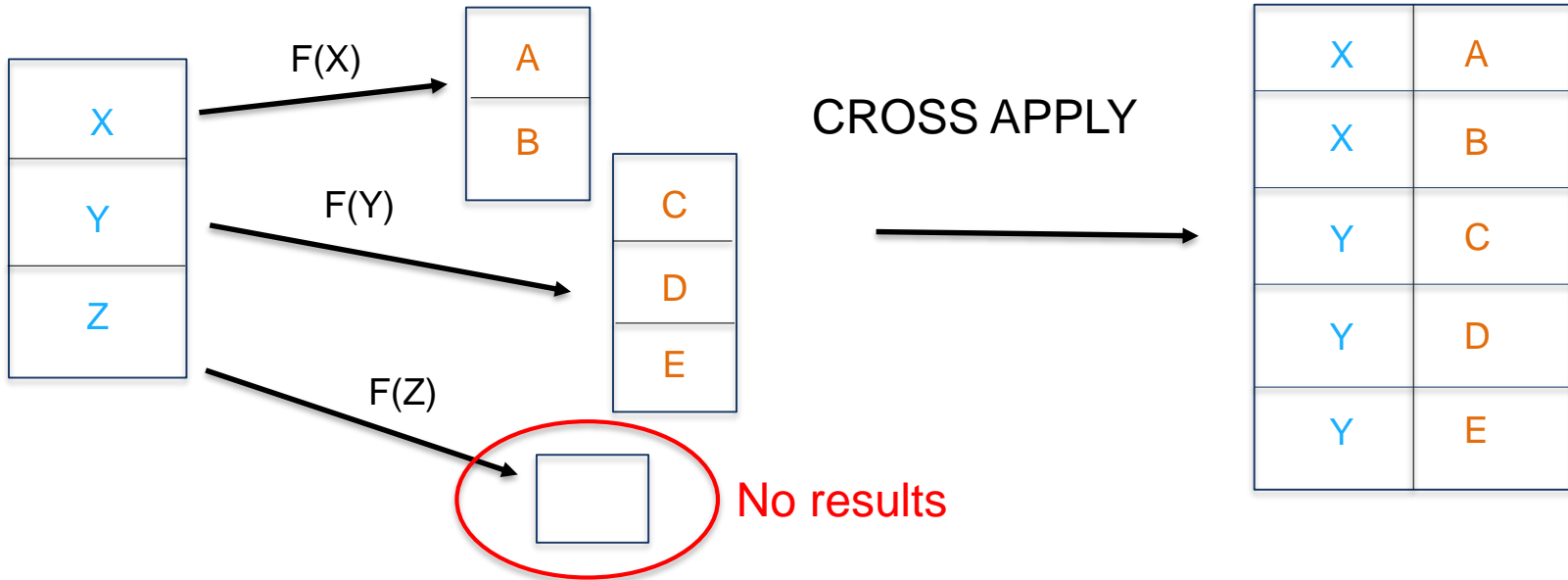
**OUTER APPLY** can be assimilated to an **OUTER JOIN** (LEFT, RIGHT, FULL): where the function does not return a row, the input row is still included in the final output, with NULLs in the columns provided by the function.

# CROSS APPLY

Apply right set per left row using correlations. Left rows discarded if right side is empty

**Left table**

**Function (F) acts as a right table**



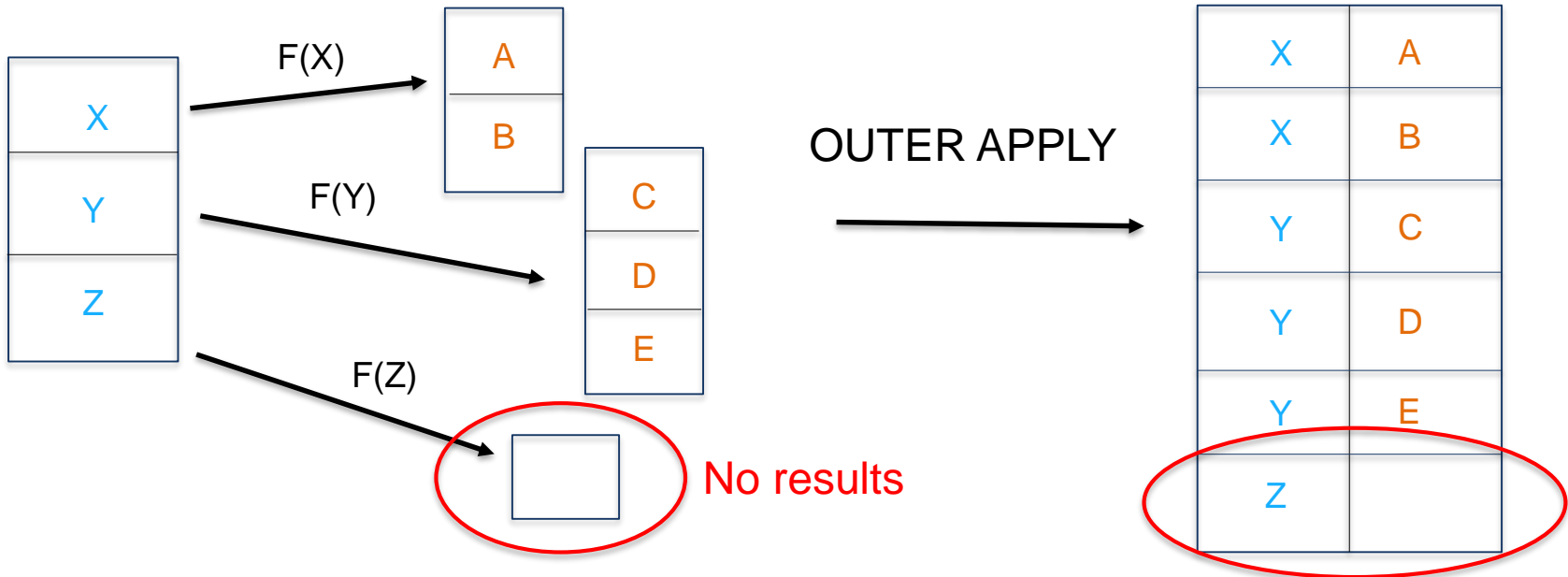
```
SELECT C.custid, C.companyname, O.orderid, O.orderdate, O.empid
FROM Sales.Customers AS C
CROSS APPLY dbo.GetTopOrders(C.custid, 3) AS O;
```

# OUTER APPLY

Apply right set per left row using correlations. Left rows are returned even though right side is empty

**Left table**

**Function (F) acts as a right table**



```
SELECT C.custid, C.companyname, O.orderid, O.orderdate, O.empid
FROM Sales.Customers AS C
CROSS APPLY dbo.GetTopOrders(C.custid, 3) AS O;
```

# Quick reminder: user-defined functions

---

In SQL Server there are the following user-defined functions:

- ~~Scalar functions~~: return a single value Scalar functions cannot be invoked with APPLY
- **Inline table-valued functions**: return the result of a single SELECT statement
- **Multi-statement table-valued functions**: use complex code, to populate and return a table variable
- **CLR table-valued functions**: written in .NET language and embedded inside SQL Server.

# In-line TVF functions

---

**HINT:** try to always write your functions in the form of **in-line Table Valued Functions**.

An in-line TVF consists of a single SELECT statement. You can think of an iTVF as a view with parameters parameters. And mostly, the query optimizer consider the function like a view and is capable of expanding the code and apply its full range of optimization for best performances.

# In-line TVF functions

## --Transact-SQL Inline Table-Valued Function Syntax

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( [ select_stmt ] ) ]
[ ; ]
```



# DEMO

---

## 1. Calculate time differences

- Age and seniority for every employee
- Average period of stay in a department

# A classic problem: string split

---

String split solution is a common request in all forums and support sites. You can find a bunch of documentation on-line. Suppose you receive a string in this format:

Molise	Campobasso;Isernia
--------	--------------------

And want to translate the row into columns with a function:

Molise	Campobasso
Molise	Isernia

You can do it with a SQL user-defined function.

And remember ... SQL Server 2016 introduced the function **STRING\_SPLIT**. You don't need to built your own UDF anymore!

# DEMO

---

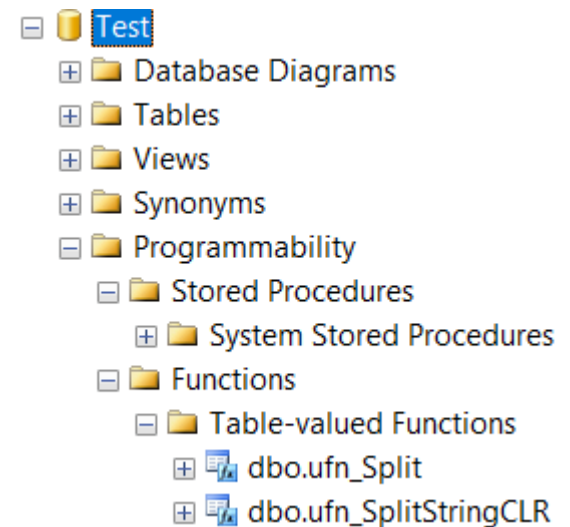
## 2. String split with T-SQL user-defined function

# String split with CLR

String splitting can even be attained by writing a CLR (Common Language Runtime) function in .NET and then applying it to the table.

You can use CLR functions to accomplish tasks which are not possible by T-SQL or can use lots of resources. CLR can be usually implemented where there are operations or methods which can be complicated for T-SQL.

Once you've written your function, you can use it to a table with APPLY operator.



# DEMO

---

## 3a. String split with CLR function

# DEMO


---

## 3b. Retrieve external files info

# Browsing by hierarchy

EmployeeID

ManagerID



EmployeeKey	ParentEmployeeKey
1	18
2	7
3	14
4	3
5	3
6	267
7	112
8	112
9	23
10	189
11	3
12	189
13	3
14	112

There are many ways for browsing an employees hierarchy in T-SQL.

For example with a **recursive CTE**.

Or with a **recursive TVF**, which calls itself by a **CROSS APPLY**.

# DEMO

---

## 4. Hierarchy



# Top N of ...

---

Another classic problem is to retrieve the top N rows of a table.

This task can't be achieved with a query or a subquery.

You can get it with a function and APPLY operator.

# DEMO

---

## 5. Top N of ...

# Unpivot

Take a look at the table with attendance data. Every user is registered for single week, 5 days in a row.

We want to calculate the total attendance days for a single user for a single session.

	idTable	SessionId	UserNa...	WeekOfY...	SessionStartD...	SessionEndD...	Day1	Day2	Day3	Day4	Day5
242	4151	4843812D-8EE9-4A45-B608-0036D1A92287	10329	10	2016-02-29	2016-03-04	Attended	Attended	Attended	Attended	Attended
243	3627	4843812D-8EE9-4A45-B608-0036D1A92287	10329	6	2016-02-01	2016-02-05	Attended	Attended	Attended	Attended	Attended
244	9973	7AFA2617-F80B-48CD-8DBB-173A6F5764B3	10329	38	2016-09-12	2016-09-16	Attended	Attended	Attended	Attended	Attended
245	11320	7AFA2617-F80B-48CD-8DBB-173A6F5764B3	10329	42	2016-10-10	2016-10-14	Attended	Attended	Attended	Attended	Attended
246	11256	7AFA2617-F80B-48CD-8DBB-173A6F5764B3	10329	46	2016-11-07	2016-11-11	Attended	Attended	Attended	Attended	Attended
247	10758	A129FBF9-80F9-4960-BE84-883B00AEA607	10329	43	2016-10-17	2016-10-20	Attended	Attended	Attended	Attended	
248	86	147432C9-0D0F-4C7A-AFC2-8E67F147F6D4	10330	24	2016-06-06	2016-06-07	Attended	Attended			
249	14024	FD337AF7-5BEB-4A23-A776-1CFA2DC6D2C4	10330	6	2016-02-03	2016-02-05	Attended	Attended	Attended		
250	14025	FD337AF7-5BEB-4A23-A776-1CFA2DC6D2C4	10331	9	2016-02-24	2016-02-26	Attended	Attended	Attended		
251	1307	2760161F-A815-41C9-A967-AE414CE51CBA	10334	24	2017-06-16	2017-06-17	Attended	Attended			
252	1308	2760161F-A815-41C9-A967-AE414CE51CBA	10334	44	2016-10-28	2016-10-29	Attended	Attended	Attended	Attended	
253	915	2760161F-A815-41C9-A967-AE414CE51CBA	10334	10	2017-03-09	2017-03-12	Attended	Attended			
254	916	2760161F-A815-41C9-A967-AE414CE51CBA	10334	13	2017-03-31	2017-04-01	Attended	Attended	Attended	Attended	
255	1462	2760161F-A815-41C9-A967-AE414CE51CBA	10334	3	2017-01-20	2017-01-21	Attended	Attended			
256	1463	2760161F-A815-41C9-A967-AE414CE51CBA	10334	6	2017-02-10	2017-02-11	Attended	Attended	Attended	Attended	

To achieve the result, we can unpivot the data and sum by column. The query can be wrapped into a function.

# Unpivot with table value constructor

Another way to unpivot a table is to use the **Table Value Constructor**. It consists of the VALUES clause in form of a table to specify a set of row value expressions to be constructed into a table. The T-SQL table value constructor allows multiple rows of data to be specified in a single DML statement.

Can be used instead of a table in a query.

```
VALUES ( <row value expression list> ) [ ,...n ]
```

```
<row value expression list> ::=  
    {<row value expression> } [ ,...n ]
```

```
<row value expression> ::=  
    { DEFAULT | NULL | expression }
```

# Table Value Constructor

---

APPLY and VALUES can be combined together to allow correlations.

They can be used for:

- Reuse of column aliases throughout the query
- Solve complex unpivoting tasks

```
SELECT custid, salesyear, qty, val
FROM dbo.Sales
CROSS APPLY
    ( VALUES(2012, qty2012, val2012),
            (2013, qty2013, val2013),
            (2014, qty2014, val2014) ) AS A(salesyear, qty, val);
```

# DEMO

---

## 6. Unpivot

# Playing with dates

In our table, attendance is expressed as period between a start and a end.

Say we want to see for every attendee the full list of days he was in the classroom.

We can use both CROSS JOIN or CROSS APPLY methods to achieve the goal.

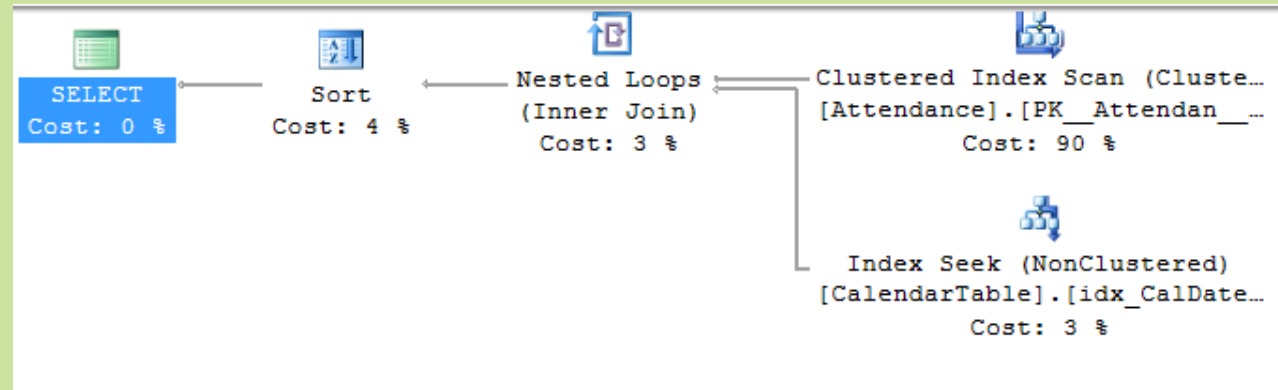
```
--  
SELECT * FROM dbo.Attendance WHERE UserName = N'10001'
```

idTa...	SessionId	UserNa...	WeekOfY...	SessionStartD...	SessionEndD...	Day1	Day2	Day3	Day4	Day5
2386	310974DC-247C-4AFB-B427-881185992974	10001	22	2016-05-23	2016-05-24	Attended	Attended			
5179	55925F09-E0B2-412B-969C-E5CE9699BCD0	10001	15	2016-04-04	2016-04-05	Attended	Attended			
8940	7ADEAA4D-6740-4DD9-88FD-9327F568ECE2	10001	11	2016-03-09	2016-03-09	Attended				
12144	E5284417-1D24-4151-837A-8BD791CDEB15	10001	39	2016-09-19	2016-09-23	Attended	Attended	Attended	Attended	Attended
13443	E5284417-1D24-4151-837A-8BD791CDEB15	10001	47	2016-11-14	2016-11-15	Attended	Attended	Attended	Attended	Attended
13557	E5284417-1D24-4151-837A-8BD791CDEB15	10001	43	2016-10-17	2016-10-19	Attended	Attended	Attended	Attended	Attended

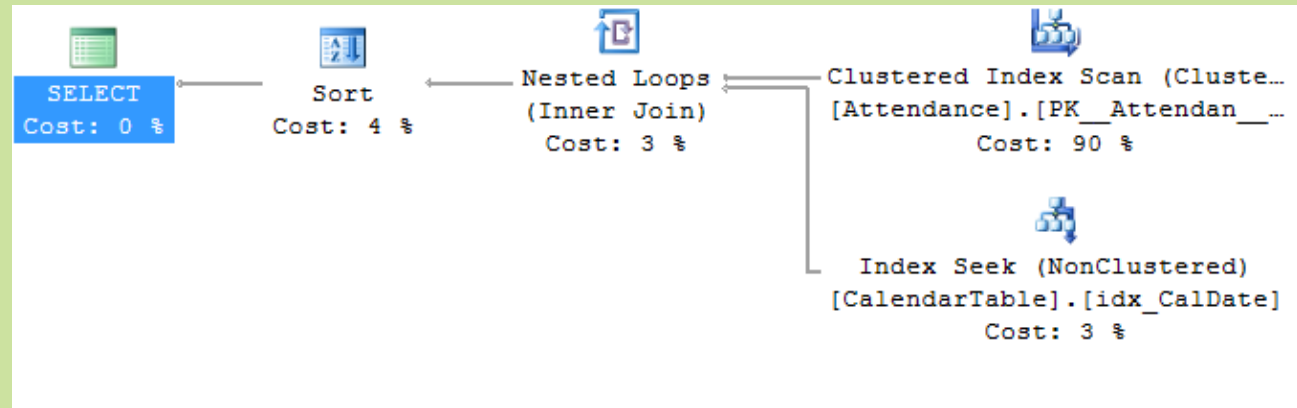
# DEMO

## 7. Date ranges

CROSS JOIN



CROSS APPLY





# Using DMF for troubleshooting

---

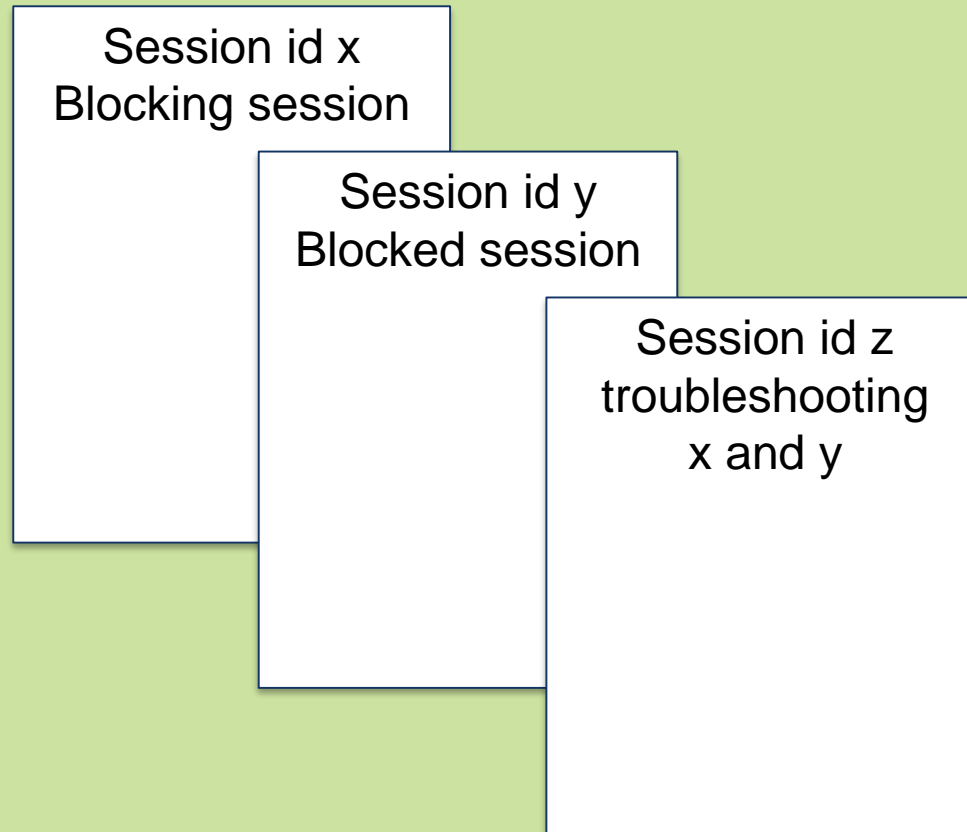
You can use Dynamic Management Views (DMV) and Dynamic Management Functions (DMF) to return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.

For example, in case of blocking you can apply the function `sys.dm_exec_sql_text` to the DMV `sys.dm_exec_connections` to get the SQL code executed at the time of blocking.

# DEMO

---

## 8. DMV e DMF



# OFFSET – FETCH filter

---

Retrieve a fixed number of rows from sorted output. Introduced in SQL Server 2012.

There are two arguments of the ORDER BY clause that let you retrieve a fixed number of rows:

- **OFFSET <N> ROWS**, which you use to specify the line number from which to start retrieving results
- **FETCH NEXT <N> ROWS ONLY**, which you use to specify how many lines to retrieve

Used to format output on a fixed form, for example paging in a web page. The number of rows retrieved can be set through a variable.

# OFFSET – FETCH filter

```
SELECT  
  FirstName + ', ' + LastName  
FROM  dbo.DimEmployee  
ORDER BY FirstName  
OFFSET 10 ROWS  
FETCH NEXT 5 ROWS ONLY;
```

OFFSET



Employee	
1	A. Scott, Wright
2	Alan, Brewer
3	Alejandro, McGuel
4	Alex, Nayberg
5	Alice, Ciccu
6	Amy, Alberts
7	Andreas, Berglund
8	Andrew, Hill
9	Andrew, Cencini
10	Andy, Ruth
11	Angela, Barbariol
12	Anibal, Sousa
13	Annette, Hill
14	Annik, Stahl
15	Arvind, Rao
16	Ashvini, Sharma
17	Barbara, Decker
18	Barbara, Moreland
19	Baris, Cetinok
20	Barry, Johnson
21	Belinda, Newman



FETCH

# DEMO

---

## 9. OFFSET FETCH

November 26°, 2016



#566 | PARMA 2016

# Expression aliases

---

Take a look at the following query:

```
SELECT  
    YEAR(OrderDate) AS OrderYear,  
    MONTH(OrderDate) AS OrderMonth,  
    SalesAmount  
FROM dbo.FactResellerSales  
WHERE OrderYear = 2013
```

Does it work?

Msg 207, Level 16, State 1, Line 11  
Invalid column name 'OrderYear'.

# Expression aliases

And this query?

```
SELECT
```

```
    YEAR(OrderDate) AS OrderYear,
```

```
    MONTH(OrderDate) AS OrderMonth,
```

```
    SalesAmount
```

```
FROM dbo.FactResellerSales
```

```
WHERE OrderDate > '20121231'
```

```
ORDER BY OrderMonth
```

Not allowed alias

Allowed alias

Why?

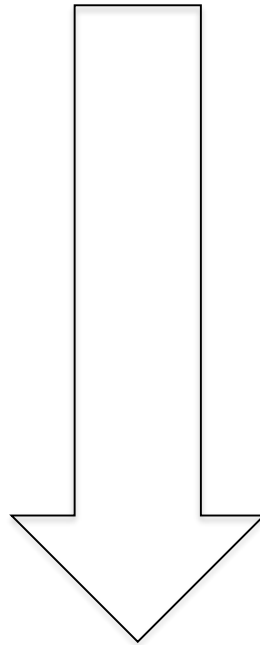
Results		
OrderYear	OrderMonth	SalesAmount
2013	1	2753,988
2013	1	63,90
2013	1	125,982
2013	1	226,758
2013	1	338,994

# Expression aliases

---

It depends on the order the SQL clauses are processed in a query:

1. **FROM**
2. **WHERE**
3. **GROUP BY**
4. **HAVING**
5. **SELECT**
6. **ORDER BY**
7. **TOP / OFFSET-FETCH**



WHERE is processed before SELECT, so alias declared in SELECT can't be recognized by WHERE



# DEMO

---

## 10. Expression aliases

# Shredding XML

---

The process of converting XML to relational tables is called ***shredding***.

You can use the *nodes* method of the xml data type and then apply another method to the result (*value()*, *query()*, *exist()*, *modify()* ).

All of these methods accept an XQuery expression as a parameter.

CROSS APPLY can apply *nodes()* to each row in an input table to return information in a relational format.

# DEMO

---

## 11. XML Shredding

November 26°, 2016

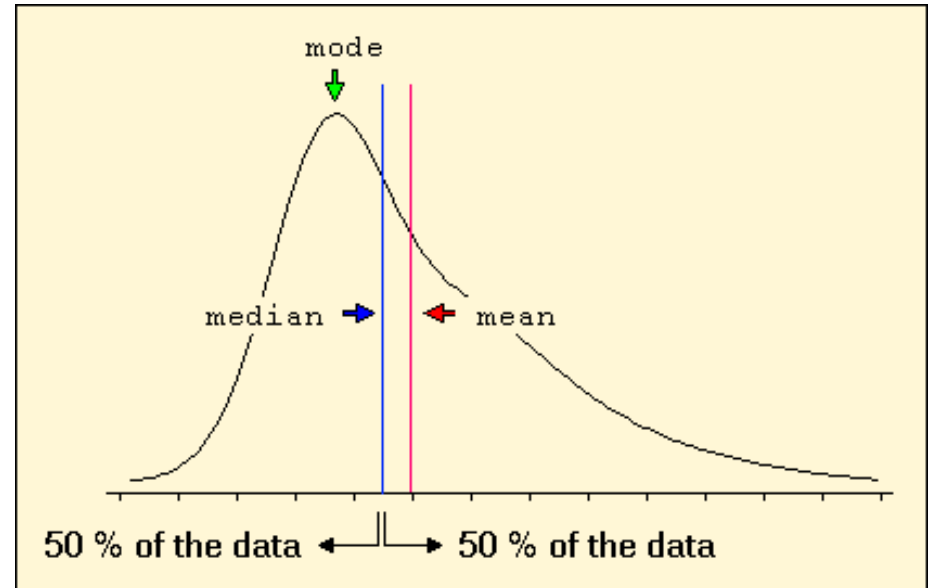


#566 | PARMA 2016

# Median value

Definition from Wikipedia

*“The median is the numerical value separating the higher half of a data sample, a population, or a probability distribution, from the lower half.”*



Median calculation isn't natively supported in SQL Server. In forums and articles we can find many examples in T-SQL. Most of which use CROSS APPLY.

# Median value

If there are an even number of item, the median is then usually defined to be the mean (average) of the two middle values. For example in the dataset:

1, 2, 3, 4, 5, 6, 8, 9

The median is the mean of the middle two numbers: this is  $(4 + 5) \div 2$ , which is 4.5.

If there is an odd number of numbers, the middle one is picked. For example, consider the set of numbers:

1, 3, 3, 6, 7, 8, 9

This set contains seven numbers. The median is the fourth of them, which is 6.

1, 3, 3, **6**, 7, 8, 9

Median = **6**

1, 2, 3, **4**, **5**, 6, 8, 9

Median =  $(4 + 5) \div 2$   
= **4.5**

Finding the median in sets of data with an odd and even number of values.

# DEMO

---

## 12. Median

November 26°, 2016



#566 | PARMA 2016

# Implicit APPLY

As Itzik Ben-Gan showed, with the introduction of the APPLY operator you can refer to a table UDF in a subquery and then pass columns from outer table as inputs (correlations).

He called this feature **implicit APPLY**.

```
SELECT C.custid, C.companyname,  
       (SELECT COUNT(DISTINCT empid)  
        FROM dbo.GetTopOrders(C.custid, 3) AS 0) AS numemps  
FROM Sales.Customers AS C;
```

Call the function in the subquery,  
passing a column from the outer table

Outer table

# Lateral Derived Table

---

Curiously, standard SQL has a similar feature called ***lateral derived table***.

The standard parallel to CROSS APPLY is to use CROSS JOIN, but to prefix the derived table with the keyword LATERAL to indicate that it's a lateral derived table. The operator visits the left side first, and applies the lateral derived table to each left row.

T-SQL doesn't support lateral derived tables since it has its own solution using APPLY, but, if it did, the code would have looked like this:

```
CROSS JOIN LATERAL  
LEFT OUTER JOIN LATERAL
```

Don't run it since it's not supported in SQL Server



# Conclusions

---

Using the APPLY operator, you can apply the result of a function to an input table.

This feature allows you to perform a wide range of operation difficult to execute with standard SQL.

APPLY comes with two formats:

CROSS APPLY analog to an INNER JOIN

OUTER APPLY analog to an OUTER JOIN

# Resources

---

Microsoft Virtual Academy – Boost your T-SQL with the APPLY operator

<https://mva.microsoft.com/en-US/training-courses/boost-your-tsql-with-the-apply-operator-8270>

Paul White - Understanding and Using APPLY (Part 1)

<http://www.sqlservercentral.com/articles/APPLY/69953/>

Paul White - Understanding and Using APPLY (Part 2)

<http://www.sqlservercentral.com/articles/APPLY/69954/>

Robert Sheldon - SQL Server APPLY basics

<https://www.simple-talk.com/sql/t-sql-programming/sql-server-apply-basics/>

Itzik Ben-Gan - Logical Query Processing: The FROM Clause and APPLY

<http://m.sqlmag.com/sql-server/logical-query-processing-clause-and-apply>

# Resources

---

Dwain Camps. Calculating the Median Value within a Partitioned Set Using T-SQL

<https://www.simple-talk.com/sql/t-sql-programming/calculating-the-median-value-within-a-partitioned-set-using-t-sql/>

Aaron Bertrand. What is the fastest way to calculate the median?

<https://sqlperformance.com/2012/08/t-sql-queries/median>

Aaron Betrand. Split strings the right way – or the next best way

<https://sqlperformance.com/2012/07/t-sql-queries/split-strings>

Adam Machanic. SQLCLR String Splitting Part 2: Even Faster, Even More Scalable

[http://sqlblog.com/blogs/adam\\_machanic/archive/2009/04/28/sqlclr-string-splitting-part-2-even-faster-even-more-scalable.aspx](http://sqlblog.com/blogs/adam_machanic/archive/2009/04/28/sqlclr-string-splitting-part-2-even-faster-even-more-scalable.aspx)

Seth Delconte. Manipulating XML Data in SQL Server

<https://www.simple-talk.com/sql/database-administration/manipulating-xml-data-in-sql-server/>

# Resources

---

Itzik Ben-Gan. Logical Query Processing: What It Is And What It Means to You

<http://sqlmag.com/sql-server/logical-query-processing-what-it-and-what-it-means-you>

Itzik Ben-Gan. SQL Server 2005's Apply, Part1

<http://sqlmag.com/sql-server-2005/sql-server-2005s-apply-part-1>

# Q&A

---

Questions?

November 26°, 2016



#566 | PARMA 2016



**#sqlsatParma**  
**#sqlsat566**

<http://speakerscore.com/>

THANKS!

November 26°, 2016



#566 | PARMA 2016