



APPLY

- **Cross Apply**  (no null values) and **Outer Apply**  (include null values)
- Apply is like join, but one side of the apply is a **table valued expression**, in which a value (usually a column) in each row of the other side of the apply is passed into that expression.
- <https://www.mssqltips.com/sqlservertip/1958/sql-server-cross-apply-and-outer-apply/>

```
SELECT DeptID, DeptName, DeptMgrID, EmpID, EmpLastName, EmpSalary
FROM Departments d
CROSS APPLY dbo.GetReports(d.DeptMgrID) ;
```

```
SELECT DeptID, DeptName, DeptMgrID, EmpID, EmpLastName, EmpSalary
FROM Departments d
OUTER APPLY dbo.GetReports(d.DeptMgrID) ;
```

Pivot

- Pivot (aggregation) rotates a table valued expression 90 deg. and turn values in one column into multiple columns. Unpivot is the opposite operation

```
SELECT DaysToManufacture, AVG(StandardCost) AS AverageCost
FROM Production.Product
GROUP BY DaysToManufacture;
```

DaysToManufacture	AverageCost
0	5.0885
1	223.88
2	359.1082
4	949.4105

Pivot

```
-- Pivot table with one row and five columns
SELECT 'AverageCost' AS Cost_Sorted_By_Production_Days,
[0], [1], [2], [3], [4]
FROM
(SELECT DaysToManufacture, StandardCost
  FROM Production.Product) AS SourceTable
PIVOT
(
  AVG(StandardCost)
  FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;
```

Cost_Sorted_By_Production_Days	0	1	2	3	4
AverageCost	5.0885	223.88	359.1082	NULL	949.4105

Pivot

```
USE AdventureWorks2014;
GO
SELECT VendorID, [250] AS Emp1, [251] AS Emp2, [256] AS Emp3, [257] AS Emp4, [260] AS Emp5
FROM
(SELECT PurchaseOrderID, EmployeeID, VendorID
FROM Purchasing.PurchaseOrderHeader) p
PIVOT
(
COUNT (PurchaseOrderID)
FOR EmployeeID IN
( [250], [251], [256], [257], [260] )
) AS pvt
ORDER BY pvt.VendorID;
```

VendorID	Emp1	Emp2	Emp3	Emp4	Emp5
1492	2	5	4	4	4
1494	2	5	4	5	4
1496	2	4	4	5	5
1498	2	5	4	4	4
1500	3	4	4	5	4

UnPivot

- Unpivot removes NULL values in the pivoted table and it **can't reverse the aggregation**
 - here we only get the **total** orders for each employee from each vendor
- you have to **list all possible** values/cols from your pivoted table

```
-- Unpivot the table.
SELECT VendorID, Employee, Orders
FROM
    (SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
     FROM pvt) p
UNPIVOT
    (Orders FOR Employee IN
     (Emp1, Emp2, Emp3, Emp4, Emp5))
AS unpvt;
GO
```

VendorID	Employee	Orders
1	Emp1	4
1	Emp2	3
1	Emp3	5
1	Emp4	4
1	Emp5	4
2	Emp1	4
2	Emp2	1

Cube (T-SQL)

- Cube is a subclause of Group By. Used to generate multiple grouping sets. It creates “group by”s for all possible combinations of columns.
- For cube(a,b), it generates (a,b), (a,null), (null,b) and (null,null)
- <https://www.sqlservertutorial.net/sql-server-basics/sql-server-cube/>
- Example in next slide

Cube (T-SQL)

```
select name,deptid, sum(sal)
from emp
group by cube(name,deptid)
```

```
select name,deptid,sal from emp
```

	name	deptid	sal
1	tom	1	1000
2	jack	1	9999
3	jerry	1	8000
4	anna	2	10...
5	zoey	2	10...
6	tiffany	2	4000
7	tom	1	1000
8	dick	3	9999
9	jeff	3	10...
10	bill	3	8000

	name	deptid	(No column name)
1	jack	1	9999
2	jerry	1	8000
3	tom	1	2000
4	NULL	1	19999
5	anna	2	10000
6	tiffany	2	4000
7	zoey	2	10000
8	NULL	2	24000
9	bill	3	8000
10	dick	3	9999
11	jeff	3	10000
12	NULL	3	27999
13	NULL	NULL	71998
14	anna	NULL	10000
15	bill	NULL	8000
16	dick	NULL	9999
17	jack	NULL	9999
18	jeff	NULL	10000
19	jerry	NULL	8000
20	tiffany	NULL	4000
21	tom	NULL	2000
22	zoey	NULL	10000

Rollup

- the ROLLUP operator is used to calculate sub-totals and grand totals for a set of columns passed to the “GROUP BY ROLLUP” clause
- ROLLUP creates a hierarchy based on the columns passed in
- in the below example, rollup will create group set according to
 - country + region
 - country only
 - all countries

```
SELECT Country, Region, SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
```


Cube vs Rollup

- CUBE will produce all possible combinations
 - country + region
 - country only
 - all country + all regions
 - **region only -> 2nd level in the hierarchy, skipped in ROLLUP**
- in the below example, ROLLUP will create group set according to
 - country + region
 - country only
 - all countries + all regions
- <https://www.sqlservercentral.com/articles/the-difference-between-rollup-and-cube>

```
SELECT Country, Region, SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
```

Temp Tables

- Stored in `tempdb`, physical table, with all the features of normal tables.
- Declare using `#tableName` or `##tableName`
- Create temp tables with
 - `CREATE TABLE`
 - `SELECT INTO`
- Suitable for temporarily storing a large set of data.

```
SELECT
    product_name,
    list_price
INTO #trek_products --- temporary table
FROM
    production.products
WHERE
    brand_id = 9;
```

Temp Tables

- Local temp tables start with #
 - only useable within the current session/connection
 - local temp tables are automatically dropped when you close the connection that created it.
- Global temp tables start with ##
 - useable across all connections
 - global temp tables are dropped once the connection that created it closed and the queries against this table from other connections completes.
- You can also drop temp tables manually with DROP TABLE #xxxxx

Local Variable

- an object that can hold a single data value of a specific type, typically used:
 - As a counter, often in loops
 - To hold a data value to be tested by a control-of-flow statement.
 - To save a data value to be returned by a stored procedure or function.

Local Variable

- Declare a variable with the **DECLARE** keyword
- variables can **only be referenced in the batch/procedure it's declared**
- <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/variables-transact-sql?view=sql-server-ver15>
- Examples on next page

Local Variable

```
DECLARE @LastName NVARCHAR(30), @FirstName NVARCHAR(20), @StateProvince NCHAR(2);
```

```
USE AdventureWorks2014;
GO
DECLARE @EmpIDVariable INT;

SELECT @EmpIDVariable = MAX(EmployeeID)
FROM HumanResources.Employee;
GO
```

*If a SELECT statement returns more than one row and the variable references a non-scalar expression, the variable is set to the value returned for the expression in the last row of the result set.

```
-- Create the table.
CREATE TABLE TestTable (cola INT, colb CHAR(3));
GO
SET NOCOUNT ON;
GO
-- Declare the variable to be used.
DECLARE @MyCounter INT;

-- Initialize the variable.
SET @MyCounter = 0;

-- Test the variable to see if the loop is finished.
WHILE (@MyCounter < 26)
BEGIN;
    -- Insert a row into the table.
    INSERT INTO TestTable VALUES
        -- Use the variable to provide the integer value
        -- for cola. Also use it to generate a unique letter
        -- for each row. Use the ASCII function to get the
        -- integer value of 'a'. Add @MyCounter. Use CHAR to
        -- convert the sum back to the character @MyCounter
        -- characters after 'a'.
        (@MyCounter,
        CHAR( ( @MyCounter + ASCII('a') ) ));
    -- Increment the variable to count this iteration
    -- of the loop.
    SET @MyCounter = @MyCounter + 1;
END;
GO
SET NOCOUNT OFF;
GO
-- View the data.
SELECT cola, colb
FROM TestTable;
GO
DROP TABLE TestTable;
GO
```

Table Variable

- a special type of local variable that helps to store data temporarily
 - insert, update, delete are all allowed just like on a regular table
 - PK, unique, not null, check constraints are all supported, but not FK
 - truncate, alter, explicit indexes are not allowed
 - indexes can be created implicitly through PK and Unique constraints
 - rollback has no effect on table variables

```
DECLARE @LOCAL_TABLEVARIABLE TABLE
(column_1 DATATYPE,
 column_2 DATATYPE,
 column_N DATATYPE
)
```

Temporal Table

- A table maintains its historical information automatically.
- It has a data table and a history table. Two columns of datetime2 and a PERIOD column will need to be added to those tables.
- Based on command, rows for a certain time will be returned automatically.
- Use “As Of”, “FROM TO”, “Between And”, “Contained In” and “All” to query the data.
- Select * will not return hidden columns.

```
SELECT * FROM Employee
FOR SYSTEM_TIME
  BETWEEN '2014-01-01 00:00:00.0000000' AND '2015-01-01 00:00:00.0000000'
  WHERE EmployeeID = 1000 ORDER BY ValidFrom;
```

```
/*State of entire table AS OF specific date in the past*/
SELECT [DeptID], [DeptName], [SysStartTime],[SysEndTime]
FROM [dbo].[Department]
FOR SYSTEM_TIME AS OF '2015-09-01 T10:00:00.7230011' ;
```



Temporal Table

- ```
CREATE TABLE Department
(
 DeptID INT NOT NULL PRIMARY KEY CLUSTERED
 , DeptName VARCHAR(50) NOT NULL
 , ManagerID INT NULL
 , ParentDeptID INT NULL
 , ValidFrom DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL
 , ValidTo DATETIME2 GENERATED ALWAYS AS ROW END NOT NULL
 , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON);
```

# Temp Table vs Table Variable

| Temporary Table                                                                                                                                                                                                                                                                                                                        | Table Variable                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Temporary tables can be used in Stored Procedures, Triggers and Batches but <b>not in user defined functions</b>                                                                                                                                                                                                                       | Table variables <b>can</b> be used in <b>user defined functions, stored procedures, and batches</b>                                                                                   |
| <b>Local</b> temporary tables are temporary tables that are available only to the session that created them. <b>Global</b> temporary tables are temporary tables that are <b>available to all sessions and all the users</b> .                                                                                                         | Its <b>scope</b> is in the stored procedure, user defined function or batch <b>where it is declared</b> like any local variable we create with a DECLARE statement.                   |
| <b>Local</b> temporary tables are automatically destroyed at the end of the procedure or session that created them. <b>Global</b> temporary tables are dropped automatically when the last session using the temporary table has completed<br>We can also drop temporary tables explicitly using drop command similar to normal table. | Table variables are <b>automatically cleaned up</b> at the end of the user defined function, stored procedure, or batch in which they are defined.                                    |
| Temporary table name can be of maximum 116 characters                                                                                                                                                                                                                                                                                  | Table variable name can be of maximum 128 characters                                                                                                                                  |
| PRIMARY KEY, UNIQUE, NULL, CHECK etc can be implemented at the time of creating temporary tables using CREATE TABLE statement or <b>can be added after the table has been created. FOREIGN KEY not allowed.</b>                                                                                                                        | PRIMARY KEY, UNIQUE, DEFAULT values, NULL, CHECK can be added, but they <b>must be incorporated with the creation of the table</b> in the DECLARE statement. FOREIGN KEY not allowed. |

# Temp Table vs Table Variable

| Temporary Table                                                                                                                                                                    | Table Variable                                                                                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Temporary table supports adding Indexes explicitly <b>even after creation</b> and it can also have the implicit Indexes which are the result of Primary and Unique Key constraint. | Table Variables doesn't <b>allow the explicit addition of Indexes after it is declared</b> , the only means is the implicit indexes which are created as a result of the Primary Key or Unique Key constraint defined at the time of declaring Table Variable. |
| Temporary tables can also be directly created and data can be inserted using <b>Select Into</b> statement without creating a temporary table explicitly.                           | Table variables <b>can't be created using Select Into</b> statement because being a variable <b>it must be declared before use</b>                                                                                                                             |
| The SET IDENTITY_INSERT statement is supported in temporary table                                                                                                                  | The SET IDENTITY_INSERT statement is not supported in table variables                                                                                                                                                                                          |
| We can't return a temporary table from a <b>user-defined function</b>                                                                                                              | We can return a table variable from a user-defined function                                                                                                                                                                                                    |
| Temporary Table can be <b>truncated</b> like normal table                                                                                                                          | Table variables can't be truncated like normal table or temporary tables.                                                                                                                                                                                      |
| <b>Stored Procedures will not recompile for every run</b>                                                                                                                          | When using temp variable in Stored Procedures, the SP <b>will</b> recompile for <b>every subsequent calls</b> .                                                                                                                                                |
| <b>All DML will be rolled back when rolling back a transaction</b>                            | Changes to the data will <b>not</b> be rolled back when rolling back a transaction                                                                                                                                                                             |