향상된 기능의 MSSQL Server 2005 T-SQL

- 작성일: **2005-02-12**
- 작성자: 이재학
- 목 차

  - Data Type(2)
  - Partitioned Function & Table(3)
  - Index(8)
  - Data Manipulation Language(Insert, Update, Delete)(12)
  - Common Table Expressions(CTE)(19)
  - Pivot & UnPivot(22)
  - Apply(24)
  - Windowing Function(27)
  - Try … Catch(29)
  - Top & TableSample(33)

Data Type

기존에는 varchar, nvarchar, varbinary는 8000바이트의 제한을 가졌었다. 그러나 2005 버전에서는 max 키워드를 사용하여 2GB까지 사용할 수 있게 되었다. XML 데이터 타입도 새로 만들어졌고, 역시 2GB까지 사용할 수 있다.

```
Use AdventureWorks
Go

Create Table SalesData.Max_Varchar_Table(
       Max_Varchar Varchar(Max)
,      Max_nVarchar nVarchar(Max)
,      Max_varBinary varBinary(Max)
)
Go

Create Table SalesData.Max_XML_Table(
       Max_XML_Col XML)
Go
```

Partitioned Function & Table

파티션된 테이블은 테이블을 물리적인 다른 공간으로 나누어 저장되고, 읽을 수 있는 기능이다. 이전 버전에서는 이와 같은 기능을 Partitioned View를 통해 구현하였으나 제약사항이 있었다. 테이블이 파티션되면 각각의 파티션은 여러 파일그룹으로 분산시킬 수 있다. 이것은 관리상 성능상 이점이 있다. 멀티프로세서 시스템에서는 더 유리해진다. 오라클에는 이미 구현된 기능이나 일반적으로는 많이 쓰지 않는다. 그러지 않았으면……

$partition 키워드

```
[ database_name. ] $partition.partition_function_name(expression)
```

이 연산자는 파티션 함수와 사용되므로 아래의 파티션 함수에서 예를 보도록 하자.

CREATE PARTITION FUNCTION

```
CREATE PARTITION FUNCTION
partition_function_name(input_parameter_type)
AS RANGE [ LEFT | RIGHT ]
FOR VALUES ( [ boundary_value [ ,...n ] ] )
[ ; ]
```

Beta1까지만 해도 Hash라는 키워드가 있었는데 없어졌다. 정식버전이 출시되기 전까지 어떻게 더 변화할지 모르겠다. 일단 다음의 예를 보자.

```
IF EXISTS (SELECT * FROM sys.partition_functions WHERE name =
'SalesHashPF1')
       DROP PARTITION FUNCTION SalesHashPF1
GO

CREATE PARTITION FUNCTION SalesHashPF1(int)
AS RANGE RIGHT FOR VALUES (10000, 20000, 30000, 40000, 50000)

SELECT $Partition.SalesHashPF1(SalesOrderID) AS Partition,
   COUNT(*) AS [COUNT] FROM SalesOrderHeader
   GROUP BY $partition.SalesHashPF1(SalesOrderID)
   ORDER BY Partition
GO
```

```
Partition   COUNT
----------- -----------
1           3806
2           4054
3           10000
4           10000
5           3659

(5 row(s) affected)
```

지금 생성한 파티션 함수는 0 ~ 9999, 10000 ~ 19999 ... 의 식으로 만단위로 파티션을 나눈 것이다. 그러므로 해쉬를 이용한 것과는 달리 각각의 파티션마다 로우의 개수가 틀리게 나온다.

3번째 파티션의 경우 함수에 RIGHT 연산자를 사용했으므로 오른쪽을 기준으로 10000번으로 자른 것이다. 그러므로 3번째 파티션의 경우 20000부터 시작하고, 만약 LEFT연산자를 사용했을 경우는 결과값이 20001부터 시작함을 알 수 있다.

```
SELECT TOP 1 SalesOrderID FROM SalesOrderHeader
WHERE $partition.SalesHashPF1(SalesOrderID) = 3

SalesOrderID
------------
20000

(1 row(s) affected)

ALTER PARTITION FUNCTION SalesHashPF1 ()
SPLIT RANGE (100)
GO

SELECT $Partition.SalesHashPF1(SalesOrderID) AS Partition,
   COUNT(*) AS [COUNT] FROM SalesOrderHeader
   GROUP BY $partition.SalesHashPF1(SalesOrderID)
   ORDER BY Partition
```

```
GO

Partition   COUNT
----------- -----------
2           3806
3           4054
4           10000
5           10000
6           3659

(5 row(s) affected)
```

파티션된 테이블을 만드는 순서는 다음과 같다.

1. Create Partition Function
2. Create Partition Schema 또는 Create Partition Table

Create Partition Function

```
--파일그룹생성
Use Master
Go
Alter Database AdventureWorks
Add FileGroup Data2
Go

Alter Database AdventureWorks
Add File(
                    Name = File2
             ,      FileName = 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\File2_Data.mdf'
             ,      Size = 50 MB
             ,      MaxSize = 200MB
             ,      FileGrowth = 10 %)
To FileGroup Data2
Go
```

```
--파티션함수 생성
USE AdventureWorks
GO

IF EXISTS (SELECT * FROM sys.partition_functions WHERE name =
'fn_p_func1')
	DROP PARTITION FUNCTION fn_p_func1
GO

Create Partition Function fn_p_func1(Int)
As Range Left For Values(0, 1, 2, 3, 4)
Go

Select Top 5
	$Partition.fn_p_func1(DaysToManufacture) As Parition_Number
From Production.Product
Go

/*결과

Parition_Number
---------------
1
1
2
1
2

(5 row(s) affected)
Table 'Product'. Scan count 1, logical reads 3, physical reads 0,
read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

*/

Select $Partition.fn_p_func1(9)
```

```
/*결과

-----------
6

(1 row(s) affected)

*/
```

Create Partition Schema 또는 Create Partition Table

```
Create Partition Schema p_Schema
As Partition fn_p_func1
To (AdventureWorks_Data, File2)
Go

Create Partition Table p_Table(
       ProductID Int
,      ProductType Int)
On fn_p_func1(ProductType)
Go
```

Index

인덱스도 많이 달라졌다. 눈에 띄는 것은 논클러스터드 인덱스의 리프노드에 인덱스를 구성하는 컬럼이 아닌 다른 컬럼도 저장을 할 수 있는 것이다. (Include 키워드) 이렇게 함으로써 인덱스만을 접근하여 쿼리에 대한 결과를 가져 올 수 있게 할 수 있는 것이다. 클러스터드 인덱스를 한 개만 만들 수 있었지만 2개 이상의 클러스터드 인덱스를 가지는 효과를 가질 수 있다. 그렇지만 인덱스의 크기가 상대적으로 커진다.

```
ALTER INDEX { index_name | ALL }
    ON <object>
    { REBUILD
        [ [ WITH ( <rebuild_index_option> [ ,...n ] ) ]
          | [ PARTITION = partition_number
                [ WITH ( <single_partition_rebuild_index_option>
                        [ ,...n ] )
                ]
            ]
        ]
    | DISABLE
    | REORGANIZE
        [ PARTITION = partition_number ]
        [ WITH ( LOB_COMPACTION = { ON | OFF } ) ]
    | SET ( <set_index_option> [ ,...n ] )
    }
[ ; ]

<object> ::=
{
    [ database_name. [ schema_name ] . | schema_name. ]
    table_or_view_name
}

<rebuild_index_option > ::=
{
    PAD_INDEX  = { ON | OFF }
```

```
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = number_of_processors
}

<single_partition_rebuild_index_option> ::=
{
    SORT_IN_TEMPDB = { ON | OFF }
  | MAXDOP = number_of_processors
}

<set_index_option>::=
{
    ALLOW_ROW_LOCKS= { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
}
```

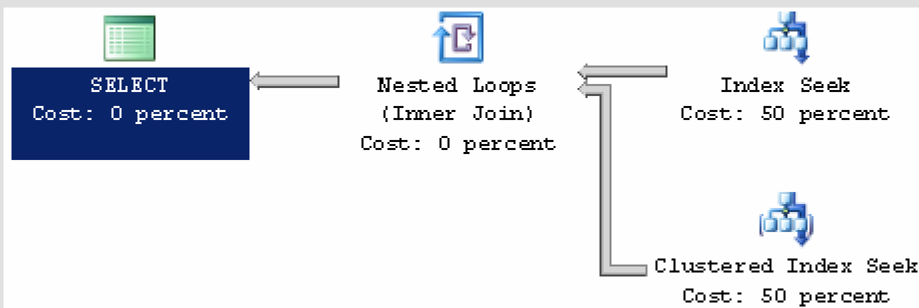
다음은 그 예이다.

```
--인덱스 사용하지 못하도록 Disable
Alter Index idx_CustID
On SalesData.CustID
Disable

--DBCC DBREINDEX 와 같다.
Alter Index idx_CustID
On SalesData.CustID
Rebuild

--DBCC INDEXDEFRAG와 같다.
```

```
Alter Index idx_CustID
On SalesData.CustID
Reorganize

Alter Index idx_CustID
On SalesData.CustID
Set (
        ALLOW_ROW_LOCKS = ON
,       ALLOW_PAGE_LOCKS = ON)
```

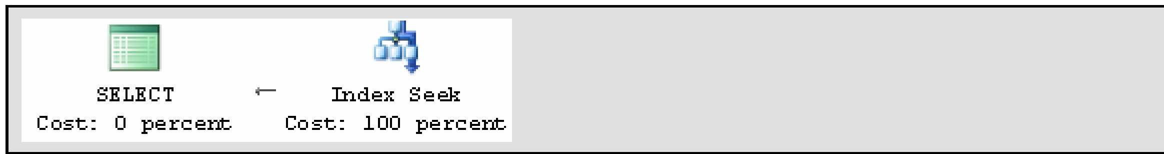
다음은 Include 키워드에 대한 예이다. 눈 여겨 볼 것은 Bookmark Lookup 대신 논클러스터드 인덱스와 클러스터드 인덱스가 Nested Loop Join이 된 것이다. 이 쿼리는 Name 컬럼에 잡힌 인덱스를 사용하여 쿼리한 것이다.

```
Select Name, MakeFlag From Production.Product
Where Name = 'BB Ball Bearing'
```



Name에 잡힌 인덱스를 Drop하고 MakeFlag를 Include하여 인덱스를 다시 만들어 보자.

```
Drop Index Production.Product.AK_Product_Name
Go

Create Index AK_Product_Name
On Production.Product(Name)
Include (MakeFlag)
Go

Select Name, MakeFlag From Production.Product
Where Name = 'BB Ball Bearing'
Go
```


SELECT
Cost: 0 percent
Index Seek
Cost: 100 percent

Data Manipulation Language(Insert, Update, Delete)

데이터조작어(DML)에도 많은 변화가 일어났다. Insert, Delete, Update 중에도 어떤 데이터가 Insert되는지 Delete 되는지 알 수 있다. 이것은 몇 가지 키워드만 알면 된다. common_table_expression(CTE)에 대해서는 뒤에서 다루도록 하겠다.

Insert

달라진 문법은 다음과 같다.

```
[ WITH <common_table_expression> [ ,...n ] ]
INSERT
    [ TOP ( expression ) [ PERCENT ] ]
    [ INTO]
    {
    <object> | rowset_function_limited
      [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
    }
{
    [ ( column_list ) ]
    [ <OUTPUT Clause> ]
    { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] )
    | derived_table
    | execute_statement
    }
}
    | DEFAULT VALUES
[; ]

<object> ::=
{
    [ server_name . database_name . schema_name .
      | database_name .[ schema_name ] .
      | schema_name .
    ]
```

```
    table_or_view_name
}
<OUTPUT_CLAUSE> ::=
{
    OUTPUT <dml_select_list> [ ,...n ]
    INTO @table_variable
}
<dml_select_list> ::=
{ <column_name> | scalar_expression }

<column_name> ::=
{ DELETED | INSERTED | from_table_name } . { * | column_name }

*OUTPUT 키워는 Insert, Update, Delete 모두 동일 적용
```

Top 키워드는 Select 절의 Top키워드와 같다. 다음의 예를 보면 바로 알 수 있는 키워드이다.

```
USE AdventureWorks;
GO
IF OBJECT_ID (N'HumanResources.NewEmployee', N'U') IS NOT NULL
    DROP TABLE HumanResources.NewEmployee;
GO
CREATE TABLE HumanResources.NewEmployee
(
    EmployeeID INT NOT NULL,
    LastName Name NOT NULL,
    FirstName Name NOT NULL,
    Phone Phone NULL,
    AddressLine1 NVARCHAR(60) NOT NULL,
    City NVARCHAR(30) NOT NULL,
    State NCHAR(3) NOT NULL,
    PostalCode NVARCHAR(15) NOT NULL,
    CurrentFlag Flag
);
GO
INSERT TOP (10) INTO HumanResources.NewEmployee
```

```
    SELECT --290 rows
        e.EmployeeID, c.LastName, c.FirstName, c.Phone,
        a.AddressLine1, a.City, sp.StateProvinceCode,
        a.PostalCode, e.CurrentFlag
    FROM HumanResources.Employee e
        INNER JOIN Person.Address AS a
        ON e.AddressID = a.AddressID
        INNER JOIN Person.StateProvince AS sp
        ON a.StateProvinceID = sp.StateProvinceID
        INNER JOIN Person.Contact as c
        ON e.ContactID = c.ContactID;
GO
SELECT Count(*) cnt
FROM HumanResources.NewEmployee;
GO

cnt
-----------
10

(1 row(s) affected)
```

Output 키워드도 추가되었다. Insert 하면서 내가 Insert한 데이터에 대한 조회가 가능하다. 다음은 그 예이다.

```
--Drop Table #Temp
Create Table #Temp(id int, name varchar(10))

Declare @InsertedTable Table(id int, name varchar(10))
Insert Into #Temp
Output Inserted.id, Inserted.name Into @InsertedTable
Values(1, 'yasi')

Select * From @InsertedTable
Go

id          name
```

```
----------- ----------
1          yasi


(1 row(s) affected)
```

Update

달라진 문법은 다음과 같다.

```
[ WITH <common_table_expression> [...n] ]
UPDATE
    [ TOP ( expression ) [ PERCENT ] ]
    { <object> | rowset_function_limited
     [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
    }
    SET
        { column_name = { expression | DEFAULT | NULL }
          | { udt_column_name.{ { property_name = expression
                                | field_name = expression }
                               | method_name ( argument [ ,...n ] )
                              }
            }
          | column_name { .WRITE ( expression , @Offset , @Length ) }
          | @variable = expression
          | @variable = column = expression } [ ,...n ]
        } [ ,...n ]
    [ <OUTPUT Clause> ]
    [ FROM{ <table_source> } [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF
                  { { [ GLOBAL ] cursor_name }
                      | cursor_variable_name
                  }
                ]
              }
            }
    ]
```

```
    [ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]

<object> ::=
{
    [ server_name . database_name . schema_name .
    | database_name .[ schema_name ] .
    | schema_name .
    ]
    table_or_view_name
}
```

Insert와 별다른 차이는 없다. 단지 Update연산은 Delete후 Insert 작업이므로 Deleted, Inserted 키워드를 사용하여 Delete되고 Insert되는 데이터를 알 수 있다.

```
--Drop Table #Temp
Create Table #Temp(id int, name varchar(10))

Declare @InsertedTable Table(Old Varchar(10), New Varchar(10), Name
Varchar(255))
Insert Into #Temp Values(1, 'yasi')

Update #Temp
Set Name = 'yasicom'
Output Deleted.Name, Inserted.Name, Suser_Name() Into @InsertedTable
Where id = 1

Select * From @InsertedTable

Old        New        Name
---------- ---------- -------------
yasi       yasicom    YASINOTE\yasi

(1 row(s) affected)
```

Delete

Delete의 달라진 문법은 다음과 같다.

```
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
    [ TOP ( expression ) [ PERCENT ] ]
    [ FROM ]
    { <object> | rowset_function_limited
      [ WITH ( <Table Hint> [ ...n ] ) ]
    }
    [ <OUTPUT Clause> ]
    [ FROM <table_source> [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF
                   { { [ GLOBAL ] cursor_name }
                       | cursor_variable_name
                   }
                ]
              }
            }
    ]
    [ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]

<object> ::=
{
    [ server_name.database_name.schema_name.
      | database_name. [ schema_name ] .
      | schema_name.
    ]
    table_or_view_name
}
```

역시 Insert, Update와 별차이 없다. Output 키워드는 다음과 같이 사용된다.

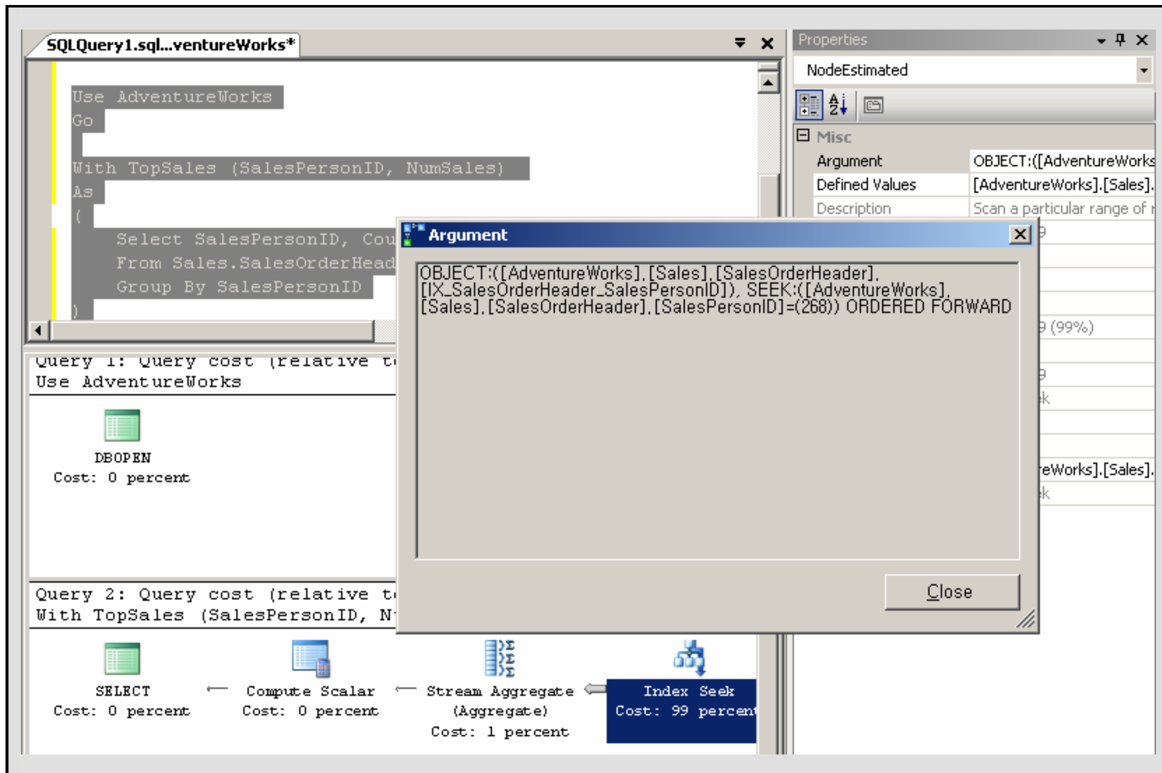
```
--Drop Table #Temp
Create Table #Temp(id int, name varchar(10))
```

```
Declare @InsertedTable Table(Deleted1 Varchar(10), Deleted2
Varchar(10))
Insert Into #Temp Values(1, 'yasi')
Insert Into #Temp Values(2, 'terry')

Delete #Temp
Output Deleted.Id, Deleted.Name Into @InsertedTable

Select * From @InsertedTable

Deleted1   Deleted2
---------- ----------
1          yasi
2          terry

(2 row(s) affected)
```

Common Table Expressions(CTE)

Common Table Expressions(CTE)는 SQL-99에 정의된 내용이다. 이전에는 View를 만들거나 In-Line View를 사용하였으나, 이제는 With 키워드를 사용하여 처리할 수 있다.

```
Use AdventureWorks
Go

With TopSales (SalesPersonID, NumSales)
As
(
        Select SalesPersonID, Count(*)
        From Sales.SalesOrderHeader
        Group By SalesPersonID
)
Select * From TopSales
Where SalesPersonID Is Not Null
And SalesPersonID = '268'
Order By 1
```



CTE를 사용하면 다음과 같은 이점이 있다.

- 소스가 단순해 진다.
- 순환관계를 간단히 해결할 수 있다.

순환관계를 해결하려면 Union All 키워드와 CTE를 사용해야 한다. 다음은 그 예이다.

```
USE AdventureWorks
GO
WITH DirectReports(ManagerID, EmployeeID, EmployeeLevel) AS
(
    -- 1
    SELECT ManagerID, EmployeeID, 0 AS EmployeeLevel
    FROM HumanResources.Employee
    WHERE ManagerID IS NULL
    UNION ALL
    --2
    SELECT e.ManagerID, e.EmployeeID, EmployeeLevel + 1
    FROM HumanResources.Employee e
        INNER JOIN DirectReports d
```

```sql
        ON e.ManagerID = d.EmployeeID
)
SELECT *
FROM DirectReports
WHERE EmployeeLevel <= 2
--Option (MaxRecursion 100) - Default 100, between 0 and 32,767
GO
```


일종의 Self-Join 기법을 이용한 것이다. 그러나 역시 순환적인 것이 보인다. 실행계획을 보아도 Spool 연산자가 보인다. Union All을 사용한 것은 Start 지점을 찾기 위함이다. 즉, 1은 Start 지점(Root)을 나타낸다. CTE의 처음은 ManageID가 Null인 EmployeeID을 찾는다. 2는 처음에 아무것도 리턴되지 않는다. 2번째 호출될 때는 처음 우리가 찾았던 ManageID가 Null인 EmployeeID과 HumanResources.Employee.ManagerID를 조인시켜서 찾는 방식으로 순환된다. 중간에 Assert 연산자가 보이는데 이것은 MaxRecursion의 기본값이 100이기 때문에 생긴 것이다.

Pivot & UnPivot

2005버전에서는 피벗을 쉽게 할 수 있다고 한다. 이것도 만만치는 않다. 문법은 다음과 같다.

```
SELECT * FROM table_source
PIVOT ( aggregate_function ( value_column )
FOR pivot_column
IN ( <column_list> )
) table_alias
```

다음은 그 예제이다.

```
Use AdventureWorks
Go

With Table_Source As
(
        Select Color, Count(*) Cnt
        From Production.Product
        Group By Color
)
Select Red, Blue, Yellow
From Table_Source
Pivot (
        Sum(Cnt)
        For Color In ([Red], [Blue], [Yellow])
) As Pvt

Red         Blue        Yellow
----------- ----------- -----------
38          26          36
```

다음은 위의 결과를 UnPivot한 것이다.

```
Use AdventureWorks
Go
```

```
With Table_Source As
(
        Select Color, Count(*) Cnt
        From Production.Product
        Group By Color
)
Select Color, Cnt
From(
        Select Red, Blue, Yellow
        From Table_Source
        Pivot (
              Sum(Cnt)
              For Color In ([Red], [Blue], [Yellow])
        ) As Pvt ) T
UnPivot(
        Cnt For Color In ([Red], [Blue], [Yellow])
) As UnPvt

Color    Cnt
------- -----------
Red      38
Blue     26
Yellow   36
Warning: Null value is eliminated by an aggregate or other SET operation.

(3 row(s) affected)
```

Apply 연산자

Apply연산자는 From절의 왼쪽에 일반적인 Table_Source가 위치하고, 오른쪽에 테이블 데이터타입을 리턴하는 함수가 위치했을 때 왼쪽의 Table_Source로부터 매개변수를 입력 받는 경우에 쓰이는 연산자이다. Cross연산자는 왼쪽과 오른쪽이 매칭되는 경우에만 쓰이고, Outer의 경우는 모든 로우를 대상으로 한다.

```
--Create Employees table and insert values
CREATE TABLE Employees
(
  empid   int         NOT NULL,
  mgrid   int         NULL,
  empname varchar(25) NOT NULL,
  salary  money       NOT NULL,
  CONSTRAINT PK_Employees PRIMARY KEY(empid),
)
GO
INSERT INTO Employees VALUES(1 , NULL, 'Nancy'   , $10000.00)
INSERT INTO Employees VALUES(2 , 1   , 'Andrew'  , $5000.00)
INSERT INTO Employees VALUES(3 , 1   , 'Janet'   , $5000.00)
INSERT INTO Employees VALUES(4 , 1   , 'Margaret', $5000.00)
INSERT INTO Employees VALUES(5 , 2   , 'Steven'  , $2500.00)
INSERT INTO Employees VALUES(6 , 2   , 'Michael' , $2500.00)
INSERT INTO Employees VALUES(7 , 3   , 'Robert'  , $2500.00)
INSERT INTO Employees VALUES(8 , 3   , 'Laura'   , $2500.00)
INSERT INTO Employees VALUES(9 , 3   , 'Ann'     , $2500.00)
INSERT INTO Employees VALUES(10, 4   , 'Ina'     , $2500.00)
INSERT INTO Employees VALUES(11, 7   , 'David'   , $2000.00)
INSERT INTO Employees VALUES(12, 7   , 'Ron'     , $2000.00)
INSERT INTO Employees VALUES(13, 7   , 'Dan'     , $2000.00)
INSERT INTO Employees VALUES(14, 11  , 'James'   , $1500.00)
GO
--Create Departments table and insert values
CREATE TABLE Departments
(
  deptid    INT NOT NULL PRIMARY KEY,
```

```
  deptname  VARCHAR(25) NOT NULL,
  deptmgrid INT NULL REFERENCES Employees
)
GO
INSERT INTO Departments VALUES(1, 'HR',           2)
INSERT INTO Departments VALUES(2, 'Marketing',    7)
INSERT INTO Departments VALUES(3, 'Finance',      8)
INSERT INTO Departments VALUES(4, 'R&D',          9)
INSERT INTO Departments VALUES(5, 'Training',     4)
INSERT INTO Departments VALUES(6, 'Gardening', NULL)
Go

CREATE FUNCTION dbo.fn_getsubtree(@empid AS INT) RETURNS @TREE TABLE
(
  empid   INT NOT NULL,
  empname VARCHAR(25) NOT NULL,
  mgrid   INT NULL,
  lvl     INT NOT NULL
)
AS
BEGIN
  WITH Employees_Subtree(empid, empname, mgrid, lvl)
  AS
  (
    -- Anchor Member (AM)
    SELECT empid, empname, mgrid, 0
    FROM employees
    WHERE empid = @empid

    UNION all

    -- Recursive Member (RM)
    SELECT e.empid, e.empname, e.mgrid, es.lvl+1
    FROM employees AS e
      JOIN employees_subtree AS es
        ON e.mgrid = es.empid
```

```
  )
  INSERT INTO @TREE
    SELECT * FROM Employees_Subtree

  RETURN
END
GO

SELECT *
FROM Departments AS D
  CROSS APPLY fn_getsubtree(D.deptmgrid) AS ST
Go
```

	deptid	deptname	deptmgrid	empid	empname	mgrid	lvl
1	1	HR	2	2	Andrew	1	0
2	1	HR	2	5	Steven	2	1
3	1	HR	2	6	Michael	2	1
4	2	Marketing	7	7	Robert	3	0
5	2	Marketing	7	11	David	7	1
6	2	Marketing	7	12	Ron	7	1
7	2	Marketing	7	13	Dan	7	1
8	2	Marketing	7	14	James	11	2
9	3	Finance	8	8	Laura	3	0
10	4	R&D	9	9	Ann	3	0
11	5	Training	4	4	Margaret	1	0
12	5	Training	4	10	Ina	4	1

```
SELECT *
FROM Departments AS D
  Outer APPLY fn_getsubtree(D.deptmgrid) AS ST
Go
```

	deptid	deptname	deptmgrid	empid	empname	mgrid	lvl
1	1	HR	2	2	Andrew	1	0
2	1	HR	2	5	Steven	2	1
3	1	HR	2	6	Michael	2	1
4	2	Marketing	7	7	Robert	3	0
5	2	Marketing	7	11	David	7	1
6	2	Marketing	7	12	Ron	7	1
7	2	Marketing	7	13	Dan	7	1
8	2	Marketing	7	14	James	11	2
9	3	Finance	8	8	Laura	3	0
10	4	R&D	9	9	Ann	3	0
11	5	Training	4	4	Margaret	1	0
12	5	Training	4	10	Ina	4	1
13	6	Gardening	NULL	NULL	NULL	NULL	NULL

http://www.DataBaser.Net

Windowing Function

이제 2005버전에도 Windowing Function이 생겼다. 많이 생기지는 않았지만 그래도 목마른 자에게 한 모금의 물과도 같을 것이다. Window는 하나의 논리적인 부분집합을 말한다. 테이블에서의 Window는 현재 행을 기준으로 기술해준 특정 범위가 하나의 Window가 되는 것이다. 이 특정 범위는 전체가 될 수도 있고, 부분이 될 수도 있다. 단지 이 범위(Window)는 반드시 순서를 가져야 한다. 그러므로 Order By절은 필수요소이다. 2005버전에서는 4개의 Windowing Function을 제공한다. 현재는 Beta2 버전이라 4개로 한정짓지는 않겠다. 정식버전에서는 더 많은 Windowing Function이 생겨나길 기대하면서..

Windowing Function	**Description**
Rank	윈도우 내의 순위(1, 2, 2, 4, 5, 6 …)
Dense_Rank	윈도우 내의 순위(1, 2, 2, 3, 4, 5 …)
Row_Number	윈도우 내의 행 번호
NTile	윈도우 내의 나누기 번호

위 표는 각 함수의 간단한 설명이다. 예를 보면 간단히 이해할 수 있다.

```
USE AdventureWorks
GO

Select
        SalesPersonID
,       SalesQuota
,       Row_Number() Over(Order By SalesPersonID) Row_Number
,       Row_Number() Over(Partition By SalesQuota
                       Order By SalesPersonID) Row_Number_Part
,       Rank() Over(Order By SalesQuota Desc) Rank
,       Dense_Rank() Over(Order By SalesQuota Desc) Dense_Rank
,       NTile(4) Over(Order By SalesQuota Desc) NTile
From Sales.SalesPerson
```

	SalesPersonID	SalesQuota	Row_Number	Row_Number_Part	Rank	Dense_Rank	NTile
1	275	300000.00	2	1	1	1	1
2	279	300000.00	6	2	1	1	1
3	287	300000.00	14	3	1	1	1
4	276	250000.00	3	1	4	2	1
5	277	250000.00	4	2	4	2	1
6	278	250000.00	5	3	4	2	2
7	280	250000.00	7	4	4	2	2
8	281	250000.00	8	5	4	2	2
9	282	250000.00	9	6	4	2	2
10	283	250000.00	10	7	4	2	3
11	285	250000.00	12	8	4	2	3
12	286	250000.00	13	9	4	2	3
13	289	250000.00	16	10	4	2	3
14	290	250000.00	17	11	4	2	4
15	268	NULL	1	1	15	3	4
16	284	NULL	11	2	15	3	4
17	288	NULL	15	3	15	3	4

이제 이전 버전처럼 순위를 구하기 위해 복잡한 쿼리를 작성할 필요가 없으며, 오라클의 RowNum을 구현할 수 있게 되었다. 물론 오라클의 RowNum보다는 부하가 많은 것은 사실이지만 그래도 이게 어디인가?

Try … Catch

이전 버전에서는 예외처리가 좀 구렸던 것이 사실이다. 이런 것을 반영하듯 Try … Catch 구문이 생겨나 버렸다. 이 구문을 사용하기 위해서는 "If @@Error" 코드가 주로 사용되기도 하며, XACT_ABORT ON 옵션이 필요할 수도 있다.

```
BEGIN TRY
     { sql_statement | statement_block }
END TRY
BEGIN CATCH
     { sql_statement | statement_block }
END CATCH
[ ; ]
```

다음은 Try … Catch의 간단한 예이다. 위의 문법과 같이 중간에 Go 와 같은 실행단위의 문자가 들어가면 안 된다.

```
BEGIN TRY
    SELECT *
        FROM sys.messages
        WHERE message_id = 21;
END TRY
-- GO
-- The previous GO breaks the script into two batches,
-- generating syntax errors. The script runs if this GO
-- is removed.
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO
```

또한 다음과 같이 Error Message를 뿌려줄 수도 있다.

```
USE AdventureWorks;
GO
BEGIN TRANSACTION;
GO
```

```
BEGIN TRY
    -- Generate a constraint violation error.
    DELETE FROM Production.Product
        WHERE ProductID = 980;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_MESSAGE() as ErrorMessage;
END CATCH
GO

ROLLBACK TRANSACTION;
GO
```

Try … Catch를 이용하는 또 다른 2가지 방법을 소개하도록 하겠다. 첫 번째 방법은 @@Error를 이용하는 방법이다. 예제에서는 Insert에 문제가 없다면 Commit 하고, 문제가 발생하면 Catch로 이동하는 형식을 취하고 있다. 예제에서는 XACT_ABORT ON과 @@Error가 필요하다. @@Error는 옵션이며, XACT_ABORT는 반드시 필요하다.

```
Use Tempdb
Go

CREATE TABLE dbo.DataTable
       (ColA int PRIMARY KEY, ColB int)
CREATE TABLE dbo.ErrorLog
       (ColA int, ColB int, error int, date datetime)
GO

CREATE PROCEDURE dbo.AddData @a int, @b int
AS
SET XACT_ABORT ON
BEGIN TRY
       BEGIN TRAN
             INSERT INTO dbo.DataTable VALUES (@a, @b)
       COMMIT TRAN
```

```
END TRY

BEGIN CATCH
        DECLARE @err int
        SET @err = @@error --trap the error number
        ROLLBACK TRAN
        INSERT INTO dbo.ErrorLog VALUES (@a, @b, @err, GETDATE())
END CATCH
GO

EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1, 3 --violates the primary key

Select * From dbo.ErrorLog
Go

ColA        ColB        error       date
----------- ----------- ----------- -----------------------
1           3           2627        2005-02-12 13:54:54.377

(1 row(s) affected)
```

두 번째 예제에서 눈여겨 볼 것은 XACT_STATE()를 이용한 것이다. -1 이면 트랜잭션이 어떤 이유에서인지는 모르지만 실패한 것을 나타내며, 1이면 성공이다. 0이면 진행 중이다. 이러한 것을 이용하여 XACT_ABORT ON과 COMMIT TRAN에 대해서 주석처리 하여 다른 방법을 사용하였다. 또한 무조건 Error가 발생한 것처럼 가장해야 하므로 RAISERROR를 사용한 것을 볼 수 있다. 이 예제에서 RAISERROR는 반드시 필요하다.

```
Use Tempdb
Go

CREATE TABLE dbo.DataTable
        (ColA int PRIMARY KEY, ColB int)
CREATE TABLE dbo.ErrorLog
        (ColA int, ColB int, error int, date datetime)
GO
```

```
CREATE PROCEDURE dbo.AddData @a int, @b int
AS
--SET XACT_ABORT ON
BEGIN TRY
       BEGIN TRANSACTION;
             INSERT INTO dbo.DataTable VALUES (@a, @b)
       RAISERROR(N'Throw an error.', 16, 1); -- Need
       --COMMIT TRAN
END TRY

BEGIN CATCH
   DECLARE @err int
   IF (XACT_STATE()) = -1
        ROLLBACK TRANSACTION;
             SET @err = @@error --trap the error number
             INSERT INTO dbo.ErrorLog VALUES (@a, @b, @err, GETDATE())
    IF (XACT_STATE()) = 1
        COMMIT TRANSACTION;
END CATCH
GO

EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1, 3 --violates the primary key

Select * From dbo.ErrorLog
Go
```

Top & TableSample

2000 버전에서는 Top의 제한이 좀 많이 있었다. 2005 버전에서는 Top으로 결정하는 Stop Key의 개수(또는 양)이 변수로 사용될 수 있다. 물론 Select 문장도 들어갈 수 있다. 숫자 형만 되면 된다.

```
Use AdventureWorks
Go

Declare @i int
Set @i = 5

Select Top (@i) ProductID, Name, ProductNumber
From Production.Product

ProductID   Name                                               ProductNumber
----------- -------------------------------------------------- -------------------------
1           Adjustable Race                                    AR-5381
2           Bearing Ball                                       BA-8327
3           BB Ball Bearing                                    BE-2349
4           Headset Ball Bearings                              BE-2908
316         Blade                                              BL-2036

(5 row(s) affected)

Select Top (Select Count(*) From Production.Product
            Where ProductNumber = 'AR-5381')
          ProductID, Name, ProductNumber
From Production.Product

ProductID   Name                                               ProductNumber
----------- -------------------------------------------------- -------------------------
1           Adjustable Race                                    AR-5381

(1 row(s) affected)
```

```
Select Top (Select Count(*) From Production.Product
            Where ProductNumber = 'AR-5381') Percent
          ProductID, Name, ProductNumber
From Production.Product

ProductID   Name                                               ProductNumber
----------- -------------------------------------------------- -------------------------
1           Adjustable Race                                    AR-5381
2           Bearing Ball                                       BA-8327
3           BB Ball Bearing                                    BE-2349
4           Headset Ball Bearings                              BE-2908
316         Blade                                              BL-2036
317         LL Crankarm                                        CA-5965

(6 row(s) affected)
```

2000 버전에서는 랜덤으로 10개의 로우를 가져오려면 NewID() 함수를 사용해서 정렬해야 했다. 그러나 2005 버전에서는 TableSample이라는 키워드로써 이러한 기능을 수행한다.

```
Use AdventureWorks
Go

--2000 version
Select Top 5 ProductID, Name, ProductNumber
From Production.Product
Order By NewID()

--5개의 로우 리턴
Select ProductID, Name, ProductNumber
From Production.Product TableSample (5 Rows)

--5%의 로우 리턴
Select ProductID, Name, ProductNumber
From Production.Product TableSample (5 Percent)

--Top과 달리 변수가 통하지 않는다.
Declare @i int
```

```
Set @i = 5

Select ProductID, Name, ProductNumber
From Production.Product TableSample (@i Percent)

Msg 497, Level 15, State 1, Line 4
Variables are not allowed in the TABLESAMPLE or REPEATABLE clauses.
```