SQL Server – EXECUTE Statement with RESULT SET Clause

SQL Server 2012 introduced a RESULT SET clause to the EXECUTE statement.

It can be used to specify alternate data types and column names for result sets returned by an EXECUTED statement or Stored Procedure.

The following example shows its use with an ad-hoc query example.

- The first query uses the RESULT AS clause to define the names datatypes for three returned columns.
- The second query uses a CAST and column aliasing to achieve the same result.

```
--Rename and retype results from a SELECT statement
EXEC ('SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee')
WITH RESULT SETS
(
   ([Reporting Level] VARCHAR(3),
    [ID of Employee] int NOT NULL,
    [Employee Job Title] nvarchar(50) NOT NULL )
);

--The above could be done as follows
SELECT
    CAST(OrganizationLevel AS VARCHAR(3)) AS [Reporting Level],
    BusinessEntityID AS [ID of Employee] ,
    JobTitle AS [Employee Job Title]
FROM HumanResources.Employee
```

100 %

Results Messages

	Reporting Level	ID of Employee	Employee Job Title
1	NULL	1	Chief Executive Officer
2	1	2	Vice President of Engineering
3	2	3	Engineering Manager
4	3	4	Senior Tool Designer
5	3	5	Design Engineer
6	3	6	Design Engineer
7	3	7	Research and Development Manager

	Reporting Level	ID of Employee	Employee Job Title
1	NULL	1	Chief Executive Officer
2	1	2	Vice President of Engineering
3	2	3	Engineering Manager
4	3	4	Senior Tool Designer
5	3	5	Design Engineer
6	3	6	Design Engineer
7	3	7	Research and Developmen...

The RESULT SET clause is more useful when working with Stored Procedures that provide no opportunity to change the column names defined within the stored procedure or the data types derived in the underlying Transact SQL code within the procedure.

The following example shows a stored procedure definition and then an EXECUTE statement that changes the column names and data types of the results set returned by the stored procedure.

```
CREATE PROC GetEmployeeLevel
AS
    SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee
GO

EXEC GetEmployeeLevel
WITH RESULT SETS
(
   ([Reporting Level] VARCHAR(3),
    [ID of Employee] int NOT NULL,
    [Employee Job Title] nvarchar(50) NOT NULL )
);
```

100 %

Results Messages

	Reporting Level	ID of Employee	Employee Job Title
1	NULL	1	Chief Executive Officer
2	1	2	Vice President of Engineering
3	2	3	Engineering Manager
4	3	4	Senior Tool Designer
5	3	5	Design Engineer
6	3	6	Design Engineer
7	3	7	Research and Development Manager
8	4	8	Research and Development Engineer
9	4	9	Research and Development Engineer
10	4	10	Research and Development Manager
11	3	11	Senior Tool Designer
12	4	12	Tool Designer
13	4	13	Tool Designer
14	3	14	Senior Design Engineer
15	3	15	Design Engineer
16	1	16	Marketing Manager
17	2	17	Marketing Assistant

The following example demonstrates that multiple results sets can be handled where a stored procedure returns more than one result set.

```
CREATE PROC GetEmployeeNamesAndTitles
AS
    SELECT BusinessEntityID, FirstName + ' ' + LastName FROM Person.Person
    SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee
GO

EXEC GetEmployeeNamesAndTitles
WITH RESULT SETS
(
    (
        [ID Of Employee] INT NOT NULL,
        [Name Of Employee] VARCHAR(100)
    ),
    (
        [Reporting Level] VARCHAR(3),
        [ID of Employee] INT NOT NULL,
        [Employee Job Title] NVARCHAR(50) NOT NULL
    )
);
```

100 %

Results Messages

	ID Of Employee	Name Of Employee
1	285	Syed Abbas
2	293	Catherine Abel
3	295	Kim Abercrombie
4	2170	Kim Abercrombie
5	38	Kim Abercrombie
6	211	Hazem Abolrous

	Reporting Level	ID of Employee	Employee Job Title
1	NULL	1	Chief Executive Officer
2	1	2	Vice President of Engineering
3	2	3	Engineering Manager
4	3	4	Senior Tool Designer

The examples are based on the AdvetureWorks2014 database. Here is the code for the examples in this document:

```
--Using the RESULT SETS clause of the EXECUTE Statement

--Rename and retype results from a SELECT statement
EXEC ('SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee')
WITH RESULT SETS
(
   ([Reporting Level] VARCHAR(3),
    [ID of Employee] int NOT NULL,
    [Employee Job Title] nvarchar(50) NOT NULL )
);

--The above could be done as follows
SELECT
       CAST(OrganizationLevel AS VARCHAR(3)) AS [Reporting Level],
       BusinessEntityID AS [ID of Employee] ,
       JobTitle AS [Employee Job Title]
FROM HumanResources.Employee


--With a Stored Procedure
CREATE PROC GetEmployeeLevel
AS
       SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee
GO

EXEC GetEmployeeLevel
WITH RESULT SETS
(
   ([Reporting Level] VARCHAR(3),
    [ID of Employee] int NOT NULL,
    [Employee Job Title] nvarchar(50) NOT NULL )
);

--Stored Procedure that returns two results sets
CREATE PROC GetEmployeeNamesAndTitles
AS
       SELECT BusinessEntityID, FirstName + ' ' + LastName FROM Person.Person
       SELECT OrganizationLevel, BusinessEntityID, JobTitle FROM HumanResources.Employee
GO

EXEC GetEmployeeNamesAndTitles
WITH RESULT SETS
(
       (
              [ID Of Employee] INT NOT NULL,
              [Name Of Employee] VARCHAR(100)
       ),

   (
              [Reporting Level] VARCHAR(3),
              [ID of Employee] INT NOT NULL,
              [Employee Job Title] NVARCHAR(50) NOT NULL
   )
);
```

If you are itching to learn more why not book on to our SQL Server Database Querying training courses? *This link will take you to the course outlines:*

http://ptr.co.uk/databases-business-intelligence-courses.

