Module 5: Subqueries, Derived Tables and Common Table Expressions

Subqueries

A ***subquery*** is a query inside of another query that acts as an expression. Subqueries range from the extremely simple to the complex. Often, subqueries can be used as an alternative to a JOIN operation to produce the same result set. There's often little performance difference between a SELECT query containing a subquery and an equivalent query written using a JOIN.

There are two types of subqueries:

1. **Non-correlated** – a subquery that can be run independently of the outer query.
2. **Correlated** – a subquery that is dependent on values in the outer query.

Non-correlated Subqueries

Often you'll want to use a subquery as an expression in a SELECT list. The rules for doing this generally follow the same rules as any other expression.

Consider the query below:

```
SELECT
	c.Make
	,c.Model
	,c.Year
	,s.SalesAmount
	,(SELECT avg(SalesAmount) FROM Sales) as [Average Sales Amount]
	,s.SalesAmount - (SELECT avg(SalesAmount) FROM Sales) as [Difference]
FROM Sales s
JOIN Cars c
ON s.CarID=c.CarID;
```

The inner query is an expression that finds the average sales amount for all sales. We call it non-correlated because it can be run as an independent query:

```
	SELECT avg(SalesAmount) FROM Sales
```

The above query can be run on its own and doesn't depend on any values from the outer query.

Here's another example of a non-correlated subquery.

```
SELECT
	Make
	,Model
	,Year
FROM Cars
WHERE CarID IN (SELECT TOP 5 CarID from Sales ORDER BY SalesAmount);
```

In this case, the subquery is producing a list of Car ID's used with the IN operator.

Here's another example of an uncorrelated subquery used with a comparison operator in a WHERE clause.

```
SELECT
	p.FirstName
	,p.LastName
	,crs.Make
	,crs.Model
	,crs.Year
FROM dbo.Sales s
JOIN dbo.Customers c
ON s.CustomerID=c.CustomerID
JOIN dbo.Person p
ON c.PersonID=p.PersonID
JOIN dbo.Cars crs
ON s.CarID=crs.CarID
WHERE s.SalesAmount=(SELECT MAX(SalesAmount) FROM dbo.Sales);
```

In this case, the subquery is returning the highest sales amount for any sale.

Correlated Subqueries

A subquery that is dependent upon one or more values in the outer query is called a *correlated subquery*. In the query below, the correlated subquery within the outer query's WHERE clause finds the average sales amount for the current sales person. The result is a list of car sales for which the sales amount was greater than or equal to the average sales amount for that sales person.

```
SELECT
	c.Make
	,c.Model
	,c.Year
	,s1.SalesAmount
	,s1.SalesDate
FROM dbo.Sales s1
JOIN dbo.Cars c
ON s1.CarID=c.CarID
WHERE s1.SalesAmount >=
(Select avg(s2.SalesAmount) from dbo.Sales s2 where s2.SalesPersonID=s1.SalesPersonID)
```

The following query uses a correlated subquery in the WHERE clause to produce a list of all car sales in which the sales amount was greater than or equal to the average sales amount for that customer's demographic group.

```
SELECT
	c1.CustomerID
	,s1.SalesAmount
	,s1.SalesDate
FROM dbo.Sales s1
JOIN dbo.Customers c1
```

```
ON s1.CustomerID=c1.CustomerID
WHERE s1.SalesAmount >=
(SELECT AVG(s2.SalesAmount)
FROM dbo.Sales s2
JOIN dbo.Customers c2
ON s2.CustomerID=c2.CustomerID
JOIN dbo.CustomerDemographics cd
ON c2.DemographicsGroup=cd.DemogroupID
WHERE c1.DemographicsGroup=c2.DemographicsGroup)
```

Derived Tables

Just as queries can be used as scalar and single-column expressions in other queries, queries can also be used as table expressions. We call the table resulting from such a query a *derived table*. The query we use for the derived table must be enclosed in parentheses, must be given a table alias, and must provide an alias for any computed columns. Also, we cannot reference the derived table multiple times in the same query.

In the query below, a derived table named sa is created then queried:

```
SELECT
       sa.SalesPersonID
       ,sa.MeanSale
FROM
(
SELECT
       SalesPersonID,
       AVG(SalesAmount) AS [MeanSale]
FROM dbo.Sales
GROUP BY SalesPersonID
) sa
```

Here's another example. Let's assume you want to produce a list of sales people including their full names and contact information, along with the total number of cars each sold in 2012. One approach we might take would be to first write a query that would return the number of cars sold by each sales person's personID value, then use this query as a derived table in a join with the dbo.Person table.

Consider the following query:

```
SELECT
       p.LastName + ', ' + p.FirstName as [Sales Person]
       ,p.Email
       ,p.HomePhone
       ,ps.[Total Cars Sold]
FROM
dbo.Person p
JOIN
(
SELECT
       p.PersonID
       ,COUNT(s.SaleID) as [Total Cars Sold]
FROM dbo.Sales s
```

```
JOIN dbo.SalesPeople sp
ON s.SalesPersonID=sp.EmployeeID
JOIN dbo.Person p
ON sp.PersonID=p.PersonID
WHERE year(s.SalesDate) = 2012
GROUP BY p.PersonID
) ps
ON p.PersonID=ps.PersonID
```

What we've done is define derived table ps by using a SELECT query that produces the number of cars sold by each sales person along with the PersonID value associated with that person. In the above case, we simply join our derived table ps to dbo.Person on the PersonID column. We can then refer to it in the SELECT list as we would any ordinary table.

Here's another example. In this case, we're using one derived table to find the average sales price for each customer demographic group, and another derived table to find the average sales price for each customer. We're then joining the two derived tables together with dbo.Person, dbo.Customer, and dbo.Sales to produce the difference between each sale and the demographic group average, and between each sale and the customer's own average sale price.

```
SELECT
       p.LastName + ', ' + p.FirstName as [Customer Name]
       ,p.Email
       ,s.SalesDate
       ,c.DemographicsGroup
       ,cd.[Mean Sale] as [Demographics Group Average Sale]
       ,custavg.[Mean Sale] as [Average Sale for Customer]
       ,s.SalesAmount - cd.[Mean Sale] as [Distance from Demo Group Average Sale]
       ,s.SalesAmount - custavg.[Mean Sale] as [Distance from Customers Own Average]
FROM dbo.Person p
JOIN dbo.Customers c
on p.PersonID=c.PersonID
JOIN dbo.Sales s
ON c.CustomerID=s.CustomerID
JOIN (
SELECT
       cd.DemogroupID
       ,AVG(s.SalesAmount) as [Mean Sale]
FROM dbo.CustomerDemographics cd
JOIN dbo.Customers c
on c.DemographicsGroup=cd.DemogroupID
JOIN dbo.Sales s
on c.CustomerID=s.CustomerID
GROUP BY cd.DemogroupID
) cd
on c.DemographicsGroup=cd.DemogroupID
JOIN (
SELECT
       s.CustomerID
       ,AVG(s.SalesAmount) as [Mean Sale]
FROM dbo.Sales s
GROUP BY s.CustomerID
) custavg
```

```
ON c.CustomerID=custavg.CustomerID
ORDER BY p.LastName,p.FirstName
```

Common Table Expression

An alternative to the derived table is the Common Table Expression (abbreviated as CTE). The CTE uses the WITH clause to define a table which exists within the scope of a single SELECT query. The CTE is a lot like a derived table, but you can reference it multiple times within the same query, and the CTE can call itself.

For additional information on Common Table Expressions, see https://technet.microsoft.com/en-us/library/ms190766.aspx.

The query below demonstrates the use of a derived table. The WITH clause defines a CTE name cte_TotalSalesByMMY which is then queried in the SELECT query of which it is a part.

```
WITH cte_TotalSalesByMMY (Make, Model, Year, TotalSales)
AS
(
SELECT
	c.Make
	,c.Model
	,c.Year
	,SUM(s.SalesAmount) AS [TotalSales]
FROM dbo.Sales s
JOIN dbo.Cars c
ON s.CarID=c.CarID
GROUP BY c.Make,c.Model,c.Year
)
SELECT
	Make
	,Model
	,Year
	,TotalSales
FROM cte_TotalSalesByMMY
ORDER BY TotalSales DESC
```

In the query below, the Common Table Expression contains a list of the CustomerID and average sales amount for cars purchased by that customer. The CTE is then joined to the dbo.Customers and dbo.Person table in order to find the first and last name of the customer.

```
WITH cte_MeanSaleCustomer(CustomerID,MeanSalesAmount)
as
(
SELECT
	CustomerID
	,AVG(SalesAmount) AS MeanSalesAmount
FROM dbo.Sales
GROUP BY CustomerID
)
SELECT
	p.LastName + ', ' + p.FirstName AS [Customer Name]
	,cte.MeanSalesAmount
```

```
FROM dbo.Customers c
JOIN dbo.Person p
ON c.PersonID=p.PersonID
JOIN cte_MeanSaleCustomer cte
ON c.CustomerID=cte.CustomerID
ORDER BY p.LastName, p.FirstName
```

In the query below we'll use a CTE to calculate the mean sale price for each demographic group, and then join this to dbo.Customers in order to list the demographic group data and group mean sale price for each customer.

```
WITH cteDemoGroupAverages (DemoGroupID,[Mean Sales Amount])
as
(
SELECT
       cd.DemogroupID
       ,AVG(s.SalesAmount) as [Mean Sales Amount]
FROM dbo.Sales s
JOIN dbo.Customers c
ON s.CustomerID=c.CustomerID
JOIN dbo.CustomerDemographics cd
ON c.DemographicsGroup=cd.DemogroupID
GROUP BY cd.DemogroupID
)
SELECT
       p.LastName
       ,p.FirstName
       ,cd.MaritalStatus
       ,cd.Gender
       ,cd.IncomeBracket
       ,cd.RuralUrban
       ,cte.[Mean Sales Amount] as [Mean Group Sales Amount]
FROM dbo.Customers c
JOIN cteDemoGroupAverages cte
ON c.DemographicsGroup=cte.DemoGroupID
JOIN dbo.Person p
ON c.PersonID=p.PersonID
JOIN dbo.CustomerDemographics cd
ON cd.DemogroupID=c.DemographicsGroup
```

Additional Reading

For more information on subqueries, see https://technet.microsoft.com/en-us/library/ms189575.aspx.