essentialSQL

Subqueries – Introduction to a Subquery in SQL

这份教程是 subquery（子查询）的大致介绍。主要的应用场景和示例在另外几份单独的 PDF 中有详细展开。

The purpose of this article is to introduce you to subqueries and some of their high-level concepts. There are a lot of details to cover in order to learn sub queries, but you'll see we cover those in depth in later articles.

All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my Guide *Getting Started Using SQL Server*.

Subqueries in SQL

Subqueries provide a powerful means to combine data from two tables into a single result. You can also call these nested queries. As the name implies, subqueries contain one or more queries, one inside the other.

Subqueries are very versatile and that can make them somewhat hard to understand. For most cases use them anywhere you can use an expression or table specification.

For example, you can use subqueries in the SELECT, FROM, WHERE, or HAVING clauses. Depending on how they are used, a subquery may return a single value or multiple rows.

In later articles we'll get into the specifics on using subqueries throughout your SELECT statements such as the SELECT, WHERE, FROM, and HAVING clauses. They can be quite slippery and hard to wrap your brain around. Perhaps it would help to see a couple of examples.

Example Subqueries

Subqueries make it possible for you to write queries that are more dynamic and data driven. For instance using a subquery you can return all products whose ListPrice is greater than the average ListPrice for all products.

You can do this by having it first calculate the average price and then use this to compare against each product's price.

```
SELECT ProductID,
       Name,
       ListPrice
FROM   production.Product
WHERE  ListPrice > (SELECT AVG(ListPrice)
                    FROM   Production.Product)
```

subquery

子查询总是被括号包起来的。

这个例子中，子查询返回 ListPrice 的平均值，使 outer query 可以查询大于平均值的 Product。

下面的一节解释了这个查询的执行过程，不需要看，因为这个查询很直观。

Subquery Breakdown

Let's break down this query so you can see how it works.

Step 1: First let's run the subquery:

```
SELECT AVG(ListPrice)
FROM Production.Product
```

It returns **438.6662** as the average ListPrice

Step 2: Find products greater than the average price by plugging in the average ListPrice value into our query's comparison

```
SELECT ProductID,
       Name,
       ListPrice
```

```
FROM   production.Product
WHERE  ListPrice > 438.6662
```

As you can see, by using the subquery we combined the two steps together. The subquery eliminated the need for us to find the average ListPrice and then plug it into our query.

This is huge! It means our query automatically adjusts itself to changing data and new averages.

Hopefully you're seeing a glimpse into how subqueries can make your statements more flexible. In this case, by using a subquery we don't need to know the value for the average list price.

We let the subquery do the work for us! The average value is calculated on-the-fly; there is no need for us to "update" the average value within the query.

Being able to dynamically create the criterion for a query is very handy. Here we use a subquery to list all customers whose territories have sales below $5,000,000.

第二个例子，subquery 返回了多个值，配合 WHERE ... IN ...使用。

```
SELECT DISTINCT CustomerID
FROM   Sales.SalesOrderHeader
WHERE  TerritoryID IN (SELECT TerritoryID
                       FROM   Sales.SalesTerritory
                       WHERE  SalesYTD < 5000000)
```

Subquery

We build the list "live" with the IN operator, this avoid us from having to hard code the values.

It may help to see how to execute this query step by step:

Step 1: Run the subquery to get the list of territories that had year to date sales less than 5,000,000:

```
SELECT TerritoryID
FROM   Sales.SalesTerritory
WHERE  SalesYTD < 5000000
```

This returns **2,3,5,7,8** as a list of values.

Step 2: Now that we have a list of values we can plug them into the IN operator:

```
SELECT DISTINCT CustomerID
FROM   Sales.SalesOrderHeader
WHERE  TerritoryID IN (2,3,5,7,8)
```

Again, we could have just broken this down into multiple steps, but the disadvantage in doing so is we would have to constantly update the list of territories.

Things to Keep in Mind

- A subquery is just a SELECT statement inside of another.
- They are always enclosed in parenthesis ().
- A subquery that returns a single value can be used anywhere you would use an expression, such as in a column list or filter expression.
- A subquery that returns more than one value is typically used where a list of values, such as those used in and IN operator.
- Warning! Subqueries can be very inefficient. If there are more direct means to achieve the same result, such as using an inner join, you're better for it.
- You can nest subqueries up to thirty two levels deep on SQL server. To be honest, I can't imagine why! I've see maybe four deep at most. Realistically I think you only go one or two deep.

Don't worry, if a lot of this seems confusing at the moment. You still need to learn where the use subqueries within the SELECT statement. Once you see them in

action, the above points become clearer.

Where Can you use Subqueries?

In future articles we'll explore the four ways you can use a subquery in a SELECT statement. They are:

1. In a SELECT clause as a column expression
2. In a WHERE clause as a filter criteria
3. In the FROM clause as a table specification
4. In a HAVING clause as a group selector (filter criteria)

As you can see subqueries aren't that intimidating. The key to understanding them is understanding their role in the SQL statement. By now you know that the column list in a SELECT statement can only be one value.

It follows that a subquery used in a column list can only return one value.

In other cases, such as when being used with the IN operator, which operates on multiple values, it makes sense then that the subquery can return more than one row.

Remember! I want to remind you all that if you have other questions you want answered, then post a comment or tweet me. I'm here to help you.