Article: SQL Server - Subqueries
Date: 21/12/2011
Posted by: HeelpBook Staff
Source: Link
Permalink: Link

SQL Server - Subqueries

You've seen the Russian dolls where you've got one doll, inside another, inside another, etc.?

A **SQL *subquery*** is very much like this; it's tucked inside other commands, or even other **SQL** subqueries.

They may be placed inside one of the following commands:

- **SELECT**
- **SELECT…INTO**
- **INSERT…INTO**
- **DELETE**
- **UPDATE**

Subqueries are used to refine a **SQL** query when you'd rather just use one operation to find out what you need to know rather than two. This is best when you are planning to use a particular query over again in the future, as for a monthly sales figure or other regular metric.

Using SQL Subqueries

Sometimes when you're working on a **SQL** query, you find that you need to find out something else to finish your operations properly. For instance, suppose you need to find the top customers at your grocery store who have shopped during a certain period? Or the opposite, find among customers who have shopped during a certain period which spend the most money?

A ***SQL subquery*** is perfect for finding out this information. In plain ***English***, your command would be something like "***Select the best customers Where (Select customers who shopped during January)***" .

Assuming you have your top customers segregated in a second table, in **SQL** code this would translate to:

```
SELECT "TopCustomer" WHERE (SELECT "shoppingdate" = "October")
```

Another example is when you need to copy a table or portion of a table into a new table so you can perform other operations on it.

```
INSERT INTO TableNew SELECT TableNeeded
```

Where this gets really confusing is when you start nesting subquery after subquery. Like the **Russian** dolls, you can insert new subqueries for as long as you can keep track of them. Most **SQL** programmers are creating subqueries long before they start to use them.

It's just that most people create new tables from the nested query to perform the operations in the outside query before actually compiling subquery statements.

Three Types of Subquery

Subqueries can form three types of statements:

- Comparison: involving **ANY**, **ALL**, or **SOME**, this subquery compares an expression with the results of a subquery;
- Expression: Subqueries starting with **[NOT] IN** have their results searched for a particular expression;
- Sql statement: This is a standard **SELECT** statement. It starts with **[NOT] EXISTS;**

Syntax will be as follows:

```
SELECT "column_name1"
FROM "table_name"
WHERE "column_name2" [Comparison Operator]
(SELECT "column_name1"
FROM "table_name"
WHERE [Condition])
```

The comparison operator can be **-+/*** or any word comparison such as **LIKE**. You should notice that the **Column Name** for each **SELECT** operation should be the same, though the table name can vary.

Examples:

A retail store has been losing ***money*** by offering its customers too many discounts. In order to control discounts on a product, you might need to know which items you were running a **10%**, **20%**, or **30%** discount on.

Your **SQL** query would look something like this:

SELECT * FROM Product	This selects all records from the Product table.
WHERE Price > ANY	This guarantees that any product with a price is returned.
(SELECT Price FROM OrderPrice	This lets you select the Price from the OrderPrice table, where you'll see both regular discounts and customer-applied discounts
WHERE Discount >=10%);	This restricts your selection to only those with a discount of 10% or better.

You would have returned in the new table every record that has a discount price of **10%** or more. If you needed to change the discount percentage, that's easy to do.

You could even do the discount by dividing the original price by the actual sale price; it would simply require inserting a new subquery in the **10%** spot performing that operation.