In the last article, I alluded to SQL subqueries which we'll discuss in more detail this week.

Subqueries are extremely useful, particularly for web-based database applications where you need to take two queries and manually put them together to reach a desired result -- subqueries allow SQL to do all of the heavy lifting! Subqueries can also be used in many cases to replace a self-join (or vice-versa). A SQL join is usually quicker but, as we've discussed many times before, there is usually more than one way to perform any given SQL task.

A query in a query?

The subquery is fairly straightforward part of the SQL specification. In a nutshell, a subquery is a SQL SELECT statement that is placed in the predicate of any other SQL statement we've explored -- SELECT, INSERT, UPDATE, or DELETE. You're quite smart enough on your own to figure them out without my intervention, but we'll cover them here to make sure we've hit everything!

A subquery can be used in a number of scenarios:

- SELECT/UPDATE/DELETE .... WHERE (SELECT ...) which can be used to filter data before an action is applied to the results of that filter;
- INSERT INTO.... SELECT .... which can be used to copy tables or portions of tables into a new table for further manipulation;
- Another subquery which can then be nested again up to the limits of your database platform -- or your sanity and understanding.

You've probably found yourself at various points in SQL development mentally creating subqueries in your head -- things like finding all of your high-volume customers who are also the ones that pay on time or maybe updating information about all the employees that are also managers in the company. Every major RDBMS lets you do at least some level of subquerying to address that exact issue.

Using a subquery

Let's say we want to find the names of all of the managers in the Employees table. Starting with the following table

<table>
<thead>
<tr>
<th>Employees</th>
<th>EmployeeID</th>
<th>EmployeeName</th>
<th>ManagerID</th>
</tr>
</thead>
</table>

Do you have questions or comments for John Paul Ashenfelter about subqueries?

Post your comments
we want to first select all of the values for `ManagerID` and then associate them with a name. We can do that with the following set of queries

```
SELECT EmployeeName AS Employee
FROM Employees
WHERE EmployeeID IN (SELECT DISTINCT ManagerID FROM Employees)
```

The queries are addressed from the inside out, so the first step is to perform the statement

```
SELECT ManagerID FROM Employees
```

which returns the result set (61, 63). This means the outer query becomes

```
SELECT EmployeeName AS Employee
FROM Employees
WHERE EmployeeID IN (61, 63)
```

which then gives us the record set ("Sue Smith", "Troy Parker").

The only caveat with subqueries is that you must be aware of exactly what the query will return as far as fields, field types, and values. Our subquery in this example returns a single column of values which are valid values for the `WHERE EmployeeID IN` clause. If the subquery returned the employee name, for example, you'd get a data type mismatch because "Sue Smith" is a string and `EmployeeID` is a numeric field -- `WHERE EmployeeID IN ('Sue Smith'....)` makes no sense. This caveat is especially true when you are using a `SELECT` subquery inside of an `INSERT INTO` statement -- both the number of fields, their order, and the data types must match up or the `INSERT` will fail.

## Next steps

This article on subqueries is the final article in our introduction to the core SQL statements for manipulating data. Later, we'll move into SQL that manipulates the database itself, but before we do that, I'll devote the next several columns to the world of set algebra and the SQL commands relating to the `UNION` and `INTERSECTION` statements. Until then, feel free to contact me with your comments and questions.