CHAPTER 17

Introduction to Subqueries

About this chapter

This chapter shows how to use the results of one query as part of another SELECT statement. This is a useful tool in building more complex and informative queries.

Contents

Preparing to use subqueries

Sometimes it is useful to use the results of one statement as part of another statement.

Example 1

For example, suppose that you need a list of order items for products that are low in stock.

You can look up the products for which there are less than 20 items in stock in the **product** table.

❖ **To list all products for which there are less than 20 items in stock:**

♦ Type the following:

```
SELECT id, description, quantity
FROM product
WHERE quantity < 20
```

id	description	quantity
401	Wool cap	12

This query shows that only wool caps are low in stock.

Example 2

You can list all the order items for wool caps with the following query:

❖ **To list all orders for wool caps, most recent first:**

♦ Type the following:

```
SELECT *
FROM sales_order_items
WHERE prod_id = 401
ORDER BY ship_date DESC
```

id	line_id	prod_id	quantity	ship_date
2082	1	401	48	1994-07-09
2053	1	401	60	1994-06-30
2125	2	401	36	1994-06-28
2027	1	401	12	1994-06-17
2062	1	401	36	1994-06-17

This two-step process of identifying items low in stock and identifying orders for those items can be combined into a single query using subqueries.

A simple subquery

SQL provides another way to find orders for items low in stock. The following query incorporates a **subquery**.

Example 1

❖ **To list order items for products low in stock:**

♦ Type the following:

```
SELECT *
FROM sales_order_items
WHERE prod_id IN
   (  SELECT id
      FROM product
      WHERE quantity < 20 )
ORDER BY ship_date DESC
```

id	line_id	prod_id	quantity	ship_date
2082	1	401	48	1994-07-09
2053	1	401	60	1994-06-30
2125	2	401	36	1994-06-28
2027	1	401	12	1994-06-17
2062	1	401	36	1994-06-17

By using a subquery, the search can be carried out in just one query, instead of using one query to find the list of low-stock products and a second to find orders for those products.

The subquery in the statement is the phrase enclosed in parentheses:

```
(  SELECT id
   FROM product
   WHERE quantity < 20 )
```

The subquery makes a list of all values in the **id** column in the **product** table satisfying the WHERE clause search condition.

Example 2

Consider what would happen if an order for ten tank tops were shipped so that the **quantity** column for tank tops contained the value 18. The query using the subquery, would list all orders for both wool caps and tank tops. On the other hand, the first statement you used would have to be changed to the following:

```
SELECT *
FROM sales_order_items
WHERE prod_id IN ( 401, 300 )
ORDER BY ship_date DESC
```

The command using the subquery is an improvement because it still works even if data in the database is changed.

Example 3

As another example, you can list orders for everything *except* those products in short supply with the query:

```
SELECT *
FROM sales_order_items
WHERE prod_id NOT IN
   (  SELECT id
      FROM product
      WHERE quantity < 20 )
ORDER BY ship_date DESC
```

Comparisons using subqueries

Two tables in the sample database are concerned with financial results. The **fin_code** table is a small table holding the different codes for financial data and their meanings:

❖ **To list the contents of the fin_code table:**

♦ Type the following:

```
SELECT *
FROM fin_code
```

code	type	description
e1	expense	Fees
e2	expense	Services
e3	expense	Sales & Marketing
e4	expense	R&D
e5	expense	Administration
r1	revenue	Fees
r2	revenue	Services

The **fin_data** table holds financial data for each financial code for each quarter.

❖ **To list the contents of the fin_data table:**

♦ Type the following:

```
SELECT *
FROM fin_data
```

year	quarter	code	amount
1992	Q1	e1	101
1992	Q1	e2	403
1992	Q1	e3	1437
1992	Q1	e4	623
1992	Q1	e5	381

The following query uses a subquery to list just the revenue items from the **fin_data** table.

❖ **To list the revenue items from the fin_data table:**

♦ Type the following:

```
SELECT *
FROM fin_data
WHERE fin_data.code IN
     ( SELECT fin_code.code
       FROM fin_code
       WHERE type = 'revenue' )
```

year	quarter	code	amount
1992	Q1	r1	1023
1992	Q2	r1	2033
1992	Q3	r1	2998
1992	Q4	r1	3014
1993	Q1	r1	3114

This example has used qualifiers to clearly identify the table to which the **code** column in each reference belongs. In this particular example, the qualifiers could have been omitted.

Notes about subqueries

Subqueries are restricted to one column name listed between SELECT and FROM: one select-list item. The following example does not make sense, since SQL would not know which column from **fin_code** to compare to the **fin_data.code** column.

```
SELECT *
FROM fin_data
WHERE fin_data.code IN
    (  SELECT fin_code.code, fin_code.type
       FROM fin_code
       WHERE type = 'revenue' )
```

Further, while subqueries used with an IN condition may return several rows, a subquery used with a comparison operator must return only one row. For example the following command results in an error since the subquery returns two rows:

```
SELECT *
FROM fin_data
WHERE fin_data.code =
    (  SELECT fin_code.code
       FROM fin_code
       WHERE type = 'revenue' )
```

The IN comparison allows several rows. Two other keywords can be used as qualifiers for operators to allow them to work with multiple rows: ANY and ALL.

The following query is identical to the successful query above:

```
SELECT *
FROM fin_data
WHERE fin_data.code = ANY
   (  SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```

While the = *ANY* condition is identical to the IN condition, ANY can also be used with inequalities such as, or, to give more flexible use of subqueries.

The word ALL is similar to the word ANY. For example, the following query lists financial data that is not revenues:

```
SELECT *
FROM fin_data
WHERE fin_data.code <> ALL
   (  SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```

This is equivalent to the following command using NOT IN:

```
SELECT *
FROM fin_data
WHERE fin_data.code NOT IN
   (  SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```

Using subqueries instead of joins

Suppose you need a chronological list of orders and the company that placed them, but would like the company name instead of their customer ID. You can get this result using a join as follows:

Using a join

❖ **To list the order id, date, and company name for each order since the beginning of 1994:**

♦ Type the following:

```
SELECT sales_order.id,
      sales_order.order_date,
      customer.company_name
FROM sales_order
   KEY JOIN customer
WHERE order_date > '1994/01/01'
ORDER BY order_date
```

id	order_date	company_name
2473	1994-01-04	Peachtree Active Wear
2474	1994-01-04	Sampson & Sons
2036	1994-01-05	Hermanns
2106	1994-01-05	Salt & Pepper's
2475	1994-01-05	Cinnamon Rainbow's

Using a subquery

The following statement obtains the same results using a subquery instead of a join:

```
SELECT sales_order.id,
   sales_order.order_date,
   (  SELECT company_name FROM customer
      WHERE customer.id = sales_order.cust_id )
FROM sales_order
WHERE order_date > '1994/01/01'
ORDER BY order_date
```

The subquery refers to the **cust_id** column in the **sales_order** table even though the **sales_order** table is not part of the subquery. Instead, the **sales_order.cust_id** column refers to the **sales_order** table in the main body of the statement. This is called an **outer reference**. Any subquery that contains an outer reference is called a **correlated subquery**.

A subquery can be used instead of a join whenever only one column is required from the other table. (Recall that subqueries can only return one column.) In this example, you only needed the **company_name** column so the join could be changed into a subquery.

If the subquery might have no result, this method is called an **outer join**. The join in previous sections of the tutorial is more fully called an **inner join**.

Using an outer join

❖ **To list all customers in Washington State together with their most recent order ID:**

♦ Type the following:

```
SELECT company_name, state,
    (  SELECT MAX( id )
       FROM sales_order
       WHERE sales_order.cust_id = customer.id )
FROM customer
WHERE state = 'WA'
```

company_name	state	MAX(id)
Custom Designs	WA	2547
It's a Hit!	WA	(NULL)

The **It's a Hit!** company placed no orders, and the subquery returns NULL for this customer. Companies who have not placed an order would not be listed if an inner join was used.

You could also specify an outer join explicitly. In this case a GROUP BY clause is also required.

```
SELECT company_name, state,
   MAX( sales_order.id )
FROM customer
   KEY LEFT OUTER JOIN sales_order
WHERE state = 'WA'
GROUP BY company_name, state
```