Perspective

Subqueries are a powerful tool that you can use to solve difficult problems. Before you use a subquery, however, remember that a subquery can often be restated more clearly by using a join. If so, you'll typically want to use a join instead of a subquery.

If you find yourself coding the same subqueries in multiple places, you should consider creating a view for that subquery as described in chapter 12. This will help you develop queries more quickly since you can use the view instead of coding the subquery again. In addition, since views typically execute more quickly than subqueries, this may improve the performance of your queries.

Terms

subquery
introduce a subquery
nested subquery
correlated subquery
uncorrelated subquery
inline view
comment
pseudocode
common table expression (CTE)
recursive query
recursive CTE

Exercises

1. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

```
SELECT DISTINCT vendor_name
FROM vendors JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
ORDER BY vendor_name
```

2. Write a SELECT statement that answers this question: Which invoices have a payment total that's greater than the average payment total for all invoices with a payment total greater than 0?

   Return the invoice_number and invoice_total columns for each invoice. This should return 20 rows.

   Sort the results by the invoice_total column in descending order.

3. Write a SELECT statement that returns two columns from the General_Ledger_Accounts table: account_number and account_description.

   Return one row for each account number that has never been assigned to any line item in the Invoice_Line_Items table. To do that, use a subquery introduced with the NOT EXISTS operator. This should return 54 rows.

   Sort the results by the account_number column.

4. Write a SELECT statement that returns four columns: vendor_name, invoice_id, invoice_sequence, and line_item_amount.

   Return a row for each line item of each invoice that has more than one line item in the Invoice_Line_Items table. *Hint: Use a subquery that tests for invoice_sequence > 1.* This should return 6 rows.

   Sort the results by the vendor_name, invoice_id, and invoice_sequence columns.

5. Write a SELECT statement that returns two columns: vendor_id and the largest unpaid invoice for each vendor. To do this, you can group the result set by the vendor_id column. This should return 7 rows.

   Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return a single value that represents the sum of the largest unpaid invoices for each vendor.

6. Write a SELECT statement that returns the name, city, and state of each vendor that's located in a unique city and state. In other words, don't include vendors that have a city and state in common with another vendor. This should return 38 rows.

   Sort the results by the vendor_state and vendor_city columns.

7. Use a correlated subquery to return one row per vendor, representing the vendor's oldest invoice (the one with the earliest date). Each row should include these four columns: vendor_name, invoice_number, invoice_date, and invoice_total. This should return 34 rows.

   Sort the results by the vendor_name column.

8. Rewrite exercise 7 so it gets the same result but uses an inline view instead of a correlated subquery.

9. Rewrite exercise 5 so it uses a common table expression (CTE) instead of an inline view.