**Using Subqueries in the WHERE Clause**

Let us consider the scenarios as supported by the Payables database. One simple example is that we need to know names of all the vendors in the same state as one specific company, BFI Industries. We can easily plan out a query in the form of

/\* Q12.1a \*/

**SELECT Name**

**FROM Vendors**

**WHERE State =** <theState>**;**

However, without the knowledge of the value to match (theState) in the WHERE clause, the query as listed above cannot be executed. Actually, that value can be retrieved by another SELECT statement in the same pattern:

/\* Q12.1b \*/

**SELECT State**

**FROM Vendors**

**WHERE Name = 'BFI Industries';**

The resultant query (with a subquery embedded) is shown as Q12.1c, with result set returned from SQL Server shown in Figure 12.2.

/\* Q12.1c \*/

**SELECT Name**

**FROM Vendors**

**WHERE State = (**

**SELECT State**

**FROM Vendors**

**WHERE Name = 'BFI Industries'**

**);**

**Subquery Returning One Single Value.** Let us consider another scenario, where we will need to retrieve number of all invoices with an invoice total amount that is higher than the average total amount of all of the invoices. Due to the nature of aggregate functions, only one value will be returned for the whole group of entries.

Again, we can break this into two steps:

* First, we will use a (sub)query to retrieve the average total amount;
* Then, we will embed the subquery into a main query to retrieve the invoice numbers as required.

The subquery will use an aggregate function AVG() for calculating the average value, as we learned in Chapter 7. As shown in Figure 12.3, the average amount is retrieved with Q12.2a as 1545.4426.

/\* Q12.2a \*/

**SELECT AVG(InvoiceTotal) AS AverageTotalAmount**

**FROM Invoices;**

When using Q12.2a in a main query (Q12.2b), a greater than operator (>) can be used to express the desired criterion and the required invoice numbers returned.

/\* Q12.2b \*/

**SELECT InvoiceNumber, InvoiceTotal**

**FROM Invoices**

**WHERE InvoiceTotal >**

**(SELECT AVG(InvoiceTotal)**

**FROM Invoices);**

The InvoiceTotal values are also included for an easy check of the results. (An interested reader may want to add an ORDER BY clause to sort the result set so as to check the comparison more easily.)

**Subquery Returning Multiple Values.** Let us assume now that we are interested in knowing which vendors have made invoices after a specific vendor, Wang Laboratories, Inc. made its invoices. To follow the strategy from previous examples, we can

* Use a subquery to retrieve dates of Wang Laboratories invoices; and
* Use the subquery in the main query for comparison.

For simplicity, we will use the specific vendor’s ID, which is kept in the Invoices table to avoid the need for table join. An alias, Vendor81Date is used to emphasize the meaning of the output.

/\* Q12.3a \*/

**SELECT InvoiceDate AS Vendor81Date**

**FROM Invoices**

**WHERE VendorID = 81**

**ORDER BY 1;**

As shown in Figure 12.5, the result set includes 4 invoice dates, which are sorted for clarity.

When using Q12.3a into a main query, we mean to retrieve those invoices with a “later” date, which conceptually may be expressed with InvoiceDate > Vendor81Date. Careful readers may already have a question: which date? There are 4 of them! It is logically ambiguous. If we use the subquery as shown in Figure 12.5, the query will fail and an error message will be displayed.

The right way of forming a valid comparison is to use an addition keyword, either ALL or ANY, in front of the greater-than-sign (>) or another comparison operator, such as equal-sign (=). When ALL is used (as in Q12.3b), only invoices with dates later than *all* the four Vendor81Date’s will be retrieved. Actually, using the ALL keyword in this case is equivalent to say “greater than the latest Vendor81Date”, or 2010-09-14 (Figure 12.5).

/\* Q12.3b \*/

**SELECT InvoiceNumber, InvoiceDate**

**FROM Invoices**

**WHERE InvoiceDate >**

**ALL (SELECT InvoiceDate**

**FROM Invoices**

**WHERE VendorID = 81);**