

Discussion

Calling functions from queries opens up many ways of working with your data. The function used in this example is short and processes an easy calculation. But do note that this function actually calls another function, `Weekday`. The point is that you can develop sophisticated functions that perform all types of processing.

Both custom functions and built-in functions can be called from a query. Further, the query does not have to be a select query; you can call functions from action queries (discussed in Chapter 3) as well. Figure 2-14 shows an example of using two built-in functions, `IIf` and `Weekday`, to update the value in a field. If the date falls on the weekend, `-Weekend` is appended to the client name.

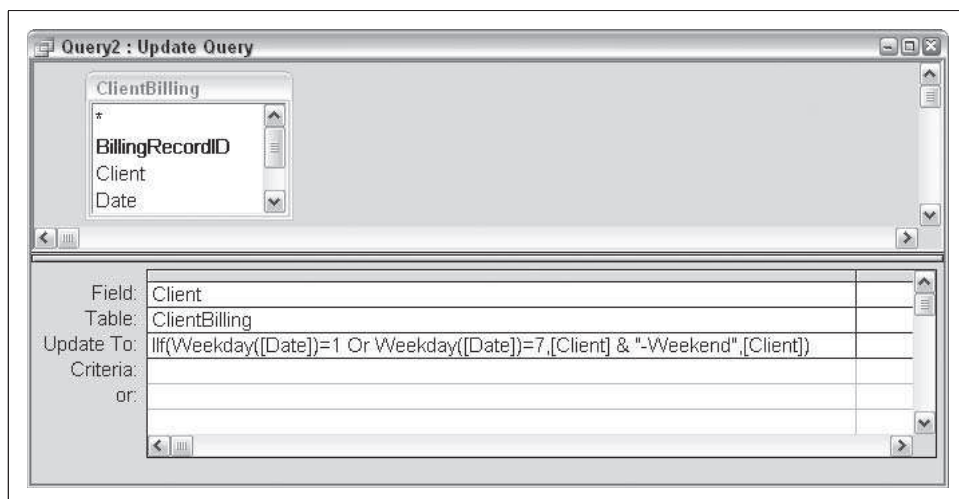


Figure 2-14. Built-in functions used in an update query

Take a closer look at the expression used in the Update To row:

```
IIf(Weekday([Date])=1 Or Weekday([Date])=7,[Client] & "-Weekend",[Client])
```

If the weekday is a 1 or a 7 (a Sunday or a Saturday), `-Weekend` is appended to the client name; otherwise, the client name is used for the update as is. In other words, every row is updated, but most rows are updated to the existing value.

2.5 Using Regular Expressions in Queries

Problem

Regular expressions provide the ability to set up sophisticated string patterns for matching records. Can a regular expression be used as the criterion in a query?

Solution

Regular expressions, popularized by Perl and other Unix-based languages, provide powerful string-matching capabilities. For Access and other Windows-based applications, adding a reference to the VBScript Regular Expressions library makes using regular expressions possible. You can set up the reference in the Visual Basic Editor (VBE). To display the VBE from Access, just press Alt-F11. While in the VBE, use the Tools → References menu option to display the References dialog. Then set a reference to Microsoft VBScript Regular Expressions 5.5 (your version number may be different), as shown in Figure 2-15.

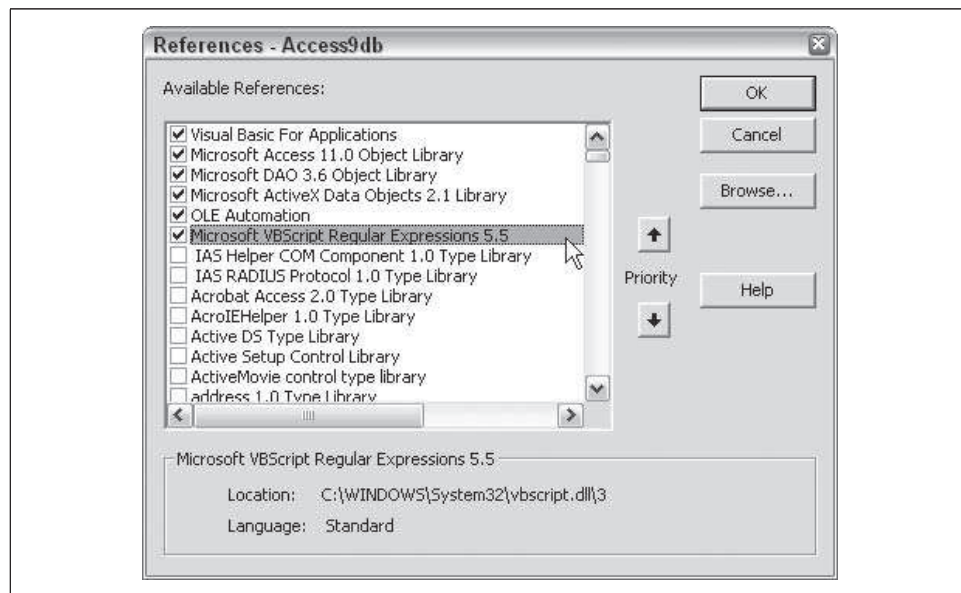


Figure 2-15. Setting a reference to the VBScript Regular Expressions library

Figure 2-16 shows a table with hypothetical transaction records.

The values in the TransactionRecord field are concatenations of separate codes and values that follow this pattern:

- The first character is a letter that signifies a type of transaction. For example, an A could mean an adjustment, and a D could mean a deposit.
- The next two characters are numbers that represent a transaction type.
- The next two characters are a department code. For example, LE is Legal, SA is Sales, HR is Human Resources, WA is Warehouse, and so on.
- The last three numbers are an amount.

TransactionRecord
A49CO409
D97HR309
R44AD833
F29CO480
S52WA810
A68WA163
S51LE727
A07SA808
R75LE898
S81HR505
A89SA624
A61CO545
F97AD289
R51SA149
D50WA419
F90AD693

Figure 2-16. A table with transaction records

Given this pattern, let's construct a query that will identify transactions that are either adjustments or deposits, are for the Sales department, and are for amounts of 500 or greater. Put another way, our task will be to identify transaction records consisting of an A or a D, followed by any two numbers, followed by SA, followed by a number that is 5 or greater, and any two other numbers.

To make use of pattern matching, we'll use a custom function to create a regular expression (regex) object. In a separate code module, enter this function:

```
Function validate_transaction(transaction_record As String, _
    match_string As String) As String
    validate_transaction = "Invalid Record"
    Dim regexp As regexp
    Set regexp = New regexp
    With regexp
        .Global = True
        .IgnoreCase = True
        .Pattern = match_string
        If .Test(transaction_record) = True Then
            validate_transaction = "Valid Record"
        End If
    End With
    Set regexp = Nothing
End Function
```

The transaction record and the regular expression pattern to match are given to the function as arguments. The function code initially sets the return value to Invalid Record. A regexp object is set, and the transaction record is tested against the pattern. If the transaction record matches the pattern, the return value is changed to Valid Record.

Now, we'll call this function from a query. Figure 2-17 shows a select query that places the call to the function in the second column. Note that the column with the function to call is set to a descending sort. This is so all valid records will appear at the top. Unlike a typical select query that returns just the records that match the criteria, in this case all records are returned, so it makes sense to group together all the valid records.

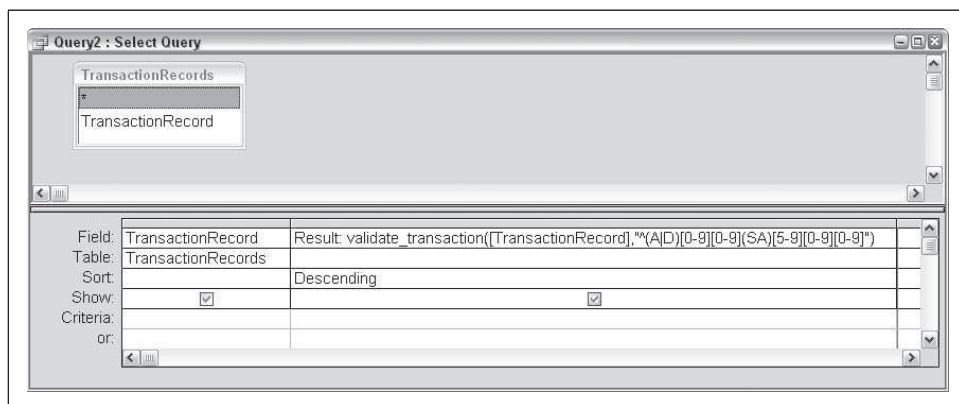


Figure 2-17. A regular expression pattern coded into a query

Figure 2-18 shows the result of running the query. The valid records are at the top.

Discussion

The pattern used for matching the transaction records look like this:

```
^(A|D)[0-9][0-9](SA)[5-9][0-9][0-9]
```

A detailed explanation of regular expression syntax is beyond the scope of this recipe, but in a nutshell, here is what the pattern calls for:

- (A|D) indicates to match an A or a D.
- [0-9] indicates to match any single numerical digit between 0 and 9 (effectively, any number).
- (SA) indicates to match SA.
- [5-9] indicates to match any numerical digit between 5 and 9. Since this is followed in the pattern by two more occurrences of [0-9] (i.e., any number), taken together, a value of 500 or greater is sought.

TransactionRecord	Result
A36SA646	Valid Record
A23SA718	Valid Record
D10SA788	Valid Record
D27SA705	Valid Record
A30SA808	Valid Record
A98SA779	Valid Record
A89SA586	Valid Record
D33SA890	Valid Record
D55SA842	Valid Record
A89SA624	Valid Record
A44SA620	Valid Record
D82SA517	Valid Record
A07SA608	Valid Record
R82LE703	Invalid Record
S17AD772	Invalid Record
R10AD572	Invalid Record
S25WA851	Invalid Record
R72CO833	Invalid Record
F20HR427	Invalid Record

Figure 2-18. Records are validated using a regular expression

Test is just one method of the regexp object. There also are Execute and Replace methods. Execute creates a collection of matches. This is useful in code-centric applications, where further processing would take place in a routine. The Replace method replaces a match with a new string.

See Also

- Recipe 2.4
- *Mastering Regular Expressions* by Jeffrey E. F. Friedl (O'Reilly)
- *Regular Expression Pocket Reference* by Tony Stubblebine (O'Reilly)

2.6 Using a Cartesian Product to Return All Combinations of Data

Problem

I have a list of teams and a list of locations with ballparks. I wish to create a master list of all the combinations possible between these two lists.