Macro Magic

Macros allow you to integrate existing Excel reports with a new information system

By Rick Collard

Many water and wastewater professionals use Microsoft Excel extensively, producing reports and graphs that summarize lab and operations data. Some of these Excel reports may be simple one-sheet tabular layouts while others may be elaborate multi-sheet reports with graphs. Over the years, a water or wastewater department may come to rely on many Excel-based reports both for internal purposes and to submit data to regulatory agencies. When the department considers the advantages of a new information system, such as a laboratory information management system (LIMS), the notion of abandoning their familiar reports produces understandable reservations.

During nearly a decade implementing MSC-LIMS, our firm's LIMS software, we have encountered this scenario many times. Fortunately, there is a solution.

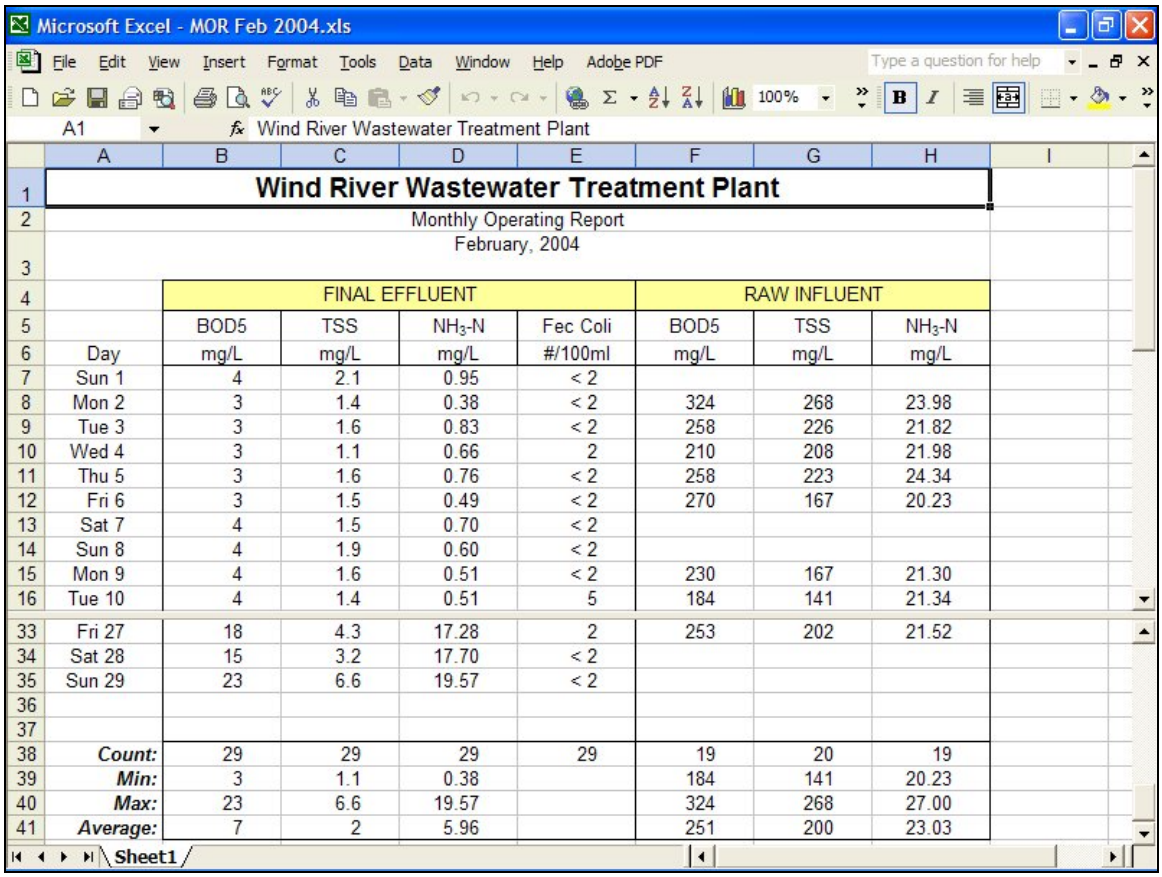
This article demonstrates how existing Excel reports can be integrated with new information systems while completely preserving the report's layout and appearance. Using a combination of Excel formulas, lookup tables, and even a little Visual Basic for Applications (VBA) code, step by step instructions show you how to add the necessary infrastructure to turn an existing Excel report into a template that will automate report production. By gathering the necessary data from the new information system, Excel reports that were formerly completed manually can now be produced automatically with the click of a button.

There are many advantages to leveraging existing Excel reports with this strategy. First, report recipients continue to receive information in formats already familiar and perhaps required. Second, no time is wasted attempting to reproduce existing reports in the new information system. Third, producing reports in an electronic Excel format preserves the option for further data analysis, which may be one of the reasons Excel was originally chosen. Fourth, many users are already comfortable with Excel so report maintenance is simplified and new reports are easily created. Finally, the techniques presented in this article can be used with a number of information systems including LIMS, SCADA, operations data systems, and even simple Excel-based data systems.

A Simple Tabular Report Example

Let's begin by examining the simple tabular report shown in Figure 1. This example is an excerpt of a wastewater treatment plant's Monthly Operating Report, simplified by including only partial data for the plant's final effluent and raw influent. In practice the report would likely include additional plant locations and results either in more columns or on additional worksheets.

This report structure is commonly used in water and wastewater plant operations. Each row lists the analytical and operational results for various plant processes or locations for one day. The report presents data for one complete month with appended summary statistics. Manually completing this report at month's end may require gathering data from many sources.



Microsoft Excel - MOR Feb 2004.xls

File Edit View Insert Format Tools Data Window Help Adobe PDF

A1 Wind River Wastewater Treatment Plant

	A	B	C	D	E	F	G	H
1	**Wind River Wastewater Treatment Plant**							
2	Monthly Operating Report							
3	February, 2004							
4		FINAL EFFLUENT				RAW INFLUENT		
5		BOD5	TSS	NH_3-N	Fec Coli	BOD5	TSS	NH_3-N
6	Day	mg/L	mg/L	mg/L	#/100ml	mg/L	mg/L	mg/L
7	Sun 1	4	2.1	0.95	< 2			
8	Mon 2	3	1.4	0.38	< 2	324	268	23.98
9	Tue 3	3	1.6	0.83	< 2	258	226	21.82
10	Wed 4	3	1.1	0.66	2	210	208	21.98
11	Thu 5	3	1.6	0.76	< 2	258	223	24.34
12	Fri 6	3	1.5	0.49	< 2	270	167	20.23
13	Sat 7	4	1.5	0.70	< 2			
14	Sun 8	4	1.9	0.60	< 2			
15	Mon 9	4	1.6	0.51	< 2	230	167	21.30
16	Tue 10	4	1.4	0.51	5	184	141	21.34
33	Fri 27	18	4.3	17.28	2	253	202	21.52
34	Sat 28	15	3.2	17.70	< 2			
35	Sun 29	23	6.6	19.57	< 2			
36								
37								
38	***Count:***	29	29	29	29	19	20	19
39	***Min:***	3	1.1	0.38		184	141	20.23
40	***Max:***	23	6.6	19.57		324	268	27.00
41	***Average:***	7	2	5.96		251	200	23.03

Sheet1

Figure 1 Existing Monthly Operating Report

Gather the Data

One of the major components of any information system is a central data repository or *database*. Whether the database is an industry standard relational database management system or a proprietary data storage system, it helps users organize, maintain, secure and analyze important data. A fundamental benefit of any database is its ability to store a data value once and then employ that value wherever necessary, thereby avoiding redundant data entry and its potential for error. Using a LIMS example, the result for BOD at final effluent for a sample collected on February 1, 2004 (cell B7 in Figure 1) should be either manually entered or imported into the LIMS only once. The result value may then appear on many screens and reports as needed without additional data entry.

Once our data safely resides in the database, we can put it to work. To get the data into our Excel report we can use one of two options. We can either *pull* the data into the Excel workbook or we can *push* the data from the database. That is, the process is driven

either from Excel or from the information system. Sometimes a combination of push and pull may be necessary. The technique you choose may depend on the organization and structure of your information system's database, the availability of suitable query and data export tools, or other factors. For example, users of our MSC-LIMS software take advantage of its query-by-example (QBE) and Structured Query Language (SQL) tools to query the required data and push it directly to their chosen Excel template. If your information system provides similar tools, you may find them simpler than building your own query interface in Excel.

Excel includes very capable data query tools (see the Import External Data item on Excel's Data menu for options). Other data access interfaces such as Data Access Objects (DAO) and ActiveX Data Objects (ADO) can also be used within VBA macros. Other Excel files and any data source supporting the Open Database Connectivity (ODBC) standard can also be queried. We will use DAO to pull the data into our Excel workbook example.

To begin, save the Excel workbook shown in Figure 1 as a new Excel template file (*.xlt). Our Excel template is actually an empty report, from which we will derive each month's report. Create the template and then delete the existing data, preserving all labels, formatting, and formulas. Now rename the report worksheet MOR, and add two new worksheets to the template: Query and LookupTable. The Query sheet will provide our user interface to query the required data, which will be added to the LookupTable sheet.

There are three variables required to locate the value for cell B7 in Figure 1: the analyte (BOD), the location (Final Effluent), and the collected date (2/1/2004). Since all of the results on our report follow this convention, we will construct a three-variable lookup table.

Admittedly, you may need the assistance of your information system's vendor to construct a suitable query using the appropriate data access interface with proper database connection and query syntax. However, after this step has been successfully completed for one report, only simple modifications should be necessary for other reports.

On the new Query worksheet (Figure 2), designate cells where users will enter the date range of the data to be queried and name them "StartDate" and "EndDate" respectively (Insert | Name | Define). Excel's named ranges provide a descriptive name to a cell or range of cells. Using named ranges will help make our formulas more readable and keep our macro flexible. In fact, because our macro will retrieve the start and end dates on the Query sheet by name, no changes to VBA code are necessary if these cells are later moved.

Now add "Start Date:" and "End Date:" labels to adjacent cells and add instructions for users. While we could invoke a macro from Excel's Tools menu, using a command button instead keeps things simple and intuitive for report users. To add a command button Active X control (a graphic object placed on a worksheet and used to enter data or perform an action) select the command button option in Excel's Control toolbox (View | Toolbars | Control Toolbox), then click on the Query worksheet to add the new button.

Right click the new command button and choose Properties. Change the Caption property to “Generate Report”.

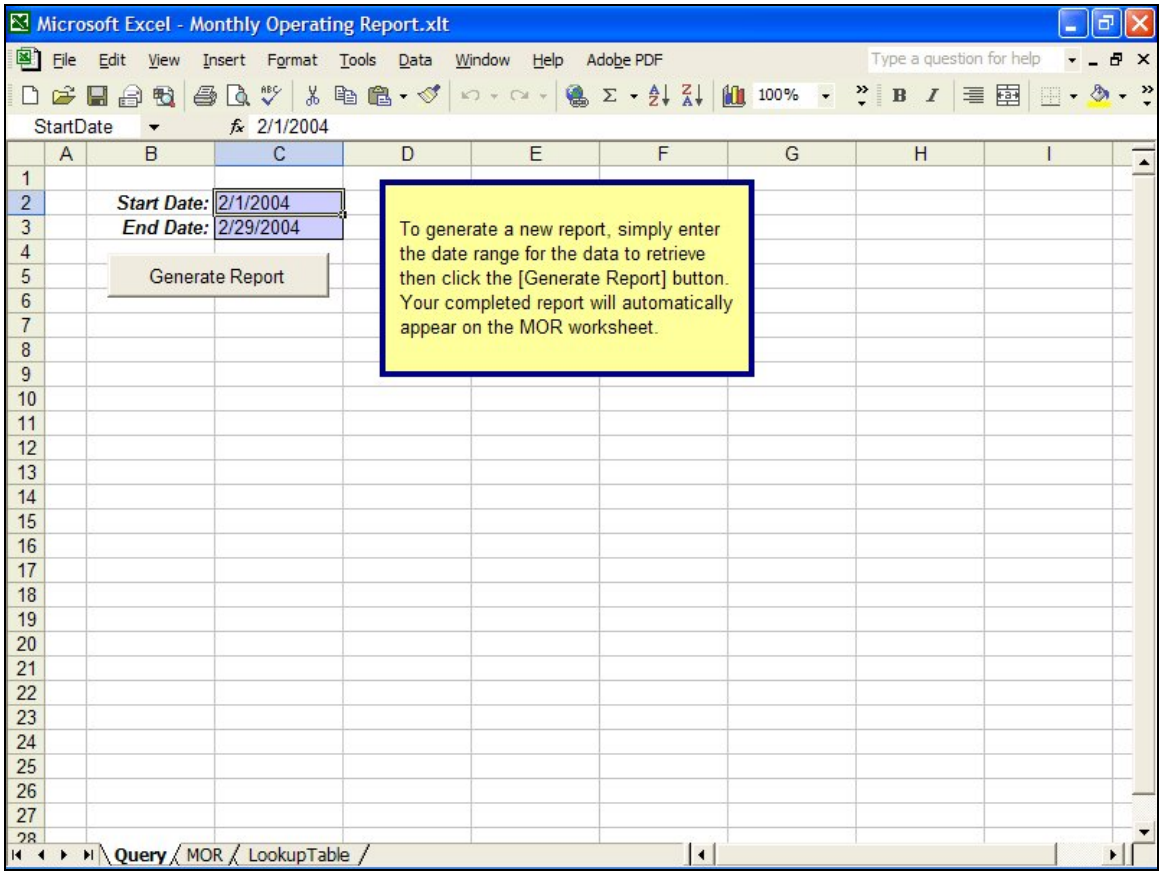


Figure 2 The New Query Worksheet

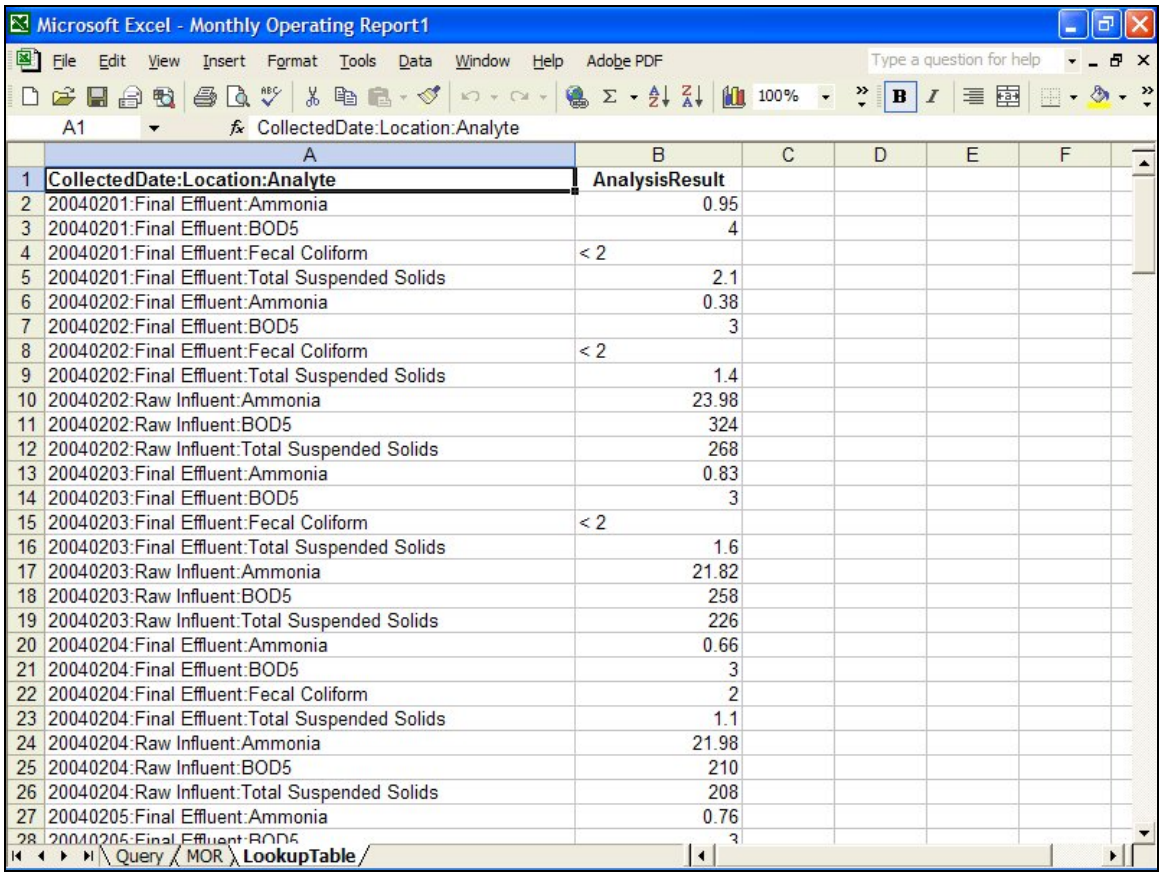
Now we need to associate a VBA macro to the command button’s click event. Right click the button and choose Properties again. Following standard VBA object naming conventions, change the Name property to “cmdGenerateReport”, which will help make our VBA code more readable. Now right-click the command button and choose View Code. Excel will open the VBA editor and automatically insert the following empty macro for the button’s default click event.

```
Private Sub cmdGenerateReport_Click()

End Sub
```

We can place whatever VBA code we need in the empty macro. In this instance, the single macro will do all the work necessary to produce our final report. Our example’s complete VBA macro is listed in Figure 5. Below, we will describe the code added to the macro, listing brief VBA code excerpts for each phase of the report generation process. Do not be discouraged if VBA syntax appears ominous. While some familiarity with Excel’s object model and the VBA programming language are required to directly enter and edit VBA code, you can use Excel’s macro recording feature to generate VBA code

that you can then copy and paste into your own macros. This is an excellent technique to create useable VBA code excerpts while learning VBA. Use Tools | Macro | Record New Macro to start recording a new macro, and then use Excel's user interface to perform whatever actions are required. Use Tools | Macro | Stop Recording then use Tools | Macro | Visual Basic Editor and you will find your recorded macro in a new Module in the Project Explorer window (View | Project Explorer).



	A	B	C	D	E	F
1	CollectedDate:Location:Analyte	AnalysisResult				
2	20040201:Final Effluent:Ammonia	0.95				
3	20040201:Final Effluent:BOD5	4				
4	20040201:Final Effluent:Fecal Coliform	< 2				
5	20040201:Final Effluent:Total Suspended Solids	2.1				
6	20040202:Final Effluent:Ammonia	0.38				
7	20040202:Final Effluent:BOD5	3				
8	20040202:Final Effluent:Fecal Coliform	< 2				
9	20040202:Final Effluent:Total Suspended Solids	1.4				
10	20040202:Raw Influent:Ammonia	23.98				
11	20040202:Raw Influent:BOD5	324				
12	20040202:Raw Influent:Total Suspended Solids	268				
13	20040203:Final Effluent:Ammonia	0.83				
14	20040203:Final Effluent:BOD5	3				
15	20040203:Final Effluent:Fecal Coliform	< 2				
16	20040203:Final Effluent:Total Suspended Solids	1.6				
17	20040203:Raw Influent:Ammonia	21.82				
18	20040203:Raw Influent:BOD5	258				
19	20040203:Raw Influent:Total Suspended Solids	226				
20	20040204:Final Effluent:Ammonia	0.66				
21	20040204:Final Effluent:BOD5	3				
22	20040204:Final Effluent:Fecal Coliform	2				
23	20040204:Final Effluent:Total Suspended Solids	1.1				
24	20040204:Raw Influent:Ammonia	21.98				
25	20040204:Raw Influent:BOD5	210				
26	20040204:Raw Influent:Total Suspended Solids	208				
27	20040205:Final Effluent:Ammonia	0.76				
28	20040205:Final Effluent:BOD5	3				

Figure 3 The LookupTable Worksheet After Querying Data

Our macro must first pull the necessary data from the database into a form our template can use. So that we can use Excel's VLOOKUP ("vertical" lookup) function to retrieve report data, we must create a single column for our three-variable lookup. This is easily accomplished by concatenating the three values into a single text value, separating the values by a colon (:) for better readability. In our example, the macro's first step is to query a LIMS database using the start and end dates entered into the cells on the Query sheet as criteria and copy the data to our LookupTable sheet beginning at cell A2 (see Figure 3). In practice, the macro should first verify that valid start and end dates were entered, but we've omitted that step to simplify our example. The following VBA code in our macro uses DAO to query data from an Access database:

```
Set db = DBEngine.OpenDatabase("C:\LabDatabase.mdb", False, True)
sStartDate = Worksheets("Query").Range("StartDate").Value
sEndDate = Worksheets("Query").Range("EndDate").Value
```

```
sSQL = "SELECT CollectedDate & ':' & Location & ':' " & _
       "Analyte, AnalysisResult " & _
       "FROM qryMORData " & _
       "WHERE CollectedDate BETWEEN " & _
       "#" & sStartDate & "# AND #" & sEndDate & "# " & _
       "ORDER BY CollectedDate, Location, Analyte;"
Set rs = db.OpenRecordset(sSQL, dbOpenDynaset, dbReadOnly)
Worksheets("LookupTable").Range("A2").CopyFromRecordset rs
```

The data retrieved by the SQL statement in the code above may include more data than is required by our report. The idea is to add additional query criteria to minimize the amount of data transferred while ensuring all required data is included in our LookupTable sheet. Notice that, in Figure 3, the dates in our database are retrieved using YYYYMMDD format since the VLOOKUP function requires an ordered list to search.

Next, our macro updates a LookupTable named range to encompass all of the populated cells on the LookupTable sheet using the following VBA code:

```
ActiveWorkbook.Names("LookupTable").RefersTo = _
    "=LookupTable!" & Worksheets("LookupTable").UsedRange.Address
```

Use Lookup Formulas to Insert Results

Now that our macro has retrieved the necessary data and our LookupTable named range refers to all of the data on the LookupTable sheet, we can use formulas to display results on our MOR report sheet. The general format of Excel's VLOOKUP function is:

```
VLOOKUP(lookup_value, table_array, col_index_num, range_lookup)
```

where lookup_value is the value to be found in the first column of the table; table_array is the table of information to search; col_index_num is the column number in table_array from which the matching value is returned; and range_lookup is a logical value that specifies whether you want VLOOKUP to find an exact match or an approximate match. Setting range_lookup to FALSE will find an exact match. In cell B7 on the MOR sheet, insert the following formula, which uses our three-variable lookup table to retrieve the final effluent BOD result for February 1, 2004:

```
=VLOOKUP("20040201:Final Effluent:BOD", LookupTable, 2, False)
```

The formula above looks for the value "20040201:Final Effluent:BOD" in the first column of our LookupTable. If an exact match is found the value from the second column, our result, is displayed in the cell.

This formula works quite well but there is room for improvement. What happens when we copy the formula to cell B8? We need to edit the formula and change the date value to 20040202. If we copy the formula to a cell in column C we need to change BOD to TSS. Copy the formula to a cell in column F and Final Effluent must be changed to Raw Influent.

To copy our VLOOKUP formula to other cells without editing, we need to construct a more general-purpose formula. Not only will this simplify development but template maintenance will be easier when inevitable changes are required. To make our formula generic we will obtain our three variables from adjacent cells. Column A on the MOR sheet contains dates so we can use its values in our formulas. All the values in a single column are for the same Location:Analyte pair so we insert a row which we will eventually hide (row 7 in Figure 4) to maintain these values. After inserting the row and adding the value "Final Effluent:BOD" in cell B7 our new general-purpose formula in cell B8 is:

```
=VLOOKUP(TEXT($A8, "yyyymmdd") & ":" & B$7, LookupTable, 2, False)
```

In the formula above, we have used the TEXT function to convert the date in the column A cell to YYYYMMDD format, which is then concatenated with a colon (:) and the Location:Analyte value. A mix of absolute and relative cell references allows this formula to be copied to other cells. We only need to ensure that our Location:Analyte pair exists in row 7. However, before we copy the formula there is still room for improvement. As it stands, our formula will display a zero if there is no value in the second column of our lookup table for the matching lookup value. Also, if the lookup value is not found in the lookup table the formula will display Excel's #N/A error value. To correct these deficiencies we can use a combination of Excel's IF, ISERROR, and ISBLANK functions:

```
=IF(ISERROR(expression), "",IF(ISBLANK(expression), "", expression))
```

In this formula, if the result of the expression is either an error or blank, an empty string ("") is displayed. Otherwise, the formula displays the result of the expression. Substituting our VLOOKUP expression three times in the formula produces the following ghastly looking but nevertheless complete formula:

```
=IF(ISERROR(VLOOKUP(TEXT($A8, "yyyymmdd") & ":" & B$7, LookupTable, 2,
False)), "",IF(ISBLANK(VLOOKUP(TEXT($A8, "yyyymmdd") & ":" & B$7,
LookupTable, 2, False)), "", VLOOKUP(TEXT($A8, "yyyymmdd") & ":" & B$7,
LookupTable, 2, False)))
```

Use Named Ranges for Flexibility

Because we want our report to be adaptable to any calendar month and because we will occasionally use the report for less than a complete month's worth of data, we only copy our lengthy VLOOKUP formula to cells in our row 8. We will make our VBA macro copy the formulas to consecutive rows below, one row for each date within the date range entered on the Query sheet. For ease and flexibility we created a OneDayResults named range that encompasses our cells A8 through H8 (see Figure 4). Just as we did earlier with the StartDate and EndDate ranges, we use the OneDayResults named range to allow us to later add additional columns to our report with no changes required to our macro. We simply adjust our OneDayResults range to include the additional columns. The following VBA code in our macro copies the OneDayResults range to consecutive rows for the date range queried:

```
nDays = Worksheets("Query").Range("EndDate").Value - _
        Worksheets("Query").Range("StartDate") + 1
Worksheets("MOR").Range("OneDayResults").Copy _
    Destination:=Worksheets("MOR").Range("OneDayResults").Resize(nDays)
```

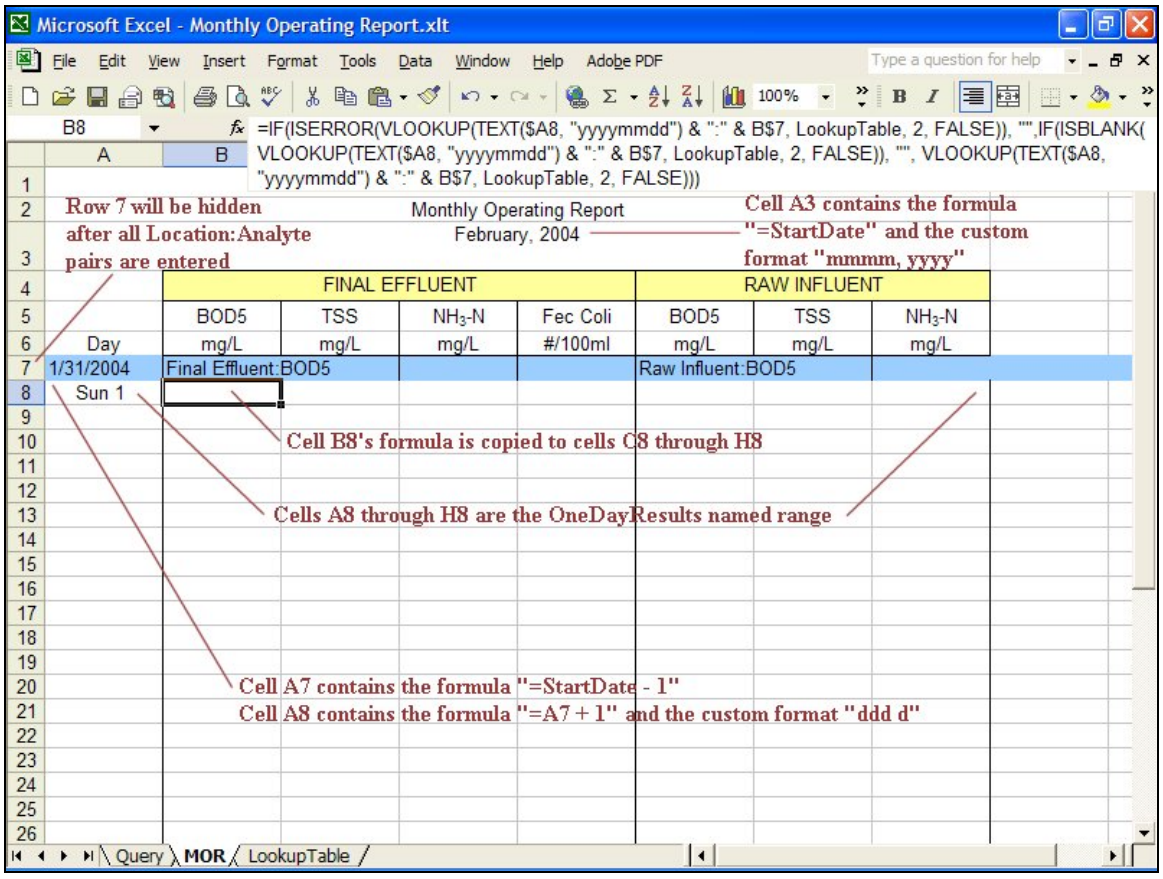


Figure 4 Adding Report Formulas

Our macro concludes with the commands below, which disable the Generate Report button and switch the workbook to the MOR sheet to display the finished report:

```
cmdGenerateReport.Enabled = False
Worksheets("MOR").Select
```

Our StartDate named range also comes in handy to display dates on the MOR sheet. In cell A7 in our hidden row we added the formula "=StartDate-1" which displays the day before the start date entered on the Query sheet. In cell A8, we added the formula "=A7+1", which falls within our OneDayResults range and will be copied by the macro to consecutive rows, to display consecutive dates for the entire date range queried.

Putting it all Together

With our template saved, report users need only double-click the template file in Windows Explorer to begin a new report. If the template was saved with the Query sheet selected, users will begin on the Query sheet. Users enter a date range and click the

"Generate Report" button, and the final report appears on the MOR sheet. What could be simpler?

Recall that our macro did all the work to generate the report using the following steps:
1. Query the database for the required data using the entered date range as criteria.
2. Write the data to the LookupTable worksheet and update the LookupTable named range so it includes all data rows.
3. Copy the OneDayResults named range on the MOR worksheet for each day on the report.
4. Disable the "Generate Report" button and switch to the MOR sheet.

The complete cmdGenerateReport_Click macro is listed in Figure 5. The macro could be extended to perform additional tasks. For example, the macro could replace all of the lengthy VLOOKUP formulas on the MOR sheet with their values using Excel's Copy and Paste Special features. Then, after the formulas have been replaced with values the macro could safely delete the Query and LookupTable sheets leaving only the final report. If the Query sheet is deleted the macro is also deleted, eliminating possible Excel security warnings when others open the completed report workbook.

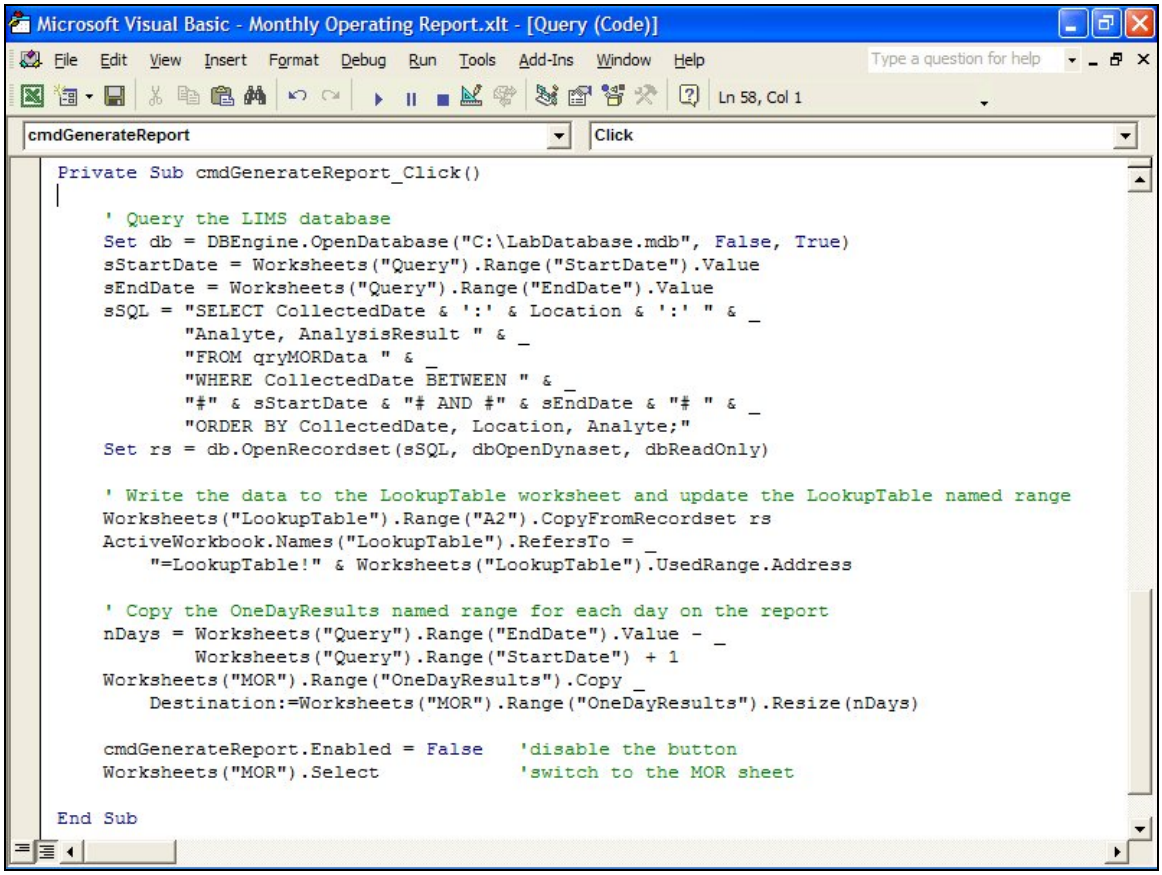


Figure 5 cmdGenerateReport_Click Macro

Conclusions

Expand on the techniques described in this article and you can automate more elaborate multi-sheet report and graph solutions. You will also discover you can use similar techniques to push data from existing data entry workbooks into your information system. If you have been manually completing your Excel reports, you will enjoy the significant benefits of automated report generation. Also, generating the reports at any time from data maintained in your new information system eliminates the need to archive monthly reports.

There is no need to waste time and energy reproducing your reports in a new environment, and you will avoid retraining the recipients of your reports because you have completely preserved their original layout and appearance.

Use the techniques described above and those trusted Excel reports you and your colleagues have created will continue to serve you well with your new information system. No longer must reluctance to abandon the tried and true present an obstacle to further productivity.

Rick Collard, *software engineer, is the founder of Mountain States Consulting LLC (Jackson, Wyoming), www.msc-lims.com. Reach him at 307-733-1442 or info@msc-lims.com.*

This article originally appeared in the September 2004 issue of *Water Environment & Technology* published by the Water Environment Federation® (Alexandria, Virginia), www.wef.org.