by Roger Jennings

TRACK CLIENT/SERVER COST-PERFORMANCE

A REALITY CHECK ON THE USAGE AND PERFORMANCE OF YOUR CLIENT/SERVER APPLICATIONS WILL PAY DIVIDENDS.

Most large firms that implement client/server database applications want answers to two fundamental questions: how many employees are using the application on a regular basis and how is the application performing? The answer to the first question provides management with the application cost per user. For example, if the application development cost was $250,000 and 500 people use it regularly, the cost per user is $500. That's not too much more than a couple copies of shrink-wrapped productivity software, so management is likely to be happy with a number in the range of $500 and under. Cost-per-user calculations do not take into account the productivity gains from the application, but productivity gains often are difficult to measure and the metrics usually are suspect.

Other cost savings, such as eliminating the printing and distribution of voluminous paper reports, can be more accurately quantified. Like it or not, you'll still find that management relies primarily on the cost per user to measure the success of a client/server database application. Down the road, MIS staff will want to know the support cost per user hit. This figure is calculated by counting the number of times per month users log into the database and run at least one query, then dividing the count into the monthly support cost. There is no generally accepted ballpark figure for support cost per user hit, but it should fall into the same range as support costs for the firm's other client/server applications.

Answering the second question presents a greater challenge. Assuming that the application provides users with the information they need in a usable format, performance measurement boils down to checking application operating speed. How long does the user have to wait to acquire and display the requested information? Overall productivity lags as a result of inordinate delays to log on to the database, return rows, perform inserts or updates, or format the information on the display. Users become reluctant to run the application and some give up in disgust, increasing the cost per user.

Performance problems usually are attributable to bad database design, an overloaded back-end server, slow front-end software or hardware, or network bottlenecks. Although a variety of general-purpose database and network performance analysis tools are available, these tools are costly and seldom provide the level of detail needed to accurately evaluate the performance of a specific client/server application. Fortunately, it's not too difficult to add a table, a form, and a few chunks of Visual Basic code to generate your own custom client/server performance monitor. A reality check on the usage and performance of your client/server application is likely to pay dividends in your own performance review.

National Semiconductor Corp. has implemented a method

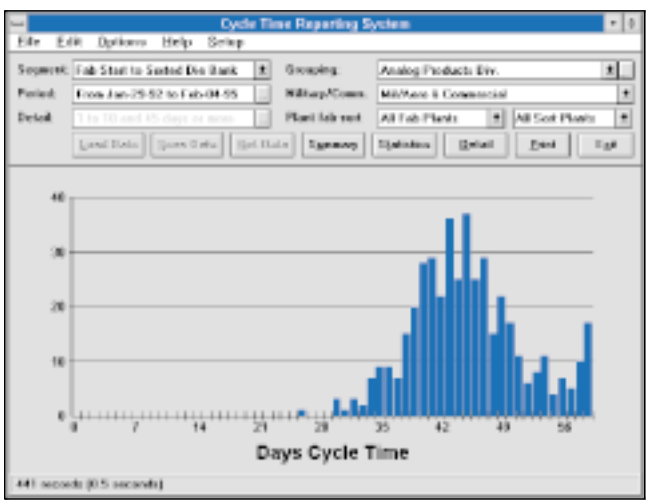
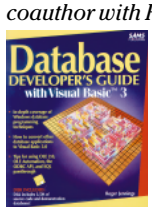


FIGURE 1 ***Take the Broad-Brush Approach.*** *CycleTime displays a histogram of the number of days necessary to process an order or to perform a step in the manufacturing and distribution process. This screen shot, which uses simulated data, shows the cycle time for semiconductor wafer fabrication (Fab Start) to the point where individual chips are cut from the wafer, tested, and sorted by quality criteria (Sorted Die Bank).*

Roger Jennings, a principal of OakLeaf Systems, is a consultant specializing in Windows client/server database front ends. He has more than 25 years of computer-related experience, and is the author of four programming books: Using Access 2 for Windows, Special Edition, *and* Discover Windows 3.1 Multimedia *for Que Books; and* Access 2 Developer's Guide *and* Database Developer's Guide with Visual Basic 3.0 *for Sams Publishing. He's also the coauthor with Peter Hipson of Sams'* Database Developer's Guide with Visual C++ 2.0. *Roger recently completed* Unveiling Windows 95 *for Que Books, is in the midst of writing* Using Windows Desktop Video *for Que, and is finishing the workflow-oriented* OLE 2 Developer's Guide with Visual Basic for Applications *for Sams. Reach him on CompuServe at 70233,2161.*



for figuring the cost per user and performance metrics for its CycleTime client/server application. If you've read my prior Database Design columns, you've seen descriptions of other components of this application and some of the Visual Basic code to implement specific features. CycleTime consists of a Visual Basic 3.0 front end to Sybase 10 running under AIX on an IBM RISC/6000 Unix box.

National's Larry Newcomer designed CycleTime to provide users with realtime workflow metrics for order processing and for the various steps involved in manufacturing tens of thousands of different semiconductor products. CycleTime displays a summary histogram chart of cycle times (see Figure 1), plus detailed data in spreadsheet format (see Figure 2). Creating the histogram requires an average of 40 two-column rows from the server. Users specify the categories of products and the time period for the display in specially designed dialogs. The detail sheet displays individual order line items or manufacturing/shipping meeting criteria set by the user.

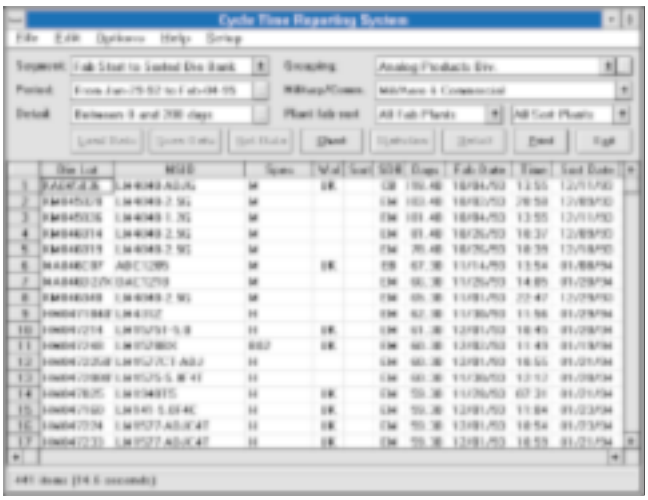


FIGURE 2 ***Display the Nitty-Gritties.*** *Users can choose the range of cycle times (in days) for the detail spreadsheet display of the records that underlie the histogram in Figure 1. If the user wants all of the records for a month or quarter, the number of rows might range in the thousands. The user is warned ahead of time how many rows the detail query will return. If it's more than 1000, an "Are you sure?" message box appears.*

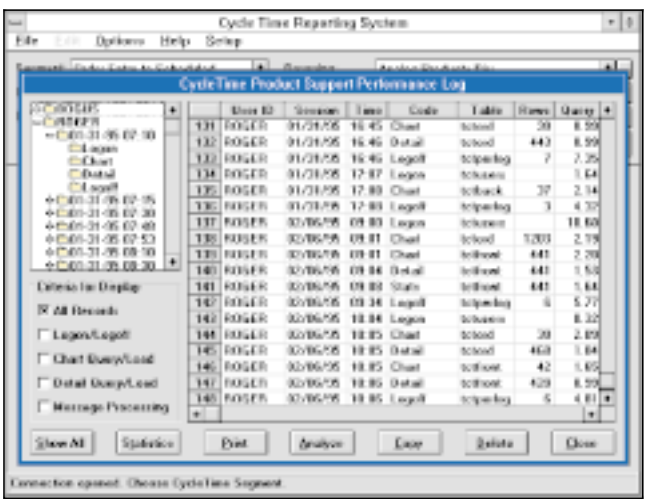


FIGURE 3 ***Get the Lowdown on Performance.*** *The spreadsheet displays records from a snapshot of all the records in the server's tctperlog table. Select records for an individual user, a specific user session, or a single operation within a session by expanding the outline and clicking on the appropriate text element.*

ACCURATE STATS

It's not uncommon for users to set criteria that return a few thousand rows, averaging 15 columns each. CycleTime can print or copy to the clipboard both the histogram and all or parts of the detail sheet. Getting accurate statistics for manufacturing and distribution cycles can involve returning tens of thousands of single-column rows, if the statistics aren't generated by a stored procedure. CycleTime users connect to Sybase through National's world-wide TCP/IP WAN, so network bottlenecks could seriously affect CycleTime's overall performance, especially for detail and statistics queries. Some users use 9600-bps dial-up connections to CycleTime, so the performance of the Shiva LANRover's modem bank also is important.

MANAGEMENT WILL PROBABLY BE HAPPY WITH A COST PER USER OF $500.

The majority of CycleTime's operations involve two steps: executing the query and displaying the query result set (see Figure 3). The performance log and message reply systems are built into the production version of CycleTime, but the menu items to display the support dialogs only appear to those who have a specially-coded TXT file in their \CYCLTIME directory. National's performance monitor logs these CycleTime operations, identified in the Code column of the spreadsheet:

- Logon: The time required for the user to make a connection to the database and the time required to query for pending messages from CycleTime product support. When users first log on to CycleTime, they are requested to provide information about their location and method of connection, which is stored in the tctusers table.
- Chart query and display: The time to execute the query and return rows for histogram data, the time to display the histogram, and the number of rows returned. Rows per second is calculated from this data.
- Detail query and display: The time to execute the query for the spreadsheet data, the time to insert the data in the cells and display the spreadsheet, and the number of rows. Rows per second is calculated for both query execution and display.
- Statistics query: The time to execute the query to return the number of days (float) required by the specified operation for each manufacturing or distribution lot and the number of rows returned. Rows per second is calculated.
- Message transmission: The time to receive one or more messages from CycleTime product support (GetMsgs), acknowledge receipt of a message (AckReplies), and send a message (SendMsg).
- Logoff: The time required to insert the stored performance log records in the Sybase 10 tctperlog table. Rows per second is calculated.

The records for the Detail operation provide the most valuable performance data, because you can identify whether a performance problem is related to database/networking issues or the user's hardware. If the rows per second for the query are in the same range as for other users, but the rows per second to

display the detail data exhibits a low comparative value, it's likely that the user has a slow PC, too many apps open (forcing a disk-swap operation before displaying the data), or a slow graphics subsystem. The entries for message transmission serve to track usage of the messaging system. A high utilization rate might indicate to MIS that other database applications deserve their own messaging system.

GENERATING LOG RECORDS

Getting accurate performance data requires that the front end generate the log records. A performance logging system that relies solely on server performance data won't identify front-end problems. Thus the CycleTime front end creates and stores a record for each operation in the tctperlog table of a local copy of CTUSERS.MDB. CTUSERS.MDB also stores a set of custom user preferences. Ultimately preferences will be stored on the server for roaming users.

Records are identified by an unique user ID and a date/time field, which is constant for each session, and an additional date/time field whose value is set at the end of each operation. A local table is used to save log data so the records can be added to the server's tctperlog table in a single transaction during logoff at the end of the session, improving update speed by a second or so. In the event of a lost connection or a crash, the unsent log records are transmitted the next time the user logs off.

A LogOn record without a corresponding LogOff record for a session indicates the user encountered a serious problem. The subprocedure AddToLocalLog adds a log record to the local tctperlog table (the tblLog Table object). AddToLocalLog is called by the subprocedure for the final element of the logged operation and is passed variables for the operation code (strOpCode), the table in use (strOpTable), the number of records in the snapshot created by the query (lngOpRecords), the time to execute the query (sngOpQuery), and the time to display the result (sngOpDisplay), if applicable (see Listing 1). strUserID and varSession are declared as Global, because the value of these two variables remains the same for a single session.

When the user ends the session by clicking on the Exit button or by closing the main CycleTime form, the Form_Unload event-handler tests the RecordCount property of tblLog. If the user has caused an operation other than a LogOn to occur, the Form_Unload event-handler calls the

```
Sub AddToLocalLog (strOpCode As String, _
  strOpTable As String, lngOpRecords As Long, _
    sngOpQuery As Single, sngOpDisplay As Single)
  'Add record to local log (tctperlog) for operation
  tblLog.AddNew
  tblLog.Fields(0) = strUserID
  tblLog.Fields(1) = varSession
  tblLog.Fields(2) = Now
  tblLog.Fields(3) = strOpCode
  tblLog.Fields(4) = strOpTable
  tblLog.Fields(5) = lngOpRecords
  tblLog.Fields(6) = sngOpQuery
  tblLog.Fields(7) = sngOpDisplay
  tblLog.Fields(8) = False
  tblLog.Update
End Sub
```

LISTING 1 ***Log Performance Locally.*** *This code adds a record to the local copy of the tctperlog table of CTUSER.MDB for each logged operation. Using the AddNew method instead of a SQL INSERT statement, and specifying index values rather than the names of the fields, will contribute to the overall speed of the local INSERT operation.*

```
Sub SendLocalLogToSybase ()
  'Transmit content of local tctperlog
  'table to Sybase tctperlog

  Dim strLogSQL As String
  Dim lngRecs As Long
  Dim sngStart As Single

  On Error GoTo SendLogError

  sngStart = Timer
  Screen.MousePointer = 11
  frmPerfLogoff.Show
  DoEvents
  tblLog.MoveFirst
  tblLog.MoveLast
  If tblLog.RecordCount > 1 Then
    'Send the log data to Sybase
    tblLog.MoveFirst
    Do Until tblLog.EOF
      'Create the INSERT statement
      strLogSQL = "BEGIN TRAN INSERT tctperlog VALUES('"
      strLogSQL = strLogSQL & tblLog.Fields(0) & "', '"
      strLogSQL = strLogSQL & tblLog.Fields(1) & "', '"
      strLogSQL = strLogSQL & tblLog.Fields(2) & "', '"
      strLogSQL = strLogSQL & tblLog.Fields(3) & "', '"
      strLogSQL = strLogSQL & tblLog.Fields(4) & "', "
      strLogSQL = strLogSQL & tblLog.Fields(5) & ", "
      If tblLog.Fields(3) = "Logoff" Then
        'Substitute the projected logoff time
        If tblLog.Fields(5) > 0 Then
          'Safety net for divide by 0
          strLogSQL = strLogSQL & (tblLog.Fields(5) _
              + 1) * (Timer - sngStart) / tblLog._
              Fields(5) & ", "
        Else
          strLogSQL = strLogSQL & (Timer - sngStart) _
              & ", "
        End If
      Else
        strLogSQL = strLogSQL & tblLog.Fields(6) & ", "
      End If
      strLogSQL = strLogSQL & tblLog.Fields(7) & ", 0)"
      strLogSQL = strLogSQL & " COMMIT TRAN"
      'Send the log records to Sybase
      lngRecs = dbSybase.ExecuteSQL(strLogSQL)

      'Mark the record sent
      If lngRecs Then
        tblLog.Edit
        tblLog.Fields(8) = True
        tblLog.Update
      End If
      tblLog.MoveNext
    Loop

    'Clear the sent records from the local
    'tctperlog table
    strLogSQL = "DELETE FROM tctperlog WHERE sent <> 0"
    dbUsers.Execute (strLogSQL)
  End If
  Screen.MousePointer = 0
  Unload frmPerfLogoff
  Exit Sub

SendLogError:
  Screen.MousePointer = 0
  Unload frmPerfLogoff
  MsgBox "Error sending log to Sybase server."

  Exit Sub
End Sub
```

LISTING 2 ***Use a Transaction to Send the Log to the Server.*** *You don't save a lot of time by wrapping the log entries with BEGIN TRANS and COMMIT TRANS, but every second counts when you have a lot of simultaneous users. You can test the difference between individual INSERTs and a single transaction by deleting the BEGIN TRANS and COMMIT TRANS statements in the following code. The frmPerLogoff form tells the user that CycleTime is sending the performance log records.*

SendLocalLogToSybase subprocedure; otherwise no log records are transmitted and the LogOn record is deleted from tblLog. The objective is to eliminate spurious log records, which otherwise would be generated when the user decides to exit CycleTime without running a query.

SendLocalLogToSybase works with individual INSERT operations or as a single transaction in order to compare the performance improvement offered by transaction processing (see Listing 2). Delete BEGIN TRANS and COMMIT TRANS from the strLogSQL string to see the difference in query execution time. Notice that the actual LogOff time must be projected, because the logoff operation doesn't complete until CycleTime transmits the SQL statement to the server.

If you've read my prior Database Design columns, you know I'm a fan of SQL pass-through for Visual Basic and Access applications. SQL pass-through, which sends SQL statements directly to the server, consistently outpaces conventional SELECT, INSERT, and UPDATE queries that must pass though the JET engine's query optimizer and ODBC's query parser.

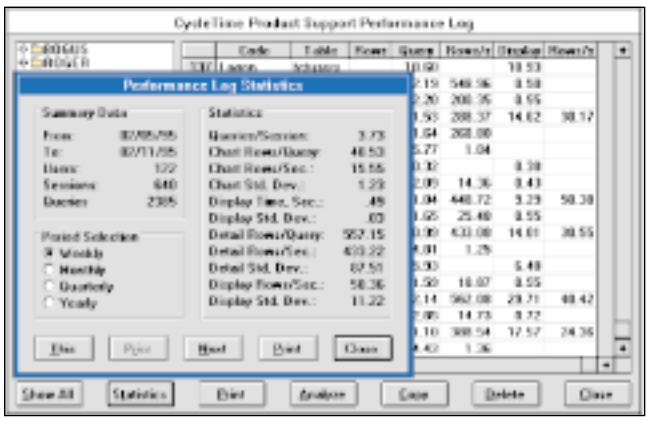


FIGURE 4 ***Statistics Don't Lie.*** *Opening the Performance Log Statistics form makes a pass through a Snapshot of the tctperlog records for the selected time period and calculates the values for the labels. The default period is this week (beginning with Sunday), but you can choose monthly, quarterly, and yearly statistics.*

FACING THE PERFORMANCE REVIEW

Writing the code to store and forward log records to a server table is quite simple and doesn't add much overhead to your front end. The problem is that you must design forms and write code to let the support folks review and analyze the performance log to determine if anything's amiss. The design of the log review form and the log analysis tools is especially important if you have a few hundred users logging on a couple of times per day, which can result in 1000 or more records daily.

HOW LONG DOES THE USER HAVE TO WAIT FOR QUERIED DATA?

The Performance Log's Form_Open event sends a query to the server that returns all of the records in tctperlog to the ssLog Snapshot object. Users of the performance log features of CycleTime are on the same LAN as the server, so the time to retrieve a few thousand tctperlog records is acceptable. A single pass through the Snapshot populates the outline control at three indent levels. (see Listing 3.) When you click on a text element of the outline control to select records for a particular user, session, or operation (or click on the All button to display all records), the code populates the spreadsheet with the type(s) of records specified by the check boxes in the Criteria for Display frame (see Listing 4). Populating the spreadsheet almost always takes longer than creating the Snapshot.

```
Sub GetLogHeaders ()
  'Purpose: Get log entries from Sybase for outline control

  Dim strLogSQL As String    'Message SQL statement
  Dim strUser As String      'User ID of sender
  Dim varSession As Variant  'Session date/time
  Dim intIndex As Integer    'Index to outline control item
  Dim intItem As Integer     'Index to snapshot record
  Dim strMsg As String       'Message box/caption message

  Screen.MousePointer = 11

  'Create the log query and send to Sybase
strLogSQL = "SELECT * FROM tctperlog ORDER BY user_id,_
   session, op_time"
  Set ssLog = _
    dbSybase.CreateSnapshot(strLogSQL, DB_SQLPASSTHROUGH)

  otlLog.Clear
  If ssLog.RecordCount > 0 Then
    'Get the actual record count and
    'set the spreadsheet rows
    ssLog.MoveLast
    ssLog.MoveFirst
    sstLog.MaxRows = ssLog.RecordCount

    Do While Not ssLog.EOF
      If ssLog.Fields(0) <> strUser Then
        'Add a user
        otlLog.AddItem ssLog.Fields(0), intIndex
        otlLog.Indent(intIndex) = 1
        otlLog.ItemData(intIndex) = intItem
        strUser = ssLog.Fields(0)
        intIndex = intIndex + 1
      End If

      If ssLog.Fields(1) <> varSession Then
        'Add a session
        otlLog.AddItem Format$(ssLog.Fields(1), _
        "mm-dd-yy hh:mm"), intIndex
        otlLog.Indent(intIndex) = 2
        otlLog.ItemData(intIndex) = intItem
        varSession = ssLog.Fields(1)
        intIndex = intIndex + 1
      End If

      'Add the log record
      otlLog.AddItem (ssLog.Fields(3)), intIndex
      otlLog.Indent(intIndex) = 3
      otlLog.ItemData(intIndex) = intItem
      intIndex = intIndex + 1
      intItem = intItem + 1

      ssLog.MoveNext
    Loop
  End If

  Screen.MousePointer = 0
End Sub
```

LISTING 3 ***Populate the Outline Control.*** *The Form_Open event calls the GetLogHeaders subprocedure to create a snapshot of all records in tctperlog. A pass through the ssLog snapshot adds items to the otlLog outline control at one or more of the three indent levels: user, session, and operation. When a new record is encountered, three items are added, one at each indent level; two records are added for each session conducted by a particular user.*

```
Sub FillLogRow (fIncrement As Integer)
  'Purpose: Fill a single row of the spreadsheet
  'Returns: fIncrement = True to insert the row

  'Check to see if row is valid
  If Not chkAll.Value Then
    If ssLog.Fields(3) = "Stats" Then
      Exit Sub
    End If
    If Not chkLogOnOff.Value And (ssLog.Fields(3) _
        = "Logon" Or ssLog.Fields(3) = "Logoff") Then
      Exit Sub
    End If
    If Not chkChart.Value And ssLog.Fields(3) _
        = "Chart" Then
      Exit Sub
    End If
    If Not chkDetail.Value And ssLog.Fields(3) _
        = "Detail" Then
      Exit Sub
    End If
    'All message entries are included
    If Not chkMsgs.Value And InStr_
        ("GetMsgsSendMsgAckReplies", ssLog.Fields(3)) _
        > 0 Then
      Exit Sub
    End If
  End If

  'Fill a single row of the log sheet
  sstLog.Col = 1
  sstLog.Text = ssLog.Fields(0)
  sstLog.Col = 2
  sstLog.Text = Format$(ssLog.Fields(1), "mm/dd/yy")
  sstLog.Col = 3
  sstLog.Text = Format$(ssLog.Fields(2), "hh:mm")
  sstLog.Col = 4
  sstLog.Text = ssLog.Fields(3)
  sstLog.Col = 5
  sstLog.Text = ssLog.Fields(4)
  sstLog.Col = 6

  'Rev 11 1/31/95 RJ Don't display zero entries
  If ssLog.Fields(5) > 0 Then
    sstLog.Text = ssLog.Fields(5)
  End If
  sstLog.Col = 7
  If ssLog.Fields(6) > 0 Then
    sstLog.Text = ssLog.Fields(6)
  End If
  sstLog.Col = 8

  'Calculate query rows per second
  If ssLog.Fields(5) > 0 And ssLog.Fields(6) > 0 Then
    sstLog.Text = ssLog.Fields(5) / ssLog.Fields(6)
  End If
  sstLog.Col = 9
  If ssLog.Fields(7) > 0 Then
    sstLog.Text = ssLog.Fields(7)
  End If

  'Calculate display rows per second for Detail only
  sstLog.Col = 10
  If ssLog.Fields(3) = "Detail" And ssLog.Fields(7) _
      > 0 Then
    sstLog.Text = ssLog.Fields(5) / ssLog.Fields(7)
  End If

  fIncrement = True
End Sub
```

LISTING 4 ***Fill Columns Only with Valid Data.*** *The check boxes in the Criteria for Display frame (see Figure 3) determine the types of records to display when you click the text element of an outline control item or click on the All button to display all log records. Because zero values are invalid, zero-value cells are left blank to improve spreadsheet readabilty. The spreadsheet control is FarPoint Technologies' Spread/VBX 2.0.*

Figure 4 shows the Performance Log Statistics form opened over the spreadsheet with the seven right columns visible. The spreadsheet rows-per-second data for the chart (histogram) and detail (spreadsheet) queries, as well as the detail display rows per second, are calculated as the spreadsheet is filled from the snapshot. Zero values appear as empty cells to improve readability. Opening the Performance Log Statistics form initiates a pass through the snapshot records for the default period, the current week beginning with Sunday.

The Period Selection option buttons change the duration of the period, and the This, Prior, or Next buttons execute the code to run comparative statistics for other periods. The period you choose for statistics data becomes the period for analyzing operational exceptions The values for standard deviation generated by opening the Statistics form are needed to set the criteria so as to display only those records for underachieving client operations. CycleTime defines a slow operation as one that doesn't achieve a row-per-second rate of at least the average minus twice the standard deviation for all like operations within a period.

The CycleTime logging system is designed for client/server environments, but it's appropriate for multiuser VB and Access apps using shared MDB files. The code to generate the performance log records adds little overhead to your front end. You take only a small performance hit during front-end shutdown to send the performance log data to the server or to the shared MDB file. Once you create the performance log review forms and their accompanying code, you can re-use them in other front ends with minor alterations. The payoff is the ability to provide you and your client or employer with exact cost-per-user data and to pinpoint bottlenecks. A performance log guarantees you a few extra points on *your* next performance review. ■