Optimizing Application Performance

R:BASE Technologies, Inc.

Optimizing Application Performance

by R:BASE Technologies, Inc.

If your organization relies on R:BASE data, optimizing the performance of your code and database can increase productivity.

Many factors, both large and small, can affect database performance, so fine-tuning your code and database is essential.

Optimizing Application Performance

Printed: August 2014 in Murrysville, PA

First Edition

Table of Contents

Part I

1 Optimizing the SELECT Command

The SELECT command supports variations to return identical results. Using the best optimized syntax, will ensure the information is retrieved quickly.

The following examples show a traditional approach versus the optimized syntax.

Example 01.

```
-- Traditional Command
SELECT ALL FROM Customer WHERE CustState = 'CA' OR CustState = 'MA' OR CustState =
'PA'

-- Optimized Command
SELECT ALL FROM Customer WHERE CustState IN (CA,MA,PA)
```

Example 02.

```
-- Traditional Command
SELECT ALL FROM Customer WHERE CustState <> 'CA' OR CustState <> 'MA' OR CustState <>
'PA'

-- Optimized Command
SELECT ALL FROM Customer WHERE CustState NOT IN (CA,MA,PA)
```

Example 03.

```
-- Traditional Command
SET VAR vNewTransID INTEGER = 0
SELECT (MAX(TransID)) INTO vNewTransID INDIC iv1 FROM InvoiceHeader
SET VAR vNewTransID = (.vNewTransID + 1)

-- Optimized Command
SET VAR vNewTransID INTEGER = 0
SELECT (MAX(TransID)+1) INTO vNewTransID INDIC iv1 FROM InvoiceHeader
```

Example 04.

```
-- Traditional Command
SELECT ALL FROM InvoiceHeader WHERE TransDate >= 1/1/2014 AND TransDate <= 07/31/2014

-- Optimized Command
SELECT ALL FROM InvoiceHeader WHERE TransDate BETWEEN 1/1/2014 AND 07/31/2014
```

Part II

2 Fewer Results in Form Pop-up Menus

Pop-up Menus offer the end user a pop-up dialog window to choose from displayed values. However, filling the contents of the pop-up with too many values will increase the amount of time the user must search for their desired value, and will exhaust time when the dialog loads the available options.

There are several ways to avoid populating the pop-up menu with too many values:

1. Use the "Distinct Value(s)" check box within the Pop-up Menu settings.

2. Issue a DIALOG command before the form is loaded to prompt the end user for data specific to the results of the pop-up menu. Using the input within the WHERE Clause settings of Pop-up Menu will minimize the number of records returned.

3. Add a dynamic WHERE Clause within the Pop-up Menu settings in order to prompt the user for a more specific result.

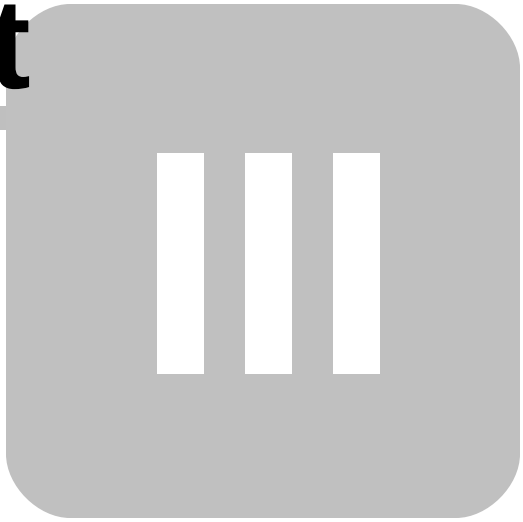
   Examples of dynamic WHERE Clauses:

Example 01:
```
-- Dynamic WHERE Clause (Company LIKE)
Message:
Enter First Few Characters of Company
WHERE Clause:
WHERE Company LIKE '&%' ORDER BY Company
```

Example 02:
```
-- Dynamic WHERE Clause (Company CONTAINS)
Message:
Enter Company Name:
WHERE Clause:
WHERE Company CONTAINS '&' ORDER BY Company
```

Example 03:
```
-- Dyanmic WHERE Clause (CustState =)
Message:
Enter State:
WHERE Clause:
WHERE CustState = '&' ORDER BY Company
```

Part



3 Fewer Results in Form Lookup Controls

Lookup Combo Boxes, Lookup List Boxes and Lookup List Views are powerful controls to use in R:BASE forms for displaying records. However, filling the contents of the controls with too many records will decrease the speed in which the R:BASE form will load and refresh. It is also important to consider how many lookup controls are used on the form in relation to the number of records they will retrieve.

There are several ways to avoid populating Lookup controls with a great number of records:

1. Use the "Display Distinct Value(s)" check box within the object's settings.

2. Issue a DIALOG command before the form is loaded to prompt the end user for data specific to the results of the Lookup. Using the input within the Lookup's WHERE Clause will minimize the number of records returned.

3. Issue a DIALOG command before the form is loaded to prompt the end user for data specific to the results of the Lookup. Then, create a VIEW with resulting values and use the Lookup to display the resulting value(s).

Part IV

4 Command Syntax in Form EEPs

As more and more R:BASE users discover just how powerful Entry/Exit Procedures (EEPs) can be in forms processing, several methods have emerged that can help improve their performance.

Use Custom EEPs and Custom Form Actions
With the enhancement that all Custom Form Actions and Custom EEPs (Forms/Reports/Labels) now run in memory instead of temporary files (version 7.6 and higher), another way to speed up the execution of EEPs is to move all the command file EEPs to Custom EEPs. This eliminates the need for R:BASE to create the temporary files to the disk when executing a command file EEP. You can make the transition to Custom EEPs by performing the following:

1. In the Form Designer, select one of the controls which references a command file EEP.
2. Right click on the control and select "Properties'.
3. From the "EEPs" tab, select the "Edit EEP..." button, which will display your EEP code in the R:BASE Editor.
4. Use the keyboard shortcut [Ctrl] + [A] to highlight the entire contents of the command file EEP.
5. Use the keyboard shortcut [Ctrl] + [C] to copy the contents to the Windows clipboard.
6. Close the R:BASE Editor window to return to the control properties dialog.
7. Select the "Edit Custom EEP..." button, which will open the R:BASE Editor to store the code, only this time the commands will be stored within the form itself.
8. Use the keyboard shortcut [Ctrl] + [V] to paste the entire contents of the command file EEP.
9. Select the "OK" button to return to the control properties dialog.
10. Remove the EEP file name from the "Custom" field so the EEP does not fire from the command and the Custom EEP, and only the Custom EEP.

The control properties dialog should now alter the color of the "Edit Custom EEP..." button to yellow.

Leave the EEP Quickly
When writing an EEP, decide in the first couple of lines whether the user is going to stay in the EEP or return to the form. For example, let's say you want the user to execute hundreds of lines of code if they press [F2] inside a field. Here is one way to accomplish this:

```
SET VAR vLAST = (LASTKEY(0))
IF vLAST = '[F2]' THEN
{HUNDREDS OF LINES OF CODE}
ENDIF
RETURN
```

This EEP will work, but even if the user doesn't press [F2], it requires the reading of hundreds of lines of code. The following structure is more efficient:

```
SET VAR vLAST = (LASTKEY(0))
IF vLAST <> '[F2]' THEN
   RETURN
ENDIF
   {HUNDREDS OF LINES OF CODE}
RETURN
```

This EEP is quicker because if the user doesn't press [F2], it evaluates only four lines of code. The bulk of the code is evaluated only if the user does press [F2]. So, when processing lots of conditional commands, use a GOTO command to jump to the end of the command file or use a RETURN command as soon as the process is completed.

Keep the User Informed
Speed is a matter of perception as much as a value to measure. In fact, the fastest EEP is only as fast as the user perceives it to be. Even if your EEP takes only 10 seconds or less to execute, by not informing the user that something is going on, you risk leaving the impression that something is wrong with the form.

To prevent user frustration, put a PAUSE command at the beginning of the EEP that lets the user know the EEP is being processed. Something like "Calculating ... Please Stand By ..." is probably sufficient.

Example code for a PAUSE dialog with an oscillating gauge can be something like:

```
PAUSE 3 USING 'Calculating ... Please Stand By ...' +
CAPTION 'Calculating ...' ICON APP +
OPTION GAUGE_VISIBLE ON +
|GAUGE_COLOR [R218,G228,B246] +
|GAUGE_INTERVAL 10 +
|MESSAGE_FONT_NAME VERDANA +
|MESSAGE_FONT_SIZE 10 +
|MESSAGE_FONT_COLOR BLUE +
|THEMENAME Longhorn
 -- Put your code, that takes a lot of time, here ...
 -- Use CLS command to clear the PAUSE 3 dialog
CLS
RETURN
```

Part V

5 Moving Form Lookup Variables into Custom EEPs

When upgrading databases from older versions, it is important to take into account if and how R:BASE may work differently. Applications created with legacy logic can perhaps be improved upon in new releases.

For instance, in legacy versions of R:BASE, form variables were not calculated automatically. A RECALC VARIABLES command was needed in an EEP to refresh the variables and generate different results. In newer releases of R:BASE (7.0 and higher), form variables are recalculated automatically when the cursor moves from field to field.

Because of this logic change in R:BASE, the response time may be longer in forms with many lookup variables which are based upon large tables.

To retain that same performance when using the form, the lookup variables can be populated within an EEP.

In the following variables list, there are 11 lookups performed for the "Client" table based upon a provided client identification number. The variables were meant to display a range read-only information about the client.

```
Form :  OrderEntry
Main Table :  Orders
1  : TEXT   vClientFirstName = CFirstName IN Client WHERE ClientID = ClientID
2  : TEXT   vClientLastName = CLastName IN Client WHERE ClientID = ClientID
3  : TEXT   vClientCompany = CCompany IN Client WHERE ClientID = ClientID
4  : TEXT   vClientAddress1 = CAddress1 IN Client WHERE ClientID = ClientID
5  : TEXT   vClientAddress2 = CAddress2 IN Client WHERE ClientID = ClientID
6  : TEXT   vClientCity = CCity IN Client WHERE ClientID = ClientID
7  : TEXT   vClientState = CState IN Client WHERE ClientID = ClientID
8  : TEXT   vClientZipCode = CZipCode IN Client WHERE ClientID = ClientID
9  : TEXT   vClientPhone = CPhone IN Client WHERE ClientID = ClientID
10 : TEXT   vClientFax = CFax IN Client WHERE ClientID = ClientID
11 : TEXT   vClientEmail = CEmail IN Client WHERE ClientID = ClientID
```

Instead, the variable values can be populated using a SELECT command placed within an "On Exit" Custom EEP, within the control where the client ID was entered.

```
-- On Exit EEP
SET VAR vClientFirstName = NULL
SET VAR vClientLastName = NULL
SET VAR vClientCompany = NULL
SET VAR vClientAddress1 = NULL
SET VAR vClientAddress2 = NULL
SET VAR vClientCity = NULL
SET VAR vClientState = NULL
SET VAR vClientZipCode = NULL
SET VAR vClientPhone = NULL
SET VAR vClientFax = NULL
SET VAR vClientEmail = NULL
SELECT +
   CFirstName, +
   CLastName, +
   CCompany, +
   CAddress1, +
   CAddress2, +
   CCity, +
   CState, +
   CZipCode, +
```

```
   CPhone, +
   CFax, + +
   CEmail +
INTO +
   vClientFirstName INDIC iv1, +
   vClientLastName INDIC iv1, +
   vClientCompany INDIC iv1, +
   vClientAddress1 INDIC iv1, +
   vClientAddress2 INDIC iv1, +
   vClientCity INDIC iv1, +
   vClientState INDIC iv1, +
   vClientZipCode INDIC iv1, +
   vClientPhone INDIC iv1, +
   vClientFax INDIC iv1, +
   vClientEmail INDIC iv1 +
FROM Client WHERE ClientID = .vClientID
RECALC VARIABLES
RETURN
```

In the following, lookup data within a "ThirdParty" table, based upon a third party payer ID, variables were also used to display a range read-only information.

```
12 : TEXT   vTPPContactName = TPPContactName IN ThirdParty WHERE TPP_ID = TPP_ID
13 : TEXT   vTPPCompany = TPPCompany IN ThirdParty WHERE TPP_ID = TPP_ID
14 : TEXT   vTPPConract = TPPContract IN ThirdParty WHERE TPP_ID = TPP_ID
15 : TEXT   vTPPCertificate = TPPCertificate IN ThirdParty WHERE TPP_ID = TPP_ID
16 : TEXT   vTPPNotes = TPPNotes IN ThirdParty WHERE TPP_ID = TPP_ID
```

These variable values can be populated using a similar SELECT command placed within an "On Exit" Custom EEP, within the control where the third party payer ID was entered.

```
SET VAR vTPPContactName = NULL
SET VAR vTPPCompany = NULL
SET VAR vTPPConract = NULL
SET VAR vTPPCertificate = NULL
SET VAR vTPPNotes = NULL
SELECT +
   TPPContactName, +
   TPPCompany, +
   TPPContract, +
   TPPCertificate, +
   TPPNotes ,+
INTO +
   vTPPContactName INDIC iv1, +
   vTPPCompany INDIC iv1, +
   vTPPConract INDIC iv1, +
   vTPPCertificate INDIC iv1, +
   vTPPNotes INDIC iv1 +
FROM ThirdParty WHERE TPP_ID = .vTPP_ID
RECALC VARIABLES
RETURN
```

Other variables which perform calculations must remain in the Form Variables. The variables will continue to be updated if any changes are made to the variables contained within the expressions.

```
17 : TEXT   vDayOfWeek = (SGET((TDWK(DateContarct)),3,1))
18 : CURRENCY   vSubTotal = (SUM(ExtPrice))
19 : CURRENCY   vFreight = (.vSubTotal * .01)
```

```
20 : CURRENCY   vSalesTax = (.vSubTotal * .07)
21 : CURRENCY   vInvoiceTotal = (.vSubTotal + .vFreight + .vSalesTax)
```

Note:

Using such technique, make sure to predefine all variables with appropriate data type as On Before Start EEP, and clear all necessary variables as On Close EEP.

```
-- Example
-- On Before Start EEP
   SET VAR vClientFirstName TEXT = NULL
   SET VAR vClientLastName TEXT = NULL
   SET VAR vClientCompany TEXT = NULL
   SET VAR vClientAddress1 TEXT = NULL
   SET VAR vClientAddress2 TEXT = NULL
   SET VAR vClientCity TEXT = NULL
   SET VAR vClientState TEXT = NULL
   SET VAR vClientZipCode TEXT = NULL
   SET VAR vClientPhone TEXT = NULL
   SET VAR vClientFax TEXT = NULL
   SET VAR vClientEmail TEXT = NULL
   SET VAR vTPPContactName TEXT = NULL
   SET VAR vTPPCompany TEXT = NULL
   SET VAR vTPPConract TEXT = NULL
   SET VAR vTPPCertificate TEXT = NULL
   SET VAR vTPPNotes TEXT = NULL
   RETURN

-- On Close EEP
   CLEAR VARIABLES iv%,vClient%,vTPP%
   RETURN
```

You will also need to create two variables as Form Expression to capture the values for entered ClientID and TPP_ID columns.

```
-- Example
nn : INTEGER   vClientID = (ClientID)
nn : INTEGER   vTPP_ID = (TPP_ID)
```

Part VI

6 Finding Minimum and Maximum Values

This section demonstrates techniques for finding the minimum and maximum values from the same table.

Note the use of two separate COMPUTE commands. Example 2 runs faster because R:BASE allows both COMPUTEs to be executed in one command. Example 3 is the fastest because it uses the SELECT INTO *varlist* command.

```
SET VAR t1 TIME = NULL
SET VAR t2 TIME = NULL
SET VAR vDiff INTEGER = NULL

--Example 1 - Slow
SET VAR t1 = (.#TIME)
SET VAR vMin INTEGER
SET VAR vMax INTEGER
COMPUTE vMin AS MIN Altitude FROM Airports
COMPUTE vMax AS MAX Altitude FROM Airports
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
PAUSE 2 USING .vDiff

--Example 2 - Faster
SET VAR t1 = (.#TIME)
SET VAR vMin INTEGER
SET VAR vMax INTEGER
COMPUTE vMin AS MIN Altitude, vMax AS MAX Altitude +
  FROM Airports
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime

--Example 3 - Fastest
SET VAR t1 = (.#TIME)
SET VAR vMin INTEGER
SET VAR vMax INTEGER
SELECT MIN (Altitude), MAX (Altitude) INTO vMin, vMax +
  FROM Airports
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime
```

Part VII

7 Surrounding the WHERE Clause with Parentheses

You can easily gain processing speed by NOT surrounding your WHERE clauses with parentheses where non-text values are being used. When R:BASE sees parentheses, the values are evaluated as text, even though the values are not. By removing the parentheses, your code speeds will increase.

```
SET VAR t1 TIME = NULL
SET VAR t2 TIME = NULL
SET VAR vDiff INTEGER = NULL
SET VAR v1 TEXT = 'Seattle'
SET VAR v2 INTEGER = 93
SET VAR v3 INTEGER = 1000

--Example 4 - Slow
SET VAR t1 = (.#TIME)
SET VAR CheckNum INTEGER = 0
WHILE CheckNum < 40 THEN
SELECT AptCode INTO vAptCode INDICATOR ivAptCode +
FROM Airports WHERE (StCode = .v2 AND CityName +
CONTAINS .v1 AND Runway > .v3)
SET VAR CheckNum = (.CheckNum + 1)
ENDWHILE
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime

--Example 5 - Faster
SET VAR t1 = (.#TIME)
SET VAR CheckNum INTEGER = 0
WHILE CheckNum < 40 THEN
SELECT AptCode INTO vAptCode INDICATOR ivAptCode +
FROM Airports WHERE StCode = .v2 AND CityName +
CONTAINS .v1 AND Runway > .v3
SET VAR CheckNum = (.CheckNum + 1)
ENDWHILE
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime
```

Part VIII

8 Accumulating Data with SELECT

With R:BASE's procedural language, you can use various code structures to accumulate variables from the data values in two different tables. Example 6 uses two DECLARE CURSOR commands to repeatedly locate the values and do the accumulations until all data has been used. Example 7 is faster. It uses one DECLARE CURSOR and one COMPUTE command to accumulate the sum. Example 8, the fastest, uses an INSERT command with a SELECT statement to accumulate the sum.

```
SET VAR t1 TIME = NULL
SET VAR t2 TIME = NULL
SET VAR vDiff INTEGER = NULL

--Example 6 - Slow
SET VAR t1 = (.#TIME)
SET VAR vLength = 0

DROP TABLE Totals
CREATE TABLE Totals (State TEXT 15, Length INTEGER)
DECLARE c1 CURSOR FOR SELECT State, StCode +
FROM States +
WHERE State IN ('Washington', 'Oregon', 'California')
OPEN c1
WHILE SQLCODE = 0 THEN
FETCH c1 INTO vState INDICATOR ivState, +
  vStCode INDICATOR ivStCode
IF SQLCODE <> 0 THEN
BREAK
ENDIF
DECLARE c2 CURSOR FOR SELECT Runway +
FROM Airports WHERE StCode = .vStCode
OPEN c2
WHILE SQLCODE = 0 THEN
FETCH c2 INTO vRunway INDICATOR ivRunway
IF SQLCODE = 0 THEN
SET VAR vLength = (.vLength + .vRunway)
ENDIF
ENDWHILE
INSERT INTO Totals VALUES (.vState, .vLength)
DROP CURSOR c2
ENDWHILE
DROP CURSOR c1
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime

--Example 7 - Faster
SET VAR t1 = (.#TIME)
DROP TABLE Totals
CREATE TABLE Totals (State TEXT 15, Length INTEGER)
DECLARE c1 CURSOR FOR SELECT State, StCode FROM States +
WHERE State IN ('Washington', 'Oregon', 'California')
OPEN c1
WHILE SQLCODE = 0 THEN
FETCH c1 INTO vState INDICATOR ivState, +
  vStCode INDICATOR ivStCode
```

```
IF SQLCODE <> 0 THEN
BREAK
ENDIF
COMPUTE vLength AS SUM Runway FROM Airports +
WHERE StCode = .vStCode
INSERT INTO Totals VALUES (.vState, .vLength)
ENDWHILE
DROP CURSOR c1
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime

--Example 8 - Fastest
SET VAR t1 = (.#TIME)
DROP TABLE Totals
CREATE TABLE Totals (State TEXT 15, Length INTEGER)
INSERT INTO Totals SELECT State, SUM(Runway) +
FROM States,Airports +
WHERE States.StCode = Airports.StCode +
AND State IN ('Washington', 'Oregon', 'California') +
GROUP BY State
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime
```

Part

9 Using Nested Cursors

Nested cursors must be used correctly for efficient execution of code. Notice how the following examples use the DECLARE CURSOR command differently. Both examples work correctly, but only the second attains normal processing speed.

In Example 9, the DECLARE CURSOR is placed inside the WHILE loop of the first cursor. This requires the second DECLARE CURSOR command to execute every time the first command finds a row. In Example 10, both DECLARE CURSOR commands are executed at the start of the program, and the second DECLARE CURSOR command is opened inside the WHILE loop of the first command. By restricting the second DECLARE CURSOR from executing as often, this example runs much faster.

```
SET VAR t1 TIME = NULL
SET VAR t2 TIME = NULL
SET VAR vDiff INTEGER = NULL

--Example 9 - Slow
SET VAR t1 = (.#TIME)
DECLARE c1 CURSOR FOR SELECT State, StCode +
FROM States +
WHERE State IN ('Washington', 'Oregon', 'California')
OPEN c1
WHILE SQLCODE = 0 THEN
FETCH c1 INTO vState INDICATOR ivState, +
  vStCode INDICATOR ivStCode
IF SQLCODE <> 0 THEN
BREAK
ENDIF
DECLARE c2 CURSOR FOR SELECT Runway +
FROM Airports WHERE StCode = .vStCode
OPEN c2
SET VAR vLength = 0
WHILE SQLCODE = 0 THEN
FETCH c2 INTO vRunway INDICATOR ivRunway
IF SQLCODE = 0 THEN
IF vRunway > 5000 THEN
SET VAR vRunway = (.vRunway + 100)
ELSE
SET VAR vRunway = (.vRunway + 50)
ENDIF
UPDATE Airports SET Runway = (.vRunway) +
WHERE CURRENT OF c2
ENDIF
ENDWHILE
DROP CURSOR c2
ENDWHILE
DROP CURSOR c1
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime

--Example 10 - Faster
SET VAR t1 = (.#TIME)
SET VAR vRunway INTEGER = NULL, vStCode INTEGER = NULL
DECLARE cursl CURSOR FOR SELECT State, StCode +
```

```
FROM States +
WHERE State IN ('Washington', 'Oregon', 'California')
DECLARE c2 CURSOR FOR SELECT Runway FROM Airports +
WHERE StCode = .vStCode
OPEN c1
WHILE SQLCODE = 0 THEN
FETCH c1 INTO vState INDICATOR ivState, +
  vStCode INDICATOR ivStCode
IF SQLCODE <> 0 THEN
BREAK
ENDIF
OPEN c2
SET VAR vLength = 0
WHILE SQLCODE = 0 THEN
FETCH c2 INTO vRunway
IF SQLCODE = 0 THEN
IF vRunway > 5000 THEN
SET VAR vRunway = (.vRunway + 100)
ELSE
SET VAR vRunway = (.vRunway + 50)
ENDIF
UPDATE Airports SET Runway = (.vRunway) +
WHERE CURRENT OF c2
ENDIF
ENDWHILE
CLOSE c2
ENDWHILE
DROP CURSOR c1
SET VAR t2 = (.#TIME)
SET VAR vDiff = (.t2 - .t1)
SET VAR vTime = (RTIME(0,0,.vDiff))
PAUSE 2 USING .vTime
```

Part X

10 Command Syntax Techniques

Use the following techniques in your command files and EEPs to speed up processing:

- **Group similar commands**

  R:BASE loads into memory the information needed to process each command. If the command is already in memory, R:BASE does not need to read the disk again. For example, try grouping separate SET VARIABLE commands into one SET VARIABLE whenever possible.

  When UPDATE commands are used for the same table using the same WHERE Clause, the commands should also be combined.

```
UPDATE Donors SET Contrib = .vAmount WHERE DonorID = .vDonorID
UPDATE Donors SET Renew = .vRenew WHERE DonorID = .vDonorID
UPDATE Donors SET GiftDate = .vDate WHERE DonorID = .vDonorID
UPDATE Donors SET Gift = .vNewGift WHERE DonorID = .vDonorID
```

  The following command combines the four UPDATE commands into one.

```
UPDATE Donors SET Contrib = .vAmount, Renew = .vRenew, GiftDate = .vDate, Gift =
.vNewGift WHERE DonorID = .vDonorID
```

- **Minimize disk access**

  Minimize disk access by using Custom EEPs, which run from the computer's memory, and by combining small command files into command blocks within a procedure file.

- **Use WHILE loops**

  Use WHILE loops instead of IF structures or GOTO processing whenever possible. All the commands contained within a WHILE loop are completely read and are retained in memory as long as the WHILE loop is processing. Use BREAK or change the condition for a natural exit rather than using GOTO to exit from a WHILE loop.

- **Replace SET VARIABLE with SELECT INTO for Table Lookups**

  When assigning column values to one or more variables, it is better to use SELECT ... INTO rather than SET VARIABLE. SELECT INTO is the SQL compliant command when capturing table data into variables, and allows for specifying an INDICATOR variable which indicates if a column value is NULL.

  SET VARIABLE syntax:

```
SET VARIABLE vPhone = EmpPhone IN Employee WHERE EmpLastName = 'Smith' AND
EmpFirstName = 'George'
```

  SELECT Syntax:

```
SELECT EmpPhone INTO vPhone INDICATOR ivPhone FROM Employee WHERE EmpLastName =
'Smith' AND EmpFirstName = 'George'
```

- **Use forward GOTO searches**

  Use forward GOTO searches whenever possible. When R:BASE encounters a GOTO, it performs a forward search for the matching label more efficiently than by seeking it in memory, even though the labels are retained internally.

- **Predefine variables**

  Make sure that the data type of each variable is unambiguous by explicitly assigning the data type when the variable is defined.

- **Reduce redundancy**

Eliminate duplication of command sections where possible. If you have a set of commands duplicated in several places within a form, use a "Custom Form Action" which can be called upon at any place in the form. If you have a set of commands duplicated in several places within the entire application, use stored procedures, which can be called upon from almost any place within the application. Like Custom EEPs, stored procedures run from the computer's memory.

- **Eliminate rules checking**

  Set RULES OFF when not needed for data verification. This saves time by preventing R:BASE from checking the data for rule violations. It may also be possible to eliminate the rule with a constraint.

- **Experiment with Manual Table-Order Optimization**

  By default, R:BASE uses its own internal algorithm optimizer that determines the best order for joining tables. You can turn off the automatic optimizer using the MANOPT setting. MANOPT (default:OFF) disables the automatic table-order optimization that R:BASE performs when running queries. This gives maximum control over the order in which columns and tables are assembled in response to a query. With MANOPT set to ON, R:BASE uses the order of the tables in the FROM clause and the order of the columns in the column list of the SELECT clause to construct the query.

Part XI

11 Feedback

Suggestions and Enhancement Requests:

From time to time, everyone comes up with an idea for something they'd like their software to do differently.

If you come across an idea that you think might make a nice enhancement, your input is always welcome.

Please submit your suggestion and/or enhancement request to the R:BASE Developers' Corner Crew (R:DCC) and describe what you think might make a nice enhancement. In R:BASE, the R:DCC Client is fully integrated to communicate with the R:BASE development team. From the main Menu Bar, choose "Help" > "RBG9 R:DCC Client". If you do not have a login profile, select "New User" to create one.

If you have a sample you wish to provide, have the files prepared within a zip archive prior to initiating the request. You will be prompted to upload any attachments during the submission process.

Unless additional information is needed, you will not receive a direct response. You can periodically check the status of your submitted enhancement request.

If you are experiencing any difficulties with the R:DCC Client, please send an e-mail to rbg9rdcc@rbase.com.

Reporting Bugs:

If you experience something you think might be a bug, please report it to the R:BASE eXtreme Developers' Corner Crew. In R:BASE, the R:DCC Client is fully integrated to communicate with the R:BASE development team. From the main Menu Bar, choose "Help" > "RBG9 R:DCC Client". If you do not have a login profile, select "New User" to create one.

You will need to describe:

- What you did, what happened, and what you expected to happen
- The product version and build
- Any error messages displayed
- What computer operating system is in use
- Anything else you think might be relevant

If you have a sample you wish to provide, have the files prepared within a zip archive prior to initiating the bug report. You will be prompted to upload any attachments during the submission process.

Unless additional information is needed, you will not receive a direct response. You can periodically check the status of your submitted bug.

If you are experiencing any difficulties with the R:DCC Client, please send an e-mail to rbg9rdcc@rbase.com.

Part XII

12 Useful Resources

. R:BASE Home Page:	http://www.rbase.com
. R:BASE eXtreme Home Page:	http://www.rbaseextreme.com
. Up-to-Date R:BASE Updates:	http://www.rupdates.com
. Sample Applications:	http://www.rbasecommunity.com
. General R:BASE Syntax:	http://www.rsyntax.com
. Technical Documents - From The Edge:	http://www.razzak.com/fte
. More Sample Applications:	http://www.razzak.com/sampleapplications
. Education and Training:	http://www.rbaseuniversity.com
. Upcoming Events:	http://www.rbase.com/events
. R:BASE Beginners Tutorial:	http://www.rtutorial.com

Index

Notes