RPG WEB DEVELOPMENT

Using Embedded SQL to Process Multiple Rows

By Jim Cooper

In the March article, "RPG Web Development, Getting Started" (www.IceBreak4RPG.com/articles.html), the wonderful and exciting world of RPG Web development was introduced. This article focuses on using embedded SQL in RPG Web applications to process multiple rows from a database table. Embedded SQL is very powerful when developing RPG Web applications, especially when processing tables based on selection criteria.

FIGURE 1 Select Store and Department

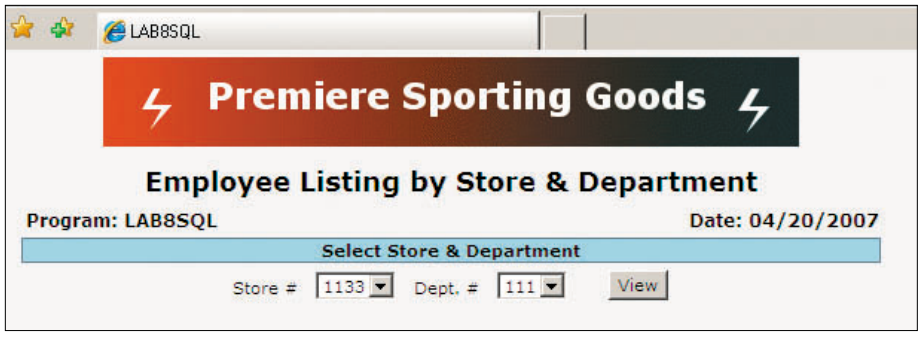
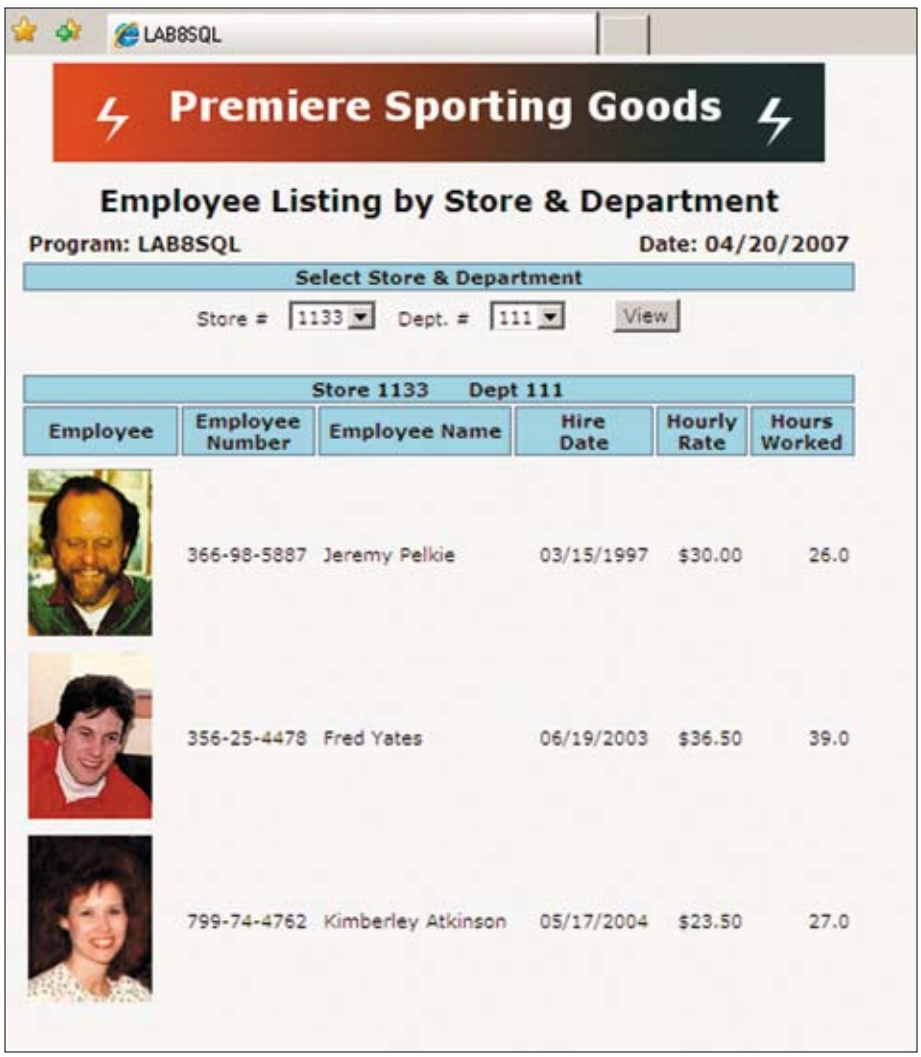


FIGURE 2 Employee Listing



Note: Space restrictions prevent me from displaying the entire RPG program and HTML user interface for this application. If you wish to examine the entire running application and source code, visit www.IceBreak4RPG.com, click RPG templates, and click LAB8sql.rpgle.

THE APPLICATION

The human resource manager of Premiere Sporting Goods wants a Web application that displays employees by department within store. In this application, the user is provided with two drop-down lists, shown in **Figure 1**, from which to select the store and department for the group of employees that are to be displayed.

The values for the drop-down lists are hard coded in the HTML document in **Figure 5**. (In the next article, I will illustrate how to populate these drop-down lists dynamically using an AJAX call to an RPG program.)

Once the user selects the store and department numbers and clicks on the View button, the RPG program retrieves the required rows from the database table and displays them as a table (subfile) in the browser.

Figure 2 illustrates the results when store 1133 and department 111 are selected. The rows are sorted by hire date, last name and first name.

This RPG Web application consists of three source files:

- The CSS (cascading style sheet) provides the rules for how different elements of the Web page are rendered to the browser. The CSS used in this application is a standard CSS used for all my Web applications and stays constant.
- The (X)HTML document, called LAB8sql.html in this example, contains the user interface or presentation layer.
- The RPG program, called LAB8sql.rpgle in this example, contains the business logic.

EMBEDDING SQL WITHIN RPG

Embedded SQL statements must be identified or delimited so the SQL pre-compiler can process them. Prior to V5R4, SQL statements began with /EXEC SQL and ended with /END-EXEC, with the slash in column 7. With V5R4, SQL statements can begin with EXEC SQL and end with a semicolon (;). Most SQL statements have several options depending on the requirements. In this article, I focus only on those SQL statements and options that are necessary to develop this RPG Web application.

DEVELOPING AN RPG WEB APPLICATION WITH EMBEDDED SQL

This program selects and processes multiple rows from the database table depending upon the store and department selected. **Figure 3** identifies the steps required to process multiple rows from a table using SQL.

FIGURE 3 Processing Multiple Rows

1. Declare the SQL Cursor
   - WHERE clause to SELECT group
   - ORDER BY clause to sort selected rows
2. Open the Cursor
3. Fetch the next row (First row)
   - Check for no_rows_found
4. Process multiple rows from Cursor
   - Set Markers
   - Include TABLE_ROW
   - Fetch the next row
5. Monitor for SQL error
6. Close the Cursor

STEP 1: DECLARE THE SQL CURSOR

The openCursor procedure in **Figure 4** contains the DECLARE CURSOR and the OPEN CURSOR statements. The DECLARE CURSOR statement defines a cursor called empCursor. A cursor is a set of rows called a result table that is returned from the query specified in the SELECT statement. No processing takes place with the DECLARE CURSOR statement. It defines the cursor that will contain the result table. Like the database tables from which the data is retrieved, a result table has rows and columns.

FIGURE 4 openCursor Procedure

```
P openCursor      B
  /free
  /exec sql
     DECLARE  empCursor CURSOR FOR
     SELECT   employeeNo,
              firstName,
              lastName,
              hireDate,
              hourlyRate,
              hrsWorked
     FROM     empPayTBL
     WHERE    storeNo = :StoreNbr
     AND      deptNo = :DeptNbr
     ORDER BY hireDate, lastName, firstName
  /end-exec

  /exec sql
     OPEN empCursor
  /end-exec
  /end-free
 P openCursor      E
```

SELECT Statement

The SELECT statement is an embedded statement within the DECLARE CURSOR statement that specifies the query that retrieves data from the database table. The result is stored in a result table. The columns (fields) that follow the SELECT keyword

determine which columns from the database table are returned to the result table. The SELECT statement in Figure 4 selects the employeeNo, firstName, lastName, hireDate, hourlyRate, and hrsWorked columns from the empPayTBL database table. The order they are specified will be the order that they are returned in the result table. You can select as many columns from the database table as necessary or use an asterisk (*) to select all columns. The table name specified in the FROM clause identifies the database table that will be queried to retrieve the desired result table.

WHERE Clause

The WHERE clause in Figure 4 includes the selection criteria used by the SELECT statement to determine which rows are returned from the database table. In this example, there are two comparisons linked together by the AND connector. The AND connector means that both comparisons must be true before the row is selected and returned. When the SELECT statement is executed, the WHERE clause compares the storeNo and deptNo columns (fields) from the empPayTBL database table with the storeNbr and deptNbr host variables containing the selections made by the user. The storeNbr must be equal to storeNo and deptNbr must be equal to deptNo for the row to be returned. Notice that all host variables specified within SQL statements must be preceded by a colon (:).

Host Variables in WHERE Clause

Let us examine how the store and department numbers are selected by the user and placed into the host variables. When the user selects the store and department from the drop-down lists in **Figure 1**, the values are placed in the form variables htmStoreNo and htmDeptNo

FIGURE 5 HTML Drop-down lists

```
<tr>
  <th colspan="6">
    Select Store &amp; Department
  </th>
</tr>
<tr>
  <td colspan="6" class="dropdown">
    Store #  
    <select name="htmStoreNo">
      <option value="1133"> 1133
      <option value="2257"> 2257
      <option value="4464"> 4464
      <option value="5003"> 5003
      <option value="7315"> 7315
      <option value="8950"> 8950
    </select>
     
    Dept. #  
    <select name="htmDeptNo">
      <option value="111"> 111
      <option value="222"> 222
      <option value="333"> 333
      <option value="444"> 444
      <option value="555"> 555
      <option value="666"> 666
    </select>
         
    <input type="submit"
    name="search_button" value="View" />
  </td>
</tr>
```

in the HTML code in **Figure 5**. The user clicks on View and a request is made to the server to run the RPG program. At the same time, the HTML form containing the values for htmStoreNo and htmDeptNO is submitted to the RPG program.

When the RPG program begins execution, the values in htmStoreNo and htmDeptNo are moved to the host variables storeNbr and deptNbr as shown in **Figure 6**. The host variables storeNbr and DeptNbr are later used in the WHERE clause to determine the rows to select from the database table.

FIGURE 6 Save selected values

```
storeNbr = (formNum('htmStoreNo'));
deptNbr  = (formNum('htmDeptNo'));
```

ORDER BY Clause

The ORDER BY clause is used to sort the result table. Thus, the rows are sorted hire date (major sort), last name (intermediate sort), and first name (minor sort).

STEP 2: OPEN THE CURSOR

The OPEN statement in the openCursor procedure in **Figure 4** allocates memory for the empCursor cursor and does any other housekeeping tasks depending how the cursor is being used. The OPEN statement executes the SELECT statement for the DECLARE CURSOR. If necessary, the OPEN statement builds an access path. Once the access path is established, the result table is created based on the selection criteria.

STEP 3: FETCH THE NEXT ROW

The FETCH NEXT statement retrieves rows into the program. In Figure 7, the FETCH NEXT statement retrieves a single row from the cursor and populates the host variable :employeeNo, :firstName, :lastName, :hourlyRate, and :hrsWorked.

FIGURE 7 fetchNext Procedure

```
P fetchNext       B
D fetchNext       PI
 /free
 /exec sql
    FETCH NEXT
      FROM empCursor
      INTO :employeeNo,
           :firstName,
           :lastName,
           :hireDate,
           :hourlyRate,
           :hrsWorked
 /end-exec
 /end-free
P fetchNext       E
```

The SELECT statement identifies rows that contain the column values the program wants. However, SQL does not retrieve any data until the FETCH statement is issued. When the program retrieves data, the values are placed into the host variables specified with the INTO clause. In this example, the host variables are defined in a host structure, which is an external data structure, based on the database table being queried:

```
D extTableDS     E DS                  extname(empPayTBL)
```

Since this data structure contains the record format for the database table, it acts as a host structure containing the host variables used as the target for the INTO clause of the FETCH NEXT statement.

Check for NO ROWS FOUND Condition

There may be occasions when there are no employees assigned to a particular department at a store. When this happens, the store and department selection does not return any rows to the result table. As a result, the SQLSTATE return code (discussed later) needs to be monitored after the first FETCH NEXT statement for the end of table (SQL_EOT) as shown in Figure 8. If the SQLSTATE returns a value of '02000', end of the table was detected. Since this end of table condition occurred on the first FETCH statement, this indicates that there were no rows selected with the SELECT statement.

FIGURE 8 Check for no Rows Found

```
D SQL_EOF         C                   const('02000')
  . . .
  fetchNext();
  if sqlstate = SQL_EOT;
    includeHTML('no_rows_found');
    return;
  endIf;
```

STEP 4: PROCESS MULTIPLE ROWS FROM THE CURSOR

Cursors are usually controlled with a DO WHILE loop. The loop allows each row to be fetched one at a time. **Figure 9** illustrates the processing required as the program processes multiple rows from the result table. The SQLSTATE = SQL_OK condition controls whether the DO WHILE loop is executed. The SQL_OK variable is initialed to zeroes to indicate a record is successfully retrieved from the empCursor cursor. A FETCH NEXT statement is used to retrieve the next row before the next cycle of the DO WHILE loop. This process continues one row at a time until all rows in the result table have been processed. When all of the rows have been processed, SQLSTATE returns a value of '02000' indicating the end of the table has been reached.

FIGURE 9 Processing Multiple Rows

```
dow sqlstate = SQL_OK;
   setMarkers();
   includeHTML('table_row');
   fetchNext();
 enddo;
 monitorSQL();
```

Using Markers in the HTML Document

The IceBreak Application Server includes an extension called markers that are used in the HTML document and RPG program. In the HTML code, markers begin with a dollar ($) sign and identify the location where the RPG program will insert dynamic data at runtime. In the HTML code in **Figure 10**, there are five markers. These markers are not defined in the RPG program; however, at runtime the RPG program places dynamic values into these markers. As a result, when the page is rendered to the browser, the dynamic values appear on the Web page.

FIGURE 10 Markers in HTML Document

```
<!--#tag="table_row"-->
  <tr>
    <td>             <%$ employeeNo %> </td>
    <td>             <%$ fullName   %> </td>
    <td class="num"> <%$ hireDate   %> </td>
    <td class="num"> <%$ hourlyRate %> </td>
    <td class="num"> <%$ hrsWorked  %> </td>
  </tr>
```

Markers in the RPG Program

In the RPG code in **Figure 11**, markers are referenced using the setMarker statement. These setMarker statements are used to transfer the dynamic values from the RPG program variables to the markers specified in the HTML document.

Since storeNo and deptNo are numeric variables, they are converted to character values with the %char function before being moved to the markers, which must contain character data. The advantage of using markers is that one person can be developing the HTML document containing the user interface and another person can be developing the RPG program containing the business logic. The two people do not need to know the skill set of the other person. Instead, they just need to know the names of the markers that will be used to communicate between the RPG program and the HTML document.

STEP 5: MONITOR FOR SQL ERROR

Every time SQL performs an input/output operation on a database table, the operating system issues a return code to the program indicating the resulting state of the SQL statement. Two SQL variables, SQLCODE and SQLSTATE, contain the return code and can be monitored and tested to determine the result of the operation. Both SQLCODE and SQLSTATE are derived from the SQL standard, but SQLCODE has been marked deprecated. Therefore, new applications are strongly encouraged to use SQLSTATE. SQLSTATE is a five-character variable that can contain five digits (0-9) or letters (A-Z). The five-character variable contains two parts that represent codes of various error and warning conditions. The first two characters indicate the general class of the condition; the last three characters indicate a subclass of the general condition. A successful state is indicated by the return code '00000'. The meanings of the general class values are Successful (00), Warning (01), No Data (02), and Error (03 through ZZ).

Every program must anticipate an end-of-table condition whenever a cursor is used to fetch rows. When the SQLSTATE variable contains a value of '02000', this means that either the end of the table has been reached or there were no rows in the table to begin with. In Figure 9, SQLSTATE is tested against the named constant SQL_OK to determine when the end of the table is reached. This condition occurs when the FETCH NEXT statement has retrieved the last row in the result table and the program issues a subsequent FETCH NEXT.

FIGURE 11 setMarker Procedure in RPG Program

```
P setMarkers      B
 /free
  setMarker('fullName'  : %trim(firstname) + ' ' + %trim(lastName));
  setMarker('employeeNo': %editW(employeeNo  : '0   -  -    '));
  setMarker('hireDate'  : %char(hireDate : *USA));
  setMarker('hourlyRate': %editC(hourlyRate  : '3' : '$'));
  setMarker('hrsWorked' : %editC(hrsWorked   : '3'));
 /end-free
P setMarkers      E
```

STEP 6: CLOSE THE CURSOR

Once the selected rows have been processed and the end of the table has been reached, the cursor should be closed. The operating system automatically closes cursors under certain conditions, but it is recommended that the cursor be closed when finished. The empCursor is closed in the closeCursor procedure in **Figure 12**.

COMPILE THE RPG PROGRAM WITH EMBEDDED SQL

The source type of SQLRPGLE must be used when compiling an RPG program containing embedded SQL. SQLRPGLE programs go through a two-step compile process. Before the RPG code is sent to the standard RPG compiler, it is first passed through the SQL pre-compiler. The pre-compiler's main job is to validate all of the SQL statements and convert them to dynamic program calls. The altered code is then passed to the main RPG compiler. The source type SQLRPGLE is specified in the first line of the RPG program as

```
<%@ language="SQLRPGLE" . . . %>
```

Once the RPG and HTML source files are created and saved in the IFS folder, the application can be compiled into a native program (.PGM) object. To compile, just request the application in a browser by entering http://server_name/LAB8sql.rpgle in the Address box, where server_name is the name of the server and LAB8sql.rpgle is the name of the RPG program. If the program is new or has been changed, the just-in-time compiler compiles the application. If the program compiles without errors, a native program object (*PGM) is created and stored in the library that is assigned to the application server. If the program does not compile, a compile listing is returned to the browser identifying the errors.

RUN THE RPG APPLICATION IN A BROWSER

The IceBreak Application Server does not execute the source files in the IFS folder. Only the program object (*PGM) stored in the library is executed. Therefore, once an application is completed, only the program object needs to be available to run the application. To run this application, enter http://server_name/LAB8sql.rpgle in a browser where server_name is the name of the server and LAB8sql.rpgle is the name of the RPG program. The browser makes the request to the application server, which retrieves and executes the program object from the library. The result is a Web page displayed in the browser as shown in **Figure 2**.

FIGURE 12 closeCursor

```
P closeCursor     B
 /free
 /exec sql
    CLOSE empCursor
 /end-exec
 /end-free
P closeCursor     E
```

VIEW SOURCE

A Web page rendered to the browser by an RPG application only contains HTML. To verify this, go to www.IceBreak4RPG.com/templates.html, run LAB8sql.rpgle and click Page/View Source in the browser.

C*RN BYTES

By Ken Davis

TEC is a success
There's never excess,
And most people who come
Think it's the Bes'.

Jon and Susan
Ain't ever nefarious,
And when they talk,
They're even hilarious.

Now take our friend
Glenn Gunderman...
The stuff he does
is a wonder, man!
For TUG he works late,
His time dedicate,
So no one can steal
his thunder, man.

Our Wende's the best
She's smart, and nice too,
Without our Ms. Boddy,
What would TUG do?

You will see that there is no RPG code or business logic sent to the browser. The user can only see the HTML generated from the RPG program. You will also notice that the RPG program generated several table rows, which are a result of looping (doW/endDo) through the employee file.

ICEBREAK APPLICATION SERVER

Modern Web languages have a scripting language, an easy to use user interface, and an HTTP/application server that runs the applications. The application in this article demonstrates how easy it is to deploy an RPG Web application with embedded SQL. This is accomplished with the IceBreak Application Server, one of the most exciting development technologies to be developed for System i in recent years.

IceBreak is a powerful HTTP/application server that runs natively on System i, iSeries, and AS/400. IceBreak installs in minutes with little configuration, and has an integrated development feature that allows developers to build and deploy Web applications using the native ILE languages RPG and COBOL and other technologies such as SQL, XHTML, XML, Web Services, and AJAX. IceBreak is not a tool but a powerful advanced HTTP/application server that does not require CGI, Java, Apache, WebSphere (WAS), PASE, WebFacing, HATS, or third-party generator tools. With IceBreak, developers benefit from a single, integrated application-hosting environment. The IceBreak Application Server provides the best Web infrastructure to take advantage of the ILE environment.

Jim Cooper *has been a Professor at Lambton College in Sarnia, ON for 23 years where he has taught System i (AS/400) technology since 1991. After discovering and testing the IceBreak Application Server technology, and realized that it was the most advanced technology for modern RPG Web development, he formed System & Method International (North America) to distribute the IceBreak Application Server technology in North America. Jim can be reached at jac@system-method.com or 519-464-6646.*