TechTip: Use a Stored Procedure as Your Data Source in DB2 Web Query for i

Published Friday, 24 July 2009 01:00 by MC Press On-line [Reprinted with permission from iTechnology Manager, published by MC Press, LP; http://www.mcpressonline.com.]

Written by Anita Corcoran - Anita.Corcoran@StoneMor.com

Build queries with optional input parameters using existing business logic.

Do you need the ability to build queries with optional input parameters? Do you need your queries to use existing business logic? Did you know you can accomplish both goals by using a stored procedure as your data source in DB2 Web Query?

When we purchased DB2 Web Query, our shop needed to create new reports for a department that reports accounting transactions to various state agencies and banking institutions. The business logic for the reports would be the same regardless of what state or bank was requesting the data, but the level of detail, the columns included, and the filtering could be different each run. Some states wanted details; some only summary. Some required information monthly; some quarterly. Some states wanted us to include beginning and ending balances along with the totals for the period being reported; others wanted information related only to period. The legacy reports that we were replacing were written in RPG and had no input parameters other than Month. They provided no ability to filter data. They had no ability to drop unwanted columns, nor did they let the users choose detail or summary output.

We knew that DB2 Web Query would give us what we needed. We also realized that we could "front end" multiple DB2 Web Queries with one stored procedure that could dynamically build an SQL SELECT statement that would allow us to provide optional input parameters. Additionally, by keeping the business logic in one stored procedure, we could eliminate duplicating/cloning it in multiple queries.

In this TechTip, I'll discuss writing a free-format SQLRPGLE program that receives parameters and dynamically builds a SELECT statement with a WHERE clause that includes only the optional parameters that the user populated. I'll discuss registering the SQLRPGLE program as a stored procedure and building a synonym over it. I'll show you how to build a summary query and a drill-down query that both use the same stored procedure's synonym.

Step 1: Creating Your Stored Procedure

If you are new to creating stored procedures in the System i, don't be intimidated. A stored procedure is only a program that you call from within SQL. You can write a stored procedure in languages you are already fluent in: RPG, CL, SQL, and others. Your stored procedure will select the records that match the

selection criteria in your parameters. If applicable, your stored procedure can use existing business logic to segment your data. The stored procedure will then return your data as a result set to DB2 Web Query.

In our case, we have existing business logic that evaluates records in a large accounting transaction file. In addition to State, Location, and Contract#, each record carries a transaction code, period, and amount. The transaction code defines whether the transaction amount should be classified as Servicing$, Adjustment$, or Receipts$. Our stored procedure (STOREDPROC) uses that business logic to segment the transactions into the appropriate column. Additionally, STOREDPROC is using the parameters to filter the data in the file. The Start period and End period parameters are mandatory. The State, Location, and Contract# parameters are optional, and the SELECT statement built by STOREDPROC includes them in the WHERE clause only if they have a value.

Our parameters are defined in prototype STOREDPROC

```
  //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  // Input parms
  //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

d inputparm       pr                  extpgm('STOREDPROC')
d  iState                      2
d  iLocation                   3
d  iContract                   9
d  isPeriod                    6
d  iePeriod                    6

d inputparm       pi
d  pState                      2
d  pLocation                   3
d  pContract                   9
d  psPeriod                    6
d  pePeriod                    6
```

The mainline code of STOREDPROC is very simple. Subroutine @build_select builds the dynamic SQL SELECT statement. Subroutine @open_cursor uses the SELECT statement to prepare and open the cursor for return to DB2 Web Query.

```
  //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  //  work fields
  //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
d stm1            s           1000a
d q               c                   ''''
  //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 /free

  // build dynamic SQL select statement and include those
  // parms supplied by the user
```

```
  exsr     @build_select;

  // use the select sttmt to open a cursor and send back result set
  exsr     @open_cursor;

  // all done
  *inlr = *on;
  return;
```

The first subroutine, @build_select illustrates two things:

- The use of the existing business logic to segment the data into the appropriate columns
- The use of the optional parameters in the WHERE clause

```
// ------------------------------------------------------
begsr @build_select;
// ------------------------------------------------------

 stm1 = 'SELECT +
   state, +
   location, +
   contract, +
   sum(case when (period < ' + pSperiod+ ') then amount else 0 end) +
     as begBal$, +
   sum(case when (tranCode = 20) +
     and (period >= ' + pSperiod+ ') then amount else 0 end) +
     as periodSer$, +
   sum(case when (tranCode in(0,6,9)) +
     and (period >= ' + pSperiod+ ') then amount else 0 end) +
     as periodAdj$, +
   sum(case when (tranCode in(2,3,7)) +
     and (period >= ' + pSperiod+ ') then amount else 0 end) +
     as periodRcpt$, +
   sum(amount) +
     as EndBal$ +

  FROM ACCTGFILE +
  WHERE tranCode in (0,2,3,6,7,9,20) +
    and period <= ' + q + pEperiod+ q;

  if pState > *blank;
    stm1 = %trimr(stm1) + ' and state = ' + q + pState + q;
  endif;

  if pLocation > *blank;
    stm1 = %trimr(stm1) + ' and location = ' + q + pLocation + q;
  endif;

  if pContract > *zeros;
    stm1 =  %trimr(stm1) +  ' and contract = ' + q  + pContract + q;
```

```
  endif;

  stm1 =  %trimr(stm1) +
   ' GROUP BY state, location, contract +
     ORDER BY state, location, contract';

 endsr;
```

The second subroutine, @open_cursor, uses the string just built to prepare, declare, open, and return a cursor as a result set to DB2 Web Query.

```
  // ----------------------------------------------------
  begsr @open_cursor;
  // ----------------------------------------------------

   exec sql prepare stmt from :stm1;
   exec sql declare C1 cursor with return to client for stmt;
   exec sql open C1;
   exec sql set result sets cursor C1;

  endsr;
```

After a successful compile of your program, you can register it as a stored procedure using the following SQL command:

```
CREATE PROCEDURE STOREDPROC
(IN STATE    CHAR(2) ,
 IN LOCATION CHAR(3) ,
 IN CONTRACT CHAR(9) ,
 IN SPERIOD  CHAR(6) ,
 IN EPERIOD  CHAR(6) )
 DYNAMIC RESULT SETS 1
 LANGUAGE RPGLE
 SPECIFIC       STOREDPROC
     DETERMINISTIC
 READS SQL DATA
 CALLED ON NULL INPUT
 PARAMETER STYLE SQL
```

Step 2: Testing Your Stored Procedure

You are now ready to test your stored procedure. The steps are simple: you simply use the Run SQL Scripts tool in iSeries Navigator to call your stored procedure with your parameters. An excellent TechTip by Kevin Forsythe details how to use iSeries Navigator to test a stored procedure.

Make sure to pass a value to your mandatory parameters. Play around with passing some of your optional parameters, and check the result set returned by your stored procedure. When you have

verified that your SQL SELECT statement in your stored procedure is passing back correct results, you are ready to create a synonym.

Step 3: Creating a Synonym over Your Stored Procedure

These instructions assume you are familiar with creating a synonym over a table. To create a synonym over a stored procedure, select Stored Procedures in the "Restrict object type to" drop-down box as shown in Figure 1.

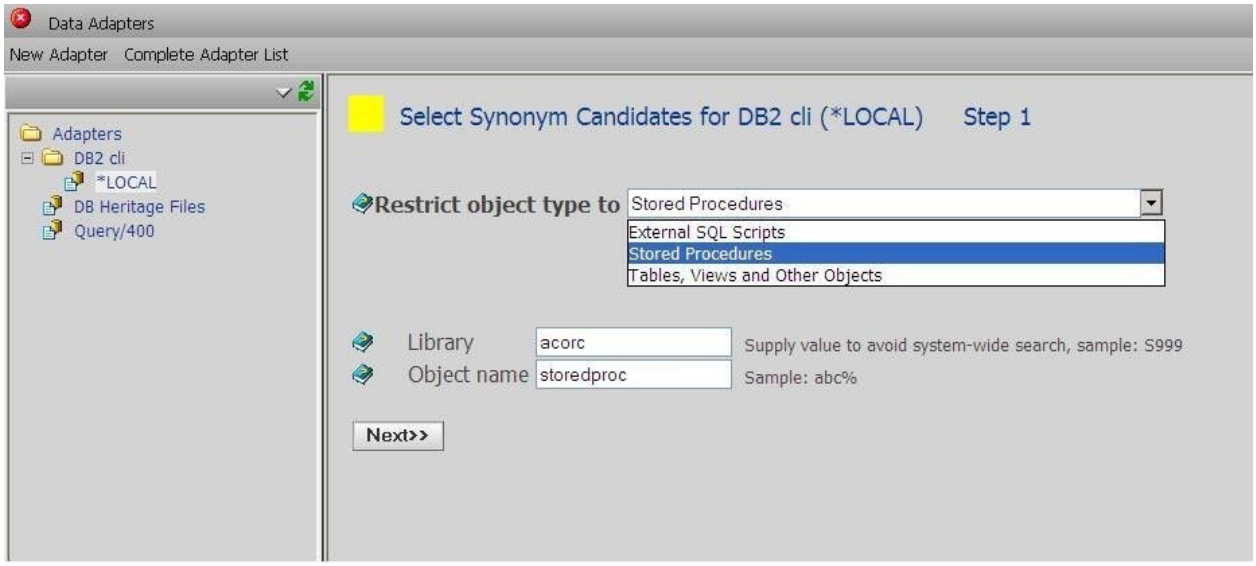


Figure 1: Choose the Stored Procedures option for Select Synonym Candidates.

The Step 3 page of the Create Synonym pages will display the input parameters you defined in your stored procedure. When you click the Create Synonym button, your stored procedure will be called, and if it runs successfully and returns a result set, your synonym will be created and will include the parameters and all the fields in the result set. If your stored procedure includes any parameters that are mandatory for a successful run, then key in a valid value for those parameters as shown in Figure 2 before clicking Create Synonym.

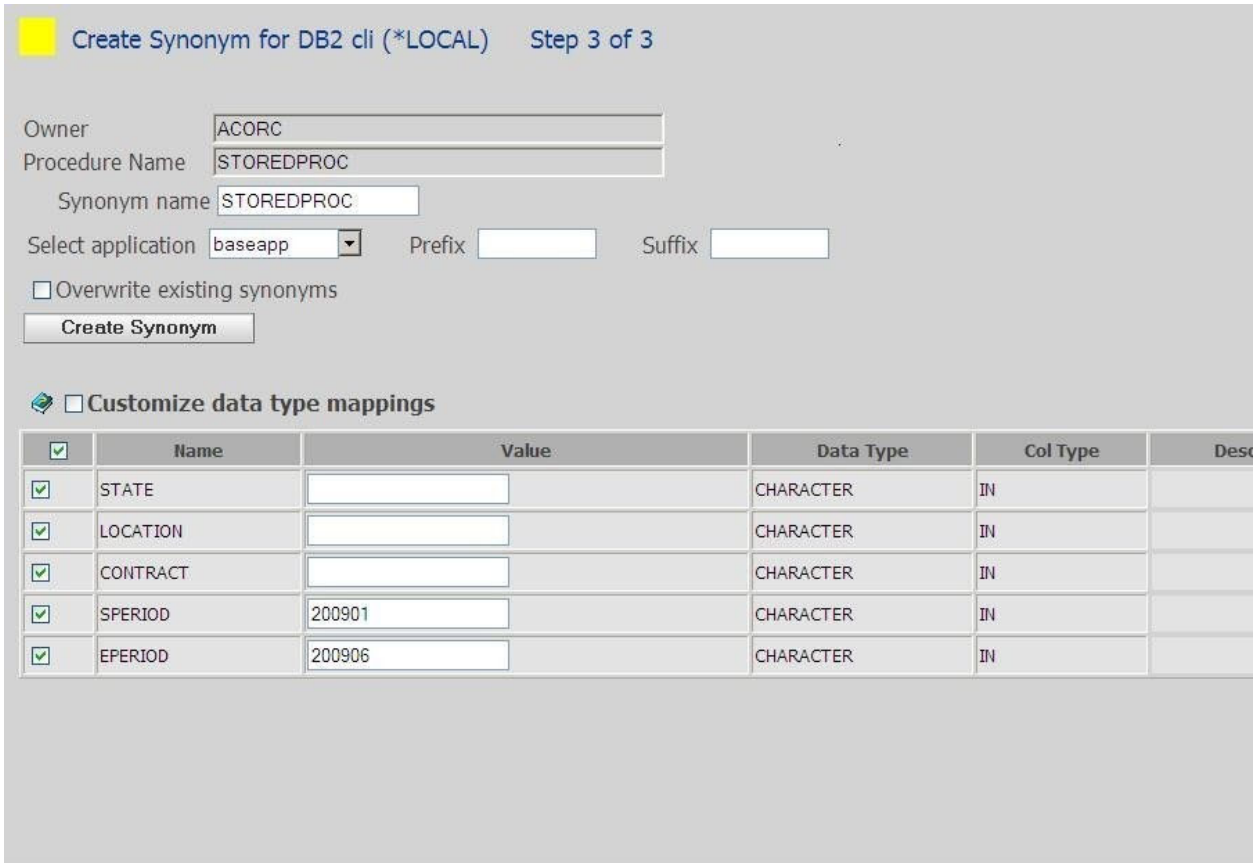


Figure 2: Complete the Create Synonym process.

Step 4: Using the Stored Procedure's Synonym in a Summary and Drill-Down Set of Web Queries

You can select the synonym over your stored procedure in DB2 Web Query and use it like any other synonym. The only difference you will notice is that your field list will contain separate segments for your parameters (segment INPUT) and the fields in your result set (segment ANSWERSET1). See Figure 3.

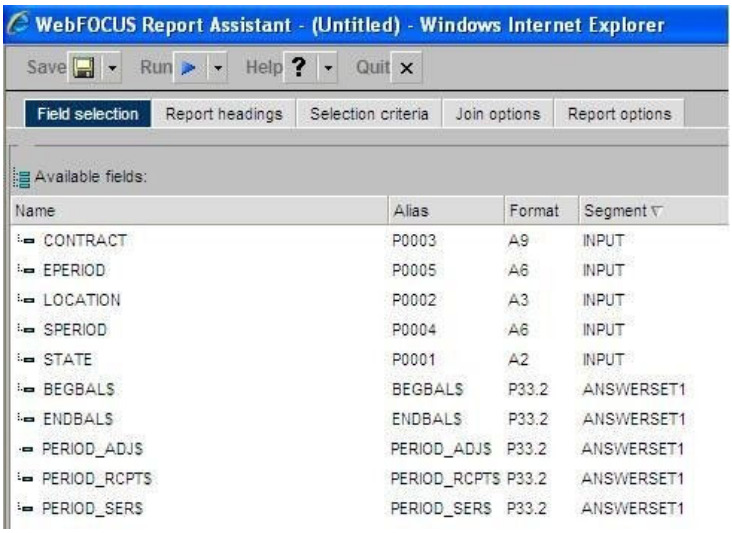


Figure 3: This is the field list in Web Query for synonym STOREDPROC.

To filter the data returned by the stored procedure, include all your parameters in the Selection Criteria Window, as shown in Figure 4.

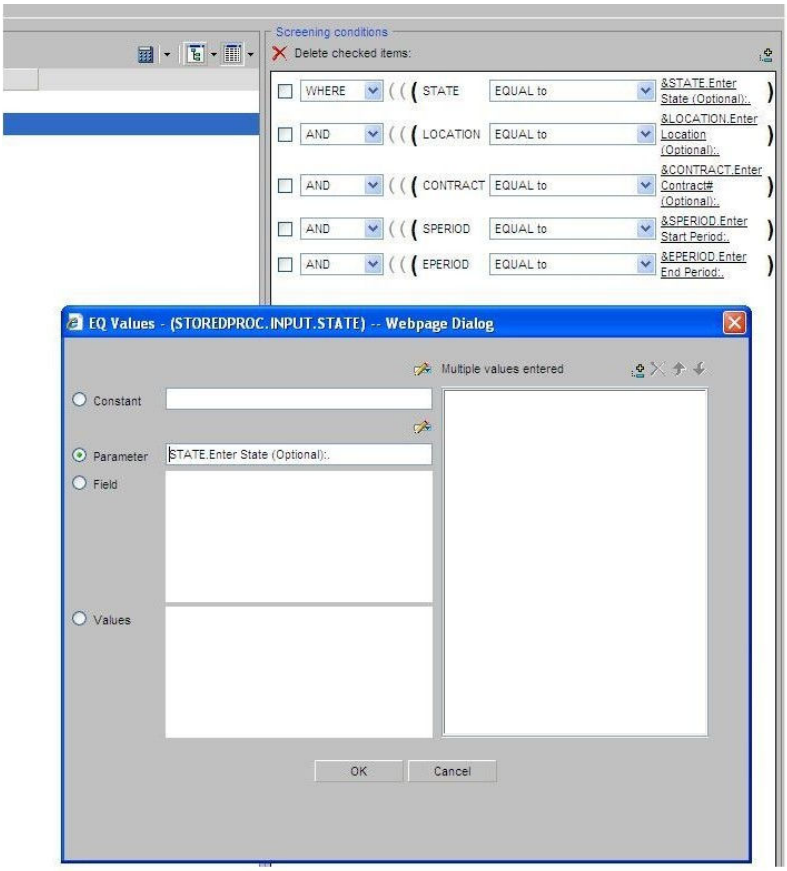
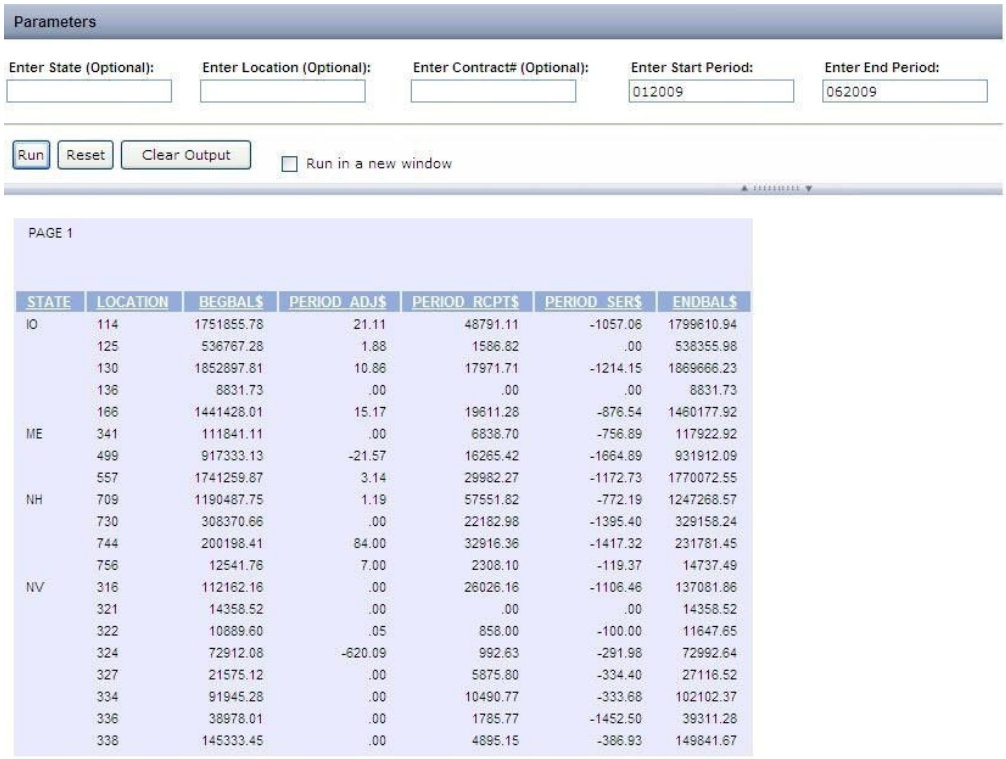


Figure 4: Include your parameters in the Selection Criteria window.

An example of the resulting simple summary query is shown in Figure 5. Note that we passed values into only the mandatory parameters.



STATE	LOCATION	BEGBAL$	PERIOD_ADJ$	PERIOD_RCPT$	PERIOD_SER$	ENDBAL$
IO	114	1751855.78	21.11	48791.11	-1057.06	1799610.94
	125	536767.28	1.88	1586.82	.00	538355.98
	130	1852897.81	10.86	17971.71	-1214.15	1869666.23
	136	8831.73	.00	.00	.00	8831.73
	166	1441428.01	15.17	19611.28	-876.54	1460177.92
ME	341	111841.11	.00	6838.70	-756.89	117922.92
	499	917333.13	-21.57	16265.42	-1664.89	931912.09
	557	1741259.87	3.14	29982.27	-1172.73	1770072.55
NH	709	1190487.75	1.19	57551.82	-772.19	1247268.57
	730	308370.66	.00	22182.98	-1395.40	329158.24
	744	200198.41	84.00	32916.36	-1417.32	231781.45
	756	12541.76	7.00	2308.10	-119.37	14737.49
NV	316	112162.16	.00	26026.16	-1106.46	137081.86
	321	14358.52	.00	.00	.00	14358.52
	322	10889.60	.05	858.00	-100.00	11647.65
	324	72912.08	-620.09	992.63	-291.98	72992.64
	327	21575.12	.00	5875.80	-334.40	27116.52
	334	91945.28	.00	10490.77	-333.68	102102.37
	336	38978.01	.00	1785.77	-1452.50	39311.28
	338	145333.45	.00	4895.15	-386.93	149841.67

Figure 5: Here are the results of a simple summary query.

You can turn this simple report into the parent in a drill-down set by selecting one of the fields and drilling down to a Web Query that uses the same STOREDPROC as its data source. You can, as appropriate, use fields from the answer set as the value for the parameters to be passed to the drill-down query. In this example, shown in Figure 6, we are allowing the user to drill down on the Location field. We are passing to the drill-down query the values for State and Location in the answer set row selected. The remaining parameters passed are identical to the parameters on the parent Query.

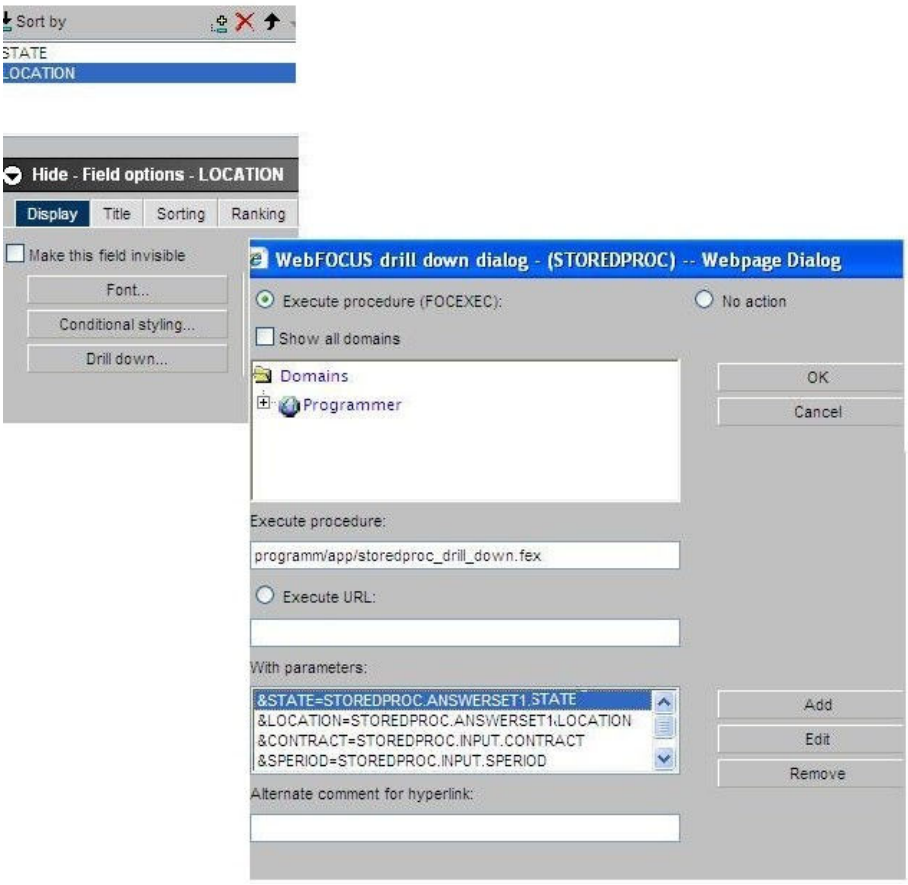


Figure 6: Change the parameters used by the parent query when it executes the drill-down.

I hope this simple example will inspire you to try this in your own shop. In future TechTips, I'll expand this example to include the addition of the input parameter values, user, and run date/time in the DB2 Web Query header and footer. I'll also show you a way get your stored procedure's synonym to work with the user's library list.