Creating SQL Stored Procedures with Text Templates

J. Michael Morrill, Shamrock Computer Resources, Moline, Illinois

Abstract

This paper will present text templates and how they were used to create SQL stored procedures for DB2®. Text templates are text strings with imbedded variable fields which are resolved at program execution time. An example of a text template is a form letter created with word processor software. The form letter contains variable fields imbedded within static text. When form letter is merged with a file, the imbedded variable fields are replaced with data from the file and the result is a document containing both the static text and variable data.

The same principle can be applied to executable code. A text template containing the skeleton of an SQL statement can be merged at program execution time with data to create a complete executable SQL statement. The skeleton of the SQL statement can be stored in a database table as opposed to being imbedded within the program that executes the statement. A skeleton SQL statement stored in the database is known as a stored procedure.

Introduction

In September of 1994, the author started designing a new application. The purpose of the application was to provide a user interface to an existing DB2 database. The interface was to be constructed using SAS/AF® running under MVS® and TSO®.

The database contained data used to print parts catalogs. The purpose of the application was to allow catalog writers to modify existing catalogs and to create new catalogs using all or part of existing catalogs. During the design of this application, the author realised that many of the data manipulation functions of the application were similar to one another. For example the function to copy a section from one catalog to another was a subset of the function to create a new catalog by copying data from an existing catalog. The two functions would use duplicate SQL statements in two different programs. The author concluded that much development work could be saved and the overall quality of the application improved if the redundant SQL statements were eliminated.

Therefore the author created a new sub-system that did three things:

- Stored the data manipulation SQL statements in a database as text templates.
- Filled in the fields of the text templates with data from a global data area.
- Executed the SQL statements stored in the database.

The application programs used the sub-system to execute data altering SQL statements instead of using SQL statements imbedded within the programs.

The new sub-system had two major components, text templates and SQL procedures.

Text Templates

The following is an example of a text template:

```
UPDATE LISTED PARTS SET
  STATUS_CODE = (STATUS)
WHERE (COND_1) (COND_2)
```

The strings (STATUS) (COND_1) and (COND_2) are variables. Variables are denoted by strings enclosed in braces. If the variables in the example above are set to the following values:

- (STATUS) is set to ‘R’
- (COND_1) is set to CATALOG = 1234
- (COND_2) is set to AND PART = 928347

the result would be:

```
UPDATE LISTED PARTS SET
  STATUS_CODE = 'R'
WHERE CATALOG = 1234
  AND PART = 928347
```

This template is designed to set the column STATUS_CODE equal to the value ‘R’ for either the whole catalog or just a part within the catalog. If the STATUS_CODE was to be set to ‘R’ for the entire catalog, the variable (COND_2) is set to blank. The resulting SQL statement would be:

```
UPDATE LISTED PARTS SET
  STATUS_CODE = 'R'
WHERE CATALOG = 1234
```
Setting the Values of Variables

The values of the variables are not passed from the application to the sub-system as a parm list. Instead the application program and the sub-system pass data via Named Items in a SCL list. The SCL list used is defined by value of the function ENVIYLIST('L'). This list is referred to as the Global Data Area. Manipulation of the Global Data Area is done by macros shown in Table I.

The problem with the example template is that it requires the application program to know too much about the SQL statement being used to update the STATUS_CODE, thus decreasing the re-usability of the skeleton SQL statement. The application program has to know: the DB2 column names of the table it is updating; which DB2 column to use when updating a catalog and which column to use when updating a part; and the structure of the condition statements. The application program also has to contain logic to build the conditional statements. The problems are solved by the new template:

```
UPDATE LISTED_PARTS SET
  STATUS_CODE = '{STATUS}',
  WHERE CATALOG =
    {-E CATALOG_NUMBER}
    {-C AND PART = {PART_NUMBER}}
```

This example introduces several new features. First notice that the variable {STATUS} is enclosed in quotes. The Resolver is insensitive to quotes in the template, therefore the application program does not need to store the quotes in the variable. Second the string {-E CATALOG_NUMBER} introduces Resolver processing flags. Processing flags control how the template is resolved by the Resolver. In the string {-E CATALOG_NUMBER} the sub-string -E is the processing flag, the sub-string CATALOG_NUMBER is the variable and the whole string {-E CATALOG_NUMBER} is a field. In this example, the processor flag 'E', signals the Resolver to return an error code to the calling application program if the variable CATALOG_NUMBER contains a missing or null value.

The field {-C AND PART = {PART_NUMBER}} introduces conditional text. Conditional text is static text within a field that will only be included in the resolved field if none of the variables within the field contain null or missing values. The field also introduces the field within the field structure. The flag 'C' signals the Resolver to treat all the remaining text within the field, that is not enclosed in braces, as conditional text. Text that is enclosed in braces is treated as a field. The Resolver examines all fields within the conditional text fields and resolves the variables within those fields. If any variable contains a null value or if the calling application program did not store a value for the variable.

Merging Templates and Variables

The program in the sub-system that merges the template with the values of the variables is called the Resolver.

The application program stores the value of the variables as well as the name of the template to be resolved in a Global Data Area and then calls the Resolver. The Resolver retrieves the name of the template from the Global Data Area, fetches the named template from the table where the templates are stored, and replaces the variables with the corresponding values from the Global Data Area. The process of merging the text template with the values of the variables is called resolving.

### Table 1

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDASET</td>
<td>Insert a new variable (named item) or set the value of an existing variable in the Global Data Area</td>
</tr>
<tr>
<td>GDADEL</td>
<td>Remove a variable from the Global Data Area</td>
</tr>
<tr>
<td>GDAGET</td>
<td>Get the value of a variable from the Global Data Area. If the variable is not on the Global data area, return a missing indicator.</td>
</tr>
</tbody>
</table>

Table 2 explains the parameters used in the macros.

### Table 2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL</td>
<td>Value of the variable to be add to the Global Data Area, or the new value of a variable already in the Global Data Area.</td>
</tr>
<tr>
<td>VAR</td>
<td>The name of the variable to be added, changed, read or removed.</td>
</tr>
<tr>
<td>PVAR</td>
<td>The name of the program variable to receive the value read in the GDAGET macro.</td>
</tr>
</tbody>
</table>

Using the examples above, the following is an example of how a SAS/AF program would set the value of the variables and then call the sub-system.

```
%GDASET(VAL="R", VAR=STATUS);
%GDASET(VAL=CATALOG = 1234, VAR=COND_1);
%GDASET(VAL=AND PART = 928347, VAR=COND_2);
CALL DISPLAY("SUBSYSTEM");
```
Applications Development

in the Global Data Area, the conditional text will not be included in the final resolved template.

In this example in the application program sets the variables as follows:

{STATUS} is set to R
{CATALOG_NUMBER} is set to 1234
{PART_NUMBER} is set to 928347

the result would be:

UPDATE LISTED_PARTS SET
STATUS_CODE = 'R'
WHERE CATALOG = 1234
AND PART = 928347

If the application program sets the variables to:

{STATUS} is set to R
{CATALOG_NUMBER} is set to 1234
{PART_NUMBER} is set to NULL

the result would be:

UPDATE LISTED_PARTS SET
STATUS_CODE = 'R'
WHERE CATALOG = 1234

Storing and Using Stored Procedures

The templates are stored in a DB2 table called the Template Table. Table 3 shows the structure of this table. Each row in the DB2 holds one line of template text. There a template with five lines of text will use five rows in the Template Table.

Table 3

<table>
<thead>
<tr>
<th>Column</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATE_NAME</td>
<td>CHARACTER 8</td>
<td>The name of the text template.</td>
</tr>
<tr>
<td>SEQUENCE_NUMBER</td>
<td>INTEGER</td>
<td>The sequence number of the text line. The first text line in a template has a sequence number of 1.</td>
</tr>
<tr>
<td>TEXT_LINE</td>
<td>CHARACTER 60</td>
<td>A line of text in the template.</td>
</tr>
</tbody>
</table>

The SQL statements stored in template form are called stored procedures. Stored procedures are used by the application programs in lieu of imbedded SQL. Figure 1 illustrates the data flow and steps used to execute a stored procedure.

Figure 1 Data flow of the execution of a stored procedure

In the first step, the application program stores the data needed for the stored procedure in the Global Data Area.

In the second step, the application program calls the Resolver. The Resolver fetches the stored procedure from the template table and resolves all fields with data from the Global Data Area. The resolved stored procedure is now in the form of executable SQL statements. The Resolver calls the Execution Control Program and passes the SQL statements.

The Execution Control Program is the final step. The Execution Control Program passes the SQL statements to DB2 via the SAS® SQL Pass Through facility. DB2 executes the statement and returns an SQL code. The SQL return code from the execution is stored in the Global Data Area.

Below is an example of how the code in a SCL program would appear to execute a stored procedure. The variable TEXT_TEMPLATE contains the name of a stored procedure. The Execution Control Program stores the DB2 SQL return code in the variable SQL_RESULT.

```sas
%GDASET(VAL=R, VAR=STATUS);
%GDASET(VAL=1234, VAR=CATALOG_NUMBER);
%GDASET(VAL=928347, VAR=PART_NUMBER);
%GDASET(VAL=UPDSTAT, VAR=TEXT_TEMPLATE);
CALL DISPLAY("RESOLVER");
%GDASET(pVAR=RETURN_CODE, VAR=SQL_RESULT);
%GDADEL(VAR=STATUS);
%GDADEL(VAR=CATALOG_NUMBER);
%GDADEL(VAR=PART_NUMBER);
%GDADEL(VAR=UPDSTAT, VAR=TEXT_TEMPLATE);
IF RETURN_CODE NE 0 THEN DO;
   CALL DISPLAY("ERRORMSG");
END;
```
Benefits of Stored Procedures

At first glance, the example above would make it appear far easier to use imbedded SQL statements via the SUBMIT block instead of stored procedures. To see the advantage of stored procedures, one must expand their vision to include the application as a whole. Just as database normalization requires us to eliminate redundant data across a database, good application construction requires us to eliminate redundant code across an application.

From the example above, imagine that three application programs set the status code. Each of these programs can set the status code for individual part or for a whole catalog. If we use imbedded SQL, we would need six SQL statements. With six SQL statement, there are six places for the programmer to make a mistake. If there was a change in business policy which required a change in the SQL statement which updated the status code, six SQL statements across three programs would have to be changed. If any SQL statement was not changed, the result would be a difficult to locate bug in the application system.

When stored procedures are used, all data altering SQL is stored in one place and redundant SQL code can be eliminated. Application construction time is decreased and application quality is increase because any data altering SQL statement need only be written and tested once.

Conclusion

The superior design of the SAS/AF product made it possible to create and use DB2 stored procedures in spite of the fact DB2 does not provide a native stored procedure facility. Stored procedures sharply reduced the resources and time required to develop the original application and enhancements. Maintenance costs have been reduced because most of the changes to the application can be done by changing the text in the stored procedure instead the code in the SAS/AF programs.

A spin-off of the effort to create stored procedures is the template text facility, which is now being used, with excellent results, in other SAS based applications.

Endnotes

1 The exact nature of this application and the data within the application is confidential. Therefore certain facts have been distorted to hide the identity of the application and the identity of the client for whom this application was created. The distorted facts are immaterial to this paper.

2 This name should not be confused with the Global SCL List which is defined by the function ENVLIST(G).

3 Listings of these macros is available from the author upon request.

4 Just before this paper was submitted for publication, IBM announced the release of DB2 Version 4. This version of DB2 does have a stored procedure facility. However, the IBM implementation of stored procedures consists of imbedded SQL within third generation languages. Furthermore, at this time, the author knows of no way to call DB2 stored procedures from SAS.

Acknowledgments

Memory Turner of Shamrock Computer Resources Ltd. and Jean Schmid, a freelance technical writer, for editing this paper.

Wendy Anderson of Shamrock Computer Resources Ltd., Bob Hippler and Tracy Hill of Deere and Company, for helping me construct the system.

Trademarks

SAS and SAS/AF are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicate USA registration.

MVS, TSO and DB2 are registered trademarks of IBM Corporation in the USA and other countries. ® indicate USA registration.

The Author

J. Michael Morrill is a Managing Consultant for Shamrock Computer Resources Ltd. in Moline, Illinois. He specializes in Data Modeling, Data Base Design and Data Warehouse Design and Building. He also writes a column for the Journal of Systems Management.

He can be reached at:
Shamrock Computer Resources Ltd.
800 36th Avenue, Suite 101
Moline, Illinois 61265
Phone: (309) 762-7626
Fax: (309) 762-9542
Email: JMICHAEL@SROCK.COM