

Using the DB2 System Catalog to Generate SAS programs

by
John M Morrill

Introduction

This paper describes an application in which a SAS program uses the DB2 catalog to generate another SAS program. The generated program copies data from a DB2 table, converts it to Comma Delimited records, and writes the data to a text file. Due to space limitations, the source code of this program is not reproduced here. However, it is available from the author free of charge.

Instead this paper will describe the technique this application employs to generate a SAS program from the DB2 catalog. This techniques can be used to write other applications.

History

Soon after the creation of a DB2 data warehouse, the customers needed to use data from the warehouse in electronic spreadsheets. However, the Medical Center was only getting started with client-server applications, and lacked any method to easily convert and download data from the warehouse. Data processing people could convert and download data, but it was a labor intensive process that required writing a COBOL program for each download the customers requested.

The lead time for a download was two weeks. Because many customers needed data on shorter notice, they often had to print out the data and laboriously re-enter it into their spreadsheet.

After the Medical Center acquired SAS/ACCESS, the data processing people switched to writing download programs in SAS, and reduced the lead time to four working days. However, due to the shortage of SAS programmer resources and because customers often needed downloads within hours, a more efficient way had to be found to write download programs.

Why SAS

In the Medical Center, the only alternative to SAS was COBOL. The SAS/CONNECT software was not an option. The author considered writing a COBOL program which would use a dynamic SQL SELECT statement to convert

DB2 tables into Comma Delimited text files. The author decided against this approach for the following reasons:

1. Dynamic SQL is not well documented. The author was unable to find any good reference material on the subject of programming a dynamic SQL SELECT statement in a COBOL program. The author would have to spend days learning the programming technique.
2. COBOL does not have stream output. A large programming effort would have been required to format an output Comma Delimited text record using the COBOL language.
3. The resulting program would have been inflexible. Because a large amount of the coding in the COBOL program would have been used for the dynamic SQL SELECT statement and formatting the output, it would have been very difficult to improve the program. The author estimates that the COBOL program would have contained 1000 - 1500 lines of source code. The current SAS program that does the same function contains 140 statements.

Functional Overview

The SAS application has three major sections. The first section connects to the DB2 catalog, also known as the meta data. The next section generates source code. The generated source code is written to a SAS include file. The third section executes the generated code. This overview describes the technique of using meta data to dynamically write an executable program.

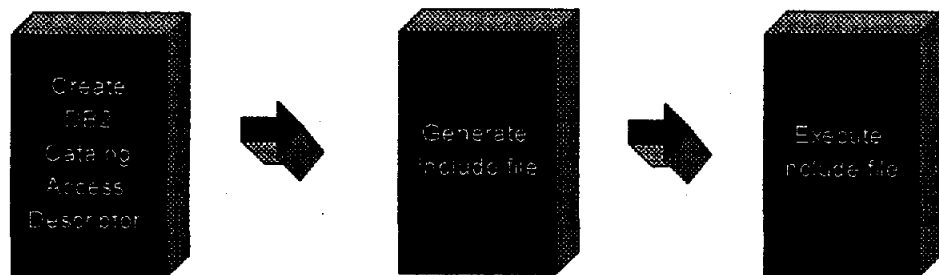


Figure 1 Functional Overview of the application.

This technique can be used with any almost any language and any situation where the function to be performed is always the same but the extract nature of the data is not known until execution time.

Detailed Description

Figure 2 is a data flow chart of the application. Notice how the data store **SAS include file** becomes **Execute include file**. The data itself becomes executable instructions. The author was not able to find any agreed upon notation to chart this event although is a common occurrence. For example, every time a computer executes a program, it reads the executable code from disk as data, loads the code into memory and then executes the code.

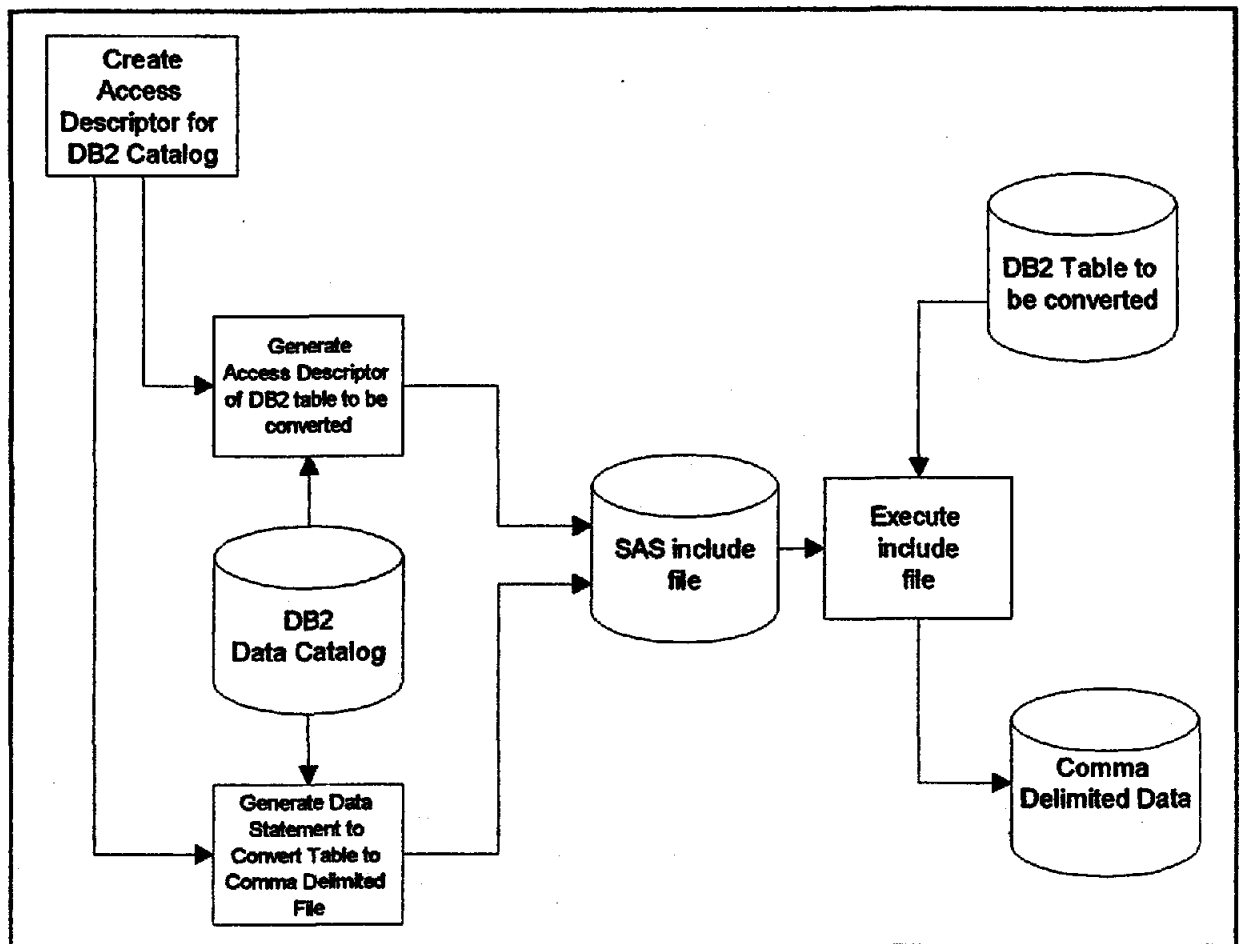


Figure 2 Data flow chart of SAS conversation program

The detailed explanation of this program is presented below in outline form.

1. Create Access Descriptor for DB2 Catalog.

An Access Descriptor to the SYSIBM.SYSCOLUMNS table is created. Next an Access View is created which limits the access to only the columns of the DB2 Table to be converted.

2. Generate Access Descriptor of DB2 table to be converted.

Using the meta data from the DB2 catalog, both a SAS access descriptor and view is generated then written to the include file.

3. Generate Data Statement to Convert Table to Comma Delimited File

Again, using the meta data from the DB2 catalog, a SAS Data Statement is generated and written to the include file.

4. Execute the include file.

The include file contains an Access Descriptor and View and a Data Statement. The Data Statement contains a SET verb to input the DB2 Table to be converted. A FILE statement and OUTPUT verb within the Data Statement is used to write the Comma Delimited Data.

Pseudo Code

Below is pseudo code of the application. For this pseudo code we will use the table definition in chart 1. The name of the table is CUST.ACCOUNT. The Comma Delimited test file will be called ACCOUNT.TXT. Comments in the pseudo code are italicized.

SYSIBM.SYSCOLUMNS			
NAME	COLTYPE	LENGTH	SCALE
Customer_Number	Decimal	9	0
Account_Balance	Decimal	9	2
Customer_Name	Character	20	

Chart 1 Example definition of DB2 table to be converted to Comma Delimited

/ Program Start */*

INPUT FROM CONSOLE +DB2_TABLE_NAME+,
+COMMA_DELIMITED_NAME+;

CREATE ACCESS TO SYSIBM.SYSCOLUMNS, ACCESS IS NAMED
META_DATA, SELECT ROWS WHERE TBCREATOR || TBNAME =
+DB2_TABLE_NAME+;

DATA _NULL_;
SET META_DATA;
OUTFILE IS CONVERT_DATA;

/ If the first row of meta data, write out code to create first part of access
descriptor */*

IF _N_ = 1
WRITE "CREATE ACCESS TO " +DB2_TABLE_NAME+
", ACCESS IS NAMED DB2_DATA"
END_IF;

/ Rename the columns so that there will number be a problem with SAS limitations
on variable names */*

WRITE "RENAME " +NAME+ "TO CN" || _N_;
IF _N_ = LAST_OBSERVATION
WRITE ";;"
END_IF;
END_DATA_STATEMENT;
/ Now generate the Data Step */*

DATA _NULL_;
SET META_DATA;
OUTFILE IS CONVERT_DATA APPEND MODE;

```

IF _N_ = 1
  WRITE "DATA _NULL_;"
  WRITE "SET DB2 _DATA;"
  WRITE "OUTFILE IS " +COMMA_DELIMITED_NAME+ ";"
  WRITE "WRITE"
END_IF

/* GENERATE CODE TO WRITE OUT EACH COLUMN */

IF +COLTYPE+ = "CHARACTER"
  WRITE "DBL_QUOTE " NO_RETURN
END_IF;
WRITE "+CN+" || _N_ CONVERT_TO_SAS_FORMAT(+COLTYPE+,
+LENGTH+, +SCALE+) NO_RETURN
IF _N_ IS NOT LAST_OBSERVATION
  WRITE "COMMA" NO_RETURN
IF +COLTYPE+ = "CHARACTER"
  WRITE "DBL_QUOTE " NO_RETURN
END_IF;

IF _N_ = LAST_OBSERVATION
  WRITE ";"
END_IF;

WRITE RETURN;

IF _N_ = LAST_OBSERVATION
  WRITE "END_DATA_STATEMENT"
END_IF;

END_DATA_STATEMENT;

%INCLUDE CONVERT_DATA

/* End of Generator Program */

```

Below is what the generated program would look like:

```

CREATE ACCESS TO SYSIBM.SYSCOLUMNS, ACCESS IS NAMED
DB2_DATA

```

```
RENAME CUSTOMER_NUMBER TO CN1
RENAME ACCOUNT_BALANCE TO CN2
RENAME CUSTOMER_NAME TO CN3;
```

```
DATA _NULL_;
SET DB2_DATA;"
OUTFILE IS ACCOUNT.TXT;
WRITE CN1 9. COMMA
CN2 9.2 COMMA
DBL_QUOTE CN3 $CHAR20. DBL_QUOTE ;
END_DATA_STATEMENT;
```

Conclusion

The techniques described in this paper can be used in many other applications. The basic flow is that a program inputs meta data, generates code, then executes the generated code. Using this technique allows the programmer to write programs where the details of the data to be processed are not known until execution time, thus saving the programmer from having to repetitively write the programs that perform the same function on different data.

Author

John M. Morrill

The Ohio State University Medical Center
Room A062 SL

Columbus, Ohio 43210-1240

Email: jmorril@magnus.acs.ohio-state.edu Phone: (614) 293-3860