EFFICIENTLY ACCESS DATA WAREHOUSE WITH PROC SQL
PASS-THROUGH FACILITY AND MACRO
James Yang, Highmark Blue Cross Blue Shield, Pittsburgh, PA

ABSTRACT
PROC SQL Pass-Through facility has superior performance over SAS/ACCESS view descriptors on Highmark Information Warehouse (MVS DB2), especially when joining the tables. If a relatively small SAS® data set needs to join with the Data Warehouse tables, the key values can be put into macro variables and then passed to either the DB2 query with Pass-Through facility, or the where clause on SAS/ACCESS view descriptors. Although the CPU time goes down dramatically in both cases provided the key is indexed on DB2, the Pass-Through facility is still much efficient than SAS/ACCESS view descriptors. Based on this information, macro %JOINDB2 has been developed to bypass the restrictions on statement length (about 32KB) and day-time CPU time, which are governed by DB2. These programs can be adapted for other DBMS and platforms.

The greatest impact on performance is caused by the transfer of data between the DBMS and the SAS System. Pass-Through facility passes queries to DBMS and takes advantage of DBMS optimizer and indexes. Macro variables pass the key values to DBMS, retrieve and transfer only a small subset of data. Performance improvement not only saves computing resources, but also increases productivity and makes information available quickly (especially for interactive applications and working on mainframe with day-time CPU restriction).

BACKGROUND
On Highmark Information Warehouse under OS/390 (MVS), there are SAS/ACCESS view descriptors for the DB2 tables. The claims tables have 10-35 millions rows and 110-140 columns. Using the view descriptors to retrieve data usually takes more than an hour of CPU time and needs to be run over night (there is a CPU time governor of 195 seconds during the day), even when joining with a very small SAS data set. The poor performance restricts the productivity and information on-time delivery. This paper shows that PROC SQL Pass-Through facility performs much better than SAS/ACCESS view descriptors. Based on this information, macro %JOINDB2 has been developed to join large DB2 tables (1 or more than 1) with a SAS data set.

WAYS AND PERFORMANCES
Programs 1-10 on Appendix I (JCL not included) are used to pull drug claims from the drug table (35-million rows x 112 columns) for 581 CHF members in the SAS data set OUT.CHFBM with product in (60,80,81,82) and service date between Jan 1, 1996 and Jun 30, 1997. DWH.V_RALS_DRUG and HI.W.DW16ALL are the DB2 name and the SAS/ACCESS view descriptor for the drug table, respectively. Two keys are used to identify the claims: EMPIDORG and PATDOB on the drug table, EMPIDORG and BIRTH_DT on the data set OUT.CHFBM. EMPIDORG, PRODUCT, and SVIDDATE are indexed on the DB2 drug table. All programs were run in batch mode with SAS 6.08 on OS/390 (MVS). The final output data sets are identical under PROC COMPARE. Two important performance metrics, CPU time and EXCP count (Input/Output operations) are used to compare the performances.

Program 1 uses PROC SORT to sort the view descriptor and direct the output to a temporary data set DRUG, then merge with OUT.CHFBM using a DATA step.
Program 2 extracts data from the view descriptor to a work data set DRUG, then sorts it, merges it with OUT.CHFBM using a DATA step.

Program 3 uses SAS/ACCESS view descriptor within PROC SQL to join the drug table with SAS data set OUT.CHFBM.
Program 4 uses PROC SQL Pass-Through facility to retrieve data directly from DB2 drug table, then join with OUT.CHFBM.
Program 5, 6, 7, and 8 lists the EMPIDORG key values from OUT.CHFBM into a macro variable &LISTKEY, then adds it into the where clause for the view descriptor or the pass-through facility in program 1, 2, 3, and 4, respectively. They first retrieve a small subset of claims with EMPIDORG in &LISTKEY from drug table, then join/merge with OUT.CHFBM to match EMPIDORG and birthdays.

Program 9 modifies program 7 by removing the KEEP statement after the view descriptor and specifying the needed variables in the SELECT clause. Program 10 is revised from program 8 by including all drug table columns inside the Pass-Through (DB2 query) and then selecting needed columns in PROC SQL select clause.

The performances for program 1-10 are listed below. Among the 10 programs, program 8 has the best performance.

<table>
<thead>
<tr>
<th>Programs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXCP</td>
<td>failed</td>
<td>failed</td>
<td>80.36</td>
<td>22.19</td>
<td>0.27</td>
</tr>
<tr>
<td>Programs</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>CPU (min)</td>
<td>0.23</td>
<td>0.22</td>
<td>0.07</td>
<td>0.22</td>
<td>0.22</td>
</tr>
<tr>
<td>EXCP</td>
<td>failed</td>
<td>583</td>
<td>506</td>
<td>965</td>
<td></td>
</tr>
</tbody>
</table>

Program 11 and 12 on Appendix I use the Pass-Through facility to join the SAS data set OUT.CHFBM with two DB2 tables: case table DWH.V_RALS_IP_CLAIM (13 millions rows x 139 columns) and inpatient service table DWH.V_RALS_CASE (0.64 millions rows x 139 columns). Columns CASEID, EMPIDORG, and PRODUCT are indexed. Program 11 joins OUT.CHFBM with the service table first, then with the case table. Program 12 joins the two large DB2 tables inside DB2 first, then joins with OUT.CHFBM. Their performances are:

<table>
<thead>
<tr>
<th>Programs</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (min)</td>
<td>12.03</td>
<td>9.73</td>
</tr>
<tr>
<td>EXCP</td>
<td>2681</td>
<td>5.01</td>
</tr>
</tbody>
</table>

DISCUSSIONS
A view descriptor is a SAS data view which can be used as a SAS data set. It defines some or all of the data described by an access descriptor, which describes a DBMS table's format and contents to the SAS System. If the view descriptor is used in a DATA step, PROC step, or PROC SQL query with a SAS WHERE statement, the SAS/ACCESS interface view engine will try to pass the WHERE conditions to the DBMS for processing as long as the clause is valid for the DBMS. Certain SAS conventions such as datetime formats can be translated to their DBMS-specific SQL equivalents.

PROC SQL Pass-Through facility establishes a connection with a DBMS and enables you to interact directly with the DBMS. You can send DBMS-specific (especially data formats) SQL statements to the DBMS for processing. The data will be returned in SAS formats to the SAS System.
The Pass-Through facility enables sending saving of program 8 over program 7. The efficiency of transferring saving of program 4 over program 3, and the 68.2% CPU time columns you specify. This is the main reason why the Pass-Through facility and view descriptors can pass from the SAS data set into the where clause. This is done in program 8. Program 2 failed at EXCP=36240 and CPU=80.94min. It took 80.02 CPU time to extract about 12 millions rows off the view descriptor, and then failed at PROC SORT because of insufficient space. It wouldn’t outperform PROC SQL in program 3 even if there was enough space. Therefore, it is not good to sort/extract DBMS data through view descriptors using PROC SORT or DATA step, unless it is necessary, or a very strict where clause can be specified to subset only a small part of the data as in program 5 and 6. But they still can’t beat their PROC SQL version as in program 7.

Both SAS/ACCESS view descriptors and PROC SQL Pass-Through facility use the SAS/ACCESS interface view engine to send and receive data between the DBMS and the SAS system. They both can pass the where clauses to DBMS for processing. In many cases, they should have equal efficiency. But in DB2, unlike some other DBMSs (INGRES, ORACLE, SYBASE, SQL Server, etc.), the KEEP and VAR statements are not passed to the DBMS through view descriptor for processing. So, all columns (110-140) are transferred from DB2 to the SAS System. At this point it is practical and reasonable that the view descriptors define the entire DB2 table (all rows and columns) to fit different end users and business needs. However, Pass-Through only transfers the columns you specify. This is the main reason why the Pass-Through has superior performance over the view descriptors on Highmark Data Warehouse. It can explain the 72.4% CPU time saving of program 4 over program 5, and the 88.2% CPU time saving of program 8 over program 7. The efficiency of transferring less columns (so data) can be seen by comparing program 10 with program 8. And not passing the KEEP statement to DB2 explains the equal efficiency among program 9, 7, and 10.

The Pass-Through facility enables sending DBMS-specific SQL statements to the DBMS for processing and then transferring the results back to the SAS system for further processing. It lets you access DBMS data more directly, more transparently, and with more control. The DBMS optimizer can take advantage of indexes on DBMS columns to process a query more quickly and efficiently, particularly when joining large tables as shown in program 12 (19.1% CPU saving and 81.3% EXCP saving compared to program 11). Also, Pass-Through enables DBMS optimizer to speed up the data summary (AVG, COUNT, SUM functions, GROUP BY clause, etc.). It is better than transferring a lot of data then summarizing them, which is the case when using view descriptors to summarize data.

However, when joining DBMS tables with SAS data sets, neither Pass-Through facility nor view descriptors can take advantage of the DBMS optimizer and indexes. They both have to transfer the data (after the where clause processing) from DBMS to SAS, and then let SAS process the join. This has two disadvantages: more data needs transferred and indexes are lost after the transfer. One way to work around this is to load the SAS data sets into DBMS temporarily using PROC DBLOAD. However, many DBMS sites do not allow users to create even temporary DBMS tables. Fortunately, Pass-Through facility and view descriptors can pass the where clauses to DBMS for processing. In most cases, it is more efficient for the DBMS to process the where criteria and then transfer a subset of rows to the SAS System for further processing. Therefore, another way is to embed all the key values from the SAS data set into the where clause. This is done in program 5, 6, 7, and 8 by using a macro variable %LISTKEY to list the key values. The performances are improved dramatically by about 99.7% CPU time saving over their respective non-macro-variable versions -- program 3 and 4 (program 1 and 2 ran out of resources).

The dramatic improvement on performance with macro variable %LISTKEY is obviously very attractive. However, when the SAS data set is large, it can’t be passed to DB2. You will get a DB2 error: statement is too long. DB2 enforces a 32KB limit on its SQL. The macro %JOINDB2 on Appendix II bypasses this restriction and automates the process for multiple DB2 tables.

Macro %JOINDB2 is self-documented. In brief, it calculates the number of key values for each pass (stored in &KEY1PASS) and the number of passes needed for all the unique key values in the SAS data set. This is done based on the key type (numeric or character) and its display length. Then a nested macro %LISTKEY is used to list the allowed number (&KEY1PASS) of unique key values into a macro variable %LISTKEY based on the looping control variable &I, which is used to loop through all unique key values. Nested macro %QUERY builds the DB2 query based on &LISTKEY, &DB2WHERE, and &DB2COLM. Then the last nested macro %JOIN1TBL calls %LISTKEY and %QUERY and append the data to a previous pass. Finally, %JOIN1TBL is called for each table, and another looping control variable &I is used to loop through all the DB2 tables. Data from each table are appended to the output data set &OUTDS.

When there are more than one key to join, select the key indexed on DB2 tables, or the key that can return less data. Then join/merge the returned data with the SAS data set on other keys.

Theoretically, the SAS data set can be of any size. But practically, it should not exceed 5% of the DB2 table volume according to an empirical rule formulated by John Polec. Otherwise you may not be able to have the performance gain based on the random access through indexes.

%JOINDB2 has been extensively used for many projects to improve productivity and deliver information even faster than expected. At the same time, it saves a lot of computing resources. As long as the key is indexed, you can always have substantial improvement on performance. Even if the key is not indexed, it may still give you a performance gain by avoiding the heavy data transfer from DB2.

Appendix III shows an example of using %JOINDB2. It joins the SAS data set OUT.MED99212 with 5 DB2 claim tables (total about 94 millions of rows). The key EMPID00 on OUT.MED99212 has 22733 unique key values. EMPID00 is the indexed key on the DB2 tables. It only took 1.13 CPU minutes.

CONCLUSION

The greatest impact on performance is caused by the transfer of data between the DBMS and the SAS System. Whenever possible, select only needed columns, put strict conditions on the where clause, and pass more processing to the DBMS. This will return less data and take advantage of DBMS’s power.

On DB2, PROC SQL Pass-Through outperforms view descriptors because the KEEP and VAR statements can’t be passed to DB2 with view descriptors. Also, Pass-Through facility can take advantage of DBMS optimizer and indexes. When joining DBMS tables with SAS data set, embedding the key values into the where clause passed to DBMS can improve the performance dramatically. This is where the power of macro %JOINDB2 comes from.

Because of its great performance gain, macro %JOINDB2 also attracts other Highmark Data Warehouse users with no or little SAS knowledge. Further enhancement of %JOINDB2 is to develop two macros %FLATIN and %FLATOUT for non-SAS-users. %FLATIN will read the key values embedded by the users. %FLATOUT will automatically write out the data retrieved from %JOINDB2 to a flat file using the standard Data Warehouse formats in the view descriptors. Furthermore, another
macro %QUERYDB2 in plan will be used to query DB2 tables only (without joining with external data sources), especially for non-SAS-users.

REFERENCES


ACKNOWLEDGMENTS

I would like to thank John Newell who reviewed this paper and gave me time to write it. Also, I’d like to thank JoAnn Matthews and Benjamin Venator for their reviews and suggestions.

SAS and SAS/ACCESS are registered trademarks of SAS Institute Inc. in the USA and other countries.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The programs on Appendixes are the properties of Highmark Blue Cross Blue Shield.

CONTACT INFORMATION

James Yang  
Highmark Blue Cross Blue Shield  
120 Fifth Avenue, Suite P2501  
Pittsburgh, PA 15222-3099  
(412)544-2433  
James.Yang@Highmark.com

### Appendix I

**Program 1 (PROC SORT with view descriptor)**

```
PROC SORT DATA=HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT WHERE=(PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D)) OUT=DRUG; BY EMPIDORG PATDOB; RUN;
DATA OUT.SVDSST;
MERGE DRUG(IN=D) OUT.CHFMB(RENAME=(BIRTH_DT=PATDOB) IN=C); BY EMPIDORG PATDOB; IF D AND C; RUN;
```

**Program 2 (DATA step with view descriptor)**

```
DATA DRUG;
SET HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT)
WHERE PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D;
RUN;
PROC SORT DATA=DRUG; BY EMPIDORG PATDOB; RUN;
DATA OUT.SVDS;
MERGE DRUG(IN=D) OUT.CHFMB(RENAME=(BIRTH_DT=PATDOB) IN=C); BY EMPIDORG PATDOB; IF D AND C; RUN;
```

**Program 3 (PROC SQL with view descriptor)**

```
PROC SQL STIMER;
CREATE TABLE OUT.SVSQL AS
SELECT  DRUG.* FROM  HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT)
AS DRUG ,
OUT.CHFMB AS CHF
WHERE PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D AND
DRUG.EMPIDORG=CHF.EMPIDORG AND PATDOB=CHF.BIRTH_DT;
%PUT SQLRC="&SQLRC" SQLOBS="&SQLOBS" SQLOOPS="&SQLOOPS";
QUIT;
```

**Program 4 (PROC SQL with Pass-Through facility)**

```
PROC SQL STIMER; CONNECT TO DB2(SSID=WDB2);
CREATE TABLE OUT.PTSQL AS
SELECT DRUG.* FROM CONNECTION TO DB2(SELECT EMPIDORG,PATDOB,NETPAY,DAYSSUPP,SVCDATE,LINEDISP,PRODUCT FROM DWH.V_RALS_DRUG
WHERE PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '1996-01-01' AND '1997-06-30')
AS DRUG ,
OUT.CHFMB AS CHF
WHERE DRUG.EMPIDORG=CHF.EMPIDORG AND PATDOB=CHF.BIRTH_DT;
%PUT SQLRC="&SQLRC" SQLOBS="&SQLOBS" SQLOOPS="&SQLOOPS";
%PUT SQLXRC="&SQLXRC" SQLXMSG="&SQLXMSG";
DISCONNECT FROM DB2;
QUIT;
```
Program 5 (PROC SORT with macro variable and view descriptor)

```
PROC SQL NOPRINT ;  SELECT DISTINCT TRIM(EMPIDORG) INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
PROC SORT DATA=HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT WHERE=EMPIDORG IN (&LISTKEY) AND PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D ) ) OUT=DRUG;  BY EMPIDORG PATDOB;  RUN;
DATA OUT.SVDSLKST;
  MERGE DRUG(IN=D) OUT.CHFMB(RENAME=(BIRTH_DT=PATDOB) IN=C);  BY EMPIDORG PATDOB;  IF D AND C;
RUN;
```

Program 6 (DATA step with macro variable and view descriptor)

```
PROC SQL NOPRINT ;  SELECT DISTINCT TRIM(EMPIDORG) INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
DATA DRUG;  SET HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT);
  WHERE EMPIDORG IN (&LISTKEY) AND PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D;
RUN;
PROC SORT DATA=DRUG;  BY EMPIDORG PATDOB;  RUN;
DATA OUT.SVDSLK;
  MERGE DRUG(IN=D) OUT.CHFMB(RENAME=(BIRTH_DT=PATDOB) IN=C);  BY EMPIDORG PATDOB;  IF D AND C;
RUN;
```

Program 7 (PROC SQL with macro variable and view descriptor)

```
PROC SQL NOPRINT ;  SELECT DISTINCT TRIM(EMPIDORG) INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
PROC SQL STIMER;
  CREATE TABLE OUT.SVSQLLK AS
  SELECT DRUG.*   FROM HIW.DW116ALL(KEEP=EMPIDORG PATDOB NETPAY DAYSSUPP SVCDATE LINEDISP PRODUCT)
  AS DRUG,    OUT.CHFMB AS CHF
  WHERE DRUG.EMPIDORG IN (&LISTKEY)
  AND PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D
  AND DRUG.EMPIDORG=CHF.EMPIDORG AND DRUG.PATDOB=CHF.BIRTH_DT;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
QUIT ;
```

Program 8 (PROC SQL with macro variable and Pass-Through facility)

```
PROC SQL NOPRINT ;  SELECT DISTINCT TRIM(EMPIDORG) INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
PROC SQL STIMER ;   CONNECT TO DB2(SSID=WDB2);
  CREATE TABLE OUT.PTLK AS
  SELECT DRUG.*   FROM CONNECTION TO DB(SELECT EMPIDORG,PATDOB,NETPAY,DAYSSUPP,SVCDATE,LINEDISP,PRODUCT
  FROM DWH.V_RALS_DRUG  WHERE
  EMPIDORG IN (&LISTKEY)
  AND PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '1996-01-01' AND '1997-06-30'
  ) AS DRUG ,           OUT.CHFMB AS CHF
  WHERE DRUG.EMPIDORG=CHF.EMPIDORG AND PATDOB=BIRTH_DT;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
%PUT SQLXRC=**&SQLXRC** SQLXMSG=**&SQLXMSG**;
DISCONNECT FROM DB2;
QUIT ;
```

Program 9 (PROC SQL with macro variable, view descriptor, but no KEEP statement)

```
PROC SQL NOPRINT ;  SELECT DISTINCT TRIM(EMPIDORG) INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
PROC SQL STIMER ;  CREATE TABLE OUT.SVSQLLK2 AS
  SELECT DRUG.EMPIDORG,PATDOB,NETPAY,DAYSSUPP,SVCDATE,LINEDISP,PRODUCT
  FROM HIW.DW116ALL AS DRUG,  OUT.CHFMB AS CHF
  WHERE DRUG.EMPIDORG IN (&LISTKEY) AND PRODUCT IN (60,80,81,82) AND SVCDATE BETWEEN '01JAN1996'D AND '30JUN1997'D
  AND DRUG.EMPIDORG=CHF.EMPIDORG AND DRUG.PATDOB=CHF.BIRTH_DT;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
QUIT ;
```
Program 10 (PROC SQL with macro variable, Pass-Through facility, but select all columns in Pass-Through)

```sql
PROC SQL NOPRINT;  SELECT DISTINCT "'||TRIM(EMPIDORG)||'" INTO :LISTKEY SEPARATED BY ',' FROM OUT.CHFMB;  QUIT;
PROC SQL STIMER;  CONNECT TO DB2(SSID=WDB2);
CREATE TABLE OUT.PTLKASSV AS
SELECT DRUG.EMPIDORG ,PATDOB,NETPAY,DAYSSUPP,SVCDATE,LINEDISP,PRODUCT
FROM   CONNECTION TO DB2(SELECT *  FROM DWH.V_RALS_DRUG
WHERE EMPIDORG IN (&LISTKEY)
AND PRODUCT IN (60,80,81,82) AND
SVCDATE BETWEEN '1996-01-01' AND '1997-06-30' ) AS DRUG,
OUT.CHFMB AS CHF
WHERE DRUG.EMPIDORG=CHF.EMPIDORG AND PATDOB=BIRTH_DT;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
%PUT SQLXRC=**&SQLXRC** SQLXMSG=**&SQLXMSG**;
DISCONNECT FROM DB2;  QUIT;
```

Program 11 (PROC SQL joins DB2 tables outside Pass-Through facility)

```sql
PROC SQL STIMER;  CONNECT TO DB2(SSID=WDB2);
CREATE TABLE OUT.PTJOS AS
SELECT  EMPIDORG, PATDOB, CA.CASEID, ADMDATE, NEXTADM, DSTATUS
FROM     (SELECT IP .*   FROM   CONNECTION TO DB2(SELECT DISTINCT EMPIDORG, PATDOB, CASEID
FROM DWH.V_RALS_IP_CLAIM) AS IP ,    OUT.CHFMB AS CHF
WHERE IP.EMPIDORG=CHF.EMPIDORG AND PATDOB=BIRTH_DT
WHERE IP.EMPIDORG=CHF.EMPIDORG AND PATDOB=BIRTH_DT
) AS IC,
CONNECTION TO DB2(SELECT CASEID,ADMDATE,NEXTADM,DSTATUS FROM DWH.V_RALS_CASE
WHERE PRODUCT IN (60,80,81,82) AND DCHDATE BETWEEN '1996-01-01' AND '1997-06-30' ) AS CA
WHERE CA.CASEID=IC.CASEID;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
%PUT SQLXRC=**&SQLXRC** SQLXMSG=**&SQLXMSG**;
DISCONNECT FROM DB2;
QUIT;
```

Program 12 (PROC SQL joins DB2 tables inside Pass-Through facility)

```sql
PROC SQL STIMER;  CONNECT TO DB2(SSID=WDB2);
CREATE TABLE OUT.PTJIS AS
SELECT  DW.EMPIDORG,PATDOB,CASEID,ADMDATE,NEXTADM,DSTATUS
FROM     CONNECTION TO DB2(SELECT DISTINCT IP.EMPIDORG,IP.PATDOB,CA.CASEID,ADMDATE,NEXTADM,CA.DSTATUS
FROM DWH.V_RALS_IP_CLAIM AS IP, DWH.V_RALS_CASE AS CA
WHERE CA.CASEID=IP.CASEID AND CA.PRODUCT IN (60,80,81,82) AND
CA.DCHDATE BETWEEN '1996-01-01' AND '1997-06-30' )   AS DW,
OUT.CHFMB AS CHF
WHERE  DW.EMPIDORG=CHF.EMPIDORG AND PATDOB=BIRTH_DT;
%PUT SQLRC=**&SQLRC** SQLOBS=**&SQLOBS** SQLOOPS=**&SQLOOPS**;
%PUT SQLXRC=**&SQLXRC** SQLXMSG=**&SQLXMSG**;
DISCONNECT FROM DB2;
QUIT;
```

Appendix II

```sql
/* PROC SQL Pass-Thru facility has superior performance over SAS/ACCESS views on our Data Warehouse (DB2). The greatest impact on performance is caused by the transfer of data between DBMS and the SAS System. * Pass-Thru facility passes queries to DBMS for processing and take advantage of DBMS optimizer and indexes, especially when joining * large tables or summarizing data inside DBMS. * Pass-Thru facility can go further with the macro %JOINDB2, when a SAS data set needs to join with large DB2 tables. This macro * lists all the unique key values from the SAS data set into a macro variable, and then passes it into the where clause inside the DB2 * query. It bypasses the length limit on SQL query set by DB2 (about 32KB, a little bit lower than the storage limit for macro variables * on MVS) by automatically calculating the passes needed and passing all key values to as many DB2 tables as you want. Also, it allows */
```
you to specify other selection criteria and the columns you need. The performance gain can bypass the CPU governor set by DB2
(195 seconds on our site). If the volume of the SAS data set is no more than 5% of the DB2 tables (10-35 millions for the claim tables),
macro %JOINDB2 may give you a substantial improvement on performance (from the empirical rule formulated by John Polec).

Also, this methodology (list key values into a macro variable and embed it into the where clause) can improve SAS/ACCESS views' performance dramatically, when joining SAS data set with the view descriptior. The SAS/ACCESS interface view engine will send the parts of the where clause that relate to the single DBMS table only off to the DBMS engine, and retrieve only the needed rows. This also gives the DBMS a chance to put the index to work.

Positional Parameters:

* JOINDS the SAS data set needed to join with the DB2 tables.
* KEY the name of the key variable in JOINDS.
* KEYLEN the length of KEY when fully displayed, especially for numeric KEY with more than 8 bytes (include decimal).
* TABLES the names of the DB2 tables (separated by ""," if more than one).
* DB2KEY the name of the key column on the DB2 tables.
* DB2COLM the DB2 columns needed.
* OUTDS the output SAS dataset name.

Keyword Parameters:

* KEYTYPE the data type of the KEY (C for character, N for numeric, default is C).
* DB2WHERE other selection criteria in the where clause of DB2 query (default is blank – no other criteria).

The Keyword parameters can be specified in any order, but must be after the last positional parameter. If they are omitted, the default values will take effect. The positional parameters must be specified and positioned in the same order as in the macro.

%MACRO JOINDB2(JOINDS,KEY,KEYLEN,TABLES,DB2KEY,DB2COLM,OUTDS, DB2WHERE= , KEYTYPE=C);

/*------ gather joined SAS data set info & calculate key values/per pass----*/
PROC SORT DATA=&JOINDS(KEEP=&KEY) NODUPKEY OUT=UNIQKEYS; BY &KEY; RUN;
DATA _NULL_ ;  IF 0 THEN SET UNIQKEYS NOBS=OBS; CALL SYMPUT('OBS', LEFT(PUT(OBS,8.)));  STOP; RUN;
/* get the number of OBS and store it in &OBS at compile time */
%put NOTE: There are &OBS unique &KEY in data set &JOINDS.;
%IF %UPCASE(&KEYTYPE)=N
%THEN %LET BYTE1KEY=% EVAL(&KEYLEN+1);                   /*1 counts for the comma separating numeric key values*/
%ELSE %LET BYTE1KEY=% EVAL(&KEYLEN+3);                   /*3 counts for the comma & apostrophe separating character key values*/
%LET KEY1PASS=% SYSFUNC(FLOOR(31000/&BYTE1KEY));     /* calculate key values per pass, 1KB reserved for long &DB2WHERE */
%LET PASS1TBL=% SYSFUNC(CEIL(&OBS/&KEY1PASS));       /* calculate passes needed for each table */
%put NOTE: &KEY1PASS &KEY can be passed in each pass.;
%put NOTE: &PASS1TBL passes are required for each table.;

/*--------- list key values into macro variable &LISTKEY---------*/
%MACRO LISTKEY;
PROC SQL NOPRINT;
%IF %UPCASE(&KEYTYPE)=N
%THEN %LET BYTE1KEY=% EVAL(&KEYLEN+1); /*1 counts for the comma separating numeric key values*/
%ELSE %LET BYTE1KEY=% EVAL(&KEYLEN+3); /*3 counts for the comma & apostrophe separating character key values*/
%LET KEY1PASS=% SYSFUNC(FLOOR(31000/&BYTE1KEY)); /* calculate key values per pass, 1KB reserved for long &DB2WHERE */
%LET PASS1TBL=% SYSFUNC(CEIL(&OBS/&KEY1PASS)); /* calculate passes needed for each table */
%put NOTE: &KEY1PASS &KEY can be passed in each pass.;
%put NOTE: &PASS1TBL passes are required for each table.;
QUIT;
%MEND LISTKEY;

/*----- pass query into DB2 to improve performance-----*/
%IF %BQUOTE(&DB2WHERE) NE %THEN %LET DB2WHERE=%STR(AND &DB2WHERE);
%MACRO QUERY;
PROC SQL;
CONNECT TO DB2(SSID=WDB2);
CREATE TABLE DAT1PASS AS
SELECT * FROM CONNECTION TO DB2 (SELECT &DB2COLM FROM &TABLE WHERE &DB2KEY IN (&LISTKEY) &DB2WHERE FOR FETCH ONLY);
%PUT SQLRC=""%SQLRC"" SQLOBS=""%SQLOBS"" SQLOOPS=""%SQLOOPS"";
%PUT SQLXRC=""%SQLXRC"" SQLXMSG=""%SQLXMSG"";
DISCONNECT FROM DB2;
QUIT;
%MEND QUERY;
/*----join with one table &PASS1TBL times----*/

%MACRO JOIN1TBL;
   %DO I=1 %TO &OBS %BY &KEY1PASS ;               /*pass &KEY1PASS key values per time*/
      %LISTKEY; %QUERY;
      /*concatenate data from each pass*/
   %END;
%MEND JOIN1TBL;

/*--------- join with all tables---------*/

%IF %SYSFUNC(EXIST(&OUTDS)) %THEN %DO;   PROC DELETE DATA=&OUTDS; RUN;    %END;         /*clear up before join*/
%IF %SYSFUNC(EXIST(DAT1TBL)) %THEN %DO;   PROC DELETE DATA=DAT1TBL; RUN;   %END;        /*clear up before join*/

%LET J=1; % LET TABLE=% SCAN(&TABLES,&J,',');
%DO %WHILE (&TABLE NE ); %JOIN1TBL;
PROC APPEND BASE=&OUTDS DATA=DAT1TBL; RUN;                  /*concatenate data from each table*/
PROC DELETE DATA=DAT1TBL; RUN;        /*avoid duplicate data through append in next run*/
%LET J=% EVAL(&J+1);       %LET TABLE=%SCAN(&TABLES,&J,'');
%END;
%MEND JOINDB2;

Appendix III

Example of Using %JOINDB2 (CPU Time: 1.13min, EXCP Count: 3490)

FILENAME MACRO 'PHEDI.RY1997.SASMACRO.BN.DATA';
OPTIONS ERRORS=2 BUFNO=8 MAUTOSOURCE SASAUTOS=MACRO;
%LET EXCLU=%STR ( ('290'<=DX1 AND DX1<'317') OR ('9650'<=DX1 AND DX1<'9651') OR ('9658'<=DX1 AND DX1<'9659') OR ('967'<=DX1 AND DX1<'968') OR ('9685'<=DX1 AND DX1<'9686') OR ('969'<=DX1 AND DX1<'970' ) );   /*EXCLU MH*/
%LET WHERE=%STR ( (PROC1='99212') AND NOT (&EXCLU) AND PATDOB <= '1977-12-31' AND
         SVCDATE BETWEEN '1997-01-01' AND '1997-12-31' AND LINEDISP IN ('1','4' )  AND
         (CLAIMTYP=3 AND CAPIND='Y' AND NOT (PLACE2 BETWEEN 1 AND 19 OR PLACE2=21) ) ); /*OP ENCOUNTER*/
%LET TABLES=%STR ( DWH.V_RALS_IP_CLAIM, DWH.V_RALS_OP_CLAIM1, DWH.V_RALS_OP_CLAIM2, DWH.V_RALS_OP_CLAIM3, DWH.V_RALS_OP_CLAIM4 );
%LET DB2COLM=%STR ( EMPIDORG,PATDOB,SVCDATE,OSVCPRV,LINEDISP,CLAIMNUM );

******PASS KEY VALUES OF EMPID00 & RETURN A SUBSET FROM 5 SERVICE TABLES*****;
%JOINDB2(OUT.MED99212,EMPID00,11,&TABLES,EMPIDORG,&DB2COLM,OUT.JOINDB2,DB2WHERE=&WHERE);

******************MATCH ANOTHER KEY: PATDOB*******************
PROC SORT DATA=OUT.JOINDB2(RENAME=(EMPIDORG=EMPID00)); BY EMPID00 PATDOB; RUN;
PROC SORT DATA=OUT.MED99212; BY EMPID00 PATDOB; RUN;
DATA OUT.JOINDB2; MERGE OUT.JOINDB2(IN=J) OUT.MED99212(IN=M); BY EMPID00 PATDOB; IF J AND M; RUN;