PROCESSING LARGE SAS® AND DB2® FILES:
CLOSE ENCOUNTERS OF THE COLOSSAL KIND

Judy Loren, ASG, Inc.
Alan Dickson, ASG, Inc.

Introduction

Over the last few years, a number of papers have been presented at SUGI and regional conferences discussing an increasingly common problem: how to extract data from a DB2 table when the list of keys you want is in SAS. Several techniques have been discussed and all of them work, up to a point. However, as the size of the SAS dataset increases, problems arise. Most of the techniques become less viable, and some fail altogether.

The ultimate answer to this problem lies in increasing the intelligence and power within the SAS/ACCESS interface engines themselves. Discussions with various representatives of SAS Institute indicate that the issue is less one of technological constraints, and more one of priorities. Until enough users request these enhancements, other items will get the developers' attention. As more and more sites migrate systems to DB2, the requests will undoubtedly grow, but if you are already experiencing these problems, perhaps it's time for a word with your friendly SAS Rep.

This paper will review briefly a variety of techniques to bring SAS datasets and DB2 tables together, discuss their pros and cons, and try to indicate situations where they are and are not suitable. An attempt is made to measure relative efficiencies by indicating common performance statistics such as EXCP counts and CPU time. Test results are shown at two levels: looking up 5 million keys from a 29 million record table, and looking up only 2,000 keys from the same table. We then provide guidelines for choosing among the techniques based on your circumstances.

The Problem

For the purposes of this paper we will use a real world example from one of our clients. The Customer Marketing System consists of a large normalized database containing information on 29 million customers. We frequently need to extract information on groups of these customers that number in the millions.

The specific example we will use is looking up the corresponding address ID numbers for a list of customer IDs. In the code that follows, the SAS dataset CUSTIDS contains one variable, CUST_ID. The name of the Access Descriptor for the DB2 table containing the address ID for every CUST_ID is CUST_ADDR.

Types of Techniques

The techniques we tried can be divided into 4 general categories:
1) SAS dataset and DB2 access descriptor;
2) using macros as an intermediary between the two;
3) both files in DB2;
4) both files in SAS.

Within these categories, you can use the SAS MERGE or SQL logic. And SQL can be used in SAS or passed through to DB2. Results vary markedly in resource use.

In addition, you need to be aware of the differences in the way multiple hits and no-hits are handled by SAS and SQL. In SAS, 2 identical keys being MERGEd with 3 identical keys results in 3 records; in SQL, 6 records will result. SAS will pass non-matches forward to the result file, SQL will not. You can regenerate these by merging your original SAS dataset with the results of the join. [SQL under SAS has a LEFT JOIN that will accomplish the same thing; this is not available in DB2 and cannot be used with pass-through.]

SAS Merge using a SAS file and an Access Descriptor

From the point of view of the programmer, the easiest way to merge SAS datasets with DB2 tables is using the MERGE statement and the Access Descriptor for the DB2 table.

DATA MATCH;
MERGE SAS.CUSTIDS (IN=WANTED)
   DBPl.CUST_ADDR;
   BY CUST_ID;
   IF WANTED;

This technique sends a request to DB2 for the whole table ORDERed BY the BY variable CUST_ID.

SELECT "CUST_ID", "ADDR_ID"
FROM "PROD"."AA003IT_CUST_ADDR"
ORDER BY "CUST_ID"
FOR FETCH ONLY

DB2 has to create a sorted copy of the whole lookup table in workspace before sending it over to SAS. This technique does work, but the time and resources involved sent us looking for alternatives.
**SQL Without Pass-Through**

Since Version 6 of the SAS System was released, PROC SQL has been an integral part of the base product. It can be used against SAS datasets or DB2 tables (via Access Descriptors) equally easily. The following code was tried:

```sql
PROC SQL;
  CREATE TABLE MATCH AS
  SELECT CUST_ID, ADDR_ID
  FROM DBP1.CUSTADDR A,
       SAS.CUSTIDS B
  WHERE A.CUST_ID = B.CUST_ID;
```

Again, the generated code will cause DB2 to ship the entire table across the interface, although without the ORDER BY clause this time.

```sql
SELECT "CUST_ID", "ADDR_ID"
FROM "PROD"."AA003IT_CUST_ADDR"
FOR FETCH ONLY
```

This technique failed for the full 5 million OBS due to insufficient space in a WORK.UTILITY file. We did get a successful run at 2,000 OBS, but the CPU time at that level approached that of the MERGE at 5 million. We had to look elsewhere for improvements in performance.

**Pass-Through**

In Technical Report P-221, SAS introduced pass-through, a technique that allows you to hand SQL directly to DB2 for execution. Since DB2 must execute the query as written, you use DB2 table names rather than access descriptors, and you can use only DB2 SQL. Not all features of PROC SQL in SAS are available in DB2; for example, DB2 will not recognize LEFT JOINs or CASE logic.

In the techniques that follow, pass-through was found to perform better than using access descriptors.

**Using Macros 1: Generating List for WHERE Clause**

In the two examples we showed above, inefficiencies arose because SAS was not making use of indexes on the DB2 lookup table. In both cases, a sequential pass of the lookup table was required. You can allow DB2 to use its index by passing explicit values from SAS across the interface.

One technique for doing this involves using a macro to generate a long WHERE clause on your SQL. This technique has been presented at a number of regional and national SAS conferences, by Paul Kent of SAS Institute, Craig Gavin of Atlantic Research Corporation, and others. In essence, the macro takes the observations in the SAS dataset and creates a list of values to be used with the IN operator. In the code that follows, S2000 is a 2,000 OBS sample of our CUSTIDS file.

```sql
%MACRO WRITE;
  %LET VAR=WHER;
  %DO I=1 %TO &NUMWHERE;
    &&&VAR&I
  %END;
%MEND WRITE;

DATA _NULL_;
  LENGTH Y $ 20;
  SET SAS.S2000 NOBRS=TOTAL END=DONE;
  IF _N_=1 THEN CALL SYMPUT('NUMWHERE',PUT(TOTAL,4.));
  IF NOT DONE THEN Y="" ||CUST_ID|| ',' ;
  ELSE Y="" ||CUST_ID|| '' ;
  CALL SYMPUT('WHERI',LEFT(_N_),Y);
RUN;
```

```sql
PROC SQL;
  CONNECT TO DB2 (SSID=DBP1);
  %PUT &SQLXMSG;
  CREATE TABLE MATCH AS
  SELECT * FROM CONNECTION TO DB2
  (SELECT CUST_ID, ADDR_ID
   FROM PROD.AA003IT_CUST_ADDR
   WHERE CUST_ID IN (%WRITE));
  %PUT &SQLXMSG;
  DISCONNECT FROM DB2;
  %PUT &SQLXMSG;
```

We tested this technique and found that it does indeed work and work well for 2,000 observations, but it failed when we tried 2,250 observations. We suspect this was due to the length of the WHERE clause. The number of observations this technique can handle is probably related to the length of each value. In our case, the Customer Id numbers in the list were each 11 digits long.
Using Macros 2:
Embedding Single Indexed Accesses in a Loop

As we have already discussed, SAS/ACCESS will take advantage of the indexes when hard-coded values are passed to the interface. Can we loop through the SAS dataset of keys and generate a query for each? Absolutely!

```sas
OPTIONS NONOTES NOSOURCE NOSOURCE2 ERRORS=!O;
DATA_NULL;
SET SAS.S2000 NOBS=MAXOBS;
CALL SYMPUT(,MAX',MAXOBS);
STOP;
RUN;

%MACRO LOOP;
%DO 1=1 %TO &MAX;
  DATA....NULL_; INPT = INPUT("&I",BEST8.);
  SET SAS.S2000 POINT=IN_PT;
  CUST_IDX="ICUST_ID";
  CALL SYMPUT(IN_CUST,CUST_IDX);
  STOP;
RUN;

%ENDLOOP;

0728
```

This technique does work, although you may want to suppress your SASLOG. With approximately 25 lines of LOG being generated for each key you look up, it will eat up spool space at a phenomenal rate. Specifying //SASLOG DD DUMMY in the JCL, or using OPTIONS NONOTES NOSOURCE2; will suppress your SAS log. The latter approach has the advantage of giving error messages if they occur, but PROC SQL has a known bug which generates a blank line for every key you access, so the SASLOG will still get quite large for large numbers of keys.

Not surprisingly, this technique generates extremely large EXCP counts and the CPU time is relatively high. Because of this we discontinued testing at 15,000 records, but it does provide a solution for a one-time need where the number of keys is in the thousands.

Use of MODIFY

Another technique for getting SAS to use DB2 indexes when looking up values from a SAS dataset in a DB2 table was suggested by Paul Kent in a paper presented at SUGI 17 in Hawaii. This was to use the MODIFY statement to retrieve the record you need, but not actually UPDATE the record. This will not work unless you have update access to the DB2 table. In our case, the tables we had to read were in production, and we could not have update authority under any circumstances.

DB2 and DB2

What about letting DB2 handle the merge? You can load the SAS dataset into a DB2 table and use pass-through SQL to perform the subsetting and return a SAS dataset. (A DB2 utility may be more efficient than PROC APPEND when loading very large tables. See your DB2 administrator for more information.) This technique does allow DB2 to make use of its indexes. Whether or not DB2 will use the index can be determined by running an EXPLAIN on the SQL you plan to use. Check with your local DB2 administrator for additional help.

```sas
PROC SQL;
CONNECT TO DB2 (SSID=DBPl);
CREATE TABLE MATCH AS
SELECT * FROM PROD.AA0071T_CUST_IDS A,
     PROD.AA0031T_CUST_ADDR B
WHERE A.CUST_ID = B.CUST_ID;
DISCONNECT FROM DB2;
RUN;

PROC APPEND BASE=MATCH DATA=TEMP;
RUN;
%END;
%MEND LOOP;
```

The problem with having DB2 handle the merge is not in DB2 itself but in the way DB2 is administered. Table creation is often restricted or completely prohibited. If you have one lookup function that you use frequently, you may be able to get your system administrator to create a one-column table that you can reuse as needed. Each time you have a list of keys to lookup, you can load them into that table, perform the merge in DB2, create the resulting SAS dataset, then empty the table for the next time. This solution works well when 1) you are using one lookup repeatedly and 2) the lookup table resides in the same subsystem as your one-column table.
Note that SQL without pass-through will have the same problems noted earlier, even when both tables are in DB2. And while it is possible to define an access descriptor to the SAS dataset now loaded into DB2 and MERGE the two files using access descriptors, our results indicate this is the worst of both worlds (as one would expect).

SAS and SAS

While the DASD requirement might make this option seem undesirable and not viable, copying the lookup table into SAS and performing the MERGE in SAS actually ranks high in terms of CPU utilization. Figuring in the cost of moving the lookup table into SAS, this technique is even more efficient than the DB2 join provided you lookup 4 or more lists against one copy of the table. Getting the SAS dataset in key variable sequence can be done with an ORDER BY on the extracting SQL or with PROC SORT. The former is more efficient if an index exists for the key variable on the lookup table in DB2.

Once the lookup table exists as a SAS dataset, you can choose from three techniques. One is the standard SAS MERGE, one is PROC SQL and one is Indexed Lookup (new in version 6.07). The SAS MERGE requires that both files be sorted or indexed by the BY or key variable, and will carry through key values that don't hit the lookup table to the output file. PROC SQL costs more and treats multiple keys in both files differently from the SAS MERGE. Indexed Lookup requires an index on the lookup table for the key variable, and returns only one observation per key value.

Indexed lookup makes use of the KEY= option on the SET statement, introduced in SAS Technical Report P-222, Changes and Enhancements to Base SAS Software Release 6.07. Once an index has been created for a dataset on a variable, the KEY= option allows direct access to that dataset on values of that variable.

Indexed Lookup:

PROC DATASETS LIBRARY=SAS2;
MODIFY CUSTADDR;
INDEX CREATE CUST_ID;

DATA MATCH;
SET SAS1.CUSTIDS;
SET SAS2.CUSTADDR KEY=CUST_ID;

This technique is extremely efficient in terms of CPU time, especially at low volumes, once the index has been created. Its limitation is that it will return only one row of the lookup table per key value.

Results

Tables 1 and 2 show the statistics on test runs of the techniques discussed above. Table 1 pertains to very large lists of keys (5 million in our example); Table 2 refers to small lists (in the thousands). In both cases the lookup table in DB2 contained about 29 million rows.

For the record, these tests were run on a 9021-740 CPU running MVS/ESA, Version 2.3 of DB2 and Version 6.07 of SAS. Conditions were not constant across all tests. Because we were using live tables in production, we had to use resources as we could find them. Hence, there were updates that occurred between tests of different techniques, accounting for some of the differences in counts. Also, the lookup table was reorganized between the MERGE and PROC SQL tests on the 2,000 OBS SAS file and the DB2 access descriptor. That PROC SQL performed better than the MERGE at this level may be due to that reorg. Judging from the same tests at the 5 million level, PROC SQL is actually less efficient.

We mention the inconsistency of test conditions only to direct attention away from small discrepancies in counts; the table shows clearly the significant savings from choosing your access technique wisely. The most efficient methods are an order of magnitude removed from the least efficient. Note that lookup is most efficient when both files are in the same environment. Not surprisingly, if you are going to look up one list, it makes the most sense to move that list into the DB2 environment and perform a join there. However, the one-time merge cost in SAS is so much lower that it doesn't take many lookups to justify the cost of moving the entire lookup table into SAS.

At lower volumes, the savings of using a direct lookup technique instead of a sequential pass more than justify the additional coding effort. And once you have set up the code, it can be reused easily.

Conclusion

This research was done because of a real world need to bring together large SAS files and large DB2 tables. When these files both contain millions of records, inefficiencies that seem small in theory pass the point of mere significance and become obstacles. The tests were designed to discover the best way of accomplishing huge merges between SAS and DB2. We have focused almost exclusively on CPU time, although EXCP counts are included in the results table. We derived the following guidelines for our own use and offer them for your consideration.
Suggested Guidelines

Note: The key points of the following discussion are diagrammed in Figure 1.

The first consideration is the number of keys you need to look up. If it is less than 2,500, then the generated WHERE Clause approach is best. Otherwise, the next factor is frequency of access relative to the frequency of update of the lookup table.

For one-shot or infrequent accesses, the best solution is to write the list of keys into a DB2 table and perform the join in DB2. If you are not allowed write authority to DB2, the technique of looping through single indexed accesses works reasonably well for lists of keys in the tens of thousands. It does, however, have some disadvantages such as the need for log suppression.

For situations where the list of keys is hundreds of thousands, or millions, or where multiple accesses will be needed against the lookup table, extracting the entire DB2 table into SAS is the best solution.

To get your SAS copy in proper sequence for a MERGE, you can use either ORDER BY in your SQL code or PROC SORT afterward. The former is probably better when a DB2 index exists on the key variable. Again, use of the DB2 EXPLAIN can provide some guidance.

Once you have decided to extract the lookup table into SAS, the following will help you decide which SAS technique to use. If you confidently expect only one record in your lookup table (now in SAS) for each key and you want to extract data frequently from this table, your most efficient technique is to create an index on the lookup table for the key variable and use Indexed Lookup to extract your data. However, if you suspect you will get multiple lookup table records for one key value, or you will not extract data from the lookup table very frequently, you should use a simple SAS merge to lookup your data.

From the results discussed above, an additional "hybrid" technique may have suggested itself to you as it did to us. That is, if the list of keys could be pre-processed (by a macro, for example) into subsets which would be smaller than the breakpoint of the generated WHERE Clause approach, then the subsets could be run separately and the output concatenated with PROC APPEND as was done in the Loop technique.

We did not have the code for this technique fully developed by press time for this presentation, but you can obtain an update on our investigations by contacting us as outlined below.

References


Scott, Stephen [1993] "Why We Replaced DB2 Software with SAS Software as a Relational DBMS for a 30-Gigabyte User Information System" pp 187-196 SUGI 18 Proceedings

Trademarks

SAS and SAS/ACCESS are registered trademarks of SAS Institute Inc., Cary, NC.

DB2 is a registered trademark of IBM Corporation.

For Further Information

The authors can be reached by mail at
P. O. Box 306
Cumberland Center, ME 04021
by phone at
(207) 846-8957
or through the Internet at
71311.3341@compuserve.com

We welcome feedback, new ideas, and information on your experiences with DB2 and SAS.
## Table 1
### TEST RESULTS
Looking up 5 Million OBS

<table>
<thead>
<tr>
<th>TECHNIQUE</th>
<th>EXTRACT FILE SIZE</th>
<th>EXCP</th>
<th>TCB</th>
<th>CLOCK</th>
<th>1/ No Hits</th>
<th>2/ Mult. Keys</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SAS &amp; DB2 ACCESS DESCRIPTOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS MERGE</td>
<td>5,422,987</td>
<td>5,903</td>
<td>197.45</td>
<td>760.00</td>
<td>Y</td>
<td>S</td>
<td>Insufficient space in file <a href="mailto:WORK.@T00002.UTILITY">WORK.@T00002.UTILITY</a></td>
</tr>
<tr>
<td>PROC SQL</td>
<td>Failed</td>
<td>40,456</td>
<td>269.82</td>
<td>304.40</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>2. Using Macros</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated WHERE IN clause</td>
<td>Failed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Works only up to 2,250 OBS</td>
</tr>
<tr>
<td>Series of Single Indexed Lookups</td>
<td>17,547</td>
<td>399,000</td>
<td>18.41</td>
<td>139.30</td>
<td>N</td>
<td>J</td>
<td>Too inefficient for 5 million; tested at 15,000</td>
</tr>
<tr>
<td>3. DB2 &amp; DB2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load file of keys into DB2</td>
<td>5,098,525</td>
<td>5,222</td>
<td>1.82</td>
<td>6.70</td>
<td></td>
<td></td>
<td>DB2 utility more efficient than PROC APPEND</td>
</tr>
<tr>
<td>Pass-through SQL Join of 2 DB2 tables</td>
<td>6,181,881</td>
<td>3,039</td>
<td>25.02</td>
<td>135.10</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>MERGE Access Descriptors</td>
<td>5,605,610</td>
<td>2,840</td>
<td>222.78</td>
<td>1,101.50</td>
<td>Y</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>PROC SQL on Access Descriptors</td>
<td>Failed</td>
<td>48,240</td>
<td>269.03</td>
<td>318.30</td>
<td>N</td>
<td>J</td>
<td>Insufficient space in file <a href="mailto:WORK.@T00002.UTILITY">WORK.@T00002.UTILITY</a></td>
</tr>
<tr>
<td>4. SAS &amp; SAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract CUST_ADDR into SAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ORDER BY more efficient if DB2 index exists</td>
</tr>
<tr>
<td>SAS MERGE</td>
<td>5,422,987</td>
<td>17,549</td>
<td>7.54</td>
<td>18.80</td>
<td>Y</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>PROC SQL</td>
<td>5,973,709</td>
<td>16,722</td>
<td>10.00</td>
<td>24.20</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>PROC SQL LEFT JOIN</td>
<td>5,991,647</td>
<td>12,724</td>
<td>10.37</td>
<td>46.30</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>Indexed Look-up</td>
<td>5,098,525</td>
<td>73,224</td>
<td>6.55</td>
<td>112.60</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create Index on CUST_ADDR Extract</td>
<td>103,000</td>
<td>13.06</td>
<td>36.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1/ Y=Keys that don't find records on lookup table are carried through to output file with missing lookup data.  
N=Keys that don't find records on lookup table do not appear on output file.  
2/ S=SAS MERGE logic governs multiple keys on both files.  
J=SQL Join logic governs multiple keys on both files.  
N=Only one lookup record returned for each key.
## Table 2

### TEST RESULTS

Looking up 2,000 OBS

<table>
<thead>
<tr>
<th>TECHNIQUE</th>
<th>EXTRACT FILE SIZE</th>
<th>EXCP</th>
<th>TCB</th>
<th>CLOCK</th>
<th>1/ No Hits</th>
<th>2/ Mult. Keys</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. SAS &amp; DB2 ACCESS DESCRIPTOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS MERGE</td>
<td>2,344</td>
<td>596</td>
<td>191.98</td>
<td>519.40</td>
<td>Y</td>
<td>S</td>
<td>Before Re-Org.</td>
</tr>
<tr>
<td>PROC SQL</td>
<td>2,350</td>
<td>626</td>
<td>173.38</td>
<td>233.50</td>
<td>N</td>
<td>J</td>
<td>After Re-Org.</td>
</tr>
<tr>
<td><strong>2. Using Macros</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated WHERE IN clause</td>
<td>2,339</td>
<td>765</td>
<td>0.10</td>
<td>4.30</td>
<td>N</td>
<td>S</td>
<td>Works only up to 2,250 OBS</td>
</tr>
<tr>
<td>Series of Single Indexed Lookups</td>
<td>2,341</td>
<td>55,165</td>
<td>2.47</td>
<td>23.00</td>
<td>N</td>
<td>J</td>
<td>Works only up to 15,000 OBS</td>
</tr>
<tr>
<td><strong>3. DB2 &amp; DB2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>4. SAS &amp; SAS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extract CUST_ADDR into SAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS MERGE</td>
<td>2,343</td>
<td>12,297</td>
<td>75.53</td>
<td>195.90</td>
<td>Y</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>PROC SQL Join</td>
<td>2,339</td>
<td>12,304</td>
<td>5.66</td>
<td>12.30</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>PROC SQL LEFT JOIN</td>
<td>2,344</td>
<td>12,297</td>
<td>5.51</td>
<td>13.50</td>
<td>N</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>Indexed Look-up</td>
<td>2,000</td>
<td>4,581</td>
<td>0.02</td>
<td>2.00</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create Index on CUST_ADDR Extract</td>
<td>103,000</td>
<td>13.06</td>
<td>36.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1/ Y=Keys that don't find records on lookup table are carried through to output file with missing lookup data.
N=Keys that don't find records on lookup table do not appear on output file.

2/ S=SAS MERGE logic governs multiple keys on both files.
J=SQL Join logic governs multiple keys on both files.
N=Only one lookup record returned for each key.

At press time no tables existed for testing.
Figure 1
Guidelines Flowchart

<2,500

Size of Key List?

> 2,500

Frequency of Access?

<= 3

Write Access to DB2?

No

Yes

Size of Key List?

> 1/500th of Lookup Table

Less than 1/500th of Lookup Table

Generated WHERE clause

DB2 Join

Single Indexed Access Loop

SAS Extract of Lookup Table