Access DB2 Tables Using Keylist Extraction

Berwick Chan, Kaiser Permanente, Oakland, Calif

Raymond Wan, Raymond Wan Associate Inc., Oakland, Calif

Introduction

The performance issue associated with DB2 is now very popular and certainly volumes can be written on it. Ever since the ACCESS product for DB2 appeared the SAS community has been involved in this debate. There have been quite a few papers written on this subject over the last few years. The most recent are the papers by Stephen Scott(1), and Judy Loren and Alan Dickson(2) at SUGI in Dallas earlier this year. This paper is not concerned with the wider problem of database design or rescuing a mainframe that has ran out of steam when SAS programmers try to lay claim to data in a DB2 data warehouse. Instead, we are trying to determine what happens with SAS and DB2 in a somewhat representative data extraction operation. The objective is to get a handle on what ate up the CPU cycles in these routine operations, hoping that this can help programmers in designing the most efficient way to carry out these operations. This is not a comparison of the relative efficiency of SAS and DB2. They are designed to serve different purposes and have different capabilities and their overheads are therefore different.

The operation that will be analyzed in this paper is table look-up. The main data table in our example consists of two fields. One is a sequential numeric key and the other a 10 byte character data field. The second table has one field which is a randomly selected subset of the sequential numeric key in the first table (see Appendix B for the SAS code used to create the tables). This second table is matched to the first table by the key field and then a file containing the matched subset is produced. Testing is done on two platforms. One is a 486DX2 PC with 16M of main memory running OS/2, SAS 6.08 and DB2/2. The test tables have 100,000 and 200 rows respectively. The other test bed is an MVS mainframe (#IBM3090) running SAS 6.07 and DB2 version 2. CPU time as reported by the SAS log is used to compare the various techniques. The operation is carried out interactively on the PC and in batch on the mainframe. This makes it possible to analyze the problem

independently of the application environment and operating system.

First this paper will look at what happens to the merge/join operation if both tables are in SAS. We will look at the various ways this could be carried out and why some ways are faster than others. This will provide a baseline measure of efficiency for comparison against the results of the following studies. Then, both SAS datasets are set up as DB2 tables and the various ways of obtaining the extracted data in SAS are discussed. Lastly, the impact of having the main data table in DB2 while the smaller tables is in SAS is explored. Test statistics of three cases are printed in Appendix A.

Case 1: Both files in SAS

Three methods were used here. One is a straight merge with a BY statement. The second uses the SET with a KEY= option. The third uses PROC SQL to join the two files. These are all tried and true methods. The main difference among them is how I/O is handled. The main findings are:

- An index does not speed up in MERGE. However the second and third method can take advantage of the index and thus speed up the operation.
- PROC SQL can benefit from an index but it is usually slower than data steps. However the dataset does not need to be in sort order. Without an index this is a slow method because the join is a Cartesian product and therefore has to perform 200x100,000 comparisons in our study.
- The second method is the fastest but there is overhead in creating the index.
- It is not clear whether the non-unique index can significantly affect the result.
- The results in MVS are concise with OS/2.

Case 2: Both tables in DB/2

In this case, the users have a lot more choice in methods. The biggest hurdle is data base administration. In most sites users do not have the freedom to set up tables in production regions. Sometimes they cannot even create views. We avoided this problem by performing our testing on a PC.

In table 2a, both test tables reside in DB2 and a view of table A joined to table B is created. All of the tests access this view from SAS. All the results are similar, as expected. In this instance ACCESS views work fine because only the product from the join is moved from DB2 to SAS.

In table 2b the same DB2 tables are here but view was not used. In this instance it is clear that ACCESS view is at a disadvantage because all the tables have

to be shipped from DB2 to SAS. Here the PASSTHRU method allows the user to bypass this and let DB2 do the work first, resulting in a 40 fold improvement in speed. This performance gap will increase with an increase of the table sizes.

Other findings are:-

- Users have more flexibility if the view can be created in DB2.
- Access view cannot make use of an index in DB2 tables because it is a sequential engine.
- Uniqueness of index makes a significant difference in results. In our case it is 40 times slower than the same operation on unique index.

Case 3: Large table in DB2 and extraction keys in SAS

This is the situation most often faced by SAS users. The DB2 table is indexed because otherwise the operation would generally be impractical if large tables are involved (large DB2 tables are generally indexed anyway). Without the use of an index, the two fast methods in this case (the code generation methods) would be even slower than the two 'standard' methods.

The first 'standard' method used an ACCESS view in a DATA STEP MERGE. The second one used the PASSTHRU facility in PROC SQL. These methods are slow because, once again, the table has to be moved from DB2 to SAS before joining. The ACCESS view is slower than SQL because the table is extracted with an 'ORDER BY' condition in DB2.

Several 'non-standard' methods have been published previously (refer to papers by Paul Kent and Judy Laren). The main idea here is to get the extraction key list to DB2 by embedding them in some kind of dynamic SQL code to be subsequently sent to DB2. The result of the SQL code is then sent back to SAS, thus avoiding sending the whole table from DB2 to SAS as in the previous two methods. The variations are therefore in how the SQL code is written. Here, SAS code is generated in a DATA STEP and then INCLUDED back. This is generally easier to write and maintain compared to macro based methods. (See Appendix B for SAS code generator)

One limitation mentioned by Loren's paper was that there is a 32,765 byte limit existing on the length of a single SQL code segment sent to DB2. A long extraction key list cannot therefore be sent in one single step. This can be handled by breaking up the SQL code into multiple segments and then use PROC APPEND to concatenate the pieces back together. Before this limit is reached, the DB2 optimizer could suspend the use of the index or even stop the operation. On the PC, it is safer to use no more that 12 keys in each SQL segment and

on the mainframe the limit should be set to about 140.

In the first code generation method the SQL code used an 'IN' predicate followed by a list of extraction keys. The key list is then used to do a table lookup on the DB2 table. The second method used a single key extract and then used a 'UNION' verb to concatenate the extracted rows. The second method does not work well on the PC because the names of the columns are lost and DB2 generates column names using its own convention. This does not happen in MVS. Also in the SELECT statement the 'FOR FETCH ONLY' clause enables DB2 to fetch rows more efficiently.

Conclusion

If one looks at the fastest methods in each group the ratios would be approximately All SAS FILES : ALL DB2 TABLES : Mixed TABLES = 1 : 5 : 28. Even though this ratio may change with size of tables and other operating factors, it is clear that if possible one should try not to mix DB2 tables and SAS files.

Some additional conclusions are:

Things to avoid :

- Moving large amounts of data from DB2 to SAS or vice versa.
- Cartesian joins.
- Request data with an 'order by' condition as view descriptor does.

Must dos :

- Careful planning.
- Make use of index if possible.
- Use debug options to examine traffic between DB2 and SAS.
- Use DB2 EXPLAIN tools to improve efficiency of SQL code.

References

(1) Scott, Stephen (1994) 'SAS vs DB2 as a Relational DBMS for END Users' paper presented at SUGI 19

(2) Loren, Judy & Dickson Alan 'Processing Large SAS and DB2 Files' paper presented at SUGI 19

(3) Kent, Paul (1992) 'Fun with the SQL Procedure - An Advanced Tutorial' pp 167-175 SUGI 17 Proceedings

(4) Kent, Paul 'Inside Proc SQL's Query Optimiser', SAS Institute Inc., Cary, N.C.

Table 1 : Both files in SAS

Method	PC / No Index	PC / Indexed	MVS / No Index	MVS / Indexed
Data step, merge by key	6.6 sec	30 sec.	1.23 sec	2.45 sec
Data Step, set key=	NA	1.04 sec	NA	.09 sec
Proc SQL	11.2 sec.	1.3 sec.	1.11 sec	.14 sec

Table 2a : Both tables in DB2/2
(indexed, on PC only, view created in DB2)

Method	CPU Time
Passthru, use view created in DB2	5.05 sec
Access view, data step	6.00 sec
Access View, sas datasets created by Proc SQL	5.15 sec.

Table 2b : Both tables in DB2/2
(indexed, on PC only, no DB2 view)

Method	CPU Time
Passthru, join table in DB2 region	5.12 sec
Access view, data step, merge by key	232 sec
Access view, both tables join by Proc sql in SAS	343 sec.

Table 3 : Large table in DB/2 smaller one in SAS

Method	PC / Indexed	MVS / Indexed
Data step, merge access view and SAS dataset	238 sec	7.97 sec
Proc sql, pass thru	188 sec	8.85 sec
Code generation, with 'IN' predicate	28.41 sec	.95 sec
Code generation, with 'UNION' verb	28.26 sec	.92 sec

APPENDIX B

```
/* Codes that create SAS data files */

DATA A (drop=i) ;
DO i = 1 TO 100000;
  AKEY +1 ;
  FIELD = 'ABCD' || PUT (AKEY,Z6.0) ;
  OUTPUT ;
END ; RUN ;

DATA B (DROP=I) ;
DO I = 1 TO 100000 ;
  BKEY + 1 ;
  IF RANUNI(805372) LT .002 THEN
  OUTPUT ;
END ;
RUN ;


/* Codes that generate dynamic SQL using  */
/* 'IN' predicate                         */

FILENAME OUT 'C:\SAS608\KEYLIST.TXT' ;
DATA C ; AKEY=. ; FIELD='        ' ; RUN ;

DATA _NULL_ ;
 SET B END=EOF ;
 FILE OUT ;
 IF MOD(_N_,11)=1 THEN
PUT
 'proc sql ;'
/'connect to dbmgr (database=TEST) ;'
/'create table CC as'
/'select * from connection to dbmgr ('
/'  select * from A where AKEY in (' akey ',' ;

IF MOD(_N_,11) NOT IN (0,1) AND NOT EOF THEN
PUT akey ',' ;

IF (MOD(_N_,11)=0 OR EOF) THEN
PUT  akey ') for fetch only) ;'
/'QUIT; '
/'PROC APPEND BASE=C DATA=CC ; RUN ; ' ;
RUN ;

%INCLUDE OUT / NOSOURCE2 ;

/* Codes that generate dynamic SQL using */
/* 'UNION' verb                          */

FILENAME OUT 'C:\SAS608\KEYLIST.TXT' ;

DATA C ; AKEY=. ; FIELD = '        ' ; run ;

DATA _NULL_ ;
 SET B END=EOF ;
 FILE OUT ;
 IF MOD(_N_,15)=1 THEN
PUT
 'proc sql ;'
/'connect to dbmgr (database=test) ;'
/'create table cc as'
/'select * from connection to dbmgr (' ;

PUT
' select * from a where akey=' AKEY @;

IF NOT (MOD(_N_,15)=0 OR EOF) THEN PUT
' union ' ;

ELSE PUT
') ;'
/'quit; '
/'proc append base=c data=cc; run ; ' ;

RUN ;
%INCLUDE OUT ;
```