Paper 76-28

Comparative Efficiency of SQL and Base Code When Reading from Database Tables and Existing Data Sets

Steven Feder, Federal Reserve Board, Washington, D.C.

ABSTRACT

In this paper we compare SQL code to BASE code, looking at both DATA step and PROC code. We make these comparisons reading data from both DB2 tables and existing SAS© data sets.

INTRODUCTION

Running SAS Version 8.1 on MVS, we start by reading data sets and build to summarizing data. Overall we find that SQL code performs more efficiently than BASE when reading DB2 tables, but PROC's can be superior for summarizations when reading existing data sets. We also find the same results when running SAS on NT, reading the same DB2 tables and data sets. To exclude the effect of other programs running on the system, the comparisons are limited to CPU time. Elapsed times are noted as a precaution against major differences in non-CPU time affecting the directions of results.

READING DATA FROM DB2

Starting with the basics and reading 1,000,000 observations from a DB2 table, there was no difference between using SQL and DATA step code.

```
options obs=1000000;
libname in db2 ssid=dsn authid=cra;
data sasout.crt_msa_area;
set in.crt_msa_area;
run;
proc sql;
create table sasout.crt_msa_area as
select * from in.crt_msa_area;
quit;
```

	Time (Minutes:Seconds)	
Code Type	CPU	Elapsed
SQL	01:23.0	01:37.0
DATA STEP	01:23.4	01:33.6

Similarly, using a WHERE clause, there was no difference when subsetting with an IN operator for 6 values with 375,408 resulting observations, or when subsetting for one value with 110 observations resulting, all read again from the DB2 table. For a simple read of a table, SQL and DATA step operate with similar efficiency.

```
libname in db2 ssid=dsn authid=cra;
data sasout.crt_msa_area_2;
set in.crt_msa_area;
where state_code in(26,27,29,36,37,39);
run;
proc sql;
create table sasout.crt_msa_area_2 as
select * from in.crt_msa_area
where state_code in(26,27,29,36,37,39);
quit;
data sasout.crt_msa_area_2;
set in.crt_msa_area;
where county_code=28;
run;
proc sql;
create table sasout.crt_msa_area_2 as
select * from in.crt_msa_area
where county_code=28;
quit;
```

	Time (Minutes:Meconds)	
Code Type	CPU	Elapsed
Where 6 Values		
SQL	00:39.7	00:47.9
DATA STEP	00:39.6	00:52.1
Where 1 Value		
SQL	00:04.3	00:07.1
DATA STEP	00:04.3	00:07.0

READING DATA FROM DATA SETS

Reading from an existing SAS data set, DATA step code showed a slight advantage in all 3 of the cases tested above, when reading 1,000,000 observations without subsetting, and in the 2 subsetting clauses. From this, we conclude that for a simple read of an existing data set, DATA step code is slightly preferable. All of the code was the same as above except that the LIBREF with the DB2 engine was deleted, and a DD statement to an existing SAS data set used instead, with the following results:

	Time (minutes:seconds)	
Code Type	CPU	Elapsed

Simple Read		
SQL	00:06.8	00:22.6
DATA STEP	00:05.3	00:23.9
Where 6 values		
SQL	00:04.5	00:15.7
DATA STEP	00:04.1	00:16.9
Where 1 value		
SQL	00:01.8	00:13.6
DATA STEP	00:01.6	00:09.1

UNIQUE COMBINATIONS OF VARIABLES

After establishing first that there was no difference between SQL and DATA step code in reading a second table subset for 354318 observations, simple summary reporting of all 8280 unique combinations of two variables was tested. For SQL, this was a select distinct of the two variables.

```
proc sql;
create table temp2 as
select distinct fcex8577, fcex8578
from in.CUV_fcex01
where fcex8577 <=50;
quit;
```

For BASE code, a number of variations were tested. A DATA step with a BY statement selected the unique combinations, of course preceded by a PROC SORT, which was preceded by a DATA step initially reading the table from DB2. (See table entry for METHOD 1.)

```
data temp;
set in.CUV_fcex01(keep=fcex8577 fcex8578);
where fcex8577 <=50;
run;
proc sort;
by fcex8577 fcex8578;
run;
data temp2;
set temp;
by fcex8577 fcex8578;
if first.fcex8578;
run;
```

The SQL was dramatically more efficient, requiring less than a third of the time. Reading the PROC SORT directly from DB2 and skipping the first DATA step did not help. (See table entry for METHOD 2.)

```
proc sort data=in.CUV_fcex01
(where=(fcex8577 <=50)
keep=fcex8577 fcex8578) out=temp;
by fcex8577 fcex8578;
where fcex8577 <=50;
run;
data temp2;
set temp;
by fcex8577 fcex8578;
if first.fcex8578;
run;
```

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:08.4	00:09.3
METHOD 1		
DATA STEP	00:30.0	00:33.5
SORT	00:02.7	00:05.3
DATA STEP	00:01.5	00:02.1
TOTAL	00:34.2	00:40.9
METHOD 2		
SORT	00:33.6	00:36.5
DATA STEP	00:01.5	00:02.2
TOTAL	00:35:1	00:38:8

But, surprisingly for those of us less active on the database side, when reading from a table with a BY statement, no prior PROC SORT is required. However, this one DATA step, comparable to the single SQL step in coding complexity, though marginally superior to the other DATA step methods, still was outperformed by the SQL code above by a factor of four.

```
libname in db2 ssid=dsn authid=fdrp;
data temp2;
set in.CUV_fcex01(keep=fcex8577 fcex8578);
by fcex8577 fcex8578;
where fcex8577 <=50;
if first.fcex8578;
run;
```

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:08.4	00:09.3
DATA STEP	00:34.0	00:37.3

Since the tab values (not the frequency values) of a PROC FREQ on the DB2 table produce the same summary of unique combinations, this method was also tested. Interestingly, though the PROC FREQ was marginally superior to the DATA step, it still lost to the SQL by a factor of three. Where a simple read with a select distinct is sufficient, SQL should be the code of choice.

```
proc freq data=in.CUV_fcex01
(keep=fcex8577 fcex8578) noprint;
tables fcex8577*fcex8578/out=temp2
(keep=fcex8577 fcex8578);
where fcex8577 <=50;
run;
```

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:08.4	00:09.3
FREQ	00:30.6	00:32.4

As an aside it should be noted that, when sorting directly from DB2, if the subsetting is done with a where statement rather than as an option on the input table, the penalty is huge, multiplying processing time by a factor of three.

When selecting unique combinations from an existing data set, the SQL code and BASE (SORT, then DATA step) code performed similarly, but the PROC FREQ took significantly less time. This points in the direction of favoring SQL when working with DB2 data, but identifying the right PROC for existing datasets.

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:05.3	00:08:1
SORT	00:03.6	00:06.4
DATA STEP	00:01.5	00:02.2
TOTAL	00:05.1	00:08.6
FREQ	00:03.3	00:04.4

SUMMARIZING A VARIABLE

For summing a variable over unique combinations of two other variables, SQL code again outperformed BASE code by a factor of 3 when reading from DB2. Running a PROC MEANS marginally outperformed DATA step code. But when reading from an existing dataset, SQL code was only slightly more efficient than a PROC SORT and DATA step, and both were outperformed by the PROC MEANS by a factor of 2. So SQL code should be the code of choice when reading from DB2 and performing a summing operation at the same time, but a PROC MEANS is the gold standard for summing from an existing data set.

```
/* SQL */
proc sql;
create table temp2 as
select distinct fcex8577, fcex8578,
sum(fcex8600) as sum8600
from in.CUV_fcex01
where fcex8577 <=10
group by fcex8577, fcex8578;
quit;

/* DATA STEP */
proc sort data=in.CUV_fcex01;
by fcex8577 fcex8578;
run;
data temp;
set in.CUV_fcex01(keep=fcex8577 fcex8578
fcex8600);
by fcex8577 fcex8578;
where fcex8577 <=10;
retain sum8600;
if first.fcex8578 then sum8600=0;
sum8600=sum8600+fcex8600;
if last.fcex8578 then do;
output temp;
end;
drop sum8600;
run;

/* PROC MEANS */
proc means data=in.CUV_fcex01 noprint;
class fcex8577 fcex8578;
var fcex8600;
output out=temp2(where=(_type_=3))
sum=sum8600;
where fcex8577 <=10;
run;
```

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Summarize a Variable		
FROM DB2		
SQL	00:06.9	00:08.9
DATA STEP	00:29.6	00:35.8
MEANS	00:27.0	00:31.6
FROM DATASET		
SQL	00:04.5	00:08.8
BASE CODE		
SORT	00:03.5	00:07.0
DATA STEP	00:01.8	00:02.8
TOTAL	00:05.3	00:09.8
MEANS	00:02.5	00:03.5

RESULTS ON NT

The above series of tests was also run on a Windows NT installation of SAS Version 8.1, reading the same DB2 tables. The only change was in the LIBNAME statement, now using the ACCESS to ODBC engine to access a driver for the DB2 server:

```
libname in odbc complete="dsn=M1DB2P;
Uid=xxxxx;pwd=xxxxx" schema=fdrp;
```

While the magnitude of the differences varied, in all cases, the comparisons between coding techniques paralleled those on MVS.

Reading from DB2

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Simple Read		
SQL	00:35.4	01:04.8
DATA STEP	00:33.6	01:06.9
Where 6 values		
SQL	00:13.3	00:32.7
DATA STEP	00:12.7	00:34.1
Where 1 value		
SQL	00:00.0	00:07.0
DATA STEP	00:00.0	00:07.1

Reading from Existing Data Set

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Simple Read		
SQL	00:03.3	00:17.9
DATA STEP	00:02.5	00:17.4
Where 6 values		
SQL	00:02.5	00:13.4
DATA STEP	00:02.5	00:13.5
Where 1 value		
SQL	00:01.5	00:10.8
DATA STEP	00:01.6	00:10.8

Report Unique Combinations from DB2

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:00.3	00:11.1
METHOD 1		
DATA STEP	00:10.0	00:26.2
SORT	00:01.9	00:08.9
DATA STEP	00:00.1	00:00.2
TOTAL	00:12.0	00:35.3
METHOD 2		
SORT	00:09.7	00:31.8
DATA STEP	00:00.4	00:00.4
TOTAL	00:10.1	00:32.2
FREQ	00:09.5	00:24.3

Report Unique Combinations from Existing Data Set

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Report Unique Combinations		
SQL	00:04.6	00:26.8
SORT	00:03.8	00:27.3
DATA STEP	00:00.3	00:00.6
TOTAL	00:04.1	00:27.9
FREQ	00:03.2	00:23.1

Summarizing a Variable

	Time (minutes:seconds)	
Code Type	CPU	Elapsed
Summarize a Variable		
FROM DB2		
SQL	00:00.2	00:08.3
DATA STEP	00:07.4	00:25.7
MEANS	00:07.6	00:25.8
FROM DATASET		
SQL	00:02.7	00:06.3
BASE CODE		
SORT	00:01.5	00:05.0
DATA STEP	00:00.8	00:02.0
TOTAL	00:02.3	00:07.0
MEANS	00:00.4	00:02.0

CONCLUSION

Comparisons between SQL and DATA step code may vary based on platform, operating systems, DB2 tuning, and other factors. Small differences in the results desired can produce much larger differences in the needed code, with corresponding larger differences in comparative efficiency. SQL was similar or superior in the environment tested for reading and simple summarizing from DB2. The PROC's tested were superior for simple summarizing from existing data sets. These conclusions are best seen not as an ending point but as an approach to benchmarking for an actual application. They may best serve as a quide to what to test--when limits on testing demand efficiency in benchmarking as well as in coding.

TRADEMARK INFORMATION

CONTACT INFORMATION

Steven Feder
Federal Reserve Board, Mail Stop 157
Washington, D.C. 20551
202-452-3144
email: steven.h.feder@frb.gov