Performance Considerations for the Database Engines
Howard Plemmons Jr., SAS Institute Inc., Cary, NC

ABSTRACT

This paper identifies some of the issues to consider when accessing your Data Base Management System (DBMS) through the SAS Version 6 database engines. Both SAS and DBMS performance issues are identified throughout the paper. The focus of this paper is on the current Version 6.06 engines: SYSTEM 2000, DB2, SQL/DS, RdbVMS, and ORACLE.

INTRODUCTION

The SAS Version 6 database engines provide transparent access to data stored in a user DBMS. With transparent access comes a performance cost, which can be measured in SAS and DBMS processing time. This paper discusses DBMS Indexing, DBMS locking, joining DBMS data, SAS WHERE statement processing, multiple access, SAS data set options along with suggestions on how to tune the database engines. The information provided will help you use the SAS Version 6 database engines more efficiently.

The Data

The following SAS DATA step generates test data for the DBMS examples in this paper:

```sas
data Ai;
  drop 1 It;
  It. date0;
  do i = 1 to 10000;
    ampno = 1 + 3000;
    un = 24590000 + 1;
    agl = 18 + modi, 9);
    if age > 18 and age <= 21 then degree = 6;
    else if age > 21 and age <= 25 then degree = 1;
    else degree = 3;
    salary = age * 1000 * (degree * 500);
    startl = k - (1 age - 18 ) * 365 ;
    comment1 = 'Employee of XYZ company in good standing';
    c1 = 'See personal file for additional comments';
    c2 = 'This is a comment to make the data take up more space';
    end;
  run;
```

The following index was created after the databases were loaded:

Index on value: EMPNO, order (ASCENDING)

The performance statistics shown in the examples are percentages. The larger time is considered 100 percent and the other a percentage of the larger. The data size of 10000 rows is relatively small and your performance percentage may be larger or smaller depending on your database size, DBMS tuning and overall system performance.

Version 6 Database Engines

What is a Version 6 database engine? Figure 1 shows the relationship between the SAS System and your DBMS via the Version 6 database engines.

1. The SAS System evaluates the PRINT procedure request and notes the type of the DATA parameter. In this case X,Y is a view descriptor.
2. The SAS engine supervisor examines the engine identifier in the view and loads the appropriate Version 6 database engine.
3. The database engine processes the view descriptor and prepares to operate on PROC PRINT requests.
4. PROC PRINT makes successive calls to retrieve records.
5. The database engine processes the call from PROC PRINT, retrieves the next qualifying record from the database and returns it to PROC PRINT.

The view descriptor can contain DBMS selection or subsetting criteria, or a SAS WHERE statement can be used to subset the data. This information determines the number of records returned to the engine from the database and returns it to PROC PRINT for processing.

INDEXING

Database indexing provides you with the potential for faster DBMS data access by shortening the time required to locate records in the database. You might think that this is the best solution to accessing data within the DBMS; however, indexes, at time, can be a burden on performance. Consider the following potential drawbacks when using indexes:

- Is there a lot of activity against the database? If so, the index or indexes on a database may have to be updated for every database modification. The overhead of index maintenance could offset the performance gain through index access depending on database activity.
**Version 6 database engines.**

Even when you consider the cost of indexing, it is still an essential times on the SAS statements:

The two views contain a selection criteria for a specific record which needs index processing then it might be advantageous to create and then delete the index after processing is complete. This could save index maintenance time on a database with a lot of activity during the month.

Even when you consider the cost of indexing, it is still an essential tool with the potential for significant performance increases. Table 1 below shows performance gains using DBMS indexes through the Version 6 database engines.

**Table 1** was generated by processing and comparing execution times on the SAS statements:

```sas
PROC PRINT DATA = SASUSER.EMPLIVA; RUN;
PROC PRINT DATA = SASUSER.EMPLIVA; RUN;
```

The two views contain a selection criteria for a specific record which resides near the end of the database, specifically WHERE EMPNO = 11227. The view EMPLIVA processes a database that has an index on EMPNO and the view EMPLIVA processes a database without an index. The data is a 10000 record database created using the DATA step defined in The Data earlier in this paper. 

The percentages shown in the table below were calculated using the following:

\[ \text{percent improvement} = \left(1 - \left( \frac{a}{b} \right) \right) \times 100; \text{ where } a \leq b \]

The second part of Table 1 was generated by processing and comparing execution times on the SAS statements:

```sas
PROC PRINT DATA = SASUSER.EMPLIVA; WHERE EMPNO = 11227; RUN;
PROC PRINT DATA = SASUSER.EMPLIVA; WHERE EMPNO = 11227; RUN;
```

**Table 1**

<table>
<thead>
<tr>
<th>DBMS</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SQL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Access (where in view)</td>
<td>61</td>
<td>22</td>
<td>42</td>
<td>23</td>
<td>82</td>
</tr>
<tr>
<td>Index Access (SAS WHERE statement)</td>
<td>9</td>
<td>22</td>
<td>42</td>
<td>12</td>
<td>82</td>
</tr>
</tbody>
</table>

Note: index access displayed as a percentage improvement over non-index access.

The performance difference between the two tables is the result of obtaining the number of observations in the database. With the relational engines this is done by issuing a SQL "select count statement.

With the SYSTEM 2000 engine the observation count is set to the largest number of records ever stored in the database.

To improve performance when using database indexing, consider putting the selection criteria in the SAS/ACCESS view descriptor. The selection criteria will then be used in the SQL select count statement for DB2, SQL/DS and ORACLE.

### LOCKING

Locking is used by your DBMS to ensure data integrity. Simply stated, locking would permit only one person updating the data at a time. For example, if you were changing the telephone number for an employee and at the same time someone else wanted to change the address, the second person would be denied access until you completed your change.

The SAS Version 6 database engines abide by all locking rules for your DBMS; however, several issues arise when accessing the same database record with SAS software. This is best illustrated by reviewing DBMS locking elements along with the SAS full-screen update procedure example.

**Table 2**

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE/TABLE/DATABASE</td>
<td>ROW/TABLE/ROW</td>
</tr>
<tr>
<td>ROW/SPACE/DATABASE</td>
<td>ROW/PAGE/TABLE/DBSPACE</td>
</tr>
</tbody>
</table>

**LOCKING**

Locking is used by your DBMS to ensure data integrity. Simply stated, locking would permit only one person updating the data at a time. For example, if you were changing the telephone number for an employee and at the same time someone else wanted to change the address, the second person would be denied access until you completed your change.

The SAS Version 6 database engines abide by all locking rules for your DBMS; however, several issues arise when accessing the same database record with SAS software. This is best illustrated by reviewing DBMS locking elements along with the SAS full-screen update procedure example.

**Table 2**

<table>
<thead>
<tr>
<th>Locking</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SQL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE/TABLE/DBSPACE</td>
<td>ROW/TABLE/ROW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LOCKING**

Locking is used by your DBMS to ensure data integrity. Simply stated, locking would permit only one person updating the data at a time. For example, if you were changing the telephone number for an employee and at the same time someone else wanted to change the address, the second person would be denied access until you completed your change.

The SAS Version 6 database engines abide by all locking rules for your DBMS; however, several issues arise when accessing the same database record with SAS software. This is best illustrated by reviewing DBMS locking elements along with the SAS full-screen update procedure example.

**Table 2**

<table>
<thead>
<tr>
<th>Locking</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SQL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE/TABLE/DBSPACE</td>
<td>ROW/TABLE/ROW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LOCKING**

Locking is used by your DBMS to ensure data integrity. Simply stated, locking would permit only one person updating the data at a time. For example, if you were changing the telephone number for an employee and at the same time someone else wanted to change the address, the second person would be denied access until you completed your change.

The SAS Version 6 database engines abide by all locking rules for your DBMS; however, several issues arise when accessing the same database record with SAS software. This is best illustrated by reviewing DBMS locking elements along with the SAS full-screen update procedure example.

**Table 2**

<table>
<thead>
<tr>
<th>Locking</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SQL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE/TABLE/DBSPACE</td>
<td>ROW/TABLE/ROW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td>ROW/PAGE/TABLE/DBSPACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Your ability as a SAS user to control the locking environment within your DBMS may be limited.

Several performance issues need to be considered when using the full screen update procedure example that follows.

**An Example Using the FSEDIT Procedure**

PROC FSEDIT permits you to add or modify individual rows in your DBMS. Using the FSEDIT procedure to access your database is accomplished by reading the view descriptor and then the database. Using the FSEDIT procedure with the locking scheme shown in Table 3 raises several performance issues. Table 3 and the associated text define the outcome of two users attempting to access the same database record using PROC FSEDIT and details the locking conditions you may experience when attempting this type of access:

<table>
<thead>
<tr>
<th>DBMS</th>
<th>User 1</th>
<th>User 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM 2000</td>
<td>Locked</td>
<td>A</td>
</tr>
<tr>
<td>DB2</td>
<td>Locked</td>
<td>B</td>
</tr>
<tr>
<td>SQL/DS</td>
<td>Locked</td>
<td>C</td>
</tr>
<tr>
<td>ORACLE</td>
<td>Locked</td>
<td>A</td>
</tr>
<tr>
<td>Rdb/VMS</td>
<td>Locked</td>
<td>A</td>
</tr>
</tbody>
</table>

A. User 2 cannot update the data that User 1 has locked. You can update any other nonlocked row in a table.

B. User 2 cannot update the data that User 1 has locked. In this case the locking is more severe. User 2 cannot update any other row in that page of data until User 1 scrolls off the page or ends his FSEDIT session. If the lock is on the table then User 2 cannot gain access until User 1 ends the FSEDIT session.

C. User 2 cannot update the data that User 1 has locked. The current lock environment in the table User 1 is accessing may determine availability of data to User 2. If the lock is on the row, then User 2 can access other rows in the table. If update locks are held on the page, then User 2 cannot edit any rows on the page. If the lock is held at the table then User 2 cannot edit any rows in the table.

The locking scheme employed by your DBMS could limit use of the FSEDIT procedure for B above. Also if you use FSEDIT on a table you may lock out other non-SAS applications since FSEDIT requests an update intent lock.

Considering locking conditions B and C, the FSEDIT procedure used as a mult-user tool on the same table causes locking contention; however, as a single access tool FSEDIT provides full screen edit capability of data directly in your database. This would give you the power to manipulate or add to rows in your DBMS. If you execute the FSBROWSE procedure, or the full screen FS VIEW procedure in browse mode, you avoid the update locks on your database and still have the ability to examine your data in full screen mode.

**JOINING**

The ability to combine data from several different sources provides you with a very powerful tool. Along with this power comes several performance considerations identified in an Example Using the SQL and PRINT Procedure.

**An Example Using the SQL and PRINT Procedure**

You have two tables in your DBMS that need to be combined in order to produce a report. The first table is the 10000 rows defined in the Data section earlier in this paper. The second table contains a degree number and the name of the degree. You want to join the two tables and print the degree name along with other information from the first table. You have created view descriptors and decide to use PROC SQL and PROC PRINT to do the processing. The source code below shows the SAS statements you would enter:

```
proc sql;
  create view work.myrept as
  select empno, degrname, sex, salary1
  from dbms.tabl, dbms.tab2
  where empno > 8500 and empno < 8500
  and tab1.degree1 = tab2.degree;
proc print data=work.myrept; run;
```

**PROC SQL and PRINT processing:**

1. PROC SQL creates a view in your SAS data library that contains information about the database views that are to be joined. This view also contains information about what is to be returned and the join criteria or WHERE clause.

2. PROC PRINT processes the SQL view, which in turn processes the DBMS using the database engine and views TAB1 and TAB2. The rows PROC PRINT requests are returned by the database engine and the join takes place in the SQL engine.

**Performance considerations:**

1. As a general rule the closer you push the processing to the database the better the performance. After you create the PROC SQL view in the DBMS, then access and view descriptors would be built from the database view. The SAS/ACCESS view descriptor is then used in the subsequent PROC PRINT statement:

```
proc print data=work.dbmsview; run;
```

In this case the database engine is still accessing the data; but since the view and where criteria are contained in the DBMS as a DBMS view, the joining takes place there. Table 4 shows the performance gain of allowing the DBMS to do the join as compared to the PROC SQL statements above. As in the indexing section, the figures shown in Table 4 are calculated and shown as a percentage improvement.


Table 4

<table>
<thead>
<tr>
<th>DBMS</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SOL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW</td>
<td>39</td>
<td>15</td>
<td>14</td>
<td>27</td>
<td>20</td>
</tr>
</tbody>
</table>

Note: The table above shows the percentage improvement of joining two tables within the DBMS over PROC SQL. Joining the two tables. The 10000 record table has an index established on the EMPNO column.

2. The situation of joining data in which a Cartesian product is the result should be avoided. For example, if you were joining two 10000 record tables together and the Cartesian product is the result, you would retrieve 100 million records (10000 x 10000). You can avoid this by using joins where the criteria contain an equal condition. Using the PROC SQL statements earlier in this section avoids creating the Cartesian product by matching on degree.

WHERE PROCESSING

The database selection criteria and the SAS WHERE statement are very powerful data manipulation tools. Their effective use can significantly increase performance when using the Version 6 database engines. The engine performance can be increased by effectively using the SAS WHERE statement in combination with the database WHERE clause. For example you wish to create a view that accesses employee ID numbers less than 3500. You first create an access descriptor, if one does not exist for the database, then a view descriptor. While creating the view you enter the database selection criteria, EMPNO < 3500, using the Selection Criteria Entry window in the ACCESS procedure. The selection criteria you enter in the view will always be used when accessing the database. The importance of combining the DBMS selection criteria and the SAS WHERE statement is shown in the following examples.

You create a view descriptor MINE.EMPLYOEE with a selection criteria of EMPNO < 3500 and now want to access the data using PROC PRINT. You enter:

```
PROC PRINT DATA = MINE.EMPLOYEE; run;
```

The database records that meet the selection criteria are returned to PROC PRINT, in this case all employees with an employee number less than 3500.

To incorporate a SAS WHERE statement into the print request you enter:

```
PROC PRINT DATA = MINE.EMPLOYEE;
  WHERE EMPNO > 3500;
run;
```

The SAS WHERE statement is evaluated and passed down if the Version 6 database engine can process it. The SAS WHERE statement is joined to the WHERE clause in the view with the AND operator. The database records with employee numbers between 3250 and 3500 will be returned to PROC PRINT.

If the Version 6 engine cannot process part or all of the SAS WHERE statement, then the records that the engine returns would be filtered again by the SAS engine supervisor. In the example code earlier in this section, if the engine could not process the SAS WHERE statement, all records with EMPNO less than 3500 are returned. The engine supervisor would then screen the records for EMPNO greater than 3250 before returning the record to PROC PRINT.

Consider the following when formulating database selection criteria and the SAS WHERE statement:

- Selection criteria in the SAS/ACCESS view descriptor is always used in any database access using the view. This tool can be used to subsect the quantity of data retrieved from large databases and to limit the access to data in your databases for performance or security purposes. Using the view above, you will never be able to examine employees with EMPNO greater than or equal to 3500.

- The SAS WHERE statement augments the existing database selection criteria. It is passed to the Version 6 database engine for processing whenever possible. You can increase performance when using the SAS WHERE statement inside the full screen procedures FSEDIT and FSVIEW. The SAS WHERE statement can be used to quickly access the data you want to view or modify. The SAS WHERE statement entered while executing FSEDIT or FSVIEW would be joined with an AND to any selection criteria in the view descriptor.

EXTRACT VERSUS MULTIPLE ACCESS

In Version 5 you were required to extract data from your DBMS into a SAS data set before you could use the SAS System. In Version 6 you can process directly against your database; however, the extraction process may still be a performance alternative. The following section identifies some of the pros and cons in using the extract than process method.

Extracting Data Before Processing

Using the example in An Example Using the SQL and Print Procedures, you execute PROC PRINT several times in order to perfect the output. Each time it executes it requests records by calls through the database engine. The database is accessed and the records returned back through the engine to PROC PRINT. Table 5 shows how many executions of PROC PRINT using the database engine would be necessary to justify the extract and PROC PRINT process.

Table 5

<table>
<thead>
<tr>
<th>DBMS</th>
<th>DB2</th>
<th>ORACLE</th>
<th>Rdb/VMS</th>
<th>SOL/DS</th>
<th>SYSTEM 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTRACT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>PRINT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

The numbers in the table show when it is more efficient in terms of CPU time to extract the data into the SAS System and invoke PROC PRINT in the extracted SAS data set.

Consider the following before deciding to extract the data into the SAS System:

- The data that is extracted may not be current. If the database or SAS data set is modified, the copy is not identical to the database. This may lead to erroneous results when processing or reporting on the data in the SAS data set.
The additional disk space required to maintain an additional copy of your data may be prohibitive.

Database security may be compromised by taking data out of the database and storing it in SAS data sets. The extract action itself may generate additional security considerations.

SAS DATA SET OPTIONS

SAS data set options can be used on the view descriptor to pass information to the engine supervisor and the engine. The following examples identify some of the options to consider when using the Version 6 database engines:

Example 1

The following print statement would eliminate the columns SSN and SALARY from the records returned to PROC PRINT:

```sas
proc print data=sasuser.empliv (drop ssn salary);
run;
```

The DROP option does not provide you with a noticeable performance increase. What it does provide is the ability to use a single view descriptor to produce a variety of reports.

Example 2

The following PRINT statement indicates the last observation that is to be returned to PROC PRINT:

```sas
proc print data=sasuser.empliv (obs = 10);
run;
```

The value of the OBS= option does not restrict the internal processing that the database undergoes to retrieve the observation. What it does restrict is the number of calls PROC PRINT makes for database records. The OBS= option cannot be used with a SAS WHERE statement or FIRSTOBS= option. The view descriptor selection criteria can be used with the OBS= option, but will retrieve a maximum of 10 observations based on the selection criteria.

Example 3

The following DATA step extracts from the database into a SAS data set using a view descriptor in the SAS SET statement:

```sas
data work.a;
set sasuser.empliv (where = (empno < 3010));
run;
```

The SAS WHERE statement is appended to the database selection criteria and the SAS data set WORK.A is created from the qualifying records.

The SAS data set options discussed in this section are a flexible tool that you can use to improve performance by passing additional requirements to the SAS engine supervisor and the Version 6 database engine.

SUMMARY

The Version 6 database engines provide you with transparent access to data stored in your DBMS. The many different features and options you can use with these engines can create performance problems if not used properly. This paper presents some of the issues that must be considered when using the SAS System to access your database. Your access to tools and information that directly manipulate the database may be restricted; therefore, consider the following items when using SAS software to access your DBMS:

- Ensure that the way you want to access data is known to your DBA. They may be able to create indexes that will decrease the resources required to access your data.

- Accessing the DBMS through the SAS System creates locks within the database environment. Remember to terminate SAS full screen procedures that obtain locks on your database when you are finished. If you do not, then you may impact others within and outside the SAS environment who want to access the database data.

- When joining database tables together or database tables to SAS data sets remember to use equal conditions in the join criteria. This provides a connection between the two tables to be joined (for example value x in table 1 = value y in table 2) and eliminates the possibility of joins creating Cartesian products.

- Enter the selection criteria in the view descriptor whenever possible. Also remember to use the SAS WHERE statement to further subset the data you want to retrieve. These subsetting elements can save a considerable amount of processing time when accessing database data with SAS software.

- Consider the extract and process procedure when using SAS software to access your database. This facility permits you to extract data into SAS data sets for processing. This would alleviate the DBMS data access locking and performance concerns but may cause additional problems as identified in Extract Versus Multiple Access.

ADDITIONAL INFORMATION

The CPUs, operating systems and database release information currently in use at SAS Institute are as follows.

MVS: 3090 model 600 running ESA 3.1.0e
DB2 release: 1.3
SYSTEM 2000 release: 11.6

CMS: 3090 model 600 running VM/XA™ 5.5
SQL/DS release: 2.2

VAX: VAX/VMS™ 6000 series model 450
VMS operating system running V5.3-1
ORACLE release: 5.1.22
Rdb/VMS release: V3.0-0
VAX SQL release: V2.0-001

The percentages shown in the tables were calculated from data gathered by executing the sample SAS statements in the paper. It was assumed that the load on the processing environment was consistent from one SAS statement execution to the next. The testing was performed during normal business hours, except for SQL/DS, which was run during off hours.

REFERENCES

Additional information about the topics presented in this paper can be found in the following SAS publications:

• SAS/ACCESS Interface to DB2: Usage and Reference, Version 6, First Edition

• SAS/ACCESS Interface to SQL/DS: Usage and Reference, Version 6, First Edition

• SAS/ACCESS Interface to ORACLE: Usage and Reference, Version 6, First Edition

• SAS/ACCESS Interface to Rdb/VMS: Usage and Reference, Version 6, First Edition


SAS, SAS/ACCESS and SYSTEM 2000 are registered trademarks of SAS Institute Inc., Cary, NC, USA.

DB2, SQL/DS, SQL, and VM/XA are trademarks of International Business Machines Corporation. Rdb/VMS, VMS, and VAX are trademarks of Digital Equipment Corporation. ORACLE is a registered trademark of Oracle Corporation.