

Paper 159-28

Performance Tuning SAS/ACCESS for DB2

Scott Fadden, IBM , Portland, OR

INTRODUCTION

SAS solutions are renowned for their capabilities to access, transform and analyze data. IBM's DB2 database software is the worldwide market share leader in the database industry (Gartner Group, May 2001). Gartner Group recently estimated that by 2004 the average enterprise would be able to successfully manage 100 terabytes of data, a task for which DB2 is ideally suited. When using SAS with DB2 in today's complex business environments the goal is to leverage the capabilities of both products to achieve the best result

The intent of this paper is to explore the impacts of different SAS and DB2 configurations by outlining different methods of accessing your DB2 data with a focus on performance. We look at the performance trade-offs of choosing different access methods, SAS System 9 application parameters and DB2 8.1 configuration options. We also highlight new features in SAS System 9 and DB2 version 8.1.

Throughout this paper we use examples from a test system to help us measure the performance impact of SAS/ACCESS tuning options. The test environment consists of SAS System 9 and DB2 version 8.1 running on a single four processor Unix system with 4GB of memory and 40 fibre channel disks.

The performance results in this paper are provided as a demonstration of the impact of various configuration options. They are not intended as a method of comparing dissimilar configurations. Different configurations may yield different results (i.e., your mileage may vary).

WHAT IS SAS/ACCESS FOR DB2?

SAS and DB2 communicate via the SAS/ACCESS for DB2 product. SAS/ACCESS for DB2 is a member of a large family of data access products offered by SAS Institute. SAS/ACCESS allows the power and flexibility of SAS software to be used for analyzing and presenting data directly from DB2. Your DB2 data appears native to the SAS software so you can use SAS features and functionality to perform extracts of information without having to learn Structured Query Language (SQL).

SAS/ACCESS for DB2 translates read and write requests from SAS into the appropriate calls for DB2. The result of these calls is to surface data in one of the following forms: as logical views to the native data source or extracts of native data into SAS data set form.

SAS/ACCESS engine functionality ranges from "automatic" behind the scenes operations with minimal database knowledge to "completely flexible" allowing a DBA to finely tune the data access components of a SAS application. The method you use depends on your infrastructure, database expertise and operational goal. In this paper we examine the translation process from a SAS application to the corresponding SQL required to exchange information with DB2.

In the latest versions of SAS/ACCESS for DB2 (SAS System 9) and DB2 UDB (version 8.1) the joint processing capabilities have been greatly expanded. These new capabilities include: SAS/ACCESS threaded reads and expanded load support; DB2 CLI LOAD and improved multi-row fetch performance.

WHAT IS REQUIRED FOR SAS TO ACCESS DATA FROM MY DB2 DATABASE?

There are many configuration options that provide flexibility in architecting a unique SAS/DB2 solution. SAS and DB2 may run on the same server or on different servers. Multiple SAS servers can access a single database server or one SAS server can access multiple database servers. SAS and DB2 run on many different platforms. These platforms need not be the same for these components to interact. For example, your DB2 data warehouse could be running on AIX and your SAS server on Windows. For this paper we concentrate on our test environment configuration with SAS and DB2 running on the same server.

Required software:

- DB2 version 7.1 or higher
- Base SAS version 9.0
- SAS/Access for DB2

No modifications to your DB2 database are necessary for the addition of SAS/ACCESS. SAS/ACCESS communicates with the database server using the DB2 Call Level Interface (CLI) API included in the DB2 client software. Other SAS application packages that run atop Base SAS can be added as needed but are not required for running basic SAS applications. However your environment is designed, there are a few basic software components that are required: DB2 version 7.1 or higher, Base SAS System 9 and SAS/ACCESS for DB2.

DATA PROCESSING MODELS

SAS supports many data processing models, which allows you the flexibility to design a solution that meets your needs. The performance of each solution can vary greatly. Though every environment is unique there are some factors to consider when developing your data processing model:

HOW LONG DOES IT TAKE TO GENERATE THE RESULT?

It is not uncommon to take four, eight, ten hours or more in a large data warehouse environment to generate a result that is used for multiple SAS operations. In this case it is best to keep the result set around so you do not need to re-query the database each time you want to run another analysis.

WHAT IS THE SIZE OF THE RESULT SET?

If the result set is large consider storing it in a data mart. In many situations it is a good idea to move this data to a system separate from the database server.

HOW MANY TIMES ARE THE RESULTS ACCESSED?

If the result set is general enough and you need to run multiple SAS procedures against the same data then you will want to maintain a copy of the result set. Even if it only take 1 hour, for example, to generate the result if 20 people need the data you could save 19 hours of processing.

HOW MANY USERS WILL BE USING SAS AGAINST THE RESULT SET?

If your data needs to be accessed by multiple users it would be best to store the results in an offline data mart.

HOW FAST IS THE COMMUNICATION LINK BETWEEN SAS AND DB2?

This will determine whether you store the results locally or remotely. For example, if the database server is in a different city than the SAS server it may be a good idea to set up a local data mart so SAS requests do not need to be passed over the WAN.

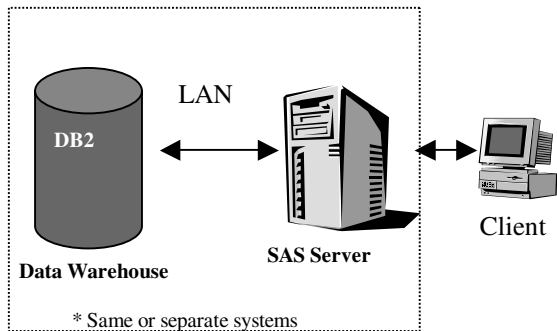


Fig.1 SAS Server accessing the database directly

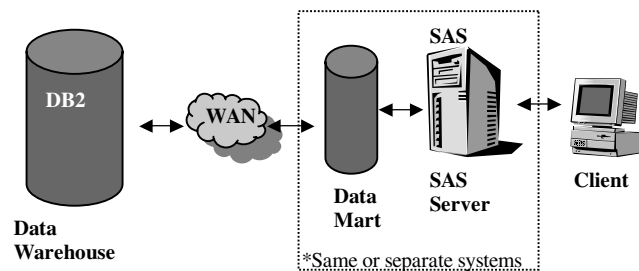


Fig.2 SAS Server accessing a data Mart

USAGE SCENARIOS

The following example solutions illustrate the criteria above. These scenarios assume that the query that generates the result takes many hours to complete and is very resource intensive. In this case you may want to save a result set somewhere for reuse.

SCENARIO 1

You are the only one using the data. You only need to access the database once and you can reuse the result set to complete your analysis.

In this case, it would probably be best to create a local copy of the results and use that for your analysis.

SCENARIO 2

Ten to twenty SAS users need access to the data and query it multiple times using different procedures.

In this case it would be best to save the result of the initial query in a data mart so all the SAS users can access it.

SCENARIO 3

Your SAS users require input from many different data sources.

In many SAS environments data are collected by one or more DBA's, and sent to the SAS user for analysis. You can streamline this process by retrieving the data directly from the source. SAS allows you to execute procedures against multiple data sources in a single step. In this case SAS does all the data collection for you. You can also use IBM's federated database technology in DB2. Federated servers provide a single view to multiple data sources as if they all existed on a single DB2 server. This allows the DBA to provide a secure central data access point for the SAS users. Both ways simplify the collection of information from various sources.

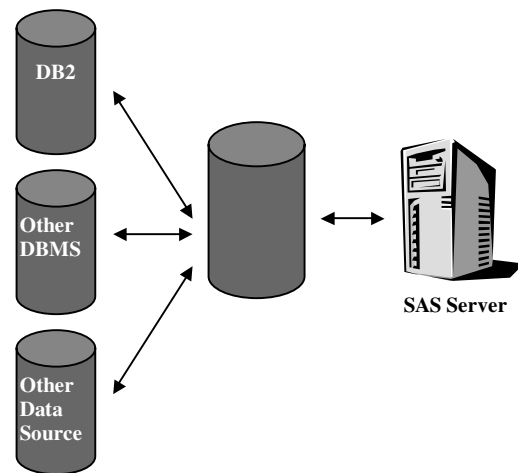


Fig. 3 Using a gateway to multiple data sources.

SCENARIO 4

You use SAS to analyze the status of your just-in-time inventory system.

In this case the data need to be up-to-date. Accessing live DB2 data is the answer. To enable this type of evaluation you may consider using DB2 Automatic Summary Tables (AST) or the new DB2 version 8.1 Multidimensional Clustering (MDC) feature to improve query performance.

ACCESSING DB2 DATA USING SAS

To begin, we need to understand how SAS processes data. We look at an example of how SAS processes data from a SAS data set and compare it to how SAS processes the same data stored in a DB2 database.

Using Base SAS most analysis procedures require that the input be from one or more preprocessed SAS data sets. Other procedures and DATA steps are designed to prepare the data for processing. For example, we take a look at the execution steps of a print procedure. To print census data and sort the output by state using a SAS data set you run `sort (proc sort)` then `print (proc print)`.

```
/*Step 1, Sort the SAS data set */
proc sort data=census.hrecs;
  by state;
```

```
run;

/* Step 2, Print the results */
proc print data=census.hrcs(keep=State
serialno );
    by state;
run;
```

Code Example: 1

To process this request in SAS you run `proc sort` first to order the data then run `proc print` to produce the report. Using SAS/ACCESS to read from DB2 as your data source can make executing these procedures more efficient. In this example, if your data source were DB2 you would not need to pre-sort the data for `proc print`. SAS/ACCESS automatically generates the SQL `order by` clause and the database orders the result. This is supported through the SAS/ACCESS translation to SQL engine.

DATABASE ACCESS FROM SAS

There are two different ways to connect to your DB2 database from SAS. You can define a connection using the libname engine or connect directly to the database.

When accessing the database using the libname engine SAS automatically translates the SAS application request to SQL. Translation to SQL means that SAS processes the SAS application code and generates the appropriate SQL to access the database. Explicit pass-through is a mechanism that allows you to pass unaltered SQL directly to the database server. Explicit SQL pass-through is useful for adding database only operations to your SAS application and is only accessible using the SQL procedure (`proc sql`).

Most SAS procedures and *DATA* steps use the SAS/ACCESS SQL translation engine. Code Example: 2 shows an example of SAS to SQL translation for the `print` procedure above. When this same procedure is executed using SAS/ACCESS against a DB2 library the code is translated into SQL for processing by DB2.

```
/*SQL Generated or Proc Print */
Select "state","serialno"
From hrcs
Order By State;
```

Code Example: 2

There are various reasons for using each type of access. We take a look a few examples of each.

WHEN WOULD YOU USE SAS/ACCESS TRANSLATION TO SQL?

- When you want to use SAS data access functionality (threaded reads, for example)
- When you are joining data from multiple data sources
- When the application needs to be portable to different data sources
- When the procedure requires it. (e.g., `proc freq`, `proc summary`)

WHEN WOULD YOU USE EXPLICIT SQL PASS-THROUGH?

- For non-SAS processing.
- When you want to use DB2 specific SQL
- In general, non-SAS processing executed from a SAS

application.

Non-SAS processing means that SAS does not need to manipulate the data and all the work is done in the DB2 database server. In this paper we focus on the SAS/ACCESS translation to SQL engine and how DB2 data are processed.

USING THE SAS/ACCESS LIBNAME ENGINE

SAS provides a standard data access mechanism called the libname engine. This interface is used by SAS to access data libraries (a data source). We look at the relational database library and how it interacts with DB2.

Performance tip:

It is always best to limit the amount of data transferred between DB2 and SAS. If you only need a few columns for your analysis list the columns you need from the source table. In the example above changing

```
set census.source1;
to
set census.source1 (keep=state
puma);
tells SAS to only retrieve the columns needed.
```

The SAS libname engine allows you to easily port applications to run against different data sources. You can modify a SAS statement that uses a SAS data set and run it against a table in your DB2 database simply by changing the libname definition. For example, here is a script that accesses a SAS data set named `mylib.data1` to generate frequency statistics.

```
/* Define the directory /sas/mydata
as a SAS library */
libname mylib data=/sas/mydata;

/* Run frequency statistics against
the data1 data set */
proc freq data=mylib.data1;
    where state='01';
    table state tenure yrbuilt yrmoved
    msapmsa;
run;
```

Code Example: 3

To run frequency statistics for the `data1` table in your DB2 database change the libname statement to access the database instead of the SAS dataset:

```
libname mylib data=DB2 user=me
using=password;
```

Code Example: 4

This is the only change necessary to run the `freq` procedure against the data in your DB2 database. This method is helpful because it simplifies your SAS applications by making your code more portable to different environments. In this example access to the database is handled by SAS/ACCESS. SAS/ACCESS generates the SQL required to retrieve the data from the database. For the statement above (Code Example: 3) SAS generates the SQL (Code Example:

5):

```
SELECT "STATE", "TENURE", "YRBUILT",
"YRMOVED", "MSAPMSA"
FROM data1
WHERE state = '01';
```

Code Example: 5

Using the libname engine is the easiest way to access data from your SAS application because SAS generates the appropriate SQL for you. To better understand SAS SQL translation let us take a look at how a SAS DATA step would be processed by SAS.

In this example the DATA step reads and filters the data from the source table `source1` and writes the results to the database table `results1`. The result table contains all the housing records for state id '01'.

```
data census.results1;
  set census.source1;
  where state = '01';
Run;
```

Code Example: 6

SAS begins by opening a connection to the database to retrieve metadata and create the `results1` table. That connection is then used to read all the rows for `state='01'` from `source1`. SAS opens a second connection to write the rows to the newly created `results1` table. If this were a multi-tier configuration the data would have to travel over the network twice. First, from the database server to the SAS server, then from the SAS server back to the database server. SAS processes the data this way to support data from multiple sources. For example, you could read data from one database server and write to the results to a different database. In this example, since we are using a single data source and SAS does not need to process the data, we can make this operation more efficient. To make this operation more efficient we use another method of data access called explicit SQL pass-through using the `sql` procedure. To make this statement explicit we added the "connect to" and the "execute() as db2" syntax.

```
proc sql;
  connect to db2 (database=census);

  execute(Create Table results1
  like source1) as db2;

  execute(Insert Into results1
  Select *
  From source1
  Where state = '01') by db2;

  disconnect from db2;
quit;
```

Code Example: 7

On the test system the original DATA step (see Code Example: 6) executed in 33 seconds, the `proc sql` version executed in 15 seconds. Changing to explicit processing improved the performance of the operation by 64%. In this case, since all the work can be done at the database, explicit SQL is the most efficient way to process the statement.

You may wonder, since we were using `proc sql`, why we used

explicit SQL instead of SQL translation (also called implicit SQL pass-through). We used explicit SQL because an implicit `proc sql` statement is processed using the same SQL translation engine as a DATA step. If we pass this same statement as implicit SQL, SAS breaks the statement into separate select and insert statements. In this example the performance gain we realized using explicit SQL resulted from all the processing being handled by the database. If we were to develop a general rule of thumb here, it would be something like: If SAS does not need to process it, let the database do the work. Explicit SQL is the best way to make sure this happens.

Performance Tip:

Let the database do as much work as possible if SAS does not need to process the data. Explicit SQL is usually the best way to make sure this happens.

When SAS is executing a procedure it is important to understand which operations are done in DB2 and which operations the SAS server is processing.

WHAT FUNCTIONS ARE PASSED TO DB2 FOR PROCESSING?

Most SAS procedures use SAS/ACCESS SQL translation so it is important to understand what functions are passed to the database for processing. To enable the most functions to be passed to the database set the `SQL_FUNCTIONS=all` libname option. SAS pushes the following functions down to DB2 for processing:

ABS	FLOOR	LOWCASE (LCASE)
ARCOS (ACOS)	LOG	UPCASE (UCASE)
ARSIN (ASIN)	LOG10	SUM
ATAN	SIGN	COUNT
CEILING	SIN	AVE
COS	SQRT	MIN
EXP	TAN	MAX

Applying these functions in the database can improve analysis performance. Each aggregate (vector) function (examples: AVE, SUM) that DB2 processes means fewer rows of data are passed to the SAS server. Processing non-aggregate (scalar) functions (ABS, UPCASE etc...) takes advantage of DB2 parallel processing.

LOADING AND CREATING DATA

SAS provides powerful extraction, transformation and load (ETL) capabilities. Therefore it is often used to load data into the database. SAS supports three methods of loading data into DB2: Import, Load and CLI LOAD. SAS accesses these load options through the bulk load interface.

If you have a procedure or DATA step that creates a DBMS table from flat file data, for example, the default load type is IMPORT. This option is best for small loads because it is easy to use and the user requires only insert and select privileges on the table. To enable bulk load using import you need to set the DATA step option `BULKLOAD=yes` (Code Example: 8).

If you need to load large amounts of data quickly you should use the LOAD or CLI LOAD bulk load options. If you are using DB2 v8.1 CLI LOAD is the recommended method of loading data.

To use the DB2 Load feature you need to add the `BL_REMOTE_FILE=<xxx>` DATA step option (Code Example: 9).

The `BL_REMOTE_FILE` option defines a directory for SAS to use as temporary file storage for the load operation. To process a load, SAS reads and processes the input data and writes it to a DB2 information exchange format (IXF) file. The IXF file is then loaded into the database using DB2 LOAD. Using the LOAD option requires the `BL_REMOTE_FILE` directory to have enough space to store the entire load dataset. It also requires the directory defined by `BL_REMOTE_FILE` be accessible to the DB2 server instance. This means it is on the same machine as DB2, NFS mounted or otherwise accessible as a file system. This can be an issue if you are loading large sets of data.

New in DB2 version 8.1 and SAS System 9 is support for DB2 CLI LOAD. CLI LOAD uses the same high performance LOAD interface but allows applications to send the data directly to the database without having to create a temporary load file. CLI LOAD saves processing time because it does not have to create a temporary file and eliminates the need for temporary file system space. CLI LOAD also allows data to be loaded from a remote system. To enable the CLI LOAD feature using SAS set the `BL_LOAD_MODE=CLILOAD` DATA step option instead of `BL_REMOTE_FILE` (Code Example: 10).

We ran performance comparisons between the different load options to give you an idea of the performance differences. This test executes a DATA step that loads 223,000 rows into a single database table.

```
/* Method: Import */
data HSET(BULKLOAD=YES);
  <...DATA step processing ...>
run;
```

Code Example: 8

```
/* Method: Load */
data HSET(BULKLOAD=YES
  BL_REMOTE_FILE="/tmp");
  <...DATA step processing ...>
run;
```

Code Example: 9

```
/* Method: CLI Load */
data HSET( BULKLOAD=YES
  BL_METHOD=CLILOAD );
  <...DATA step processing ...>
run;
```

Code Example: 10

Load Method	Time (seconds)
Import (Code Example: 8)	76.69
Load (Code Example: 9)	55.93
CLI LOAD (Code Example: 10)	49.04

As you can see there is a 36% performance gain over import by using CLI LOAD. All of these load options require that the table does not exist before the load.

CREATING TABLES

You can create database tables using SAS/ACCESS. SAS/ACCESS actually creates DB2 tables for you automatically like it does a SAS dataset. When SAS automatically creates a database table using the default options it uses three DB2 data types double, varchar and date. If you would like the table created with different data types use the `DBTYPE=` dataset option. In this example the `SERIALNO` and `PUMA` columns are set to DB2 data types when the table is created.

```
data census.results1
  (dbtype=(SERIALNO='bigint' PUMA='char(25)'));
  set census.source1;
  where state = '01';
Run;
```

Code Example: 11

To set any DB2 specific table creation options you should use the `CREATE_TABLE_OPTS libname` option. `CREATE_TABLE_OPTS` appends whatever you include to the end of the create table statement. Let's take a look at an example. The following code creates a table that contains all the rows from `source1` where the state is '01'.

```
data census.results1;
  set census.source1;
  where state = '01';
Run;
```

Code Example: 12

If this table is being created in a partitioned database you could specify the partitioning key by adding:

```
data census.results1
  (DBCREATE_TABLE_OPTS='PARTITIONING
  KEY(SERIALNO)');
  set census.source1;
  where state = '01';
Run;
```

Code Example: 13

To create the `results1` table SAS generates the create table syntax and adds the `DBCREATE_TABLE_OPTS` at the end of the statement.

```
CREATE TABLE RESULTS1
  (SERIALNO double, PUMA varchar(20), [column
  list...])
  PARTITIONING KEY(SERIALNO);
```

Code Example: 14

The table `results1` is created with the data partitioned across nodes by `serialno`.

RETRIEVING THE DATA INTO SAS

SAS/ACCESS provides different ways to retrieve data from your DB2 database. You can control access to partitioned tables, use multiple SAS threads to scan the database and utilize CLI multi-row fetch capabilities. SAS version 9 and DB2 version 8.1 have brought great improvements in the read performance of SAS applications.

The SAS threaded read is new to SAS System 9. Threaded read allows SAS to extract data from DB2 in parallel; this can be helpful with a large or partitioned database. In DB2 version 8.1 the impact of multi-row fetch has been improved, increasing read performance up to 45% over a single-row fetch.

WHAT ARE THE PERFORMANCE IMPACTS OF THESE OPERATIONS?

In this section we test three different SAS tuning parameters which can improve the speed of data transfer from DB2 to SAS: `READBUFF`, `DBSLICEPARM` and `DBSLICE`. `READBUFF`, `DBSLICEPARM` and `DBSLICE` correspond to the DB2 functions multi-row fetch, `mod()` and `nodenumber` respectively.

Functionality Match-up	
SAS Function	DB2 Function
<code>READBUFF</code>	Multi-Row Fetch
<code>DBSLICEPARM</code>	<code>mod()</code>
<code>DBSLICE</code>	<code>nodenumber</code>

To examine the performance differences between these options we ran frequency statistics against a database table and measured the execution time. To generate frequency information SAS needs to see all the rows in the table. Since the math is not complex this a good test of I/O performance between SAS and DB2.

The first test was run using the default read options: single-row fetch and non-threaded read:

```
libname census db2 db=census user=db2inst1
using=password;

proc freq data=census.hrecs_db
    table state tenure yrbuilt yrmoved
    msapmsa;
run;
```

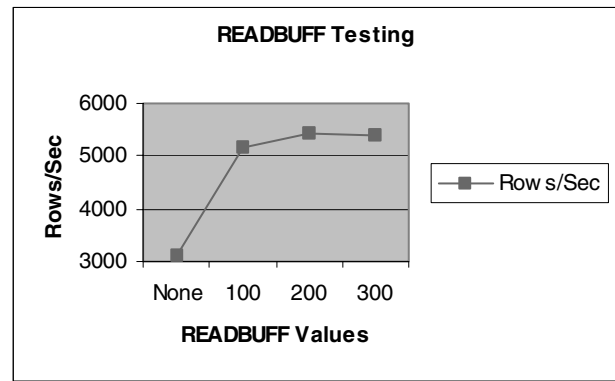
Code Example: 15

This test ran in 72.02 seconds. Using the default options SAS executes a single thread that reads one row at a time through the DB2 CLI interface. This is not efficient when you are returning large result sets. Transfer speed can be greatly improved by sending multiple rows in each request. DB2 supports this type of batch request in CLI using the multi-row fetch feature. SAS/ACCESS supports the DB2 multi-row fetch feature via the `libname READBUFF` option. We added `READBUFF=100` to the `libname` statement and ran the test again.

```
libname census db2 db=census user=db2inst1
using=password READBUFF=100;
```

Code Example: 16

This time the frequency procedure took only 43.33 seconds to process. That is a 40% performance improvement over a single row fetch. We tested this procedure with some other values of `READBUFF` and found the optimal value is somewhere between 200 and 250 which allowed the query to run in 40.97 seconds. So from here on we leave `READBUFF` at 200 and test the new multi-threaded read options.



SAS version 9 introduces a new data retrieval performance option called threaded read. The threaded read option works on the divide and conquer theory. It breaks up a single select statement into multiple statements allowing parallel fetches of the data from DB2 into SAS. SAS/ACCESS for DB2 supports the `DBSLICEPARM` and `DBSLICE` modes for threaded reads.

On a single-partition DB2 system you can use the `DBSLICE` or `DBSLICEPARM` option. We started by testing the automatic threaded read mode by setting `DBSLICEPARM`. When you use this option, SAS/ACCESS automatically determines a partitioning scheme for reading the data using the `mod()` database function. We tested the `freq` procedure using `dbsliceparm=(all,2)` which creates two threads that read data from the database.

```
proc freq data=census.hrecs_db
    (dbsliceparm=(all,2));
    table state tenure yrbuilt
    yrmoved msapmsa;
run;
```

Code Example: 17

DBSLICEPARM=(ALL,2)

ALL: Makes all read-only procedures eligible for threaded reads.

2: Starts two read threads.

When this statement is executed SAS creates two queries to access the database:

Generated Query 1:

```
SELECT "STATE", "TENURE", "YRBUILT",
"YRMOVED", "MSAPMSA"
FROM HRECS_DB
WHERE
({FN MOD({FN ABS("SERIALNO")},2)}=0
OR "SERIALNO" IS NULL ) FOR READ ONLY
```

Generated Query 2

```
SELECT "STATE", "TENURE", "YRBUILT",
"YRMOVED", "MSAPMSA"
FROM HRECS_DB
WHERE
({FN MOD({FN ABS("SERIALNO")},2)}=1
OR "SERIALNO" IS NULL ) FOR READ ONLY
```

Code Example: 18

In this example SAS automatically generates two queries the first with the `mod(serialno,2)=0` predicate and the second with the `mod(serialno,2)=1` predicate. These queries are executed in parallel. In this test the same statement ran in 36.56 seconds, 10% faster than using `READBUFF` alone. If you are running a partitioned database you can use the `DBSLICE` threaded read mode.

We set up a two partition database and tested the `DBSLICE` threaded read mode. When the `DBSLICE` option is specified SAS opens a connection to each partition and retrieves the data directly from the specified node. To do this you need to tell SAS what partitions you would like to access and the partitioning key.

We configured a two-logical partition DB2 database on the test server and added the `DBSLICE` syntax in place of `DBSLICEPARM` in the SAS script.

```
proc freq data=census.hrecs_db
  (DBSLICE= ("NODENUMBER(serialno)=0"
  "NODENUMBER(serialno)=1"));
  table state tenure yrbuilt yrmoved
  msapmsa;
run;
```

Code Example: 19

This time the query ran in 35.13 seconds, 14% faster than the `READBUFF` option alone. As you can see, these threaded read options help improve data extraction performance. This was a small two-partition system. As the database size or number of partitions increases, these threaded read options should help even more.

Data read performance is most important when using SAS to extract data from DB2 but it is also important to tune your applications that add rows to the database.

We started the insert evaluation by testing different values of the `DBCOMMIT` `DATA` step parameter. `DBCOMMIT` is the number of rows inserted into the database between transaction commits. To understand how this works lets look at what is happening behind the scenes when inserting a row into the database.

To insert one row into a database table there are many operations that take place behind the scenes to complete the transaction. For this example we will focus on the transaction logging requirements of an SQL `insert` operation to demonstrate the impact of the SAS `DBCOMMIT` option.

The DB2 database transaction log records all modifications to the database to ensure data integrity. During an insert operation there are multiple records recorded to the database transaction log. For an insert the first record is the insert itself followed by the commit record that tells the database the transaction is complete. Both of these actions are recorded in the database transaction log in separate log records. For example, if you were to insert 1000 rows with each row in it's own transaction it would require 2000 (1000 insert and 1000 commit) transaction log records. If all these inserts were in a single transaction you could insert all the rows with 1001 transaction log records (1000 insert and 1 commit).

You can see that there is a considerable difference in the amount of work required to insert the same 1000 rows depending on how the transaction is structured. By this logic, if you are doing an insert, you want to set `DBCOMMIT` to the total number of rows you need to insert. This requires the least amount of work, right? Not quite, as with any performance tuning there are tradeoffs.

For example, if you were to insert 1 million rows in a single transaction, this will work but it requires a lock to be held for each row. As the numbers of locks required increases the lock management overhead increases. With this in mind you need to tune the value of `DBCOMMIT` to be large enough to limit commit processing but not so large that you encounter long transaction issues (locking, running out of log space etc). To test insert performance we used a `DATA` step that processed the `hrecs` table and created a new `hrecs_temp` table containing all the rows where `state = '01'`.

```
libname census db2 db=census user=db2inst1
  using=password;

data census.hrecs_temp (DBCOMMIT=10000);
  set census.hrecs;
  where state = '01';
run;
```

Code Example: 20

The default for `DBCOMMIT` is 1000. We started testing with 10 just to see the impact.

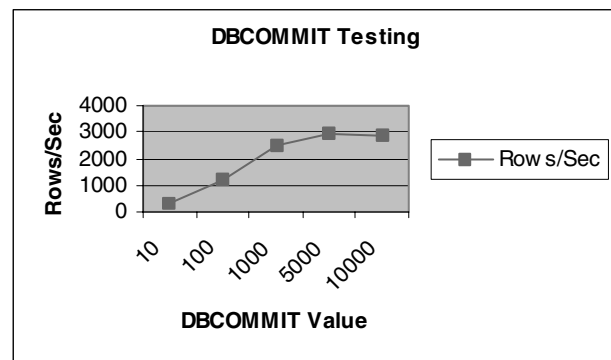
DBCOMMIT	Time (seconds)
10	277.80
100	70.61
1,000 (default)	34.29
5,000	29.5
10,000	29.64

As you can see the default works pretty well. In some situations, like this one, larger values of `DBCOMMIT` can yield up to a 16% performance improvement. We found in this test that the best value was between 1,000 and 5,000 rows per commit.

Performance Tip

Start with `DBCOMMIT` between 1,000 and 5,000 and tune from there.

We can also see that at some point, increasing the value of `DBCOMMIT` no longer improves performance (Compare 5,000 to 10,000).



In fact, if the dataset were large enough we would probably see a decrease in performance with extremely high values of `DBCOMMIT`. Now that we have `DBCOMMIT` tuned we take a look at another parameter that impacts insert performance `INSERTBUFF`.

INSERTBUFF is another tunable parameter that affects the performance of SAS inserting rows into a DB2 table. INSERTBUFF is a CLI parameter similar to READBUFF but for inserts. It tells the CLI client how many rows at a time to send to the DB2 server. To enable insert buffering you need to set two libname options: INSERT_SQL and INSERTBUFF.

```
libname census db2 db=census user=db2inst1
using=password;

data census.hrecs_temp (INSERT_SQL='Yes'
INSERTBUFF=10 DBCOMMIT=5000);
  set census.hrecs;
  where state = '01';
run;
```

Code Example: 21

INSERT_SQL must be set to "Yes". INSERTBUFF is an integer ranging from 1 to 2,147,483,648. We did some testing with different values of INSERTBUFF to see what impact it would have on this same DATA step.

INSERTBUFF	Time (seconds)
1	34.25
10	30.14
25	29.56
50	29.62
100	30.68

Increasing the value of INSERTBUFF from 1 to 10 improved the performance 12%. Increasing the value over 10 did not have a significant impact on performance.

THE DBA CORNER

HOW DOES SAS USE MY DATABASE?

As DB2 DBAs we see SAS as a consumer of database resources. In this section we highlight a few topics of interest to the DB2 DBA. We look at the way SAS connects to the database and uses other database resources as well as some debugging tips for SAS applications in a SAS/ACCESS for DB2 environment.

CONNECTIONS

Each SAS client may open multiple connections to the database server. When a SAS session is started a single data connection is opened to the database. This connection is used for most communication from SAS to DB2. If the SAS application requires a list of DB2 tables, executing `proc datasets` for example, a second, utility connection is created. This utility connection is designed to allow SAS to collect this information without interfering with the original data connection. It also allows these utility operations to exist in a separate transactional context minimizing the locks required on the database catalogs, for example. Once opened these two connections remain active until the SAS session is completed or the libname reference or database connection (opened using `connect to...`) is explicitly closed. Other connections may be opened automatically during a SAS session. These connections are closed when the operation for which they were opened is completed. For example, if you read from one table and write to a new table, SAS opens two connections: The original connection to retrieve the data and a connection to write the data into the new table. In this example the connection used to write the data will be closed when that DATA step is completed.

RESOURCE CONSUMPTION

As DBAs we are always interested in understanding what impact an application is going to have on the database. SAS workloads can vary greatly depending on your environment but here are a few places to start evaluating your situation:

- Each SAS user is the equivalent of a single database Decision Support (DS) user. Tune the same for SAS as you would for an equivalent number of generic DS users.
- Tune to the workload. Just like any other DS application, understanding the customer requirements can help you to improve system performance. For example, if there is a demand for quarterly or monthly data, using a Multidimensional Clustering (MDC) table for the data may be appropriate.
- SAS is a decision support tool; if you need data from an operational data store consider the impact on your other applications. To offload some of the work you may consider creating a data mart to provide data to your SAS customers.
- In most environments the SAS server is located on a separate system from your database. Business analysis often requires many rows to be retrieved from the database. Plan to provide the fastest network connection possible between these systems to provide the greatest throughput.

RULES OF THUMB

- Try to pass as much where clause and join processing as you can to DB2.
- Return only the rows and columns you need. Whenever possible do not use a "select * ..." from your SAS application. Provide a list of the necessary columns, using `keep=(var1, var2...)`, for example. To limit the number of rows returned include any appropriate filters in the where clause.
- Provide Multidimensional Clustering (MDC) tables or Automatic Summary Tables (AST) where appropriate to provide precompiled results for faster data access.
- Use SAS threaded read (DBSLICEPARM, DBSLICE) and multi-row fetch (READBUFF) operations whenever possible.
- When loading data with SAS use the Bulk Load method CLI Load.

DEBUGGING

If you want to see what SQL commands SAS is passing to the database, enable the `sastrace` option. For example, here is the syntax to trace SAS/ACCESS SQL calls to DB2:

```
options sastrace ".,.,d" sastraceloc=saslog;
```

Applying a "d" in the fourth column of the sastrace options tells SAS to report SQL sent to the database. For example this SAS procedure:


```
data a.emp3;
set emp;
run;
```

Is logged as (the SQL commands are highlighted):

```
455 1356046811 rtmdoit 0 DATASTEP
DB2_5: Prepared: 456 1356046811 rtmdoit 0
DATASTEP
SELECT * FROM EMP3 WHERE 0=1 FOR READ ONLY
457 1356046811 rtmdoit 0
DATASTEP
458 1356046811 rtmdoit 0 DATASTEP
DB2: COMMIT performed on connection 1. 459
1356046812 rtmdoit 0 DATASTEP
DB2: AUTOCOMMIT is NO for connection 2 460
1356046812 rtmdoit 0 DATASTEP
461 1356046812 rtmdoit 0 DATASTEP
DB2_6: Executed: 462 1356046812 rtmdoit 0
DATASTEP
CREATE TABLE EMP3 (name VARCHAR(5),dept
VARCHAR(3),age FLOAT) 463
1356046812 rtmdoit 0 DATASTEP
464 1356046812 rtmdoit 0 DATASTEP
DB2: COMMIT performed on connection 2. 465
1356046812 rtmdoit 0 DATASTEP
466 1356046812 rtmdoit 0 DATASTEP
DB2_7: Prepared: 467 1356046812 rtmdoit 0
DATASTEP
INSERT INTO EMP3 (name,dept,age) VALUES ( ?
, ? , ? ) 468 1356046813
rtmdoit 0 DATASTEP
469 1356046813 rtmdoit 0 DATASTEP
470 1356046813 rtmdoit 0 DATASTEP
DB2_8: Executed: 471 1356046813 rtmdoit 0
DATASTEP
Prepared statement DB2_7 472 1356046813
rtmdoit 0 DATASTEP
473 1356046813 rtmdoit 0 DATASTEP
NOTE: There were 1 observations read from the
data set WORK.EMP.
DB2: COMMIT performed on connection 2. 474
1356046813 rtmdoit 0 DATASTEP
NOTE: The data set A.EMP3 has 1 observations
and 3 variables.
DB2: COMMIT performed on connection 2. 475
1356046813 rtmdoit 0 DATASTEP
DB2: COMMIT performed on connection 2. 476
1356046813 rtmdoit 0 DATASTEP
```

Code Example: 22

Hint: To find SQL information search the log for "DB2_".

Sastrace displays the processing details of the SAS script. The log includes the exact SQL commands that are submitted to the database. In the example above we executed a *DATA* step that SAS translated it into three SQL statements

```
SELECT * FROM EMP3 WHERE 0=1 FOR READ ONLY;

CREATE TABLE EMP3
(name VARCHAR(5),
dept VARCHAR(3),
age FLOAT);

INSERT INTO EMP3 (name,dept,age)
VALUES ( ? , ? , ? );
```

Code Example: 23

You can see what SAS/ACCESS is requesting on the database by enabling CLI trace. The DB2 CLI trace feature is useful for debugging SAS/ACCESS interaction with DB2. There are two ways to enable CLI trace: the DB2 Command Line Processor (CLP) command, "update cli cfg" or edit the sqllib/cfg/db2cli.ini file. To enable tracing using "update cli" enter

```
db2 UPDATE CLI config FOR common USING
TraceFileName /tmp/mytracefile
```

Then

```
db2 UPDATE CLI config FOR common
USING trace 1
```

If you choose to edit the db2cli.ini file directly add trace and TraceFileName in the common section

```
[COMMON]
trace=1
TraceFileName=/tmp/mytracefile
```

When you enable CLI tracing DB2 begins tracing all CLI statements executed on the server. DB2 will continue to trace CLI commands until you disable CLI tracing by setting trace to 0. Be careful, if this is a busy server you could collect huge amounts of output. It is best to run CLI trace on a test server running a single SAS session, if possible.

The saslog and DB2 log (sqllib/db2dump/sqdb2diag.log) are also useful places to look for information when you are troubleshooting.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Scott Fadden
 IBM
 315 SW 5th Ave
 Floor 10
 Portland, Oregon 97204
 (503) 525-7584
sfadden@us.ibm.com
www.ibm.com

SAS, SAS/ACCCESS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

DB2 is a registered trademark of IBM in the USA and other countries.

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