Paper 360-2011

Best Practices for the SAS® Scoring Accelerator for DB2®

Matthias Nicola, IBM Silicon Valley Lab, San Jose, CA
Kim Sherrill, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS Scoring Accelerator for DB2 enables you to deploy scoring models as user-defined functions in DB2. You can then use these functions in SQL statements to perform scoring inside DB2, directly on existing database tables. This paper presents best practices for DB2 database administrators and for developers that write SQL queries with scoring functions for a DB2 database. We provide performance guidelines for using the SAS Scoring Accelerator in a partitioned DB2 database (IBM® InfoSphere™ Data Warehouse) as well as tips for writing efficient SQL with scoring functions. The topics include result set handling, SQL coding, fenced vs. unfenced functions, logging, database and table partitioning, and workload management.

INTRODUCTION

The amount of business data that enterprises generate and collect continues to increase rapidly. Analyzing this data and gaining valuable business insight becomes ever more challenging. The SAS Scoring Accelerator for DB2 addresses this challenge by bringing the analytical capabilities of the SAS Enterprise Miner closer to the data that is stored in DB2 databases [1].

Traditionally, scoring models developed in the SAS Enterprise Miner can be applied to data in a DB2 database only after the data is extracted from the database and stored in a SAS Data Set. This means that data inside the DB2 database is duplicated outside the database (storage cost). Also, the network latency involved in moving large amounts of data delays the actual execution of a scoring model (performance cost). Additionally, there can be data governance cost in terms of data security outside the database.

The SAS Scoring Accelerator for DB2 changes this picture significantly and eliminates the extra storage, performance, and data governance cost. Scoring models developed in the SAS Enterprise Miner are converted by the Scoring Accelerator into C libraries that can be registered as user-defined functions (UDFs) in DB2. Once registered, the scoring functions can be used in any SQL statement just like in-built SQL functions in DB2. The scoring models are then executed by DB2's parallel query engine as part of the general SQL processing, directly on DB2 tables.

While previously the data was moved to the scoring models, the SAS Scoring Accelerator moves the scoring models to the data and into the database. After all, it's a lot cheaper to move a function than to move large amounts of data.

First, this paper briefly describes DB2 and its Data Partitioning Features (DPF) as well as the SAS Scoring Accelerator for DB2. Then we discuss a set of best practices that help ensure efficient operation of SAS scoring functions inside DB2.

DB2, INFOSPHERE WAREHOUSE, AND THE IBM SMART ANALYTICS SYSTEM

IBM's DB2 database is the backbone of the IBM InfoSphere Warehouse, which in turn is a key component of the IBM Smart Analytics System. InfoSphere Warehouse includes DB2 with the Database Partitioning Feature (DPF), the Warehouse Design Studio, a data movement and transformation component, and other tools that are useful in a business intelligence environment [2]. The IBM Smart Analytics System is a preconfigured hardware and software solution, combining InfoSphere Warehouse with a balanced configuration of servers and storage subsystems [3].

The DB2 Database Partitioning Feature (DPF) allows administrators to distribute a database across multiple database partitions in order to enable parallel processing of large data volumes. The rows of large database tables are evenly distributed across the database partitions through a hash function. The database partitions can then work in parallel, each processing a subset of a table ("divide and conquer").

The database partitions can work in parallel on the same server, separate servers, or a combination of both. For example, Figure 1 shows a configuration with eight DB2 database partitions running on four physical servers using two storage subsystems. Depending on the actual hardware selected, there could be more than or fewer than two database partitions per server. The partitioning is setup by the database administrator but it is transparent to users and application – they see and use a single database system.

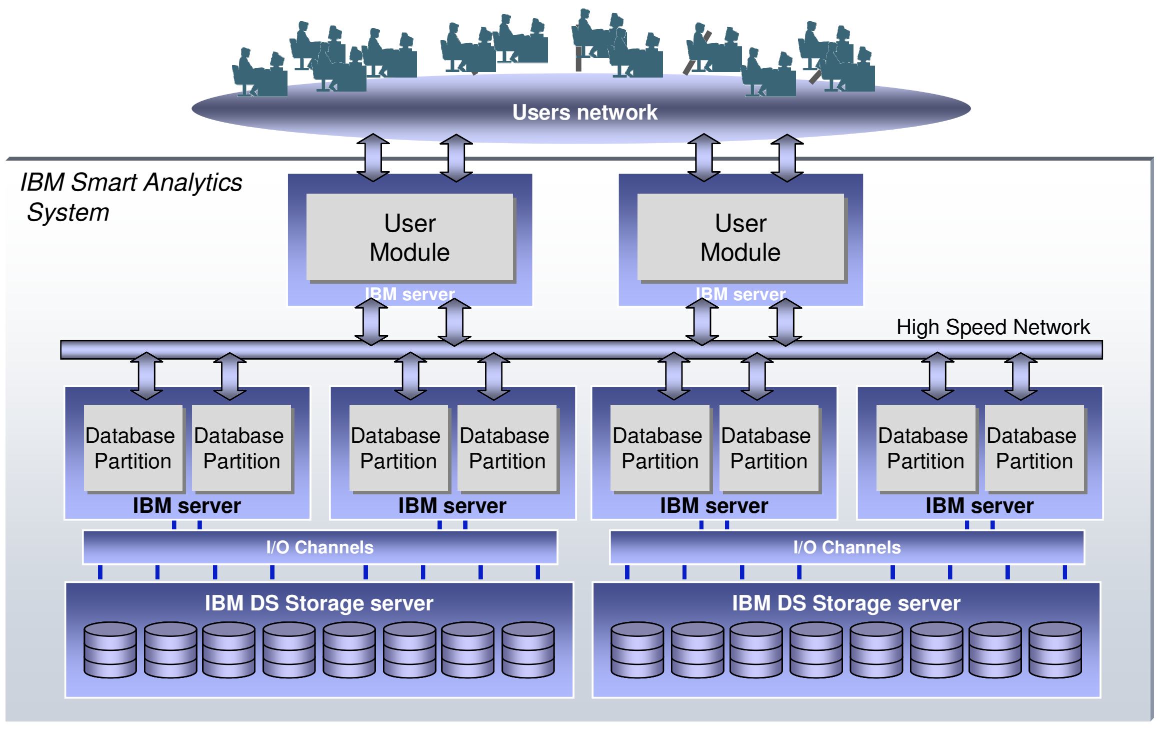

Figure 1: Sample configuration of a partitioned DB2 database

The DB2 Database Partitioning Feature is critical for the performance and scalability of the SAS Scoring Accelerator for DB2. The larger the number of database partitions the smaller the amount of data per partition and the greater the degree of parallelism for data processing. Consequently, database partitioning shortens the elapsed time for queries that analyze large amounts of data. This is important for the Scoring Accelerator, because the SAS scoring functions that run inside DB2 are automatically replicated across all database partitions. As a result, DB2's proven capabilities for parallelism and scaling are automatically applied to in-database scoring. The in-database scoring functions simply ride on DB2's existing infrastructure for parallel data processing. This is how DB2 enables massively parallel scoring on large amounts of data.

THE SAS SCORING ACCELERATOR FOR DB2

The basic steps for using Scoring Accelerator are the following:

1. Design a scoring model in the SAS Enterprise Miner, version 5.3 or higher
2. In the SAS Enterprise Miner, use the Score Code Export node to export the scoring code together with meta data of your model to the file system (see [1] for details).
3. In the SAS Program Editor, run the macro %INDB2PM and set the IDCONN macro variable to define connection and authentication details for the DB2 database and schema that you want to use, as described in [1].
4. Run the macro %INDB2_PUBLISH_MODEL. This macro registers the code of the scoring function as a User-Defined Function (UDF) in your DB2 database. See [1] for a description of the parameters for this macro.

Assume that this process has created two scoring UDFs, **`REG_EM_CLASSIFICATION`** and **`REG_EM_EVENTPROBABILITY`**, in your DB2 database. These UDFs are *scalar* functions, which means that they

- operate on one row at a time
- typically take a set of columns names as input arguments
- compute a single result value.

You can use these UDFs in any SQL statement and from any DB2 interface. For example, the SQL query in Figure 2 reads rows from the table `mydata` and applies both scoring functions to each selected row. For each row it returns the row's key (`id`) and two scores, a classification and a probability as defined in the original model.

```
SELECT id, REG_EM_CLASSIFICATION(count, dif_srvr, flag, hot,
             sam_srat, service, srv_cnt) as EM_CLASSIFICATION,
           REG_EM_EVENTPROBABILITY(count, dif_srvr, flag, hot,
             sam_srat, service, srv_cnt) as EM_EVENTPROBABILITY
FROM mydata
WHERE ...;
```

Figure 2: Sample query with scoring functions

Note: the number of scoring UDFs in the SQL statement can significantly affect performance. Typically, the more UDFs you include in a single query, the longer the execution time of the query.

BEST PRACTICES AND PERFORMANCE GUIDELINES

The following subsections discuss recommendations for using the SAS Scoring Accelerator in a DB2 database.

FENCED vs. UNFENCED USER-DEFINED FUNCTIONS

The Scoring Accelerator allows scoring functions to run as external user-defined functions in DB2. "External" means that the functions are implemented in C code rather than SQL. DB2 can run such external UDFs in **fenced** mode or **unfenced** mode.

- **Fenced** means *protected*. The C code that implements the UDF runs in a separate process from the DB2 server process. The benefit of a fenced UDF is safety, i.e. an error in the UDF might abort the UDF but cannot bring down the entire database server. However, there is some communication overhead between the DB2 server process and the fenced mode process that runs the UDF.
- **Unfenced** means *unprotected*. The UDF code runs within the DB2 server process, in the same address space. This is very efficient and provides performance benefits. Tests have shown that unfenced UDFs can be on average 30% faster than fenced UDFs.

Recommendation: Use fenced-mode UDFs for the initial testing of a new in-database scoring model. Once it is confirmed that the UDFs are stable, redeploy them in unfenced mode for best performance. The macro %INDB2_PUBLISH_MODEL has a parameter "MODE" that accepts the values "PROTECTED" and "UNPROTECTED" to choose fenced or unfenced, respectively. The default is "PROTECTED", i.e. fenced.

STORING SCORING RESULTS IN A DB2 TABLE

SQL statements that apply scoring functions to data in DB2 tables can return the computed scores directly to the application that issued the SQL, or they can write the scoring results to a DB2 table as shown in Figure 3.

```
INSERT INTO scoring_result(id, class, prob)
  SELECT id, REG_EM_CLASSIFICATION(count, dif_srvr, flag, hot,
                sam_srat, service, srv_cnt) as EM_CLASSIFICATION,
             REG_EM_EVENTPROBABILITY(count, dif_srvr, flag, hot,
                sam_srat, service, srv_cnt) as EM_EVENTPROBABILITY
  FROM mydata
  WHERE ...;
```

Figure 3: Inserting scoring results into a database table

Recommendation: If the results of an in-database scoring operation need to be stored in a database table, explicitly issue an SQL INSERT statement such as the one in Figure 3. For example, you can issue such INSERT statements through the EXECUTE() function of a SAS SQL Proc. Do *not* use the SAS client to specify "*db2 libref*" as the target for the scoring results, as this requires results to be fetched from DB2 into the SAS client only to be sent back to DB2, which is inefficient and incurs unnecessary network overhead.

If you want to store scoring results in a DB2 table *and* retrieve them back to a SAS Data Set for post processing, you should first issue an INSERT statement such as in Figure 3 and then a separate SELECT statement to read from the table `scoring_result` into the SAS client. The INSERT and the subsequent SELECT from the results table can also be combined in a single SQL statement, as shown in Figure 4:

```
SELECT * FROM NEW TABLE
 (INSERT INTO scoring_result(id, class, prob)
   SELECT id, REG_EM_CLASSIFICATION(count, dif_srvr, flag, hot,
                sam_srat, service, srv_cnt) as EM_CLASSIFICATION,
              REG_EM_EVENTPROBABILITY(count, dif_srvr, flag,hot,
                sam_srat, service, srv_cnt) as EM_EVENTPROBABILITY
   FROM mydata
   WHERE ...);
```

Figure 4: Combining Select and Insert in a single "select-from-insert" statement

REDUCED LOGGING OVERHEAD FOR DB2 RESULT TABLES

Recommendation: If you use INSERT statements such as in Figure 3 to write scoring results into a target table, you might observe high database logging activity or even log full errors. To avoid log full errors, increase the size of the database log as needed by adjusting the database configuration parameter for log file size (`logfilsz`), number of primary log files (`logprimary`) and secondary log files (`logsecond`).

Alternatively, you can temporarily disable logging for the DB2 target table with the following ALTER TABLE statement:

```
ALTER TABLE scoring_result ACTIVATE NOT LOGGED INITIALLY;
```

Note that this statement has effect only for a single transaction, so you must issue it right before the INSERT statement and in the same transaction. After the commit of this transaction, logging is automatically reactivated for the table.

Be aware that disabling logging in this manner implies that the target table is not recoverable in case of an error. If the INSERT statement fails while the table is in the NOT LOGGED INITIALLY state, the only option is to drop and recreate the table. This might be acceptable if the table was empty and the INSERT can simply be rerun. However, dropping the table would be unacceptable if the table already contained existing data that is difficult, expensive, or impossible to reproduce. Hence, use the NOT LOGGED INITIALLY option with care.

PARTITIONING OF RESULT TABLES

In a partitioned DB2 database, large database tables are typically distributed across database partitions. For this distribution, one or multiple columns of a table must be declared as the distribution key. For each row, DB2 applies a hash function to the distribution key to hash the row to one of the database partitions. Rows with the same values in the distribution key are located on the same partition.

Recommendation: If data from a source table is read for scoring, and if the scoring results are inserted into another database table, then the target table should be partitioned in the same manner as the source table. This ensures that each insert of a scoring result happens at the same database partition where the corresponding source row has been retrieved and scored. Such "collocation" avoids the shipping of scoring results across DB2 partitions and is beneficial for performance.

Let's look at an example. Figure 5 shows the definition of the table `mydata` that contains the source data and the table `scoring_result` into which the scoring results are inserted. Both tables are distributed across the database partitions by hashing of the column `id`. If a row in `mydata` has the same id as a row in scoring_result, then those rows reside on the same database partition. They are collocated. If you use an insert statement such as the one in Figure 3 or Figure 4, then the scoring results for any given row from table mydata will always be inserted into table scoring_result on the same partition, without incurring inter-partition communication. This is recommended for best performance.

```
CREATE TABLE mydata(id BIGINT, count INT, dif_srvr CHAR(5), flag INT,
           hot INT, sam_srat DOUBLE, service VARCHAR(15), srv_cnt INT)
  DISTRIBUTE BY HASH (id);

CREATE TABLE scoring_result(id BIGINT, class VARCHAR(10), prob DOUBLE)
  DISTRIBUTE BY HASH (id);
```

Figure 5: Source and target table are collocated

RECOMMENDED DB2 FIXPACKS

Prior to DB2 9.5 Fixpack 7 or DB2 9.7 Fixpack 3, using ORDER BY clauses in the SQL queries that contain scoring UDFs may lead to suboptimal performance. This has been resolved in DB2 9.5 FP7 and DB2 9.7 FP3, where using ORDER BY clauses and scoring functions in the same query will perform well.

Recommendation: For best performance, upgrade your DB2 installation to DB2 9.5 Fixpack 7 or DB2 9.7 Fixpack 3 before using the SAS Scoring Accelerator for DB2.

USING THE DB2 WORKLOAD MANAGER TO CONTROL IN-DATABASE SCORING JOBS

Running scoring models inside a database adds additional resource consumption to an existing data warehouse workload. In particular, advanced and complex scoring models can be quite CPU intensive. Proper workload management is often required to meet service level agreements, maintain system stability, and prevent overly expensive queries from draining your system resources. The DB2 Workload Manager enables you to achieve these goals. It allows you to recognize and classify incoming work and control the resource consumption for different classes of workloads.

Recommendation: Use the DB2 Workload Manager to maintain control of the activities in your data warehouse by detecting and limiting the number of scoring jobs that run concurrently in your system.

A complete introduction to the DB2 Workload Manager is beyond the scope of this paper. In this section we only provide a glimpse of its capabilities. Please refer to the resources listed at the end of the paper for more detailed information and workload management best practices [6] [7] [8] [9].

Workload management in DB2 can be roughly divided into three phases:

1. *Identify workloads* and assign them to *service classes*.

   Incoming work can be identified (recognized) based on application names, client user IDs, client workstation names, session authorization IDs, or other attributes that are detected when a workload connects to the database. Incoming connections that are not recognized are assigned to a default workload.

   Another option is to classify incoming SQL statements based on their *estimated* execution cost that is projected by the DB2 optimizer. For example, you can define cost ranges to group SQL statements into light, medium, and heavy work. However, this approach is insufficient for SQL statements that contain scoring UDFs deployed by the Scoring Accelerator, because the DB2 optimizer has no knowledge about the execution cost of these UDFs. Therefore, you should ensure that all applications that use the scoring UDFs can be identified based on application name or other connection attributes.

2. *Manage* your workloads by defining limits and thresholds for the corresponding service classes. You can also define actions that are triggered when a threshold is exceeded.
3. *Monitor* the system to understand workload behavior and adjust workload management settings if needed.

Repeat steps 1 through 3 to refine your workload management configuration iteratively. In some cases you may have to start with monitoring to learn about your system and the work that it performs.

After you *identify* workloads and assign them to service classes (step 1 above), you can *manage* service classes in the following ways, all of which are optional:

- Set *thresholds* (limits) for quantities such as:
    - Maximum elapsed time
    - Number of concurrent connections to the same service class
    - Number of concurrent workload activities (such as queries)
    - Number of rows returned
    - etc.
- Define actions that are triggered when a *threshold* is exceeded, such as:
    - *Stop execution* of the activity that exceeds the threshold and return an error code to the application.
    - *Continue execution* but collect information so that an administrator can adjust workload management settings later.
    - *Queue activities*. For example, if a concurrency threshold is exceeded then the incoming job is queued for later execution. Optionally, you can specify a maximum queue length, and if the queue length is exceeded, the *stop execution* action is triggered.
- Set priorities to control resource consumption:
    - Set the relative CPU priority of a service class.
      On Linux and UNIX, CPU priorities range from -20 to 20. Negative numbers denote higher priority. The default priority is 0.
    - Set the I/O priority (prefetch priority) of a service class
    - Set an external workload management tag to control the resource consumption via the AIX Workload Manager

In many cases it is better to limit resource consumption with concurrency thresholds than with CPU and I/O priorities. There are several reasons for this recommendation:

- Concurrency thresholds are very useful to reflect system capacity and to express the desired distribution of resources across different service classes.
- Throttling queries by reducing their priority means that the queries take longer and they hold resources for a longer time, such as memory, locks, or temporary space on disk. Occupying these resources longer than necessary can sometimes have a negative impact on overall system performance.
- Limiting the number of concurrent activities reduces the contention for system resources.
- It is often better to delay the start of an SQL statement than to slow it down.

The following is an example of a simple workload manager configuration in DB2. This example is for illustration purposes only. In a real system you need to decide which actual workload policies are appropriate for your system and your users. Although you can use statements such as the following to set up a workload management configuration, you can also use the visual *Workload Manager Configuration Tool* that is part of the IBM Optim Performance Manager and described in resources [8] and [9].

```
-- Create a workload "wl_scoring" identified by application name "sas",
-- assign it to a service class "sc_scoring", and enable its usage:

CREATE SERVICE CLASS sc_scoring;

CREATE WORKLOAD wl_scoring CURRENT CLIENT_APPLNAME('sas')
       SERVICE CLASS sc_scoring COLLECT ACTIVITY DATA WITH DETAILS;

GRANT USAGE ON WORKLOAD wl_scoring TO PUBLIC;


-- For the service class "sc_scoring", set the following 3 policies:

---- 1. Define a threshold to limit the number of concurrent activities
---- to five. Additional incoming queries will be queued for later
---- execution ("continue") when current queries have completed:
CREATE THRESHOLD queue_activities FOR SERVICE CLASS sc_scoring
  ACTIVITIES ENFORCEMENT DATABASE
  WHEN CONCURRENTDBCOORDACTIVITIES > 5 CONTINUE;

---- 2. Define a threshold for the number of rows returned (max: 10M)
---- The threshold action is "continue", which does not affect running
---- queries but enables you to detect and record such events:
CREATE THRESHOLD big_result_sets FOR SERVICE CLASS sc_scoring
  ACTIVITIES ENFORCEMENT DATABASE
  WHEN SQLROWSRETURNED > 10000000 CONTINUE;

---- 3. Define an elapsed time threshold, so that queries running
---- longer than 30 minutes are terminated:
CREATE THRESHOLD long_running_queries FOR SERVICE CLASS sc_scoring
  ACTIVITIES ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME > 30 MINUTES STOP EXECUTION;


-- Set up monitoring for the policies that are defined above.
---- a) Create and enable an event monitor for activities:
CREATE EVENT MONITOR emon_activities FOR ACTIVITIES WRITE TO TABLE;
SET EVENT MONITOR emon_activities STATE 1;

---- b) Create and enable an event monitor for thresholds:
CREATE EVENT MONITOR emon_thresholds
  FOR THRESHOLD VIOLATIONS WRITE TO TABLE;
SET EVENT MONITOR emon_thresholds STATE 1;
```

```
-- Use queries such as the following to monitor system activity.

---- Find any threshold violations:

SELECT thresholdid, appl_id, activity_id, threshold_maxvalue,
       threshold_predicate, threshold_action, time_of_violation
FROM thresholdviolations_emon_thresholds
ORDER BY threshold_action, threshold_predicate, time_of_violation;

---- Find the Top 5 longest running queries:

SELECT SUBSTR(appl_id,1,26) as appl_id,
       SUBSTR(CHAR(activity_id),1,10) AS activity_id,
       SUBSTR(appl_name, 1,10) AS appl_name,
       SUBSTR(activity_type,1,10) AS type,
       TIMESTAMPDIFF(2, CHAR(time_completed-time_started)) AS totaltime,
       SQLCODE, SUBSTR(session_auth_ID,1,8) AS user,
       SUBSTR(service_superclass_name,1,20) AS service_superclass_name
FROM activity_emon_activities AS A
WHERE partition_number = current dbpartitionnum
ORDER BY totaltime DESC
FETCH FIRST 5 ROWS ONLY;
```

SUMMARY

The SAS Scoring Accelerator for DB2 is a simple yet very powerful solution to deploy scoring models inside DB2 and run them directly against database tables or views. It is recommended that you initially deploy new scoring models in "fenced" mode for testing purposes. Once you have confirmed that they run stable you can redeploy them in "unfenced" mode for better performance. You can store scoring results in a DB2 table, if you wish. In this case, performance is best if you perform the scoring and insertion of the results in a combined INSERT-SELECT statement. Also, the results table should be partitioned like the source data to avoid unnecessary shipping of rows across database partitions. And finally, proper workload management is important to control the resource consumption of SAS scoring jobs in your data warehouse and to meet service level agreements. Make yourself familiar with the DB2 Workload Manager and consider using the workload manager tooling in the IBM Optim Performance Manager.

RESOURCES

[1] SAS Scoring Accelerator for DB2, User's Guide and Administrator's Guide: http://support.sas.com/documentation/onlinedoc/scoraccldb2/index.html

[2] IBM InfoSphere Data Warehouse (based on DB2 with DPF): http://www.ibm.com/software/data/infosphere/warehouse

[3] IBM Smart Analytics System: http://www.ibm.com/software/data/infosphere/smart-analytics-system

[4] DB2 Best Practices: http://www.ibm.com/developerworks/data/bestpractices

[5] DB2 Information Center: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

[6] Introduction to DB2 Workload Management, IBM White Paper, http://www.ibm.com/developerworks/forums/servlet/JiveServlet/download/1116-166950-13965175-231542/Introduction%20to%20DB2%20workload%20management.pdf

[7] Best Practices for DB2 Workload Management, http://www.ibm.com/developerworks/data/bestpractices/workloadmanagement/

[8] DB2 Workload Management using Performance Optimization Feature, http://www.ibm.com/developerworks/offers/lp/demos/summary/im-optimwlm.html

[9] IBM Optim Performance Manager for DB2 for Linux, UNIX, and Windows, IBM Redbook, February 2011, see Chapter 11"Workload Manager Configuration Tool" http://www.redbooks.ibm.com/abstracts/sg247925.html?Open

CONTACT INFORMATION

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

Matthias Nicola
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95123
mnicola@us.ibm.com

Kim Sherrill
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Kim.Sherrill@sas.com