Database ACCESS basics: SQL and data step interface to relational databases

Ashley H. Sanders, USDA, Beltsville, MD

ABSTRACT

SAS/ACCESS® provides virtually seamless interaction with relational databases such as DB2®. The SQL pass-through facility gives direct access to the database, but requires knowledge of the native SQL syntax. Alternately, the libname statement can be used to assign SAS® library references to database objects, allowing database tables to be used like SAS data sets. SAS SQL functions are available that cannot be used with the pass-through facility. For data step diehards, database tables can be treated (almost) just like indexed SAS data sets. SAS statistical procedures can also reference database tables directly using the libref name. This paper covers the basics of accessing database data from SAS using SQL and data step language, using the database with SAS procedures, and getting information about the database delivered through SAS.

INTRODUCTION

Many businesses utilize the power of SAS® analytics while using a separate relational database management system (RDBMS or DBMS) for data storage. SAS/ACCESS® allows these components to work together. Actually a family of separate products, each SAS/ACCESS engine functions to translate requests from SAS into the appropriate query syntax for a specific DBMS or file structure. Specific DBMS access engines are available for DB2®, ORACLE®, OLE DB, SYBASE®, Teradata®, Informix®, MySQL, MS SQL Server, and ODBC. Other data sources that can be connected to with SAS/ACCESS include ADABAS®, and PC file formats including .DBF, and .XLS. The connection to these data sources is virtually transparent and can be used to read, update, create and delete data tables or records in the native data source. Each connection to the data source can be defined using the LIBNAME statement. In a windowing environment, the connection can be defined from the Explorer window. If it is necessary to pass non-ANSI standard query language to the DBMS, the SAS Pass-Through Facility is available to pass native SQL directly to the DBMS without translation or optimization by SAS. This paper covers basic considerations of utilizing DBMS through SAS including LIBNAME syntax, optimization of SQL, use of DBMS in the Data Step, and data translation issues.

LIBNAME STATEMENT

Since SAS v7, the general LIBNAME syntax has been extended to cover connections to other data sources. For review, an example of a basic LIBNAME statement is:

```
LIBNAME mydata 'C:\project\sasdata' ;
```

mydata is a shortcut name to associate with the physical location ('C:\project\sasdata') recognized by the OS. A few options are also available that affect how data in the defined library is handled. One is the ACCESS= option which can be READONLY or TEMP (no CPU used to monitor data integrity).

```
LIBNAME mydata 'C:\project\sasdata' ACCESS=READONLY ;
```

To associate a *libref* (mydata is the *libref* in the above example) with a DBMS database, this syntax is extended to include the SAS/ACCESS engine name and connection options:

```
LIBNAME mydata db2 DEFER=YES ACCESS=READONLY ;
```

Notice that the file location is gone. 'db2' specifies which ACCESS engine to use and the DEFER= option specifies that the connection not be made until it is referenced. The connection can be terminated by de-assigning the libref using the LIBNAME statement with no parameters:

```
LIBNAME mydata ;
```

The libname option ACCESS=READONLY is still valid for the DB2 engine. Connection and libname options control how SAS manages the actual connection to the DBMS and define how data objects are processed or handled. Many libname options are available for use with each DBMS data sources. Their availability or default behavior is often DBMS-specific.

CONNECTING TO THE DATA SOURCE

Programming Options

The default user information for connections to DB2 comes from the session variables $USER and $PASSWD. The default for connections to Oracle is OPS$sysid. If this is appropriate, the USER= and PASSWORD= options are not required. USER= and PASSWORD= must be used together. The default user connections are preferable because they use the security already in place on your system to protect user/password information. If user and password must be specified within the SAS session, it is generally not advisable to store those passwords in open SAS source code. This can be avoided in several ways. From a windowing interface, the libname option DBPROMPT=YES causes a window that interactively prompts you for the DBMS connection options the first time the libref is used. To force this prompt to appear when the LIBNAME statement is processed, add the option DEFER=NO.

```
LIBNAME mydata db2 DBPROMPT=YES DEFER=NO ;
```

If a window interface is not available, the password can be entered at program execution as a parameter, to be stored in the automatic macro variable &SYSPARM. The connection options would then be:

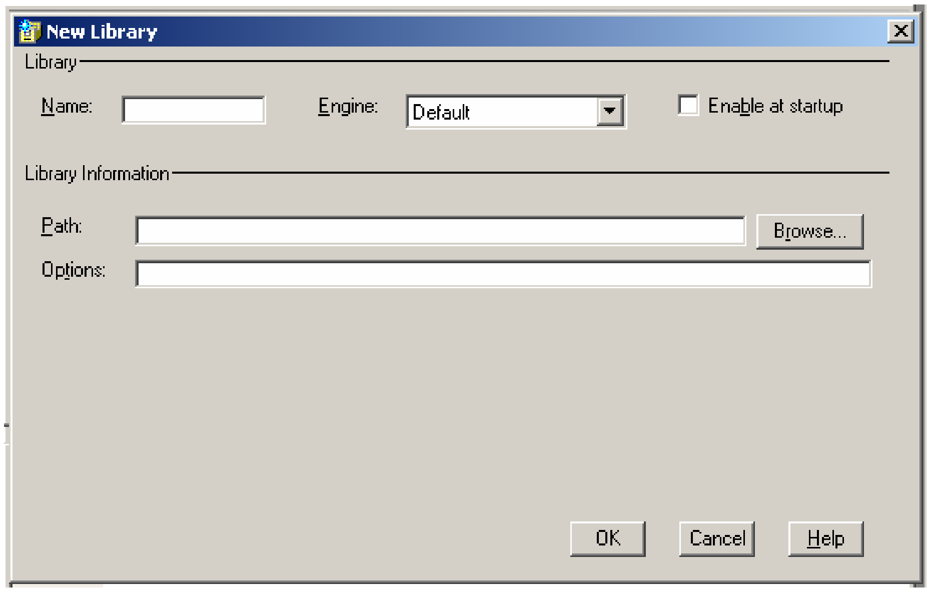
```
LIBNAME mydata db2 USER=topcoder PASSWORD=&sysparm ;
```

The password can also be stored in a SAS data set with system access limited to the session user. Read into a macro variable, it can be used as &sysparm is above.

For DB2, other important options are DATASRC= (alias DATABASE=, or DB=) which is set to the name of the DB2 database and SCHEMA= which is set to the appropriate DB2 schema. If these are omitted, SAS will connect to the default instance for your DB2 configuration. The PATH= option for Oracle connections specifies the Oracle driver, node, and database. My SQL connections can include the DATABASE= SERVER= and PORT= options. For PC file formats available options can depend on the specific file format. One important option for .XLS files is the HEADER= (alias HDR=, GETNAMES=). When set to YES this option specifies that the first row of data be used for column (variable) names.

Defining Connections Interactively

Libraries, including those referencing DBMS and other non-SAS data sources, can also be defined from the SAS explorer window. Select the Explorer window and view the 'Contents of SAS environment' (this is the highest view reached using the 'up one level' tool) and select 'Libraries.' From the 'File' pull-down menu select 'New' to bring up the New Library window



The name you enter will be the libref used to refer to data in the library. Select the appropriate Engine for your data source from the pull down list. The 'Library Information' options will change depending on the engine you select.

Checking the 'Enable at startup' box will run the libname definition anytime you open an interactive SAS session (The way to make libref available in programs run in batch mode is to include a libname statement in your AUTOEXEC file. Libraries created this way are also made available in interactive sessions run in the same environment).

Warning: Using 'Enable at startup' with a DBMS causes a session invocation of a connection to the database that sits around waiting for requests (even if the DEFER option is used, the connection will persist after the first use). While this may not noticeably affect CPU use (when they are idle, they don't use much), some DBMS are configured to allow only a limited number of simultaneous connections.

Once you have assigned a library reference to the data source, you can reference tables as if they were SAS data sets. Some important differences should be remembered when working with DBMS data. While SAS data sets can be sorted, this has no meaning in the context of DBMS data. Also, SAS has DBMS specific default behavior for translating SAS data types to and from DBMS data types. Most data manipulations that involve DBMS data are best handled by the DBMS; program to maximize work done by the DBMS and limit SAS data handling to operations that require SAS functions and procedures.

USING DBMS DATA IN THE DATA STEP

Maybe you are a data step diehard, or maybe you are reading raw data with the infile statement. Maybe you are using options specific to the data step like ‘first.’ and ‘last.’ variables or the lag or retain functions. Whatever your purpose, DBMS data is available in the data step using a libref defined in the LIBNAME statement.

BASIC DATA STEP

Retrieving DBMS data

The SET statement is the most familiar way to read stored data into a SAS data set. In this example the DBMS table BREED_TABLE contains cattle breed names and their 2-letter breed codes.

```
libname db2lib db2 DB=mydbms SCHEMA=db2inst1 DEFER=Yes ACCESS=READONLY;
data breeds;
    set db2lib.BREED_TABLE;
run;
```

The libref db2lib was created in the LIBNAME statement and points to the DBMS database and schema that includes BREED_TABLE. Data step options are available. For example:

```
data breeds;
    set db2lib.BREED_TABLE (where=(SPECIES_CODE=’B’));
    drop SPECIES_CODE;
run;
```

The where clause limits the data set breeds to cattle (bovine) breeds although BREED_TABLE includes breeds for cattle (SPECIES=’B’) and goats (caprine, SPECIES=’C’)

BREED_TABLE is relatively small, so for the DBMS to check each record for matches to the where clause is not prohibitively time consuming, but knowledge of the available DBMS indexes is important when extracting data from large tables. Lactation records for cows (~90% of data) and goats (~10% of data) are stored together in the DB2 table LACT_TABLE. The table has the primary index SPECIES_CODE + BREED_CODE + ANIM_KEY + CALV_PDATE (CALV_PDATE is a 5-digit perpetual (SAS) date). The task is to extract lactation records for Holstein (BREED_CODE=’HO’) cows from 2000 to 2002. Consider the code:

```
data ho_lacs (where=(BREED_CODE=’HO’ and
                     “01jan2000”d le CALV_PDATE le “31dec2002”d));
    set db2lib.LACT_TABLE;
run;
```

This program will run, and produce the correct results, but it represents the worst case for extracting data from DBMS tables. The ENTIRE DBMS table must be read into the program data vector (PVD), and the sub-setting is done by SAS as records are written out to the data set. Basic programming sense tells us to move the where clause to the set statement.

```
data ho_lacs;
    set db2lib.LACT_TABLE (where=(BREED_CODE=’HO’ and
                                  “01jan2000”d le CALV_PDATE le “31dec2002”d));
run;
```

This is an improvement, however the DBMS must still touch every record in the table before passing those matching the where clause back to SAS for processing (**Note**: the SAS date constant “01jan2000”d has no meaning to the DBMS. SAS translates the constants to numeric values to create a valid DBMS query). This is not be so terrible since more than 80% of all the records are Holstein cows, but what if the task is to extract records for Toggenburg (BREED_CODE=’TO’) goats (<1% of all records)? BREED_CODE by itself identifies the records we want because ‘TO’ is not a cattle breed, but SPECIES_CODE comes before BREED_CODE in the primary index for LACT_TABLE. For the DBMS to make use of this index, both must be accounted for in the where clause.

Thus the most efficient code becomes:

```
data to_lacs;
   set db2lib.LACT_TABLE (where=(SPECIES_CODE=’C’ and BREED_CODE=’TO’ and
                                “01jan2000”d le CALV_PDATE le “31dec2002”d));
run;
```

The DBMS will use the index to locate and retrieve only the small percentage of records requested. The DBMS will make use of a partial key, thus to extract all of the goat records it is possible to use a partial key including only SPECIES_CODE.

Creating DBMS data

It is possible to use SAS to create, modify, or delete DBMS tables. Maintaining data integrity, managing data access, and maintaining database integrity can be complicated when using a non-native application and is outside the scope of this paper. One important reason for creating DBMS tables however is to facilitate complex joins of DBMS data with other data sources. This will be covered in more detail later. When creating DBMS data using the data step, it can help the performance of DBMS joins later if the data type is defined on the data statement.

```
libname mydb2 db2 DB=mydbms;
filename bigzip pipe "zcat cows_hol.Z 2>bout.2";
data mydb2.cows (dbtype=(brd='CHAR(2)' cty='CHAR(3)' idn='CHAR(12)')
                      dbnull=(_ALL_=no))
   infile bigzip;
   input brd $2. cty $3. idn $12. ;
run;
```

The libref mydb2 was created in the LIBNAME statement and points to the DBMS database. Since no schema, or user are specified, the DB2 default environment for temporary user tables will be used (i.e. the session user must have some write privileges). The file cows_hol.Z is a very large zip file so the UNIX command zcat is used to read records out of the archive one at a time without needing the space to inflate the whole file at once. Some IDs include 12 numeric digits, but others include characters. If the transaction set includes only IDs that are all numeric, the default data type would be numeric, and these data could not be matched to the character field in the DBMS. Matching the data type of analogous data in the database eliminates the need for translation during processing, so it is generally more efficient. DBMS and SAS handle missing or null data differently. If you have null or missing data, be sure to read the DBMS specific references to understand how this data will be treated.

OTHER DATASTEP TOOLS

BY group processing

The BY statement in the data step makes the automatic variables first.*variable* and last.*variable* available. To use the BY statement for DBMS data, the table must be indexed on the BY variables. The BY statement can include the full index key, or a partial key.

DBKEY= ,DBINDEX=, and KEY=

Let’s say you have a mammoth data step which chugs through millions of records in the most convoluted data step imaginable, and right in the middle, what is really needed for some of the records is a piece of related data stored in a DBMS table. It is possible to use a SET statement to go out to the DBMS and get that bit of information only if and when it is needed. For example, in processing cow records the birth date of offspring resulting from embryo transfer (ET) is used to calculate the date the cow became an embryo donor.

```
if nocalf=1 then do;
   set db2lib.DONOR_DAM_TABLE (keep=ANIM_KEY PROG_BIRTH_PDATE dbkey=ANIM_KEY)
         key=dbkey;
         if _error_ then _error_ = 0;
         if PROG_BIRTH_PDATE ^= . then etdate=PROG_BIRTH_PDATE-280;
end;
```

If the cow has no next calving (e.g. if she donated her embryo), the SET statement references the DBMS table DONOR_DAM_TABLE which has an index on ANIM_KEY (or ANIM_KEY may be the valid partial key for a compound index). The KEY= statement tells SAS to use the same variable in the current PDV record as the look-up key for retrieving a record from the DBMS. You cannot retrieve multiple rows from the DBMS from a single look up record. Likewise, if no match is found, an error code is returned. re-setting _error_ to 0 prevents all the non-matches from cluttering the log. Additional processing for non matching records could also be nested with _error_ = 0 in a DO loop using if _error_ then do. Another automatic variable _IORC_, is created when you use the SET statement with the KEY= option. Using this variable for error checking can be more complex than using the simple _error_ return code. For a full explanation see the SAS documentation on error processing.

This process is most efficient when a small number of look-ups are to be done against a large DBMS. Although it is generally quite fast, it does require a separate query to the database for each record processed. Also, all the variables in the key must be included in the record from the PDV, with the same names as the variables in the DBMS table.

While DBKEY= identifies the key or partial key (variables) for the DBMS index, the values of the DBINDEX= option are YES (look for an appropriate index to use) or NO (don't look for an index), or the name of the specific DBMS index to use. DBINDEX can be used like DBKEY above, in the data step, or it can be used as an option on the LIBNAME statement that defines the DBMS libref. If DBINDEX is used, the value of KEY= is the name of the DBMS index. Use of DBKEY= overrides DBINDEX=.

DBCONDITION= can also be used as an option on the SET statement to accomplish this type of look-up. It allows the specification of DBMS-specific SQL query clauses, which SAS passes directly to the DBMS for processing. For information on structuring these clauses, see the documentation for SAS data set options for relational databases, and the documentation for the DBMS data source.

SQL

To work with DBMS data, experience with SQL syntax and PROC SQL is almost imperative. A complete introduction to SQL is outside the scope of this paper. For a solid foundation in SAS SQL, try the SAS training course "SQL Processing with the SAS System." Many excellent references are also available from SAS Press.

Since version 8, data set options have been available in PROC SQL. Care must be taken when using these options on DBMS data sources because not all options can be translated to the DBMS. The SAS SQL query optimization process is fairly robust, and SAS generally attempts to translate as much of a DBMS query as it can to pass to the DBMS for processing, but if the query contains SAS specific language that cannot be parsed out of the overall query, SAS will attempt to process the query by bringing the entire DBMS data table into SAS data space. This is almost always undesirable, if not impossible due to lack of resources.

While SAS may know what to do with DBMS data, the DBMS has no idea what to do with SAS data. On the other hand, while SAS is a fairly powerful data processor, that is the sole function of the DBMS, and thus getting it to do as much of the work as possible is desirable. SAS will, of course, do exactly what it is told to do, and it is regretfully simple to write perfectly valid SQL statements that prove the importance of these facts. Knowing how SQL statements will be processed is often as important as knowing what the end product will be.

BASIC SQL

Retrieving DBMS data

Whether the desired information is output or used to create a SAS data set, DBMS data can be read using the defined libref:

```
proc sql;
    create table breeds as
    select * from db2lib.BREED_TABLE
    where SPECIES_CODE='B';
quit;
```

SAS creates a query string from the program statements in valid ANSI standard SQL which it passes to the DBMS for processing. SAS functions are available, and translated if possible. Otherwise they are performed after data is retrieved from the DBMS. This is important, because all records which might meet a sub-setting requirement based on a function that must be performed by SAS will need to be retrieved from the DBMS for processing.

```
proc sql;
    create table breeds as
    select * from db2lib.BREED_TABLE
    where SPECIES_CODE=’B’ and substr(BREED_CODE,1,1)=’H’;
quit;
```

The SUBSTR function is not available in all DBMS, and even for those where it is (the DB2 function name is SUBSTRING), SUBSTR is processed by SAS unless the libname option SQL_FUNCTIONS= is set to ALL. In this example, SAS would pass the first part of the where clause to the DBMS to retrieve all of the records with SPECIES_CODE=’B’ and then check the returned records for matches to the SUBSTR function. In this case, only a few possible breed codes would match the SUBSTR so the problem can be avoided by restructuring the query using the IN operator

```
proc sql;
    create table breeds as
    select * from db2lib.BREED_TABLE
    where SPECIES_CODE=’B’ and BREED_CODE in (’HO’, ‘HI’);
quit;
```

This entire where clause would be passed to the DBMS for processing.

Combining SAS or raw data with DBMS data

Often, data from an outside source needs to be combined with DBMS data, but not stored permanently in the DBMS itself. To combine large non-DBMS data sets with data in the DBMS, it is most efficient to load the data into temporary space in the DBMS so that the data joining can be processed by the DBMS. Consult the database administrator to be sure access to temporary write space is available. You can use the data step to create the DBMS table, PROC SQL, or the DBLOAD procedure (however, in version 9, DBLOAD is no longer recommended)

```
data mydb2.cows;
    set sav.cow_data (keep=cowbreed cowid);
    by cowbreed cowid;
    if first.cowid;
run;

proc sql;
    select SPECIES_CODE, ANIM_KEY, ANIM_ID_NUM, COUNTRY_CODE, BREED_CODE, SEX_CODE
    from db2lib.ID_XREF_TABLE id, mydb2.cows cows
    where id.SPECIES_CODE='0' and
          id.ANIM_ID_NUM=cows.cowid and
          id.BREED_CODE=cows.cowbreed and
          id.COUNTRY_CODE='USA' and
          id.SEX_CODE='F';
quit;
```

In this example, the cow records were already stored in a SAS data set. The key variables were loaded into a temporary DBMS table COWS. This data was then available for an SQL join to data in the DBMS. Retrieving DBMS keys (ANIM_KEY) could be done through the data step, as shown in the previous sections, but for larger transaction sets, or when complex joins to data in multiple DBMS tables is required, this is a more efficient way to get the DBMS to process the data.

GRAB BAG

Using a libref, DBMS data can be used directly in PROCs. Any data set options that are valid in the PROC and valid for the DBMS are available.

The SQL pass-through facility allows for explicit ANSI (or DBMS specific) standard SQL code to be passed directly to the database. This is sometimes more efficient than the query that would be created by SAS. This is an advanced topic that requires knowledge of DBMS specific SQL so it has been omitted from discussion in this paper.

Store DBMS libname statements in the SAS macro library. Include the DEFER=yes option to keep from making unnecessary connection to the DBMS. Programs that require DBMS access require only a single line of code, librefs are standardized across programs, and nobody needs to remember all the libname options.

Use DBMS names for data whenever feasible. Use ALL_CAPS or some other easy identifier for DBMS variables.

The DB2 schema SYSCAT includes metadata tables; among them are TABLES, COLUMNS, and INDEXES (All DBMS keep this information somewhere). These tables can be accessed by SAS using a LIBNAME statement with the SCHEMA= option. Create a tool for getting a report of tables and table properties, including valid indexes so that SAS programmers can remain up to date on what data is available.

```
libname db2lib db2 datasrc=yourdb access=readonly
    schema=SYSCAT;
```

This is an example of the LIBNAME statement for accessing SYSCAT tables; ‘yourdb’ is the name of the db2 database.

```
proc sql;
    create table db2_tables as
          select trim(tb.TABNAME) as tname, substr(tb.TBSPACE,1,10) as tt,
                trim(tb.REMARKS) as rmk
          from db2lib.TABLES tb, db2lib.COLUMNS col
          where trim(col.TABSCHEMA)="YOURSCHEMA" and
                tb.TABNAME=col.TABNAME and
                tb.TABSCHEMA=col.TABSCHEMA
          order by tname;TABLES;
quit;
```

This creates a list of available tables with the tablespace they occupy and the remarks describing the tables contents stored in the TABLES table; ‘YOURSCHEMA’ is the primary schema for your database (DB2INST1, for example).

```
proc sql;
    create table col_&sysparm as
          select *
          from db2lib.COLUMNS
          where TABNAME = "&sysparm" and TABSCHEMA = "YOURSCHEMA"
quit;

data ndx_&sysparm (drop= TABNAME TABSCHEMA) ;
    length col $256;
    set db2lib.INDEXES (where=(TABNAME="&sysparm" and TABSCHEMA=" YOURSCHEMA")
keep=TABNAME TABSCHEMA COLNAMES INDNAME
    DEFINER UNIQUERULE NUMRIDS INDEXTYPE STATS_TIME);
run;
```

These examples use PROC SQL and the data step to retrieve columns in (col_&sysparm) and indexes on (ndx_&sysparm) the table named in the automatic variable &SYSPARM when the program is called. The three data sets created here can be sent to any kind of output, including HTML documents which could be posted on an internal website.

One feature of the INDEX table is the STATS_TIME variable. This gives the time of the last table re-org (process of updating metadata about DBMS tables that the DBMS uses to optimize data processing). If the table has been created, but the table re-org process was not completed, this column will be blank. For simple query clauses, the presence of a valid (matching the query) index is enough for the SAS query optimizer to pass the query to the DBMS for processing. For complex joins, however, SAS uses a combination of metadata, including information created by the re-org process. If this data is not available in the DBMS (the missing STAT_TIME is an indicator of this), SAS will not pass the query to the DBMS, but will try to process the query itself. This can have very undesirable results. Also, if many records have been added or changed in the DBMS since STAT_TIME, the optimizer may not make the best decisions for efficient processing of the data. When debugging DBMS queries, it can be a good idea to check the value of the STAT_TIME variable.

CONCLUSIONS

The SAS/ACCESS engine opens up the power of DBMS data management to SAS applications and makes the flow of data between storage (in the DBMS) and function (SAS analytical power) fairly seamless. To work effectively and efficiently, knowledge of the DBMS structure, understanding of SQL and the function of the SAS/ACCESS engine are required. The most successful programmer will also be willing to experiment with system-based optimization to determine which procedures produce the most satisfactory results based on correctness, and use of resources (memory, CPU time, and programmer time).

The paper is intended only to give examples of the types of syntax and programming that are possible using DBMS data in SAS processing and analysis. This is partly because the specific options available differ substantially between DBMS ACCESS engines. At the same time, the implication of the libref for DBMS data is that all the programming possibilities for SAS data can be utilized for DBMS data. SAS documentation is the best resource for determining the compatibility of SAS code and any specific DBMS.

REFERENCES

SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999, 300pp.

ACKNOWLEDGMENTS

SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.

DB2 and Informix are a Registered Trademarks of IBM.

ORACLE is a registered trademark of Oracle Corporation.

SYBASE is a registered trademark of Sybase Incorporated.

Teradata is a registered trademark of NCR

ADABAS is a registered trademark of Software AG

CONTACT INFORMATION

Ashley H. Sanders
Animal Improvement Programs Laboratory
BARC-West, Bldg. 005, Rm. 312
Beltsville, MD 20705-2350
Work Phone: 301-504-8667
Fax: 301-504-8092
Email: asanders@aipl.arsusda.gov
Web: http://aipl.arsusda.gov