Paper 8520-2016

From SAS® Data Management to Big Data Appliances: How SAS/ACCESS® Makes Life Easier

Magali Thésias, Senior Consultant, Deloitte Belgium

ABSTRACT

In the fast changing world of Data Management, new technologies to handle increasing volumes of (big) data are emerging every day. Many (large) companies are struggling in dealing with these technologies and more importantly on integrating them in their existing data management processes.
Moreover, they also want to rely on the knowledge built by their teams in existing products and implementing change to learn new technologies can therefore be a costly procedure.

SAS® is perfectly fitting in this situation by offering a suite of software that can be set up to work with any third-party database through the usage of the corresponding SAS/ACCESS®. Indeed, for every new database technology SAS® is releasing a specific SAS/ACCESS® allowing users to develop and migrate SAS® solutions almost transparently. Only few techniques have to be known by your users to combine the power of SAS® with a third-party (big) database.

This paper will help companies on dealing with the integration of rising technologies in their current SAS® Data Management platform using SAS/ACCESS®.
More specifically the focus will be on best practices, coming from project experiences, to succeed such an implementation integrating SAS® Data Management with the PureData for Analytics Appliance as an example.

INTRODUCTION

Volume of data to manage today is becoming bigger and bigger and its generation rate is increasing very rapidly. To illustrate this fact, we can tell that 90% of all the data in the world has been generated over the last 2 years[1]. New technologies are developed to help companies handle this huge volume of data. Amongst them, the Massively Parallel Processing (MPP), enabling splitting data and queries across a large number of nodes in order to perform simultaneous computation.

MPP technology allows Extract-Transform-Load (ETL) processes and Analytics to run faster; solving the case when execution time is becoming too big for the windows time frame allowed. Furthermore, companies can now integrate new data generated outside their organization and combine them to their data warehouse that was historically containing inside generated data only. This integration enables companies to discover new business facts and opens new opportunities by analyzing data (e.g. recommendation system based on clickstream analytics, review engine to capture shopping experience,…). Some current MPP technology available on the market are IBM PureData for Analytics appliance (previously named Netezza), Oracle Exadata, Teradata…, etc.

When companies are investing in this kind of technology, they want to switch system quickly without too much affecting what has already been done during the past years. SAS® is providing an easy way of migrating SAS® development between data sources: SAS/ACCESS® software. It is a way to read, write and update data stored on a third-party data source transparently as if it was a native SAS® source. Some best practices and specifics setup have to be applied to leverage the computing power of the third-party data source. This paper will explain them by using the PureData for Analtyics appliance as an example (historically named Netezza).

[1] SINTEF: ScienceDaily. 22 May 2013 Available at https://www.sintef.no/en/news/big-data--for-better-or-worse/

IN-DATABASE PROCESSING – BEST PRATICES

When implementing in-database processing tasks, some specific aspects of the data flow have to be taken into account. Indeed, when you are working with a third-party database, you would like to avoid overwhelming your network with unnecessary data transfer, especially with big (volume of) data.
By using the related SAS/ACCESS®, SAS® will automatically convert tasks into SQL code that is compliant with your third-party database.

In some condition, covered later on this paper, code cannot be converted. In this case, data will be downloaded on the SAS® server where the data transformation will take place before pushing-back the entire dataset to the third-party database.

To avoid these unnecessary Input/Output (I/O) the following best practices should be followed:

- Use an explicit pass-through SQL code when a task cannot be converted automatically into a database specific SQL code.
- Avoid unnecessary I/O between your SAS® server and your database: redirect your SAS® work by using a third-party database library, letting it to be the physical host of your datasets.
- Leverage the bulkload capabilities of your third-party database. Especially in case of a Big Data appliance like PureData for Analytics that is not optimized for single update or insert. Instead of doing a single update/insert, delete all rows that you want to update and append them using bulkload.
- While developing your tasks make sure that you use the database specific SQL functions.

Best practice to leverage in-database processing is to implement Extract-Load-Transform (ELT) flows instead of ETL (see **Figure 1**).

- Extract:
  - Access any data source or system.
  - Efficiently extract only the data needed.
- Load:
  - Stage data inside a database platform.
  - Utilize high-speed loaders to load data fast.
- Transform:
  - Transform inside the database.
  - Use database specific SQL, user defined functions, etc.
  - Leverage the database resources.

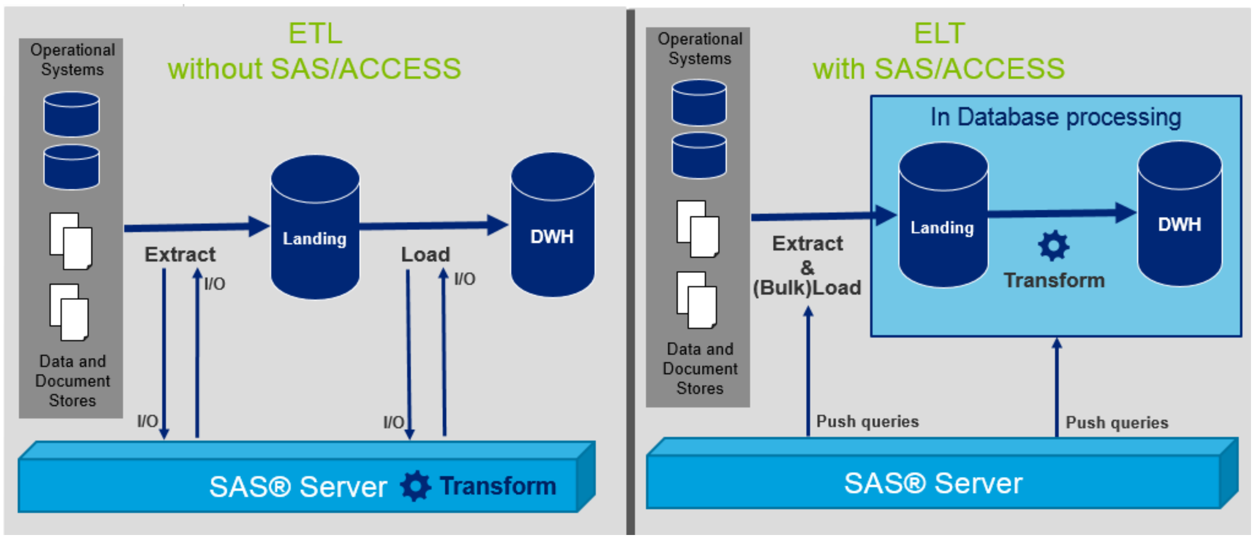


Figure 1: Processing differences between ETL and ELT using SAS/ACCESS®

SAS/ACCESS® USAGE IN SAS® BASE

SAS/ACCESS® has two faces using either an implicit pass-through or an explicit one based on the SAS® SQL pass through facility.

Implicit pass-through is automatically generating database specific SQL statements that which are submitted to the third-party database for processing. This is leveraged by the SAS® LIBNAME engine:

```
LIBNAME libref ENGINE <connection-options> <LIBNAME-options>;
```

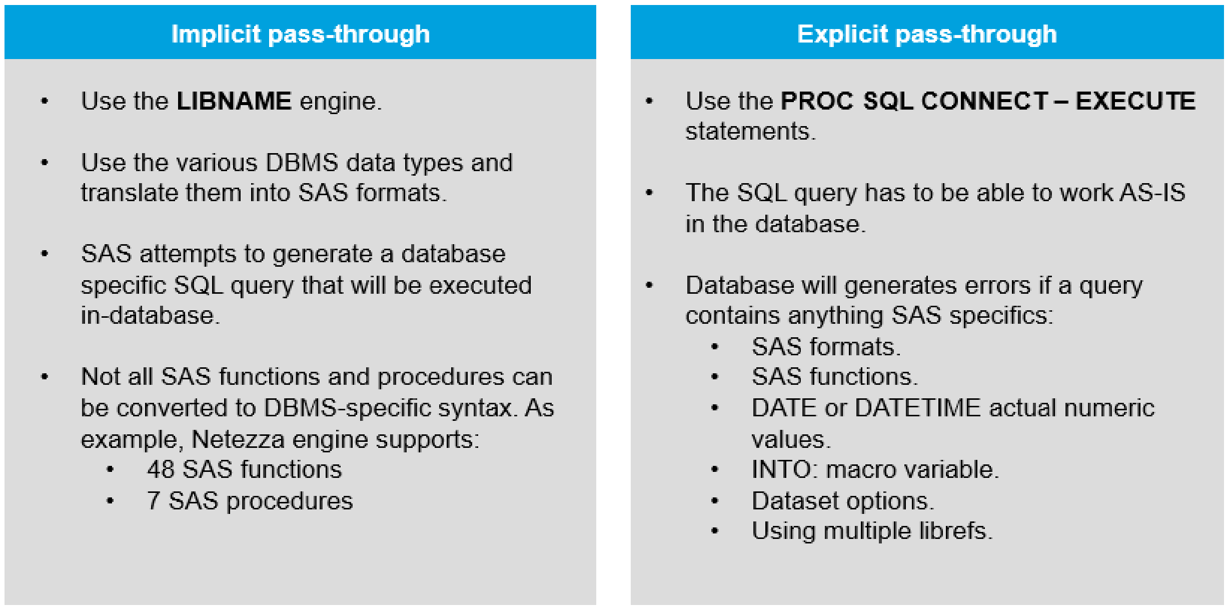
Explicit pass-through allows SAS/ACCESS® to send SQL queries directly to the third-party database for processing. This is done by writing PROC SQL CONNECT, EXECUTE statements:

```
PROC SQL;
   CONNECT to ENGINE (<connection-options>);
   EXECUTE ( query ) BY ENGINE;
   DISCONNECT FROM ENGINE;
QUIT;
```

IMPLICIT VS EXPLICIT PASS-THROUGH

Implicit pass-through is the key component of a migration; it will automatically convert what has already been done into a database specific SQL statement. However not all SAS® functions and procedures can be converted. You must check the list of supported function and procedures related to your SAS/ACCESS® engine.

Explicit pass-through is not interpreted by SAS® and will be directly sent to the third-party database for execution. Therefore you cannot use any SAS® specifics statements in your code. Figure 2 summarizes existing differences between the two pass-through methods.



Implicit pass-through

- Use the **LIBNAME** engine.
- Use the various DBMS data types and translate them into SAS formats.
- SAS attempts to generate a database specific SQL query that will be executed in-database.
- Not all SAS functions and procedures can be converted to DBMS-specific syntax. As example, Netezza engine supports:
  - 48 SAS functions
  - 7 SAS procedures

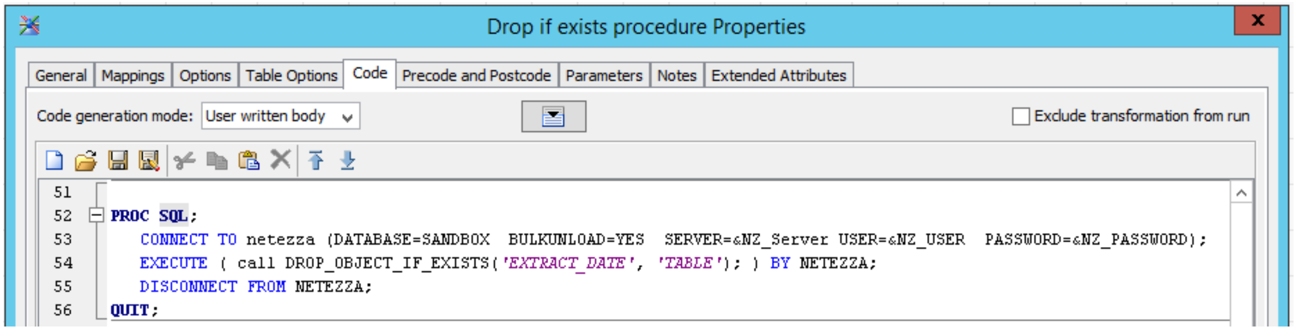
Explicit pass-through

- Use the **PROC SQL CONNECT – EXECUTE** statements.
- The SQL query has to be able to work AS-IS in the database.
- Database will generates errors if a query contains anything SAS specifics:
  - SAS formats.
  - SAS functions.
  - DATE or DATETIME actual numeric values.
  - INTO: macro variable.
  - Dataset options.
  - Using multiple librefs.

Figure 2 Summary of differences between implicit and explicit pass-through

Explicit pass-through example – Stored Procedure

As shown on Figure 3, a useful use case of coding a SQL explicit pass-through is when you need to call a stored procedure.



Code generation mode: User written body

Exclude transformation from run

```
PROC SQL;
   CONNECT TO netezza (DATABASE=SANDBOX  BULKUNLOAD=YES  SERVER=&NZ_Server USER=&NZ_USER  PASSWORD=&NZ_PASSWORD);
   EXECUTE ( call DROP_OBJECT_IF_EXISTS('EXTRACT_DATE', 'TABLE'); ) BY NETEZZA;
   DISCONNECT FROM NETEZZA;
QUIT;
```

Figure 3 Example of a stored procedure call

LIBNAME OPTIONS

Here is an example of PureData for Analytics libname statement:

```
LIBNAME nz NETEZZA DATABASE='SANDBOX' SERVER='XX.XX.XX.XXX' SCHEMA='admin'
USER=admin PASSWORD="XXXXX";
```

Besides, we will see in the following sections some useful libname options which will be applied to each tables registered in this library.

SQL functions

By default, the implicit pass-through is converting a subset SAS® functions into a database specific SQL statement. Setting the libname option `SQL_FUNCTION=ALL` allows you to use all existing functions.

However, pay attention as this extended set of functions can work differently than expected. This is especially true for every date and time functions that could be handled differently depending of your third-party database.

Unload data

By setting the libname option `BULKUNLOAD=YES` you will leverage the external table facility which is the fastest way to unload data from PureData for Analytics.

Although, you want to process tasks in-database as much as possible, some of them will request to unload data to the SAS® server. Nevertheless, this option will optimize the unloading speed performances.

Execute in-database

`DIRECT_SQL=YES` libname option is pushing in-database processing all the generated SQL from PROC SQL statements that is the default behavior of the Netezza engine. Table 1 is showing the different values that can be passed to this libname option to prevent in-database processing. Therefore, the in-database execution of the generated SQL code can be fine-tuned depending of the situation.

NO or NOGENSQL	Generated SQL from PROC SQL is not passed for processing.
NONE	Generated SQL from PROC SQL, SAS functions, joins and WHERE is not passed for processing.
NOWHERE	WHERE clauses from SAS data step, generated and explicit PROC SQL are not passed for in-database processing
NOFUNCTIONS	SQL containing functions are not passed.
NOMULTOUTJOINS	Multiple outer joins are not passed; other type of joins are still passed.

Table 1 DIRECT_SQL options

DATA STEP AND PROCEDURE OPTIONS

SAS/ACCESS® also allows users to specify options to be applied to specific tables only. The scope of these options is local to the table unlike to the libname ones having a global scope.

As shown on Table 2, local options are specified either in a SAS® data step or in a procedure.

Data Step	`DATA tablename(<options>);`
	`   SET tablename;`
	`RUN;`

PROC SQL	`PROC SQL;`
	`   CREATE TABLE tablename (<options>) AS`
	`   SELECT *`
	`      FROM tablename;`
	`QUIT;`

Table 2 Usage of SAS® options in a data step or procedure

Loading data

Similarly to the bulkunload libname option, the dataset option `BULKLOAD=YES` leverages the bulkload facility by using external table to load data into the database at the fastest throughput allowed by your network.

The next sections will present the following other useful options to manage the loading and unloading behavior. Output 1 shows how the loading options are used during the in-database processing.

```
data nz.bulkload_nz_table(BULKLOAD=yes BL_USE_PIPE=NO BL_DELIMITER=',' 
                                                  BL_OPTIONS="maxerrors 100");
   set sasdata.input_table;
run;
```

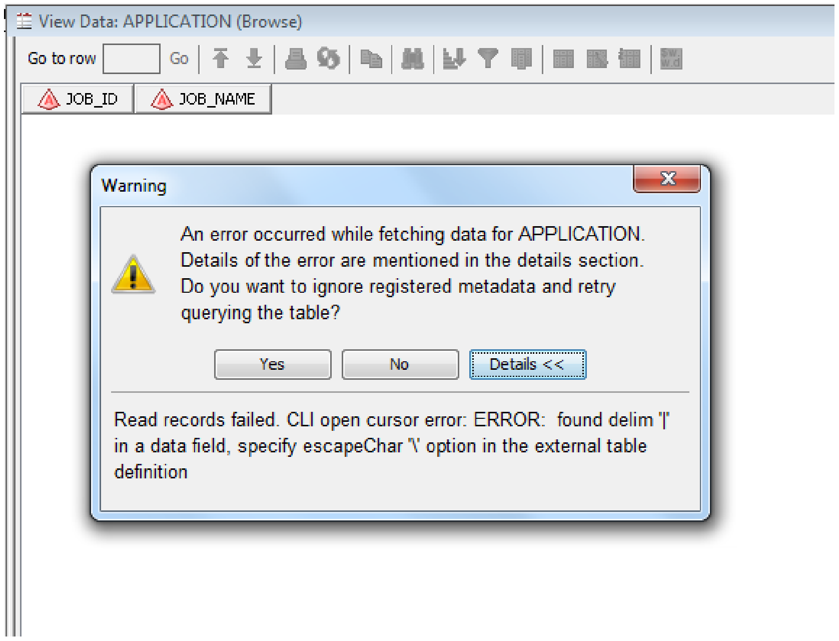
```
NETEZZA: COMMIT performed on connection 2.
NOTE: There were 366 observations read from the data set
SASDATA.input_table.
NOTE: The data set nz.BULKLOAD_NZ_TABLE has 366 observations and 73
variables.

NETEZZA_3: Executed: on connection 2
INSERT INTO SANDBOX.ADMIN.BULKLOAD_NZ_TABLE SELECT * FROM  EXTERNAL
'BL_INPUT_TABLE_8040E521-ACB6-40C4-96C2-A0715672581B.dat' USING ( DELIMITER
',' REMOTESOURCE 'ODBC'  NULLVALUE '' maxerrors 100)
```

Output 1 In-database generated code

Column delimiter

The default delimiter used by the bulk(un)load facility differs between third-party databases. In the case of PureData for Analytics datasource the default delimiter used is the pipe symbol ("|").

As illustrated by Figure 4, an error occurs when your data contains the default character delimiter. To avoid this error, you will need to override it by using the dataset option `BL_DELIMITER='<char_delimiter>'`.



An error occurred while fetching data for APPLICATION.
Details of the error are mentioned in the details section.
Do you want to ignore registered metadata and retry querying the table?

Read records failed. CLI open cursor error: ERROR: found delim '|' in a data field, specify escapeChar '\' option in the external table definition

Figure 4 Error due to of the BL_DELIMITER default value in SAS® Data Integration Studio

Pipe usage

By default, bulk(un)load facilities are using a named pipe to transfer data between SAS® and the third-party database. If your dataset can contain null values then you need to use the `BL_USE_PIPE=NO` option to avoid getting an error due to consecutives delimiter.

This option will force SAS/ACCESS® to use flat file to transfer data instead of a named pipe.

Loading options

The BL_OPTIONS= allows you to specify specific loading of options used by the third-party database. It's used as the following: BL_OPTIONS='option <…, option> '.

Table 3 shows some of the most used loading of options for the PureData for Analytics appliance.

Option	SQL
AllowReplay	MAX_QUERY_RESTARTS
DateDelim	DATEDELIM
DateStyle	DATESTYLE
DecimalDelim	DECIMALDELIM
Encoding	ENCODING
MaxErrors	MAXERRORS
MaxRows	MAXROWS

TimeDelim	TIMEDELIM
TimeStyle	TIMESTYLE

Table 3 Example of some useful loading parameters

Create table options

When creating tables via SAS/ACCESS®, some database specific options can be added to the generated create table statement. This is done by using the option `DBCREATE_TABLE_OPTS='<options>'`.

For example, PDA is based on a MPP architecture, so to avoid data skew, while creating a table, it is essential to specify a distribution key. Table 4 shows how to use it.

Data Step	`DATA nz.USER (BULKLOAD=YES DBCREATE_TABLE_OPTS='DISTRIBUTE ON (AGE)');`
	`   SET sasdata.USER;`
	`RUN;`

PROC SQL	`PROC SQL;`
	`   create table nz.USER (BULKLOAD=YES`
	`      DBCREATE_TABLE_OPTS='DISTRIBUTE ON (AGE)') as`
	`      (select * from sasdata.USER);`
	`QUIT;`

Table 4 Distribution key passed to PDA via a SAS® option

Additionally, a specific SAS/ACCESS® for Netezza option exists to specify directly the distribution key of a table: `DISTRIBUTE_ON='<key1, key2/ random>'`.

Handling null values

PDA is handling NULL values differently than SAS®. Indeed, for PDA two NULL values do not have the same internal representation and are therefore not equal. This behavior becomes an issue when joining tables having NULL values.

SAS/ACCESS® is providing two options to handles missing values: `NULLCHAR=` and `NULLCHARVAL=`. By setting `NULLCHAR=NO` it will force NULL values stored in SAS® datasets to be treated as a character or string specified in the option `NULLCHARVAL='<character>'`.

SYSTEM OPTIONS

SASTRACE and SASTRACELOC are very useful system options which enable a user to see what is really happening on the third-party database regarding the execution but also the submitted generated code.

```
            OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
```

Output 2 shows an example of how an error can be interpreted by using the output of sastrace.

```
Netezza_20: Executed on connection 4
CREATE EXTERNAL TABLE '/tmp/tablename' using(DELIMITER '|' REMOTESOURCE
'ODBC' NULLVALUE '') AS SELECT
…
ERROR: CLI open cursor error: ERROR: found delim '|' in a data field,
specify escapeChar '\' option in the external table definition.
```

Output 2 SASLOG output generated by the SASTRACE system option

Explanation of the log: in-database processing cannot be executed, so an external table is created to download data on the SAS® server. The default delimiter (pipe) is used but this character is found in one of table columns.

To fix this use, if it is not possible to modify the code to push it for in-database processing, then data will have to be unloaded properly by using the bl_delimiter option to specify another character delimiter that does not exist in your data.

LEVERAGING SAS/ACCESS® IN SAS® DATA INTEGRATION STUDIO

SAS® Data Integration Studio is a visual design tool for building, implementing and managing data integration processes regardless of data sources, applications, or platforms[2].

In other words, it allows you to create ELT (or ETL) processes without having to write SAS® base code (or a minimal amount). This software will automatically generate specific database SQL statement based on the table library used at each step leveraging SAS/ACCESS® capabilities.

All the options seen in the previous sections are re-usable in this tool. This section focuses on how to set-up and leverage the best performances of SAS/ACCESS®.

BEST PRACTICES IMPLEMENTATION

As a reminder, best practice is to create ELT data flow by making sure that each step will be executed in-database. SAS® Data Integration Studio has a useful button to perform this check (Figure 5).
A little “N” (standing for Netezza) marks transformations that will be executed in-database on the upper right corner.

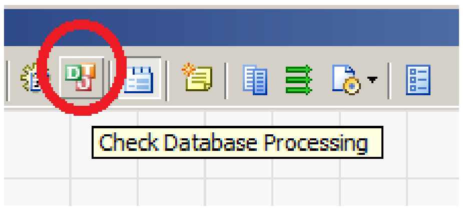


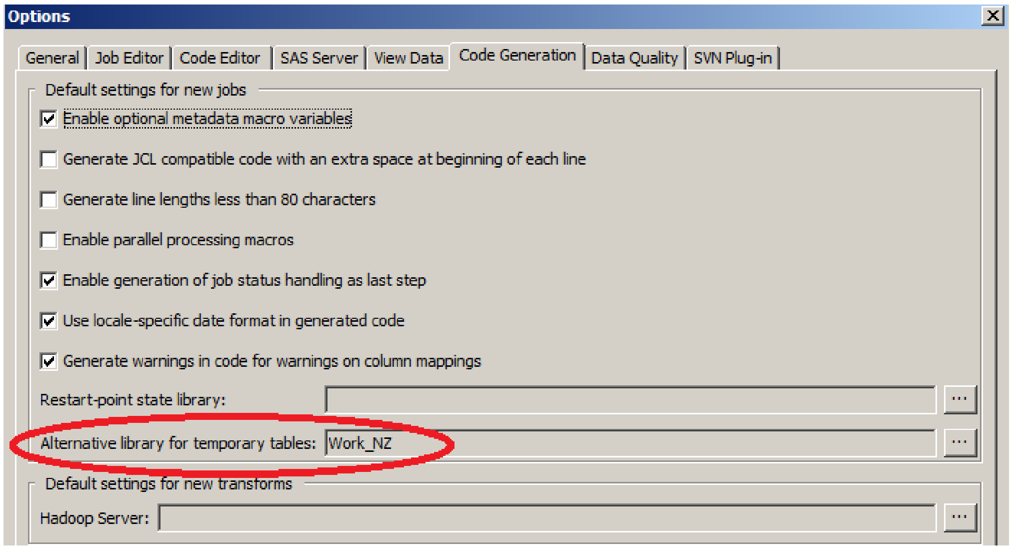
Figure 5 In-database processing check

The remainder of this section will focus on how to implement the ELT best practices with SAS® Data Integration Studio:

1. Redirecting the SAS® work to a third-party database library. The global option can be setup under **Tools > Options > Code Generation** tab and redirect the SAS® work to a third-party database

[2] http://support.sas.com/software/products/etls/

library (Figure 6).


Options

General | Job Editor | Code Editor | SAS Server | View Data | Code Generation | Data Quality | SVN Plug-in

Default settings for new jobs

- [✓] Enable optional metadata macro variables
- [ ] Generate JCL compatible code with an extra space at beginning of each line
- [ ] Generate line lengths less than 80 characters
- [ ] Enable parallel processing macros
- [✓] Enable generation of job status handling as last step
- [✓] Use locale-specific date format in generated code
- [✓] Generate warnings in code for warnings on column mappings

Restart-point state library:

Alternative library for temporary tables: Work_NZ

Default settings for new transforms

Hadoop Server:

Figure 6 Redirect the SAS® work

This redirection of the default SAS® work to a third-party database library will affect every transformation added to jobs. Intermediary results will be stored in an automatically created third-party table or view.

2. Leverage the bulkload capabilities of your third-party database. Do not perform single update or insert but instead delete all the rows that you want to update and append them using bulkload. The table loader transformation holds the little "N" showing that it will be executed in-database (Figure 7).

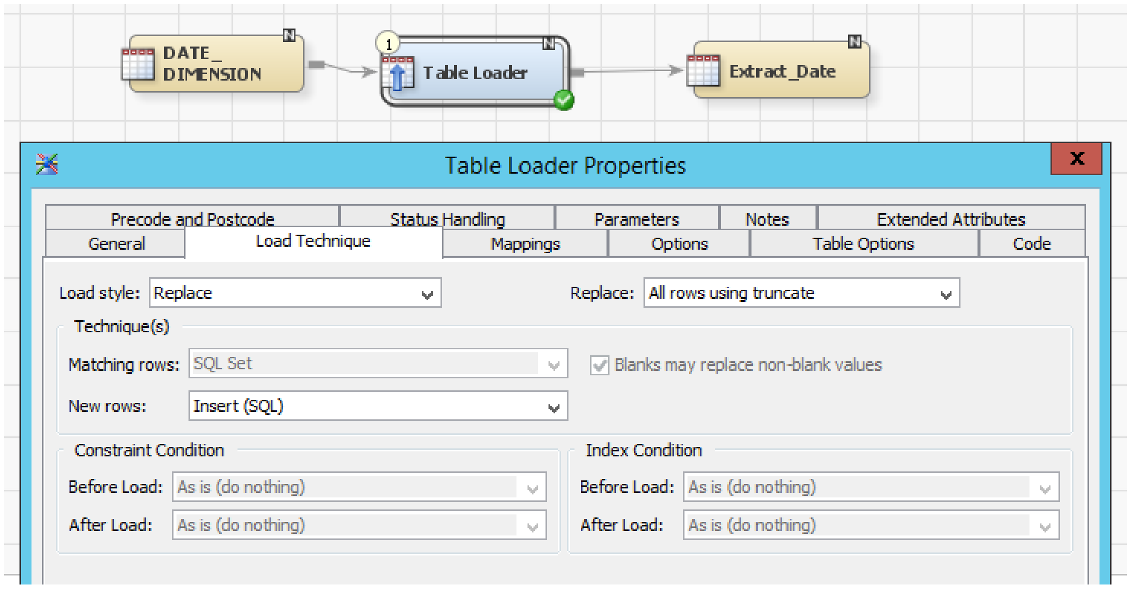


Table Loader Properties

Precode and Postcode | Status Handling | Parameters | Notes | Extended Attributes
General | Load Technique | Mappings | Options | Table Options | Code

Load style: Replace
Replace: All rows using truncate

Technique(s)

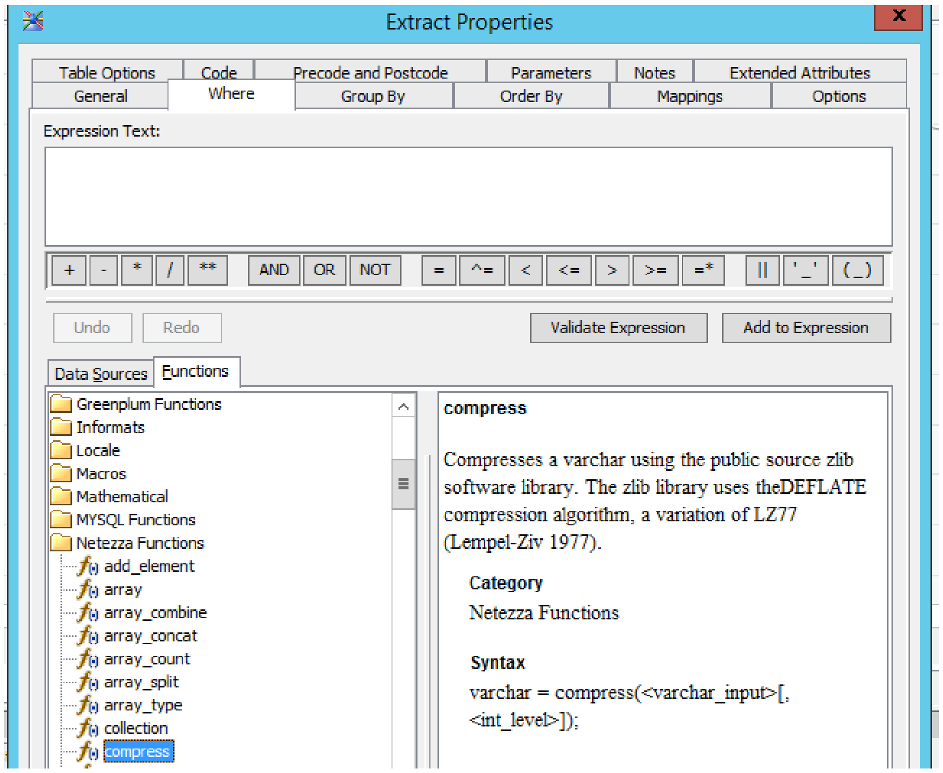
Matching rows: SQL Set
[✓] Blanks may replace non-blank values
New rows: Insert (SQL)

Constraint Condition
Before Load: As is (do nothing)
After Load: As is (do nothing)

Index Condition
Before Load: As is (do nothing)
After Load: As is (do nothing)

Figure 7 Implement bulkload instead of single update or insert

3. To push in-database processing the SQL generated code, use database specific SQL functions in the transformation as shown at Figure 8.



compress

Compresses a varchar using the public source zlib software library. The zlib library uses theDEFLATE compression algorithm, a variation of LZ77 (Lempel-Ziv 1977).

Category

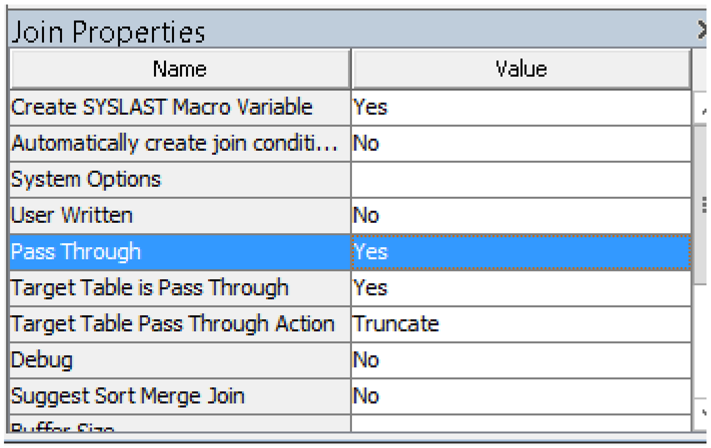
Netezza Functions

Syntax

varchar = compress(<varchar_input>[, <int_level>]);

Figure 8 select the database specifics functions

4. When joining table make sure to push SQL join in database processing by setting up the option in: **Join Properties > Pass Through > Yes** (Figure 9**).** As the generated SQL code will be directly send to the third-party database, pay attention not to use any SAS® specific functions in the expression builder.



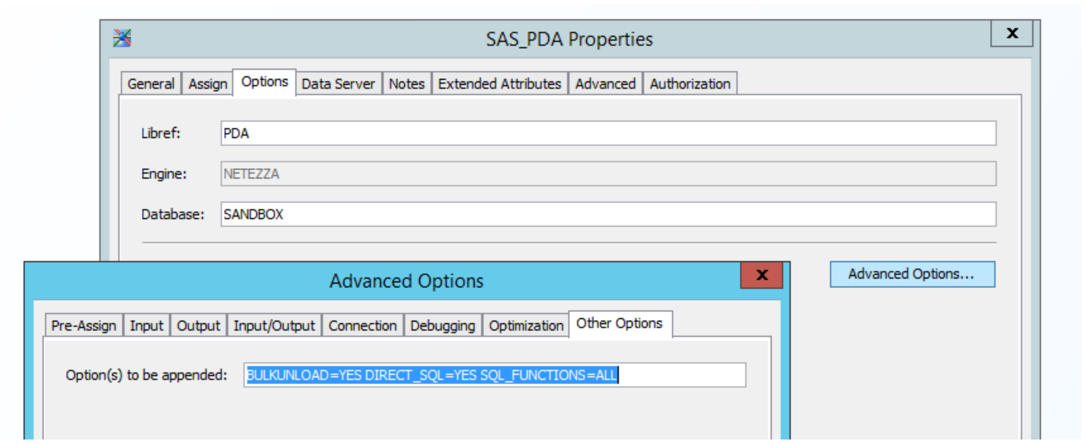
Join Properties

Name	Value
Create SYSLAST Macro Variable	Yes
Automatically create join conditi…	No
System Options	
User Written	No
Pass Through	Yes
Target Table is Pass Through	Yes
Target Table Pass Through Action	Truncate
Debug	No
Suggest Sort Merge Join	No
Buffer Size	

Figure 9 In-database processing of SQL join

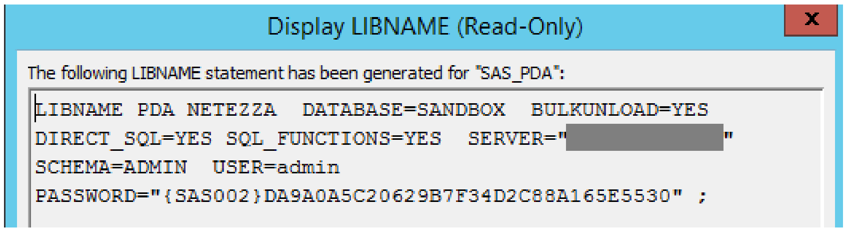
LIBNAME OPTIONS

The libname options explained at previous sections can be setup under libname properties: **Options > Advanced Options> Other Options > Option(s) to appended (**Figure 10**).** Resulting libname statement can be seen with a right click on the libname **> Display LIBNAME** (Figure 11).



Libref: PDA
Engine: NETEZZA
Database: SANDBOX
Option(s) to be appended: BULKUNLOAD=YES DIRECT_SQL=YES SQL_FUNCTIONS=ALL

Figure 10 Setting up the libname options



The following LIBNAME statement has been generated for "SAS_PDA":

```
LIBNAME PDA NETEZZA  DATABASE=SANDBOX  BULKUNLOAD=YES
DIRECT_SQL=YES SQL_FUNCTIONS=YES  SERVER="[redacted]"
SCHEMA=ADMIN  USER=admin
PASSWORD="{SAS002}DA9A0A5C20629B7F34D2C88A165E5530" ;
```

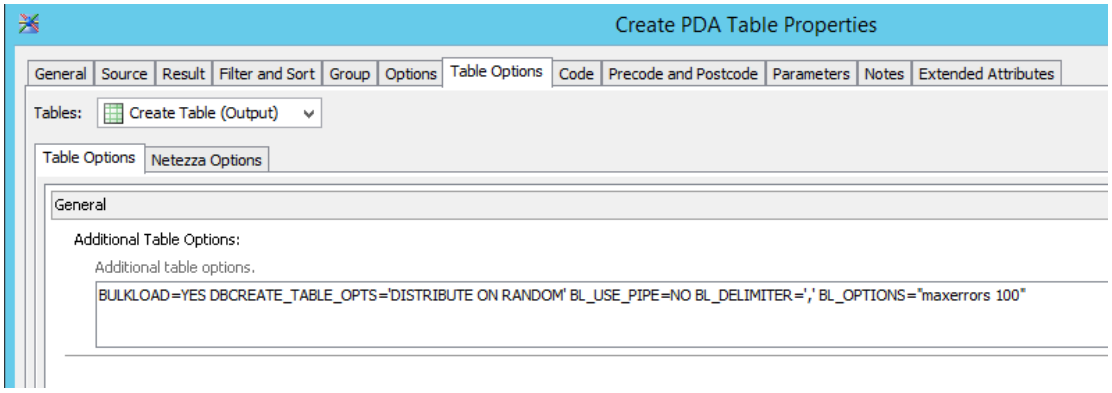
Figure 11 generated libname statement

DATA STEP AND PROC SQL OPTIONS

The SQL transformation named “Create Table” can be used to create a third-party database table. No database specific options will be send to the database. Therefore, the following option has to be set to “No”: **Options > Database pass-through > SQL pass-through**.

Figure 12 shows an example of how the proc sql options can be used to create and load data in a table. These options are located on **Table Options > Select the output table > General**. Figure 13 shows the code automatically generated by SAS® and the translated code sent for execution to the third-party database.

Note that, alternatively, the bulkload option can be set to “Yes” in: **Options> General > Bulkload**.



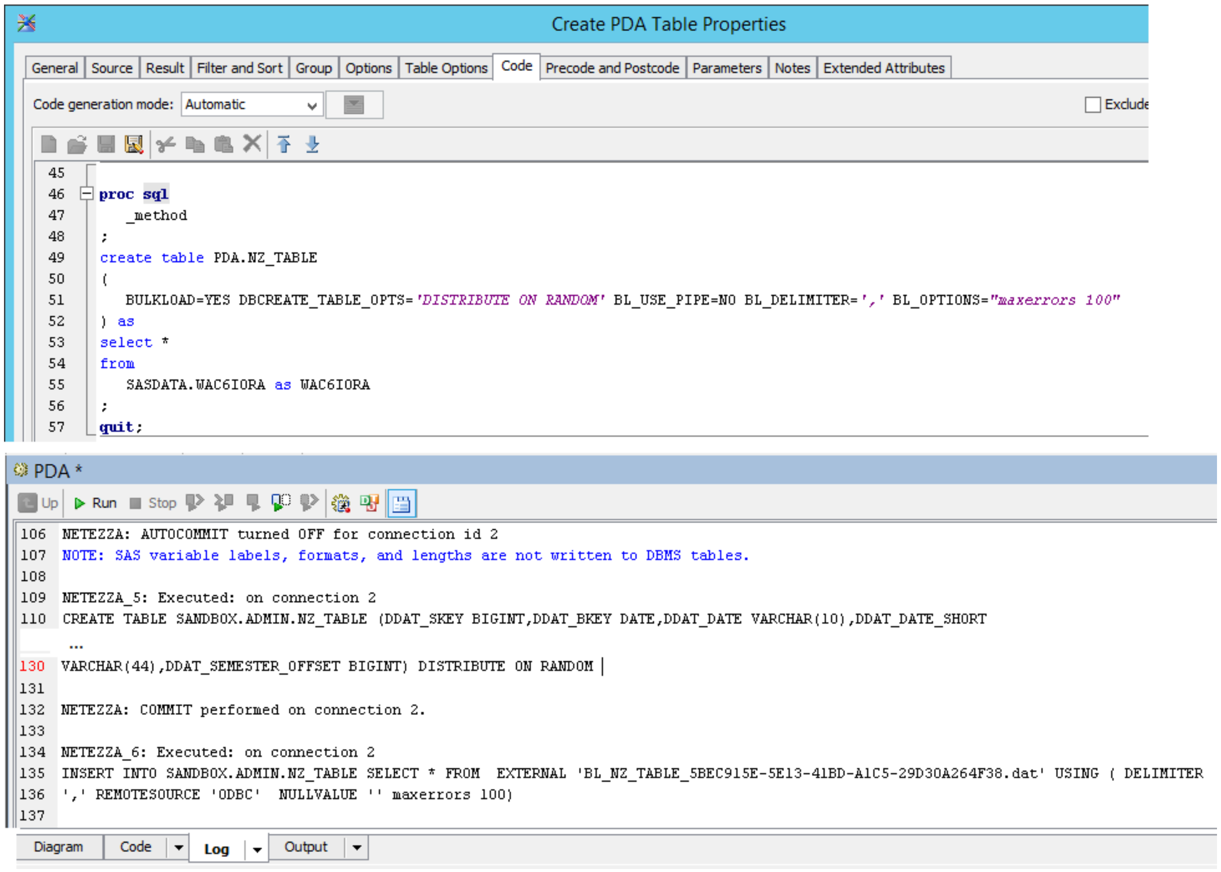
Tables: Create Table (Output)

Additional Table Options:

Additional table options.

BULKLOAD=YES DBCREATE_TABLE_OPTS='DISTRIBUTE ON RANDOM' BL_USE_PIPE=NO BL_DELIMITER=',' BL_OPTIONS="maxerrors 100"

Figure 12 Create table options


Create PDA Table Properties
General | Source | Result | Filter and Sort | Group | Options | Table Options | Code | Precode and Postcode | Parameters | Notes | Extended Attributes
Code generation mode: Automatic
Exclude

```
proc sql
   _method
;
create table PDA.NZ_TABLE
(
   BULKLOAD=YES DBCREATE_TABLE_OPTS='DISTRIBUTE ON RANDOM' BL_USE_PIPE=NO BL_DELIMITER=',' BL_OPTIONS="maxerrors 100"
) as
select *
from
   SASDATA.WAC6IORA as WAC6IORA
;
quit;
```

```
NETEZZA: AUTOCOMMIT turned OFF for connection id 2
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.

NETEZZA_5: Executed: on connection 2
CREATE TABLE SANDBOX.ADMIN.NZ_TABLE (DDAT_SKEY BIGINT,DDAT_BKEY DATE,DDAT_DATE VARCHAR(10),DDAT_DATE_SHORT
 ...
VARCHAR(44),DDAT_SEMESTER_OFFSET BIGINT) DISTRIBUTE ON RANDOM

NETEZZA: COMMIT performed on connection 2.

NETEZZA_6: Executed: on connection 2
INSERT INTO SANDBOX.ADMIN.NZ_TABLE SELECT * FROM  EXTERNAL 'BL_NZ_TABLE_5BEC915E-5E13-41BD-A1C5-29D30A264F38.dat' USING ( DELIMITER
',' REMOTESOURCE 'ODBC'  NULLVALUE '' maxerrors 100)
```

Figure 13 Generated creation table code

Similarly, the “Table Loader” transformation can be parametrized with loading options in**: Options > Loader > Additional data table options**. As seen previously, no database specific options will be sent, so **Options > Additional Options > Use the optimized pass-through facility for SQL statements** has to be set to “No”.

SYSTEM OPTIONS

System options can be specified on global scope for a job in job **Properties > Options > General** (Figure 14).

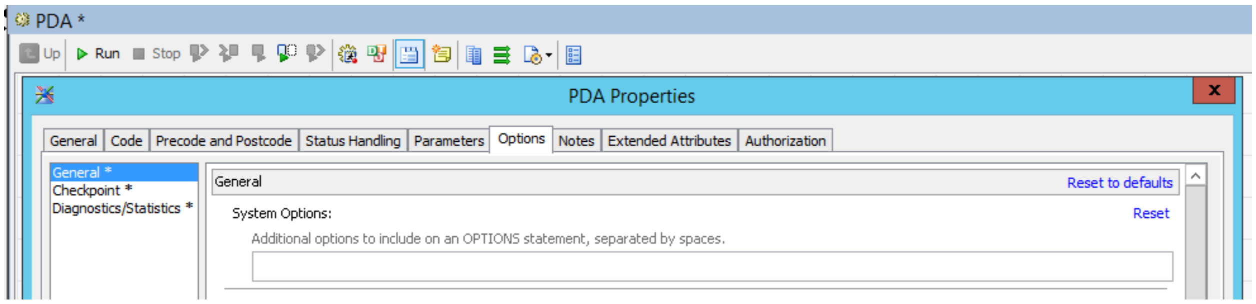
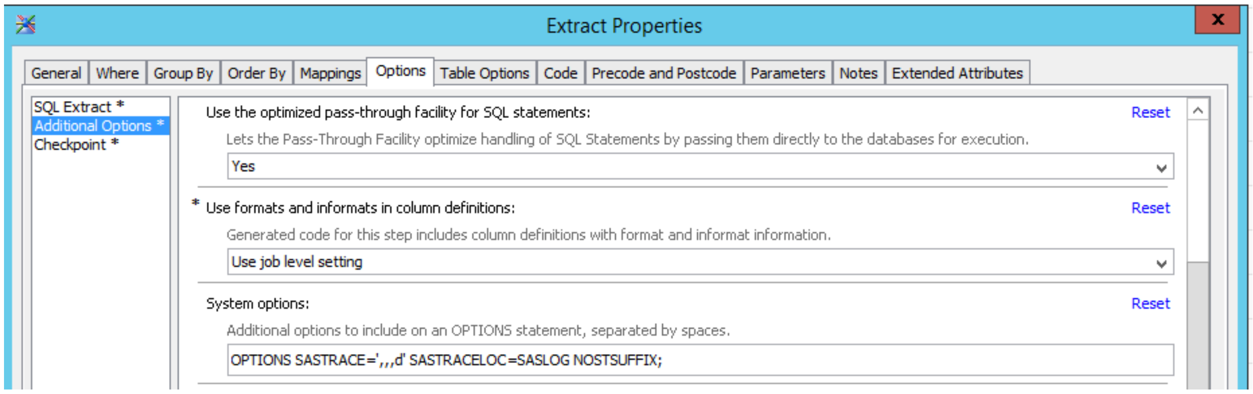

PDA *
Up | Run | Stop
PDA Properties
General | Code | Precode and Postcode | Status Handling | Parameters | Options | Notes | Extended Attributes | Authorization
General *
Checkpoint *
Diagnostics/Statistics *
General
Reset to defaults
System Options:
Reset
Additional options to include on an OPTIONS statement, separated by spaces.

Figure 14 System options applicable to a job

It can also be specified for a transformation in transformation **Properties > Options > Additional Options > System options** (Figure 15).



Use the optimized pass-through facility for SQL statements:

Lets the Pass-Through Facility optimize handling of SQL Statements by passing them directly to the databases for execution.

Yes

* Use formats and informats in column definitions:

Generated code for this step includes column definitions with format and informat information.

Use job level setting

System options:

Additional options to include on an OPTIONS statement, separated by spaces.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
```

Figure 15 System options applicable to a transformation

CONCLUSION

SAS/ACCESS® is providing an automatic translation of the SAS® code. Therefore, SAS® users can rely on their knowledge and SAS® developments are back-end independent. Indeed, the only extra knowledge needed is a set option and how to combine them to fine-tune the in-database processing. Nevertheless, explicit pass-through SQL code can be sent to the third-party database.

REFERENCES

Gaurav K Agraval. 2012. “SAS® In-Database Capability – Smart Architecture.” *Proceedings of the SAS Global 2012 Conference*, Cary, NC: SAS Institute Inc.

SAS/ACCESS 9.4 for Relational Databases: Reference, Seventh Edition.

SAS In-Database Products: Administrator's Guide and SAS In-Database Products: User's Guide.

CONTACT INFORMATION

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

Thésias Magali
Deloitte Belgium
mthesias@deloitte.com
http://www2.deloitte.com/be/en.html