ABSTRACT
Does SAS® implicit SQL pass-through sometimes fail to meet your needs? Do you sometimes need to communicate directly with your Oracle® or DB2® database in that database's native language? Explicit SQL pass-through might be your solution.

The author briefly introduces syntax for explicit SQL pass-through queries before showing examples of specific situations when explicit pass-through queries solve problems when extracting data. The author discusses the relationship between processing location and processing speed. She also gives specific examples of how differences between Oracle, DB2, and SAS sometimes make it necessary to do the initial extraction or transformation of data via pass-through. The examples used to illustrate the differences between the RDBMS and SAS include numeric precision and naming conventions. A brief discussion of differences in SQL dialects and functions between systems is also included.

INTRODUCTION
This paper assumes the reader has access to the following products: Base SAS® and SAS/Access®, which allows SAS to communicate with over 60 different types of data sources outside of SAS (see SAS/Access documentation for a full listing). Several terms related to database connectivity are introduced and defined in this section prior to mentioning them in the body of the paper.

A Relational Database Management System (RDBMS) is a collection of normalized data sets where the tables are linked on key fields to store and retrieve data. Common examples include IBM® DB2®, Microsoft® SQL Server®, MySQL™, Sybase®, IBM® Informix®, Teradata®, Oracle®, and Microsoft® Access® databases. Each RDBMS has its own native dialect of Structured Query Language (SQL). SQL can also be used within SAS by invoking the SQL Procedure.

Open Database Connectivity (ODBC), which was designed to access relational databases using SQL, allows the user to name and configure connections to various data sources using an appropriate driver and then use the data source name to link to tables from within another database system (see screenshot below for an example of where to configure ODBC connections). MS Access, for example, can use ODBC connections to link to SQL Server tables. SAS can also use the ODBC engine to connect to tables in a variety of other types of databases. A SAS user can reference the ODBC connection in a LIBNAME statement to connect to previously configured data sources. More recently, SAS has added an interface to Object Linking and Embedding Database (OLEDB), a successor to ODBC which offers better performance and connectivity to a wider range of data sources. OLEDB is less established than ODBC and outside of the scope of this paper, which focuses on relational data sources (see technote http://support.sas.com/techsup/technote/ts700.pdf and Microsoft’s documentation for more information about OLEDB as well as ftp://ftp.sas.com/techsup/download/v8papers/odbcdb.pdf for a brief comparison of ODBC and OLEDB). SAS/Access also includes direct interfaces to Oracle and DB2 databases, making it possible to connect directly to these types of databases by specifying the path and using the Oracle and DB2 engines instead of the ODBC (or OLEDB) engine.

As an alternative to the LIBNAME statement, a user can elect to use the SAS/Access interface to ODBC, Oracle, or DB2 with the SQL Pass-Through facility to directly access data in an RDBMS. A pass-through query processes data in its native environment using native SQL functions instead of having SAS do the work. The remainder of this paper focuses on how, when, and why to use explicit SQL pass-through syntax to access data that is housed in Oracle and DB2 databases.
Using a LIBNAME statement to reference a relational database is also sometimes referred to as implicit pass-through, as SAS will try to optimize a query to an RDBMS by automatically passing as much of the code as possible to the RDBMS for processing, even translating certain non-SQL statements to SQL. Enhancements have been made in SAS 9.2 which make implicit pass-through more powerful, but still with limitations which prevent some of the more complex statements from being passed. (For a discussion of these enhancements and limitations with an emphasis on Teradata, see http://support.sas.com/resources/papers/proceedings11/105-2011.pdf as well as the SAS/Access documentation and Levine’s paper on implicit pass-through: http://www2.sas.com/proceedings/sugi26/p110-26.pdf.) Also note that using a SAS function that SAS cannot convert will also cause the query to be processed in SAS rather than in the RDBMS, although SAS keeps expanding the list of functions that it can convert.

Use the SAS system option SASTRACE (PC users must use SASTRACELOC with SASTRACE to show trace results in the log) to determine which statements have been passed to the RDBMS for processing. The following statement with SASTRACE set to ‘”,d” ensures that SQL SELECT, INSERT, UPDATE, CREATE, DROP, and DELETE statements sent to the Microsoft Jet engine are identified and printed to the log.

**OPTIONS SASTRACE=’”,d’ SASTRACELOC=SASLOG NOSTSUFFIX;**

SAS macro variables SQLXMSG and SQLXRC store error codes and messages generated by the Jet engine when using the pass-through facility (for more about these system options and macro variables, refer to the SAS/Access documentation).
LIBNAME STATEMENT (IMPLICIT) VS. EXPLICIT PASS-THROUGH SYNTAX

This section uses a simple query to illustrate the differences in syntax between accessing data with a LIBNAME statement and accessing via explicit SQL pass-through.

Fig 1. Query with Libname Statement
Libname CDB odbc datasrc=db2p user=uid password=pwd schema=GWY1;
Proc sql;
Create table work.account as
Select *
From CDB.account;
Quit;

Fig 2. Pass-Through Query Syntax for ODBC Interface
Proc sql;
Connect to odbc(datasrc="db2p" user=uid password=pwd);
Create table work.account as
Select * from connection to odbc
(Select *
From GWY1.account);
Disconnect from odbc;
Quit;

Both examples shown above use ODBC as the connection interface/libname engine. However, another option is to use the SAS/Access interface to DB2 as shown in Figure 3 below:

Fig 3. Pass-Through Query Syntax for DB2 Interface
Proc sql;
Connect to db2(path="dbc08p.sys.cigna.com" user=uid password=pwd);
Create table work.account as
Select * from connection to db2
(Select *
From GWY1.account);
Disconnect from db2;
Quit;

The main difference is that the user specifies only the data source name with an ODBC connection, but uses the full path when connecting directly to DB2. SAS/Access offers direct interfaces to both DB2 and Oracle databases, among others, using the Pass-Through facility. Connecting with the DB2 or Oracle interfaces offers greater performance benefits than connecting via ODBC. However, SAS PC users who are accustomed to accessing data via ODBC connections and wish to continue to do so will still find greater efficiency with a pass-
WHY USE EXPLICIT SQL PASS-THROUGH?

The most simple and obvious answer is that processing location affects processing speed. That is, moving the data processing work as close as possible to the data storage location results in greater efficiency and minimizes I/O operations. Using explicit SQL pass-through ensures that SQL is passed to the RDBMS verbatim, rather than relying on implicit pass-through to translate code and pass portions through when it can. Using a LIBNAME statement with statements which SAS cannot pass through (such as outer joins or SAS functions which cannot be translated successfully to the RDBMS) does the processing work in SAS on the user’s PC (or a remote server if RSUBMIT is used) using SAS functions. In contrast, explicit pass-through queries do the processing work in the RDBMS where the data is stored using “native” database functions. This often avoids large data movement, since only the results are returned to SAS. Since the database is typically optimized and indexed to handle queries, multi-table, multi-field, complex joins are handled much faster with a pass-through query. Queries with complex joins are less likely to be successfully passed with implicit pass-through, necessitating explicit pass-through syntax. When entire tables have to be brought into SAS, the processing is slow, taking up lots of resources and space. When several large tables are combined with complex joins, SAS can be dramatically slower than the native database. However, this is not the only reason one might wish to use explicit pass-through. Sometimes differences in how data gets stored by SAS vs. the native database necessitate using explicit SQL pass-through in order to successfully extract data. Finally, a programmer might wish to use certain functions available in the RDBMS that are not supported by SAS. The following sections describe a few such situations.

DIFFERENCES IN NUMERIC PRECISION: ORACLE EXAMPLE

Sometimes the native database stores numeric data to a greater degree of precision than SAS is designed to handle. Oracle, for example, can store precision up to 38 digits, sometimes resulting in a loss of precision when converting numeric data to character data in SAS. One situation when this could happen is when trying to join two Oracle tables on the same 16-digit key field, but discovering that the key is numeric in one table but character in the other (critiques of database design are outside of the scope of this paper). In order to join the two tables, the data types must match. If the conversion from numeric to character (or vice versa) is performed within SAS using the put() function as shown in Figure 4, the conversion results in a loss of precision in the last digit, which then gets stored as a zero.

![Fig 4](image)

Original Query

Proc sql;
Create table work.s1_claim_row as
Select
Put(claim_uid, best16.) as claim_uid length=16
From prodj.s1_claim
Quit;

To solve this problem, the alternative in Figure 5 uses explicit pass-through syntax, which causes Oracle to do the conversion before returning the results to SAS. It is important to remember that the SAS SQL dialect is slightly different from Oracle SQL, so the put function which works in SAS does not work in the pass-through query since Oracle does not recognize it. Instead, the conversion is accomplished with the Oracle to_char() function. A SAS programmer who is also fluent in Oracle SQL can recognize the advantage of being able to code in the SQL dialect that is most appropriate at the moment.
Fig 5
Solution with Explicit Pass-Through (ODBC)

Proc sql;
Connect to odbc(datasrc=”PRODJ” user=uid password=pwd);
Create table work.s1_claim_row as
Select * from connection to odbc
    (select to_char(claim_uid,'0000000000000000') as claim_uid
    From oncsource_o.s1_claim);
Disconnect from odbc;
Quit;

Figure 6 shows the solution using the Oracle interface instead of ODBC.

Fig 6
Solution with Explicit Pass-Through (Oracle)

Proc sql;
Connect to oracle(path=”prodj.cigna.com” user=uid password=pwd);
Create table work.s1_claim_row as
Select * from connection to oracle
    (select to_char(claim_uid,'0000000000000000') as claim_uid
    From oncsource_o.s1_claim);
Disconnect from oracle;
Quit;

DIFFERENCES IN NAMING CONVENTIONS: DB2 EXAMPLE

Sometimes the issue is not related to numeric precision but to naming conventions. In this case, a SAS query results in an error message rather than apparently succeeding but returning unexpected results as in the examples above. Potential error messages include the following:

ERROR: This DBMS table or view cannot be accessed by the SAS System because it contains column names that are not unique when a SAS normalized (uppercased) compare is performed. See “Naming Conventions” in the SAS/Access documentation.

DB2 is a case sensitive database. It is possible to create column names within a table which are identical except for case differences (whether or not this is desirable again falls outside the scope of this paper). Even if one of these column names is not referenced directly in the query, if the object is referenced, the query will fail. Using an explicit pass-through query to access the data solves this problem.

Apart from issues of case, table/column name lengths and special characters can also be a problem, as shown in the following logged error message:

ERROR: TABLE NAME ‘INTERVENTION_TRACKING_INTERVENTION’ is too long for a SAS name in this context.
This error message occurs when a table name is longer than 32 characters, the maximum length allowed in SAS; DB2 table names, in comparison, allow up to 128 characters. One solution to this problem is to ask the DBA to create a view of the table with a shorter name in the DB2 database, which can then be referenced with no problem in SAS. Another solution uses pass-through query syntax to access the table.

**DIFFERENCES IN FUNCTIONS: SQL SERVER EXAMPLE**

In addition to the cases described above, sometimes a SAS programmer may wish to use explicit pass-through to take advantage of functions existing in the native RDBMS which are not available in SAS. For example, SAS supports the Rank Procedure which can be used to rank and partition data sets, including various options for dealing with ties (for more information about this procedure, see *Base SAS 9.2 Procedures Guide*). This is a very useful procedure; however, it has to be used in a separate step, and a programmer may wish to use a function within a Proc SQL statement to extract, join, rank, partition, sort, and otherwise transform the data in a single block of code. The Rank function in SAS, which takes a character argument and returns its position in the ASCII collation sequence, is neither equivalent to nor has the versatility or the range of ranking functions supported by some of the other database systems. For example, SQL Server includes ROW_NUMBER(), RANK(), DENSE_RANK(), and NTILE() functions with syntax as follows:

- `ROW_NUMBER() OVER ([<partition_by_clause>] <order_by_clause>)`
- `RANK() OVER ([<partition_by_clause>] <order_by_clause>)`
- `DENSE_RANK() OVER([<partition_by_clause>]<order_by_clause>)`
- `NTILE(integer_expression) OVER ([<partition_by_clause>] <order_by_clause>)`

The NTILE() function is shown below to order claims by paid dollar amounts and partition them into quartiles.

**Fig 8**

Explicit Pass-through with NTILE() function

```sql
Proc sql;
Connect to odbc(datasrc="SQL" user=uid password=pwd);
Select * from connection to odbc
  (Select claim_id,
   paid_amt,
   ntile(4) over (partition by claim_id order by paid_amt desc) as quartile
    From SQL.CLAIM);
```
Disconnect from odbc;

Since a descending sort order was used for this example, quartile 1 would have dollar amounts in the top 25%. Oracle calls these functions “analytic” functions and supports additional ranking functions to those available in SQL Server (for more information about Oracle analytic functions, see Oracle Database SQL Reference 10g Release 2 (10.2)).

CONCLUSIONS
Explicit pass-through can help a user decrease processing time with complicated queries, cope with differences between SAS and the native database, and work around SAS limitations. Moving processing operations closer to where the data is stored makes a query more efficient. Pass-through queries process the data in its native environment, requiring fewer I/O operations and less processing time. Sometimes differences between Oracle, DB2, and SAS such as numerical precision and naming conventions require explicit pass-through query syntax to achieve desired results (for more information, refer to SAS/Access documentation). For the SAS programmer who is more familiar with other SQL dialects, explicit pass-through may be particularly useful, allowing her to use a range of Oracle, SQL Server, or DB2 functions to do work that she cannot easily accomplish in SAS. Explicit SQL pass-through is also especially helpful for initial data extractions from large tables using complex joins on indexed fields.

REFERENCES:


Support.sas.com. Getting Started with SAS/ACCESS Interface to OLE DB.  


ACKNOWLEDGMENTS
SAS 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.
CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Jessica Hampton
CIGNA Corporation
900 Cottage Grove Rd
Bloomfield, CT 06002
Work Phone: (860) 226-1938
Email: Jessica.Hampton@cigna.com

********************************************************************************