

Using SAS/ACCESS- OLE DB for Pharmacoeconomic Analysis

Brandon Zagorski, MS, Walgreens Health Initiatives, Deerfield, IL

ABSTRACT

Pharmacoeconomic analyses are frequently replicated over time and may require data from dynamic and disparate file formats; and without an integrating software the automation of such analyses will be limited. SAS/Access-OLE DB permits the data integration from diverse file formats, such as, relational databases, flat files, and spreadsheets, without sacrificing performance. Using the SAS/Access-OLE DB Libname statement, data can be accessed in an automated fashion. Using the %put &sqlxmsg statement, the libname statement can be built and stored so that all subsequent connections will be automated.

INTRODUCTION

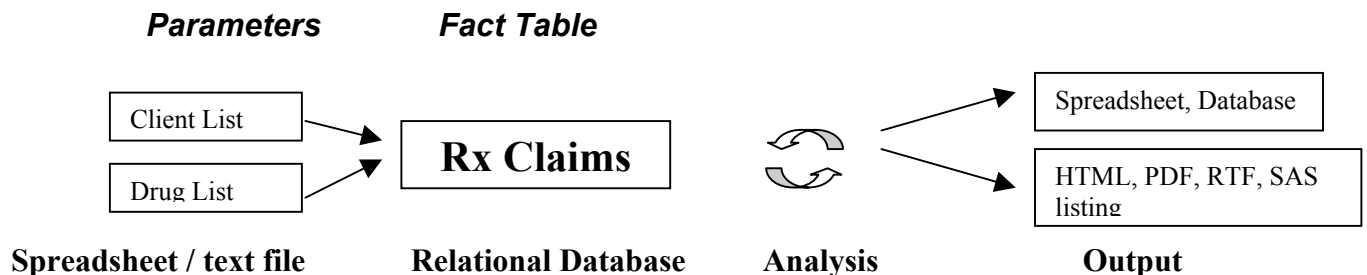
Microsoft OLE DB is an API (application programming interface) that provides access to data that can be in a database table, an email file, a text file, or another type of file. The SAS/Access OLE DB interface accesses data from these sources through OLE DB data providers such as Microsoft Access, Microsoft SQL Server, and Oracle.

Pharmacoeconomic analyses, such as cost-effectiveness, trend analysis, retrospective database analysis, may occur on an *ad hoc* or an ongoing basis involving several data sources in disparate file formats. At a Pharmacy Benefit Management company, pharmacy claims number in the millions and are often stored in a dynamic relational database (e.g. Oracle) and are ubiquitous in outcomes analyses. An effective time saving approach that enables parameter-driven queries (for example, client list, drug list, etc.) across many formats is needed.

However, many parameters that govern these analyses may be often retrieved via spreadsheets and text files. SAS/ACCESS OLE DB facilitates data integration between these file formats which allows for the entire analysis and output to be contained in one SAS code, a major advantage for organizations that need documented methodology for each analysis (Figure 1).

This paper describes the method of integration common for many types of analyses, not just pharmacoeconomics. For all methods described below, an in depth knowledge of the content and structure (e.g. table layout, indexes) of the databases is required to write optimal queries.

Figure 1. SAS/Access OLE DB Integrates Multiple File Formats in a single SAS session



USING THE LIBNAME STATEMENT

Setting up SAS libraries for the first time:

Using a prompted connection via OLE DB, you can capture the connection string in the SAS Log by using the following:

```
libname mylib oledb; %put &sysdbmsg;
```

Once the SAS library is established from this point-n-click method paste the string (init_string) that appears in the SAS log into the code, as follows:

Oracle:

```
libname oracle oledb init_string="Provider=MSDAORA.1;
Password=your_pw_here;
User ID=your_id_here;
Data Source=server_name;
Persist Security Info=True";
```

Microsoft Excel:

```
libname Excel oledb
provider="Microsoft.Jet.OLEDB.4.0"
preserve_tab_names=yes
preserve_col_names=yes
properties=('data source'="G:\yadda\yadda\name_of_file.xls")
provider_string="Excel 8.0" ;
```

Microsoft Access:

```
libname Access oledb init_string='
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=H:\yadda\yadda\name_of_file.mdb;
Persist Security Info=False;
Jet OLEDB:Database Password=XXXX';
```

Once the libraries are established with this method, queries can be automated and transfer between formats is possible.

USING THE LIBNAME STATEMENT;

Advantage: Program syntax code is exclusively in SAS

Disadvantage: Complex queries where joins between local SAS data sets and a large Oracle tables may result in long run times.

USING PROC SQL;

For larger parameter driven queries this is the preferred method. In this situation the server's native SQL is wrapped in Proc SQL, thus passing queries directly to Oracle and reducing run time:

```
proc sql;
connect to oledb(schema=pmbatch init_string="
Provider=MSDAORA.1;
Password= your_pw_here;
User ID= your_id_here;
Data Source=server_name;
Persist Security Info=True");
create table saslibname.sas_table as
select * from connection to oledb

(
Your PL/SQL Oracle query here
); quit;
```

Optimal run time works only when the tables are exclusively within the Oracle environment. When the user has read/write access writing a local SAS data set to a personal schema in Oracle will speed the process.

```
libname myschema schema=bxzagorski oledb init_string="Provider=MSDAORA.1;
Password=your_pw_here;
User ID=your_id_here;
Data Source=server_name;
Persist Security Info=True";

proc sql; drop table myschema.cost_savings;quit; /*delete existing table*/
data myschema.cost_savings; set excel.'Sheet$'n;run;
```

Caution: When uploading an excel table in this fashion, Oracle may add spaces by default at the end of character fields. In this case the trim(charvar1) is needed to delete spaces when using these fields for parameters in the PL/SQL query.

USING PROC SQL;

Advantage: Better run times

Disadvantage: Must write syntax in the native language of Server (e.g. PL/SQL) and left/right/inner join statement will not be recognized, SAS Macro parameters need a little extra syntax, and overwrite existing tables is not permitted.

CONTACT INFORMATION

I welcome any questions/comments, contact me at:

Brandon Zagorski, MS
Walgreens Health Initiatives
1417 Lake Cook Rd #L360
Chicago, IL 60015
847.964.4726
brandon.zagorski@walgreens.com
brandon@workingbikes.org

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.