Paper RF02-2014

SAS® and Relational Databases: What You Should Know Before You Code

Patricia Hettinger, Data Analyst-Consultant, Oakbrook Terrace, IL

ABSTRACT

It is a rare shop that has data in just SAS format. DB2, Teradata, Oracle, SQL Server - all of these relational databases have their own quirks that may not become apparent until your DBA calls. This paper will address what you must know before using SAS/Access for more efficient queries and consistent results. If you are experienced with SAS or SQL but not the two together, this paper is for you.

INTRODUCTION

The audience for this paper are those who have some experience with native SAS but limited experience with RDBMS databases and SAS/Access. This paper should also be helpful for people in the opposite situation, those having more database experience than SAS.

We will address the usual design differences between SAS datasets and RDBMS tables, coding considerations, the libname option versus SQL Pass-Through and some trouble-shooting hints. Although meant to be a generic discussion of RDBMS, there will be some database-specific examples which will be noted.

TERMS USED IN THIS PAPER

DBMS – General abbreviation for database management system. Includes hierarchical, relational, distributed and dimensional models.

RDBMS – relational database management system. A database system based upon the relational model introduced by E.F. Codd. Data are stored in tables as well as the relationships among the data.

DBA – database administrator. Someone with the authority to create, delete and update database objects like tables. Also sets up and enforces security rules.

SQL – Structured Query Language. The most widely used language for relational databases. RDBMS usually have extensions to the ANSI standard. SQL allows for queries, updates, structure creation and access determination.

ANSI - American National Standards Institute

SAS/Access – generic name for the SAS interface to a RDBMS. SAS has interfaces to several databases including Teradata, Oracle, DB2, SQL Server and PC files like Excel and Access.

Dataset – used here to refer to data in SAS format that can be directly used in SAS data steps and procedures.

Table – used here to refer to the basic data structure in an RDBMS.

ETL – extract, transform and load. The process by which data becomes stored in datasets and tables

Entity – a thing capable of a separate existence and can be uniquely identified. **Customer** is an example of an entity.

Primary key (PK) – a column or combination of columns that uniquely identifies an entity. For example, we may assign a numeric **Customer Id** to each customer.

Referential integrity – the notion that certain entities have mandatory relationships. For example, you wouldn't have an order without a customer.

Foreign key – an identifier in an entity that corresponds with the primary key of another entity. For example, **Customer Id** would be a foreign key in the **Order** table that would identify to whom the order belongs.

Entity-relationship diagram – a graphical rendering of the relationship between entities. The diagrams used in this paper use crows-feet notation.

DESIGN DIFFERENCES BETWEEN DATASETS AND TABLES

Datasets are known for the lack of methodology in creating and documenting them. This is partially due to the ease of creation. If you have enough disk space and write access to that space, you can create a SAS dataset. You can change the structure anytime you want. You might discover a lot of unhappy people whom you were unaware were using your dataset but you can do it.

Not so in an RDBMS. Most RDBMS of any size have a database administrator who controls what goes into the database, the access people have and the procedure to add new structures and elements. The DBA has several tools to monitor the system and see who is accessing what in detail. A more refined shop would have graphical data models so that users can see the relationships at a glance, source to target documents to detail the ETL involved, data element definitions and a central repository to store it all.
The terminology is different for a dataset versus a table also. Datasets have variables and observations. Tables have columns and rows. You will hear terms like 'primary key', 'foreign key', 'referential integrity' and 'index' much more with tables than with datasets.

The structure of the data you would query will probably be drastically different as well. It's not uncommon for datasets to have the information you need in just one or two sources. This is unlikely with an RBMS – you will probably have to connect many tables to get the same information.

For example, let's take San Vicente Center Point. SVCP is a charitable organization offering financial assistance like grocery or gas cards to those in need. The system was originally developed in SAS with data created by reading in Excel spreadsheets or text documents. There was even a rudimentary interface in SAS® SCL (Screen Control Language) for adding new clients on the fly. Customer information, assistance information and San Vicente case worker (Visitor) were all in one dataset. Each client got his/her own record. Up to three assistance requests per client were stored, along with the SVCP visitor who handled the request.

There were several reasons this was redesigned. The major reason was the present economic conditions had many people making more than three requests. More people asked for assistance as well, increasing the caseload for each visitor. Fortunately, SVCP has able to recruit more volunteers. However that caused more visitor information to be entered as well. Even worse, it was very easy to make mistakes with this information, causing the same visitor to be listed more than once in the reports. To illustrate, Gene Marcus was listed as Eugene Marcus IV, Gene Marcus IV and Gene Marcus.

Another reason for the rewrite was that the SVCP became part of a greater charitable group network. They wanted an easier way to share their information, perhaps going to a web-enabled system. It is much easier to do this with an RBMS than SAS. Not too many people are fluent in SCL anymore.

We decided to use codes for the request, outcome and assistance column to give more consistent values. A later model will include family and income information as well. Our query discussion will be limited to information stored in the original SAS dataset as shown in figure 1

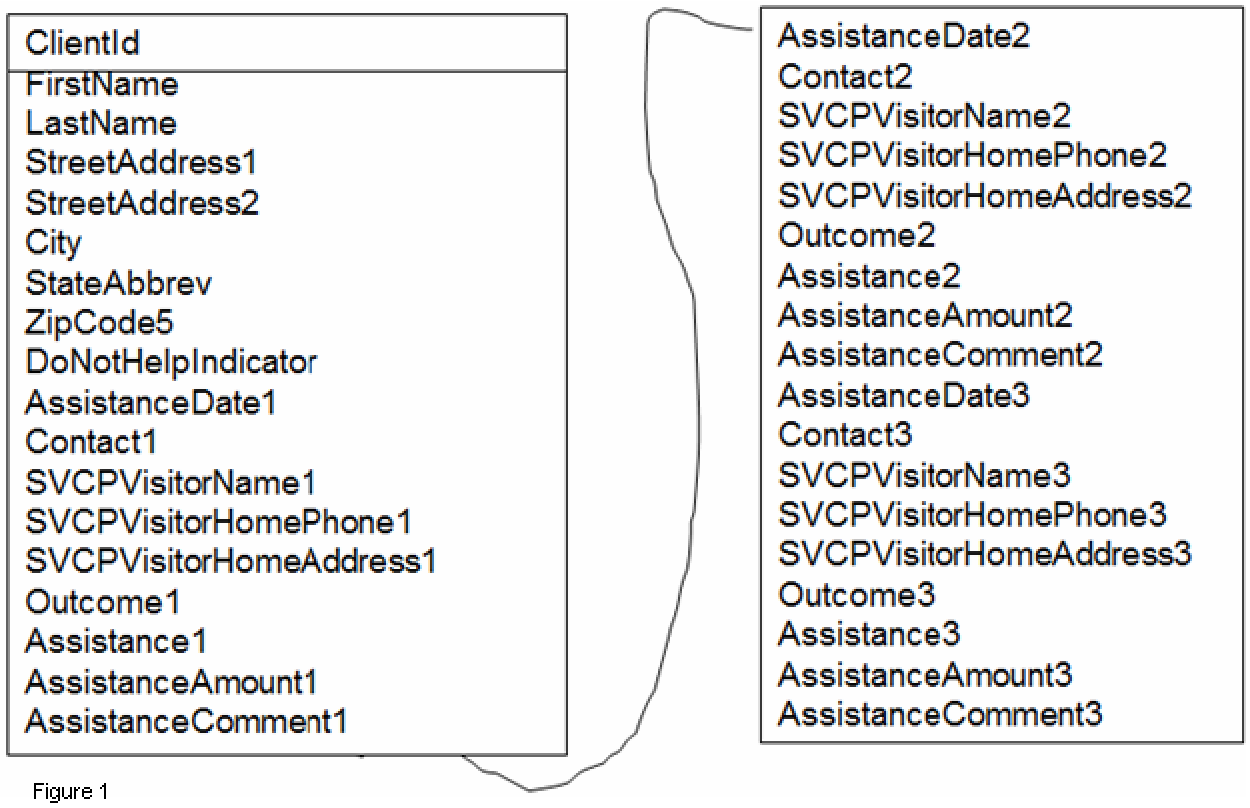
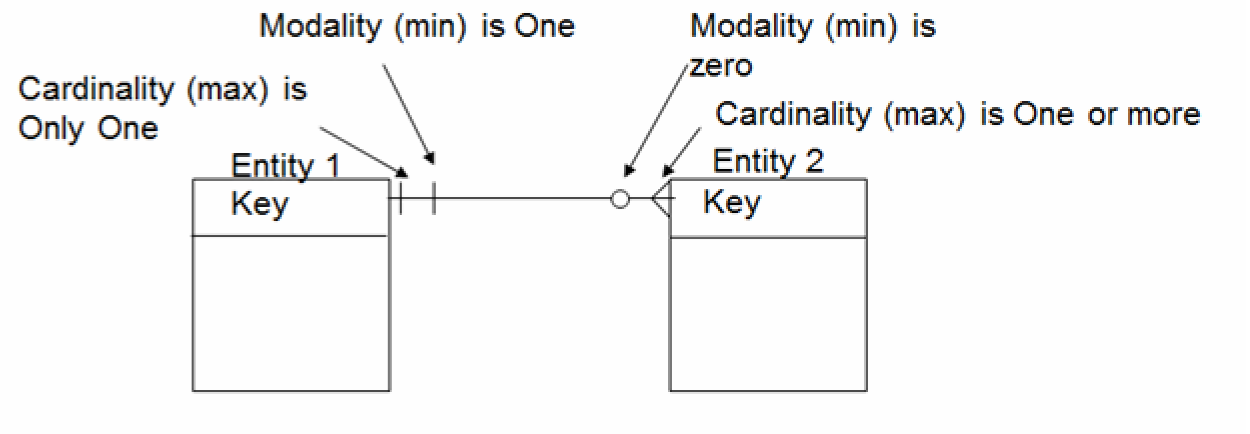


Figure 1

CROWS FOOT NOTATION

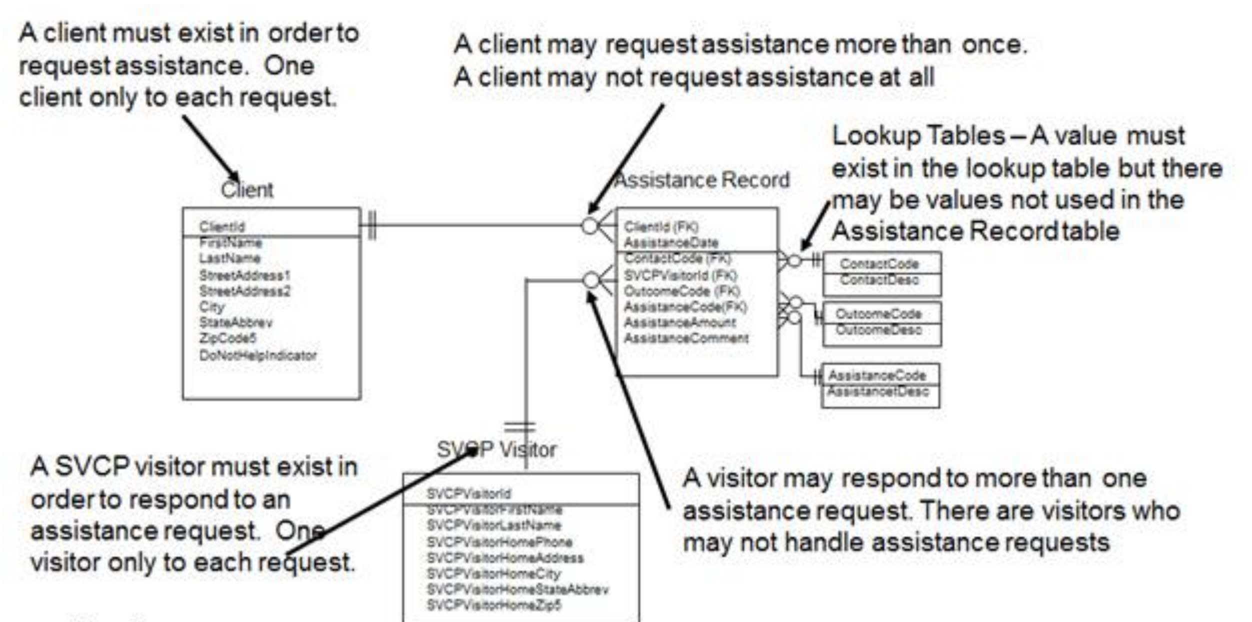
Cardinality is the maximum number of entities that can exist in a relationship. Modality is the minimum number. Figure 2 gives a typical representation of a parent/child relationship:



Entity 1 must exist for Entity 2 to exist
Entity 2 does not need to exist for Entity 1 to exist
More than one Entity 2 may exist for each Entity 1

Figure 2

After data modeling, we ended up with the entity-relationship model in Figure 3. Note that we now need to put together six different sources. At least we shouldn't have to repeat any variables:


A client must exist in order to request assistance. One client only to each request.
A client may request assistance more than once. A client may not request assistance at all
Lookup Tables – A value must exist in the lookup table but there may be values not used in the Assistance Record table
Client
ClientId
FirstName
LastName
StreetAddress1
StreetAddress2
City
StateAbbrev
ZipCode5
DoNotHelpIndicator
Assistance Record
ClientId (FK)
AssistanceDate
ContactCode (FK)
SVCPVisitorId (FK)
OutcomeCode (FK)
AssistanceCode(FK)
AssistanceAmount
AssistanceComment
ContactCode
ContactDesc
OutcomeCode
OutcomeDesc
AssistanceCode
AssistancetDesc
SVCP Visitor
SVCPVisitorId
SVCPVisitorFirstName
SVCPVisitorLastName
SVCPVisitorHomePhone
SVCPVisitorHomeAddress
SVCPVisitorHomeCity
SVCPVisitorHomeStateAbbrev
SVCPVisitorHomeZip5
A SVCP visitor must exist in order to respond to an assistance request. One visitor only to each request.
A visitor may respond to more than one assistance request. There are visitors who may not handle assistance requests
Figure 3

Something you may not be used to is the use of indexes. They comprise a significant part of this system. The original dataset didn't have any, not an uncommon situation in the SAS world. This model has nine currently and well may add more for performance reasons. An index may be used to define the primary key, enforce referential integrity or make retrieval of a subset easier.
The indexes are:
ClientId on Client – primary key
SVCPVisitorId on SVCP Visitor – primary key
ClientID, AssistanceDate on Assistance Record – primary key
ContactCode – primary key on lookup table
OutcomeCode – primary key on lookup table
AssistanceCode – primary key on lookup table
SVCPVisitorId on Assistance Record corresponding to SVCPVisitorId on SVCP Visitor – foreign key
ContactCode corresponding to ContactCode in lookup table – foreign key
OutcomeCode corresponding to OutcomeCode in lookup table – foreign key
AssistanceCode corresponding to AssistanceCode in lookup table – foreign key

ACCESSING THE DATABASE

There are two basic methods to access a database through SAS/Access. One with SQL Pass-Through through which SAS passes the SQL code directly to the database and the newer libname method which assigns a library to a database as if it were a SAS library. Using libname, all commands are then translated into SQL with varying degrees of success. One advantage of using the libname method is that it lets you use the powerful SAS® Enterprise Guide interface. It also lets you query combinations of tables and datasets with proc sql although not without issues as we will see later. As with all libname assignments, the assignment lasts throughout the SAS session, unless disconnected by the database.

In the case of SVCP data, we can assign a libname to the RDBMS like this:

```
libname SVCPDATA sqlsvr user=&userid password=&passwd;
```

If using SAS Enterprise Guide, it is a good idea to set up prompts for the user id and password. Other windowing systems can prompt you for these if you set DBPROMPT=YES. If you are operating in a batch environment, you will need to provide the user id and password in plain text somewhere. You may want to talk to your security people about using external authentication to manage users and passwords.

WHAT'S IN THE LIBRARY?

Allocating your database with the libname statement gives you a few choices of seeing the table and column names. You will probably find that a simple proc contents works best. For the SVCPDATA libname above, proc contents data=SVCPdata._all_ will give you a list of all the tables and columns in the library. An equivalent proc sql would be coded like this:

```
proc sql;
select * from dictionary.columns
where libname='SVCPDATA';
```

The proc contents will probably run much faster than the dictionary query above. In the background, it will usually issue a command to the system tables to give a list of all the table names and then a description of each table. The output will be in the usual proc contents format with the difference of having the engine = RDBMS and a new column called DBMS member type which will generally be TABLE or VIEW. The results below are from proc contents on a set of Oracle tables:

```
The CONTENTS Procedure

      Directory

Libref         MYDBLIB
Engine         ORACLE
Physical Name  elect1
Schema/User    relsas


                                        DBMS
                               Member  Member
#  Name                        Type    Type

1  NUCL12_STATS_WEEKLY         DATA    TABLE
2  NUCL12_BILL_COUNTS          DATA    TABLE
3  NUCL12_STATS_WEEKLY_2014    DATA    VIEW
4  NUCL12_ACCT_OPENS_20140525  DATA    TABLE
5  NUCL12_ACCT_OPENS_20140526  DATA    TABLE
```

The individual tables will look much the same as any SAS library with the exception of having ORACLE for the engine and <missing> for the observation count.

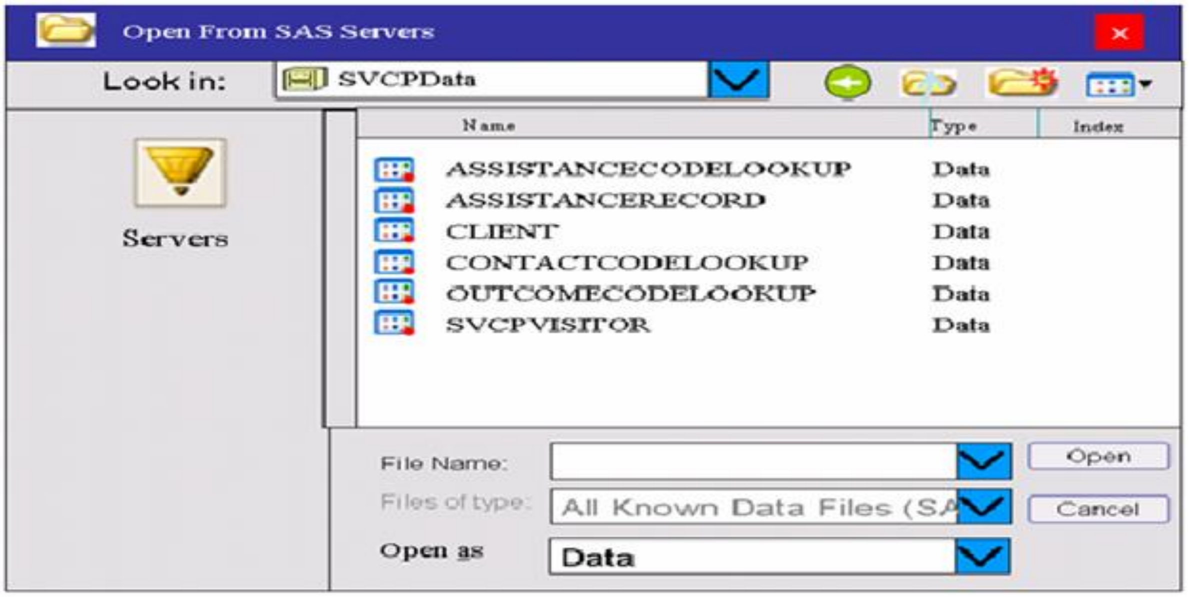
Querying the dictionary usually requires a dynamic call to the RDBMS for metadata and dynamic creation of the dictionary. Everything allocated to your session is included, whether RDBMS or SAS. This can take some time. For example, if you have SAP allocated to your session via the SAS R/3 interface with trace turned on, you will see calls very much like this:

```
r3prep().
Entering send_metadata_request_and_load_metadata()
ACCESS ENGINE: Table returned is AEOI (Revision Numbers)
```

The above message will repeat until all of the SAP table information has been returned. Once all of the SAP metadata has been collected, your R3 session will close like this:

```
Entering r3close.
Leaving r3close.
```

If you are fortunate enough to have SAS/Access along with SAS Enterprise, you will have a much easier time seeing the list of tables and their structures as figure 4 demonstrates:



Open From SAS Servers

Look in: SVCPData

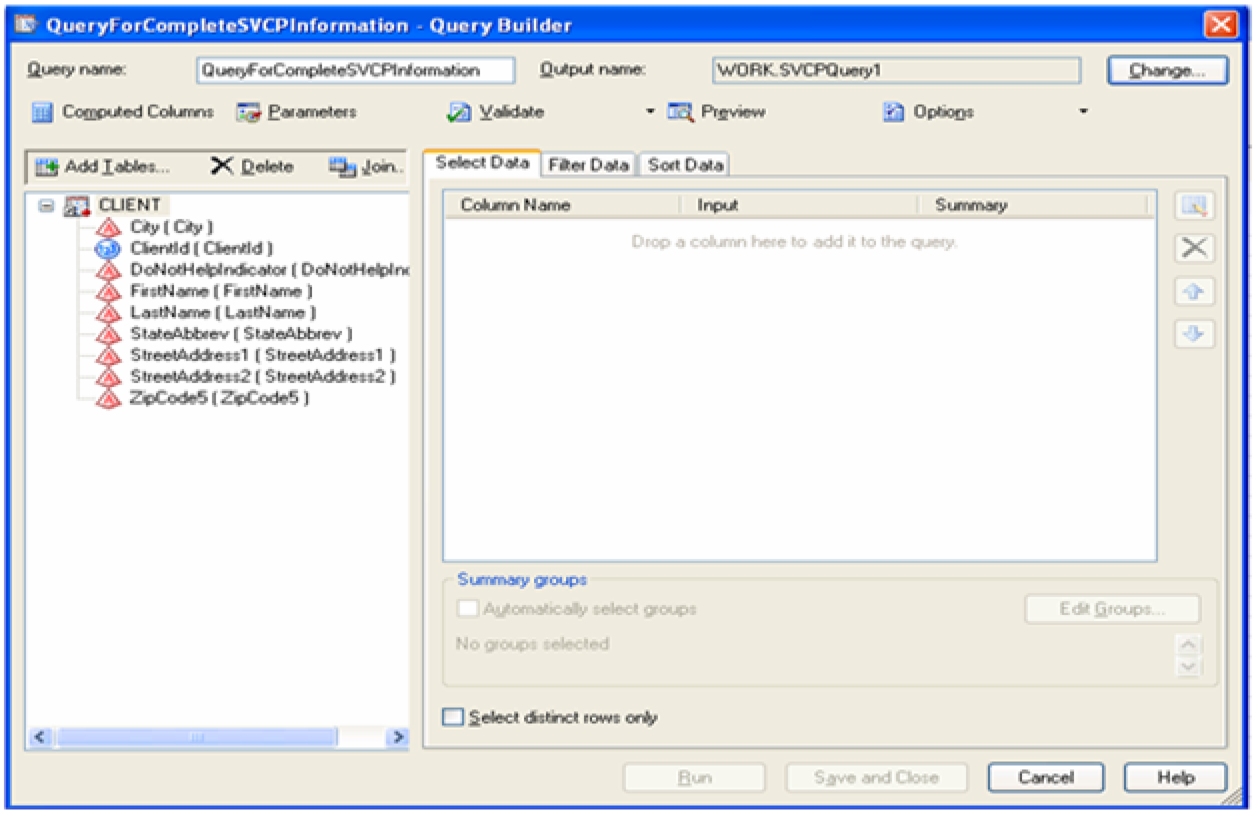
Name	Type	Index
ASSISTANCECODELOOKUP	Data	
ASSISTANCERECORD	Data	
CLIENT	Data	
CONTACTCODELOOKUP	Data	
OUTCOMECODELOOKUP	Data	
SVCPVISITOR	Data	

File Name:
Files of type: All Known Data Files (SA
Open as: Data

Figure 4

You can add any of the tables to your SAS EG project. Just turn off the option to automatically open them when adding them to avoid hanging up your session. You can add these tables much like you can any SAS library.

The Query Builder as shown in figure 5 lets you visualize the table.



QueryForCompleteSVCPInformation - Query Builder

Query name: QueryForCompleteSVCPInformation
Output name: WORK.SVCPQuery1

CLIENT
- City (City)
- ClientId (ClientId)
- DoNotHelpIndicator (DoNotHelpInc
- FirstName (FirstName)
- LastName (LastName)
- StateAbbrev (StateAbbrev)
- StreetAddress1 (StreetAddress1)
- StreetAddress2 (StreetAddress2)
- ZipCode5 (ZipCode5)

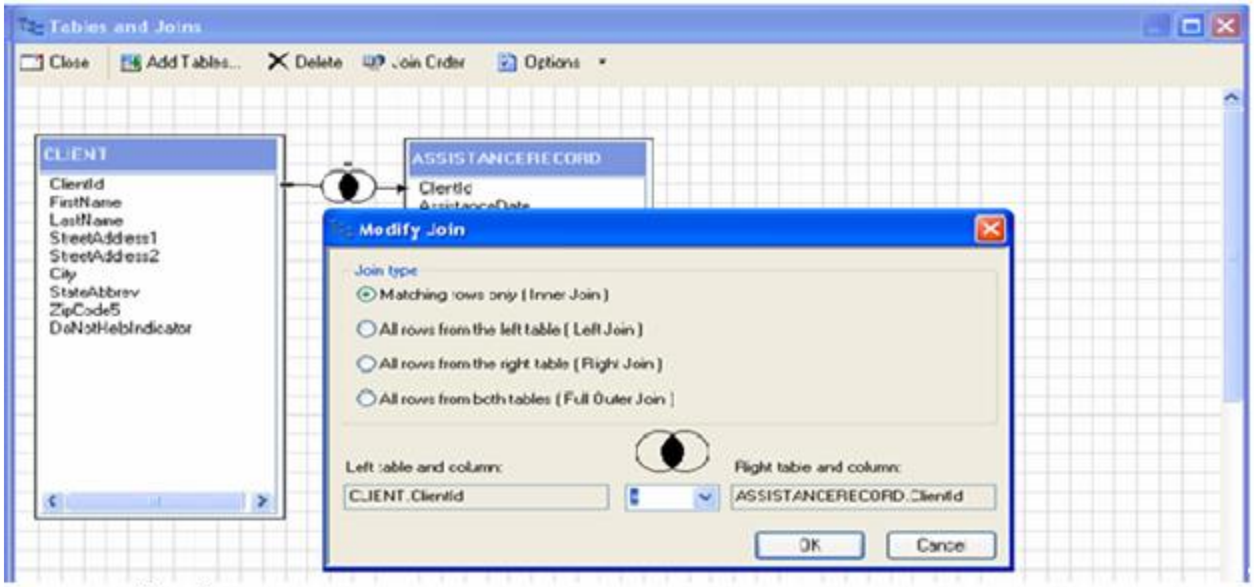
Column Name	Input	Summary

Drop a column here to add it to the query.

Summary groups
Automatically select groups
No groups selected
Select distinct rows only

Figure 5

You may query more than one table by clicking the 'Join' icon. When the join screen comes up, add a table by clicking the 'Add Table' icon. Note the default is an inner join on like-named field (figure 6):



Join type

(✓) Matching rows only (Inner Join)

○ All rows from the left table (Left Join)

○ All rows from the right table (Right Join)

○ All rows from both tables (Full Outer Join)

Left table and column: CLIENT.ClientId

Right table and column: ASSISTANCERECORD.ClientId

Figure 6

Adding all of the tables gives us the diagram in Figure 7:

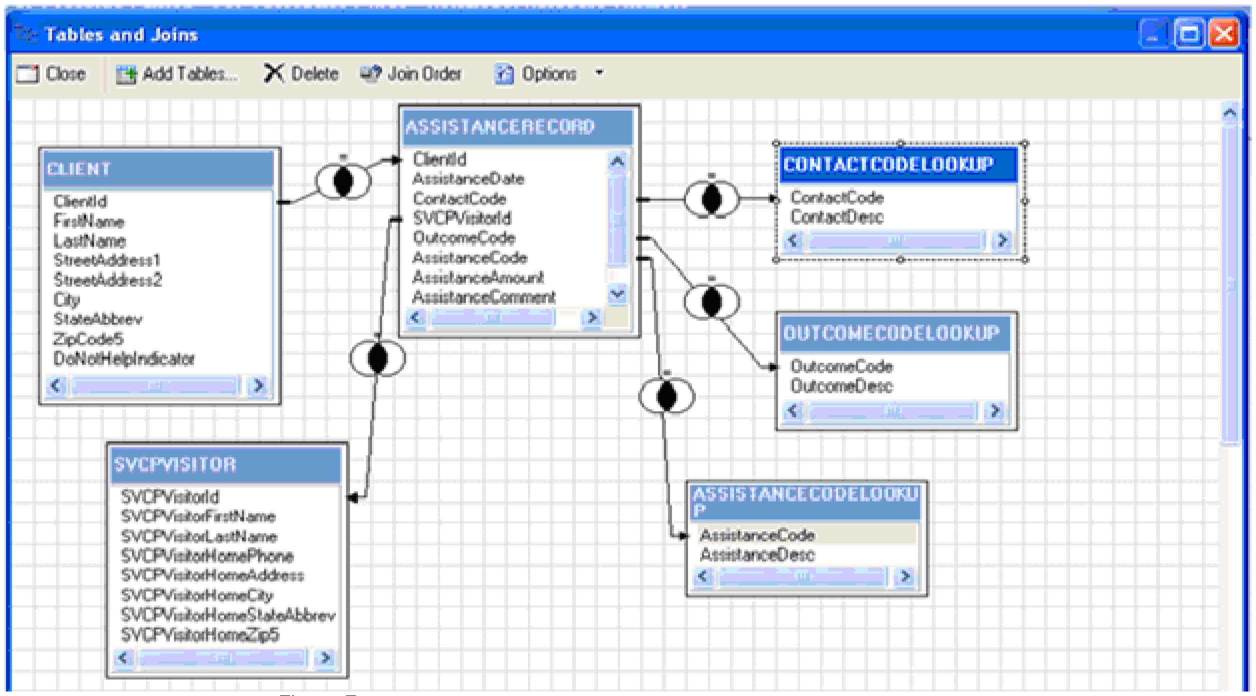
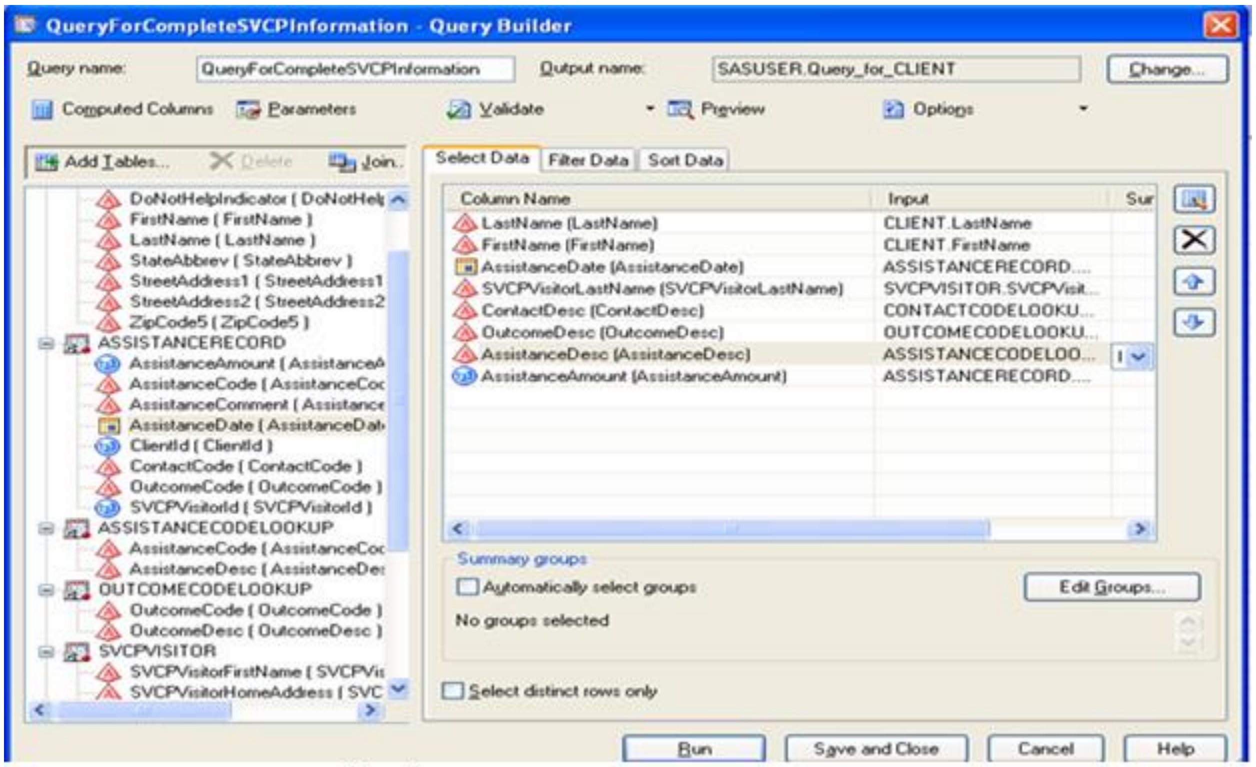


Figure 7

Closing out of this screen brings up back to the main Query Builder form where we can select the columns we want (Figure 8):



Query name: QueryForCompleteSVCPInformation
Output name: SASUSER.Query_for_CLIENT

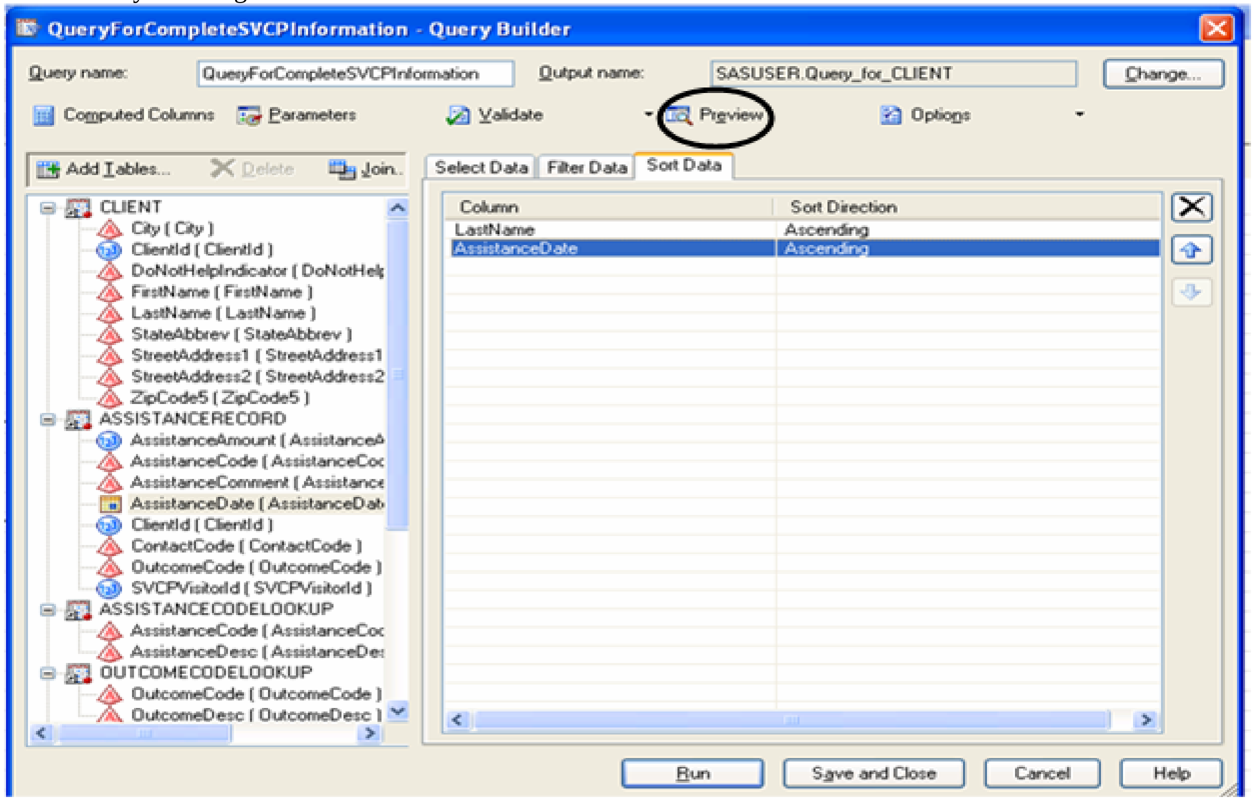
DoNotHelpIndicator (DoNotHelp
FirstName (FirstName)
LastName (LastName)
StateAbbrev (StateAbbrev)
StreetAddress1 (StreetAddress1
StreetAddress2 (StreetAddress2
ZipCode5 (ZipCode5)
ASSISTANCERECORD
AssistanceAmount (AssistanceA
AssistanceCode (AssistanceCoc
AssistanceComment (Assistance
AssistanceDate (AssistanceDat
ClientId (ClientId)
ContactCode (ContactCode)
OutcomeCode (OutcomeCode)
SVCPVisitorId (SVCPVisitorId)
ASSISTANCECODELOOKUP
AssistanceCode (AssistanceCoc
AssistanceDesc (AssistanceDe
OUTCOMECODELOOKUP
OutcomeCode (OutcomeCode)
OutcomeDesc (OutcomeDesc)
SVCPVISITOR
SVCPVisitorFirstName (SVCPVis
SVCPVisitorHomeAddress (SVC

Column Name	Input	Sur
LastName (LastName)	CLIENT.LastName	
FirstName (FirstName)	CLIENT.FirstName	
AssistanceDate (AssistanceDate)	ASSISTANCERECORD....	
SVCPVisitorLastName (SVCPVisitorLastName)	SVCPVISITOR.SVCPVisit...	
ContactDesc (ContactDesc)	CONTACTCODELOOKU...	
OutcomeDesc (OutcomeDesc)	OUTCOMECODELOOKU...	
AssistanceDesc (AssistanceDesc)	ASSISTANCECODELOO...	I
AssistanceAmount (AssistanceAmount)	ASSISTANCERECORD....	

Summary groups
☐ Automatically select groups
No groups selected
☐ Select distinct rows only

Figure 8

Let's sort by client last name and assistance data as in Figure 9. Much like Microsoft/Access, we can switch to a code view by selecting Preview:



Query name: QueryForCompleteSVCPInformation
Output name: SASUSER.Query_for_CLIENT

CLIENT
City (City)
ClientId (ClientId)
DoNotHelpIndicator (DoNotHelp
FirstName (FirstName)
LastName (LastName)
StateAbbrev (StateAbbrev)
StreetAddress1 (StreetAddress1
StreetAddress2 (StreetAddress2
ZipCode5 (ZipCode5)
ASSISTANCERECORD
AssistanceAmount (AssistanceA
AssistanceCode (AssistanceCoc
AssistanceComment (Assistance
AssistanceDate (AssistanceDat
ClientId (ClientId)
ContactCode (ContactCode)
OutcomeCode (OutcomeCode)
SVCPVisitorId (SVCPVisitorId)
ASSISTANCECODELOOKUP
AssistanceCode (AssistanceCoc
AssistanceDesc (AssistanceDe
OUTCOMECODELOOKUP
OutcomeCode (OutcomeCode)
OutcomeDesc (OutcomeDesc)

Column	Sort Direction
LastName	Ascending
AssistanceDate	Ascending

Figure 9

Our generated code is:

```
PROC SQL;
CREATE TABLE SASUSER.Query_for_CLIENT AS SELECT CLIENT.LastName,
CLIENT.FirstName,
ASSISTANCERECORD.AssistanceDate FORMAT=DATEAMPM22.,
SVCPVISITOR.SVCPVisitorLastName,
CONTACTCODELOOKUP.ContactDesc,
OUTCOMECODELOOKUP.OutcomeDesc,
ASSISTANCECODELOOKUP.AssistanceDesc,
ASSISTANCERECORD.AssistanceAmount
FROM SVCPDATA.CLIENT AS CLIENT
INNER JOIN SVCPDATA.ASSISTANCERECORD AS ASSISTANCERECORD ON
(CLIENT.ClientId = ASSISTANCERECORD.ClientId)
INNER JOIN SVCPDATA.ASSISTANCECODELOOKUP AS ASSISTANCECODELOOKUP ON
(ASSISTANCERECORD.AssistanceCode = ASSISTANCECODELOOKUP.AssistanceCode)
INNER JOIN SVCPDATA.OUTCOMECODELOOKUP AS OUTCOMECODELOOKUP ON
(ASSISTANCERECORD.OutcomeCode = OUTCOMECODELOOKUP.OutcomeCode)
INNER JOIN SVCPDATA.SVCPVISITOR AS SVCPVISITOR ON
(ASSISTANCERECORD.SVCPVisitorId = SVCPVISITOR.SVCPVisitorId)
INNER JOIN SVCPDATA.CONTACTCODELOOKUP AS CONTACTCODELOOKUP ON
(ASSISTANCERECORD.ContactCode = CONTACTCODELOOKUP.ContactCode)
ORDER BY CLIENT.LastName, ASSISTANCERECORD.AssistanceDate;
```

If you are fluent in SQL, you could write your own code and submit it from anywhere, even a batch job on Unix. One thing to keep in mind is that embedded spaces and special characters are more common in some data systems like Access or SAP. If you are writing code referencing fields like that, you prefix the name with a single quote and end with a single quote n. BI0/FICATABLE would be 'BI0/FICATABLE'n.

LIBNAME OR PASS-THROUGH?

Besides the ability to graphically set up a query in SAS® Enterprise Guide, there are other benefits to using the libname option.

One is loading a SAS dataset to an RDBMS table. It can be as simple as the Oracle load code below:

```
data mydblib.NUCL12_CUST_SRCH_IDS(bulkload=yes
aload(COMPRESS='yes');
set aload;
output mydblib.NUCL12_CUST_SRCH_IDS;
run;
```

or as a complex as this Teradata load:

```
libname s_box Teradata database = s_box server = viptdrop
dbcommit=0 override_resp_len=YES connection = shared
user = &userid password=&passwd;
```

Breaking the libname down into its components

`libname s_box`	- the name we will assign to the database
`Teradata`	- type of database, Teradata in this case
`database = s_box`	- actual name of database (optional)
`server = viptdrop`	- server through which we are accessing the database (required if you have more than one server)
`dbcommit=0` when to commit changes	- 0 means after each data step or query is completed
`override_resp_len=YES`	- the default length of the buffer used to hold data returned from Teradata to SAS will be set by Teradata not SAS
`connection = shared`	- used to eliminate any deadlocks loading tables within a single tablespace
`user = &userid`	- your userid on this system
`password=&passwd;`	- your password

The load differs slightly from its Oracle equivalent.

```
data s_box.send_curr(fastload=yes
dbtype = (acct = 'char(16)'                    -explicit definition of fields in new
table
lname= 'char(25)'
fname = 'char(20)'
ssn = 'char(9)'
)
Dbcreate_table_opts=
'primary index(acct/nomiss));                - create index on acct
Set sload; - your load file
By acct;
Output s_box.send_curr;
Run;
```

Note that the Teradata engine lets you create an index in the same step. The Oracle engine does not have this option. You will probably have to add an index later, using the Pass-Through method.

One disadvantage to the libname option is that depending on your RDBMS, you may find yourself timing out if you keep your SAS session option long enough. Then when you attempt to use the libname, it will give you login errors and probably suspend your RDBMS account. There's no real way around this but to execute your libname statement occasionally.

Pass-Through is usually better if you are doing a complex query using tables only in the database, adding an index, altering the table structure or using a function or syntax that is specific to that database system. Oracle, DB2 and Teradata will generally allow you to create temporary tables within the same query with the WITH construct. The example below is a simulation of proc freq which may perform better than a proc freq using the libname:

```
with total_count as
(select
count(*) tot_count
from leads.campaign_records
)
,Per_value
as
(select count(*) cnt_value,
Channel_code
From leads.campaign_records
Group by channel_code
)
Select
Channel_code
,cnt_value
,tot_count
,cast(100*(cnt_value/tot_count) as number(6,2)) as percent_cnt
From total_count,
Per_value
;
```

SAS simply cannot translate this as this construct doesn't exist in its syntax:

```
___
180
ERROR 180-322: Statement is not valid or it is used out of proper order
```

Since SAS is being translated to native SQL code, the more complex a query is the more likely using libnamed tables will result in strange code, perhaps not being executable or even worse possibly hanging up both your SAS session and the RDBMS.

```
b.curr_acct_bal, b.last_payment date
from s_box.send_curr inner join
custprod.account_info
on a.acct=b.acct_nbr
)
;
quit;
```

Another advantage to using Pass-Through is the majority of RDBMS's will let you see how a query will execute before actually running it. Native SAS has no such facility. You will generally have another interface available to run an EXPLAIN PLAN statement, even if something rather primitive like SQLPlus or Beteq. Explaining the plan for the previous query might result in something like this:

Explain plan for
```
WITH SUM_CONTACT AS
(SELECT
COUNT(*) AS NO_OF_CONTACTS
, MIN(CONTACT_DATE) AS FIRST_CONTACT_DATE,
MAX(CONTACT_DATE) AS LAST_CONTACT_DATE
, CUSTOMER_ID
FROM LEADS.CAMPAIGN_RECORDS GROUP BY CUSTOMER_ID)
SELECT
A.CUSTOMER_LAST_NAME
, A.CUSTOMER_TELEPHONE
, A.CUSTOMER_BIRTH_DT,
SUM_CONTACT.NO_OF_CONTACTS
, SUM_CONTACT.FIRST_CONTACT_DATE
, SUM_CONTACT.LAST_CONTACT_DATE
FROM LEADS.CAMPAIGN_RECORDS A,
SUM_CONTACT
WHERE A.CUSTOMER_ID = SUM_CONTACT.CUSTOMER_ID;
```

Your plan might look like this:

```
--SELECT STATMENT HINT*ALL ROWS
       - HASH JOIN
              - VIEW
                     - HASH GROUP BY
                            --TABLE ACCESS FULL        CAMPAIGN_RECORDS
                      -TABLE ACCESS FULL               CAMPAIGN_RECORDS
```

This is possibly one of the most expensive queries you could run. Hash joins do not use indexes and thus must read the entire table, commonly know as a full table scan It will build a hash structure for the second input before reading each row from the first input one at a time. If you find people need the SUM_CONTACT table, it might be worthwhile creating it regularly, either as dataset or a table.

The general rule of thumb is that it is better to use an index than scanning the entire table. However, that depends upon the columns of the index being used and more importantly their selectivity. The more selective the index is, the more efficient. In our query using account number, selecting 100 out of the millions of accounts made using an index very efficient. Attempting to retrieve five hundred thousand records may make the index worse than useless. Your DBA can help you understand these access paths and tune your query.

PERFORMANCE CONSIDERATIONS

One very common scenario is to 'join' a native SAS dataset with one or more database tables. If your SAS dataset is fairly large, this would result in full table scans of the database tables, then joining the results to the SAS dataset as the following Oracle SAS trace indicates. This code does a validation of certain fields in a SAS dataset compared to what should be their source in a corresponding Oracle table. We turn on the trace with this statement:

```
OPTIONS FULLSTIMER NOCENTER SAStrace='d,,d' SAStraceloc=SASlog NOSTSUFFIX;
```

The 'd..d' SAStrace setting will send all DBMS calls such as API and client calls, connection information and the row processing you see below to the log. Setting the SAStraceloc to SASlog puts the trace into the regular log. NOTSTSUFFIX gives the short version which is quite long enough as demonstrated below:

First our SAS code using Proc SQL:

```
libname prod Oracle user=&userid pw=”&passwd” ;
```

```
PROC SQL;
CREATE TABLE VALFLDS4 AS
SELECT
a.account_number
,a.dwelling_type
,A.TERRITORY
,S.TERRITORY as S_territory
,case when S.territory = a.territory then 'MATCH '
else 'NOMATCH'
end as territory_match_ind
,A.CUST_FIRST_NM
, S.CUST_FIRST_NM as S_CUST_first_nm
,case WHEN S.CUST_FIRST_NM = a.CUST_FIRST_NM then 'MATCH '
else 'NOMATCH'
end as CUST_first_nm_match_ind
,A.CUST_MIDDLE_I
, S.CUST_MIDDLE_I as S_CUST_middle_i
,case WHEN S.CUST_MIDDLE_I = a.CUST_MIDDLE_I then 'MATCH '
else 'NOMATCH'
end as CUST_MIDDLE_I_match_ind
,A.CUST_LAST_NM
, S.CUST_LAST_NM as S_CUST_last_nm
,case WHEN S.CUST_LAST_NM = a.CUST_LAST_NM then 'MATCH '
else 'NOMATCH'
end as CUST_LAST_NM_match_ind
,A.CUST_NAME_SUFFIX
, S.CUST_NAME_SUFFIX as S_CUST_NAME_SUFFIX
,case WHEN S.CUST_NAME_SUFFIX = a.CUST_NAME_SUFFIX then 'MATCH '
else 'NOMATCH'
end as CUST_NAME_SUFFIX_match_ind
from prod.cust_master as S
inner join list.campaign_curr_cust1 as a
on s.account_number=a.account_number;
```

The trace shows the code translated by SAS and passed to Oracle. Note there is an index on account_number but this request will not use it.

```
ACCESS ENGINE: Exiting dbrqsub with SQL Statement set to
SELECT "ACCOUNT_NUMBER", "TERRITORY", "CUST_FIRST_NM",
"CUST_MIDDLE_I", "CUST_LAST_NM", "CUST_NAME_SUFFIX" FROM
PROD.CUST_MASTER
```

Instead it will do a full-table scan indicated by the trace showing the number of rows fetched as below:

```
ORACLE: Rows fetched : 250
ORACLE: Rows fetched : 250,
```

repeats hundreds of times until we exit the Access engine:

```
ACCESS ENGINE: Exit yoeclos with rc=0x00000000
NOTE: Table WORK.VALFLDS created, with 9461 rows and 65 columns.
NOTE: PROCEDURE SQL used (Total process time):
real time 20:20.00
user cpu time 3:41.19
system cpu time 1:17.42
Memory 1867k
Page Faults 0
Page Reclaims 0
Page Swaps 0
Voluntary Context Switches 195607
Involuntary Context Switches 275468
Block Input Operations 0
Block Output Operations 60
```

We can run the exact same query with a smaller dataset The MULTI_DATASRC_OPT is set to 'IN_CLAUSE' (MULTI_DATASRC_OPT =IN_CLAUSE). The IN_CLAUSE constructs a 'IN' statement automatically which will facilitate use of the index:
First translation to SQL:

```
ACCESS ENGINE: Exiting dbrqsub with SQL Statement set to
SELECT "ACCOUNT_NUMBER", "TERRITORY", "TDSP", "CUST_FIRST_NM",
"CUST_MIDDLE_I", "CUST_LAST_NM", "CUST_NAME_SUFFIX",
FROM
PROD.CUST_MASTER
ACCESS ENGINE: Exiting yoepnt() current_rid=1, next_rid=1
ACCESS ENGINE: yoeget before read, current row=1, next row=1
ORACLE: orqsub()
ACCESS ENGINE: Entering dbrqsub
ORACLE: orqacol()
```
Addition of 'IN CLAUSE' to the SQL being passed to the database
```
ACCESS ENGINE: Add PROC where clause of ( ("ACCOUNT_NUMBER" IN (
'3720001022077','3720001110731','3720001201137',3720001216732',
'3720001261073' , '3720001272777',3720007721773' ) ) )
ORACLE: orqacls()
ACCESS ENGINE: Exiting dbrqsub with SQL Statement set to
SELECT "ACCOUNT_NUMBER", "TERRITORY", "TDSP", "CUST_FIRST_NM",
"CUST_MIDDLE_I", "CUST_LAST_NM", "CUST_NAME_SUFFIX",
FROM
PROD.CUST_MASTER WHERE ( ("ACCOUNT_NUMBER" IN
('3720001022077','3720001110731','3720001201137',3720001216732',
'3720001261073' , '3720001272777',3720007721773'  )
) )
ACCESS ENGINE: Return prep-only, with WHERE_BY option
ORACLE: orlock()
ORACLE: READBUFF option value set to 250.
ORACLE: Rows fetched : 100
NOTE: Table WORK.VALFLDS created, with 100 rows and 65 columns. NOTE: PROCEDURE
SQL used (Total process time):
real time 0.00 seconds
user cpu time 0.07 seconds
system cpu time 0.02 seconds
Memory 1699k
Page Faults 0
Page Reclaims 0
Page Swaps 0

Voluntary Context Switches 21
Involuntary Context Switches 117
Block Input Operations 0
Block Output Operations 0
```
Note that the FETCH message appears only once for the 100 records returned. If you rewrote this query to have a right join because you wanted to bring back all of the dataset observations regardless of whether they match the table or not, you'll go back to a full table scan and fetch all of the records from that table before putting them with your dataset even though you only get 100 records back:
```
from prod.cust_master as S
right join list.campaign_curr_cust1 as a
on s.account_number=a.account_number
ORACLE: Rows fetched : 250 (repeated a few hundred times)
ORACLE: Rows fetched : 151
NOTE: Table WORK.VALFLDS created, with 100 rows and 65 columns.
```

In this case, it's probably better to do this in two stages. The first stage you would use the inner join to bring back all of the records that matched and the second do a merge to add back the ones that didn't.

MACRO VARIABLES AND PROC SQL

The automatic macro variable SQLOBS is set to the number of observations returned from PROC SQL whether executed against datasets or tables. This value can be stored and used elsewhere such as this macro for creating an empty report:
```
%macro empty_report(numobs);
%if &numobs = 0 %then %do;
data _null_;
file print;
put 'Nothing to Report';
run;
%end;
%mend empty_report;
```

```
proc sql;
Select *
from SVCPdata.client
where lastname = 'Buffet';
quit;
%empty_report(&sqlobs);
run;
```
Output when there are no observations:
Nothing to Report

If you are using Pass-Through, two other informative automatic variables are available to you, SQLXMSG and SQLXRC. SQLXRC will return the RDBMS return code and SQLXMSG the associated message. These will give you the RDBMS specific error codes which you can look up, then take to your DBA if you need more help.

HANDLING DUPLICATE COLUMN NAMES

Duplicate column names can be very inconsistent with SAS/Access. Usually if you are accessing your database tables with Pass-Through, any duplicate columns are renamed to slightly difference names. If you are using the libname method, and something like SAS Enterprise Guide Query Builder, any duplicate column names you select will also be automatically renamed. For example, the results of this query would be different depending upon whether this query was coded for a batch job or built through Query Builder. It would also depend upon your installation.

Straight code:

```
Create table org_sale as
Select customer.name, org.name
From mydblib.customer customer
Inner join mydblib.org org
On customer.custid=org.empid
;
```

You may get either
```
Customer.name org.name
John Henderson HungerDunger, HungerDunger, HungerDunger and McCormack
```
or

```
Name
John Henderson
```

In the latter case, you would see a message in the log saying the variable 'name' already exists. When that happens, the original value is the one stored, in this case, the customer name and the second one, organization name from org, dropped.

Query Builder:

```
Create table org_sale as
Select customer.name, org.name1 (numbered by Query Builder)
From customer
Inner join org
On customer.custid=org.empid
;
```

Results will be:
```
name                name1
John Henderson      HungerDunger, HungerDunger, HungerDunger and McCormack
```

Now if you joined these by doing a merge, the last value would be the one stored:
Data org_sale;
Merge customer(in=a rename=(custid=empid))
Org(in=b); By empid;
If a and b;

Merge result set (only one has to be a dataset for these results):

```
Name
HungerDunger, HungerDunger, HungerDunger and McCormack
```

CONCLUSION

Understanding data models should aid you in understanding your data and therefore in turning data into good information. Insight into how SAS fetches and uses data from an RDBMS should give you better results with acceptable performance. This insight will also help you in obtaining support from your database administrator when needed.

RECOMMENDED READING

PROC SQL by Example – Howard Schreier, especially if you are new to PROC SQL in general
Online SAS Manual – SAS Institute, particularly the SAS/Access section for more details on your particular RDBMS.

ACKNOWLEDGEMENTS

Thanks to Joe and Paul Butkovich for your encouragement and support, especially in reviewing this paper

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author is often on the road but can be contacted at

Patricia Hettinger
Email: patricia_hettinger@att.net
Phone: 331-462-2142