Crossing the environment chasm - better queries between SAS and DB2
Mark Sanderson, CIGNA Corporation, Bloomfield, CT

ABSTRACT
The source data used by many SAS® programmers resides within non-SAS relational databases such as DB2 or Oracle. In these situations significant performance gains can be realized by adopting a "pass-through" mentality - let the database do the hard work prior to returning results to the SAS environment. This paper covers the standard approach (and some pitfalls to avoid) related to leveraging PROC SQL against DB2 tables.

INTRODUCTION
Typically, SAS users develop their querying skills strictly within the world of SAS itself. From the first query against source data, to the final output of a summary table or report, the tables and processing are all SAS-based objects or processes. However, many SAS programmers eventually reach a point where they must acquire data from an external data source. When that data source is a relational database management system (RDBMS, e.g. DB2 or Oracle), there are certain tips and tricks that make the transition a bit easier. This paper will assume knowledge of PROC SQL as a starting point and DB2 as the external database.

PASS-THROUGH QUERIES: WHERE DOES MOST OF THE PROCESSING TAKE PLACE?
When querying an external database from SAS, either the SAS engine or the external database is chosen by SAS to perform most of the work. If the external data source is of substantial size, then the most efficient option is usually to let the RDBMS process the query and return the summarized result set. Otherwise, all of the detailed data will have to be brought down to the SAS environment and processed there. Which engine is chosen to process the query depends upon the contents of PROC SQL statement. If the SAS compiler can convert native SAS syntax, keywords, and functions into DB2-compliant code, then you can code as if you never left SAS. Sometimes, however, there is no way for SAS to map a function onto DB2, and it must fall back on the intensive option of pulling all data back to SAS prior to processing it.

These facts drive programming design decisions to balance the use of familiar coding practices (SAS syntax) against efficient processing (pass-through to the DB2).

The next five listings illustrate how you would query a DB2 table, and transition your coding style from an implicit SAS style query (listing 3) to an explicit DB2 pass-through query (listing 5). SAS’s libname statement simplifies pointing your queries to an external data source. However that simplicity also makes it hard to know where your query will be executed. This is the difference between implicit and explicit queries. Implicit queries are parsed by SAS and an attempt is made so that the resulting SQL can be pushed down to DB2 in its entirety. When successfully pushed down, the efficiency of an implicit query should be similar to an explicit version. When SAS cannot push the query down to DB2, the performance hit can be substantial, and in the worst case lead to a timeout or abend if memory limits are exceeded.

Listing 1: Connecting to external data source
libname sysibm db2 sql_functions = all db_index = no;

The option sql_functions = all forces SAS to push as many SQL functions down to DB2 as it can.

Listing 2: Tracing the actual SQL generated by SAS, and where it is executed
OPTIONS sastrace=',,,d' sastraceloc=saslog nostsuffix;

Listing 3: Implicit query using PROC PRINT
PROC PRINT data=sysibm.systables(keep= name colcount);
where substr(name,1,4) = 'SYSV';
RUN;

Listing 4: Explicit DB2 pass-through query
proc sql;
   select * from sysibm.systables where substr(name,1,4) = 'SYSV';
quit;

Listing 5: Explicit IBM DB2 pass-through query
proc sql data=sasuser.systables3;
   select * from sasuser.systables3 where substr(name,1,4) = 'SYSV';
quit;

Coders' Corner
NESUG 2009
Listing 4: Implicit query using PROC SQL
PROC SQL;
    select name,
           colcount
    from sysibm.systables
    where substr(name,1,4) = 'SYSV';
QUIT;

Listing 5: Explicit query using PROC SQL and CONNECT
PROC SQL;
CONNECT to db2 (db=appdm uid=&userid pwd=&passwd);
    select *
    from connection to db2
    {
        select name,
               colcount
        from sysibm.systables
        where substr(name,1,4) = 'SYSV'
    };
    DISCONNECT from db2;
QUIT;

TRANSITIONING TO EXPLICIT QUERIES – COMMON SYNTAX ISSUES
Implicit queries provide a level of comfort because they feel like standard SAS code, and your initial coding productivity can be quite high. However, after enough code development, what often happens is you think you have written code that will successfully be passed to the DB2, but it turns out you missed one function that cannot be mapped. Then the detailed data gets pulled back to SAS for processing, and you have to rewrite the statement explicitly to avoid this inefficiency. After this happens too many times, you will begin to code using the explicit approach from the outset and skip the ambiguity of implicit queries (“Will this one work against DB2? I better turn on the trace option and check the log…”).

Several common issues arise when transitioning SAS-compliant PROC SQL statements to a DB2-compliant syntax. The effort required to resolve syntax and keyword issues is usually modest as the majority can be solved by reading the SAS log and looking up the appropriate DB2 compliant method.

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Item</th>
<th>DB2</th>
<th>SAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Syntax</td>
<td>Quotes</td>
<td>DB2 requires single quoted strings.</td>
<td>SAS can use single or double quoted strings.</td>
</tr>
<tr>
<td>2</td>
<td>Syntax</td>
<td>Dates</td>
<td>‘2007-12-31’</td>
<td>‘31Dec2008’d</td>
</tr>
<tr>
<td>3</td>
<td>Syntax</td>
<td>Group by</td>
<td>DB2 requires the explicit enumeration of items from the select statement.</td>
<td>SAS enables the convenient listing of group by columns using integers 1,2,3,...n</td>
</tr>
<tr>
<td>4</td>
<td>Operators,</td>
<td>Comparisons</td>
<td>=,&lt;,&gt;,&lt;&gt;</td>
<td>gt, it, gte, etc.</td>
</tr>
<tr>
<td>5</td>
<td>Functions</td>
<td>Dividing two integers</td>
<td>To return a floating point number, cast the numerator or denominator to double prior to division operation: 100*double(disease_cnt)/mbr_cnt as prevalence_pct</td>
<td>No issue SAS stores all numeric data as double</td>
</tr>
<tr>
<td>6</td>
<td>Keywords</td>
<td>First / Last keywords</td>
<td>rank(), row_number(), over, partition by, etc. Not illustrated below.</td>
<td>First and last items in a BY group are easy to identify via first.column_name last.column_name</td>
</tr>
</tbody>
</table>
Listing 6: Explicit query illustrating common issues #1-4

```sas
%let special_acct = %str('7654321', '1234567'); /*issue #1 – use single quotes*/
%let start_dt = '1971-01-01'; /*issue #2 – use DB2 date format*/

PROC SQL;
    CONNECT to db2 (db=db2_dmart uid=&userid pwd=&passwd);
    create table work.explicit_tbl as
        select *
        from connection to db2
            (select   CASE when account_num in (&special_acct) then 'Y'
                           else 'N'
                       END as special_account_ind,
                date(rcvd_dt_tmstp) as rcvd_dt,
                year(date(rcvd_dt_tmstp) - brth_dt) as mbr_age,
                substr(zip_10_cd, 1, 5)               as zip_5_cd,
                count(distinct mbr_id)              as mbr_id_cnt
            from     appdm.big_db2_table
            where    brth_dt <= &start_dt
            group by CASE when account_num in (&special_acct) then 'Y'
                          else 'N'
                      END,
                date(rcvd_dt_tmstp),
                year(date(rcvd_dt_tmstp) - brth_dt),
                substr(zip_10_cd, 1, 5)
            order by date(rcvd_dt_tmstp) asc
        )
    DISCONNECT from db2;
QUIT;
```

TRANSITIONING TO EXPLICIT QUERIES – JOIN ISSUES

After resolving issues with syntax or keywords, the next concept to tackle is how to join multiple datasets in DB2. A common barrier faced when extracting healthcare data is that you need to pull data from large transactional tables for a specific set of members. When this member list resides in a SAS dataset, an explicit DB2 query cannot “see” it, so an INNER JOIN approach will not work. (If the member list is small you could consider creating a string to use in an IN statement (see Yang 1998 for an expansion on this and how to work around the 32,000 character limit for string variables).)

Assuming the member list is large, there are typically two options to avoid cross environment joins. The first is to simply move your SAS member list table into DB2. This requires permission to create and insert records in a temporary DB2 table in DB2 (listing 7). The second is a process whereby your knowledge of SAS member list is such that you can recreate it with a set of queries against DB2 tables (listing 8). This will only work in limited cases where your list was created by someone else from the DB2 environment itself.

Listing 7: Avoid cross environment joins... by moving SAS table to DB2

```sas
%let guid = 123456; /*Need to generate a globally unique identifier*/
%let tmp_mbr_table= db2_dmart.tmp_mbr_&guid; /*Unique table name*/

PROC SQL;
    CONNECT to db2 (db=db2_dmart uid=&userid pwd=&passwd);
    EXECUTE (create table &tmp_mbr_table( mbr_id varchar(15) not null with default)
                in tadata_tmp index in tadata_tmp not logged initially
            ) by DB2;
QUIT;
```
/*7.2 Populate table in DB2 with the SAS dataset*/
PROC SQL;
insert into &tmp_mbr_table
    ( mbr_id )
select mbr_id
from   work.mbr_list
;
QUIT;

/*7.3 Extract data joining against the DB2 version of the member list*/
PROC SQL;
    CONNECT to db2 (db=db2_dmart uid=&userid pwd=&passwd);
    create table work.explicit_tbl as
select *
from connection to db2
    (select mbr_id,
        sum(coach_call_cnt) as coach_call_cnt
    from db2_dmart.big_nurse_call_table a
    INNER JOIN
        db2_dmart.&tmp_mbr_table
    on a.mbr_id = b.mbr_id
    group by mbr_id
    );
    DISCONNECT from db2;
QUIT;
Listing 8: Avoid cross environment joins.... an illustration of the WITH keyword more than a true solution for cross-environment joins

/*Imagine the list consisted of all continuously enrolled members with over 120 days of supplied pharmaceuticals, and and total medical spend of over $10,000. The first 4 queries recreate that logic, and then it is applied in the final output*/

PROC SQL;
CONNECT to db2 (db=db2_dmart uid=&userid pwd=&passwd);
create table work.explicit_nurse_call_tbl as
select *
from connection to db2

(WITH tmp_mbr_continuous as
  (select  mbr_id,
       sum(member_month_cnt) as member_month_cnt
  from    db2_dmart.big_eligibility_table
  group by mbr_id
  having   member_month_cnt = 12 ),

tmp_mbr_claim as
  (select  mbr_id,
       sum(pay_amt) as pay_amt
  from    db2_dmart.big_claim_table
  group by mbr_id
  having   pay_amt >= 10000 ),

tmp_mbr_pharmacy as
  (select  mbr_id,
       sum(days_supplied_cnt) as days_supplied_cnt
  from    db2_dmart.big_pharmacy_table
  group by mbr_id
  having   days_supplied_cnt >= 120 ),

  /*The final member list*/
tmp_mbr_list as
  (select a.mbr_id
  from   tmp_mbr_continuous a
  INNER JOIN
         tmp_mbr_claim      b
  on     a.mbr_id = b.mbr_id
  INNER JOIN
         tmp_mbr_pharmacy   c
  on     a.mbr_id = c.mbr_id
  ),

  /*Pull data from your table of interest, driven by the member list*/
select  a.mbr_id,
       sum(coach_call_cnt) as coach_call_cnt
from    db2_dmart.big_nurse_call_table a
        INNER JOIN
        tmp_mbr_list                   b
  on       a.mbr_id = b.mbr_id
group by mbr_id
 );
DISCONNECT from db2;
QUIT;
CONCLUSIONS
Immerse yourself in how your source database – be it DB2, Oracle, etc. – operates. The knowledge gained in SQL syntax, OLAP functions, and relational database design can only improve how you manipulate data within the SAS environment. If you make the transition to using explicit pass through queries you not only improve the efficiency of your initial extract process, but can also leverage the capabilities of your RDBMS. The pitfalls and issues during the initial transition into explicit SQL should be modest (barring a few exceptions like cross environment joins and first/last operations) and offset by less time spent waiting for extracts to complete.

REFERENCES


ACKNOWLEDGEMENTS
Thanks to Charlie Bastnagle and Di Meng for sharing their DB2 / SAS expertise and practical wisdom with regard to data extraction, among other things.

CONTACT INFORMATION
Mark Sanderson
CIGNA Healthcare
900 Cottage Grove Road
Bloomfield, CT
Work phone: 806-226-1492
Email: mark.sanderson@cigna.com

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.

Other brand and product names are registered trademarks or trademarks of their respective companies.