Paper 475-2017

What's Love Gotta Do WITH It

Jason O'Day MBA, US Bank

ABSTRACT

It has become a need-it-now world, and many managers and decision-makers need their reports and information quicker than ever before to compete. As SAS® developers, we need to acknowledge this fact and write code that gets us the results we need in seconds or minutes, rather than in hours or days. SAS is a great tool for extracting, transferring, and loading data, but as with any tool, it is most efficient when used in the most effective way. Using the SQL pass-through techniques presented in this paper can reduce run time by up to 90% by passing the processing to the database instead of moving the data back to SAS to be consumed. You can reap these benefits with only a minor increase in coding difficulty.

INTRODUCTION

The SQL Pass-Through techniques illustrated in this paper require that you either learn or know the native language of the database you are targeting, for our example we will use DB2. This is because we will be passing our SQL code to run inside the database - not on the SAS server - this requires you to understand the version of SQL your database utilizes and also its functions. (DB2 does not understand the DATEPART function for instance.) Because this paper is very code driven it will be heavy with the sample code and explanations for what it is doing. This paper will take you through the progression of using a Libname statement or implicit pass-through to pull data from a DB2 data warehouse in order to join that data within SAS and the alternative of using explicit pass-through. For more information on Implicit and Explicit Pass-Through please see the links in the Recommended Reading portion at the end of the paper. Comparing how these are run and their differing run-times will hopefully lay clear the benefits of each approach.

PROBLEM CONTEXT

Over many years I have developed reports and production data for the business users to consume. The data was stored in DB2 in several data files that needed to be joined in order to create the reports. One of the other employees was proficient in SAS and had built code to do this, but it would run for over 8 hours and then error out due to space constraints. I was able to build a process that would run in the native DB2 environment therefore reducing the runtime and the resources to complete. The new process finished in minutes and completely tied out. Throughout the rest of the paper we will be exploring the typical Implicit Pass-Through solutions to joins in SAS and our solutions which include Explicit Pass-Through.

Options, Macros and Libname statement review

To begin we specify our options, macros and LIBNAME statement. This will not be repeated as we will assume that this is the same for all the code we are reviewing:

```
%GLOBAL dbuser   dbpass     _dsn    _schema;
 %LET _dsn = dsn;
 %LET _schema = schema;
 %LET dbuser = user;
 %LET dbpass = password;

 LIBNAME mylib db2 user="&dbuser" password="&dbpass" db=&_dsn
    schema=&_schema;
```

IMPLICIT PASS-THROUGH SOLUTION FOR SIMPLE MULTI-TABLE JOIN

Please see the sample code below that uses the implicit pass-through in order to pull the data from DB2 and then join it in SAS. This is basic SAS PROC SQL code that is used to join tables once you have set

up the library names. For illustration purposes I have limited the amount of data I am processing. I will create a mere 3,093 lines after the join but if you are dealing with millions of rows of data the time to run this will grow exponentially:

-- Select records that are past due Implicit Pass-Through usage --;

```
PROC SQL;
   CREATE TABLE WORK.table1 AS
   SELECT
         A.groupkey
         ,A.group_data1
         ,CASE WHEN C.contact_ data1 = 'T' AND C.contact_data2 ^= 'T'
             THEN 1
             ELSE CASE WHEN C.contact_data1 ^= 'T' AND C.contact_data3 =
             'T' AND C.data3 ^= 'T' THEN 2
             ELSE 3   END END AS flg
         ,C.contact_data4
         ,DATEPART(B. billing_data1)          AS first_notice_date
         ,DATEPART(B. billing_data2)          AS second_notice_date
         ,DATEPART(B. billing_data3)          AS final_notice_date
         ,B.billing_groupkey
         ,B.billingkey
   FROM  mylib. group_data AS a
         INNER JOIN mylib.billing AS b ON A.groupkey = B.billing_groupkey
         INNER JOIN mylib. contact_data AS c ON A.groupkey =
               C.contact_groupkey
   WHERE B. billing_data4 ^= 'T'
   ORDER BY A. group_data1
;QUIT;
```

```
NOTE: Table WORK.table1 created, with 3093 rows and 9 columns.
NOTE: PROCEDURE SQL used (Total process time):
  real time          8.62 seconds
  user cpu time      0.22 seconds
  system cpu time    0.03 seconds
  Memory                    9465k
  OS Memory                 24428k
  Timestamp          2/14/2014  1:31:19 PM
  Page Faults               3
  Page Reclaims             2338
  Page Swaps                0
  Voluntary Context Switches       147
  Involuntary Context Switches     65
  Block Input Operations           0
  Block Output Operations          0
```

EXLPICIT PASS-THROUGH SOLUTION FOR SIMPLE MULTI-TABLE JOIN

Let's compare the above code with the Explicit pass-through SQL below. You can see the code is very similar but instead of pulling the data into SAS using the Libname engine to DB2 we are joining the data in the DB2 Server then pulling that finished data into SAS. Notice the use of "DATE()" in the example below instead of "DATEPART()" in the above. This is one of the native language differences needed in order to properly run inside DB2. You can see, based on the 'Real Time' that this pull is much faster and little additional time on the front end for coding. We first connect to DB2 then we join the tables while getting the same output as the above but at a fraction of the time. Again, imagine millions of rows of data that used to take hours to run cut down to minutes or seconds. Tell me that would not earn you points

with your boss.

```
*-- Select records that are past due Explicit Pass-Through usage --*;

   PROC SQL;

  *-- INITIAL WRAPPER START--*;
   CONNECT TO db2 AS source
             (DSN=&_dsn
              USER="&dbuser"
              PASSWORD="&dbpass"
             );
   CREATE TABLE WORK.table1 AS SELECT * FROM CONNECTION TO SOURCE
   *-- INITIAL WRAPPER END--*;

   *--CODE PASSED TO RUN INSIDE DB2--*;
       (SELECT
             A.groupkey
             ,A.group_data1
             ,CASE WHEN C.contact_ data1 = 'T' AND C.contact_data2 ^= 'T'
                   THEN 1
                   ELSE CASE WHEN C.contact_data1 ^= 'T'
                         AND C.contact_data3 = 'T'
                         AND C.data3 ^= 'T' THEN 2
                   ELSE 3 END END AS flg
             ,C.contact_data4
             ,DATE(B. billing_data1)             AS first_notice_date
             ,DATE(B. billing_data2)       AS second_notice_date
             ,DATE(B. billing_data3)       AS final_notice_date
             ,B.billing_groupkey
             ,B.billingkey
       FROM  &_schema.. group_data AS a
             INNER JOIN &_schema.. billing AS b ON A. groupkey =
                   B.billing_groupkey
             INNER JOIN &_schema.. contact_data AS c ON A. groupkey =
                   C. contact_groupkey
       WHERE B. billing_data4 ^= 'T'
       ORDER BY A. group_data1
       )

       *-- CLOSING WRAPPER START--*;
       ); DISCONNECT FROM SOURCE;
       *-- CLOSING WRAPPER END--*;

  QUIT;

 NOTE: Table WORK.table1 created, with 3093 rows and 9 columns.
 NOTE: PROCEDURE SQL used (Total process time):
   real time          .99  seconds
   user cpu time      0.06 seconds
   system cpu time    0.02 seconds
   Memory                  1459k
   OS Memory               16752k
   Timestamp          2/14/2014  1:31:21 PM
   Page Faults             3
   Page Reclaims           272
```

```
Page Swaps                    0
Voluntary Context Switches    90
Involuntary Context Switches  19
Block Input Operations        0
Block Output Operations       0
```

IMPLICIT PASS-THROUGH SOLUTION FOR MULTI-TABLE BUILD AND JOIN

But what if you were to first construct a temporary join inside DB2 before joining that result to another DB2 table? Is that even possible? It sure is. In this example, you will see how to create multiple temporary tables in the DB2 work area then join them to make one final table that is passed back to SAS. Notice that we are not passing the source tables back to SAS, just the result of the passed-through query. This is important because instead of using resources to pull loads of data that may or may not be needed we are only bringing back the final product of what we need and this is where we can save time and efficiency. The examples below have been trimmed down for illustration purposes (full versions with the run-times are in the Appendix).

```
PROC SQL;
     CREATE TABLE WORK.contact AS
     SELECT DISTINCT
           B.key
           ,UPCASE(A.bill1_data1) AS contact
           ,B.group_data1    AS name
           ,B.groupkey
           ,A.bill1_key
           ,UPCASE(C.contact_data4) AS address

           ,DATEPART(D.bill2_data1) AS date_added
     FROM mylib.bill1 AS a
           INNER JOIN mylib.group_data AS b ON A.gr_key = B.key
           INNER JOIN mylib.contact_data AS c ON C.groupkey = B.gr_key
           LEFT JOIN mylib.bill2 AS d ON D.bill2_key = A.gr_key
     ;QUIT;

     PROC SQL;
           CREATE TABLE billing AS
           SELECT
                 billing_key
                 ,invoice
                 ,DATEPART(duedate) AS due
                 ,DATEPART(added) AS date_added
                 ,SUM(billed_amt) AS amt_billed
                 ,SUM(paid_amt) AS amt_credited
           FROM mylib.billing
           WHERE DATEPART(duedate) <= "01MAY2014"D
           GROUP BY
                 billing_key
                 ,invoice
                 ,DATEPART(duedate)
                 ,DATEPART(added)
     ;QUIT;

     PROC SQL;
           CREATE TABLE type AS
           SELECT
                 BIL.invoice
```

```
            ,VAL.data1
            ,VAL.val_key
       FROM mylib.billing AS bil
      LEFT JOIN mylib.ref_values AS val ON VAL.type = BIL.type
            AND VAL.val_key=1
;QUIT;

PROC SQL;
      CREATE TABLE WORK.billed AS
      SELECT DISTINCT
            CONT.contact
            ,CONT.name
            ,CONT.key
            ,CONT.address
            ,CONT.add2
            ,BILL.invoice AS invoice_number
            ,BILL.amt_billed AS amount_billed
            ,BILL.amt_credited AS amount_credited
            ,BILL.amt_billed - CUR.amt_credited AS amount_due
            ,TYPE. data1
            ,BILL.date_added
            ,BILL.due AS due_date
      FROM WORK.contact AS cont
            LEFT JOIN WORK.billing AS bill ON
                  CUR.billing_key=HEAD. bill1_key
            LEFT JOIN WORK.type AS type ON TYPE.invoice=CUR.invoice
      WHERE CUR.amt_billed ^=0
      ORDER BY
            HEAD.key
            ,CUR.invoice
;QUIT;
```

EXLPICIT PASS-THROUGH SOLUTION FOR MULTI-TABLE BUILD AND JOIN

So, what is the alternative to the code above? The code is similar to the simple join Explicit Pass-Through code in the previous example but uses “WITH” as a way to build the temp tables in DB2. Each temporary table is enclosed in parentheses and each has a comma directly after that section except for right before the final join (This is key). To make it easier to follow there are comments after each section of code. Again, notice after the “type” section there is no comma unlike all the other sections.

```
PROC SQL;
  CONNECT TO db2 AS source
           (DSN=&_dsn
            USER="&dbuser"
            PASSWORD="&dbpass"
           );

      /* WORK.billed is the name of our output table that will get created in SAS from the pull below*/
      CREATE TABLE WORK.billed AS SELECT * FROM CONNECTION TO SOURCE

      /* Contact info  “contact” is the name of the internal temp table within DB2 we are creating*/
      (WITH contact AS
            (SELECT DISTINCT
                  ,B.key
                  ,UCASE(A.bill1_data1) AS contact
                  ,B.group_data1    AS name
```

```
            ,B.groupkey
            ,A.bill1_key
            ,UCASE(C.contact_data4) AS address

            ,DATE (D.bill2_data1) AS date_added
       FROM &_schema_name..bill1 AS a
            INNER JOIN &_schema..group_data AS b ON A.gr_key = B.key
            INNER JOIN &_schema..contact_data AS c ON
                  C.groupkey = B.gr_key
            LEFT JOIN &_schema..bill2 AS d ON D.bill2_key = A.gr_key
),
/* NOTE THE COMMA AFTER THE PARENTHESES ABOVE */

/* Billing information. “billing” is the name of the internal temp table within DB2 we are creating */
billing AS
      (SELECT
            billing_key
            ,invoice
            ,DATE(duedate) AS due
            ,DATE(added) AS date_added
            ,SUM(billed_amt) AS amt_billed
            ,SUM(paid_amt) AS amt_credited
       FROM &_schema..billing
      WHERE DATE(duedate) <= ‘05/01/2014’
      GROUP BY
             billing_key
            ,invoice
            ,DATE(duedate)
            ,DATE(added)
      ),
      /* NOTE THE COMMA AFTER THE PARENTHESES ABOVE */

/* Type. “Type” is the name of the internal temp table within DB2 we are creating */
type AS
      (SELECT
             BIL.invoice
            ,VAL.data1
            ,VAL.val_key
       FROM &_schema..billing AS bil
            LEFT JOIN &_schema..ref_values AS val ON
                  VAL.type = BIL.type
                  AND VAL.val_key=1
       )

      /*FINAL TABLE - NO COMMA GOES ABOVE*/
      SELECT DISTINCT
            CONT.contact
            ,CONT.name
            ,CONT.key
            ,CONT.address
            ,CONT.add2
            ,BILL.invoice AS invoice_number
            ,BILL.amt_billed AS amount_billed
            ,BILL.amt_credited AS amount_credited
            ,BILL.amt_billed - CUR.amt_credited AS amount_due
            ,TYPE. data1
            ,BILL.date_added
```

```
            ,BILL.due AS due_date
      FROM contact AS cont
            LEFT JOIN billing AS bill ON BILL.gs_key=CONT.gs_key
            LEFT JOIN type AS type ON TYPE.invoice=BILL.invoice
      WHERE BILL.amt_billed ^=0
   ) /*Outside WITH clause*/
   ;DISCONNECT FROM SOURCE;
QUIT;
```

CONCLUSION

Compared to the multiple hours the original process took before erring out, the DB2 explicit pass-through technique finished in less than 15 minutes conveying a massive reduction in runtime while still creating the accurate data we needed with no dropped records and no duplicates. When presenting the data and the code to the business user he was extremely happy with not only the results, but also the time savings. Per the example in the appendix below you will see that the time to run is approximately 95% faster in the explicit pass-through rather than the use of the implicit pass-through.

ACKNOWLEDGMENTS

Thank you to Michael Rainer for conveying this method to me and showing the efficiencies that can be gained. You are a Jedi Master and I am your Padawan learner.

And, thank you to Kirk Paul Lafler for proof reading my paper as well as my presentation. That is a huge help for a first time presenter and I really appreciate it.

RECOMMENDED READING

SAS Communities: “Implicit vs Explicit SQL Pass through SQL Query in SAS”
https://communities.sas.com/t5/Base-SAS-Programming/Implicit-vs-Explicit-SQL-Pass-through-SQL-Query-in-SAS/td-p/261905

Appendix:

The code below is meant to serve as a starting point for you to build out your own process. Please note the log times that verify that the DB2 Pass-Through SQL runtimes and the efficiencies gained

A TYPCIAL SOLUTION FOR MULTI-TABLE BUILD AND JOIN
Complex multi-join using the Implicit Pass-Through:

```
PROC SQL;
      CREATE TABLE WORK.contact AS
      SELECT DISTINCT
            UPCASE(A.bill1_data1) AS contact
            ,CASE WHEN C.contact_data1 = 'P' AND C.contact_data2 ^= 'P'
                  THEN 1
                  ELSE CASE WHEN C.contact_data1 ^= 'P'
                        AND C.contact_data3 = 'P'
                        AND C.contact_data2 ^= 'P' THEN 2
                  ELSE 3 END END AS flg
            ,B.group_data1    AS name
            ,B.group_data2
            ,B.groupkey
            ,A.bill1_key
            ,A.bill1_data2
            ,A.bill1_data3
            ,UPCASE(C.contact_data4) AS address1
            ,UPCASE(C.contact_data5) AS address2
            ,UPCASE(C.contact_data6) AS city
```

```
            ,UPCASE(C.contact_data7) AS state
            ,UPCASE(C,contact_data8) AS zip
            ,DATEPART(D.bill2_data1) AS added_date
      FROM mylib.bill1 AS a
            INNER JOIN mylib.group_data AS b ON A.bill1_key = B.key
            INNER JOIN mylib.contact_data AS c ON C.groupkey = B.key
            LEFT JOIN mylib.bill2 AS d ON D.bill2_key = A.bill1_key
;QUIT;
```

NOTE: Table WORK.CONTACT created, with 2505 rows and 14 columns.
NOTE: PROCEDURE SQL used (Total process time):
real time **30.17 seconds**
cpu time 5.13 seconds

```
PROC SQL;
      CREATE TABLE billing AS
      SELECT
            billing_key
            ,invoice
            ,startdate
            ,enddate
            ,transdesc
            ,DATEPART(duedate) AS due
            ,DATEPART(added) AS date_added
            ,SUM(billed_amt) AS amt_billed
            ,SUM(paid_amt) AS amt_credited
      FROM mylib.billing
      WHERE DATEPART(duedate) <= "01MAY2014"D
      GROUP BY
            billing_key
            ,invoice
            ,startdate
            ,enddate
            ,transdesc
            ,DATEPART(duedate)
            ,DATEPART(added)
;QUIT;
```

NOTE: Table WORK.BILLING created, with 387927 rows and 9 columns.
NOTE: PROCEDURE SQL used (Total process time):
real time **15.48 seconds**
cpu time 5.18 seconds

```
PROC SQL;
      CREATE TABLE applied AS
      SELECT DISTINCT
            GROUP.grpnum
            ,GROUP.key
            ,PAID.pd_data1
            ,PAID. pd_data2
      FROM mylib.group_date AS group
            LEFT JOIN mylib.applied AS paid ON
                  GROUP.key = PAID.group_key
      WHERE UPCASE(PAID.pd_data3) = 'P'
;QUIT;
```

NOTE: Table WORK.PAID created, with 1528 rows and 4 columns.

NOTE: PROCEDURE SQL used (Total process time):
real time 15.54 seconds
cpu time 0.02 seconds

```
PROC SQL;
      CREATE TABLE type AS
      SELECT
            BIL.invoice
            ,VAL.data1
            ,VAL.val_key
      FROM mylib.billing AS bil
      LEFT JOIN mylib.ref_values AS val ON VAL.type = BIL.type
            AND VAL.val_key=1
;QUIT;
```

NOTE: Table WORK.TYPE created, with 380447 rows and 3 columns.
NOTE: PROCEDURE SQL used (Total process time):
real time 3.56 seconds
cpu time 1.07 seconds

```
PROC SQL;
      CREATE TABLE WORK.billed AS
      SELECT DISTINCT
            CONT.contact_flg
            ,CONT.contact
            ,CONT.name
            ,CONT. group_data2
            ,CONT.bill1_key
            ,CONT.groupkey
            ,CONT.group_data2
            ,CONT.group_data3
            ,CONT.address1
            ,CONT.address2
            ,CONT.city
            ,CONT.state
            ,CONT.zip
            ,BILL.invoice AS invoice_nbr
            ,BILL.startdate
            ,BILL.enddate
            ,STRIP(BILL. transdesc) AS invoice_desc
            ,BILL.amt_billed
            ,BILL.amt_credited
            ,BILL.amt_billed - BILL.amt_credited AS amount_due
            ,TYPE.data1
            ,BILL.date_added
            ,BILL.due
            ,APP.amount-APP.apply   AS unapplied
      FROM WORK.contact AS cont
            LEFT JOIN WORK.billing AS bill ON BILL.gs_key=CONT.gs_key
            LEFT JOIN WORK. type AS type ON TYPE.invoice=BILL.invoice
            LEFT JOIN WORK.applied AS app ON APP.key = CONT.key
      WHERE BILL.amt_billed ^=0
      ORDER BY
            CONT.bill1_key
            ,BILL.invoice
;QUIT;
```

```
NOTE: Table WORK.BILLED created, with 2553 rows and 24 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time        12.02 seconds
      cpu time         6.28 seconds
```

Implicit PROC SQL: TOTAL TIME 76.77 seconds
Explicit Pass-Through Total Time (Shown below): 4.15 seconds

EXPLICIT PASS-THROUGH SOLUTION FOR MULTI-TABLE BUILD AND JOIN
Complex multi-join using Explicit Pass-Through to DB2:

```
PROC SQL EXEC;
      CONNECT TO db2 AS source
             (DSN=&_dsn
              USER="&dbuser"
              PASSWORD="&dbpass");
      CREATE TABLE WORK.billed AS
      SELECT * FROM CONNECTION TO SOURCE
/* Contact info  */
(WITH contact AS
      (SELECT DISTINCT
            UCASE(A.bill1_data1) AS contact
            ,CASE WHEN C.contact_data1 = 'P'
                  AND C.contact_data2 ^= 'P' THEN 1
                  ELSE CASE WHEN C.contact_data1 ^= 'P'
                        AND C.contact_data3 = 'P'
                        AND C.contact_data2 ^= 'P'
                        THEN 2
                  ELSE 3 END END AS flg
            ,B.group_data1    AS name
            ,B.group_data2
            ,B.groupkey
            ,A.bill1_key
            ,A.bill1_data2
            ,A.bill1_data3
            ,UCASE(C.contact_data4) AS address1
            ,UCASE(C.contact_data5) AS address2
            ,UCASE(C.contact_data6) AS city
            ,UCASE(C.contact_data7) AS state
            ,UCASE(C,contact_data8) AS zip
            ,DATE(D.bill2_data1) AS added_date
      FROM &_schema.bill1 AS a
            INNER JOIN &_schema..group_data AS b ON A.bill1_key = B.key
            INNER JOIN &_schema..contact_data AS c ON
                  C.groupkey = B.key
            LEFT JOIN &_schema..bill2 AS d ON D.bill2_key = A.bill1_key
      ),
/* NOTE THE COMMA AFTER THE PARENTHESES ABOVE */
/* Billing */
billing AS
      (SELECT
            billing_key
            ,invoice
            ,startdate
            ,enddate
```

```
            ,transdesc
            ,DATE(duedate) AS due
            ,DATE(added) AS date_added
            ,SUM(billed_amt) AS amt_billed
            ,SUM(paid_amt) AS amt_credited
      FROM &_schema..gsbill AS gs
      WHERE DATE(duedate) <= '05/01/2014'
      GROUP BY
            billing_key
            ,invoice
            ,startdate
            ,enddate
            ,transdesc
            ,DATE(duedate)
            ,DATE(added)
      ),
/* NOTE THE COMMA AFTER THE PARENTHESES ABOVE */
/*Applied*/
applied AS
      (SELECT DISTINCT
            GROUP.grpnum
            ,GROUP.key
            ,PAID.pd_data1
            ,PAID. pd_data2
      FROM &_schema..group AS group
            LEFT JOIN &_schema..cash AS paid ON
                  GROUP.key = PAID.group_key
      WHERE UCASE(CASH.pd_data3) = 'P'
      ),
/* NOTE THE COMMA AFTER THE PARENTHESES ABOVE */
/* Type*/
type AS
      (SELECT
            BIL.invoice
            ,VAL.data1
            ,VAL.val_key
      FROM &_schema..billing AS bil
            LEFT JOIN &_schema..ref_values AS val
                  ON VAL.type = BIL.type
                        AND  VAL.val_key=1
      )
/*FINAL TABLE - NO COMMA GOES ABOVE*/
      SELECT DISTINCT
            CONT.contact_flg
            ,CONT.contact
            ,CONT.name
            ,CONT. group_data2
            ,CONT.bill1_key
            ,CONT.groupkey
            ,CONT.group_data2
            ,CONT.group_data3
            ,CONT.address1
            ,CONT.address2
            ,CONT.city
            ,CONT.state
            ,CONT.zip
            ,BILL.invoice AS invoice_nbr
```

```
                ,BILL.startdate
                ,BILL.enddate
                ,STRIP(BILL.transdesc) AS invoice_desc
                ,BILL.amt_billed
                ,BILL.amt_credited
                ,BILL.amt_billed - BILL.amt_credited AS amount_due
                ,TYPE.data1
                ,BILL.date_added
                ,BILL.due
                ,APP.amount-APP.apply   AS unapplied
          FROM contact AS cont
                LEFT JOIN billing AS bill ON BILL.gs_key=CONT.gs_key
                LEFT JOIN type AS type ON TYPE.invoice=BILL.invoice
                LEFT JOIN applied AS app ON APP.key = CONT.key
          WHERE BILL.amt_billed ^=0
          ORDER BY
                CONT.bill1_key
                ,BILL.invoice
    ) /*Outside WITH clause*/
    ;DISCONNECT FROM SOURCE;
 QUIT;
```

```
NOTE: Table WORK.BILLED created, with 2553 rows and 25 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time          4.15 seconds
      cpu time           0.50 seconds
```

CONTACT INFORMATION

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

Jason O'Day MBA
US Bank
Jason.oday@usbank.com