Print

Efficiency Tips for Database Programming in SAS®

Activities and Practices

Course code EETDP11, prepared date: Mon, Mar 21 2022

Copyright © 2022 SAS Institute Inc., Cary, NC, USA. All rights reserved.

Activity: Setting Up Data for Exercises (REQUIRED)

This required activity is necessary to set up your practice environment and must be completed before attempting to run any demo, practice, or subsequent activity code.

All the course programs and data sets for the course *Efficiency Tips for Database Programming in SAS®* are available in the provided virtual lab environment. To access and set up your environment, please follow these steps.

1. Launch your virtual lab following the instructions provided in your course materials.

2. From the Windows desktop taskbar in the virtual lab, start Chrome.

3. On the Chrome Bookmarks bar, click the SAS Studio bookmark.

4. On the SAS Studio logon screen, click in the User ID box, and select **Student** from the stored passwords. Alternatively, you can enter **Student** for the user ID and **Metadata0** for the password. The last character of the password is a zero. When the user ID and password information has been entered, click the **Sign In** button.

5. In the Assumable Groups window, click **No**.

6. On the SAS Studio left navigation bar, click the **Explorer** icon.

7. In the Explorer pane, click the **server** icon and navigate to **Home** > **Courses** > **etdp**.

8. The data and programs used in this course are located in the **etdp** folder.
   a. The **data** folder contains the data used in this course. Additional data is stored in an Oracle database.
   b. The **demos** folder contains the programs used in course demonstrations.
   c. The **activities** and **practices** folders contain the programs used for activities and practices, respectively. Both contain a **solutions** subfolder containing peer-reviewed solution code for the activities and exercises.

9. The **etdp** folder contains a single program named **libnames.sas**. This program should be run every time SAS Studio is started. Copy the code from **libnames.sas** into the SAS Studio autoexec program as follows:
   a. Double-click the **libnames.sas** program to open it on a SAS Studio tab.
   b. Press the key combination Ctrl+a to highlight all of the code.
   c. Press the key combination Ctrl+c to copy the code.
   d. In the Options menu above the Explorer pane, select **Autoexec file**.
   e. Press the key combination Ctrl+v to paste the code into the **Autoexec** window.
   f. Click the **Run** button to run the code and wait for the Log tab to display.
   g. Ensure that there are no errors indicated in the log, and then click the **Save** button to close the Autoexec window.
   h. In SAS Studio, click the X on the tab labeled **libnames.sas** tab to close the **libnames.sas** file.

10. Verify that setup is completed as follows:
    a. Press the F9 function key to reset your SAS session. In the Reset Session window, click **Reset**.
    b. When the system has finished resetting, on the left navigation bar, click the **Libraries** icon.
    c. In the Libraries pane, verify that the libraries **DB** and **SAS** are assigned.
    d. Expand the **DB** library and verify that it contains the tables **COUNTRIES**, **CUSTOMERS**, **MORTGAGES**, and **ORDERS**.
    e. Expand the **SAS** library and verify that it contains the tables **BIG_SPENDERS**, **COUNTRIES**, and **ORDERS**.

11. Setup is now complete and your SAS session is in the state expected as the starting point for each demo. To return the system to this state at any time:
    a. Ensure you have logged in to SAS Studio using the userID/password combination **Student/Metadata0** and have accepted the default No response in the Assumable Groups window.
    b. Press the F9 function key and click **Reset** in the Reset Session window.
    c. When the reset is finished, you are ready to begin the next demo.

Efficiency Tips for Database Programming in SAS®
Lesson 01, Section 1 Activity: Implicit versus Explicit Pass-Through

Open program **et01a01.sas** in the **activities** folder. Run the program and review the Log and Results. Note that the **db** library accesses Oracle data.

This program is an example of which type of pass-through?

a. IMPLICIT
b. EXPLICIT

The correct answer is **a. IMPLICIT**. With implicit pass-through, the LIBNAME engine produces the database SQL.

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 2 Activity: Use SASTRACE

In the **activities** folder, open program **et02a01.sas**, run the program, and review the log.

Is the SQL generated by the SAS/ACCESS LIBNAME engine available for review?

No. The default destination for SASTRACE output (STDOUT) is not visible in SAS Studio.

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 2 Activity: Use SASTRACELOC

In the **activities** folder, open program **et02a02.sas**.

1. Add the option SASTRACELOC=SASLOG to the OPTIONS statement:

```
options sastrace=',,,d' sastraceloc=saslog;
```

2. Run the program and review the log.
   a. Is the SQL generated by the SAS/ACCESS LIBNAME engine available for review?
   b. Is it easy to locate the SQL within the log content?

The SQL code is available, but it is not easy to locate. The SASTRACE output appends a large volume of information to the generated SQL.

Solution Code:

```
options sastrace=',,,d' sastraceloc=saslog;
proc SQL;
select count(*)
   from db.countries
   where countryname like '%a'
;
quit;
options sastrace=off;
```

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 2 Activity: Use NOSTSUFFIX

In the **activities** folder, open program **et02a03.sas**.

1. Add the option NOSTSUFFIX to the OPTIONS statement:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

2. Run the program and review the log.

    a. Is the SQL generated by the SAS/ACCESS LIBNAME engine available for review?

    b. Is it easy to locate the SQL within the log content?

Yes, the SQL generated by the SAS/ACCESS LIBNAME engine is available, and it is easy to locate. NOSTSUFFIX suppresses most SASTRACE output except the generated SQL.

Solution Code:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc SQL;
select count(*)
   from db.countries
   where countryname like '%a'
;
quit;
options sastrace=off;
```

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 2 Activity: FedSQL Tracing

In the **activities** folder, open program **et02a04.sas**. Note that the appropriate SASTRACE options are set. Run the program and review the log.

Is the generated SQL available for review?

No, the generated SQL is not available for review. SASTRACE affects the SAS/ACCESS LIBNAME engine operation. PROC FedSQL dos not use the LIBNAME engine to access the DBMS.

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 2 Activity: Use IPTRACE

In the **activities** folder, open program **et02a05.sas**.

1. Add the IPTRACE option to the PROC FedSQL statement:

```
proc FedSQL iptrace;
```

2. Run the program and review the log.

   a. Is the generated SQL available for review?

   b. Is it easy to determine whether the entire query ran in Oracle?

Yes, the generated SQL is available for review, and it is easy to determine whether the entire query ran in Oracle. The entries in the SAS Log prefaced by "IPTRACE" include: ***FULL pushdown to ORACLE SUCCESS!*** This indicates the entire query was processed in the database.

```
proc fedsql iptrace;
select count(*)
   from db.countries
   where countryname like '%a'
;
quit;
```

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 5

Level 1: Refactor a Program for Faster Execution

This practice asks you to refactor the **et02p01.sas** program to improve efficiency by maximizing in-database processing.

1. Open the **et02p01.sas** program in SAS Studio. Review the current program code.

   a. In SAS Studio, open **practices** > **et02p01.sas**.
   b. Review the DATA step program. Note that it reads the database table **db.mortgages**, subsets rows using a subsetting IF statement, and subsets columns using a KEEP statement.

```
/*et02p01.sas*/
data bad_mortgages;
   set db.mortgages;
   if loanStatus in ('Charged Off','Default');
   keep ACCNUMBER ID AMOUNT CATEGORY LOANGRADE LOANLENGTH;
run;
options sastrace=off;
```

2. Add the appropriate options to display the native database SQL generated by the LIBNAME engine in the SAS log.

   Add an OPTIONS statement above the DATA step, and set the appropriate system options for viewing the database SQL generated by the DATA step.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
data bad_mortgages;
   set db.mortgages;
   if loanStatus in ('Charged Off','Default');
   keep ACCNUMBER ID AMOUNT CATEGORY LOANGRADE LOANLENGTH;
run;
options sastrace=off;
```

3. Run the program and review the log.
   a. Did the subsetting happen in the database or in SAS?
   b. Make a note of the real time required to execute the DATA step program.

   a. The generated native database SQL did not include a WHERE clause, so the subsetting happened in SAS.
   b. Time from the sample log: 3.10 seconds
      Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the next step should be similar.

   Partial Log:

```
ORACLE_1: Prepared: on connection 0
SELECT * FROM ETDP.MORTGAGES
…
ORACLE_2: Executed: on connection 0
SELECT statement  ORACLE_1
NOTE: DATA statement used (Total process time):
        real time           3.10 seconds
```

   View the full log here.

4. Add a new program step below the first DATA step that ensures both row and column subsetting happens in the database. You can either copy and modify the original DATA step or write a new SQL query which produces the same results. Run the original program and your modified program. Which runs faster?

   Subsetting in the database is faster, so the modified program runs faster.
   Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the previous step should be similar.

```
/* Good: DATA Step using KEEP= and WHERE */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
data bad_mortgages;
   set db.mortgages(keep=ACCNUMBER ID AMOUNT CATEGORY
                                       LOANGRADE LOANLENGTH LOANSTATUS);
   where loanStatus in ('Charged Off','Default');
   drop LOANSTATUS;
run;
options sastrace=off;
```

   Partial Log:

```
ORACLE_10: Prepared: on connection 0
SELECT  "ID", "ACCNUMBER", "CATEGORY", "AMOUNT",
"LOANLENGTH", "LOANGRADE", "LOANSTATUS" FROM
ETDP.MORTGAGES  WHERE (("LOANSTATUS" IN ('Charged Off',
'Default')))
…
NOTE: DATA statement used (Total process time):
        real time           0.33 seconds
```

   View the full log here.

   Alternate Solution Code:

```
/* Best: SQL with Select list and WHERE clause */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc sql;
create table bad_mortgages as
select AccNumber
          ,ID
          ,Amount
          ,Category
          ,LoanGrade
          ,LoanLength
   from db.mortgages
   where loanStatus in ('Charged Off','Default')
;
quit;
options sastrace=off;
```

   Alternate Solution Partial Log:

```
ORACLE_13: Prepared: on connection 0
SELECT  "ACCNUMBER", "ID", "AMOUNT", "CATEGORY", "LOANGRADE", "LOANLENGTH", "LOANSTATUS"
FROM ETDP.MORTGAGES
WHERE (("LOANSTATUS" IN ('Charged Off', 'Default')))
…
NOTE: PROCEDURE SQL used (Total process time):
          real time           0.32 seconds
```

   View the full log here.

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 5

Level 2: Refactor a Program for Faster Execution

This practice asks you to refactor the **et02p02.sas** program to improve efficiency by maximizing in-database processing.

1. Open the **et02p02.sas** program in SAS Studio. Review the program code.

   a. In SAS Studio, open **practices** > **et02p02.sas**.
   b. Review the program. Note that it first creates a copy of the database table **db.mortgages** sorted by **LoanGrade** in the **Work** library. Then it uses PROC MEANS to analyze the values for the **LoanGrade** accounts that are in default.

```
/*et02p02.sas*/
proc sort data=db.mortgages (keep=loangrade loanstatus amount)
          out=work.mortgages;
   by loangrade;
run;

title "Default Amounts by LoanGrade";
proc means data=mortgages mean sum;
   var Amount;
   where loanstatus='Default';
   class LoanGrade;
run;
title;
options sastrace=off;
```

2. Add the appropriate options to see the SQL generated by the LIBNAME engine in the SAS log.

   Add an OPTIONS statement above the DATA step to set the appropriate system options for viewing the database SQL generated by the DATA step.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc sort data=db.mortgages (keep=loangrade loanstatus amount)
          out=work.mortgages;
   by loangrade;
run;

title "Default Amounts by LoanGrade";
proc means data=mortgages mean sum;
   var Amount;
   where loanstatus='Default';
   class LoanGrade;
run;
title;
options sastrace = off;
```

3. Run the program. Review the log and answer the following questions:
   a. What indications, if any, show that sorting, subsetting, or summarization happens in the database?
   b. Make a note of the total real time required to execute both the PROC SORT step and the PROC MEANS step for later comparison to the execution time from the next step.

   a. The ORDER BY statement generated by PROC SORT indicates that sorting happened in the database.
   b. Total time from the sample log: 2 seconds
      Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the next step should be similar.

   Partial Log:

```
ORACLE_18: Prepared: on connection 0
SELECT  "LOANGRADE", "AMOUNT", "LOANSTATUS" FROM
ETDP.MORTGAGES  ORDER BY "LOANGRADE" NULLS FIRST
…
NOTE: Sorting was performed by the data source.
NOTE: PROCEDURE SORT used (Total process time):
        real time           1.27 seconds
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.33 seconds
```

   View the full log here.

4. Refactor this program so that subsetting, sorting, and summarization are performed by the database. You can either copy and modify the original program or write an SQL query to produce the same results. Run both the original program and your modified program. Review the log and answer the following questions:
   a. What are the indications that your modification resulted in additional processing by the database?
   b. Did the original or modified program run faster?

   a. The generated SQL for the modified program includes both a WHERE clause and a GROUP BY clause, indicating subsetting and summarization were performed by the database.
   b. The modified program runs significantly faster.
      Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the previous step should be similar.

```
/* Good: PROC MEANS reads the database table directly and generates SQL */
/* Deleted the PROC SORT step - not necessary */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
title "Default Amounts by LoanGrade";
proc means data=db.mortgages mean sum;
   var Amount;
   where loanstatus='Default';
   class LoanGrade;
run;
title;
options sastrace=off;
```

Partial Log:

```
NOTE: SQL generation will be used to perform the initial summarization.
…
ORACLE_22: Prepared: on connection 0
 select COUNT(*) as ZSQL1, MIN(TXT_1."LOANGRADE") as ZSQL2, COUNT(*) as ZSQL3, COUNT(TXT_1."AMOUNT") as ZSQL4, SUM(TXT_1."AMOUNT") as ZSQL5 from ETDP.MORTGAGES TXT_1
where TXT_1."LOANSTATUS" = 'Default' group by TXT_1."LOANGRADE"
…
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.28 seconds
```

View the full log here.

Alternate Solution Using SQL:

```
/* Best: Re-write as SQL */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
title "Default Amounts by LoanGrade";
proc sql;
select LoanGrade
      ,COUNT(*) as NObs
      ,mean(AMOUNT) format=dollar20.2 as Mean
      ,sum(AMOUNT) format=dollar20.2 as Sum
   from db.Mortgages
   where LoanStatus= 'Default'
   group by LoanGrade
 ;
quit;
title;
options sastrace=off;
```

Partial Log:

```
ORACLE_25: Prepared: on connection 0
select TXT_1."LOANGRADE", COUNT(*) as NObs, AVG(TXT_1."AMOUNT") as Mean, SUM(TXT_1."AMOUNT") as Sum from ETDP.MORTGAGES TXT_1
where TXT_1."LOANSTATUS" = 'Default'
group by TXT_1."LOANGRADE"
…
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.13 seconds
```

View the full log here.

Efficiency Tips for Database Programming in SAS®
Lesson 02, Section 5

Challenge: Report Mortgages Late 60 Days

This practice asks you to refactor the **et02p03.sas** program to extract rows that are 30 to 60 days delinquent as of the last update.

1. Open the **et02p03.sas** program in SAS Studio. Review the program code.

   a. In SAS Studio, open **practices** > **et02p03.sas**.
   b. Review the DATA step program. Note that it reads the database table **db.mortgages** and includes a WHERE statement intended to subset the rows based on the difference between the dates in the **Updated** and **LastPayment** columns.

```
/*et02p03.sas*/
data late60;
   set db.mortgages;
   where Updated-LastPayment between 30 and 60;
run;
options sastrace=off;
```

2. Add the appropriate options to see the SQL generated by the LIBNAME engine in the SAS log.

   Add an OPTIONS statement above the DATA step to set the appropriate system options for viewing the database SQL generated by the DATA step.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
data late60;
   set db.mortgages;
   where Updated-LastPayment between 30 and 60;
run;
options sastrace=off;
```

3. Run the program, review the log, and answer the following questions:
   a. What information in the log enables you to determine whether the subsetting happened in SAS or in the database?
   b. Was the subsetting process handled by the database or by SAS?
   c. How many rows were produced?

   a. No WHERE clause was generated for the database SQL, so subsetting did not happen in the database.
   b. The subsetting was handled by SAS.
   c. Zero rows were produced.

   Partial Log:

```
ORACLE_28: Prepared: on connection 0
SELECT  "ID", "ACCNUMBER", "SALARYGROUP", "AGE", "SALARY",
"EMPLENGTH", "CATEGORY", "AMOUNT", "INTERESTRATE",
"LOANLENGTH", "LOANGRADE", "LOANSTATUS", "LASTPAYMENT",
"CANCELLED", "CANCELLEDREASON", "PROMOTION", "UPDATED"
FROM ETDP.MORTGAGES
…
NOTE: There were 0 observations read from the data set DB.MORTGAGES.
WHERE (Updated-LastPayment<=30 and Updated-LastPayment<=60);
NOTE: The data set WORK.LATE60 has 0 observations and 17 variables.
```

   View the full log here.

4. Modify the program to properly subset the data in SAS.
   Hint: Oracle dates are transferred as SAS datetime values, not SAS date values.

   Modify the WHERE expression using the DATEPART function.

```
/* Solution 1:
   Use the DATEPART function to extract the date from each datetime value */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
data late60;
   set db.mortgages;
   where datepart(Updated)-datepart(LastPayment) between 30 and 60;
run;
options sastrace=off;
```

   Alternate Solution Code:

```
/* Solution 2:
   Use the INTCK function dtday interval to process the datetime values directly */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
data late60;
   set db.mortgages;
   where intck('dtday',LastPayment,Updated) between 30 and 60;
run;
options sastrace=off;
```

5. Run the modified program and review the log. How many rows were produced?

   16,035 rows were produced.

   Partial Log:

```
NOTE: The data set WORK.LATE60 has 16035 observations and 17 variables.
```

   View the full log here.

Efficiency Tips for Database Programming in SAS®
Lesson 03, Section 1 Activity: Examine Processing Time

Open program **et03a01.sas** from the **activities** folder. Run the program and review the log.

Which section executed in the shortest time?

a. Section 1 – Cross-library Join
b. Section 2 – Database Join

The correct answer is **b. Section 2 – Database Join**. Processing goes much faster if it can all be accomplished in the database.

Efficiency Tips for Database Programming in SAS®
Lesson 03, Section 2

Level 1: Optimize the Top 3 Suppliers Report

In this practice, rework the original **et03p01.sas** program to optimize a report about the top suppliers for our highest value customers without writing data to the database.

1. Open the **et03p01.sas** program in SAS Studio. Review the program code.

   a. In SAS Studio, open program **et03p01.sas** from the practices folder.
   b. Note that PROC SQL is performing a cross-platform inner join between the database table **db.orders** and the SAS datasets **sas.big_spenders**.

```
/*et03p01.sas*/
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
title "Top suppliers to highest value customers";

proc SQL number;
select Supplier_Name, sum(Quantity) as NumberSold
   from db.orders as c
      inner join
       sas.big_spenders as b
      on c.Customer_ID=b.Customer_ID
   group by Supplier_Name
   having NumberSold > 500
   order by 2 desc, 1
;
quit;
title;
options sastrace=off;
```

2. Run the program. Review the log and answer the following questions:
   a. What indications, if any, show that sorting, subsetting, or summarization happens in the database?
   b. Make a note of the real time required to execute the query for later comparison to the execution time for the last step.

   a. The generated database SQL does not include a WHERE clause, GROUP BY clause, or ORDER BY clause. This shows that all rows were passed to SAS and any subsetting, sorting, or summarization required was processed in SAS.
   b. Time from the sample log: 14.89 seconds
   Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the last step should be similar.

   Partial Log:

```
…Prepared: on connection 0
SELECT  "CUSTOMER_ID", "SUPPLIER_NAME", "QUANTITY"
FROM ETDP.ORDERS
…
NOTE: PROCEDURE SQL used (Total process time):
      real time           14.89 seconds
```

   View the full log here.

3. Without writing to the database, optimize the query to shorten the execution time.
   Hint: Add a SAS/ACCESS to Oracle LIBNAME statement to assign the libref **dbx** to the **student** schema using the MULTI_DATASRC_OPT=IN_CLAUSE option. Use the LIBNAME statement for the **db** libref in the **libnames.sas** program as a guide.

   a. Copy the LIBNAME statement for the **db** library from the **libnames.sas** program and paste it into your program just before the PROC SQL statement. Change the libref to **dbx** and add the MULTI_DATASRC_OPT option.
   b. Modify the SQL query to read from **dbx.orders** instead of **db.orders**.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
title "Top suppliers to highest value customers";

libname dbx oracle path="//server.demo.sas.com:1521/ORCL"
   schema=ETDP user=student password="Metadata0"
   MULTI_DATASRC_OPT=IN_CLAUSE;

proc SQL number;
select Supplier_Name
     , sum(Quantity) as NumberSold
   from dbx.orders as c
      inner join
       sas.big_spenders as b
      on c.Customer_ID=b.Customer_ID
   group by Supplier_Name
   having NumberSold > 500
   order by 2 desc, 1
;
quit;
title;
options sastrace=off;
```

4. Run the modified program, review the log, and answer the following questions:
    a. What is the difference between the database SQL generated in **step 2** and this step?
    b. Was the execution time longer or shorter than in **step 2**?

    a. The generated database SQL in **step 2** did not contain a WHERE clause, but the generated SQL in this step's program included a WHERE clause with an IN operator.
    b. The execution time for this program was shorter than the original program from **step 2**.

    Partial Log:

    ```
    Prepared: on connection 2
    SELECT  "CUSTOMER_ID", "SUPPLIER_NAME", "QUANTITY"
    FROM ETDP.ORDERS
    WHERE (("CUSTOMER_ID" IN (2579, 31756, 38470, 52537,
    59770, 91178)))
    …
    NOTE: PROCEDURE SQL used (Total process time):
          real time           3.81 seconds
    ```

    View the full log here.

5. After finishing this practice, reset your SAS session.

    Press the **F9** key and click the **RESET** button in the pop-up window.

Efficiency Tips for Database Programming in SAS®
Lesson 03, Section 2

Level 2: Optimize the Top 3 Suppliers Report

This practice asks you to rework the **et03p02.sas** program to optimize a report about the top three suppliers for the highest value customers by joining in the database.

1. Open the **et03p02.sas** program in SAS Studio. Review the program code.
   a. In SAS Studio, open the program **et03p02.sas** from the **practices** folder.
   b. Note that PROC SQL is performing a cross-platform inner join between the database table **db.orders** and the SAS data set **sas.big_spenders**.

```
/*et03p02.sas*/
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
title "Top suppliers to our highest value customers";

proc sql number;
select Supplier_Name, sum(Quantity) as NumberSold
   from db.orders as c
      inner join
       sas.big_spenders as b
      on c.Customer_ID=b.Customer_ID
   group by Supplier_Name
   having NumberSold > 500
   order by 2 desc, 1
;
quit;
title;
options sastrace=off;
```

2. Run the program. Review the log and answer the following questions:
   a. What indications, if any, show that sorting, subsetting, or summarization happens in the database?
   b. Make a note of the real time required to execute the query for later comparison to the execution time for the last step.

   a. The database SQL does not include join syntax, so the join processed in SAS.
   b. Time from the sample log: 14.84 seconds
   Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for the last step should be similar.

   Partial Log:

```
Prepared: on connection 0
SELECT  "CUSTOMER_ID", "SUPPLIER_NAME", "QUANTITY"
FROM ETDP.ORDERS
…
NOTE: PROCEDURE SQL used (Total process time):
      real time           14.84 seconds
```

   View the full log here.

3. Optimize this query to shorten the execution time by eliminating the cross-library operation and performing the join in the database.
   - Add a LIBNAME statement to connect to the database **student** schema using the libref **dbx**.
   - Copy **sas.big_spenders** into the **dbx** library.
   - Modify the SQL join to join **db.orders** to **dbx.big_spenders**.
   - After the join, drop **dbx.big_spenders**.
   - Clear the **dbx** libref.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
/* Establish the libref DBX to the STUDENT Oracle schema (writable) */
libname dbx oracle path="//server.demo.sas.com:1521/ORCL"
        schema=student user=student password="Metadata0";

/* Only required if dbx.big_spenders already exists */
proc FedSQL;
   drop table dbx.big_spenders force;
quit;

/* Copy sas.big_spenders to dbx.big_spenders */
proc append base=dbx.big_spenders
            data=sas.big_spenders;
run;

/* Join in the database */
title "Top suppliers to highest value customers";
proc sql number;
select Supplier_Name, sum(Quantity) as NumberSold
   from db.orders as c
      inner join
       dbx.big_spenders as b
      on c.Customer_ID=b.Customer_ID
   group by Supplier_Name
   having NumberSold > 500
   order by 2 desc, 1
;
drop table dbx.big_spenders;
quit;
title;
libname dbx clear;
options sastrace=off;
```

4. Run the program and review the log.

   Note: If you used PROC APPEND or PROC COPY to copy data to Oracle, you can expect to see the following warning in the log. It has no effect on program operation or efficiency:

```
WARNING: Engine ORACLE does not support SORTEDBY operations.
SORTEDBY information cannot be copied.
```

   a. What indications, if any, show that sorting, subsetting, or summarization happens in the database?
   b. How does the total real time required to execute this program compare to that of the original query in **step 2**?

   a. The database SQL includes both the join and GROUP BY syntax, so the join and summarization both processed in the database.
   b. This program ran faster than the original query in **step 2** even though more steps were required. Cumulative real time for this process in the sample log is 4.25 seconds.
   Note: Although the actual execution times will vary each time that a program is run, the relative magnitude in comparison to the execution times for **step 2** should be similar.

Partial Log:

```
NOTE: PROCEDURE FEDSQL used (Total process time):
      real time           0.14 seconds
…
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.13 seconds
…
Prepared: on connection 0
select c."SUPPLIER_NAME", SUM(c."QUANTITY") as NumberSold
from ETDP.ORDERS c inner join student.BIG_SPENDERS b
on c."CUSTOMER_ID"= b."CUSTOMER_ID"
group by c."SUPPLIER_NAME"
order by NumberSold desc NULLS LAST, c."SUPPLIER_NAME" asc NULLS FIRST

ORACLE_14: Executed: on connection 0
SELECT statement  ORACLE_13
…
NOTE: PROCEDURE SQL used (Total process time):
      real time           3.98 seconds
```

View the full log here.

Efficiency Tips for Database Programming in SAS®
Lesson 04, Section 2

Level 1: Use Existing Database SQL in a SAS Report

This practice asks you to reproduce an existing Oracle report in SAS with better formatting. You have been provided a copy of the Oracle query that generates the desired report, and you will use explicit pass-through to execute the query.

1. Open the **et04p01.sas** program in SAS Studio. Review the program code.

   a. In SAS Studio, open **practices** > **et04p01.sas**.
   b. Review the program and notice that the starter PROC SQL code contains the CONNECT, SELECT, and DISCONNECT statements required to execute database SQL using explicit pass-through. Note that the comments section contains the Oracle SQL required to generate the report.

```
/*et04p01.sas*/
/*******************************************************************************
Here is the Oracle SQL code:
select DECODE(loanstatus,'Late','Active','Current','Active','Inactive') as Status
     ,count(*) as Total
     from etdp.mortgages
     group by DECODE(loanstatus,'Late','Active','Current','Active','Inactive')
     order by 1
*******************************************************************************/
proc sql;
/* Complete the connect statement using the libref DB to provide the connection */
connect ;
select Status
      ,Total
        from connection to db
    /* Paste all of the Oracle SQL inside the parentheses */
        ()
;
disconnect from db;
quit;
```

2. Modify the starter program as follows:
   - Insert the libref for the Oracle database (**DB**) into the CONNECT USING statement.
   - Format the values in the **Total** column using the COMMA8. format for readability.
   - Copy the Oracle SQL and paste it between the parentheses in the SELECT statement.

```
proc sql;
/* Complete the connect statement using the libref DB to provide the connection */
connect using db;
select Status
      ,Total format=comma8.
    from connection to db
    /* Paste all of the Oracle SQL inside the parentheses */
    (select DECODE(loanstatus, 'Late', 'Active', 'Current', 'Active', 'Inactive') as Status
          ,count(*) as Total
       from etdp.mortgages
       group by DECODE(loanstatus,'Late', 'Active', 'Current', 'Active', 'Inactive')
       order by 1)
;
disconnect from db;
quit;
```

3. Run the program. How many loans are classified as *Active*?

   460,147 loans are classified as *Active*.

   Results:

STATUS	TOTAL
Active	460,147
Inactive	64,576

   View the results here.

Efficiency Tips for Database Programming in SAS®
Lesson 04, Section 2

Level 2: Use Existing Database SQL in a SAS Report

The Customer Support line received a phone call from a customer named Ray Karlson, but they didn't get his customer ID or call-back number. You have been tasked with finding contact information for customers with names sounding like *Ray Karlson* in the **db.Customers** table. You have been provided a sample Oracle query that phonetically finds customers whose names sound like *Teeny Gorden*.

1. Open the **et04p02.sas** program in SAS Studio. Note that the comment contains an Oracle SQL query which will produce a report of customers with names that sound like "Teeny Gorden".

    a. In SAS Studio, open **practices** > **et04p02.sas**.
    b. Review the program and notice that the comments section contains an Oracle SQL query which will produce a report of customers with names that sound like "Teeny Gorden".

```
/******************************************************************************
 Task:  The Customer Support line received a call from "Ray Karlson" but failed
        to get his Customer_ID number. Please find his Customer_ID so they can
        resolve his issue.

This Oracle SQL code finds customers with names like "Teeny Gorden":

select Customer_ID, customer_name
   from ETDP.customers
   where soundex(SUBSTR(customer_name,1,5))=soundex('Teeny')
     and soundex(SUBSTR(customer_name, INSTR(customer_name, ' ',-1,1)))=soundex('Gorden')
   order by customer_name
******************************************************************************/
```

2. Copy the Oracle SQL and modify it to find customers with names like *Ray Karlson*. Use explicit SQL pass-through to execute the query in the database and display the results in SAS. How many names are returned?

Two names are returned:

CUSTOMER_ID	CUSTOMER_NAME
75046	R.H. Karlsson
62952	R.W. Karels

View the results here.

Solution Code - CONNECT USING Statement:

```
proc sql;
connect using db;
select *
        from connection to db
        (select Customer_ID, customer_name
            from ETDP.customers
        where soundex(SUBSTR(customer_name,1,5))=soundex('Ray')
          and soundex(SUBSTR(customer_name, INSTR(customer_name, ' ',-1,1)))=soundex('Karlson')
          order by customer_name)
;
disconnect from db;
quit;
```

Solution Code - CONNECT Statement with Database Connection Arguments:

```
proc sql;
connect to Oracle (path="//server.demo.sas.com:1521/ORCL" user="student" password="Metadata0");
select *
        from connection to oracle
        (select Customer_ID, customer_name
            from ETDP.customers
        where soundex(SUBSTR(customer_name,1,5))=soundex('Ray')
          and soundex(SUBSTR(customer_name, INSTR(customer_name, ' ',-1,1)))=soundex('Karlson')
          order by customer_name)
;
disconnect from oracle;
quit;
```

Alternate Solution Using PROC FEDSQL:

```
proc FedSQL;
select *
        from connection to db
        (select Customer_ID, customer_name
            from ETDP.customers
        where soundex(SUBSTR(customer_name,1,5))=soundex('Ray')
          and soundex(SUBSTR(customer_name, INSTR(customer_name, ' ',-1,1)))=soundex('Karlson')
          order by customer_name)
;
quit;
```

Efficiency Tips for Database Programming in SAS®
Lesson 04, Section 3 Activity: Modify FedSQL SELECT

Open program **et04a01.sas** from the **activities** folder.

1. Run the program and review the Results tab.
2. Modify the FedSQL SELECT statement to apply the **comma8.** format to the **Orders** column values. Run the program and review the Log and Results tab to ensure that the correct results are produced without errors or warnings.
3. What function was used to apply the format?

Either PUT or PUTN can be used to apply the format.

Solution Code:

```
proc FedSQL;
select Source
      ,put(Orders,comma8.) as Orders
   from connection to db
   (
    select DECODE (order_type,
                           2, 'Internet',
                           3, 'Curbside') as Source
                   ,count(*) as Orders
       from etdp.orders
       where extract(year from order_date)=2020
         and order_type in (2,3)
       group by DECODE (order_type,
                                 2, 'Internet',
                                 3, 'Curbside')
    )
;
quit;
```

Alternate Solution Code:

```
proc FedSQL;
select Source
      ,putn(Orders,'comma8.') as Orders
   from connection to db
   (
    select DECODE (order_type,
                           2, 'Internet',
                           3, 'Curbside') as Source
                   ,count(*) as Orders
       from etdp.orders
       where extract(year from order_date)=2020
         and order_type in (2,3)
       group by DECODE (order_type,
                                 2, 'Internet',
                                 3, 'Curbside')
    )
;
quit;
```

Efficiency Tips for Database Programming in SAS®
Lesson 05, Section 2 Activity: Running FedSQL Code in CAS

Open program **et05a02.sas** from the **activities** folder.

1. Run the program and review the log. Note the ERROR indicating that the SQL executes in Base SAS, not CAS:

   ERROR: Table "CASDB.CUSTOMERS" does not exist or cannot be accessed
   ERROR: BASE driver, schema name CASDB was not found for this connection

2. Add the SESSREF= option to the PROC FedSQL statement so the SQL will execute in CAS, and then resubmit the program. How many customers live in Italy (Country=IT)?

   9,000

   Solution Code:

```
cas mySession;

/* Establish caslibs, assign LIBREFs for easy viewing */
caslib casDB desc="Oracle caslib"
             datasource=(srctype="oracle",path="//server.demo.sas.com:1521/ORCL",schema="ETDP"
               /* UserIDs are passed to the DBMS as-is, and Oracle stores them in UPPER CASE */
               ,username="STUDENT"
               ,pwd="Metadata0"
                )
;

proc FedSQL /* Add option here -> */ sessref=mySession;
select Country
      ,put(Count(*),comma8.) as Customers
   from casDB.Customers
   group by Country
   order by put(Count(*),comma8.) desc
   limit 5
;
quit;
cas mySession terminate;
```

Efficiency Tips for Database Programming in SAS®
Lesson 05, Section 3 Practice: Working with DBMS Tables in CAS

On the SAS Compute Server, the **db** libref provides access to tables in the Oracle ETDP schema. In this practice, we establish a CAS session and create a caslib named **casDB** to provide access to those same Oracle tables to work with them in CAS to produce a series of mortgage loan status reports for select subsidiaries of our company.

1. In SAS Studio, open the program **et05p01.sas** from the **practices** folder. Review the code in the **Setup** section.

    a. In SAS Studio, open **practices** > **et05p01.sas**.
    b. Review the code in the **Setup** section. The code first establishes a CAS session named **mySession** and then creates a caslib named **casDB** with Oracle as the data source. Next, a PROC CASUTIL step lists the files in the **casDB** caslib, and a PROC FEDSQL step lists the tables in the **db** library.

```
/*et05p01.sas*/
/******************************************************************************
 Setup
******************************************************************************/
/* Start a CAS session */
cas mySession;
/* Establish caslibs, assign LIBREFs for easy viewing */
caslib casDB desc="Oracle caslib"
             datasource=(
                srctype="oracle"
               /* UserIDs are passed to the DBMS as-is, and Oracle stores them in UPPER CASE */
               ,username="STUDENT"
               ,pwd="Metadata0"
               ,path="//server.demo.sas.com:1521/ORCL"
               /* Schema names are passed to the DBMS as-is, and Oracle stores them in UPPER CASE */
               ,schema="ETDP"
                ) libref=casDB
      ;

ods noproctitle;
ods select FileInfo;
title "List of 'Files' in the casDB caslib";
proc casutil;
   list files incaslib="casDB";
quit;

title "List of tables in the DB library";
proc FedSQL libs=(db);
select TABLE_NAME as Name
      ,TABLE_CAT as Libref
      ,TABLE_SCHEM as Schema
      ,TABLE_TYPE as Type
   from dictionary.tables
   where TABLE_SCHEM='ETDP'
   order by 1
;
quit;
title;
```

2. Execute the code in the **Setup** section and review the results to verify that the names of the ***files*** listed in the **casDB** caslib data source correspond to the names of the ***tables*** in the **db** library.

The program runs without error, and the list of files in the **casDB** caslib corresponds to the list of tables in the **db** library.

Partial Results:

List of Files in the casDB caslib

Name	Catalog	Schema	Type	Description
COUNTRIES	CASDB	ETDP	TABLE	
CUSTOMERS	CASDB	ETDP	TABLE	
MORTGAGES	CASDB	ETDP	TABLE	
ORDERS	CASDB	ETDP	TABLE	

List of Tables in the db Library

NAME	LIBREF	SCHEMA	TYPE
COUNTRIES	CASDB	ETDP	TABLE
CUSTOMERS	CASDB	ETDP	TABLE
MORTGAGES	CASDB	ETDP	TABLE
ORDERS	CASDB	ETDP	TABLE

View the results here.

3. Review **Step 1** of the program **et05p01.sas**.

   Four PROC FEDSQL steps read from the **casDB.MORTGAGES** table and produce separate reports for the Abu Dhabi, Djibouti, Gaborone, and Jakarta subsidiaries.

```
/*et05p01.sas*/
/*******************************************************************************
 Step 1
*******************************************************************************/
proc FedSQL;
title "Loan Status for Abu Dhabi";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='A'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL;
title "Loan Status for Djibouti";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='D'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL;
title "Loan Status for Gaborone";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='G'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL;
title "Loan Status for Jakarta";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='J'
   group by LoanStatus
   order by LoanStatus
;
quit;
title;
```

4. Modify **Step 1** of the program **et05p01.sas** to run the PROC FEDSQL steps in CAS.

   Add the SESSREF= option to each PROC FEDSQL statement so that the FedSQL code executes in CAS using the CAS session named **mySession**.

```
proc FedSQL sessref=mySession;
title "Loan Status for Abu Dhabi";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='A'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL sessref=mySession;
title "Loan Status for Djibouti";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='D'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL sessref=mySession;
title "Loan Status for Gaborone";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='G'
   group by LoanStatus
   order by LoanStatus
;
quit;

proc FedSQL sessref=mySession;
title "Loan Status for Jakarta";
select LoanStatus, count(*)
   from casDB.mortgages
   where substr(ID,1,1) ='J'
   group by LoanStatus
   order by LoanStatus
;
quit;
title;
```

5. Run your modified **Step 1** code. Then review the log and note how long it takes for each PROC FEDSQL step to execute.

   The program runs without error, and each FedSQL step takes about 4 seconds to execute.
   Note: The actual execution times might vary each time that a program is run, but the relative magnitude in comparison to one another will remain the same.

   Partial Log:

```
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           4.38 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           4.37 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           4.36 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           4.35 seconds
```

   View the full log here.

6. Review **Step 2** of the program **et05p01.sas**.

   Step 2 contains a PROC CASUTIL step that loads a copy of the **MORTGAGES** table to caslib **casDB** memory and then lists the in-memory tables in the caslib.

```
/*et05p01.sas*/
/******************************************************************************
 Step 2
******************************************************************************/
ods select TableInfo;
proc casutil;
   load casdata="MORTGAGES" incaslib="casDB"
        casout="MORTGAGES" outcaslib="casDB" replace;
   list tables incaslib="casDB";
quit;
```

7. Execute the **Step 2** code. Then review the log and note how long it takes to load the **MORTGAGES** table.

   The program ran without error. It took about 5 seconds to load the MORTGAGES table. **Partial Log:**

```
NOTE: PROCEDURE CASUTIL used (Total process time):
        real time           5.17 seconds
```

   View the full log here.

8. Execute your modified **Step 1** code with no additional modifications. Review the log and note how long it takes for each PROC FEDSQL step to execute. Did the PROC FEDSQL steps run faster or slower after loading the **MORTGAGES** table to the caslib memory?

   The program runs without error, and each PROC FEDSQL step takes about 0.2 second to execute. The PROC FEDSQL steps ran much faster after the table was loaded into memory.
   Note: The actual execution times might vary each time that a program is run, but the relative magnitude in comparison to one another will remain the same.

   Partial Log:

```
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           0.20 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           0.20 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           0.24 seconds
…
NOTE: PROCEDURE FEDSQL used (Total process time):
        real time           0.20 seconds
```

   View the full log here.

9. Run the code in **Step 4** of the program **et05p01.sas** to terminate the CAS session. Review the log and verify that the CAS session was terminated.

   The CAS session named **mySession** was successfully terminated.

   Code:

```
/*et05p01.sas*/
/******************************************************************************
 Step 4
******************************************************************************/
cas mySession terminate;
```

   Partial Log:

```
NOTE: Deletion of the session MYSESSION was successful.
NOTE: Request to TERMINATE completed for session MYSESSION.
```

   View the full log here.