```
/* libname sqw
"/export/viya/homes/carleighjo.crabtree@sas.com/casuser/SQLDemo/choc_enter
prise"; */

/* libname out
"/export/viya/homes/carleighjo.crabtree@sas.com/casuser/SQLDemo/choc_outpu
t"; */


/*******************************/

/* Examine PROC SQL Statements */

/*******************************/


/* Print the first 100 rows of sqw.choc_enterprise_customer to explore the
data. */

/* SELECT *: Selects all columns from the input table.
*/

/* INOBS: Limits the number of observations read in for processing.
*/

proc sql inobs=100;

select *

     from sqw.choc_enterprise_customer;

quit;


/* What are the column names and attributes?                        */

/* DESCRIBE TABLE: Prints column names and attributes to the log. */

proc sql;

describe table sqw.choc_enterprise_customer;

quit;


/* What are the distinct values in the loyalty_program column? */

/* SELECT DISTINCT: Selects unique values in the column(s).    */

proc sql;

select distinct loyalty_program

     from sqw.choc_enterprise_customer;
```

```
quit;


/* Generate a list of customers that are not currently Chocolate Club
Members. */

/* WHERE clause: Filters data.
*/

proc sql;

select customer_name, email

     from sqw.choc_enterprise_customer

     where loyalty_program="Non-Member" and email is not null;

quit;


/* How many total customers are not currently Chocolate Club Members?
*/

/* COUNT(*): Returns a count of all rows in the table including null
values.            */

/* Create new columns on the SELECT clause and assign them a name after
the AS keyword. */

/* Assign column attributes to columns on the SELECT clause.
*/

proc sql;

select count(*) as TotalNonMembers format=comma16.

     from sqw.choc_enterprise_customer

     where loyalty_program="Non-Member";

quit;


/* How many Non-Members have birthdays each month?
*/

/* GROUP BY clause: Enables processing data in groups.
*/

/* COUNT(colName): Returns the number of rows that do not have a null
value.   */

/* COUNT(colName) with GROUP BY: Returns the number of rows within each
group. */
```

```
proc sql;
select bday_month label="Birthday Month", count(customer_name) as
TotalBdays format=comma10.
     from sqw.choc_enterprise_customer
     where loyalty_program="Non-Member"
     group by bday_month;
quit;


/* Filter the report for birthday months with more than 6,000 customers to
market to. */
/* HAVING clause: Works with GROUP BY to filter grouped data.
*/
proc sql;
select bday_month label="Birthday Month", count(customer_name) as
TotalBdays format=comma10.
     from sqw.choc_enterprise_customer
     where loyalty_program="Non-Member"
     group by bday_month
     having TotalBdays> 6000;
quit;


/* Which month has the highest number of birthdays? */
/* ORDER BY clause: Determines the order of rows.   */
proc sql;
select bday_month label="Birthday Month", count(customer_name) as
TotalBdays format=comma10.
     from sqw.choc_enterprise_customer
     where loyalty_program="Non-Member"
     group by bday_month
     having TotalBdays> 6000
     order by TotalBdays desc;
quit;
```

```
/*********************************************/
/* Manipulate data with Simple CASE Expression */
/*********************************************/

/* Clean the customer_name column so it only contains the customers name
without prefixes or commas.    */

/* CREATE TABLE: Creates a table rather than a report.
*/
/* FIND function: Returns the position at which the specified string
begins.                              */
/* When the value returned from the FIND function is greated than 0, the
value was found in the string. */
proc sql;
create table out.customers as
select customer_name,
        find(customer_name, 'Mrs.', 'i') as Mrs,
        find(customer_name, 'Mr.', 'i') as Mr,
        find(customer_name, 'Ms.', 'i') as Ms,
        find(customer_name, 'Miss', 'i') as Miss,
        find(customer_name, ',') as Comma,
        email, bday_month, age, customer_self_description,
loyalty_program, customer_rk
     from sqw.choc_enterprise_customer;
quit;

/* Simple CASE Expression: Used to assign values to a new column
conditionally.      */
/* TRANWRD function: Replaces substring with designated string.
*/
/* STRIP function: Removes leading and trailing blanks.
*/
```

```
/* SCAN function: Extracts the specified word within a string.
*/

/* CATX function: Concatenates specified strings and inserts delimiter
between them. */

proc sql;

create table out.customers_cleaned as

select customer_name,

     case

          when Mrs>0 then strip(tranwrd(customer_name, 'Mrs.', ' '))

          when Mr>0 then strip(tranwrd(customer_name, 'Mr.', ' '))

          when Ms>0 then strip(tranwrd(customer_name, 'Ms.', ' '))

          when Miss>0 then strip(tranwrd(customer_name, 'Miss', ' '))

          when Comma>0 then catx(' ', scan(customer_name, 2, ','),
scan(customer_name, 1, ','))

          else customer_name

     end as Customer_Names_Cleaned label="Customer Name",

     email, bday_month, age, customer_self_description, loyalty_program,
customer_rk

     from out.customers;

quit;


/*****************************/

/* Explore dictionary tables */

/*****************************/


/* View the column names and labels in dictionary.columns.
*/

/* Dictionary tables: Contain session metadata about libraries, tables,
columns etc.  */

/* DICTIONARY.columns: Provides detailed information about all columns in
all tables. */

proc sql;

describe table dictionary.columns;
```

```
quit;

/* What columns could be primary/ foreign keys for joining tables? */
proc sql;
select *
     from dictionary.columns
     where libname="SQW" and memname like "CHOC%";
quit;

/* Which tables have the item_rk column? */
proc sql;
select libname, memname, name, type, length, label
     from dictionary.columns
     where libname="SQW" and memname like "CHOC%" and name="item_rk";
quit;

/* Which tables have the customer_rk column? */
proc sql;
select libname, memname, name, type, length, label
     from dictionary.columns
     where name="customer_rk";
quit;

/*************************/
/* Perform an INNER JOIN */
/*************************/

/* Join sqw.choc_enterprise_orders, sqw.choc_enterprise_items and
out.customers_cleaned tables. */
```

```
/* Join the orders and item tables on the item_rk column to create
ordersItems.                */

/* INNER JOIN: Returns matching rows based on join criteria.
*/

proc sql;

create table out.ordersItems as

select *

     from sqw.choc_enterprise_orders as o

          inner join sqw.choc_enterprise_item as i

     on o.item_rk = i.item_rk;

quit;


/* Join out.ordersItems with out.customers_cleaned on the customer_rk
column to create out.ordersItemsCustomers. */

proc sql;

create table out.ordersItemsCustomers as

select *

     from out.ordersItems as oi

          inner join out.customers_cleaned as c

     on oi.customer_rk= c.customer_rk;

quit;


/* Join all 3 tables at the same time to create out.ordItemsCust. */

proc sql;

create table out.ordItemsCust as

select *

     from sqw.choc_enterprise_orders as o

          inner join sqw.choc_enterprise_item as i

     on o.item_rk = i.item_rk

          inner join out.customers_cleaned as c

     on o.customer_rk=c.customer_rk;
```

```
quit;


/***********************************/
/* Explore noncorrelated subqueries */
/***********************************/


/* Generate a table of the customers that are Non-Members that have
ordered greater than the average number of items ordered. */


/* Subquery in HAVING clause: Returns values to be used in the outer
query's HAVING clause.                                          */
/*                            Must return a value or values from only one
column.                                                  */
/* Noncorrelated subquery: Executes independently of the outer query.
*/


/* Step 1: Calculate the average items ordered for Non-Members older than
21. */
proc sql;
select avg(item_qty) as AvgItemsOrdered
     from out.ordItemsCust
     where loyalty_program="Non-Member" and age>21;
quit;


/* Step 2: Calculate the total number of items each customer has ordered.
*/
/*         Filter the report for customers that have ordered more than the
average number of items ordered. */
proc sql;
select distinct customer_names_cleaned, email, sum(item_qty) as
TotalItemsOrdered
     from out.ordItemsCust
     where loyalty_program="Non-Member" and age>21 and email is not null
```

```
     group by customer_name, email
     having TotalItemsOrdered> 12.58922
     order by totalitemsordered desc;
quit;

/* Step 3: Combine the query and subquery. */
proc sql;
select distinct customer_names_cleaned, email, sum(item_qty) as
TotalItemsOrdered
     from out.ordItemsCust
     where loyalty_program="Non-Member" and age>21 and email is not null
     group by customer_name, email
     having TotalItemsOrdered> (select avg(item_qty) as AvgItemsOrdered
                                from out.ordItemsCust
                                where loyalty_program="Non-
Member" and age>21)
     order by totalitemsordered desc;
quit;

/* What happens when new data is added to the table that changes the
average generated by the subquery? */

/* Add rows to out.ordersItemsCustomers to change the AverageItemsOrdered
value.                          */
/* INSERT INTO: Adds rows to an existing table.
*/
/* VALUES clause: Assign values to columns by position.
*/
proc sql;
insert into out.ordersItemsCustomers
     (Customer_Names_Cleaned, email, loyalty_program, age, item_qty,
item_desc)
```

```
     values ("Carleigh Jo Crabtree", "CarleighJo.Crabtree@sas.com", "Non-
Member", 55, 5000, "Raspberry Chocolate Permiere")

     values ("Carleigh Jo Crabtree", "CarleighJo.Crabtree@sas.com", "Non-
Member", 55, 10000, "Dark Chocolate Raspberry Starfish");

/*  values ("Carleigh Jo Crabtree", "CarleighJo.Crabtree@sas.com", "Non-
Member", 55, 25000, "4 pc White Wedding / Party Favor with White Ribbon");
*/

quit;


/* View new average. */

proc sql;

select avg(item_qty) as AvgItemsOrdered

     from out.ordersItemsCustomers

     where loyalty_program="Non-Member" and age>21;

quit;


/* Run the query.
*/

/* Uncomment the third values statement to change the average again. Re-
run the query.                          */

/* Notice the data is updated because the subquery accesses the current
data rather than a hard coded value. */

proc sql;

select distinct customer_names_cleaned, email, sum(item_qty) as
TotalItemsOrdered

     from out.ordersItemsCustomers

     where loyalty_program="Non-Member" and age>21 and email is not null

     group by customer_name, email

     having TotalItemsOrdered> (select avg(item_qty) as AvgItemsOrdered

                                          from out.ordersItemsCustomers

                                          where loyalty_program="Non-
Member" and age>21)

     order by totalitemsordered desc;
```

```
quit;


/*************************************/
/* Explore data driven macro variables */
/*************************************/


/* What is the average amount a customer will spend on one item per order?
Put the value into a macro variable.            */
/* INTO clause: Stores values as macro variables.
*/
/* Notice extra spaces in the macro. By default, macro variables with
numeric values are formatted with the BEST8. format. */
/* TRIMMED: Used to trim leading and trailing blanks.
*/
proc sql;
select round(avg(total_line_item_sale_amt)) as AvgSpent
     into :AvgSpent
/*   into :AvgSpent trimmed */
     from out.orditemscust;
quit;
%PUT The average amount a customer spends on one item per order is
&AvgSpent;


/* Use the macro variable to count how many customers spent more than the
average on sugar free items. */
proc sql ;
title "Total Number of Sugar Free Orders Where the Sale Amount was Greater
than Average";
select count(customer_names_cleaned) as SugarFreeOrders format=comma16.
     from out.orditemscust
     where category="Sugar Free" and total_line_item_sale_amt> &AvgSpent;
title;
quit;
```

```
/* Create a table with the total profit for each category within each
product line. */

proc sql;

create table out.ProductProfit as

select product_line, category, sum(total_line_item_sale_amt) as
CategoryProfit format=dollar25.

     from out.orditemscust

     group by product_line, category;

quit;


/* Store the distinct values of product_line in a series of macro
variables. */

proc sql;

select distinct product_line

     into :Product1-

     from out.orditemscust;

quit;

%PUT _USER_;


/* Use the table and macro variables created to create bar charts of the
profit by category for each product line. */

%macro CategoryProfits;


%local i;

%do i=1 %to 6;

title "Profit by Category for the &&product&i Product Line";

proc sgplot data=out.productprofit noautolegend ;

     hbar category / response=categoryprofit stat=sum group=category
categoryorder=respdesc;

     where product_line="&&product&i";

     format categoryprofit dollar8.;
```

```
run;
title;
%end;


%mend;


/* Call the macro program to view the bar charts. */
ods graphics on;
%CategoryProfits


/* Create a macro variable list of the categories in the Assorted product
line where profit is greater than $500,000. */
/* Include quotation marks around the values so the list can be used for
filtering.                                    */
proc sql;
select quote(strip(category))
     into :TopAssortedCategories separated by ","
     from out.productprofit
     where product_line="Assorted" and categoryprofit> 500000;
quit;
%PUT &=TopAssortedCategories;


/* Create a table of the items that make up the top sales in the assorted
product line. */
proc sql;
create table out.TopAssorted as
select distinct product_line, category, package, item_desc
     from out.orditemscust
     where product_line="Assorted" and category in
(&TopAssortedCategories);
quit;
```

```
/*****************************/
/* Explore summary functions */
/*****************************/

/* Create out.stateProductLines to use for summary functions.
*/
/* Calculates how many orders were placed in each state for each product
line. */
proc sql;
create table out.StateProductLines as
select distinct state_region_nm, product_line, count(product_line) as
productlinetotal format= comma20.
     from out.orditemscust
     where state_region_nm is not null
     group by state_region_nm, product_line
     order by state_region_nm, productlinetotal desc;
quit;

/* Transpose out.stateProductLines so product lines are columns. */
proc transpose data=out.stateproductlines
out=out.spl_transposed(rename=("Misc Pack"n=Misc_Pack) drop=_name_);
     var productlinetotal;
     id product_line;
     by state_region_nm;
run;

/* COALESCE function: Returns the first non-null value from a list of
arguments. */
proc sql;
update out.spl_transposed
     set
          Snacks= coalesce(Snacks, 0);
```

```
quit;


/* Summarize down a column:
*/

/* Calculate the maximum, minimum and mean number of orders placed for the
Assorted product line. */

proc sql;

select max(Assorted) as HighestAssortedOrders format= comma16.,

       min(Assorted) as LowestAssortedOrders format= comma16.,

        mean(Assorted) as AverageAssortedOrders format= comma16.

     from out.spl_transposed;

quit;


/* Summarize across rows:
*/

/* Calculate the maximum, minimum and mean number of orders placed across
all product lines for each state. */


/* When summarizing across rows, column lists are not valid. */

proc sql;

select state_region_nm,

        max(of Assorted-- Snacks) as HighestOrders format= comma16.,

       min(of Assorted-- Snacks) as LowestOrders format= comma16.,

        mean(of Assorted-- Snacks) as AverageOrders format= comma16.

     from out.spl_transposed;

quit;


/* Use data driven macros and dictionary.columns to put column names in a
macro variable list separated by commas.  */

proc sql;

select name

     into :colNames separated by ","
```

```
     from dictionary.columns
     where libname="OUT" and memname="SPL_TRANSPOSED";
quit;

%put &colNames;
%let pl=Assorted,Misc_Pack,Candy,Drinks,Books,Snacks;
%put &=pl;

/* Use the macro variable, rather than typing column names. */
proc sql;
select state_region_nm label= "State Name",
        max(&pl) as HighestOrders format= comma16.,
       min(&pl) as LowestOrders format= comma16.,
        mean(&pl) as AverageOrders format= comma16.
     from out.spl_transposed;
quit;

/*******************************/
/* Explore boolean expressions */
/*******************************/

/* How many customers in each state are Chocolate Club Members? How many
are Non-Members?                    */

/* Step 1: Use find function to determine if Chocolate Club Member is
found in the loyalty_program column. */
/*         If it is found, the value will be greater than 0. If it is not
found, the value will be 0.      */
proc sql inobs=100;
select state_region_nm label= "State Name", loyalty_program,
customer_names_cleaned,
     find(loyalty_program, "Chocolate Club Member", "i")
```

```
     from out.orditemscust
     where state_region_nm is not null;
quit;

/* Step 2: Use boolean expression.                          */
/* Boolean expression: Evaluate to true (1) or false (0).*/
proc sql inobs=100;
select state_region_nm label= "State Name", customer_names_cleaned,
          find(loyalty_program, "Chocolate Club Member", "i")>0 as
ChocolateClubMember format= comma16.,
          find(loyalty_program, "Chocolate Club Member", "i")=0 as
NonMember format= comma16.
     from out.orditemscust
     where state_region_nm is not null;
quit;

/* Step 3: Use boolean expression with SUM function to calculate how many
members or non-members are in each state. */
proc sql;
select state_region_nm label= "State Name",
          sum(find(loyalty_program, "Chocolate Club Member", "i")>0) as
ChocolateClubMembers format= comma16.,
          sum(find(loyalty_program, "Chocolate Club Member", "i")=0) as
NonMembers format= comma16.
     from out.orditemscust
     where state_region_nm is not null
     group by state_region_nm
     order by nonmembers desc;
quit;

/************************/
/* Explore SET operators */
```

```
/************************/


/* Create two tables. One for customers who ordered chocolate bars, one
for customers that ordered jelly beans. */

proc sql;

create table out.ChocolateBars as

select *

     from out.ordItemsCust

     where category="Chocolate Bars";

create table out.JellyBeans as

select *

     from out.orditemscust

     where category="Jelly Bean";

quit;


/* Create a list of customers that ordered both chocolate bars and jelly
beans.       */

/* INTERSECT operator: Result from both queries generated.
*/

/*                    Duplicate rows are removed from both intermediate
result sets. */

/*                    Rows that are in the intermediate result sets are
selected.    */

/* NUMBER: Includes the Row number column in the results.
*/

proc sql number;

title "Chocolate Bar and Jelly Bean Lovers";

select customer_names_cleaned

     from out.ChocolateBars

intersect

select customer_names_cleaned

     from out.JellyBeans;
```

```
title;
quit;

/* This can also be done with an inner join. */
proc sql number;
select distinct cb.customer_names_cleaned
     from out.chocolatebars as cb
          inner join out.jellybeans as jb
     on cb.customer_names_cleaned=jb.customer_names_cleaned;
quit;

/* Create a list of all customers that ordered chocolate bars, except
those that also ordered jelly beans.                          */
/* EXCEPT operator: Result from both queries generated.
*/
/*                  Duplicate rows are removed from both intermediate
result sets.                                                  */
/*                  Rows from the first intermediate result set that are
NOT in the second intermediate result set are selected. */
proc sql number;
title "Chocolate Bar Lovers";
select customer_names_cleaned
     from out.ChocolateBars
except
select customer_names_cleaned
     from out.JellyBeans;
title;
quit;

/* This can also be done with a subquery. */
proc sql number;
select distinct customer_names_cleaned
```

```
     from out.chocolatebars
     where customer_names_cleaned not in (select customer_names_cleaned
                                                                  from
out.jellybeans);
quit;
```