Let’s start with a quiz

Which of the following will select employee IDs having bonuses greater than $1000? Explain your answer.

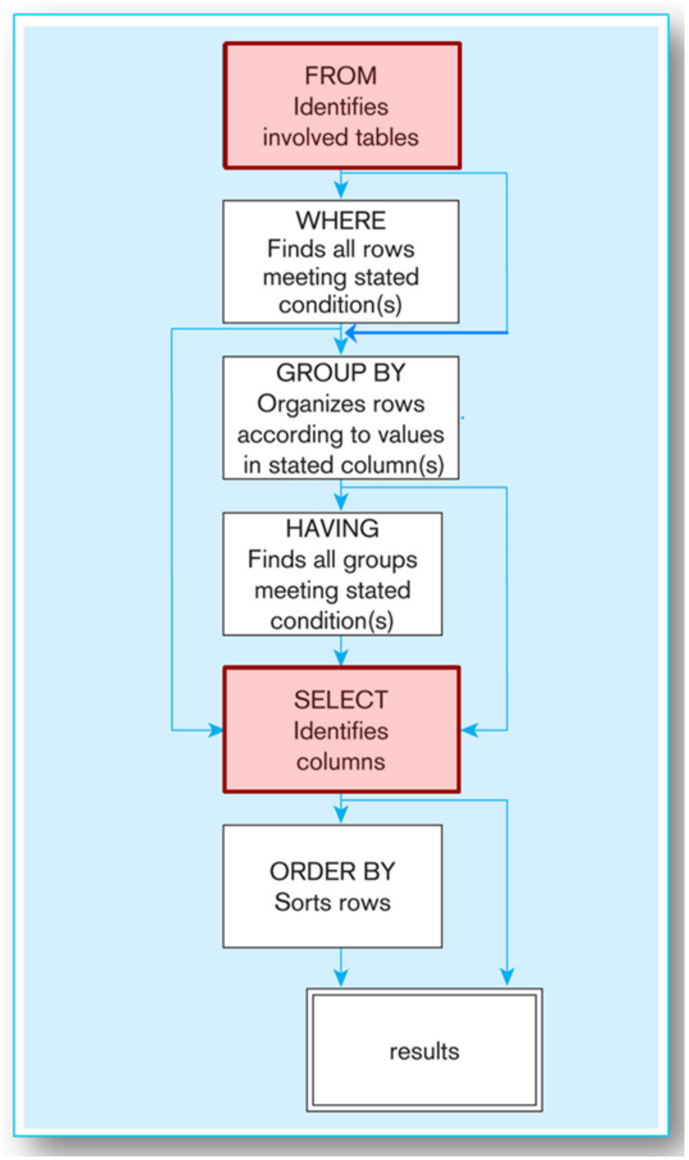
A.
```
PROC SQL;
select Employee_ID, Salary*0.1 as Bonus
        from sasuser.Employee_Payroll
  where calculated Bonus > 1000;
 Quit;
```
B.
```
PROC SQL;
select Employee_ID, Salary*0.1 as Bonus
        from sasuser.Employee_Payroll
        group by Employee_ID
        having Bonus > 1000;
  Quit;
```
C. Both of the above

D. Neither of the above

The right choice is C

The GROUP BY clause has been transformed into an ORDER BY clause because neither the SELECT clause nor the HAVING clause contains a summary function.

Also, the HAVING clause pairs with ORDER BY clause and the ORDER BY clause is executed the latest in the SQL execution sequence.


FROM
Identifies involved tables
WHERE
Finds all rows meeting stated condition(s)
GROUP BY
Organizes rows according to values in stated column(s)
HAVING
Finds all groups meeting stated condition(s)
SELECT
Identifies columns
ORDER BY
Sorts rows
results

Assign datasets to SASUSER library and practice the quiz

- Find the datasets on Blackboard under Data Files.
- Download and assign these to the SASUSER library that belongs to SAS.
- How many ways can you use to accomplish this task?
- Practice the quiz to confirm your answer.

Create tables in SAS

- Create a table from an existing table(s) and/or view(s). This is pretty common in PROC SQL. However, there is no guarantee that the table created has integrity constraints.
- Create a table that has integrity constraints
  - With PROC DATASETS: Constraints can be only assigned to an existing table.
  - With PROC SQL: Constraints can be assigned either as a new table is created or as an existing table is modified. This approach is the same as we have introduced earlier in general SQL, e.g., Oracle SQL*PLUS.

Create a table from an existing table(s)

- Create a table from a query result (which we covered earlier).

  CREATE TABLE Table_Name **AS**

  SELECT…**FROM**…

- Create a new empty table with the same columns and attributes as an existing table by using

  CREATE TABLE…LIKE…

  (Oracle does not support this)

- Create a new empty table that only contains a subset of columns of an existing table by using **CREATE TABLE…(DROP|KEEP=…) LIKE…**

CREATE TABLE…LIKE…

```
proc sql;
        CREATE TABLE work.flightdelays2
              LIKE sasuser.flightdelays;
quit;
```

```
proc sql;
  describe table
       work.flightdelays2;
quit;
```

```
create table WORK.FLIGHTDELAYS2(
bufsize=8192 )
  (
   Date num format=DATE9. informat=DATE9.,
   FlightNumber    char(3),
   Origin           char(3),
   Destination      char(3),
   DelayCategory   char(15),
   DestinationType char(15),
   DayOfWeek       num,
   Delay            num
  );
```

```
proc sql;
select * from work.flightdelays2;
quit;
```

```
NOTE: No rows were selected
```

CREATE TABLE…(DROP|KEEP=…) LIKE…

```
proc sql;
        CREATE TABLE work.flightdelays3
          (drop=delaycategory destinationtype)
               LIKE sasuser.flightdelays;
quit;
```

Note: separated by a space

```
proc sql;
  describe table
        work.flightdelays3;
quit;
```

```
create table WORK.FLIGHTDELAYS3(
bufsize=4096 )
  (
   Date num format=DATE9. informat=DATE9.,
   FlightNumber    char(3),
   Origin           char(3),
   Destination      char(3),
   DayOfWeek        num,
   Delay            num
  );
```

```
proc sql;
select * from work.flightdelays3;
quit;
```

```
NOTE: No rows were selected
```

Note: DROP= and KEEP= options are not part of ANSI and are unique to SAS PROC SQL

Note the difference

In PROC SQL

```
proc sql;
  describe table  flightdelays3;
quit;
```

In SQL*PLUS

```
describe flightdelays3
```

Practice

- Use CREATE TABLE…(KEEP=…) LIKE…to create a new empty table called Employee_payroll_T1 in the Work library based on the first four columns of the Employee_payroll in the SASUSER library.
- Create a new table called Employee_payroll_T2, which is identical to Employee_payroll_T1, by using a query.

Common comparison operators

Mnemonic	Symbol	Definition
LT	<	Less than
GT	>	Greater than
EQ	=	Equal to
LE	<=	Less than or equal to
GE	>=	Greater than or equal to
NE	< > ^=	Not equal to Not equal to (ASCII)

You can use the mnemonic or the symbols in PROC SQL

Logical operators

Mnemonic	Symbol	Definition
OR	|	or, either
AND	&	and, both
NOT	^	not, negation (ASCII)

You can use the mnemonic or the symbols in PROC SQL

The concatenation operator || (Review)

It joins two strings into one string. For example,

```
Name = FirstName || ' ' || LastName;

Name = 'John' || ' ' || 'Smith';
```

Common WHERE clause operators with examples

Operator	Example
IN	`where JobCategory in ('PT','NA','FA')`
CONTAINS or ?	`where word ? 'LAM'`
IS NULL or IS MISSING	`where Product_ID is missing`
BETWEEN – AND	`where Salary between 70000 and 80000`
SOUNDS LIKE (=*)	`where LastName =* 'SMITH'`

LIKE using % or _	`where Employee_Name like 'H%'`
	`where JobCategory like '__1'`

The CONTAINS or ? operator

Example:

```
proc sql outobs=10;
select name
        from sasuser.frequentflyers
        where name ? 'ER';
```

Name
COOPER, LESLIE
COOPER, ANTHONY
COOK, JENNIFER
FOSTER, GERALD
BRADLEY, JEREMY
BURKE, CHRISTOPHER
AVERY, JERRY
EDGERTON, JOSHUA
SAYERS, RANDY
WANG, CHRISTOPHER

The Sounds-Like (=*) operator

Example:

```
proc sql ;
select customer_id, country, gender,
customer_name
  from sasuser.customer
  where customer_firstname =* 'Jim';
quit;
```

=

```
proc sql ;
select customer_id, country, gender,
customer_name
  from sasuser.customer
  where SOUNDEX(customer_firstname)
   = SOUNDEX('Jim');
quit;
```

Customer ID	Customer Country	Customer Gender	Customer Name
17	US	M	Jimmie Evans

Note: In Oracle, you can use either

```
where customer_firstname SOUNDS LIKE 'Jim'
```

Or

```
where SOUNDEX(customer_firstname) = SOUNDEX('Jim')
```

A Review of some SAS character functions

- SCAN()
  - Returns a word from a character value
- SUBSTR()
  - Extract a substring or replaces character values
- TRANWRD()
  - Replaces or removes all occurrences of a pattern of characters within a character string

The SCAN() function

Syntax: SCAN(argument,n,delimiters)

where

argument specifies the character value, variable or expression to scan

n specifies which word to return; if the value of n is negative, the SCAN() function selects the word in the character string starting from the end of the string.

delimiters are special characters that must be enclosed in quotation marks. A blank and a comma are default delimiters.

Example:

```
SCAN('Ithaca, NY, 14853', -3, ' , ')
```

What is the result that the SCAN() function will return?

The SUBSTR() function

- **SUBSTR**(*argument,position,<n>*)

  where

  argument specifies the character value, variable or expression to scan.

  position is the character position to start from.

  n specifies the number of characters to extract. If *n* is omitted, all remaining characters are included in the substring.
- When the function is on the **right side** of an assignment statement, the function returns the requested string.

```
Example:
        City = SUBSTR('Ithaca, NY, 14853', 1, 6);
```

What will the SUBSTR() function return?

The SUBSTR() function

The SUBSTR() function can be used to replace the character values if you place the function on the **left side** of an assignment statement.

Example:

```
new_string = 'Ithaca, NY, 14853';
SUBSTR(new_string, 13, 5) = '14850'
```

What is the value of new_string after you execute the above statement?

The TRANWRD() function

TRANWRD(*source*,*target*,*replacement*)

where

source specifies the source string that you want to update

target specifies the string that SAS searches for in *source*

replacement specifies the string that replaces *target*.

Example:

```
phonenumber=TRANWRD(phonenumber, '607', '608');
```

What happens to the values of the phonenumber variable?

Update values in existing table rows

The syntax:

```
proc sql;
  update table_name
    set column =…
    where…;
```

Example:

```
proc sql;
  update work.payrollmaster_new
    set salary=salary*1.05
    where jobcode like '__1';
```

Practice: using PROC SQL