Lec1

Proc SQL with SAS

- **SQL**: STRUCTURED QUERY LANGUAGE (SQL)
- It is the standard language for relational database management systems
- In SAS, SQL is built in PROC SQL, which is an interactive procedure, so you can run it statement by statement (rather than a whole step at a time). SQL stays active until you issue the QUIT statement or the next step boundary is hit. RUN has no effect after PROC SQL
- SQL can be completed replaced by SAS data/proc steps, but under certain circumstance, SQL is more efficient

A SQL query could be:

```
PROC SQL;
CREATE TABLE table_name AS
SELECT variables FROM overview
WHERE condition
GROUP BY variables
HAVING condition
ORDER BY variables;
```

- Unlike other SAS procedures the order of clauses with a SELECT statement in PROC SQL is important. Clauses must **appear in the order shown above**.

- Variables can be renamed (or gives results a name) by using the AS option between the old name and the new name. In the SELECT clause, you can both specify existing columns (columns that are already stored in a table) and create new columns.

```
select country, year, gdp AS Gross_domestic_production from overview
where year=2001 and pop>3000;
```

- When you are using PROC SQL, you might find that the data in a table is not formatted as you would like it to appear. In PROC SQL you can use enhancements to improve the appearance of your query output:

- column labels and formats
- titles and footnotes

Column Modifier	Specifies...	Example
LABEL=	the label to be displayed for the column	select hiredate label='Date of Hire'
FORMAT=	the format used to display column data	select hiredate format=date9.

Example 1:

Given the SAS dataset WORK.ONE

```
 Name     Salary
 -----   ------
 Hans      200
 Maria     205
 Jose      310
 Ariel     523
```

The following SAS program is submitted:

```
proc sql;
  [_insert_select_clause_]
  from WORK.ONE
  ;
quit;
```

The following output is desired:

```
  Salary      Bonus
  ------     -----
    200         20
    205        20.5
    310         31
    523        52.3
```

Which SQL procedure clause completes the program and generates the desired output?

A. select Salary Bonus as Salary*.10

B. select Salary Bonus=Salary*.10 'Bonus'

C. select Salary, Salary*.10 label='Bonus'

D. select Salary, Salary*.10 column="Bonus"

- The DISTINCT keyword removes duplicate observations based on the variables named. The DISTINCT keyword applies to all columns, and only those columns, that are listed in the SELECT clause.

```
select distinct country, year from overview;
```

Example 2:
To create a list of unique Customer_Id values from the customer data set, which of the following techniques can be used?

technique 1: proc SORT with NODUPKEY and OUT=
technique 2: data step with IF FIRST.Customer_Id=1
technique 3: proc SQL with the SELECT DISTINCT statement

A. only technique 1
B. techniques 1 and 2
C. techniques 1 and 3
D. techniques 1, 2, or 3

- The * operator selects all variables, one can use the FEEDBACK option in the PROC SQL statement, which writes the expanded list of columns to the SAS log.

```
SELECT * FROM overview WHERE pop>1000;
```

- In the WHERE clause, you can specify any column(s) from the underlying table(s). The columns specified in the WHERE clause **do not have to be specified in the SELECT clause**.

- You can also use a calculated column in the WHERE clause to subset rows. However, because of how SQL queries are processed, you cannot just specify the column alias in the WHERE clause.

```
proc sql outobs=10;
select flightnumber, date, destination,
       boarded + transferred + nonrevenue as Total
from sasuser.marchflights
where calculated total < 100;
```

- When you use a column alias in the WHERE clause to refer to a calculated value, you must use the keyword CALCULATED along with the alias.

- You can also use the CALCULATED keyword in other parts of a query. To create the second calculated column, you have to specify the keyword CALCULATED in the SELECT clause.

```
proc sql outobs=10;
select flightnumber, date, destination,
       boarded + transferred + nonrevenue as Total,
       calculated total/2 as Half
from sasuser.marchflights;
```

Note: Multiple variables are delimited by a comma.

Example 3:

Given the SAS dataset WORK.ONE:

```
       Salary
      ------
         200
         205
           .
         523
```

The following SAS program is submitted:

```
proc sql;
   select *
   from WORK.ONE
   [_insert_where_clause_]
   ;
quit;
```

The following output is desired:

```
      Salary
      ------
         200
         205
         523
```

Which WHERE expression completes the program and generates the desired output?

A. where Salary is not.
B. where Salary ne missing
C. where Salary ne null
D. where Salary is not missing

- Use the DESC option with ORDER BY to reverse order data.
- In the ORDER BY clause, you can alternatively reference a column by the **column's position** in the SELECT clause list rather than by name. Use an integer to indicate the column's position.
- You can mix the two types of column references, names and numbers, in the ORDER BY clause.

```
select country, year, gdp from overview where pop>2400 order by
country /*1*/, year desc;
```

- SQL Functions:
- SQL elementary functions: **Log(), substr(), upcase(), trim(), compress(), sqrt(), abs()**
- SQL summary functions: **min(), max(), mean(), sum(), var()**

Can be used in the **SELECT, WHERE and Having** clauses

- SQL summary functions are similar to those in the data step except that if you only name one variable as an argument then they act across rows (observations) rather than across columns (variables)
- In SQL, mean(Var1) and mean(var1, var2, var3) work differently

- The **GROUP BY** clause allows you to apply the **summary functions** to a group of observations (the same as CLASS statement in PROC MEANS), you only need it when you apply **summary functions**

- If you specify a GROUP BY clause in a query that does not contain a summary function, your clause is changed to an ORDER BY clause, and a message to that effect is written to the SAS log.

- The **HAVING** clause allows you select out certain groups based on their summary values

```
SELECT country, min(gdp) FROM overview WHERE pop >2400 GROUP BY
country HAVING count(*)>=2;
```

- The COUNT function is only available in PROC SQL.

If a GROUP BY clause…	Then PROC SQL…
is not present in the query	applies the function to the entire table
is present in the query	applies the function to each group specified in the GROUP BY clause

If a summary function…	Then the calculation is…	Example

specifies one column as argument	performed down the column	proc sql;
		select avg(salary)as AvgSalary;
		from sasuser.payrollmaster;

specifies multiple columns as arguments	performed across columns for each row	proc sql outobs=10;
		select sum(boarded,transferred,nonrevenue) as Total
		from sasuser.marchflights;

If a SELECT clause…	Then PROC SQL…	Example
contains summary function(s) and no columns outside of summary functions	calculates a single value by using the summary function for the entire table or, if groups are specified in the GROUP BY clause, for each group combines or rolls up the information into a single row of output for the entire table or, if groups are specified, for each group	proc sql; select avg(salary) as AvgSalary from

contains summary function(s) and additional columns outside of summary functions	calculates a single value for the entire table or, if groups are specified, for each group, and displays all rows of output with the single or grouped value(s) repeated	proc sql; select jobcode, gender, avg(salary) as
		from sasuser.payrollmaster
		group by jobcode,gender;

Example 4:

Given the SAS data set WORK.TRANSACT:

```
 Rep        Cost  Ship
-----     -----  -----
SMITH       200     50
SMITH       400     20
JONES       100     10
SMITH       600    100
JONES       100      5
```

The following output is desired:

```
Rep
-----   ----
JONES    105
SMITH    250
```

Which SQL statement was used?

```
A. select
   rep,
   min(Cost+Ship)
from WORK.TRANSACT
order by Rep
;
```

```
B. select
   Rep,
   min(Cost,Ship) as Min
from WORK.TRANSACT
summary by Rep
order by Rep
;
```

```
C. select
   Rep,
   min(Cost,Ship)
from WORK.TRANSACT
group by Rep
order by Rep
;
```

```
D. select
   Rep,
   min(Cost+Ship)
from WORK.TRANSACT
group by Rep
order by Rep
;
```

Example 5:

Given the SAS data set WORK.ONE:

```
  Rep        Cost
 -----       ----
 SMITH       200
 SMITH       400
 JONES       100
 SMITH       600
 JONES       100
```

The following SAS program is submitted:

```
proc sql;
  select
     Rep,
     avg(Cost) as Average
  from WORK.ONE
  [either__insert_SQL_where_clause_]
  group by Rep
  [_or_ _insert_SQL_having_clause_]
  ;
quit;
```

The following output is desired:

```
 Rep      Average
 -----     -------
 SMITH       400
```

Which SQL clause completes the program and generates the desired output?

A. where calculated Average > (select avg(Cost) from WORK.ONE)
B. having Average > (select avg(Cost) from WORK.ONE)
C. having avg(Cost) < (select avg(Cost) from WORK.ONE)
D. where avg(Cost) > (select avg(Cost) from WORK.ONE)

Combining Tables Horizontally Using Proc SQL

➢ Review of data step merge

```
Data XY;
  Merge X(in=a) Y(in=b);
  By key1 key2;
  If a and b;
Run;
```

- All data sets need to be sorted before merging
- All data sets require common BY variables (BY variables have the same name in all data sets)
- **in= option** is used to track where the observations come from

➢ SQL Join is comparable to data step merges except it is more flexible

- All data sets need to be sorted beforehand – NOT required
- All data sets require common BY variables – NOT required

➢ Inner Join and Outer Join

- Inner Join: only observations existing in both data sets are kept
- Outer Join: **left join, right join and full join** – more observations will be kept than inner join
- When any type of join is processed, PROC SQL starts by generating a **Cartesian product**, which contains all possible combinations of rows from all tables.

➢ Inner Join using SQL

```
SELECT a.*, b.*
FROM disk.multiple_financial as a, disk.multiple_rating as b
WHERE a.id=b.id and a.year=b.year;
```

➢ As in the SELECT clause, you separate names in the FROM clause (in this case, table names) with commas.

```
SELECT a.*, b.*
FROM disk.multiple_financial as a INNER JOIN disk.multiple_rating as b
ON a.id=b.id and a.year=b.year;
```

- You can give data sets an alias so that you can refer to their alias rather than their actual name

- If the same variable is present in more than one table, then you must prefix its name with the name of the table it comes from whenever you reference it

➢ Outer Join using SQL

```
SELECT a.*, b.*
FROM disk.multiple_financial as a LEFT JOIN disk.multiple_rating as b
ON a.id=b.id and a.year=b.year;
```

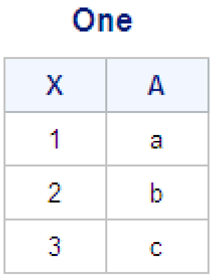
```
SELECT a.*, b.*
FROM disk.multiple_financial as a RIGHT JOIN disk.multiple_rating as b
ON a.id=b.id and a.year=b.year;
```

```
SELECT a.*, b.*
FROM disk.multiple_financial as a FULL JOIN disk.multiple_rating as b
ON a.id=b.id and a.year=b.year;
```

- Table aliases are usually optional. However, there are two situations that require their use, as shown below:
- A table is joined to itself
- You need to reference columns from same-named tables in different libraries

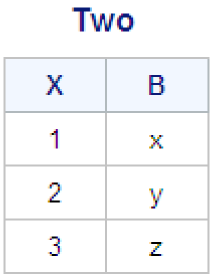
- Compare the use of SQL joins and DATA step match-merges in the following situations:

- When all the values of the selected variable (column) match



One

X	A
1	a
2	b
3	c



Two

X	B
1	x
2	y
3	z

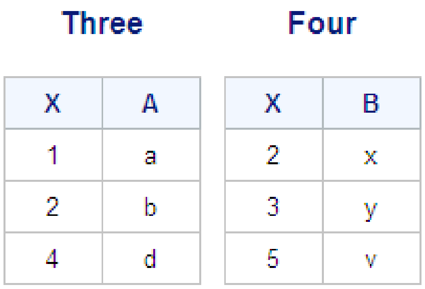
DATA Step Match-Merge

```
data merged;
merge one two;
by x; run;
proc print data=merged
noobs;
title 'Table Merged';
run;
```

PROC SQL Inner Join

```
proc sql;
title 'Table Merged';
select one.x, a, b
from one full join two
on one.x = two.x
order by x
;
```

- when only some of the values of the selected variable (column) match.

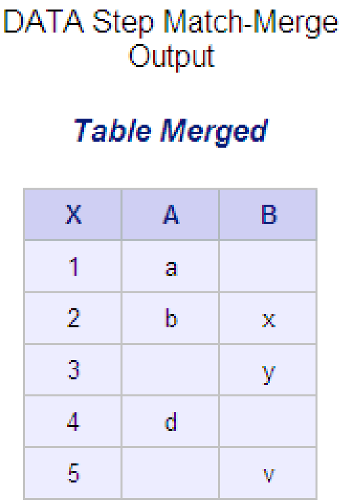


Three

X	A
1	a
2	b
4	d

Four

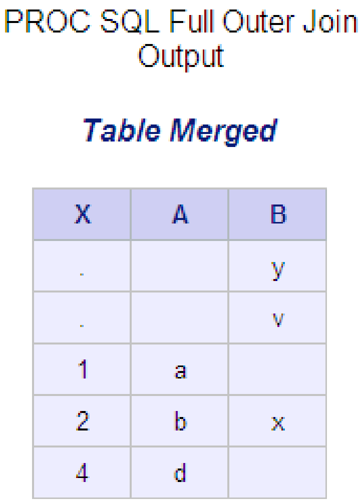
X	B
2	x
3	y
5	v



DATA Step Match-Merge Output

Table Merged

X	A	B
1	a	
2	b	x
3		y
4	d	
5		v



PROC SQL Full Outer Join Output

Table Merged

X	A	B
.		y
.		v
1	a	
2	b	x
4	d	

DATA Step Match-Merge

```
data merged;
merge three four;
by x;
run;
proc print data=merged
noobs;
title 'Table Merged';
run;
```

PROC SQL Full Outer Join

```
proc sql;
title 'Table Merged';
select three.x, a, b
from three
full join
four
on three.x = four.x
order by x;
```

- ➢ PROC SQL outer join does not overlay the two common columns by default. To overlay common columns, COALESCE function need to be used in the PROC SQL full outer join.

- ➢ The COALESCE function overlays the specified columns by checking the value of each column in the order in which the columns are listed returning the first value that is a SAS nonmissing value.

Note: If all returned values are missing, COALESCE returns a missing value.

SELECT COALESCE *(column-1<,...column-n>)*

- The COALESCE function requires that all arguments have the same data type.

- The COALESCE function overlays the specified columns by
  1. checking the value of each column in the order in which the columns are listed
  2. returning the first value that is a SAS nonmissing value.

```
data merged;
merge three four;
by x;
run;
proc print data=merged
noobs;
title 'Table Merged';
run;
```

```
proc sql;
title 'Table Merged';
select
coalesce(three.x,
four.x)
as X, a, b
from three
full join
four
on three.x = four.x;
```

Example 6:
Given the SAS data sets:

```
 WORK.ONE          WORK.TWO

 Id   Name          Id    Salary
 ---  ------        ---   ------
 112  Smith         243   150000
 243  Wei           355    45000
 457  Jones         523    75000
```

The following SAS program is submitted:

```
data WORK.COMBINE;
merge WORK.ONE WORK.TWO;
by Id;
run;
```

Which SQL procedure statement produces the same results?

```
A. create table WORK.COMBINE as
select
  Id,
  Name,
  Salary
from
  WORK.ONE
  full join
  WORK.TWO
on ONE.Id=TWO.Id
;
```

```
B. create table WORK.COMBINE as
select
   coalesce(ONE.Id, TWO.Id) as Id,
   Name,
   Salary
from
   WORK.ONE,
   WORK.TWO
where ONE.Id=TWO.Id
;
```

```
C. create table WORK.COMBINE as
select
   coalesce(ONE.Id, TWO.Id) as Id,
   Name,
   Salary
from
   WORK.ONE
   full join
   WORK.TWO
on ONE.Id=TWO.Id
order by Id
;
```

```
D. create table WORK.COMBINE as
select
   coalesce(ONE.Id, TWO.Id) as Id,
   Name,
   Salary
from
   WORK.ONE,
   WORK.TWO
where ONE.Id=TWO.Id
order by ONE.Id
;
```

Example 7:
Given the SAS data sets:

```
  WORK.ONE                WORK.TWO

  Year  Qtr  Budget      Year  Qtr  Sales
  ----  ---  ------       ----  ---  -----
  2001    3     500       2001    4    300
  2001    4     400       2002    1    600
  2003    1     350
```

The following SAS program is submitted:

```
 proc sql;
```

```
    select
      TWO.*,
      budget
    from
      WORK.ONE
      [_insert_join_operator_]
      WORK.TWO
    on ONE.Year=TWO.Year
    ;
  quit;
```

The following output is desired:

```
Year  Qtr  Sales   Budget
----  ---  -----   ------
2001    4    300      500
2001    4    300      400
2002    1    600        .
   .    .      .      350
```

Which join operator completes the program and generates the desired output?

A. left join
B. right join
C. full join
D. outer join

Combining Tables Vertically Using Proc SQL

➢ Set Operators: **UNION, INTERSECT and EXCEPT, OUTER UNION**

➢ Keyword: **ALL, CORR**

In a PROC SQL set operation, you use one of four set operators (EXCEPT, INTERSECT, UNION, and OUTER UNION) to combine tables (and views) vertically by combining the results of two queries:

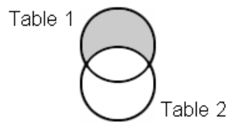
```
proc sql;
select *
from a
set-operator
select *
from b;
```

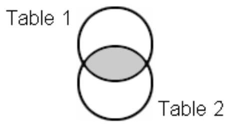
Using Multiple Set Operators

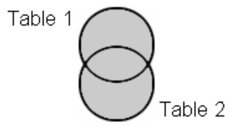
```
proc sql;
select *
from table1
set-operator
select *
from table2
set-operator
```

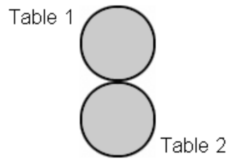
```
select *
from table3;
```

Set Operator	Treatment of Rows	Treatment of Columns	Example	

EXCEPT	Selects unique rows from the first table that are not found in the second table.	Overlays columns based on their position in the SELECT clause
		without regard to the individual column names.
			proc sql;
			select *
			from table1
			except
			select *
			from table2
				

INTERSECT	Selects unique rows that are common to both tables.	Overlays columns based on their position in the SELECT clause
		without regard to the individual column names.
			proc sql;
			select *
			from table1
			intersect
			select *
			from table2;
				

UNION	Selects unique rows from both tables.	Overlays columns based on their position in the SELECT clause
		without regard to the individual column names.
			proc sql;
			select *
			from table1
			union
			select *
			from table2;
				

OUTER UNION	Selects all rows from both tables.
	The OUTER UNION operator concatenates the results of the queries.
		Does not overlay columns.	proc sql;
			select *
			from table1
			outer union
			select *
			from table2;
				

Note: A set operator that selects only unique rows displays one occurrence of a given row in output.

When processing a set operation that displays only unique rows (a set operation that contains the set operator EXCEPT, INTERSECT, or UNION), PROC SQL makes two passes through the data, by default:

1. PROC SQL eliminates duplicate (nonunique) rows in the tables.
2. PROC SQL selects the rows that meet the criteria and, where requested, overlays columns.

For set operations that display both unique and duplicate rows, only one pass through the data (step 2 above) is required.

Three of the four set operators (EXCEPT, INTERSECT, and UNION) combine columns by overlaying them. (The set operator OUTER UNION does not overlay columns.)

When columns are overlaid, PROC SQL uses the column name from the **first table** (the table referenced in the first query). If there is no column name in the first table, the column name from the second table is used.

Keyword	Action	Used When...
ALL	Makes only one pass through the data and does not remove duplicate rows.	You do not care if there are duplicates. Duplicates are not possible. ALL cannot be used with OUTER UNION.
CORR (or CORRESPONDING)	Compares and overlays columns by name instead of by position: When used with EXCEPT, INTERSECT, and UNION, removes any columns that do not have the same name in both tables. When used with OUTER UNION, overlays same-named columns and displays columns that have nonmatching names without overlaying. If an alias is assigned to a column in the SELECT clause, CORR use the alias instead of the permanent column name.	Two tables have some or all columns in common, but the columns are not in the same order.

- To modify the behavior of set operators, you can use either or both of the keywords ALL and CORR immediately following the set operator:

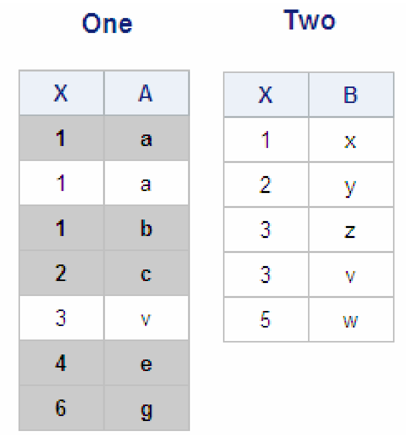
```
proc sql;
select *
from table1
set-operator <all> <corr>
select *
from table2;
```

```
proc sql;
select *
from one
except
select *
from two;
```

```
proc sql;
select *
from one
except all
select *
from two;
```

```
proc sql;
select *
from one
except corr
select *
from two;
```

```
proc sql;
select *
from one
except all corr
select *
from two;
```

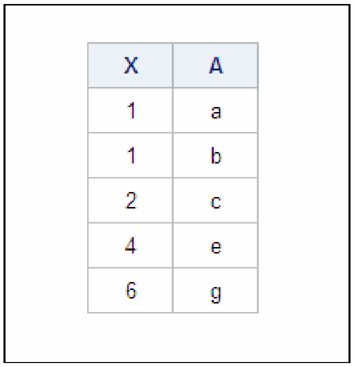


One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w



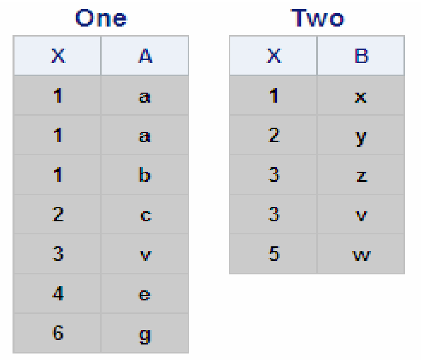
X	A
1	a
1	b
2	c
4	e
6	g

- The set operator OUTER UNION concatenates the results of the queries by the following:
  - selecting all rows (both unique and nonunique) from both tables
  - not overlaying columns.

- The ALL keyword is not used with OUTER UNION because this operator's default action is to include all rows in output.

```
proc sql;
select *
from one
outer union
select *
from two;
```

```
proc sql;
select *
from one
outer union corr
select *
from two;
```

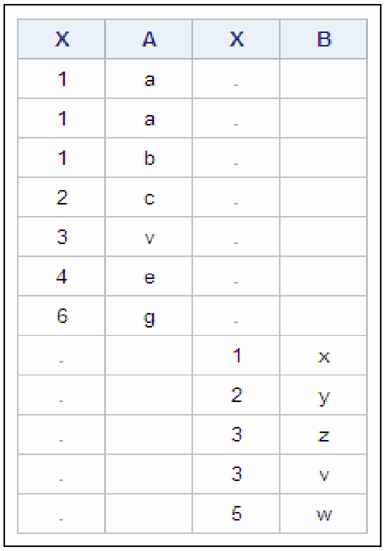


One

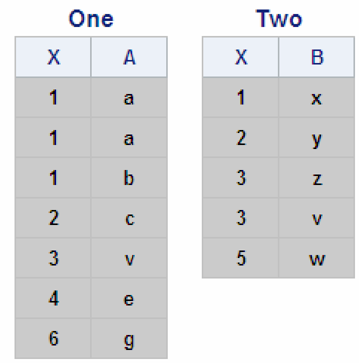
X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w



X	A	X	B
1	a	.	
1	a	.	
1	b	.	
2	c	.	
3	v	.	
4	e	.	
6	g	.	
.		1	x
.		2	y
.		3	z
.		3	v
.		5	w

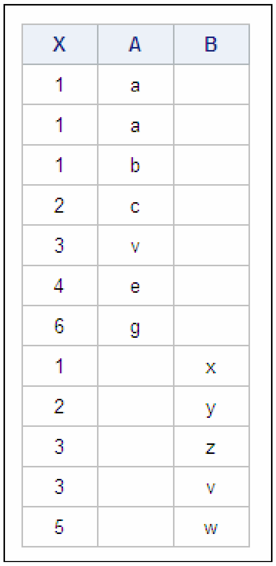


One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w



X	A	B
1	a	
1	a	
1	b	
2	c	
3	v	
4	e	
6	g	
1		x
2		y
3		z
3		v
5		w

Example 8:

Given the SAS data sets:

```
  WORK.MATH1A        WORK.MATH1B
  Name    Fi            Name   Fi
 -------  --            ------  --
 Lauren   L              Smith   M
 Patel    A              Lauren  L
 Chang    Z                Patel   A
 Hillier  R
```

The following SAS program is submitted:

```
  proc sql;
    select *
    from WORK.MATH1A
    [_insert_set_operator_]
    select *
    from WORK.MATH1B
    ;
 quit;
```

The following output is desired:

```
     Name     Fi
     -------  --
     Lauren   L
     Patel    A
     Chang    Z
     Hillier  R
     Smith    M
     Lauren   L
     Patel    A
```

Which SQL set operator completes the program and generates the desired output?

A. append corr

B. union corr

C. outer union corr

D. intersect corr

Example 9:

Given the following SAS data sets:

```
 WORK.VISIT1     WORK.VISIT2

 Id    Expense        Id    Cost
 ---   -------        ---   ----
 001       500      001    300
 001       400      002    600
 003       350
```

The following result set was summarized and consolidated using the SQL procedure:

```
Id    Cost
---   ----
001    300
001    900
002    600
003    350
```

Which of the following SQL statements was most likely used to generate this result?

```
A. select
   Id,
   sum(Expense) label='Cost'
from WORK.VISIT1
group by 1
union all
select
   Id,
   sum(Cost)
from WORK.VISIT2
group by 1
order by 1,2
;

B.
 select
   id,
   sum(expense) as COST
from
   WORK.VISIT1(rename=(Expense=Cost)),
   WORK.VISIT2
where VISIT1.Id=VISIT2.Id
group by Id
order by Id,Cost
;

C.
 select
   VISIT1.Id,
   sum(Cost) as Cost
from
   WORK.VISIT1(rename=(Expense=Cost)),
   WORK.VISIT2
where VISIT1.Id=VISIT2.Id
group by Id
order by Id,Cost
;
D.
 select
   Id,
   sum(Expense) as Cost
```

```
from WORK.VISIT1
group by Id
outer union corr
select
  Id,
  sum(Cost)
from WORK.VISIT2
group by Id
order by 1,2
;
```

Example 10:

Given the SAS data sets:

```
  WORK.CLASS1          WORK.CLASS2
  Name    Course        Name     Class
 ------  ------        -------  -----
 Lauren  MATH1          Smith    MATH2
 Patel   MATH1         Farmer   MATH2
 Chang   MATH1          Patel    MATH2
 Chang   MATH3          Hillier  MATH2
```

The following SAS program is submitted:

```
proc sql;
   select Name
   from WORK.CLASS1
   [_insert_set_operator_]
   select Name
   from WORK.CLASS2
   ;
quit;
```

The following output is desired:

```
      Name
      ------
      Chang
      Chang
      Lauren
```

Which SQL set operator completes the program and generates the desired output?

A. intersect corr
B. except all
C. intersect all
D. left except

Example 11:

Given the SAS data sets:

```
WORK.CLASS1          WORK.CLASS2

Name     Course         Name     Class
------   ------          ------   -----
Lauren   MATH1          Smith    MATH2
Patel    MATH1         Farmer   MATH2
Chang    MATH1           Patel    MATH2
                         Hillier  MATH2
```

The following SAS program is submitted:

```
proc sql;
   select Name
   from WORK.CLASS1
   [_insert_set_operator_]
   select Name
   from WORK.CLASS2
   ;
quit;
```

The following output is desired:

```
Name
------
Chang
Lauren
```

Which SQL set operator completes the program and generates the desired output?

A. intersect corr
B. except
C. intersect
D. left except

Nested Query

- SQL allows you to embed a query into another query, subquery can be uncorrelated or correlated
- Uncorrelated subquery

```
 select * from disk.multiple_financial where ID in
(select ID from disk.multiple_rating where df=1 and year=2002);

select * from disk.multiple_financial where exists
      (select ID from disk.multiple_rating where df=1 and year=2034);
```

- Correlated Subquery

A sub-query that uses values from the outer query. In this case the inner query has to be executed for every row of outer query.

```
select * from disk.multiple_financial as a where exists
(select * from disk.multiple_rating as b where a.id=b.id and
a.year=b.year and b.df=1);
```

Sometimes it is helpful to compare a value with a set of values returned by a subquery.

When a subquery might return multiple values, you must use one of the conditional operators **ANY or ALL** to modify a comparison operator in the **WHERE or HAVING clause immediately before the subquery**. For example, the following WHERE clause contains the less than (<) comparison operator and the conditional operator ANY:

```
where dateofbirth < ANY {subquery...}
where dateofbirth < ALL {subquery...}
```

CAUTION: If you create a noncorrelated subquery that returns multiple values, and if the WHERE or HAVING clause in the outer query contains a comparison operator that is not modified by ANY or ALL, the query fails.

- ➢ When the outer query contains a comparison operator that is modified by ANY or ALL, the outer query compares each value that it retrieves against the value(s) returned by the subquery. All values for which the comparison is true are then included in the query output.
- If ANY is specified, then the comparison is true if it is true for any one of the values that are returned by the subquery.
- If ALL is specified, then the comparison is true only if it is true for all values that are returned by the subquery.

Example 12:

Given the SAS data sets:

```
 WORK.EMPLOYEE          WORK.NEWEMPLOYEE

  Name      Dept        Names     Salary
  --------  -----       --------  ------
  Alan      Sales       Michelle  50000
  Michelle  Sales         Paresh  60000
```

A SAS program is submitted and
the following is written to the SAS log:

```
101  proc sql;
102    select dept, name
103    from WORK.EMPLOYEE
104    where name=(select names
                from newemployee
                where salary > 40000)
ERROR: Subquery evaluated to more than one row.
105  ;
```

```
106  quit;
```

What would allow the program to successfully execute without errors?

A. Replace the where clause with:

```
where EMPLOYEE.Name=(select Names delimited with ','
                from WORK.NEWEMPLOYEE
                where Salary > 40000);
```

B. Replace line 104 with:

```
where EMPLOYEE.Name =ANY (select Names separated with ','
                    from WORK.NEWEMPLOYEE
                    where Salary > 40000);
```

C. Replace the equal sign with the IN operator.
D. Qualify the column names with the table names.

Using In-Line Views

- ➢ An in-line view is a nested query that is specified in the outer query's FROM clause.
- ➢ Sometimes, you might want to specify an in-line view rather than a table as the source of data for a PROC SQL query.
- ➢ An in-line view selects data from one or more tables in order to produce a temporary (or virtual) table that the outer query then uses to select data for output.

```
from (select flightnumber, date,
             boarded/passengercapacity*100 as pctfull
             format=4.1 label='Percent Full'
        from sasuser.marchflights)
```

This in-line view selects two existing columns (FlightNumber and Date) and defines the new column PctFull based on the table Sasuser.Marchflights.

- An in-line view exists only during query execution.
- Because it is temporary, an in-line view can be referenced only in the query in which it is defined.
- In addition, an in-line view can be assigned an alias but it cannot be assigned a permanent name.

***Note:* Unlike other queries, an in-line view cannot contain an ORDER BY clause.**

- ➢ There are two potential advantages to using an in-line view instead of a table in a PROC SQL query:

1. The complexity of the code is usually reduced, so that the code is easier to write, and understand.

2. In some cases, PROC SQL might be able to process the code more efficiently.

- Referencing an In-Line View with Other Views or Tables

```
from sasuser.flightschedule,
    (select flightnumber, date,
            boarded/passengercapacity*100 as pctfull
            format=4.1 label='Percent Full'
       from sasuser.marchflights)
```

- Referencing Multiple Tables in an In-Line View

```
from (select marchflights.flightnumber, marchflights.date,
             boarded/passengercapacity*100 as pctfull
             format=4.1 label='Percent Full',
             delay
        from sasuser.marchflights, sasuser.flightdelays
       where marchflights.flightnumber=flightdelays.flightnumber
         and marchflights.date=flightdelays.date)
```

- Assigning an Alias to an In-Line View

```
from sasuser.flightschedule as f,
    (select flightnumber, date, boarded/passengercapacity*100
            as pctfull format=4.1 label='Percent Full'
       from sasuser.marchflights) as m
where m.flightnumber=f.flightnumber
  and m.date=f.date
```

Example 13:

The table WORK.PILOTS contains the following data:

WORK.PILOTS

Id	Name	Jobcode	Salary
001	Albert	PT1	50000
002	Brenda	PT1	70000
003	Carl	PT1	60000
004	Donna	PT2	80000
005	Edward	PT2	90000
006	Flora	PT3	100000

The data set was summarized to include average salary based on jobcode:

Jobcode	Salary	Avg
PT1	50000	60000

```
  PT1     70000   60000
  PT1     60000   60000
  PT2     80000   85000
  PT2     90000   85000
  PT3    100000  100000
```

Which SQL statement could NOT generate this result?

```
A. select
   Jobcode,
   Salary,
   avg(Salary) label='Avg'
from WORK.PILOTS
group by Jobcode
order by Id
;

B. select
   Jobcode,
   Salary,
   (select avg(Salary)
   from WORK.PILOTS as P1
   where P1.Jobcode=P2.Jobcode) as Avg
from WORK.PILOTS as P2
order by Id
;
C. select
   Jobcode,
   Salary,
   (select avg(Salary)
   from WORK.PILOTS
   group by Jobcode) as Avg
from WORK.PILOTS
order by Id
;
D. select
   Jobcode,
   Salary,
   Avg
from
   WORK.PILOTS,
  (select
     Jobcode as Jc,
   avg(Salary) as Avg
   from WORK.PILOTS
   group by 1)
where Jobcode=Jc
order by Id
;
```

Example 14:

Given the SAS data set WORK.ONE:

```
 Rep       Cost
-----       ----
SMITH      200
SMITH      400
JONES      100
SMITH      600
JONES      100
```

The following SAS program is submitted;

```
proc sql;
  select
     Rep,
     avg(Cost)
  from WORK.ONE
  order by Rep
  ;
quit;
```

Which result set would be generated?

A.

```
JONES    280
JONES    280
SMITH    280
SMITH    280
SMITH    280
```

B.

```
JONES    600
SMITH    100
```

C.

```
JONES    280
SMITH    280
```

D.

```
JONES    100
JONES    100
SMITH    600
SMITH    600
SMITH    600
```

Example 15:

The table WORK.PILOTS contains the following data:

```
 Id  Name     Jobcode  Salary
---  ---------  ----------  --------
001  Albert    PT1      50000
002  Brenda   PT1      70000
003  Carl      PT1      60000
004  Donna    PT2      80000
```

```
005  Edward  PT2      90000
006  Flora     PT3     100000
```

A query was constructed to display the pilot salary means at each level of Jobcode and the difference to the overall mean salary:

```
Jobcode    Average  Difference
----------------------------------------------
PT1         60000     -15000
PT2         85000      10000
PT3        100000      25000
```

Which select statement could NOT have produced this output?

```
A. select
   Jobcode,
   avg(Salary) as Average,
   calculated Average - Overall as difference
from
   WORK.PILOTS,
   (select avg(Salary) as Overall from WORK.PILOTS)
group by jobcode
;
B. select
   Jobcode,
   avg(Salary) as Average,
   (select avg(Salary) from WORK.PILOTS) as Overall,
   calculated Average - Overall as Difference
from WORK.PILOTS
group by 1
;
C. select
   Jobcode,
   Average,
   Average-Overall as Difference
from
   (select Jobcode, avg(Salary) as Average
   from WORK.PILOTS
   group by 1),
   (select avg(Salary) as Overall
   from WORK.PILOTS)
;
D. select
   Jobcode,
   avg(Salary) as Average,
   calculated Average-(select avg(Salary) from WORK.PILOTS)
      as Difference
from WORK.PILOTS
group by 1
;
```

Managing Index in SQL

- Creating an index for a table enables PROC SQL to locate specific rows more quickly and efficiently.
- An index is an auxiliary file that stores the physical location of values for one or more specified columns (key columns) in a table.
- In an index, each unique value of the key column(s) is paired with a location identifier for the row that contains that value.

Note: You cannot create an index on a view.

- PROC SQL step uses the **CREATE INDEX** statement to create an index for a table, and uses the **DESCRIBE TABLE** statement to display information about the index, along with other information about the table, in the SAS log:

```
proc sql;
create index empid on work.payrollmaster(empid);
describe table work.payrollmaster;
```

- To display a list of columns and column attributes for one or more tables in the SAS log you can use the **DESCRIBE TABLE** statement in PROC SQL.

Example 16:
The SAS data set WORK.CHECK has a variable named Id_Code in it. Which SQL statement would create an index on this variable?

A. create index Id_Code on WORK.CHECK;

B. create index(Id_Code) on WORK.CHECK;

C. make index=Id_Code from WORK.CHECK;

D. define index(Id_Code) in WORK.CHECK;

Example 17:
The following SAS program is submitted:

```
 proc contents data=TESTDATA.ONE;
 run;
```

Which SQL procedure step produces similar information about the column attributes of TESTDATA.ONE?

A. proc sql;

```
   contents from TESTDATA.ONE;
quit;
B. proc sql;
   describe from TESTDATA.ONE;
quit;
C. proc sql;
   contents table TESTDATA.ONE;
quit;

D. proc sql;
   describe table TESTDATA.ONE;
quit;
```

Example 18:

DESCRIBE TABLE Statement

Given the following partial SAS log:

```
  NOTE: SQL table SASHELP.CLASS was created like:
  create table SASHELP.CLASS( bufsize=4096 )
    (
     Name char(8),
     Sex char(1),
     Age num,
     Height num,
     Weight num
    );
```

Which SQL procedure statement generated this output?

A. CONTENTS FROM SASHELP.CLASS;
B. CREATE FROM SASHELP.CLASS INTO LOG;
C. DESCRIBE TABLE SASHELP.CLASS;
D. VALIDATE SELECT * FROM SASHELP.CLASS;