临床研究SAS高级编程
—— Data Process

Contents

Methods of Combining SAS Data Sets

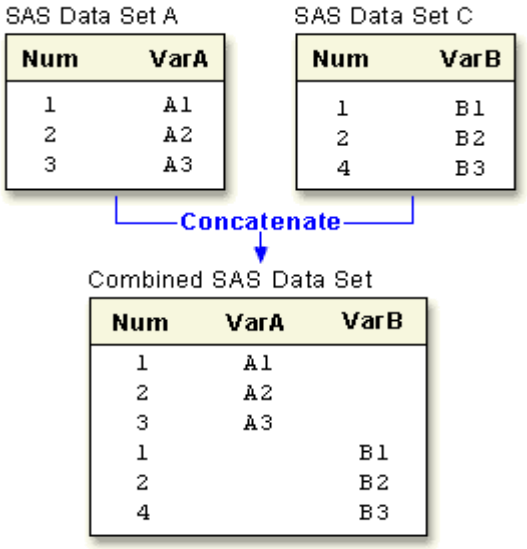
Contents

- Concatenating
- Interleaving
- Merging
- Updating
- Modifying

Concatenating (1)

Concatenating

- Append the observations from one dataset to another
- You can concatenate SAS data sets by using
  - the **SET** statement in a DATA step
  - the **APPEND** procedure



Concatenating (2)

- Concatenating data sets with the **SET** statement
  - General form, basic DATA step for concatenating:

```
DATA output-SAS-data-set;
   SET SAS-data-set-1 SAS-data-set-2;
RUN;
```

    where

    output-SAS-data-set names the data set to be created

    SAS-data-set-1 and **SAS-data-set-2** specify the data sets to be read.
  - You can specify any number of data sets in the SET statement.
  - The number of observations and variables in the new data set is the sum of the number of observations and variables in the original data sets.

Concatenating (3)

Example

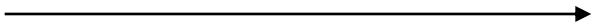
A

Num	Var
1	a1
2	a2

B

Num	Var
1	b1
2	b2

```
Data c;
   Set a b;
Run;
```



C

Num	Var
1	a1
2	a2
1	b1
2	b2

Concatenating (4)

- Using the SET statement
  - When original datasets contain different variables, new dataset have missing values for non-common variables for source observations.

A

Num	VarA
1	a1
2	a2

B

Num	VarB
1	b1
2	b2

```
Data c;
  Set a b;
Run;
```

C

Num	VarA	VarB
1	a1	
2	a2	
1		b1
2		b2

Concatenating (5)

- Original datasets contain same variables with different attributes (type, length, informat, format, label):
  - When variables have different types, SAS issues an error message and does not concatenate the datasets.

    In the previous example, if you define the variable **num** as character in dataset **A** while as numeric in dataset **B**, resubmit the program and SAS writes the following error message to the log.

    ERROR: Variable Num has been defined as both character and numeric.

    One way to correct the error is to change the type of the variable **num**.

  - When variables have different lengths, SAS takes the length from **the first data set** that contains the variable.

  - When variables have different Formats, Informats or labels, SAS takes the attribute from **the first data set** that contains the variable with that attribute.

Concatenating (6)

Concatenating data sets using the APPEND procedure

- The **APPEND** procedure has the following form:

```
PROC APPEND  BASE=base-SAS-data-set
          <DATA=SAS-data-set-to-append>
          <FORCE>;
run;
```

- ***base-SAS-data-set*** names the SAS data set to which you want to append the observations.
  If this data set does not exist, then SAS creates it. At the completion of PROC APPEND, the value of base-SAS-data-set becomes the current (most recently created) SAS data set.
- **SAS-data-set-to-append** names the SAS data set that contains the observations to add to the end of the base data set. If you omit this option, then PROC APPEND adds the observations in the current SAS data set to the end of the base data set.
- ***FORCE***
  forces PROC APPEND to concatenate the files in some situations in which the procedure would normally fail.

Concatenating (7)

- Example

A

Num	Var
1	a1
2	a2

B

Num	Var
1	b1
2	b2

```
Proc append base=a data=b;
Run;
```

C

Num	Var
1	a1
2	a2
1	b1
2	b2

Concatenating (8)

- Using the APPEND procedure
  - If the BASE= data set contains a variable that is not in the DATA= data set, then PROC APPEND concatenates the data sets and assigns a missing value to that variable in the observations that are taken from the DATA= data set.
  - If the DATA= data set contains a variable that is not in the BASE= data set, then the FORCE option in PROC APPEND forces the procedure to concatenate the two data sets. But because that variable is not in the descriptor portion of the BASE= data set, the procedure cannot include it in the concatenated data set.
  - Each data set contains a variable that is not in the other. It is only the case of a variable in the DATA= data set that is not in the BASE= data set that requires the use of the FORCE option. However, both cases display a warning in the log.

Concatenating (9)

- Original datasets contain same variables with different attributes (type, length, informat, format label):
- If a variable has different attributes in the BASE= data set than it does in the DATA= data set, then the attributes in the BASE= data set prevail. In the cases of differing formats, informats and labels, concatenation succeeds.
- If the length of a variable is longer in the BASE= data set than in the DATA= data set, then the concatenation succeeds.
- If the length of a variable is longer in the DATA= data set than in the BASE= data set, or if the same variable is a character variable in one data set and a numeric variable in the other, then PROC APPEND fails to concatenate the files unless you specify the FORCE option.

Concatenating (10)

- Using the **FORCE** option has these consequences:
  - The length that is specified in the BASE= data set prevails. Therefore, SAS truncates values from the DATA= data set to fit them into the length that is specified in the BASE= data set.
  - The type that is specified in the BASE= data set prevails. The procedure replaces values of the wrong type (all values for the variable in the DATA= data set) with missing values.

Concatenating (11)

- Differences between the **SET** Statement and the **APPEND** Procedure

Criterion	SET Statement	APPEND Procedure
Number of datasets that you can concatenate	Uses any number of data sets	Uses two data sets
Handling of data sets that contain different variables	Uses all variables and assigns missing values where appropriate	Uses all variables in the BASE= data set and assigns missing values to observations from the DATA= data set where appropriate. Requires the FORCE option to concatenate data sets if the DATA= data set contains variables that are not in the BASE= data set. Cannot include variables found only in the DATA= data set when concatenating the data sets.

Concatenating (12)

- Differences between the SET Statement and the APPEND Procedure

Criterion	**SET** Statement	**APPEND** Procedure
Handling of different formats, informats, or labels	Uses explicitly defined formats, Informats, and labels rather than defaults. If two or more datasets explicitly define the format, informat, or label, then SAS uses the definition from the dataset you name first in the SET statement.	Requires the FORCE option If the length of a variable is longer in the DATA= dataset. Truncates the values of the variable to match the length in the BASE= dataset.
Handling of different Variable types	Does not concatenate the data sets.	Requires the FORCE option to concatenate datasets. Uses the type attribute from the BASE= dataset and assigns missing values to the variable in observations from the DATA= dataset.

Interleaving (1)

Interleaving

- If you use a BY statement when you concatenate data sets, the result is **interleaving. Interleaving** intersperses observations from two or more data sets, based on one or more common variables.

- General form, basic **DATA** step for interleaving:

```
DATA output-SAS-data-set;
   SET SAS-data-set-1 SAS-data-set-2;
   BY variable(s);
RUN;
```

where

output-SAS-data-set names the data set to be created

SAS-data-set-1 and **SAS-data-set-2** specify the data sets to be read

variable(s) specifies one or more variables that are used to interleave observations.

- You can specify any number of data sets in the **SET** statement. Each input data set must be sorted or indexed in ascending order based on the **BY** variable(s).

Interleaving (2)

- Example

A

Num	VarA
1	a1
3	a2

B

Num	VarB
2	b1
4	b2

```
Data c;
    Set a b;
    By num;
Run;
```

C

Num	VarA	VarB
1	a1	
2		b1
3	a2	
4		b2

Interleaving (3)

- Understanding the **interleaving** process

When interleaving, SAS creates a new data set as follows:

- Before executing the SET statement, SAS reads the descriptor portion of each dataset that you name in the SET statement. Then SAS creates a program data vector that, by default, contains all the variables from all data sets as well as any variables created by the DATA step. SAS sets the value of each variable to missing.
- SAS looks at the first BY group in each data set in the SET statement in order to determine which BY group should appear first in the new data set.
- SAS copies to the new data set all observations in that BY group from each data set that contains observations in the BY group. SAS copies from the data sets in the same order as they appear in the SET statement.
- SAS looks at the next BY group in each data set to determine which BY group should appear next in the new data set.
- SAS sets the value of each variable in the program data vector to missing.
- SAS repeats steps 3 through 5 until it has copied all observations to the new data set.

Merging (1)

Merging

- **Merging** combines observations from two or more SAS data sets into a single observation in a new SAS data set. The new data set contains all variables from all the original data sets unless you specify otherwise.
- You can merge SAS data sets by using
  - **one-to-one *merging***
  - **match merging**

Merging (2)

- One-to-one merging
  - you do not use a **BY** statement. Observations are combined based on their positions in the input data sets.

A

Num	VarA
1	a1
2	a2

B

Num	VarB
3	b1
4	b2
5	b3

```
Data c;
   merge a b;
Run;
```

C

Num	VarA	VarB
3	a1	b1
4	a2	b2
5		b3

Merging (3)

- Match-merging
  - Merging with a BY statement enables you to match observations according to the values of the BY variables that you specify. Before you can perform a match-merge, all data sets must be sorted by the variables that you want to use for the merge.

A

Num	VarA
1	a1
2	a2

B

Num	VarB
1	b1
2	b2
3	b3

```
Data c;
    merge a b;
    by num;
Run;
```

C

Num	VarA	VarB
1	a1	b1
2	a2	b2
3		b3

Merging (4)

- In order to understand match-merging, you must understand three key concepts:

  BY variable: is a variable named in a BY statement.

  BY value : is the value of a BY variable.

  BY group : is the set of all observations with the same value for the BY variable (if there is only one BY variable). If you use more than one variable in a BY statement, then a BY group is the set of observations with a unique combination of values for those variables. In discussions of match-merging, BY groups commonly span more than one data set.

Merging (5)

- Match-merge processing
  - match-merging can be complex, depending on your data and on the output data set that you want to create.
  - To predict the results of match-merges correctly, you need to understand how the DATA step performs match-merges :

    When you submit a DATA step, it is processed in two phases:
    - **the compilation phase**
    - **the execution phase**

Merging (6)

Compilation phase

- To prepare to merge data sets, SAS
  - reads the **descriptor portions** of the data sets that are listed in the MERGE statement
  - reads the rest of the DATA step program
  - creates the **program data vector (PDV)** for the merged data set
  - assigns a tracking pointer to each data set that is listed in the MERGE statement.
- If variables that have the same name appear in more than one data set, the variable from the first data set that contains the variable (in the order listed in the MERGE statement) determines the length of the variable.

Merging (7)

Execution phase

- SAS sequentially match-merges observations by moving the pointers down each observation of each data set and checking to see **whether the BY values match**:
  - If **Yes**, the observations are written to the PDV in the order in which the data sets appear in MERGE statement. Values of any same-named variable are overwritten by values of the same-named variable in subsequent data sets. SAS writes the combined observation to the new dataset and retains the values in the PDV until the BY value changes in all the data sets.
  - If **No**, SAS determines which of the values comes first and writes the observation that contains this value to the PDV. Then the contents of the PDV are written to the new data set.

    When the BY value changes in all the input data sets, the PDV is initialized to missing.

Merging (8)

- Handling unmatched observations and missing values
  - By default, all observations that are written to the PDV, including observations that have missing data and no matching BY values, are written to the output data set. (If you specify a subsetting IF statement to select observations, then only those that meet the IF condition are written.)
    - If an observation contains **missing values for a variable**, then the observation in the output data set contains the missing values as well. Observations that have missing values for the BY variable appear at the top of the output data set.
    - If an input data set **doesn't have a matching BY value**, then the observation in the output data set contains missing values for the variables that are unique to that input data set.

Merging (9)

- Examples
  - One to one

A

Num	VarA
1	a1
2	a2

B

Num	VarB
1	b1
2	b2

```
Data c;
   Merge a b;
   By num;
Run;
```

C

Num	VarA	VarB
1	a1	b1
2	a2	b2

Merging (10)

- Examples
  - One to multiple

A

Num	VarA
1	a1
2	a2

B

Num	VarB
1	b1
1	b2

```
Data c;
    Merge a b;
    By num;
Run;
```

C

Num	VarA	VarB
1	a1	b1
1	a2	b2
2	a2	

Merging (11)

- Examples
  - multiple to multiple

A

Num	VarA
1	a1
1	a2
1	a3

B

Num	VarB
1	b1
1	b2
2	b3

```
Data c;
    Merge a b;
    By num;
Run;
```

C

Num	VarA	VarB
1	a1	b1
1	a2	b2
1	a3	b2
2		b3

Updating (1)

Updating

- **Updating** a SAS data set replaces the values of variables in one data set (the master data set) with values from another data set (the transaction data set).
- The general form of the UPDATE statement is

```
UPDATE master-SAS-data-set transaction-SAS-data-set;
       BY identifier-list;
```

where

- ***master-SAS-data-set*** is the SAS data set containing information you want to update.
- ***transaction-SAS-data-set*** is the SAS data set containing information with which you want to update the master data set.
- **identifier-list** is the list of BY variables by which you identify corresponding observations.

Updating (2)

- Example

A (Master)

Num	VarA
1	a1
2	a2
3	a3

B (Transaction)

Num	VarA
1	b1
2	b2

```
Data c;
   Update a b;
   by num;
Run;
```

C

Num	VarA
1	b1
2	b2
3	a3

Updating (3)

- Understanding how to select **BY** variables
  - The **master data set** and the **transaction data set** must be sorted by the same variable or variables that you specify in the BY statement.
  - The values of by variables should be unique for each observation in the master data set.
  - SAS warns you if it finds duplicates but proceeds with the update. It applies all transactions only to the first observation in the BY group in the master data set.

Updating (4)

- Understanding the differences between **updating** and **merging**
  - The Update statement uses only two dataset. The number of data sets that the Merge Statement can use is limited only by machine-dependent factors such as memory and disk space.
  - A BY statement must accompany an UPDATE statement. The Merge statement performs a one-to-one merge if no BY statement follows it.

Updating (5)

- The two statements also process observations differently when a dataset contains missing values or multiple observations in a By group.
  - During an update, if a value for a variable is missing in the transaction data set, SAS uses the value from the master data set when it writes the observation to the new data set. When merging the same observations, SAS overwrites the value in the program data vector with the missing value.
  - SAS does not write an updated observation to the new data set until it has applied all the transactions in a BY group. When merging datasets, SAS writes one new observation for each observation in the dataset with the largest number of observations in the BY group.

Updating (6)

Handling missing values

- Use the UPDATEMODE option on the UPDATE statement

  The Syntax is as follows:

```
UPDATE master-SAS-data-set transaction-SAS-data
-set <UPDATEMODE=MISSINGCHECK | NONMISSINGCHECK>;
BY by-variable;
```

  - The MISSINGCHECK value in the updatemode option prevents missing value in the transaction dataset from replacing values in a master data set.
  - The NONMISSINGCHECK value in the UPDATEMODE option enable missing value in the transaction dataset to replace values in a master dataset by preventing the check for missing data from being performed.

Modifying (1)

Modifying

- Modifying a SAS data set replaces, deletes,or appends observations in an existing dataset.
- Modifying a SAS data set is similar to updating a SAS data set, but the following differences exists:
  - Modifying cannot create a new data set, while updating can.
  - Unlike updating, modifying does not require that the master data set or the transaction data set be sorted.

Modifying (2)

- You can use modify statement in data step to do the following:
  - Replace values in a data set.
  - Replace values in a master data set with values from a transaction data set.
  - Append observations to an existing SAS data set.
  - Delete observations from an existing SAS data set.

Modifying (3)

- Replace values in a data set
  - The syntax for using the Modify statement and the By statement is:

```
MODIFY SAS-data-set;
```

a

Num	VarA
1	2.5
2	3.5

```
Data a;
  Modify a;
  VarA=VarA*2;
Run;
```

b

Num	VarA
1	5
2	7

Modifying (4)

- Replace values in a master data set with values from a transaction data set
  - The syntax for using the MODIFY statement and the By statement is:

```
Modify master-SAS-data-set transaction-data set;
BY by-variable;
```

The master-SAS-data-set specifies the SAS data set that you want to modify. The transaction-SAS-data-set specifies the SAS data set that provides the values for updating the master data set. The by variable specifies one or more variables by which you identify corresponding observations.

Modifying (5)

Example

Master

Num	VarA
1	a1
2	a2
3	a3

Transaction

Num	VarA
2	b1
1	b2

```
Data master;
   Modify master
        transaction;
   By num;
Run;
```

Master

Num	VarA
1	b2
2	b1
3	a3

Modifying (6)

- Adding new observations to the master data set
  - If the transaction data set contains an observation that does
  - Not match an observation in the master data set, then SAS writes a new observation to the master data set if you use an explicit output statement in your program.

master

Num	VarA
1	a1
2	a2

transaction

Num	VarA
2	b1

```
Data master;
   Modify master
          transaction;
   By num;
   Output;
Run;
```

master

Num	VarA
1	a1
2	a2
3	a3

Modifying (7)

- How the data step processes duplicate by variables
  - The data step process duplicate observations in the following way:
  - If duplicate by values exists in the master data set, then Modify applies the current transaction to the first occurrence in the master data set.
  - If duplicate by values exists in the transaction data set, then the observation are applied one on top of another so that the values overwrites each other. The value in the last transaction is the final value in the master data set.
  - If both the master and the transaction data sets contain duplicate BY values, then Modify applies each transaction to the first occurrence in the group in the master data set.

Modifying (8)

- Handling missing values
  - By default, if the transaction data set contains missing values for a variable that is common to both the master and the transaction data sets, then the MODIFY statement does not replace values in the master data set with missing values.
  - If you want to replace values in the master data set with missing values, then you use the UPDATEMODE= option on the MODIFY statement.
    - The syntax is :

```
MODIFY master-SAS-data-set transaction-SAS-data-set
    <UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;
 BY by-variable;
```

    - **MISSINGCHECK** prevents missing values in the transaction data set from replacing values in the master data set. This is the default.
    - **NOMISSINGCHECK** enables missing values in a transaction data set to replace values in the master data set by preventing the check for missing data from being performed.

Quiz

If you concatenate the data sets below in the order shown, what is the value of Sale in observation 2 of the new data set?

Sales.Reps

ID	Name
1	Nay Rong
2	Kelly Windsor
3	Julio Meraz
4	Richard Krabill

Sales.Close

ID	Sale
1	$28,000
2	$30,000
2	$40,000
3	$15,000
3	$20,000
3	$25,000
4	$35,000

Sales.Bonus

ID	Bonus
1	$2,000
2	$4,000
3	$3,000
4	$2,500

a. Missing *b.* *$30,000*

c. *$40,000* *d.* you cannot concatenate these data sets

► Correct answer: a

- The concatenated data sets are read sequentially, in the order in which they are listed in the SET statement. The second observation in **Sales.Reps** does not contain a value for Sale, so a missing value appears for this variable. (Note that if you merge the data sets, the value of Sale for the second observation is *$30,000*.)

Quiz

- What happens if you merge the following data sets by the variable SSN?

1st

SSN	Age
029-46-9261	39
074-53-9892	34
228-88-9649	32
442-21-8075	12
446-93-2122	36
776-84-5391	28
929-75-0218	27

2nd

SSN	Age	Date
029-46-9261	37	02/15/95
074-53-9892	32	05/22/97
228-88-9649	30	03/04/96
442-21-8075	10	11/22/95
446-93-2122	34	07/08/96
776-84-5391	26	12/15/96
929-75-0218	25	04/30/97

a. The values of Age in the 1st data set overwrite the values of Age in the 2nd data set.
b. The values of Age in the 2nd data set overwrite the values of Age in the 1st data set.
c. The DATA step fails because the two data sets contain same-named variables that have different values.
d. The values of Age in the 2nd data set are set to missing.

- Correct answer: b
  - If you have variables with the same name in more than one input data set, values of the same-named variable in the first data set in which it appears are overwritten by values of the same-named variable in subsequent data sets.

Quiz

If you merge Company.Staff1 and Company.Staff2 below by ID, how many observations does the new data set contain?

Company.Staff1

ID	Name	Dept	Project
000	Miguel	A12	Document
111	Fred	B45	Survey
222	Diana	B45	Document
888	Monique	A12	Document
999	Vien	D03	Survey

Company.Staff2

ID	Name	Hours
111	Fred	35
222	Diana	40
777	Steve	0
888	Monique	37

a. 4
b. 5
c. 6
d. 9

- Correct answer: c
  - In this example, the new data set contains one observation for each unique value of ID. The merged data set is shown below.

ID	Name	Dept	Project	Hours
000	Miguel	A12	Document	.
111	Fred	B45	Survey	35
222	Diana	B45	Document	40
777	Steve			0
888	Monique	A12	Document	37
999	Vien	D03	Survey	.

Transpose procedure (1)

Objective

- The transpose procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations.
- The transpose procedure can often eliminate the need to write a lengthy DATA step to achieve the same result.
- Further, the output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.
- A transposed variable is a variable that the procedure creates by transposing the values of an observation in the input data set into values of a variable in the output data set.

Refer to example 1

Transpose procedure (2)

- Syntax

```
PROC TRANSPOSE  <DATA=input-data-set><LABEL=label><LET>
      <NAME=name><OUT=output-data-set><PREFIX=prefix>;
   BY <DESCENDING> variable-1
         <…<DESCENDING> variable-n>
         <NOTSORTED>;
   COPY variable(s);
   ID variable;
      IDLABEL variable;
   VAR variable(s);
```

BY : Transpose each BY group
COPY : COPY variables directly without transposing them
ID : Specify a variable whose values name the transposed variables
IDLABEL: Create labels for the transposed variables
VAR: List the variables to transpose

Transpose procedure (3)

- Proc transpose statement
  - You can use data set options with the data= and OUT= options.

```
PROC TRANSPOSE  <DATA=input-data set> <LABEL=label>
                <LET> <NAME=name> <OUT=output-data-set>
                <PREFIX=prefix>;
```

  - **DATA**=input-data-set names the SAS data set to transpose
    Default: most recently created SAS data set
  - **LABEL**=label
    specifies a name for the variable in the output data set that contains the label of the variable that is being transposed to create the current observation.
    Default: _LABEL_
  - **LET**
    Allows duplicate values of an ID variable. PROC TRANSPOSE transposes the observation that contains the last occurrence of a particular ID value within the data set or BY group.

Transpose procedure (4)

- **NAME**=name

  Specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation.

- **Out**=output-data-set

  Names the output data set. If output-data-set does not exists, then PROC Transpose creates it by using the DATAn naming convention.

  Default: DATAn

- **Prefix**=prefix

  Specifies a prefix to use in constructing names for transposed variables in the output data set.

  For example, if PREFIX=VAR, then the names of the variables are VAR1, VAR2, …, VARn.

  Interaction: When you use PREFIX= with an ID statement, the value prefixes to the ID value.

Transpose procedure (5)

- By statement
  - Defines BY groups.
  - The syntax:

```
BY <DESCENDING> variable-1
                <…<DESCENDING> variable-n>
                <NOTSORTED>;
```

  - Required Auguments:

Variable: specifies the variable that PROC TRANSPOSE uses to form By groups.you can specify more variables.If you do not use the notsorted option in the BY statement, then either the observations must be sorted by all the variables that youspecify, or they must be indexed appropriately.Variables in a BY statement are called BY variables.

Transpose procedure (6)

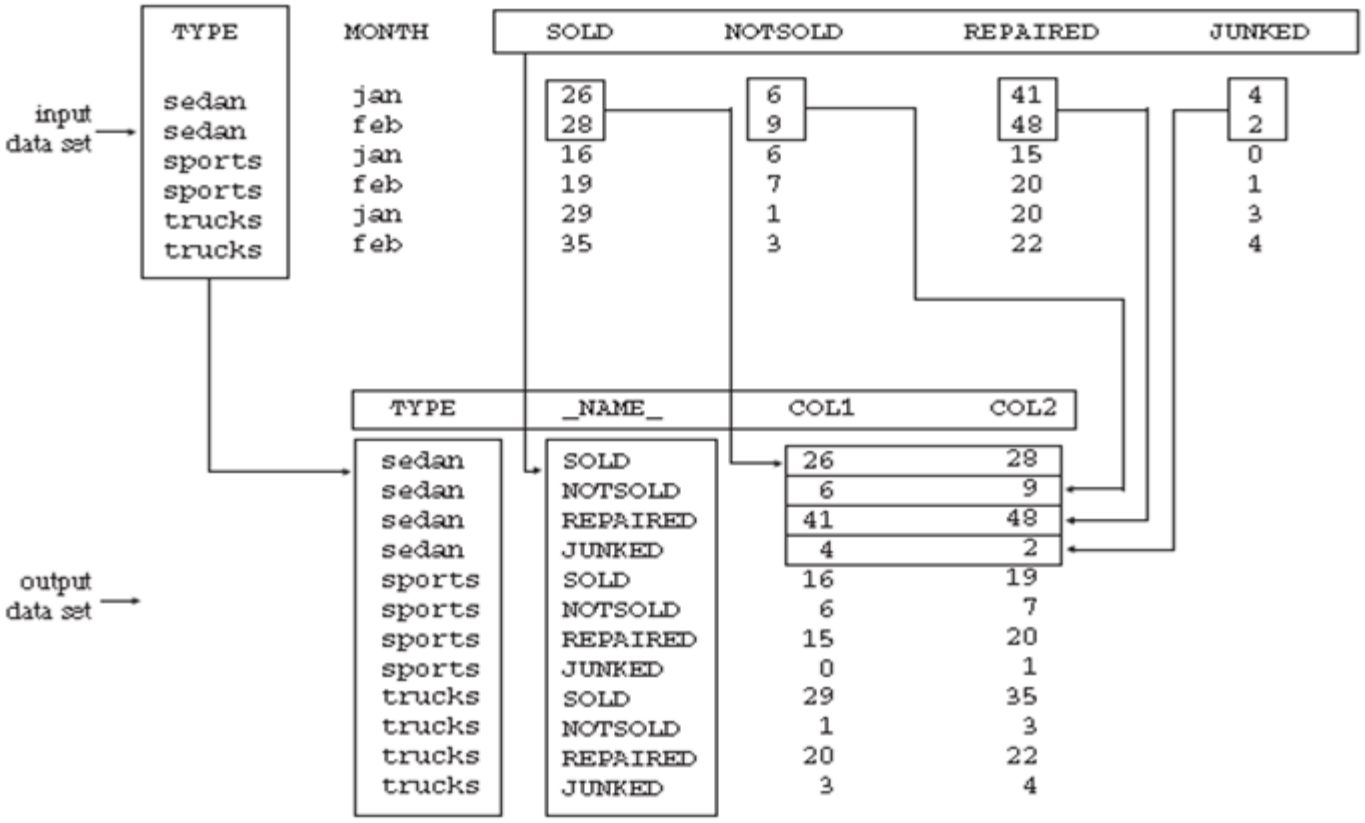
- Options
  - **DESCENDING**

    specifies that the data set is sorted in descending order by the variable that immediately follows the word DESENDING in the BY statement.
  - **NOTSORTED**

    specifies that observations are not necessarily sorted in alphabetic or numeric order.
- Transposition with BY groups
  - PROC TRANSPOSE does not transpose BY groups.
  - For each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes.
  - If a BY group in the input data set has more observations than other BY groups, then PROC TRANSPOSE assigns missing values in the output data set to the variables that have no corresponding input observations

The following figure show what happens when you transpose a dataset with BY groups.

Transpose procedure (7)



Transpose procedure (8)

Copy statement

- Copies variables directly from the input data set to the output data set without transposing them.
- The syntax:

```
COPY variable(s);
```

- Required Argument

  Variable(s)

  Names one or more variables that the COPY statement copies directly from the input data set to the output data set without transposing them.

Transpose procedure (9)

- ID statement
  - Specifies a variable in the input data set whose formatted values name the transposed variables in the output data set. (Example 2)
  - The syntax:

    ```
    ID variable;
    ```

  - Required argument

    Variable : Names the variable whose formatted values names the transposed variables.
  - Duplicate ID values
    - Typically ,each formatted ID values occurs only once in the input data set or within a BY group if using BY statement.
    - Duplicate values cause PROC TRANSPOSE to issue a warning message and stop.
    - If using LET option, the procedure will issue a warning message and transpose the observation that contains the last occurrence of the duplicate ID values.

      Refer to example 5.

Transpose procedure (10)

- Making variable names out of numeric values
  - PROC TRANSPOSE changes the formatted ID value into a valid SAS name when you use a numeric variable as an ID variable.
  - When the first character of the formatted value is numeric,the procdure
    - prefixes an underscore to the value
    - Truncates the last character of a 32-character value
  - Any remaining invalid character are replaced by underscores.
  - If the formatted value looks like a numeric constant, then PROC TRANSPOSE changes the characters ‘+’,’-’, and ‘.’ to ‘P’,’N’, and ‘D’, respectively. If the formatted value has characters that are not numerics,then PROC TRANSPOSE changes the characters ‘+’,’-’,and ‘.’ to underscore.

Transpose procedure (11)

- Missing values
  - If you use an ID variable that contains a missing value, then PROC TRANSPOSE writes an error message to the log.
  - The procedure does not transpose observations that have a missing value for the ID variable.

Transpose procedure (12)

IDLABEL statement

- Create labels for the transposed variables
- Must appear after an ID statement
- The syntax:

```
IDLABEL variable;
```

- Required argument

  Variable:

  names the variable whose values the procedure uses to label the variables that ID statement names. Variables can be character or numeric.

  Refer to example 3.

Transpose procedure (13)

- Var statement
  - Lists the variables to transpose
  - The syntax:

    ```
    Var variable(s);
    ```

  - Required argument

    Variable(s)

    Names one or more variables to transpose
  - If you omit the VAR statement, then the tranpose procedure transposes all numeric variables in the input data set that are not listed in another statement.
  - You must list character variables in a VAR statement if you want to transpose them.

Transpose procedure (14)

Results

- The transpose procedure always produce an output data set, but does not print the output data set.
- The output data set contains the following variables:
  - Variables that result from transposing the values of each variable into and observation.
  - A variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. The default variable name is _NAME_.
  - Variables that PROC TRANSPOSE copies from the input data set when you use either the BY or COPY statement.
  - A character variable whose values are the variable labels of the variables that are being transposed. The default variable name is _LABEL_.
- Attributes of transposed variables:
  - All transposed variables are the same type and length.
  - If all variables that the procedure is transposing are numeric, then the transposed variables are numeric .
  - If any variable that the procedure is transposing is character, then all transposed variables are character.
  - The length of the transposed variable is equal to the length of the longest variable that is being transposed.

Transpose procedure (15)

Names of transposed variables

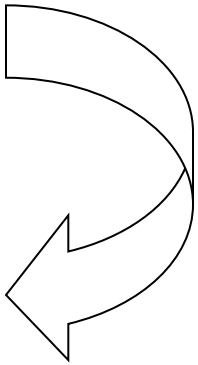
- An ID statement specifies a variable in the input data set whose formatted values become names for the transposed variables.
- The PREFIX= option specifies a prefix to use in constructing the names of transposed variables.
- If you do not use ID statement or the PREFIX=option, then PROC TRANSPOSE looks for an input variable called _NAME_ from which to get the names of the transposed variables.
- If you do not use ID statement or the PREFIX=option, and if the input data set does not contain a variable named _NAME_, then PROC TRANSPOSE assigns the names COL1,COL2,…,COLn to the transposed variable.

Transpose procedure (16)

Example 1-Performing a simple transposition

A

student	stuid	tester1	tester2
Jim	01	22	25
John	02	15	19



```
Proc transpose data=a Out=b;
Run;
```

B

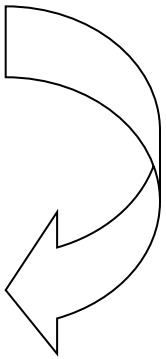
NAME OF FORMER VARIABLE	COL1	COL2
tester1	22	15
tester2	25	19

Transpose procedure (17)

Example 2-Naming transposed variables

A

student	stuid	tester1	tester2
Jim	01	22	25
John	02	15	19



```
Proc transpose data=a
     out=c
     name=Test
     prefix=sn;
  ID StuID;
Run;
```

C

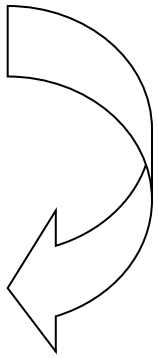
NAME OF FORMER VARIABLE	sn01	sn02
tester1	22	15
tester2	25	19

Transpose procedure (18)

Example 3-Labeling transposed variables

A

student	stuid	tester1	tester2
Jim	01	22	25
John	02	15	19



```
Proc transpose data=a
     Out=d
     name=Test
     prefix=sn;
     ID StuID;
     IDLABEL student;
Run;
```

D

NAME OF FORMER VARIABLE	Jim	John
tester1	22	15
tester2	25	19

Transpose procedure (19)

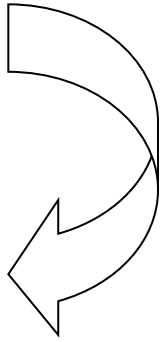
Example 4-Transposing by groups

E

Location	Date	Length1	Length2	Length3	Length4
Cole Pond	02JUN95	31	32	32	33
Cole Pond	03JUL95	33	34	37	32
Eagle Lake	02JUN95	32	32	33	.

F

Location	Date	NAME OF FORMER VARIABLE	COL1
Cole Pond	02JUN95	Length1	31
Cole Pond	02JUN95	Length2	32
Cole Pond	02JUN95	Length3	32
Cole Pond	02JUN95	Length4	33
Cole Pond	03JUL95	Length1	33
Cole Pond	03JUL95	Length2	34
Cole Pond	03JUL95	Length3	37
Cole Pond	03JUL95	Length4	32
Eagle Lake	02JUN95	Length1	32
Eagle Lake	02JUN95	Length2	32
Eagle Lake	02JUN95	Length3	33
Eagle Lake	02JUN95	Length4	.



```
Proc transpose
     data=e
     Out=f;
     Var length1-length4;
     By location date;
Run;
```

Transpose procedure (20)

- Example 5-Naming transposed variables when the ID variable has duplicate values

G

Company	Date	Time	Price
Horizon Kites	jun11	opening	29
Horizon Kites	jun11	noon	27
Horizon Kites	jun11	closing	27
Horizon Kites	jun12	opening	27
Horizon Kites	jun12	noon	28
Horizon Kites	jun12	closing	30
SkyHi Kites ju	n11	opening	43
SkyHi Kites ju	n11	noon	43
SkyHi Kites ju	n11	closing	44
SkyHi Kites ju	n12	opening	44
SkyHi Kites ju	n12	noon	45
SkyHi Kites ju	n12	closing	45

```
Proc transpose
     data=g
     Out=h let;
     By company;
    Id date;
Run;
```

H

Company	NAME OF FORMER VARIABLE	jun11	jun12	n11	n12
Horizon Kites	Price	27	30	.	.
SkyHi Kites ju	Price	.	.	44	45

Subset

Contents

- Acting on selected observations
- Creating subset of observations

Acting on selected observations (1)

Objective

- Learn how the selection process works
- Learn how to write statements that select observations based on a condition
- Learn some special points about selecting numeric and character variables

Acting on selected observations (2)

Selecting observations

- Understanding the selection process

  The most common way that SAS selects observations for action in a DATA step is through the IF-THEN statement:

```
IF condition  THEN action;
```

  For example,

```
if City='Rome' then Landcost=landcost+30;
```

  - When the condition is true, SAS takes the action in the THEN clause
  - When the condition is false, SAS ignores the THEN clause and proceeds to the next statement in the DATA step.

Acting on selected observations (3)

- Providing an alternative action
  - A second IF-THEN statement

    For example:

```
if NumberOfEvents > Nights then Calendar='Check chedule';
if NumberOfEvents <= Nights then Calendar = 'No problems';
```

  - An ELSE statement

```
ELSE action;
```

    For example;

```
if NumberOfEvents > Nights then Calendar='Check schedule';
else Calendar = 'No problems';
```

Acting on selected observations (4)

- Creating a series of mutually exclusive conditions
  - A series of IF-THEN and ELSE statements

    For example:

```
if LandCost >= 1500 then Price = 'High ';
else if LandCost >= 700 then Price = 'Medium';
else Price = 'Low';
```

  - When an observation satisfies one condition in a series of mutually exclusive IF-THEN/ELSE statements, SAS processes that THEN action and skips the rest of the statements.
  - You can increase the efficiency of a program by ordering the IF-THEN/ELSE statements so that the most common conditions appear first.

Acting on selected observations (5)

- Constructing conditions
  - Understanding construct conditions

Comparison operators

Symbol	Mnemonic Operator	Meaning
=	EQ	equal to
¬=, ^= , ~=	NE	not equal to (the ¬, ^, or ~ symbol, depending on your keyboard)
>	GT	greater than
<	LT	less than
>=	GE	greater than or equal to
<=	LE	less than or equal to

You can use either symbol or **mnemonic operators** in comparisons. But remember that Mnemonic operators are used only in comparisons
For example, the equal sign in an assignment statement must be represented by the symbol =, not the mnemonic operator.

Acting on selected observations (6)

- Using more than one comparison in a condition
  - Specifying multiple comparisons

    & or **AND**

    | or **OR**

    A condition can contain any number of ANDs, ORs, or both.
  - When comparisons are connected by AND, all of the comparisons must be true for the condition to be true.

    For example,

```
if City = 'Paris' and TourGuide = 'Lucas' then
    Remarks= 'Bilingual';
```

    The comparison is true for observations in which the value of City is Paris and the value of TourGuide is Lucas.
  - When comparisons are connected by OR, only one of the comparisons need to be true for the condition to be true.

    For example,

```
if LandCost gt 1500 or LandCost / Nights gt 200 then
    Level = 'Deluxe';
```

    Any observation in which the land cost is over $1500, the cost per night is over $200,or both, satisfies the condition. The following DATA step shows this condition.

Acting on selected observations (7)

- Using complex comparisons that requires AND and OR

  When a condition contains both ANDs and ORs, SAS evaluates the ANDs before the ORs. For example,

```
if City = 'Paris' or City = 'Rome' and TourGuide = 'Lucas' or
TourGuide = "D'Amico" then Topic = 'Art history';
```

  SAS first joins the items that are connected by AND:

```
City = 'Rome' and TourGuide = 'Lucas'
```

  Then SAS makes the following OR comparisons:

```
City = 'Paris'
    or
City = 'Rome' and TourGuide = 'Lucas'
   Or
TourGuide = "D'Amico"
```

  To group the city comparisons and the Tourguide comparisons, use parentheses:

```
if (City = 'Paris' or City = 'Rome') and
(TourGuide = 'Lucas' or TourGuide = "D'Amico") then
Topic = 'Art history';
```

  SAS evaluates the comparsions within parentheses first and uses the results as the terms of the larger comparsion.

Acting on selected observations (8)

- Abbreviating numeric comparisions

  In SAS, any numeric value other than 0 or missing is true, and a value of 0 or missing is false.

  Therefore, a numeric variable or expression can stand alone in a condition.

  For example,

```
if LandCost then Remarks = 'Ready to budget';
```

  The statement is equivalent to

```
if LandCost ne . and LandCost ne 0 then
    Remarks = 'Ready to budget';
```

  Be careful when you abbreviate comparisons with OR, it is easy to produce unexpected results

  For example, this IF-THEN statement selects tours that last six or eight nights:

```
if Nights = 6 or 8 then Stay = 'Medium';
```

  SAS treats the condition as the following comparisons:

```
Nights=6 or 8
```

  The correct way should be:

```
if Nights = 6 or Nights = 8 then Stay = 'Medium';
```

Acting on selected observations (9)

- Comparing characters
  - Use the **UPCASE** function to produce an uppercase value to compare values that may occur in different cases

    For example,

    ```
    if upcase(City) = 'MADRID' then TourGuide = 'Balarezo';
    ```
  - To compare a long value to a shorter standard, put a colon (:) after the operator, as in the example:

    ```
    if TourGuide =: 'D' then Chosen = 'Yes';
    else Chosen = 'No';
    ```

    The colon causes the SAS to compare the same number of characters in the shorter value and the longer value. Thus, in this example, all names beginning with D will make the comparison true.

Acting on selected observations (10)

- Points to understand to select a range of character values

  In computer processing, letters have magnitude. A is the smallest letter in the alphabet and Z is the largest. Therefore, the comparison A<B is true; so is the comparison D>C.

  A blank is smaller than any letter.

  The following statements divide the names of the guides into two groups beginning with A-L and M-Z by combining the comparison operator with the colon.

```
if TourGuide <=: 'L' then TourGuideGroup = 'A-L';
else TourGuideGroup = 'M-Z';
```

- All names beginning with A through L, as well as the missing value, go into group A-L. The missing value goes into that group because a blank is smaller than any letter.

Acting on selected observations (11)

- Finding a value anywhere within another character value

  The INDEX function determines whether a specified character string (the excerpt) is present within a particular character value (the source)

```
INDEX (source,excerpt);
```

- Both source and excerpt can be any kind of character expression, including character strings enclosed in quotation marks, character variables and character functions.
  - If excerpt does occur within source, then the function returns the position of the first character of excerpt, which is a positive number.
  - If it does not, then the function returns a 0.

  The following statements select observations containing the string other:

```
if index(EventDescription,'other') > 0 then
   OtherEvents = 'Yes';
else OtherEvents = 'No';
```

Creating subset of observations (1)

Objective

- Create a new SAS data set that includes only some of the observations from the input data source
- Create several new SAS data sets by writing observations from an input data source,using a single DATA step

Creating subset of observations (2)

- Selecting observations for a new SAS data set
  - Deleting observations based on a condition

    To delete an observation, first identify it with an IF condition, then use a DELETE statement in the THEN clause:

    ```
    IF condition, THEN DELETE;
    ```

    For example,

    ```
    IF  Landcost=.  THEN delete;
    ```

  - Accepting observations based on a condition

    To select only observations meeting the criterion, the subsetting IF statement selects the observations that you specify:

    ```
    IF  condition;
    ```

    For example,

    ```
    IF Nights = 6 ;
    ```

Creating subset of observations (3)

- Comparing the DELETE and subsetting IF statement
  - Main reason for choosing
    - It's usually easier to choose the statement that requires the fewest comparisons to identify the condition.
    - It's ususally easier to think in positive terms than negative terms(This favors the subsetting IF)

Creating subset of observations (4)

- Conditionally writing observations to one or more SAS data sets
  - Understanding the output statement

    Create multiple SAS data sets in a single DATA step using an OUTPUT statement

    ```
    OUTPUT <SAS-data-sets)>;
    ```

    - If not specifying a data set name, SAS writes the current observation to all data sets named in the DATA statement.
    - If specifying a data set name, SAS writes the observation to the selected data set.
    - Any data set name appearing in the output statement must also appear in the DATA step.

Creating subset of observations (5)

- Example for conditionally writing observations to multiple data sets

A

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
Judy	F	14	64.3	90
Mary	F	15	66.5	112
Philip	M	16	72	150
Ronald	M	15	67	133
William	M	15	66.5	112

```
data b c;
   set a;
   if sex='M' then output b;
   else output c;
run;
```

B

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Henry	M	14	63.5	102.5
3	Jeffrey	M	13	62.5	84
4	Philip	M	16	72	150
5	Ronald	M	15	67	133
6	William	M	15	66.5	112

C

Name	Sex	Age	Height	Weight
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Janet	F	15	62.5	112.5
Judy	F	14	64.3	90
Mary	F	15	66.5	112

Creating subset of observations (6)

- If you plan to use any OUTPUT statements in a DATA step, then you must program all output for that step with output statements
- When you use an output statement, you override the automatic output feature, thus where you place the output statement is very important. For example, if a variable value is calculated after the OUTPUT statement executes, then that value is not available when the observation is written to the output data set.
- You can write an observation multiple times to one or more data sets using output statement.

Data View (1)

Definition:

- A SAS data view is a type of SAS data set that retrieves data values from other files.
- A SAS data view contains only descriptor information such as data types and lengths of the variables (columns), plus information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendor's file formats.
- SAS data views are of member type VIEW.
- In most cases, you can use a SAS data view as though it were a SAS data file.

Data View (2)

Types of data views

- Native view

  is a SAS data view that is created either with a DATA step or with PROC SQL.

- Interface view

  is a SAS data view that is created with SAS/ACCESS software.

  - An interface view can read data from or write data to a database management system (DBMS) such as DB2 or ORACLE.
  - Interface views are also referred to as SAS/ACCESS views.

Data View (3)

Benefits of using SAS data views

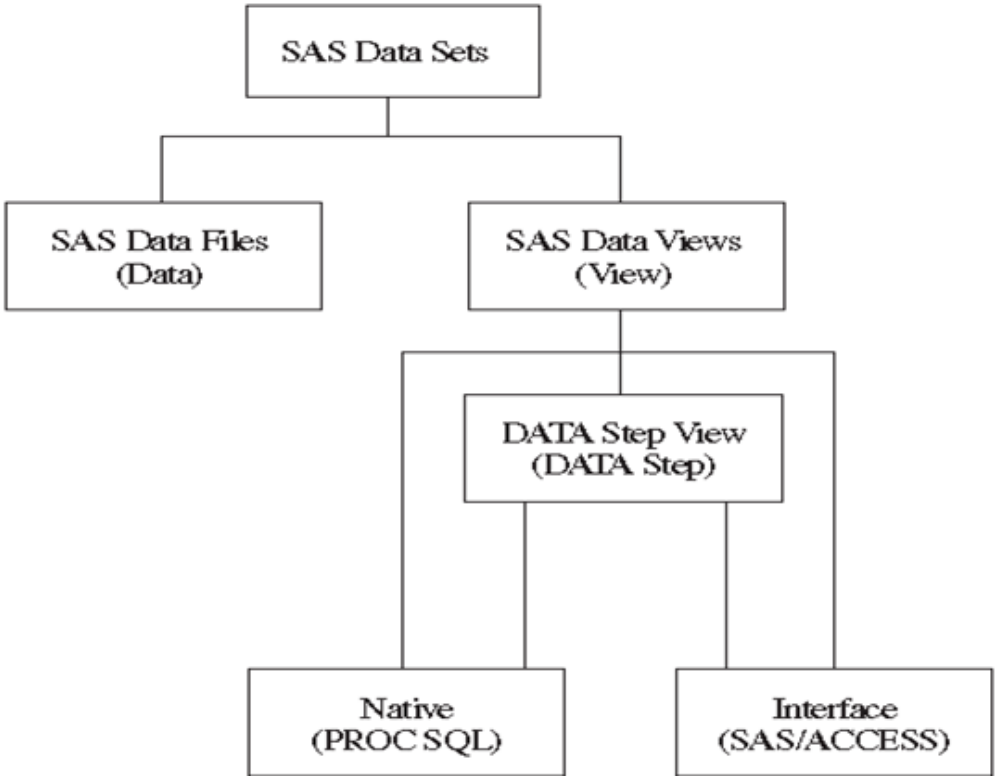
- Instead of using multiple DATA steps to merge SAS data sets by common variables, you can construct a view that performs a multi-table join.
- You can save disk space by storing a view definition, which stores only the instructions for where to find the data and how it is formatted, not the actual data.
- Views can ensure that the input data set are always current because data is derived from views at execution time.
- Since views can select data from many sources, once a view is created, it can provide many prepackaged information to the information community without the need for additional programming.

Data View (4)

- Views can reduce the impact of data design changes on users. For example, you can change a query that is stored in a view without changing the characteristics of the view's result.
- With SAS/CONNECT software, a view can join SAS data sets that reside on different host computers, presenting you with an integrated view of distributed company data.

Data View (5)

- The following figure shows native and interface SAS data views and their relationship to SAS data files:



Data View (6)

- You can use views in the following ways:
  - As an input to other DATA steps or PROC steps
  - To migrate data to SAS data files or to database management systems that are supported by SAS.
  - In combination with other data sources using PROC SQL
  - As pre-assembled sets of data for users of SAS/ASSIST software, enabling them to perform data management, analysis, and reporting tasks regardless of how the data stored.
- When to use SAS data views
  - Data files use additional disk space, data views use additional processing time
  - Data file variables can be sorted and indexed prior to use; data views must process data in its existing form during execution.

Data View (7)

Data step views

- A native view that has the broadest scope of any SAS data view.
- It contains stored DATA step programs that can read data from a variety of sources, including
  - Raw data files
  - SAS data files
  - PROC SQL views
  - SAS/ACCESS views
  - DB2, ORACLE, or other DBMS data

Data View (8)

- Create Data step views
  - Specify the VIEW= option after the final data set name in the DATA statement
  - The VIEW= option tells SAS to compile, but not to execute, the source program and to store the compiled code in the input DATA step view that is named in the option.

For example, the following example create a DATA step view named DEPT.A

```
libname  dept 'SAS-data-library';
data dept.a / view=dept.a;
 … more SAS statements …
run;
```

Data View (9)

- What can you do with a data step view
  - Directly process any file that can be read with an INPUT statement
  - Read other SAS data sets
  - Generate data without using any external data sources and without creating an intermediate SAS data files.
- Differences between Data step views and stored compiled data step programs
  - A Data step view is implicitly executed when it is referenced as an input data set by another data or proc step. Its main purpose is to provide data, one record at a time, to the invoking procedure or Data step
  - A stored compiled DATA step program is explicitly executed when it is specified by the PGM= option on a DATA statement. Its purpose is usually a more specific task, such as creating SAS data files, or originating a report.

Data View (10)

- Restrictions and requirements

  Global statements do not apply to a DATA step view.
- Performance considerations
  - DATA step code executes each time that you use a view. This may add considerable system overhead. In addition, you run the risk of having your data change between steps.
  - Depending on how many reads or passes on the data are required, processing overhead increases.
    - When one pass is requested, no data set is created. Compared to traditional methods of processing, making one pass improves performance by descending the number of input/output operations and elapsed time.
    - When multiple passes are requested, the view must build a spill file that contains all generated observations so that subsequent passes can read the same data that was read by previous passes.

Data View (11)

- Example: Merging data to produce reports

  If you want to merge data from multiple files but you don not need to create a file that contains the combined data, you can create a DATA step view of the combination for use in subsequent applications

```
libname myv9lib 'SAS-data-library';
libname v9lr 'SAS-data-library';
data myv9lib.qtr1 / view=myv9lib.qtr1;
   merge v9lr.clothes v9lr.equip;
   by date;
   total = cl_v9lr + eq_v9lr;
run;
```

- The following PRINT procedure executes the view

```
proc print data=myv9lib.qtr1;
run;
```

Data View (12)

- PROC SQL views
  - A PROC SQL view in a PROC SQL query-expression that is given a name and stored for later use.
  - When you use a PROC SQL view in a SAS program, the view derives its data from the data sets or views listed in its FROM clause.
  - A PROC SQL view can read or write data from:
    - Data step views
    - SAS data files
    - Other PROC SQL views
    - SAS/ACCESS views
    - DB2, ORACLE or other DBMS data

Data View (13)

Comparing data steps and PROC SQL views

- Data step views
  - Data step views are versatile because they use DATA step processing, including DO loops and IF-THEN/ELSE statements
  - Data step views do not have write capability; that is, they cannot directly change the data that they access.
  - There is no way to qualify the data in a DATA step view prior to using it. Therefore, even if you need only part of the data in your data view, you must load into memory the entire DATA step view and discard everything that you do not need.

Data View (14)

- PROC SQL views
  - PROC SQL views can combine data from many different file formats
  - PROC SQL views can both read and update the data that they reference.
  - PROC SQL supports more types of WHERE clauses than are available in DATA step processing and has a CONNECT TO component that allows you to easily send SQL statements and pass data to a DBMS by using the Pass-Through Facility.
  - You can also use the power of the SQL language to subset your data prior to processing it. This saves memory when you have a large view, but need to select only a small portion of the data contained in the view.
  - PROC SQL views do not use DATA step programming.

Data View (15)

SAS/ACCESS views

- A SAS/ACCESS view is an interface view, also called a view descriptor, which accesses DBMS data that is defined in a corresponding access descriptor.
- Using SAS/ACCESS software, you can create an access descriptor and one or more view descriptors in order to define and access some or all of the data described by one DBMS table or DBMS view.
- You can also use view descriptors in order to update DBMS data, with certain restrictions.
- In addition, some SAS/ACCESS products provide a dynamic LIBNAME engine interface.