

STATISTIQUES ET ÉTUDES ÉCONOMIQUES SOUS SAS

**MANIPULATION DES TABLES DE DONNÉES :
LES ÉTAPES DATA**

GÉNÉRALITÉS

DESCRIPTION D'UNE ÉTAPE DATA

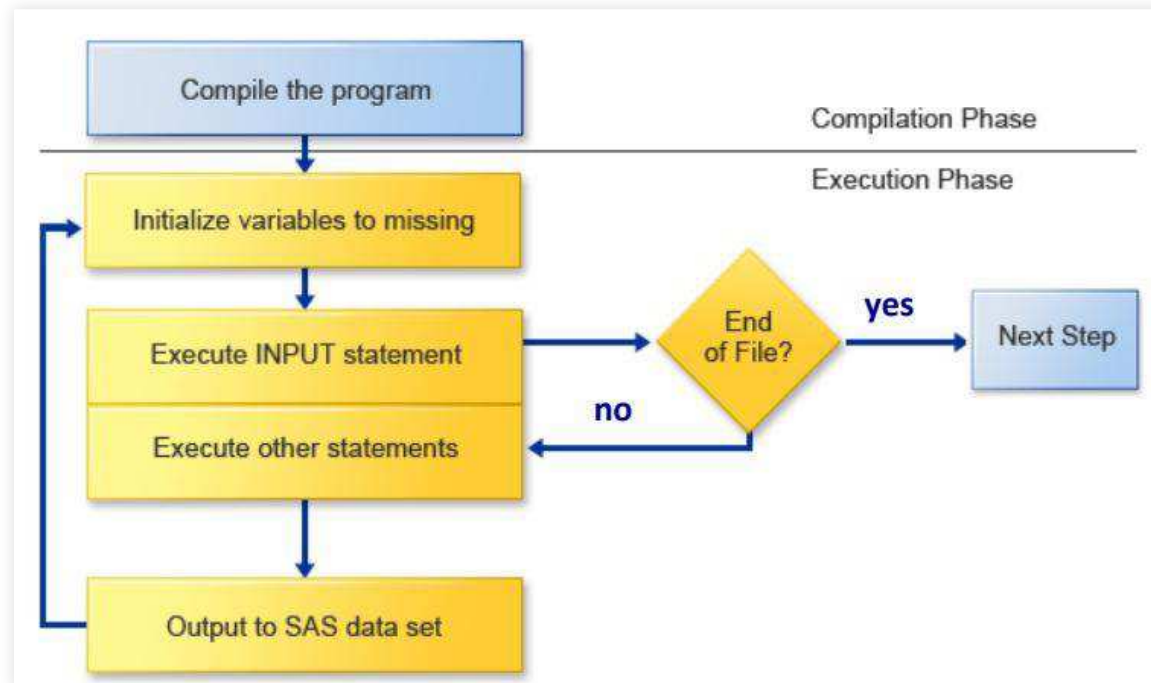
La syntaxe d'une étape DATA est la suivante

```
DATA nom_de_ma_librairie.nom_de_ma_table ;  
    [instructions] ;  
RUN ;
```

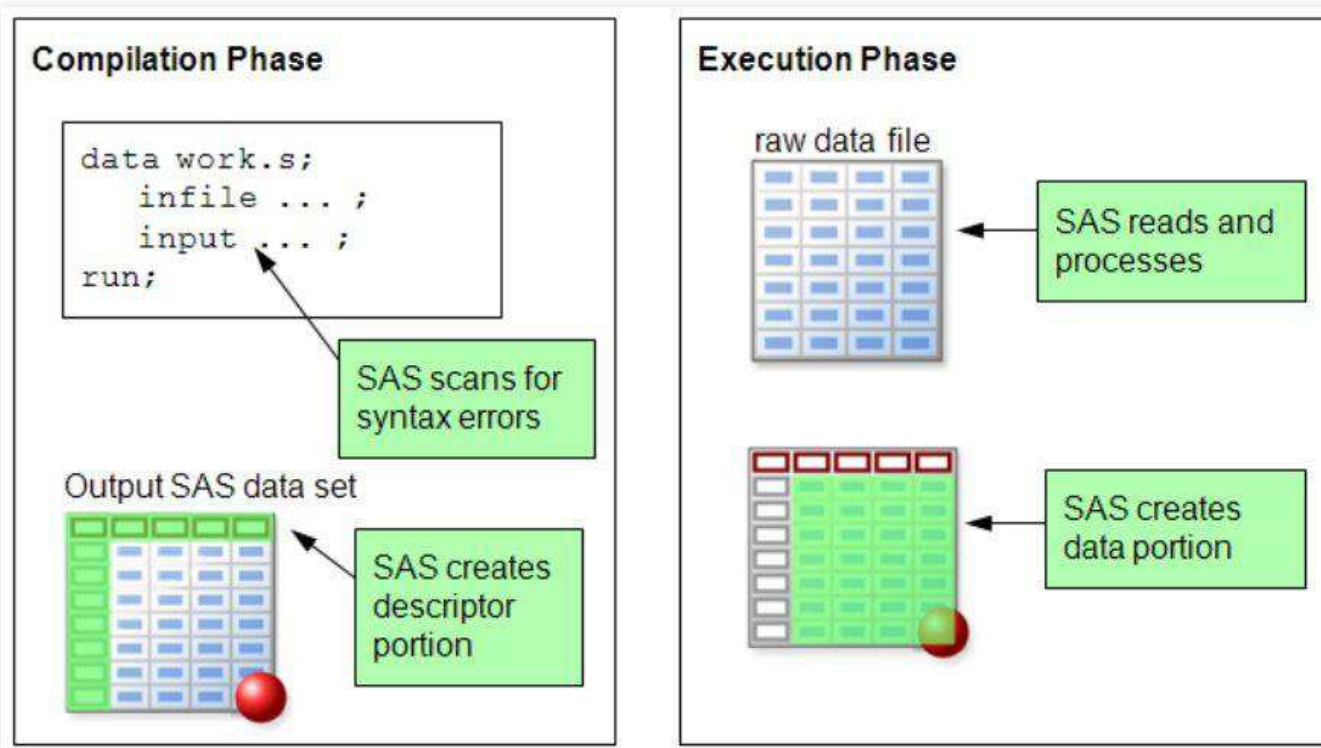
Cette étape consiste à créer une table qui sera stockée dans la
librairie sélectionnée

Par défaut, la table sera stockée dans la librairie temporaire
WORK

DÉROULÉ D'UNE ÉTAPE DATA



COMPILOATION ET ÉXÉCUTION D'UNE ÉTAPE DATA



COMPILATION DATA ILLUSTRATION

data example_x;

x = 'CAT'

y = 'DOG';

run;

```
9  data example_x;
10  x = 'CAT'
11  y = 'DOG';
12  _
22
ERROR 22-322: Erreur de syntaxe, l'une des valeurs suivantes est attendue : !, !!, &, *, **, +, -, /, ;, <, <=, <>, =, >,
><, >=, AND, EQ, GE, GT, IN, LE, LT, MAX, MIN, NE, NG, NL, NOTIN, OR, ^=, |, ||, ~=.

12  run;

NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column).
      10:5  11:5
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.EXAMPLE_X may be incomplete.  When this step was stopped there were 0 observations and 2
variables.
WARNING: Table WORK.EXAMPLE_X non remplacée car cette étape a été interrompue.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

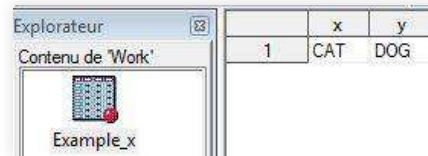
COMPILATION DATA CORRECTION

```
data example_x;  
x = 'CAT';  
y = 'DOG';  
run;
```

```
13 data example_x;  
14 x = 'CAT';  
15 y = 'DOG';  
16 run;  
  
NOTE: The data set WORK.EXAMPLE_X has 1 observations and 2 variables.  
NOTE: DATA statement used (Total process time):  
      real time          0.04 seconds  
      cpu time           0.01 seconds
```

EXECUTION DATA ILLUSTRATION

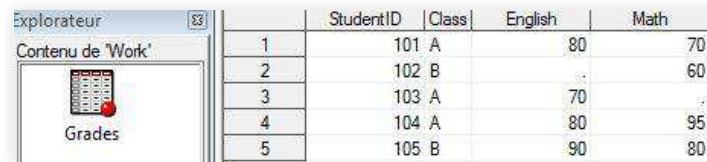
Traitement d'un cas simple (une ligne avec *output* implicite)



	x	y
1	CAT	DOG

Traitement itératif de chaque ligne

```
data grades;  
  StudentID=101; Class="A"; English=80; Math=70; output;  
  StudentID=102; Class="B"; English=. ; Math=60; output;  
  StudentID=103; Class="A"; English=70; Math=. ; output;  
  StudentID=104; Class="A"; English=80; Math=95; output;  
  StudentID=105; Class="B"; English=90; Math=80; output;  
run;
```



	StudentID	Class	English	Math
1	101	A	80	70
2	102	B	.	60
3	103	A	70	.
4	104	A	80	95
5	105	B	90	80

L'INSTRUCTION SET

DUPLICATION DE TABLE

Pour dupliquer une table existante dans une autre librairie

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set nom_ancienne_librairie.nom_ancienne_table ;  
RUN ;
```

Cette étape consiste à dupliquer l'ancienne table dans une autre librairie

SELECTION DE VARIABLE

Il est possible de conserver ou de supprimer des variables d'une table à l'aide des instructions KEEP et DROP

L'INSTRUCTION KEEP

Pour conserver une variable présente dans une table il convient d'utiliser comme suit le mot clé KEEP

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table (keep=nom_de_la_variable_à_conserver ) ;  
RUN ;
```

De manière équivalente, l'étape peut s'écrire comme suit

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table ;  
    keep nom_de_la_variable_à_conserver ;  
RUN ;
```

L'INSTRUCTION DROP

Pour supprimer une variable présente dans une table il convient d'utiliser comme suit le mot clé DROP

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table (drop=nom_de_la_variable_à_supprimer ) ;  
RUN ;
```

De manière équivalente, l'étape peut s'écrire comme suit

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table ;  
    drop nom_de_la_variable_à_supprimer ;  
RUN ;
```

SELECTION DE VARIABLE

Les quatre étapes ci-dessous renvoient la même table en résultat

```
DATA cours2.Class ;  
  
    set sashelp.Class ;  
  
    drop Age Height Weight ;  
  
RUN ;
```

```
DATA cours2.Class ;  
  
    set sashelp.Class (drop= Age Height  
Weight) ;  
RUN ;
```

```
DATA cours2.Class ;  
  
    set sashelp.Class ;  
  
    keep Name Sex ;  
  
RUN ;
```

```
DATA cours2.Class ;  
  
    set sashelp.Class (keep= Name  
Sex) ;  
RUN ;
```

SELECTION DE MODALITÉS OU FILTRAGE D'OBSERVATIONS

Pour conserver ou supprimer des modalités d'une variable présente dans une table, différentes instructions sont envisageables

Les instructions WHERE, OUTPUT et DELETE

L'INSTRUCTION WHERE

Pour conserver uniquement les observations qui respectent une condition, on peut utiliser l'instruction WHERE

Le filtre s'applique à la lecture de la table source

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table (where=[condition] ) ;  
RUN ;
```

De manière équivalente, l'étape peut s'écrire comme suit

Le filtre s'applique à l'écriture de la table cible

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table ;  
where [condition];  
RUN ;
```


LES INSTRUCTION OUTPUT ET DELETE

Pour conserver uniquement les observations qui respectent une condition, on peut utiliser les instructions OUTPUT ou DELETE

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table;  
    If [condition] then OUTPUT;  
RUN ;
```

De manière symétrique, l'étape peut s'écrire comme suit

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table;  
    If [condition] then DELETE;  
RUN ;
```

SELECTION DE VARIABLE

Les quatre étapes ci-dessous renvoient la même table en résultat

<code>DATA</code> cours2.Class_yng ;	<code>DATA</code> cours2.Class_yng ;
<code>set</code> sashelp.Class ;	<code>set</code> sashelp.Class ;
<code>if</code> Age < 15 <code>then</code> OUTPUT ;	<code>if</code> Age >= 15 <code>then</code> DELETE ;
<code>RUN</code> ;	<code>RUN</code> ;
<code>DATA</code> cours2.Class_yng ;	<code>DATA</code> cours2.Class_yng ;
<code>set</code> sashelp.Class (<code>set</code> sashelp.Class ;
<code>where</code> = (age < 15)) ;	<code>where</code> age < 15 ;
<code>RUN</code> ;	<code>RUN</code> ;

LES INSTRUCTIONS OBS ET FIRSTOBS

Il est également possible de sélectionner les variables selon leur numéro de lignes

Ici on choisit les lignes de i à j (j-i+1 observations)

Il faut que j soit supérieur ou égal à i !

```
DATA cours2.Class_5 ;  
    set sashelp.Class (firstobs = i obs = j ) ;  
RUN ;
```

Par *défaut* l'option firstobs est égal à 1, le code ci-dessous renvoie les j premières lignes du tableau

```
DATA cours2.Class_5 ;  
    set sashelp.Class (obs = j ) ;  
RUN ;
```

REMARQUES

Les instructions WHERE, FIRSTOBS, OBS, KEEP et DROP peuvent être des options de l'instruction SET

Les instructions OUTPUT et DELETE ne peuvent pas être utilisées comme des options

REMARQUES

Afin de définir le complémentaire d'une condition il est possible d'utiliser le mot clé NOT devant une condition

```
DATA cours2.Class_yng ;      DATA cours2.Class_yng ;  
    set sashelp.Class ;      set sashelp.Class ;  
    if Age < 15 then OUTPUT ;  if NOT ( Age < 15 ) then DELETE ;  
RUN ;                        RUN ;
```

REMARQUES

L'instruction WHERE doit être unique dans une étape DATA

```
DATA cours2.Class_yngM ;
```

```
set sashelp.Class ;
```

```
where ( Age < 15 and Sex = "M" ) ;
```

```
RUN ;
```

```
DATA cours2.Class_yngM ;
```

```
set sashelp.Class ;
```

```
where Age < 15 ;
```

```
where Sex = "M" ;
```

```
RUN ;
```

Dans le second exemple, seule la seconde instruction WHERE est prise en compte

REMARQUES

Il est également possible de créer plusieurs table en une seule instruction DATA

```
DATA cours2.Class_Garcons cours2.Class_filles ;  
  set sashelp.Class ;  
  if Sex = "M" then OUTPUT cours2.Class_Garcons ;  
  else OUTPUT cours2.Class_filles ;  
RUN ;
```

ATTENTION

Il est possible de modifier une table existante

Pour ce faire, il suffit d'appeler la même table dans l'instruction
DATA et l'instruction SET

La plus grande prudence doit être observée quand on modifie
une table sans quoi des données peuvent être accidentellement
et définitivement supprimées

ATTENTION

On peut supprimer définitivement toutes les observations d'une table en se trompant lors de la conservation des variables ou des modalités

Exemple

```
DATA cours2.Class;  
  set cours2.Class ( where = (Sex = 'Male')) ;  
RUN ;
```

Le système ne génère pas de message d'avertissement

Le programme s'exécute mais il ne retient que les variables telles que définies dans l'instruction where

Or Sex vaut 'M' ou 'F' , mais pas 'Male' ; la table est donc vidée de toutes les observations et ce, de manière définitive

CRÉATION ET MODIFICATION DES ATTRIBUTS D'UNE VARIABLE

CRÉATION DE VARIABLES

Pour créer une variable, SAS n'a besoin que de son nom et lui attribuer une ou plusieurs valeurs à l'aide du symbole =

```
DATA cours2.Class;  
    set cours2.Class ;  
un = 1;  
n = _N_;  
Nom = Name;  
RUN ;
```

CRÉATION DE VARIABLES

Le programme ci-dessus génère trois nouvelles variables, qui seront créées dans trois nouvelles colonnes à droite de la table, et qui prendra pour chaque observation :

- "*un*" aura pour chaque observation la valeur 1 et sera par défaut une variable numérique
- "*n*" aura pour chaque observation la valeur du numéro d'observation (i.e. le numéro de la ligne sur laquelle se trouve l'observation) et sera donc une variable numérique
- "*Nom*" aura comme valeur la valeur et les attributs de la variable Name

L'INSTRUCTION RENAME

Il est possible de renommer une variable à l'aide de l'instruction
RENAME

À la lecture

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table ( rename=(ancien_nom = nouveau_nom) ) ;  
RUN ;
```

De manière équivalente, l'étape peut s'écrire comme suit

À l'écriture

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
    set ancienne_table ;  
    rename ancien_nom = nouveau_nom ;  
RUN ;
```

L'INSTRUCTION LENGTH

Il est possible de modifier la longueur d'une variable à l'aide de l'instruction LENGTH

```
DATA nom_nouvelle_librairie.nom_nouvelle_table ;  
length ma_variable < $ > n ;  
    set ancienne_table ;  
RUN ;
```

Le symbole \$ est utilisé dans le cas de variable qualitative
n est un entier qui représente la taille maximale de la variable

L'INSTRUCTION LENGTH

L'instruction LENGTH s'utilise comme suit

```
DATA cours2.Class ;  
    length Name $45 ;  
    set sashelp.Class ;  
RUN ;
```

LES FONCTIONS PUT ET INPUT

Il est possible de convertir une variable alphanumérique en variable numérique et inversement à l'aide des fonctions PUT et INPUT

LES FONCTIONS PUT ET INPUT

Le programme suivant convertit une variable caractère en numérique

```
DATA nom_de_ma_librairie.nom_de_ma_table ;  
    set nom_ancienne_librairie.nom_ancienne_table ;  
    Format var_num n.w ;  
var_num = input (var_car , n.w) ;  
RUN ;
```

LES FONCTIONS PUT ET INPUT

Le programme suivant convertit une variable numérique en caractère

```
DATA nom_de_ma_librairie.nom_de_ma_table ;  
    set nom_ancienne_librairie.nom_ancienne_table ;  
    Format var_car $n. ;  
var_car = put (var_num , $n.) ;  
RUN ;
```

EXEMPLES DE CES FONCTIONS

A PUT() convertit une variable caractère vers une autre variable caractère.

B PUT() convertit une variable numérique vers une variable caractère avec une valeur numérique .

C PUT() convertit une variable caractère avec un format "user-defined" vers une autre variable caractère.

D INPUT() convertit une variable caractère avec une valeur numérique et un informat vers une v. numérique.

E INPUT() convertit une variable caractère avec une valeur numérique et un informat vers une v. caractère.

F INPUT() convertit une variable caractère avec une valeur numérique et un informat vers une v. numérique.

Function Call	Raw Type	Raw Value	Returned Type	Returned Value
A PUT(name, \$10.);	char, char format	'Richard'	char always	'Richard '
B PUT(age, 4.);	num, num format	30	char always	' 30'
C PUT(name, \$nickname.);	char, char format	'Richard'	char always	'Rick'
D INPUT(agechar, 4.);	char always	'30'	num, num informat	30
E INPUT(agechar, \$4.);	char always	'30'	char, char informat	' 30'
F INPUT(cost,comma7.);	char always	'100,541'	num, num informat	100541

AUTRES INSTRUCTIONS

Il est également possible de modifier le label, le format ou l'informat des variables

LES LABELS

Le label d'une variable est un champ texte, qui accepte les caractères spéciaux, et qui permet de fournir une description de la variable

Si un label est défini, c'est lui qui apparaîtra dans la fenêtre Sortie

Il se définit dans une étape DATA comme suit

```
Label variable="MON_LABEL" ;
```

LES FORMATS

Le mot clé format renvoie au format de
présentation ou de stockage

Bien qu'attribut facultatif il s'avère important

Il peut correspondre à un format prédéfini ou personnalisé

```
Format variable < $ > format n.< d > ;
```

LES INFORMATS

Le mot clé `informat` renvoie au format de **lecture**

Il peut également correspondre à un format prédéfini ou personnalisé

```
Informat variable < $ > informat n.< d > ;
```

FORMAT ET INFORMAT : EXEMPLES

```
data amount;  
input nombre: best8. salaire comma8. quantity dollar10.;  
format nombre salaire quantity dollar12.4;  
cards;  
123 23,333 $2,23,000.66  
124 30,000 $5,55,000.78  
;  
run;
```

Le Système SAS

Obs.	nombre	salaire	quantity
1	\$123.0000	\$23,333.0000	\$223000.6000
2	\$124.0000	\$30,000.0000	\$555000.7000

CONCATÉINATION, FUSION ET MISE À JOUR DE TABLES

LA CONCATÉNATION

La concaténation consiste à l'assemblage verticale de tables

On peut concaténer 2 tables facilement avec l'instruction SET

```
DATA nom_librairie.nom_table_concat ;  
    set table_1 table_2 ;  
RUN ;
```

LA CONCATÉNATION

Pour que le système empile correctement les tables il faut que les variables aient exactement le même nom et soient de même type (numérique ou caractère)

```
DATA cours2.Classe_entiere ;  
    set cours2.Class_Garcons Cours2.Class_filles ;  
RUN ;
```

Afin de prévenir les troncatures, le système renvoie un avertissement si deux variables portent le même nom et sont de même type mais présentent des longueurs différentes

LA CONCATÉNATION

Il est possible d'utiliser l'instruction BY afin que la table concaténée soit triée selon une ou plusieurs variables

Les tables sources doivent pour cela être correctement triée au préalable

LE CONCATÉNATION

On l'obtient avec la syntaxe suivante

```
DATA nom_librairie.nom_table_concat ;  
    set table_1 table_2 ;  
    by var1 var2 ... varn ;  
RUN ;
```

Il est possible de trier par autant de variables que l'on souhaite
Le tri s'effectuera selon l'ordre défini par le programme

LA CONCATÉNATION

Si nous souhaitons trier selon le sexe puis l'age des élèves, nous écrirons le programme suivant

```
DATA cours2.Classe_entiere ;  
  set cours2.Class_Garcons Cours2.Class_filles ;  
  by Sex Age ;  
RUN ;
```

REMARQUE

Le fait que les tables doivent être préalablement triées rend cette option peut utilisée

On préférera trier la table concaténée après l'opération

```
PROC SORT data=cours2.Class_Filles;
```

```
by Sex Age; RUN ;
```

```
PROC SORT data=cours2.Class_Garcons;
```

```
by Sex Age; RUN ;
```

```
DATA cours2.Classe_entiere ;
```

```
set cours2.Filles cours2.Garcons ;
```

```
by Sex Age ;
```

```
RUN ;
```

```
DATA cours2.Classe_entiere ;
```

```
set cours2.Class_Filles cours2.C
```

```
RUN ;
```

```
PROC SORT data=cours2.Classe_entie
```

```
by Sex Age; RUN ;
```

LA CONCATÉNATION

Il est possible de modifier les tables d'entrée en utilisant les options de l'instruction SET

```
DATA cours2.Classe_entiere ;  
  set cours2.Class_Garcons (options) Cours2.Class_filles (options);  
RUN ;
```


EXEMPLE

Nous souhaitons créer une table avec les filles de plus de 15 ans
et les garçons de plus de 1m60

```
DATA cours2.Classe_entiere ;  
    set cours2.Class_Garcons (  
rename =(Height=Taille_en_m) where =(Taille_en_m >= 160/2.54))  
    Cours2.Class_filles (  
rename =(Height=Taille_en_m) where =(Age>=15));  
Taille_en_m = Taille_en_m * 2.54;  
RUN ;
```

LA CONCATÉNATION

Il est possible de **traquer** la table d'où provient les données que l'on vient d'ajouter à l'aide de l'option IN

EXEMPLE

```
DATA cours2.Classe_entiere ;  
    set cours2.Class_Garcons (In = Type)  
    Cours2.Class_filles ;  
If Type = 1 then table_origine = "Garcons" ;  
else table_origine = "Filles" ;  
RUN ;
```

Le mot clé IN crée un booléen qui est égal à 1 si l'individu est contenu dans la table appelée précédemment à l'instruction et 0 sinon

LA FUSION

Nous avons vu jusqu'à maintenant la concaténation, c'est à dire l'ajout d'individus (ou de lignes) en concaténant plusieurs tables

La fusion permet d'ajouter des variables (ou des colonnes) en fusionnant à l'aide de l'instruction MERGE deux tables selon un identifiant qui complètera de manière obligatoire la clause BY

```
DATA nom_de_ma_librairie.table_fusionnée ;  
  Merge table1 table2 ;  
  by Identifiants;  
RUN ;
```

REMARQUE

De même que pour la concaténation, la clause BY indique que les tables doivent impérativement être triées préalablement à l'exécution de l'étape DATA

ATTENTION

Seuls les identifiants doivent être des variables de même noms.
Leurs valeurs sont égales entre les 2 tables.

Si deux variables (non identifiants) portent le même nom dans les deux tables, les données de la première table seront écrasées

Sans clause BY, l'instruction revient à une simple juxtaposition des deux tables

LA FUSION

La fusion consiste à l'assemblage horizontale de tables

```
DATA cours2.Notes_classe ;  
    merge cours2.Class cours2.Note ;  
    by Name ;  
RUN ;
```

LA FUSION

La fusion peut générer des valeurs manquantes dans le cas où un individu est identifié dans une table mais pas dans l'autre

Afin d'éviter ces valeurs manquantes on pourra fusionner en ne conservant que les individus d'une seule table

```
DATA cours2.Notes_classe ;  
  merge cours2.Class cours2.Note_rattrapage ( IN = A);  
  by Name ;  
  if A ;  
RUN ;
```


LA MISE À JOUR OU UPDATE

La mise à jour de table peut être vue comme un cas particulier de la fusion

Elle consiste en l'ajout de données initialement manquantes ou en une actualisation de données contenues dans une table

Elle s'effectue avec l'instruction UPDATE

```
DATA nom_de_ma_librairie.Table_update ;  
    update table1 table2 ;  
    by Identifiants ;  
RUN ;
```

LA MISE À JOUR OU UPDATE

Appliquée à notre exemple, l'update s'utilise comme suit

```
DATA cours2.note_finale ;  
  update cours2.Note cours2.Rattrapages ;  
  by Name ;  
RUN ;
```

L'instruction By nous rappelle une fois encore que les tables doivent au préalable être dûment triées

LES CONDITIONS ET LES BOUCLES

BOUCLE DO

Il existe 3 types de boucles sous SAS

Les boucles finies DO TO

Les boucles "tant que" DO WHILE

Les boucles "jusqu'à" DO UNTIL

LES BOUCLES FINIES

Les boucles finies ont la syntaxe suivante

```
do i = n to m by s ;  
    instruction_1 ;  
    instruction_2 ;  
    instruction_3 ;  
    (...)  
    instruction_n ;  
end ;
```

LES BOUCLES FINIES

Les instructions sont réalisées de manière itérative

A chaque fin de la boucle d'instructions signalée par le mot clé
end i est incrémenté de la valeur s (qui représente le pas)

Par défaut $s = 1$

La boucle s'arrête lorsque i devient strictement supérieur à m

LES BOUCLES DO WHILE

Les boucles "tant que" exécutent les instructions tant que la condition définie est vérifiée

```
do while (condition)
  instruction_1 ;
  instruction_2 ;
  instruction_3 ;
  (...)
  instruction_n ;
end ;
```

LES BOUCLES DO WHILE

Les instructions sont réalisées de manière itérative

A chaque fin de la boucle d'instructions signalée par le mot clé
end la condition est testée

La boucle s'arrête lorsque la condition n'est plus vérifiée

LES BOUCLES DO UNTIL

Les boucles "jusqu'à" exécutent les instructions tant que la condition définie n'est pas vérifiée

```
do until (condition)
  instruction_1 ;
  instruction_2 ;
  instruction_3 ;
  (...)
  instruction_n ;
end ;
```

LES BOUCLES DO UNTIL

Les instructions sont réalisées de manière itérative

A chaque fin de la boucle d'instructions signalée par le mot clé
end la condition est testée

La boucle s'arrête lorsque la condition est vérifiée

NOTE

Dans la suite de ce chapitre toutes les instructions peuvent être remplacées par des boucles d'instructions

LA CONDITION IF THEN ELSE

Certaines instructions doivent être réalisées si une condition est
au préalable remplie

Le mot clé IF permet d'introduire de la conditionnalité

```
if condition(s) then instruction ;  
else instruction ;
```

EXEMPLES

```
DATA cours2.Resultats_promo ;  
    set cours2.Note_Finale;  
    if note >= 10 then Result = "Validé";  
else Result = "Non validé" ;  
RUN ;
```

La clause Else est facultative

```
DATA cours2.Resultats_promo ;  
    set cours2.Note_Finale;  
    if note >= 10 then Output ;  
RUN ;
```

CONDITIONNALITÉ MULTIPLE

On peut définir plusieurs conditions au sein d'une étape data avec ou sans clause Else

```
DATA cours2.Resultats_session1 ;  
  set cours2.Note;  
  if note >= 10 then Result = "Validé";  
    else if note >= 8 then Result = "Rattrapage";  
      else Result = "Non validé" ;  
RUN ;
```

```
DATA cours2.Resultats_session1 ;  
  set cours2.Note;  
  if note >= 8 and note < 10 then Result = "Rattrapage";  
  if note >= 10 then Result = "Validé";  
  if note < 8 then Result = "Non validé" ;  
RUN ;
```

REMARQUES

La clause Else permet d'éviter à avoir à lister toutes les conditions

Elle permet de gérer les cas qui ne respectent pas une condition et d'éviter, dans le cas de création d'une ou plusieurs variables, de générer des valeurs manquantes

REMARQUES

Les conditions peuvent être multiples, et ce avec les opérateurs **and**, **or** et **not**

L'opérateur **and** signifie que les deux conditions doivent être vérifiées

L'opérateur **or** signifie qu'au moins une des conditions doit être vérifiée

L'opérateur **not** signifie que le complémentaire de la condition doit être vérifié

ILLUSTRATION

A	B
1	0
1	1
1	0
0	1
0	0
1	1
0	0

IF A=1 AND B=1

THEN OUTPUT

IF A=1 OR B=1

THEN OUTPUT

IF NOT B=1

THEN OUTPUT

A	B
1	1
1	1

A	B
1	0
1	1
1	0
0	1
1	1

A	B
1	0
1	0
0	0
0	0

REMARQUES

L'instruction après le mot clé **then** ou **else** doit être unique

Il est parfois nécessaire de définir plusieurs instructions

Pour cela, il est possible d'utiliser la "boucle" **do end** comme suit

```
if condition(s) then do;  
    instruction_1 ;  
    instruction_2 ;  
    ... ;  
    instruction_N ;  
end;  
else do;  
    instruction ;  
    instruction_1 ;  
    instruction_2 ;  
    ... ;  
    instruction_N ;  
end;
```

LE BLOC SELECT WHEN

Ces instructions permettent de gérer la conditionnalité multiple sur une variable unique

```
select (variable) ;  
  when condition_1 instruction_1;  
  when condition_2 instruction_2;  
  (...)  
  when condition_n-1 instruction_n-1;  
otherwise instruction_n ;  
end;
```

EXEMPLE

```
DATA cours2.reclassification ;  
  set sashelp.Class;  
  select (Sex) ;  
    when ('F') cat='Female';  
    when ('M') cat='Male';  
  otherwise cat='Indefini' ;  
end;  
RUN ;
```

LE BLOC SELECT WHEN

L'instruction Otherwise est facultative

La variable définie dans l'instruction Select est facultative et les conditions peuvent dès lors être multiples

Est-ce correct ? (longueur de Cat)

```
DATA cours2.reclassification ;  
  set sashelp.Class;  
  select ;  
    when (Sex = 'F' and Age >= 15) Cat='Female';  
    when (Sex = 'F' and Age < 15) Cat='Young Female';  
    when (Sex = 'M' and Age >= 15) Cat='Male';  
    when (Sex = 'M' and Age < 15) Cat='Young Male';  
end;  
RUN ;
```

LE BLOC SELECT WHEN

Dans le script précédent, on a un souci sur le longeur de Cat
Ce problème est ici résolu

```
DATA cours2.reclassification ;  
  set sashelp.Class;  
  select ;  
    when (Sex = 'F' and Age < 15) Cat='Young Female';  
    when (Sex = 'F' and Age >= 15) Cat='Female';  
    when (Sex = 'M' and Age >= 15) Cat='Male';  
    when (Sex = 'M' and Age < 15) Cat='Young Male';  
  end;  
RUN ;
```

CALCUL ET CRÉATION DE VARIABLES

CALCUL DE VARIABLES

Il n'est pas nécessaire de définir les variables en SAS

Le symbole = entraine l'affectation de valeurs à la variable

```
DATA nom_de_ma_librairie.nom_de_ma_table ;  
    nom_de_ma_variable = [expression valide];  
RUN;
```


EXEMPLE

Nous souhaitons calculer l'IMC (Indice de Masse Corporelle) des étudiants

```
DATA cours2.Class_entiere ;  
set cours2.Class_entiere ;  
  Poids = Weight * 0.454545;  
  Taille = (Height * 2.54)/100;  
  IMC = Poids / Taille**2;  
  Conclusion = cat('L étudiant a un IMC de ', round(IMC,0.1));  
RUN;
```

CRÉATION DE TABLES

Il est possible de créer de a à z une table en programmant une étape data ainsi que les commandes Input et Cards

```
DATA nom_de_ma_librairie.nom_de_ma_table ;  
  input nom_de_ma_variable_car $ nom_de_ma_variable_numerique ;  
  Cards;  
  modalité_car_1 modalite_num_1  
  modalité_car_2 modalite_num_2  
  ...  
  modalité_car_N-1 modalite_num_N-1  
  modalité_car_N modalite_num_N  
;  
RUN;
```

EXEMPLE

Les modalités sont séparées par des espaces " ", il est donc impossible de créer des modalités qui en sont composées

Il conviendra de modifier les " " par des "_"

```
DATA Cours2.nouveaux_eleves ;  
  input Name $ Sex $ Age Height Weight ;  
  Cards;  
  Jean-Charles M 21 58.645 102.786  
  Mathilde F 19 52.5 100  
;  
RUN;
```

Les tables sont ainsi créées directement à partir des informations renseignées par l'utilisateur

GÉNÉRATION DE VARIABLES ALÉATOIRES

Il est possible de générer une table contenant un nombre fini de réalisations d'une variable aléatoire

Par exemple, afin d'obtenir un tirage de 100 réalisations de la loi normale centrée réduite, il convient d'écrire le programme suivant

:

```
DATA loi_normale ;  
  Do i = 1 to 100 ;  
    x = rannor(0);  
    Output ;  
  End ;  
RUN;
```

REMARQUES

Ici, `rannor(0)` renvoie une réalisation de la loi normale mais beaucoup d'autres lois sont disponibles sous SAS

Afin de connaître les mots clés des lois aléatoires existantes et leur fonctionnement, l'aide SAS s'avèrera indispensable

Une liste non exhaustive est donnée dans l'introduction de ce cours

REMARQUES

Le mot clé output s'avère dans ce cas indispensable
Sans lui SAS écraserait la valeur générée par la nouvelle valeur et
ne générerait *in fine* qu'une unique valeur de x, correspondant à
la dernière réalisation

CALCUL DE CUMULÉS

Le calcul de cumulés n'est pas instantané avec le logiciel SAS

Le programme ci-dessous ne renverra que des valeurs manquantes

```
DATA cumul ;  
  set sashelp.Class ;  
  cumul = cumul + 1 ;  
RUN;
```

En effet la variable cumul n'existant pas dans la table Class, l'addition entre une valeur manquante et 1 renvoie une valeur manquante

CALCUL DE CUMULÉS

De même, en initialisant la variable Cumul à 0, le programme ne renverra pas le cumul attendu

Le programme ci-dessous ne renverra qu'une colonne de 1

```
DATA cumul ;  
  set sashelp.Class ;  
  cumul = 0 ;  
  cumul = cumul + 1 ;  
RUN;
```

En effet la variable cumul étant égale à 0, l'addition entre une valeur nulle et 1 renvoie 1 à chaque ligne de la table

Ceci est dû au mode de calcul et d'implémentation des lignes dans une table SAS

POUR ALLER PLUS LOIN

Le mode de remplissage des tables SAS fonctionnent comme une suite d'ajouts de lignes indépendants les uns des autres.

Pour palier à cela, il faut une instruction qui permettra de faire le lien entre 2 lignes successives des données entrantes.

L'INSTRUCTION RETAIN

Afin de calculer des cumulés avec SAS, il sera indispensable de maîtriser l'instruction **retain**

```
DATA cumul ;  
    set sashelp.Class ;  
    retain cumul 0 ;  
    cumul = cumul + 1 ;  
RUN;
```

Le programme ci-dessus génère une nouvelle variable numérique nommée cumul égale au numéro de ligne de la table

Le 0 initialise la variable cumul

Le **retain** permet à SAS de conserver en mémoire la dernière valeur de la variable cumul