



# Le fonctionnement interne de l'étape data

Mathieu Gaouette  
Videotron

# Plan

- Introduction
- La base
- La base « en coulisse »
- Contrôle dans l'étape data
- Comparaison avec la Proc SQL

# Introduction

- La majorité d'entre vous avez probablement été introduit à SAS via la Proc SQL.
- À moins d'avoir eu de la formation (ou lu) sur le sujet, la plupart d'entre vous utilisez probablement rarement l'étape data.
- Seriez-vous capable de nommer une particularité de l'étape data que l'on ne peut faire avec une proc SQL?
- La connaissance du fonctionnement interne de l'étape data est un atout pour utiliser celle-ci de manière efficace.

# La base

L'anatomie de l'étape data est relativement simple:

Data **<table(s) à créer >** ;

**<Des trucs! ( Définition sources, fonctions, calculs, ...)>**

Run ;

# La base - Lecture

	Lecture Table SAS	Lecture fichier plat
Identification	<b>Énoncé Set ou merge</b> <set   merge> <i>table1 table2 ...</i> [options];	<b>Énoncé statement</b> <infile> <i>fichier-externe</i> [options];
Lecture donnée	Le même énoncé	Énoncé Input

- Lire des données place l'étape data en mode itératif. L'étape data est donc répétée tant qu'il y a des données à lire...  
la plupart du temps



# La base - Écriture

	Écriture table SAS	Écriture fichier plat
Identification	<b>Énoncé data</b> <i>Data table1 table2 ... ;</i>	<b>Énoncé File</b> <i>&lt;File&gt; fichier-externe [options];</i>
Écriture donnée	<i>output [nom-table] ;</i>	énoncé put



Pour les tables SAS, si aucun énoncé output n'est spécifié alors un énoncé output implicite est produit juste avant l'énoncé « run ».



# En coulisse

- Ordre de traitement
- « Program data vector » (PDV)
- Variables automatiques du PDV
- Exemple détaillé de traitement

# Ordre de traitement

1. Initialisation de l'étape data
2. Si requis, un tampon de lecture est créé (input buffer)
3. Le PDV est créé et initialisé
4. Les tables SAS de sortie sont créées vides

**Ensuite** la première ligne de l'étape data est traitée.



# Pourquoi est-ce si important?

L'emplacement exact de certains énoncés clés n'a aucune importance

Considérons l'étape data suivante:

```
data test_no1 ;  
  val_a = 1 ; val_b = 2 ;  
  if val_a = 3 then do ;  
    drop val_a ;  
  end ;  
  else if val_b = 3 then do ;  
    drop val_b ;  
  end ;  
run ;
```

Aucunes de ces deux sous-sections ne sera traitée

```
NOTE: The data set WORK.TEST_NO1 has 1 observations and 0 variables.  
NOTE: DATA statement used (Total process time):  
      real time           0.01 seconds  
      cpu time            0.01 seconds
```

# Un autre exemple

```
data src_table_1 ;  
    val1_a = 1 ; val1_b = 1 ; val1_c = 1 ;  
run ;  
data src_table_2 ;  
    val2_a = 2 ; val2_b = 2 ; val2_c = 2 ;  
run ;  
data test_no2 ;  
    if "&SYSUSERID." eq 'gaouetm' then set src_table_1 ;  
    else set src_table_2 ;  
run ;
```

```
NOTE: There were 1 observations read from the data set WORK.SRC TABLE 1.  
NOTE: The data set WORK.TEST_NO2 has 1 observations and 6 variables.  
NOTE: DATA statement used (Total process time):  
      real time          0.01 seconds  
      cpu time           0.03 seconds
```

# PDV analysé

- Le PDV devrait être vue comme un croquis de vos données.
- Il contient toutes les variables de vos tables (même celles non conservées) ainsi que deux variables « système » :
  1. `_N_`
  2. `_ERROR_`
- Connaitre ces deux variables est un atout.

# QUIZ

- Quelle est la valeur minimale possible de la variable système `_N_`?

A) 0

**B) 1**

C) Est-ce que ces barres de soulignement font vraiment parti du nom de la variable?

# \_N\_

- Malgré la croyance populaire, \_N\_ ne représente pas le no de l'observation traitée.
- “Each time the DATA step loops past the DATA statement, the variable \_N\_ increments by 1. The value of \_N\_ represents the number of times the DATA step has iterated.” (SAS.com)
- En réalité, c'est plutôt: “The value of \_N\_ represents the number of times the DATA step has iterated plus one.”



# Utilisations fréquentes de `_N_`

- Limiter le nombre d'itérations de l'étape data :

```
if _n_ > 1000 then stop ;
```

- Effectuer une tâche une fois seulement :

```
if _n_ = 1 then do ;
```

```
    <code to be executed one time>
```

```
end ;
```

- Créer une clé unique (ou non) :

```
id_key = _n_ ;
```



# `_ERROR_`

- “is 0 by default but is set to 1 whenever an error is encountered, such as an input data error, a conversion error, or a math error, as in division by 0 or a floating point overflow. You can use the value of this variable to help locate errors in data records and to print an error message to the SAS log.” (SAS.com)

# QUIZ

- Lorsqu'une erreur de type « `_ERROR_` » est générée dans l'étape data, est-ce que celle-ci génère un avertissement (WARNING:) et/ou une erreur (ERROR:) dans le log?

A) Oui

B) Non

C) *Noui*

# Qu'est-ce qui déclenche `_ERROR_`

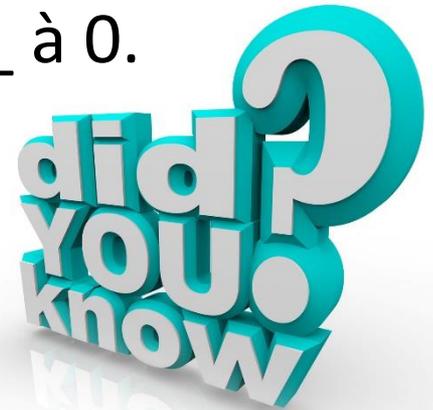
Les situations fréquentes:

- Divisions par zéro
  - Produit une « NOTE: » dans le log
- Référence invalide pour un « array »
  - Produit un « ERROR: » dans le log
- Valeur invalide pour une fonction input/put
  - Produit une « NOTE: » dans le log

# Une note à propos des NOTE:

- La fonction input a une option qui permet d'enlever les erreurs.
  - Un '?' suivi d'un espace avant le format dit à SAS de ne pas imprimer de NOTE.
  - Un double '?' suivi d'un espace avant le format va également laisser la variable `_ERROR_` à 0.

Ex: `date_n = input(date_c, ?? yymmdd10.) ;`



# Exemple détaillé

- Débutons avec deux tables SAS

	 num_key	 num_dates
1	1	20160930
2	2	20160931
3	2	20160101
4	3	20160932

	 num_key	 char_val
1	1	Hello world
2	2	Hello world
3	3	Hello world
4	4	Hello world
5	5	Hello world

...partageant une clé commune

- Nous souhaitons les fusionner puis convertir la variable « num\_dates » en date SAS.

```
data toto ;  
  retain count_obs 0 ;  
  merge src_a(in=a) src_b(in=b) ;  
  count_obs = count_obs + 1 ;  
  by num_key ;  
  if a ;  
  char_nonsense_date = input(put(num_dates,8.),yymmdd8.) ;  
  output ;  
run ;
```

# Gardons un œil sur le PDV

Iteration no 1									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
0	0	0	.	.		.	0	1	<pre>retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date =   input(put(num_dates,8.),yymmdd8.) ; output ;</pre>
0	0	0	.	.		.	0	1	
0	1	1	1	20160930	Hello world	.	0	1	
1	1	1	1	20160930	Hello world	.	0	1	
1	1	1	1	20160930	Hello world	.	0	1	
1	1	1	1	20160930	Hello world	.	0	1	
1	1	1	1	20160930	Hello world	20727	0	1	

- Le champ `count_obs` est initialisé avant que l'énoncé `retain` soit traité.
- Les champs en entrée sont initialisés à manquant avant d'être lus.
- Le champ calculé `Char_nonsense_date` se fait assigner une valeur valide.

# Gardons un œil sur le PDV

Iteration no 2									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
1	1	1	1	20160930	Hello world	.	0	2	retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date = input(put(num_dates,8.),yymmdd8.) ; output ;
1	1	1	1	20160930	Hello world	.	0	2	
1	1	1	2	20160931	Hello world	.	0	2	
2	1	1	2	20160931	Hello world	.	0	2	
2	1	1	2	20160931	Hello world	.	0	2	
2	1	1	2	20160931	Hello world	.	0	2	
2	1	1	2	20160931	Hello world	.	1	2	

- La première ligne de données est conservée dans le PDV jusqu'à ce que l'on traite l'énoncé merge.
- La conversion de date échoue alors \_ERROR\_ prend la valeur 1 et la note suivante est affichée dans le log:

**NOTE: Invalid argument to function INPUT at line 53 column 26.**

# Gardons un œil sur le PDV

Iteration no 3									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
2	1	1	2	20160931	Hello world	.	0	3	<pre>retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date =   input(put(num_dates,8.),yymmdd8.) ; output ;</pre>
2	1	1	2	20160931	Hello world	.	0	3	
2	1	1	2	20160101	Hello world	.	0	3	
3	1	1	2	20160101	Hello world	.	0	3	
3	1	1	2	20160101	Hello world	.	0	3	
3	1	1	2	20160101	Hello world	.	0	3	
3	1	1	2	20160101	Hello world	20454	0	3	

- Variable système `_ERROR_` réinitialisée à 0.
- Seconde ligne de donnée pour la clé `num_key=2` lue (Il n'y a que `num_dates` qui change).

# Gardons un œil sur le PDV

Iteration no 4									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
3	1	1	2	20160101	Hello world	.	0	4	retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date = input(put(num_dates,8.),yymmdd8.) ; output ;
3	1	1	2	20160101	Hello world	.	0	4	
3	1	1	3	20160932	Hello world	.	0	4	
4	1	1	3	20160932	Hello world	.	0	4	
4	1	1	3	20160932	Hello world	.	0	4	
4	1	1	3	20160932	Hello world	.	1	4	

- Cette itération est semblable à la seconde.
- Une nouvelle ligne de donnée est lue de chacune des deux sources (num\_key suivant).
- Une erreur est à nouveau rencontrée lors de la conversion d'une date invalide.

# Gardons un œil sur le PDV

Iteration no 5									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
4	1	1	3	20160932	Hello world	.	0	5	<pre>retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date =   input(put(num_dates,8.),yymmdd8.) ; output ;</pre>
4	1	1	3	20160932	Hello world	.	0	5	
4	0	1	4	.	Hello world	.	0	5	
5	0	1	4	.	Hello world	.	0	5	
5	0	1	4	.	Hello world	.	0	5	

- `_ERROR_` initialisé à nouveau.
- Données manquantes pour les champs de la table `src_a` puisque celle-ci ne contient pas la clé `num_key=4`.
- Puisque la variable « `in src_a` » vaut 0, l'itération arrête ici.

# Gardons un œil sur le PDV

## Iteration no 6

count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
5	0	1	4	.	Hello world	.	0	6	<pre>retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date =   input(put(num_dates,8.),yymmdd8.) ; output ;</pre>
5	0	1	4	.	Hello world	.	0	6	
5	0	1	5	.	Hello world	.	0	6	
6	0	1	5	.	Hello world	.	0	6	
6	0	1	5	.	Hello world	.	0	6	

- Données manquantes pour les champs de la table src\_a puisque celle-ci ne contient pas la clé num\_key=5.
- Puisque la variable « in src\_a » vaut 0, l'itération arrête ici puis l'étape data est terminée...

Du moins presque...



# Gardons un œil sur le PDV

Iteration no 7									
count_obs	a	b	num_key	num_dates	char_val	char_nonsense_date	_ERROR_	_N_	Program step (PVD values taken before step)
6	0	1	5	.	Hello world	.	0	7	<pre>retain count_obs 0 ; merge src_a(in=a) src_b(in=b) ; count_obs = count_obs + 1 ; by num_key ; if a ; char_nonsense_date =   input(put(num_dates,8.),yymmdd8.) ; output ;</pre>
6	0	1	5	.	Hello world	.	0	7	

- SAS boucle à nouveau pour lire les sources.
- Puisque SAS ne peut lire aucune donnée, le traitement de l'itération présente est interrompue.

# Contrôle dans l'étape data

- Le traitement conditionnel et les boucles sont de grandes forces de l'étape data.
- L'étape data de base va de haut en bas une ligne à la fois.
- Les boucles et les conditions permettent d'exécuter certains énoncés à plusieurs reprises ou même pas du tout pour certaines itérations.



Étape data

VS

Proc SQL

# Cote à cote

	Data step	Proc SQL
Jointures	Requière tri/index	Aucun requis* <input checked="" type="checkbox"/>
Unions	Oui <input checked="" type="checkbox"/>	Pas d'entrelacement
Écriture	Sorties multiples <input checked="" type="checkbox"/>	Sortie simple
Traitement conditionnel	Puissant avec peu de code <input checked="" type="checkbox"/>	Puissant mais avec un coût important sur la complexité
Aggrégations	Manuel et requière tri	Pas vraiment de limites <input checked="" type="checkbox"/>
Utilisation work	Minimal* <input checked="" type="checkbox"/>	Variable



# Qui gagne?

- Aucun, ça dépend du contexte.
- Apprenez à utiliser les deux.
- Utilisez la proc SQL pour simplifier les programmes en jumelant plusieurs tâches en une lorsque le volume de données est petit ou moyen.
- Utilisez l'étape data pour les volumes de données importants requérant des traitements conditionnels.
- De toute façon, personne ne veut vraiment voir un mécanicien combattre une vieille dame.

# Références

- <http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000961108.htm>
- <http://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#p0e0mk25gs9binn1s9jiu4otau29.htm>
- <http://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#n1g8q3l1j2z1hjn1gj1hln0ci5gn.htm>

# Ce que j'aurais aimé couvrir!

- Utilisation de multiples énoncés « set » ou « merge ».
- Joindre des tables en utilisant les formats et les tables hash.
- Travailler sur plusieurs lignes de données à la fois (avec les énoncés retains ou lag).
- L'utilisation des « arrays ».
- Les vues SAS pour traitements data en chaîne de façon efficace.



