

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

SAS propose désormais deux objets composants prédéfinis destinés à l'étape DATA : l'objet de hachage et l'objet de hachage itérateur. Ces objets vous permettent de stocker, de rechercher et de récupérer efficacement des données à l'aide de clés de consultation.

L'interface des objets composants de l'étape DATA vous permet de créer et de manipuler ces objets composants à l'aide d'instructions, d'attributs et de méthodes. Vous pouvez utiliser la marque du point pour accéder aux attributs et méthodes de l'objet composant afin :

- D'assurer le stockage en mémoire et la récupération des données à l'aide de l'objet de hachage.
- De définir un composant données et un composant clé (table et index).
- De charger des données dans l'objet de hachage à partir d'une table SAS (les données d'entrée n'ont pas besoin d'être triées).
- De consulter une ligne de données en fonction des valeurs clé.
- D'extraire des données en ordre de tri à l'aide de l'objet de hachage itérateur.
- D'ajouter ou de supprimer des lignes de données de façon dynamique.

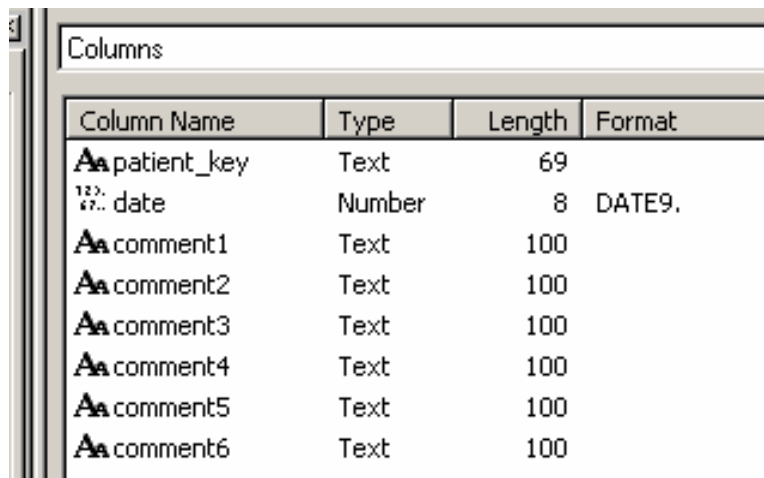
Un attribut, 14 méthodes et 2 instructions sont associés aux objets de hachage et de hachage itérateur. Toute la documentation connexe se trouve dans la documentation en ligne de SAS. Cet article présente des exemples pratiques d'utilisation de ces méthodes et instructions.

Affectation de clés de substitution à l'aide d'un objet de hachage

Des clés de substitution sont définies et utilisées lorsque la clé naturelle associée à une ligne dans un tableau provoque des difficultés. Par exemple, même si un numéro de compte est retiré au moment de la fermeture du compte, il est possible d'affecter l'ancien numéro à un nouveau compte après une période d'attente. Le cas échéant, la clé naturelle, soit le numéro de compte, n'est pas une clé unique au fil du temps. Il faut donc créer une clé de substitution qui est unique au fil du temps.

Dans cet exemple, on utilise une table SAS existante, *datamart.oldmaster*, qui comporte des données relatives à la santé de chaque patient à différentes dates. Chaque patient est identifié à l'aide d'une clé de patient unique comptant 69 caractères. Même si le caractère unique de la clé du patient ne pose pas de problème, le travailleur du savoir voudra remplacer cette longue clé par une clé de substitution numérique unique avant d'analyser les données.

Dans cet exemple, la structure de table *datamart.oldmaster* est la suivante :



Column Name	Type	Length	Format
patient_key	Text	69	
date	Number	8	DATE9.
comment1	Text	100	
comment2	Text	100	
comment3	Text	100	
comment4	Text	100	
comment5	Text	100	
comment6	Text	100	

On génère les clés de substitution de la façon suivante :

1. En reconnaissant une clé de patient qui n'a pas encore été rencontrée.
2. En affectant une valeur entière unique, l'ID du patient, à cette clé de patient.
3. En appliquant ce même ID de patient à toutes les autres lignes appartenant à cette clé de patient.

Pour de petites quantités de données, il est possible de regrouper la table d'entrée par patient à l'aide de l'étape PROC SORT. Dans une étape DATA subséquente, une valeur entière différente est affectée à la première ligne de chaque groupe, puis transférée à toutes les autres lignes du groupe. Le code qui permet d'y parvenir est affiché à la fin de cette section.

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

Pour les grandes quantités de données, l'étape PROC SORT est trop coûteuse. Par contre, un objet de hachage peut identifier efficacement le début d'un nouveau groupe sans avoir à trier la table initiale. Plus le nombre de lignes par patient est élevé, plus l'objet de hachage devient efficace.

Le code SAS permettant de mettre ce processus en œuvre à l'aide d'un objet de hachage est le suivant :

```
data datamart.newmaster(drop=nextID patient_key rc) ;
  retain nextID 1;
  length rc 8 patient_ID 8 patient_key $ 69;
  if _n_ = 1 then do;
    /* 1. create an empty Hash object */
    declare hash patients( );
    patients.definekey ("patient_key");
    patients.definedata ("patient_ID");
    patients.definedone( );
  end;
  /* 2. read the input data set sequentially*/
  set datamart.oldmaster;
  /* 3. look for a matching patient_id in Hash object */
  rc = patients.find( );
  if rc = 0 then do;
    /* 4. match found, store row with this surrogate key */
    output;
  end;
  else do;
    /* 5. match not found, create and store surrogate key */
    patient_ID = nextID;
    output;
    patients.add( );
    nextID + 1;
  end;
run;
```

Notez les instructions accompagnées de commentaires dans le code.

❖ Commentaire 1 :

- L'objet de hachage vide est créé une fois, d'où l'instruction « if _n_=1 then do; ».
- L'instruction « declare hash ... » affecte le nom « patients » à l'objet de hachage.
- L'objet de hachage vide est créé à l'aide de méthodes.
 - « Patients.definekey(...) » affecte une variable de clé à l'objet de hachage.
 - « Patients.definedata(...) » affecte une variable de données à l'objet de hachage.
 - « Patients.definedone() » crée l'objet de hachage vide en mémoire.

❖ Commentaire 2 :

- La table maîtresse (ici, datamart.oldmaster) est lue de façon séquentielle.

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

- ❖ **Commentaire 3 :**
 - Une autre méthode, « patient.find() », permet de rechercher une correspondance à patient_key.
- ❖ **Commentaire 4 :**
 - Si le code de retour (rc) = 0, cela signifie que la recherche a été fructueuse et que la ligne de sortie comporte une valeur correspondant à la variable ID du patient.
- ❖ **Commentaire 5 :**
 - nextID comporte la valeur suivante de la clé de substitution, soit patient_ID.
 - Une autre méthode, « patient.add() », permet d'ajouter une nouvelle ligne à l'objet de hachage.

Un objet de hachage crée une zone de stockage dynamique en mémoire pouvant emmagasiner chacune des paires clé de patient – ID de patient unique à mesure qu'elles sont créées. Au moment de la lecture de chaque ligne, une requête est soumise à l'objet de hachage pour déterminer si la clé de patient s'y trouve déjà. Si la clé de patient est repérée, la valeur ID du patient correspondante est retournée. Sinon, une nouvelle valeur ID du patient est générée et ajoutée à l'objet de hachage à croissance dynamique.

Dans cet exemple particulier, l'objet de hachage se comporte presque comme un format SAS très rapide. Mais contrairement à un format, l'objet de hachage peut être modifié au fil de son utilisation.

Donc, dans quelle mesure l'utilisation de l'objet de hachage est-elle efficace pour générer des clés de substitution ? Un banc d'essai a été exécuté au moyen de plusieurs tables comptant différents nombres de patients, de lignes par patients et de variables. L'objet de hachage et le tri ont tous deux été exécutés dans SAS^{MD9} sous Windows XP.

Comme le démontre le tableau d'utilisation de l'UC ci-dessous, les ressources nécessaires à l'utilisation de l'objet de hachage augmentent de façon linéaire par rapport au nombre de lignes dans la table d'entrée, mais augmentent beaucoup plus rapidement dans le cas du tri.

<i>Nombre de lignes</i>	<i>Nombre de patients</i>	<i>UC - Hachage</i>	<i>UC - Tri</i>
4374	729	0,03	0,06
86400	1728	0,17	1,17
168750	3375	0,32	2,85
532400	10648	1,06	8,58

La différence est encore plus marquée lorsqu'on compare le temps réel (ou écoulé). La différence est partiellement attribuable aux ressources d'E-S utilisées durant le tri, et reflète en partie les E-S nécessaires pour lire deux fois une table volumineuse.

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

<i>Nombre de lignes</i>	<i>Nombre de patients</i>	<i>Temps réel - Hachage</i>	<i>Temps réel - Tri</i>
4374	729	0,03	0,06
86400	1728	0,17	4,46
168750	3375	0,32	9,20
532400	10648	1,09	20,81

Enfin, le code SAS utilisé pour créer des clés de substitution à l'aide de l'étape PROC SORT est la suivante :

```
proc sort data=datamart.oldmaster out=sorted;
  by patient_key;
run;
data datamart.newmaster;
  set sorted;
  by patient_key;
  retain patient_id;
  if first.patient_key then patient_id = _n_;
  drop patient_key;
run;
```

Tel qu'indiqué au début de cette section, ce code regroupe la table d'entrée par clé de patient à l'aide de l'étape PROC SORT. Dans l'étape DATA, une valeur entière est attribuée à la valeur patient_ID au moment de la lecture de la première ligne d'un groupe. L'instruction RETAIN rend cette valeur disponible à toutes les autres lignes de ce groupe.

Repérage d'une valeur minimum dans une table de consultation

Dans cet exemple, on commence par des emplacements de clients et des emplacements de succursales. Le défi consiste à déterminer quelle est la succursale la plus rapprochée de chaque client. Les coordonnées de latitude et de longitude sont attribuées à chaque emplacement en fonction du code postal figurant dans l'adresse.

L'ID et l'emplacement de chaque client sont stockés dans la table *Work.clients*, dont la structure est la suivante :

Column Name	Type	Length	Format
clientID	Number	8	
lat	Number	8	
long	Number	8	

Le numéro de transit et l'emplacement de chaque succursale sont stockés dans la table *Work.banks*, dont la structure est la suivante :

Column Name	Type	Length	Format
transit	Text	4	
latbank	Number	8	
longbank	Number	8	

Différentes mesures peuvent être utilisées pour déterminer la distance entre deux points. Dans cet exemple, nous utiliserons une version simplifiée de la formule exacte fondée sur les coordonnées de latitude et de longitude.

Pour de petites quantités de données, il est possible d'utiliser une requête SQL. On entend ici par « petit » quelques milliers de clients et quelques centaines de succursales. La requête SQL forme un produit cartésien de tous les clients et de toutes les banques, calcule la distance entre chaque client et chaque succursale, et classe les résultats. L'étape DATA extrait la ligne de chaque client affichant la distance la plus courte. Malheureusement, même si le code SQL est facile à rédiger, il est inefficace et, tôt ou tard, manquera d'espace de travail lorsque le nombre de clients s'approche du million. Le code connexe est affiché à la fin de cette section.

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

La solution la plus efficace consiste à utiliser soit plusieurs tableaux (*arrays*) ou un seul objet de hachage pour stocker les données des succursales et boucler sur les succursales en mémoire pour chaque client. La distance est encore calculée pour chaque succursale, mais la valeur minimum est générée de façon ponctuelle. Aussi, cela nécessite très peu d'espace de travail, peu importe le nombre de clients.

Le code SAS permettant de mettre cet exemple en œuvre à l'aide d'un objet de hachage est le suivant :

```
2 data closest_bankH(keep = clientid mindist transit);
3   length rc 8 minbank $ 4;
4   /* create and load hash object */
5   if _n_ = 1 then do;
6     if 0 then set work.banks(keep=transit latbank longbank);
7     declare hash banks(dataset="work.banks");
8     banks.definekey ("transit");
9     banks.definedata ("transit", "latbank", "longbank");
10    banks.definedone( );
11    declare hiter retrieve('banks');
12  end;
13  /* search for a minimum distance */
14  set work.clients;
15  mindist=100000000;
16  rc = retrieve.first();
17  do while(rc = 0);
18    dist = sqrt(({lat - latbank})**2 + ({long - longbank})**2);
19    if dist lt mindist then do;
20      mindist = dist;
21      minbank = transit;
22    end;
23    rc = retrieve.next();
24  end;
25  /* output the result for this client */
26  transit = minbank;
27  mindist = round(mindist, .01);
28  output;
29  run;
30
```

Ces notes renvoient aux numéros de ligne figurant à gauche.

(5-12) « if _n_ = 1 » fait en sorte que l'objet de hachage est créé et chargé avec les données une seule fois.

(6) « if 0 then set » indique au compilateur SAS d'ajouter les variables transit, latbank et longbank au vecteur de données du programme (PDV), accompagnées des attributs appropriés.

(7) L'instruction « declare hash » nomme l'objet de hachage et demande que les données provenant de la table « work.banks » y soient chargées.

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

- (8) La méthode « banks.definekey » précise la variable clé destinée aux recherches.
- (9) La méthode « banks.definedata » précise les variables de données retournées par l'objet de hachage.
- (10) La méthode « banks.definedone » crée l'objet de hachage et charge les données.
- (11) « declare hiter » crée un objet de hachage itérateur relié à l'objet de hachage « banks ».
- (14) Lire les données des clients de façon séquentielle.
- (16) La méthode « retrieve.first » retourne la première ligne de l'objet de hachage relié à l'objet de hachage itérateur. Le code de retour (rc) est 0 si la lecture est fructueuse.
- (17-24) Continuer le traitement jusqu'à ce que toutes les lignes de l'objet de hachage ont été retournées.
- (18) Calculer la distance entre les points à l'aide d'une formule métrique.
- (19-22) Si la nouvelle distance est la distance minimum, stockez les nouvelles valeurs.
- (23) La méthode « retrieve.next » retourne la ligne suivante de l'objet de hachage relié à l'objet de hachage itérateur. Le code de retour (rc) est 0 si la lecture est fructueuse.
- (26-28) Indiquer la succursale la plus rapprochée de ce client.

Un banc d'essai a été exécuté au moyen de plusieurs tables comptant différents nombres de clients. Une exécution comptant trois fois plus de succursales a été effectuée. Le tableau suivant indiquant l'utilisation de l'UC compare la technique utilisant l'objet de hachage avec la technique utilisant des tableaux (*arrays*) et la technique utilisant SQL :

<i>N^{bre} de clients</i>	<i>N^{bre} de succ.</i>	<i>UC-Hachage</i>	<i>UC-Tableaux</i>	<i>UC-SQL</i>
1000	200	0,08	0,05	0,54
2000	200	0,14	0,08	1,08
8000	200	0,48	0,24	4,54
25000	200	1,43	0,70	14,46
25000	600	4,19	2,20	44,12

La différence en temps réel (ou écoulé) présente une tendance semblable :

<i>N^{bre} de clients</i>	<i>N^{bre} de succ.</i>	<i>réel-Hachage</i>	<i>réel-Tableaux</i>	<i>réel-SQL</i>
1000	200	0,08	0,05	0,45
2000	200	0,14	0,08	1,20
8000	200	0,48	0,24	7,93
25000	200	1,44	0,70	31,20
25000	600	4,20	2,20	49,47

À noter que dans cet exemple particulier, les tableaux (*arrays*) se révèlent plus efficaces qu'un objet de hachage. Ici, il n'est pas nécessaire de repérer des lignes dans l'objet de

Évitez les tris coûteux en utilisant une table de consultation dynamique
par Bill Fehlner, Institut SAS
Avril 2006

hachage en fonction d'une clé alphanumérique. Dans d'autres étapes DATA, il est nécessaire de traiter tous les éléments d'une collection à un endroit donné dans le code et de repérer des éléments individuels en fonction d'une clé à un autre endroit donné. Un objet de hachage se prête parfaitement à ce genre de situation.

Le code SAS permettant de mettre cet exemple en œuvre à l'aide de plusieurs tableaux (*arrays*) est le suivant :

```
1
2 data closest_bank(keep = clientid mindist transit);
3     /* set up arrays to hold branch information */
4     array transitno (1000) $ 4 _temporary_;
5     array latitude (1000) _temporary_;
6     array longitude (1000) _temporary_;
7     /* store bank information in arrays */
8     if _n_=1 then do bank = 1 to totobs;
9         set work.banks nobs=totobs;
10        transitno(bank) = transit;
11        latitude(bank)= latbank;
12        longitude(bank)= longbank;
13    end;
14    /* search for a minimum distance */
15    set work.clients;
16    mindist=100000000;
17    do bank = 1 to totobs;
18        dist = sqrt((lat - latitude(bank))**2 +
19                (long - longitude(bank))**2);
20        if dist lt mindist then do;
21            mindist = dist;
22            minbank = bank;
23        end;
24    end;
25    /* output the result for this client */
26    transit = transitno(minbank);
27    mindist = round(mindist,.01);
28    output;
29 run;
```

Ces notes renvoient aux numéros de ligne figurant à gauche.

- (4-6) Définir plusieurs tableaux (*arrays*) puisqu'un seul ne peut stocker des valeurs de données alphanumériques. Cela n'est pas une restriction dans le cas d'un objet de hachage.
- (8-13) Charger les tableaux (*arrays*) à partir de la table. À noter que la taille maximale des tableaux doit être connue à l'avance. Cela n'est pas nécessaire dans le cas d'un objet de hachage.
- (18-19) Chaque référence de tableau retourne une valeur unique. La méthode « find » d'un objet de hachage ou la méthode « next » d'un objet de hachage itérateur retourne une ligne complète de valeurs.

Le code SAS permettant de mettre cet exemple en œuvre à l'aide de SQL est le suivant :

```
2 proc sql;
3   create table work.temp as
4   select clientid, transit,
5          sqrt((lat - latbank)**2 + (long - longbank)**2)
6          as distance
7   from work.clients, work.banks
8   order by clientid, distance
9   ;
10 quit;
11
12 data closest_bank;
13   set work.temp(rename=(distance = mindist));
14   by clientid;
15   if first.clientid then do;
16     mindist = round(mindist, .01);
17     output;
18   end;
19 run;
20
```

Il s'agit de code SQL typiquement concis. Toutefois, l'utilisation d'un produit cartésien le rend inefficace ou impossible à utiliser avec des tables volumineuses.

Conclusion :

L'objet de hachage est un mécanisme efficace et pratique pour le stockage et la récupération rapides des données. La nature dynamique de l'objet de hachage, de même que sa capacité d'utiliser des clés alphanumériques et des clés composites, le distinguent des tableaux (*arrays*). Les deux exemples ci-inclus et l'exemple publié dans le numéro de juillet 2005 du bulletin « Insights » illustrent comment l'utilisation d'un objet de hachage peut éliminer le besoin d'effectuer une étape de tri importante.