Procédure SQL de SAS

Résumé

Cette vignette décrit l'usage de la procédure SQL qui permet l'interrogation et la gestion de tables SAS à l'aide du langage de requête standard de bases de données relationnelles.

Plan des tuteuriels :

- *Prise en main*
- *Gestion des données*
- *Graphiques*
- *Macros-commandes*
- *Bases de données*

Les procédures du module SAS/STAT sont étudiées dans les cours de statistique afférents.

1 Introduction

1.1 Langage SQL

Le langage SQL (Structured Query Language) est un langage d'interrogation de bases de données standardisé commun à la plupart des logiciels de base de données. La procédure **sql** étudiée dans cette vignette en constitue une implémentation dans la version 9.3 de SAS. Cette procédure permet d'extraire, corriger et mettre à jour des données dans une table SAS, souvent plus rapidement que par une étape data.

Le terme de *table* désigne toujours une table SAS, correspondant à un stockage de données propre à ce logiciel. On utilise également deux nouveaux types d'objets : les *vues* et les *index*. Une vue est le stockage d'une interrogation ou ensemble de requêtes : elle contient la description ou définition d'une table virtuelle. Une vue est donc une interrogation à laquelle on donne un nom, pour son usage ultérieur dans une autre procédure SAS. Le principal intérêt de définir une vue est le gain d'espace mémoire. Un index est un système de pointeurs permettant dans certains cas d'accéder plus rapidement aux informations contenues dans une table SAS.

1.2 La base de données

Création

Exécuter le programme `tuteur-sql.sas` du répertoire wikistat/data qui crée quatre tables SAS contenant des informations sur les ventes d'un grossiste en articles de sport. Consulter les tables afin de retrouver les contenus décrits ci-dessous.

Description

Les informations disponibles sont les suivantes :

1. Dans la table `produit`, on trouve dans cet ordre, les nom, coût de production et coût figurant dans le catalogue de vente des différents articles vendus par le grossiste.
2. Dans la table `client`, on trouve les noms des magasins revendeurs qui se fournissent chez le grossiste, le numéro de client de chaque magasin et sa ville d'implantation. Par exemple, il y a trois magasins Flots Bleux, leurs numéros sont 3,8 et 11, et ils sont situés à Hendaye (un magasin) et La Torche (deux magasins).
3. Dans la table `employe`, on trouve le numéro de chaque employée du grossiste, son nom, son ancienneté, sa ville, sa fonction et le numéro de l'employé qui le supervise. Par exemple, Jeanne (numéro 201, manager) est supervisée par Filémon (101, directeur). Jeanne supervise Albert, Julien, Monique et Alain.
4. Dans la table `facture`, on trouve, pour un certain nombre de ventes réalisées par le grossiste, le numéro de facture, le nom du magasin-client, son numéro, le numéro de l'employé du grossiste qui a établi la facture, le matériel vendu, la quantité vendue, et le prix de facturation unitaire (éventuellement différent du prix catalogue, car il peut prendre en compte des remises ou un coût de livraison).

1.3 Syntaxe de la procédure

Commandes

proc sql <options> ;

alter table déclaration de modification ;

create table déclaration de création ;
delete décl-destruction ;
describe décl-description ;
drop décl-suppression ;
insert décl-insertion ;
reset <options> ;
select décl-sélection ;
update décl-mise à jour ;
validate décl-évaluation ;

Remarques :

- il est inutile de répéter l'instruction **proc sql** avant chaque déclaration, sauf si l'on exécute une étape data ou si l'on fait appel à une autre procédure entre deux commandes de `sql`.
- l'instruction `run` n'est pas nécessaire.

Options

- *inobs=n* restreint le nombre d'observations traitées (par exemple dans une clause `where`) sur une table fournie en entrée de la procédure.
- *outobs=n* restreint le nombre d'observations traitées (par exemple insérées) dans une table retournée par la procédure.
- *feedback* rappelle la définition des vues parentes lors de la description d'une vue (commande `describe`).
- *noprint* pas d'édition

1.4 Procédure `sql` *vs.* étape `data`

On veut connaître la taille moyenne à 10 ans et par sexe des enfants dont la mère consommait entre 10 et 20 cigarettes par jour au moment de leur naissance. Écrire le programme permettant de calculer ces quantités à l'aide des procédures `summary`, `sort` et `print` ; puis comparer avec le programme suivant utilisant la procédure `sql`.

```
proc sql;
  select sexenf , mean(tenf_10) as tmoy
  from sasuser.statlab2
  where consm_n='10a20cig'
  group by sexenf
  order by tmoy;
```

2 L'instruction SELECT

2.1 Sélection

Exécuter et commenter le code suivant :

```
proc sql;
   title "liste des prix et des produits";
   select prodlist, nomprod from produit;
```

On peut sélectionner toutes les variables d'une table en utilisant :

```
proc sql; select * from produit;
```

Dans l'exemple suivant, on somme les années d'ancienneté des vendeurs, ville par ville. Un tri croissant est ensuite effectué. Notez la définition de l'alias `totannee` par l'instruction `as`. Notez également que cet alias est utilisé par l'instruction `order by` dans la même procédure. L'instruction `order by totannee desc` trierait par ordre décroissant.

```
proc sql;
  select villeemp, sum(anneemp) as totannee
    from employe
    where titreemp="respven"
    group by villeemp
    order by totannee;
```

Il est possible d'utiliser des expressions arithmétiques dans une clause `select`, pour réaliser des calculs sur des colonnes numériques. Par exemple tapez et commentez :

```
proc sql;
  select nomprod, prodlist, coutprod,
          prodlist-coutprod
  from produit;
```

On peut utiliser une clause `where` dans l'instruction `select`, pour garder ou retirer des observations. Exemples de syntaxes :

```
where a>b
where (a>b) and (c>d)
where (a-b)>100
where a between 5 and 15
where city not in ("paris","toulouse")
where a is null /*vrai si a est manquante*/
where a like "c%"
where a like "____e"
```

2.2 Extraction

Exercice 1 : Utilisez des clauses `where` pour afficher successivement les informations sur :

1. les employés ayant au moins 10 ans de service,
2. les employés qui ne vivent pas à 'Mimisan', et qui ont plus de 10 ans de service,
3. les employés dont le nom comence par la lettre s,
4. les employés ayant un nom de cinq lettres qui finit par un e,
5. les employés ayant travaillé 1,5 ou 10 ans,
6. les employés dont le numéro est compris entre 301 et 401,
7. les employés pour lesquels la variable `patremp` présente une valeur manquante, i.e. les employés n'ayant pas de responsable au dessus d'eux dans la hiérarchie de l'entreprise.

Les instructions `having` et `count(*)` permettent de faire comme si on appliquait une clause `where` à un groupe d'observations. L'exemple suivant permet de mieux comprendre leur utilisation. Ce code opère l'impression des seuls clients ayant deux magasins.

```
proc sql;
  title "Clients ayant deux magasins";
  select nomclient, numclient, villeclient
    from client
    group by nomclient
    having count(*)=2
    order by nomclient,2,3;
```

La commande `group by` définit le groupe qui sera évalué par `having`. Seuls les groupes dont le `having` est évalué comme vrai seront traités.

2.3 Jonction (*jointure*) de deux tables

Une requête peut porter sur plusieurs tables. Le code suivant affiche les prix de production, de catalogue et de facturation de chaque produit vendu par le client de Mimisan. La réunion de ces informations nécessite l'appel des tables `produit` et `facture`. La clause `where product.prodname=invoice.prodname` sert à réunir ces tables selon la variable commune `prodname`.

```
proc sql;
title "Information sur les produits
       vendus par Surf40";
select numfact, produit.nomprod,coutprod,
    prixfact,prodlist,nomclient,numclient
  from produit, facture
  where produit.nomprod=facture.nomprod
          and nomclient="Surf40";
```

Notez qu'il faut faire précéder, dans l'instruction `select`, le nom d'une variable apparaissant dans plus d'une des tables jointes par un nom de table. L'union des deux tables est ici faite par `from`.

Exercice 2 : Adapter le code précédent joignant les tables `client` et `facture` pour visualiser les produits vendus à Mimisan.

2.4 Jonction d'une table à elle-même

Il peut être aussi intéressant de joindre une table à elle-même pour rendre plus lisible certaines informations. Par exemple, on souhaite afficher, à partir de la table `employe`, le nom de son responsable à côté du nom de chacun des

employés. Dans ce cas, il faut d'abord dupliquer la table, ce qui peut se faire directement dans l'instruction `from` de la manière suivante : `from employe emp1, employe emp2`. Dans ce cas, `emp1` et `emp2` sont des alias (ou noms temporaires) de tables.

L'instruction `where` précise ensuite la clause :

```
where emp1.patremps=emp2.numemp.
```

Exercice 3 : Ecrivez le code permettant d'afficher le nom et la fonction de son responsable à coté du nom et de la fonction de chacun des employés.

2.5 Jonction de plusieurs tables

Il est bien sur possible de joindre plus de deux tables.

Exercice 4 : Ecrire une requête permettant d'afficher les produits vendus par Samuel à des magasins de La Torche (vous pourrez créer des alias de tables pour alléger votre code).

Une requête peut contenir une sous-requête, qui restitue une ou plusieurs valeurs ensuite utilisées par la requête qui la contient. Par exemple :

```
proc sql;
select numemp,nomclient,numclient,nomprod,numfact
  from facture
  where numemp in (216,314)
  order by 1,2,3,4;
```

renvoie des informations sur les ventes réalisées par Alain (identifiant 216) et Georges (314).

Supposons que l'on veuille afficher ces informations mais que l'on ne connaisse pas les identifiants des vendeurs. Dans le code suivant :

```
proc sql;
select numemp,nomclient,numclient,nomprod,numfact
  from facture
  where numemp in
    (select numemp from employe
                  where nomemp in ("Alain","Georges"))
  order by 1,2,3,4;
```

une sous-requête est évaluée en premier et sélectionne les identifiants d'Alain et Georges dans la table `employe`. Ces numéros sont ensuite utilisés dans la clause `where` de la requête principale pour sélectionner les lignes de facture.

Exercice 5 : Ecrivez un code utilisant une sous-requête et la clause `not in` dans la clause `where` pour afficher les produits proposés par le fournisseur mais non-vendus dans les magasins.

L'instruction `validate` permet de vérifier si la syntaxe d'une instruction `select` est correcte, sans avoir à l'exécuter. Par exemple :

```
proc sql;
  validate
  select numemp, nomemp
    from employe
    where numemp > 200 and numemp <400;
```

Crée un commentaire dans le `log`.

3 Tables et vues

La commande `create table nomtable as` permet la création ou la modification de tables ou de vues.

3.1 Création de tables

Exercice 6 : Complétez le code suivant (en remplaçant les ...) pour créer une table contenant les noms, fonctions et anciennetés des employés ayant travaillé au moins six ans pour le fournisseur.

```
proc sql;
title "Employes anciens";
create table ... as
select ...
from ...
```

```
where ...
order by ... desc;
/* affichage de la table créée */
select * from ...
```

La commande `order by ... desc` va permettre ici de trier les employés par ordre décroissant d'ancienneté.

Exercice 7 : Créez une table `nouvprix` (pour nouveaux prix), copie de la table produit.

3.2 Modification de tables

Diverses instructions permettent de modifier les données et la structure d'une table ; `alter` permet d'ajouter, supprimer des colonnes, modifier leurs attributs ; `update` permet d'ajouter/modifier des valeurs à une colonne.

Executez et commentez :

```
proc sql;
   alter table nouvprix
   add prix2011 num format=euro.;
   select * from nouvprix;
```

A noter la création de la variable `prix2011` : la commande utilisée permet de préciser que cette dernière est de type *numérique* et exprimée en euros. Les prix 2011 sont obtenus en ajoutant 20% au prix catalogue :

```
proc sql;
   update nouvprix
   set prix2011=prodlist*1.2;
   select * from nouvprix;
```

La création de `nouvprix`, l'ajout et le remplissage de prix2011 auraient pu être faits en une seule étape. On peut aussi ne modifier que certaines valeurs d'une colonne. Par exemple,

Exercice 8 : Adapter le code de l'exercice précédent pour écrire une procédure SQL qui applique en 2008 une augmentation de 20% au prix catalogue des produits coûtant moins de 240 euros, et qui laisse inchangé le prix des produits coûtant plus de 240 euros.

Le programme obtenu peut être écrit de manière plus courte, avec l'option `case ... end`

```
proc sql;
   update table
   set prix2011=prodlist *
   case when prodlist<= 240 then 1.2
   else 1
   end
```

L'instruction `insert` permet d'insérer des lignes dans une table. Exécutez l'exemple suivant :

```
proc sql;
   insert into nouvprix
     values ("cremesol",9,10,12,11,10);
   select * from nouvprix;
```

Notez que l'ordre des valeurs dans `values` doit correspondre à l'ordre des colonnes dans la table modifiée. On ne peut spécifier des valeurs que pour certaines colonnes de `nouvprix`. Dans ce cas, ces colonnes sont désignées par l'instruction `select`. La valeur d'une colonne non remplie est alors considérée comme manquante. Enfin, on peut remplacer `values` par la syntaxe suivante, qui ne nécessite pas de connaître l'ordre des colonnes :

```
set nomprod="cremesol", cost=11,
  prodlist=12, prix2011=13;
```

On peut supprimer des lignes par :

```
proc sql;
  delete from nouvprix
   where nomprod="cremesol";
        select * from nouvprix;
```

Notons qu'il est aussi possible de supprimer une table :

```
proc sql;
  drop table nomtable;
```

3.3 Utilisation des vues

L'exemple suivant illustre la création d'une vue SQL, c'est-à-dire une requête sauvegardée, à laquelle on affecte un nom par `create view nom as`. Une vue peut ensuite être appelée, comme un table, par une étape DATA, une procédure ou une instruction `select`. Cependant, contrairement à une table, une vue ne contient pas de données, et ne peut donc être mise à jour en utilisant `update`, `alte`, `insert`, `delete`. Exécutez le programme suivant qui liste les factures dont le montant dépasse 1500 euros, et affiche le nom de l'employé ayant fait la vente :

```
proc sql;
create view ventimp as
select numfact, nomclient, numclient,
       nomemp, nomprod, quantfact, prixfact
from facture as i, employe as e
where i.numemp=e.numemp and
      (quantfact*prixfact)>1500;
select * from ventimp order by numfact;
describe view ventimp;
proc univariate;
run;
```

Exercice 9 : Modifier le script précédent pour créer une vue `gdeqte` qui selectionne dans la table `facture` les factures portant sur plus de 25 articles. Executez ensuite (et commentez) :

```
proc sql;
describe view gdeqte;

proc univariate data=ventimp;
var quantfact prixfact;
run;

proc sql;
drop view gdeqte;
```

Annexe 1 : syntaxe des commandes

La commande `create`

Elle permet de créer des tables, des vues, ou des index, à partir d'autres tables ou d'autres vues.

Création d'une table

Syntaxe

create table nom-table **as** query-expression ;

create table nom-table **like** nom-table ;

create table nom-table (def-col <, def-col>) ;

La première syntaxe est utilisée pour stocker les résultats d'une interrogation. C'est une façon de créer des tables temporaires. La deuxième syntaxe est utilisée pour créer une table ayant les mêmes noms de variables et mêmes attributs qu'une autre table. La troisième syntaxe est utilisée quand on veut créer une table dont les colonnes ne sont pas présentes dans des tables déjà existantes. Les syntaxes 2 et 3 créent des tables vides, qu'il faut ensuite remplir avec la commande `insert`. Création d'une table SAS permanente dans la librairie `sql`

```
libname sql 'sql';
proc sql;
create table sql.statlab like  sasuser.statlab2;
create table sql.statlab2 as
    select sexenf, gsenf, tenf_n,
     penf_n, tenf_10, penf_10
    from sasuser.statlab2
    where (consm_n='nonfum');
```

Création d'une vue

Syntaxe

create view nom-vue **as** query-exp <**order by**item <, item>> ;

Une vue étant une interrogation stockée et ne contenant pas de données, on ne peut utiliser les instructions suivantes quand on se réfère à une vue : `insert`, `delete`, `alter`, `update`.

Création d'une vue à partir d'une table

```
create view labv2 as
       select sexenf, gsenf, tenf_n, penf_n,
        tenf_10, penf_10
       from sql.statlab2
       where (sexenf='fille');
proc print data=labv2; run;
```

Création d'un index

Un index stocke à la fois les valeurs des colonnes d'une table, et un système de directions qui permet d'accéder aux lignes de cette table à partir des valeurs de l'index. L'utilisation de l'index lors d'interrogations ou autres instructions de la procédure est déterminée par le système. L'index est automatiquement mis à jour quand on modifie la table à laquelle il est associé. Il permet d'améliorer la performance de certaines commandes, par exemple la comparaison d'une colonne indexée à une valeur constante à l'aide de l'expression `where`.

Syntaxe

create <unique > **index** nom-index **on** nom-table ;

Le mot-clé `unique` garantit que chaque valeur de la colonne indexée est unique. Ceci peut être utile quand on manipule des variables telles que le numéro de sécurité sociale. Création de l'index simple `gse` associé au groupe sanguin

```
proc sql;
create index gse on sql.statlab2 (gsenf);
```

Création de l'index composite `consm` associé à deux variables

```
proc sql;
create index consm on sql.statlab2 (consm_n,consm_10);
```

La commande `alter`

Elle permet d'ajouter ou de supprimer des colonnes dans une table SAS, ou d'en modifier les attributs (longueur, label, format).

Syntaxe

alter table nom-table
< **add** def-col <, def-col >>
< **modify** def-col <, def-col >>
< **drop** nom-col <, nom-col >> ;

Modification d'une table existante

```
alter table sql.statlab2
      add gender char(6);
```

La commande `delete`

Elle permet de supprimer des lignes dans une table.

Syntaxe

delete from nom-table < **where** sql-exp > ;

Suppression des lignes d'une table

```
delete from sql.statlab2 where gsenf='A+';
```

La commande `describe`

Elle donne la définition d'une vue, et des vues parentes si l'option `feedback` est spécifiée.

Syntaxe

describe view nom-vue ;

Description d'une vue

```
describe view labv2;
```

La commande `drop`

Elle permet de détruire indifféremment une table ou une vue.

Syntaxe

drop table nom-table < , nom-table > ;

drop view nom-vue < , nom-vue > ;

La commande `insert`

Elle permet d'ajouter des lignes à une table.

Syntaxe

insert into nom-table < (nom-col < , nom-col > >)

values (value < , value >) ;

Il existe deux autres manières d'utiliser la commande `insert` (voir l'aide en ligne).

Insertion de lignes dans une table

```
insert into sql.statlab2
       values ('fille','AB',0,0,0,0,'d')
       values ('garcon','AB',10,10,10,10,'e');
```

La commande `select`

Elle permet de sélectionner des colonnes dans une table, et d'afficher les résultats dans la fenêtre `output`.

Syntaxe

select liste d'objets **from** liste < **where** sql-exp > ;

La commande `update`

Elle permet de modifier les valeurs de certaines observations pour des colonnes d'une table existante.

Syntaxe

update nom-table **set** nom-col=sql-exp < **where** sql-exp > ;

Modification d'une table

```
update sql.statlab2
       set gender=sexenf;
```

La commande `validate`

Elle permet d'évaluer la syntaxe d'une interrogation sans l'exécuter, et retourne un message dans la fenêtre `log`.

Syntaxe

validate query-exp ;

Cette commande est essentiellement utile dans des applications utilisant des macro-variables. `validate` retourne alors une valeur indiquant si l'interrogation est valide grâce à la macro-variable `SQLRC` (SQL Return Code).

Annexe 2 : Solutions

```
/* Exercice 1*/
proc sql;
select * from employe
where anneemp>9 order by nomemp;
proc sql;
select * from employe
where (villeemp not in ("Mimisan"))
and (anneemp>9);
proc sql;
select * from employe
where nomemp like "S%";
```

```
proc sql;
select * from employe
where nomemp like "____e";
proc sql;
select * from employe
where anneemp in (1,5,10);
proc sql;
select * from employe
where numemp between 301 and 401;
proc sql;
select * from employe
where patremp is null;
/*----------------------------------------*/
/* Exercice 2*/
proc sql;
select facture.nomclient, facture.numclient,
   nomprod, prixfact
from client, facture
where facture.nomclient=client.nomclient
   and facture.numclient=client.numclient
   and villeclient="Mimisan";
/*----------------------------------------*/
/* Exercice 3*/
proc sql;
title "Renseignements employés";
select emp1.nomemp, emp1.titreemp,
   emp2.nomemp, emp2.titreemp
from employe emp1,employe emp2
where emp1.patremp=emp2.numemp;
/*----------------------------------------*/
/* Exercice 4*/
proc sql;
title "Performances Samuel";
select facture.*, client.*
from employe, facture, client
  /* mix des tables par numclient */
where facture.numclient=client.numclient
  /* choix de la ville */
  and client.villeclient="LaTorche"
  /* employe à la Torche */
  and employe.villeemp=client.villeclient
  /* nommés Samuel */
  and employe.nomemp="Samuel";
/*----------------------------------------*/
/* Exercice 5*/
proc sql;
title "invendus";
select produit.nomprod, prodlist
from produit
where produit.nomprod not in (select nomprod
   from facture);
/*----------------------------------------*/
/* Exercice 6*/
proc sql;
title "Employes anciens";
create table anciens as
select nomemp, titreemp, anneemp
from employe
where anneemp>5
order by anneemp desc;
select * from anciens;
/*----------------------------------------*/
/* Exercice 7*/
proc sql;
title "Nouveaux prix";
create table nouvprix as
select *
from produit;
proc print data=nouvprix;
run;
/*----------------------------------------*/
/* Exercice 8*/
```

```
/* Création de la colonne prix2007 */
proc sql;
alter table nouvprix
add prix2007 num format=euro.;
select * from nouvprix;
proc sql;
update nouvprix
set prix2007 = prodlist*1.2;
select * from nouvprix;
/* Début de l'exercice */
proc sql;
alter table nouvprix
add prix2008 num format=euro.;
select * from nouvprix;
proc sql;
update nouvprix
set prix2008=prodlist where
prix2007>240;
select * from nouvprix;
proc sql;
update nouvprix
set prix2008=prodlist*1.2 where
prix2007<241;
select * from nouvprix;
/*---------------------------------------*/
/* Exercice 9*/
proc sql;
create view ventimp as
select numfact,nomclient,numclient,nomemp,
   nomprod, quantfact, prixfact
from facture as i, employe as e
where i.numemp-e.numemp and
   (quantfact*prixfact)>1500;
select * from ventimp order by numfact;
describe view ventimp;
proc sql;
```

```
create view gdeqte as
select numfact, nomclient, numclient,
nomemp, nomprod, quantfact, prixfact
from facture as i, employe as e
where i.numemp=e.numemp and
quantfact>25;
select * from ventimp order by
numfact;
describe view ventimp;
proc sql;
describe view gdeqte;
proc univariate data=ventimp;
var quantfact prixfact;
run;
proc sql;
drop view gdeqte;
```