OPTIMISEZ LES PERFORMANCES DE TRAITEMENT DE VOS DONNEES AVEC SAS/ACCESS®

Caractéristiques :
Catégories : SAS/Access
OS : Windows, Unix
Version : SAS® 9.4
Vérifié en décembre 2013

Le BIG DATA nous immerge inexorablement dans un océan d'informations. Accéder à toujours plus données le plus rapidement possible n'a jamais été autant au cœur des attentes des utilisateurs. Aussi, dans ces environnements où la vitesse revêt une importance capitale, il est impensable de ne pas comprendre l'interaction entre SAS et les bases de données. Savoir quand il est plus avantageux de laisser SAS faire le traitement ou quand il est préférable que la base de données s'en charge est fondamental dans l'atteinte de meilleures performances.

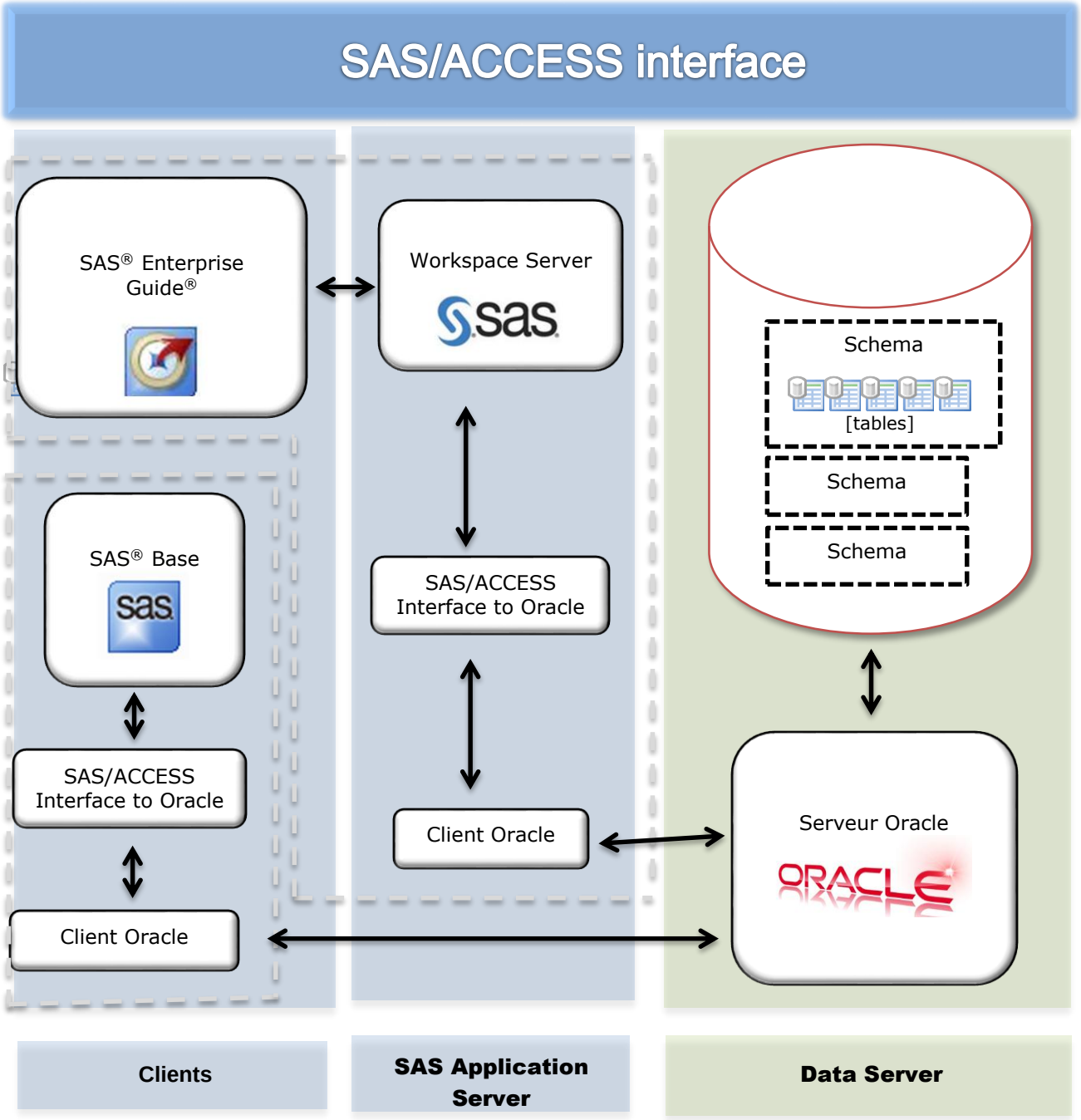
Les performances, toujours les performances... Le nerf de la guerre pour beaucoup d'entre nous. **Mais comme pour un virus, il est toujours plus facile de le traiter quand on sait à qui on a affaire.** Cet article explore les pistes pour traiter efficacement vos données externes et obtenir le meilleur gain de performance. Il s'adresse aux développeurs mais également aux architectes en charge de la mise en œuvre du système d'information.

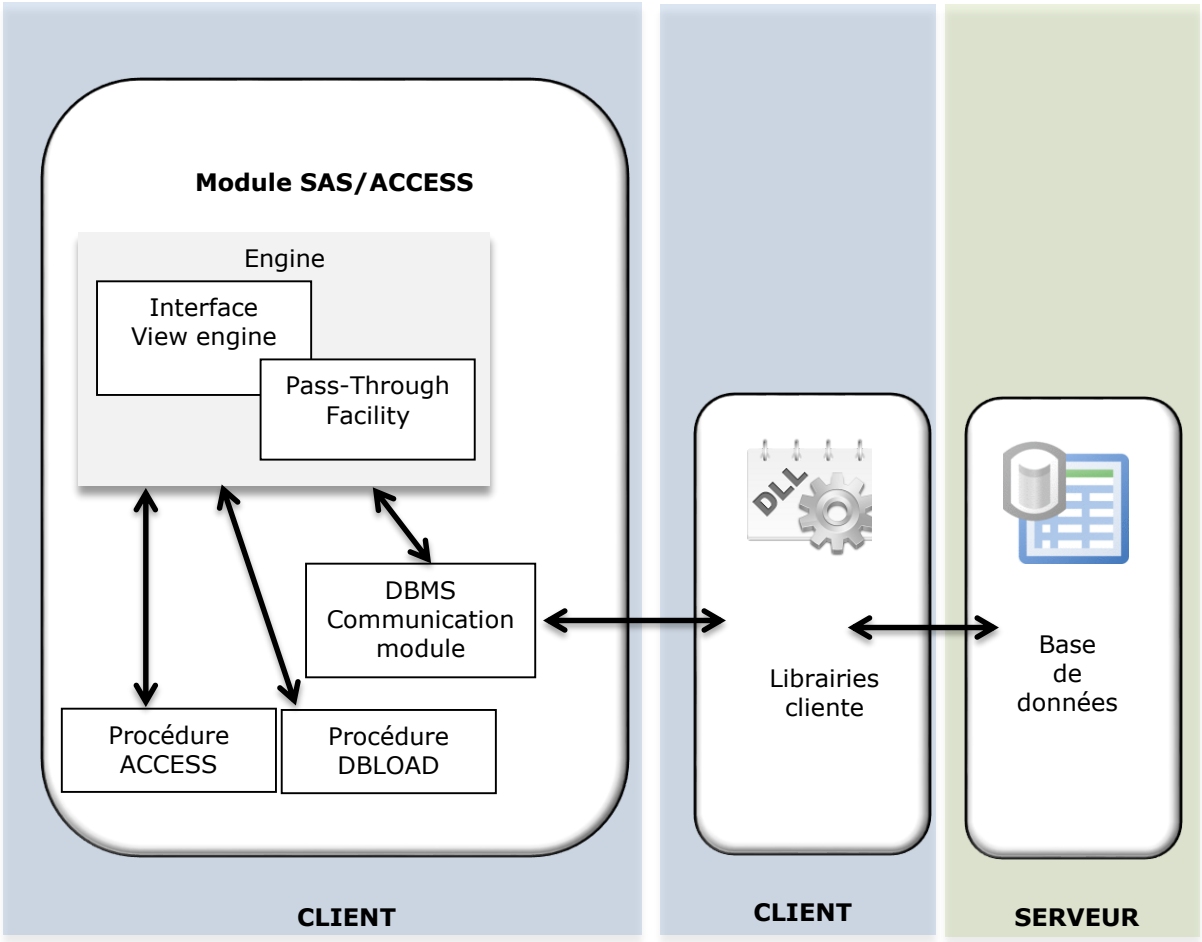
Sommaire

1. INTRODUCTION

Avant de s'attaquer au sujet central de cet article, un rapide rappel du fonctionnement du module SAS/Access peut s'avérer nécessaire. Les modules SAS/ Access sont des solutions « out-of-the-box » fournissant une connectivité entre SAS et des sources de données tiers, via le client du SGBD, y compris les data warehouse appliances, Hadoop Distributed File System et les bases de données relationnelles (Oracle, Sybase, Mysql …).

Le schéma ci-dessous montre les interactions entre les clients SAS, le module SAS/ACCESS et une base de données Oracle. Ce schéma permet d'avoir une vision globale des échanges.





- La procédure DBLOAD n'est pas disponible pour tous les modules SAS/ACCESS,
- Client et serveur peuvent être localisés sur la même machine physique,
- Les librairies clientes, nécessaires au fonctionnement du module SAS/ACCESS, sont fournies par le fournisseur de la base de données.

2. SE CONNECTER A UNE BASE DE DONNEES

2.1. Jeu de données utilisé

Pour illustrer cet article nous allons travailler sur une base de données Oracle 10G hébergée sur un serveur AIX. Nous utiliserons une table volumineuse que nous avons créée dans le schéma « scott »

Table « visiteurs ». Cette table compte 270 000 lignes.

Obs.	ID	ADRESSE_IP	NAVIGATEUR	RESOLUTION	OS	LANGUE
1	132803813811	2.11.16.89	Firefox	1366x768	Windows	fr
2	132804784754	79.94.20.11	Firefox	1366x768	Windows	fr
3	132808343969	88.178.81.31	ie	1366x768	Windows	undefined
4	132809215996	86.194.143.224	ie	1600x900	Windows	undefined
5	132810495478	109.208.129.131	Chrome	1440x900	Windows	fr
6	132812007729	213.44.56.77	ie	1192x670	Windows	undefined
7	132812856903	46.193.163.224	ie	1280x720	Windows	undefined
8	132818104765	77.195.108.21	Firefox	1366x768	Windows	fr
9	132820909291	94.217.37.52	Chrome	1152x864	Windows	de
10	132820926987	109.12.142.184	Chrome	1366x768	Windows	fr

2.2. SQL PASS-THROUGH implicite et étape DATA

Vous pouvez utiliser SAS/ACCESS pour lire, mettre à jour, insérer et supprimer des données d'un objet SGBD comme s'il s'agissait d'un ensemble de données SAS. Voici comment faire :

1. Activation d'une interface SAS/ACCESS en spécifiant un nom de moteur SGBD et les options de connexion appropriées dans une déclaration de LIBNAME,
2. Vous interagissez avec les données comme vous le faites avec n'importe quelle bibliothèque SAS classique,
3. SAS/ACCESS génère, si possible, des instructions SQL qui sont l'équivalent des procédures SAS que vous avez saisi,
4. SAS/ACCESS soumet le SQL généré pour le SGBD.

Si vous utilisez **SAS en version 64 bits,** il est nécessaire que votre client d'accès à Base de Données (librairies) soit **en version 64 bits.**

Il existe 2 méthodes pour accéder aux tables contenues dans une source de données tiers. L'une d'elle consiste à créer une bibliothèque en utilisant l'instruction **LIBNAME**.
Cette syntaxe est familière aux utilisateurs de SAS.

Dans l'exemple ci-dessous, nous créons une bibliothèque pour nous connecter à oracle :

```
LIBNAME oraloc1 ORACLE PATH=ORA10LOCHES SCHEMA=SCOTT USER=scott PASSWORD=tiger;

NOTE: Libref ORALOC1 was successfully assigned as follows:
   Engine:    ORACLE
   Physical Name: ORA10LOCHES
```

Avec une déclaration **LIBNAME**, vous pouvez utiliser votre boîte à outils SAS et manipuler vos données via l'étape DATA et les procédures habituelles comme si vous travailliez avec des données SAS ordinaires.

En effet, le module SAS/ACCESS traduit les instructions SAS en commandes SQL directement interprétables par la base de données. Si le code SAS ne peut pas être traduit en SQL, le contenu de la table est rapatrié dans SAS et le code exécuté directement par le moteur SAS.

La connexion se fait à l'exécution du **LIBNAME**. Elle reste valide tant que la session est en cours et que le **LIBNAME** n'est pas effacé. Pour en savoir plus sur le fonctionnement des connexions et des sessions avec SAS/ACCESS vous pouvez lire l'article « Comprendre les différents types de connexion lors de la définition d'une bibliothèque d'accès à une base de données »

2.3. SQL PASS-THROUGH explicite

Le SQL Pass-Through explicite est une autre façon d'établir une connexion à la base de données externe.

```
proc sql;
connect to oracle(user=scott password=tiger path=ORA10LOCHES);

create table visiteurs as select * from connection to oracle(
select * from visiteurs);
quit;
```

Dans ce cas, le code SQL est envoyé tel quel à la base de données, ce qui nécessite l'écriture d'une requête SQL compréhensible par le SGBD.

2.4. Choisir entre implicite et explicite.

Il n'y a pas de formule magique pour affirmer qu'une méthode est meilleure que l'autre.
Cela dépend des circonstances, des connaissances et de l'expérience du développeur, notamment dans le langage SQL.

Vous serez obligé de mixer le langage SAS Base et SQL.

Le langage SAS Base est utilisé pour vos besoins de reporting ou encore pour créer du code complexe (macro, array, if, boucle…).

Autant de choses que les bases de données ne savent pas faire. N'oubliez pas que l'étape DATA offre bien plus de possibilités que le SQL.

Cependant quelques points sont à prendre en compte afin de choisir la méthode en fonction :

Utilisez le SQL Pass-through explicite lorsque :

- Vous avez besoin d'utiliser une requête SQL spécifique à la base de données utilisée,
- Vous voulez garder un contrôle sur les requêtes SQL envoyées à la base de données,
- Vous voulez vous assurer que le code soit bien exécuté par le DBMS, et non rapatrié côté SAS par souci de performance

Utilisez le SQL Pass-through implicite et l'étape DATA lorsque :

- Vos besoins sont complexes pour gérer la base de données (utilisation du macro-langage, des array, de boucles, de fonctions sans équivalence par le SGBD) ou non gérables par SQL (boucles, conditions, proc tabulate, graphiques, ODS …)
- Vous êtes plus à l'aise avec le langage SAS Base
- Vos programmes doivent être capables de se connecter à différentes base de données (Oracle,Mysql,DB2…) avec seulement des modifications mineures.
- Les données du SGBD ont une taille permettant le travail dans ce mode

Enfin, voici un tableau récapitulatif de la syntaxe à utiliser (ces 3 exemples sont équivalents) :

SQL Pass-through explicite	SQL Pass-through implicite	Data step

`proc sql;`
`connect to oracle(user=scott`
`password=tiger`
`path=ORA10LOCHES);`
 	
`create table visiteurs as`
`select * from connection to`
`oracle(`
`select * from visiteurs);`
`quit;`
	`proc sql;`
	`create table visiteurs as`
	`select * from`
	`oraloc1.visiteurs;`
		`data visiteurs;`
		`set oraloc1.visiteurs;`
		`run;`

Pour plus d'informations, je vous invite à visiter la Usage NOTE 41616 qui donne des exemples de code pour vous connecter à une base Oracle en utilisant SAS/ACCESS® interface to Oracle.

3. OBTENIR LES MEILLEURS PERFORMANCE POSSIBLE

Dans le processus d'optimisation, de nombreux aspects sont à considérer.

Nous allons nous attarder sur les principaux accélérateurs de performance suivants :

- Réduire le volume de données transférées entre SAS et la base de données
- Transmettre rapidement des données nécessaires
- Optimiser le traitement dans la base de données.

3.1. Limiter le nombre de lignes et de colonnes retournées

Limiter le nombre de lignes retournées par le SGBD est un facteur de performance extrêmement important, car il faut éviter que trop de données transitent sur le réseau. Autant que possible, précisez des critères de sélection limitant le nombre de lignes que la base de données retourne à SAS. Utilisez la clause WHERE pour récupérer un sous-ensemble des données.
Si vous êtes uniquement intéressé par les premières lignes d'une table, pensez à ajouter l'option OBS=. Cette option permet de limiter le nombre de lignes transitant à travers le réseau, ce qui améliore considérablement les performances lors de la lecture de grandes tables. Elle est également utile dans la phase d'écriture du programme, pour tester la validité de son code sans chercher à récupérer toutes les observations.

Toujours pour limiter la taille des données, vous pouvez également agir sur le nombre de colonnes. Pour cela, vous pouvez utiliser l'instruction SELECT de la procédure SQL, ou par les instructions DROP/KEEP de l'étape DATA

3.2. Faire réaliser le traitement par la base de données

Pour une efficacité optimale, cela parait naturel de dire que toutes les opérations doivent être réalisées à l'intérieur de la base de données. Mais dans la réalité, la mise en œuvre n'est pas toujours aisée et ne s'avère pas toujours simple.

En effet, la solution de facilité consiste à se dire qu'en utilisant le **SQL pass-through explicite**, nous avons un contrôle total sur les requêtes SQL envoyées à la base de données. Néanmoins cela implique que **nous ne changions pas de base de données** dans le cycle de vie de votre projet et surtout cela implique également d'avoir des besoins simples. Comme indiqué dans le précédent chapitre, il est nécessaire de jongler avec les deux méthodes en fonction du besoin et du traitement que l'on souhaite mettre en œuvre.

La syntaxe SQL, bien que standard, peut varier d'une base de données à une autre. **Certaines spécificités dans la syntaxe d'une requête SQL peuvent être interprétées correctement par un moteur de base de données mais pas par un autre.** Par exemple, la soustraction de deux SELECT s'écrit « SELECT … MINUS SELECT » sous Oracle et « SELECT … WHERE NOT EXISTS … » sous SQL Server.

Dans le cadre d'une migration d'un SGBD vers un autre, cela nécessiterait une passe complète sur le code SAS et, dans le pire des scenarios, une réécriture de certaines requêtes SQL.

3.3. Tri et indexation des données

Trier des données est gourmand en ressources, que ce soit en utilisant la procédure SORT ou une clause ORDER BY. Triez les données uniquement lorsque cela est nécessaire pour votre programme.

Si vous utilisez une instruction BY, il est recommandé d'associer votre BY avec une colonne indexée. En effet, lors de l'utilisation d'un BY, le moteur SAS/ACCESS va transformer ce BY en une clause ORDER BY compréhensible par la base de données (sous condition que les éventuelles fonctions

ajoutées soit interprétables). La clause ORDER BY va trier les données avant de les retourner à SAS. Si votre table est volumineuse, ce tri peut nuire aux performances. **Utilisez une variable basée sur une colonne indexée afin de réduire cet impact négatif.**

Cependant, le choix d'indexer une table ne se fait pas à la légère. La création d'index utilise de l'espace mémoire dans la base de données, et, étant donné qu'il est mis à jour à chaque modification de la table à laquelle il est rattaché, cela peut alourdir le temps de traitement du SGBDR lors de la saisie de données. **L'entretien d'un index sur une colonne a également un impact sur les écritures de cette colonne.** Aussi, il faut que la création d'index soit justifiée et que les colonnes sur lesquelles il porte soient judicieusement choisies, notamment pour minimiser les doublons.

Pour illustrer l'impact d'un index sur les performances, nous allons utiliser une PROC SORT pour classer une table. Cette table, `visiteurs_ville`, contient 10 millions de lignes. Sa structure est la suivante :

- **Id**
- **Id_visiteur**
- **Code postal**

L'id _visiteur est à mettre en relation avec la table visiteur utilisée dans l'ensemble de cet article.

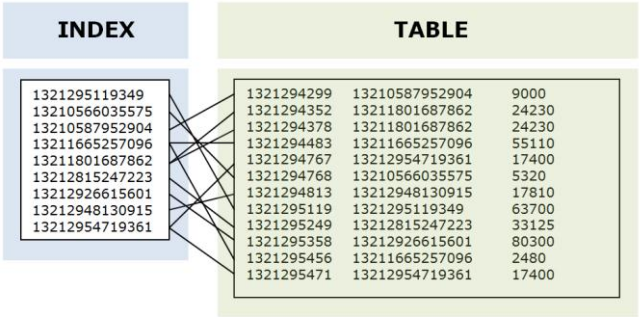
Soumettons le code suivant :

```
proc sort data=oraloc1.visiteurs_ville out=testA;
  by id_visiteur;
run;
```

```
NOTE: Sorting was performed by the data source.
NOTE: There were 10353000 observations read from the data set ORALOC1.VISITEURS_VILLE.
NOTE: The data set WORK.TESTA has 10353000 observations and 3 variables.
NOTE: PROCEDURE SORT used (Total process time):
   real time      59.42 secondes
   user cpu time    4.32 secondes
   system cpu time   0.92 secondes
   memory       2744.59k
   OS Memory      15940.00k
   Timestamp      29/08/2013 12:58:53 PM
   Step Count             10 Switch Count 1036
```

Le temps d'exécution est de 59,42 secondes.

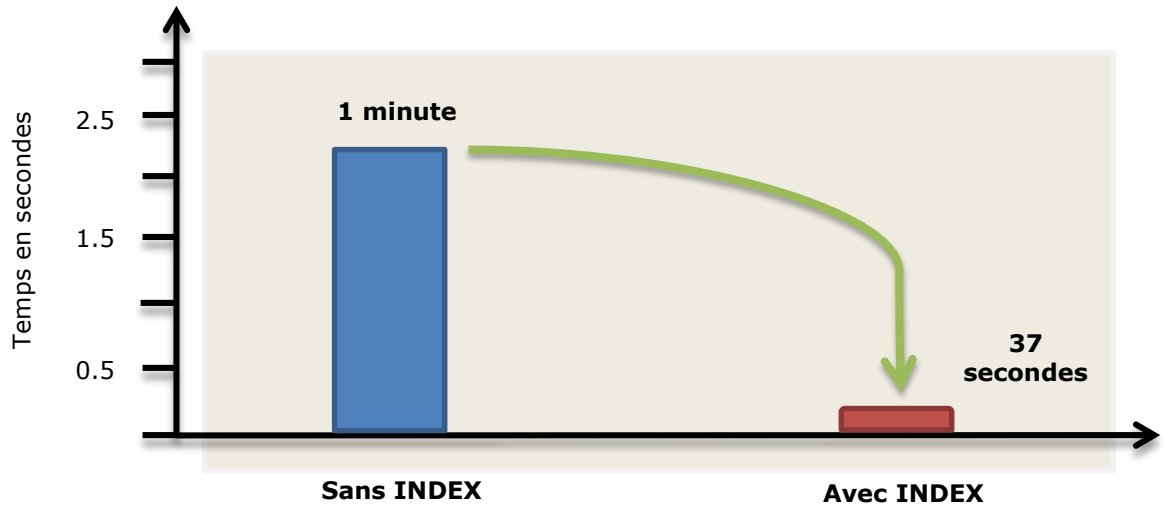
Maintenant, positionnons un index sur cette table. L'index est ajouté sur le champ id_visiteur :



Si nous resoumettons le code, le temps d'exécution passe de 59,42 secondes à 36,07 secondes :

```
NOTE: Sorting was performed by the data source.
NOTE: There were 10353000 observations read from the data set ORALOC1.VISITEURS_VILLE.
NOTE: The data set WORK.TESTB has 10353000 observations and 3 variables.
NOTE: PROCEDURE SORT used (Total process time):
```

```
real time       36.07 secondes
user cpu time    3.52 secondes
system cpu time  0.48 secondes
memory       2744.59k
OS Memory      15940.00k
Timestamp      29/08/2013 01:00:02 PM
Step Count             11 Switch Count 1036
```



3.4. Dévoiler les requêtes de SAS/ACCESS

Il faut savoir que même en présence d'un index, l'usage de ce dernier n'est pas systématique. Pour un nombre de lignes à retourner important, l'usage d'un index est plus coûteux qu'un parcours complet de la table. C'est donc très courant que les index ne soient pas utilisés dès que le moteur SQL estime que le nombre de lignes à récupérer dépasse une certaine valeur.

Lorsque vous utilisez la procédure SORT, sachez que les règles de tri de SAS et du SGBD peuvent être différentes. Utilisez l'option SAS SORTPGM pour spécifier les règles à appliquer (SORTPGM System Option: Windows)

SAS propose l'option **SASTRACE** permettant d'indiquer dans le journal SAS la requête envoyée à la base de données. Cumulée avec l'option **FULLSTIMER**, l'option **SASTRACE** se révèle d'une grande utilité.

Surtout si le temps de traitement d'un programme vous semble anormalement lent.

Selon les paramètres utilisés, l'option **SASTRACE** donne un résultat différent dans le journal SAS.

Généralement, nous utilisons la syntaxe suivante :

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

L'option **SASTRACE** peut prendre d'autres paramètres donnant ainsi accès à plus ou moins d'informations. Reportez-vous à la documentation SAS pour obtenir des détails sur ces paramètres.

Vous connaissez à présent cette option, reprenons donc les instructions SAS détaillées dans la partie 2 et observons le journal SAS maintenant que SASTRACE est positionnée :

SQL Pass-through explicite	SQL Pass-through implicite	Data step
proc sql; connect to oracle(user=scott password=tiger path=ORA10LOCHES); create table visiteurs as select * from connection to oracle(select * from visiteurs); ORACLE_1: Prepared: on connection 1 select * from visiteurs ORACLE_2: Executed: on connection 1 SELECT statement ORACLE_1	proc sql; create table visiteurs as select * from oraloc1.visiteurs; ORACLE_3: Prepared: on connection 0 SELECT * FROM VISITEURS ORACLE_4: Executed: on connection 0 SELECT statement ORACLE_3	data visiteurs; set oraloc1.visiteurs; run; ORACLE_5: Prepared: on connection 0 SELECT * FROM VISITEURS ORACLE_6: Executed: on connection 0 SELECT statement ORACLE_5

Dans les 3 exemples ci-dessus la requête SQL envoyée à la base de données est identique (`SELECT * FROM VISITEURS`)

Pour désactiver les logs SAS/ACCESS utilisez la syntaxe suivante :

```
options sastrace=off;
```

3.5. Attention aux fonctions et options SAS

Lorsque vous utilisez des fonctions SAS pour extraire des données de tables contenues dans une base de données, il arrive que les performances obtenues lors de l'extraction se trouvent très éloignées de celles espérées.

L'utilisation de l'option **SASTRACE** pour voir « l'envers du décor » s'avère alors essentielle.

Exécutons le programme suivant :

```
/* Sans condition */

proc sql;
 connect to oracle(user=scott password=tiger
 path=ORA10LOCHES);
 create table extract_navigateur as
 select * from connection to Oracle
 (select * from visiteurs);
quit;

data extract_navigateur;
set oraloc1.visiteurs;
run;

proc sql;
      create table extract_navigateur as select * from oraloc1.visiteurs;
quit;

/* Avec la condition where trim(navigateur)='ie' */

proc sql;
 connect to oracle(user=scott password=tiger
 path=ORA10LOCHES);
```

```
 create table extract_navigateur as
 select * from connection to Oracle
 (select * from visiteurs where trim(navigateur)='ie');
quit;

data extract_navigateur;
set oraloc1.visiteurs;
      where trim(navigateur)='ie';
run;

proc sql;
      create table extract_navigateur as select * from oraloc1.visiteurs
where trim(navigateur)='ie';
quit;
```

Et observons, pour chaque méthode, le temps d'exécution (en secondes) :

	Nombre de Résultats	SQL Pass-through explicite		SQL Pass-through implicite		DATA STEP	
		Total	Oracle	Total	Oracle	Total	Oracle
Sans condition	**267 148**	**5,59**	**4,47**	**4,97**	**4,10**	**5,35**	**4,29**
Avec la condition where trim(navigateur)='ie'	**87 934**	**1,75**	**1,29**	**5,08**	**4,3**	**5,70**	**4,18**

La différence est flagrante mais deux questions se posent :

- Pourquoi, le SQL Pass-through explicite est plus rapide que l'implicite, lorsque les résultats sont filtrés avec trim ? (1,75 secondes contre 5,70 secondes en implicite)
- Pourquoi, en Pass-through implicite, le temps d'exécution du traitement est similaire bien que nous n'ayons que 87 934 résultats au lieu des 267 148 que contient la table « visiteurs » ?

L'option **SASTRACE** peut nous aider à élucider ce mystère.

Après avoir positionné l'option selon la syntaxe suivante :

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

Relançons notre programme et examinons le journal SAS :

```
data extract_navigateur;
set oraloc1.visiteurs;
      where trim(navigateur)='ie';
run;

ORACLE_8: Prepared: on connection 0
SELECT "ID", "CODE_POSTAL", "ADRESSE_IP", "NAVIGATEUR", "RESOLUTION", "OS", "LANGUE",
"MOIS",
"ANNEE" FROM VISITEURS

ORACLE_9: Executed: on connection 0
SELECT statement ORACLE_8
```

La clause WHERE n'est PAS envoyée à Oracle et l'ensemble des données (270 000 lignes) est rapatrié côté SAS, laissant le traitement de la fonction TRIM à SAS.

Cet échange de données a un coût et se ressent sur le temps d'exécution de l'étape.

Effectuons maintenant le même test mais en utilisant, cette fois-ci, la fonction STRIP :

```
data extract_navigateur;
set oraloc1.visiteurs;
      where strip(navigateur)='ie';
run;

ORACLE_36: Prepared: on connection 0
SELECT "ID", "CODE_POSTAL", "ADRESSE_IP", "NAVIGATEUR", "RESOLUTION", "OS", "LANGUE", "MOIS",
"ANNEE" FROM VISITEURS WHERE ( TRIM("NAVIGATEUR") = 'ie' )

ORACLE_37: Executed: on connection 0
SELECT statement ORACLE_36

NOTE: L'étape DATA used (Total process time):
   real time      1.79 secondes
   user cpu time    0.59 secondes
   system cpu time   0.34 secondes
   memory       368.22k
   OS Memory      6768.00k
```

Remplacer TRIM par la fonction STRIP est beaucoup plus efficace. SAS remplace le STRIP par la fonction SQL TRIM. La requête avec une clause WHERE est envoyée à Oracle et cela se ressent sur le temps d'exécution. La création de la table a demandé **1,79 seconde contre 5,70 secondes** si la fonction TRIM est utilisée.

Lorsque vous utilisez des étapes DATA pour traiter les données d'une base de données et que les temps d'exécution vous semble longs, utilisez l'option **SASTRACE** pour déterminer la requête SQL envoyée par SAS.

Lorsque vous exécutez des requêtes en SQL Pass-through implicite, **toutes les fonctionnalités définies dans la norme American National Standards Institute (ANSI) SQL ne sont pas supportées.** L'effort de développement des fonctionnalités SQL par SAS a été porté sur la capacité d'interpréter de façon la plus performante possible les requêtes SQL. **En conséquence, certaines d'entre elles n'ont pas encore été mises en œuvre dans la PROC SQL.** Des fonctions SAS peuvent être utilisées pour retrouver des fonctionnalités équivalentes à celles présentes dans la norme ANSI SQL.

La documentation donne la liste des fonctions que SAS/Access envoie au SGBD. Par exemple, la liste pour SAS/Access to Oracle est donnée dans la documentation SAS/ACCESS(R) 9.4 for Relational Databases: Reference.

3.6. L'option DBCOMMIT

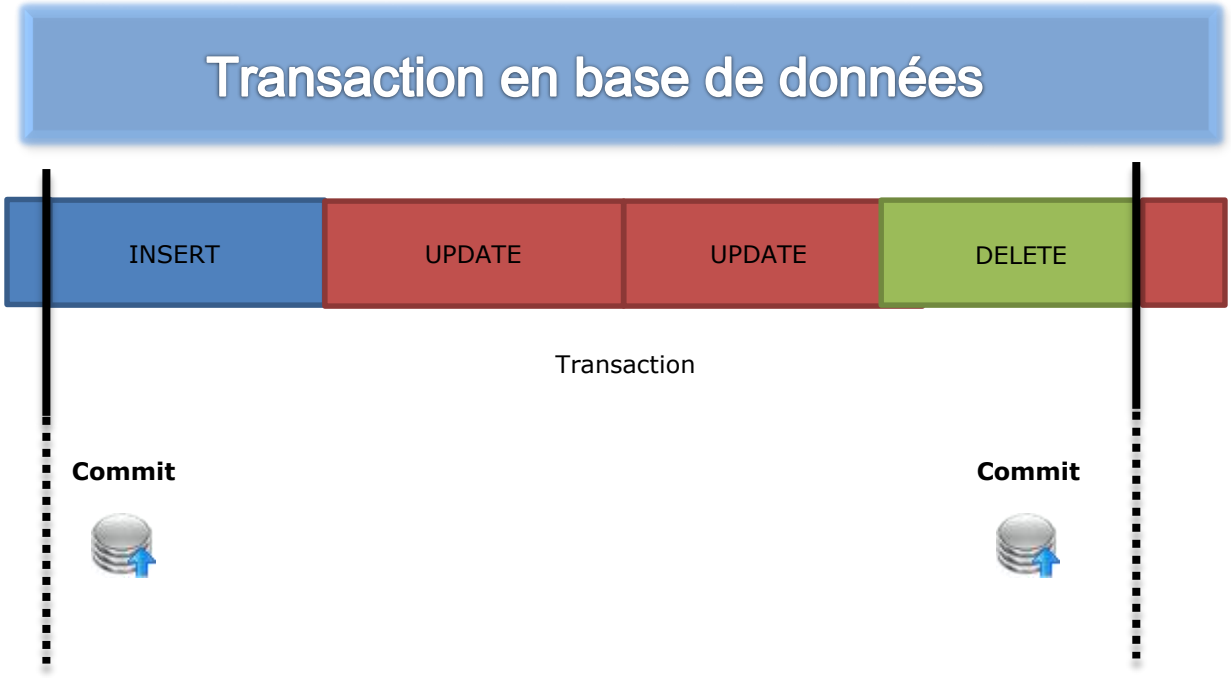
L'option **DBCOMMIT** définit le nombre de lignes insérées dans la base de données lorsqu'une transaction est validée. Pour comprendre comment cela fonctionne, regardons ce qui se passe en « coulisses » lors de l'insertion d'une ligne dans la base de données.

En effet, pour insérer une ligne dans une table de base de données il y a beaucoup d'activités qui se déroulent dans les coulisses afin de finaliser la transaction. Une transaction est un ensemble cohérent de modifications faites sur les données. Une transaction est soit entièrement annulée soit entièrement validée. L'ordre SQL COMMIT termine et valide (écrit) la transaction. L'ordre SQL ROLLBACK termine et annule la transaction. Tant qu'il n'y a pas eu COMMIT, seul l'utilisateur courant voit ses mises à jour.

Le journal des transactions de la base de données enregistre toutes les modifications apportées aux données pour assurer l'intégrité des données.

Pour information, voici les propriétés d'une transaction valide :

Atomicité	Transaction s'exécute entièrement ou pas du tout.
Consistance	Cohérence sémantique, une transaction assure l'intégrité des données.
Isolation	Pas de propagation de résultats non validés et pas d'interférence entre les données.
Durabilité	Persistance des effets validés, les effets d'une transaction validée ne sont jamais perdus.



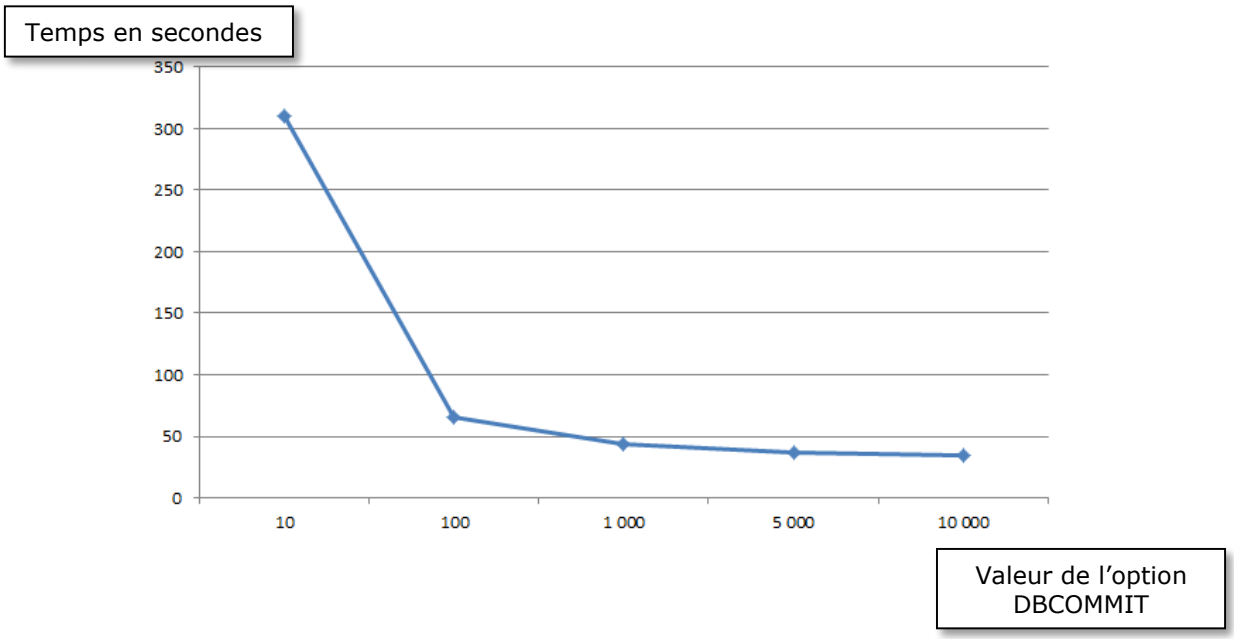
Dans SAS, par défaut, la transaction est validée tous les 1000 enregistrements. En fonction de la valeur de **DBCOMMIT,** le temps de traitement est plus ou moins long, car le moteur doit valider la transaction plus ou moins souvent.

Pour démontrer le gain de temps, utilisons le code suivant en variant la valeur de l'option **DBCOMMIT** :

```
data oraloc1. Visiteurs_test(dbcommit=100);
      set oraloc1.visiteurs;
run;
```

Par défaut, la transaction est validée tous les 1000 enregistrements. Nous avons débuté les tests à 10 afin de juger de l'impact de l'option.

Valeur de l'option DBCOMMIT	Temps d'exécution en secondes
10	**310 secondes**
100	**65 secondes**
1000 (valeur par défaut)	**44 secondes**
5000	**37 secondes**
10000	**35 secondes**

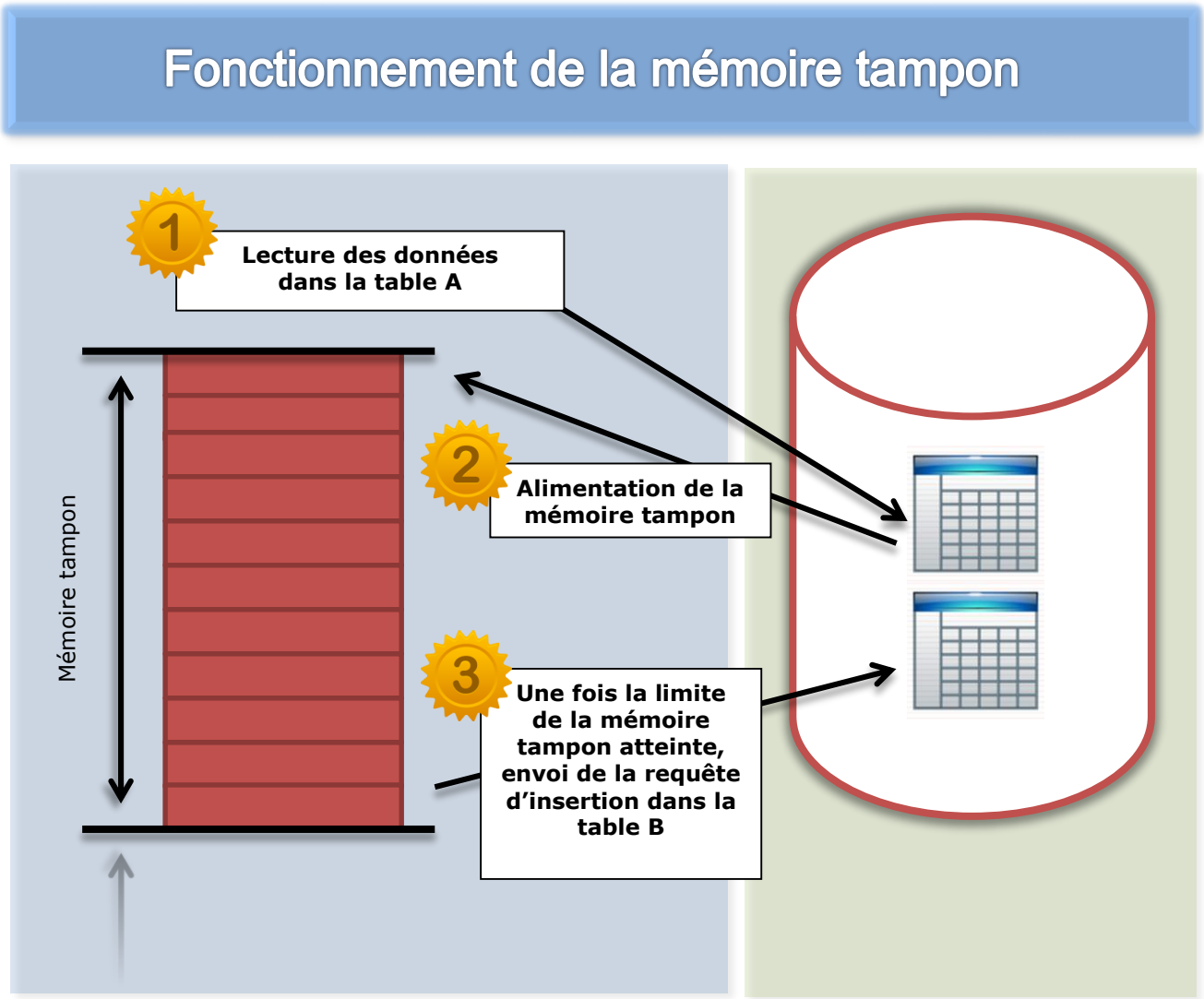


Comme vous pouvez le constater la valeur par défaut est pertinente. Dans certains cas, comme celui-ci, positionner une valeur plus grande pour le DBCOMMIT peut **améliorer les performances d'environ 16%** par rapport à la valeur par défaut.

Vous pourriez être tenté de positionner l'option DBCOMMIT à 1 million par exemple mais cela ne serait pas sans conséquence. En effet, lors d'une transaction, les champs modifiés sont verrouillés. Plus le nombre de verrous positionnés augmente, plus les ressources nécessaires à la gestion de ces verrous augmentent. C'est dans cet esprit que vous devez régler la valeur de DBCOMMIT pour qu'elle soit suffisamment grande pour limiter le processus de validation, mais pas trop grande afin d'éviter les problèmes de transaction trop longue (verrouillage, problème d'espace dans le journal de trans action).

3.7. Utilisation optimale de la mémoire tampon

La mémoire tampon est une zone de mémoire utilisée pour entreposer temporairement des données. Ainsi, lors de l'exécution d'un traitement SAS, les données envoyées vers la base de données sont stockées dans des mémoires tampon en attente de leur envoi effectif.



Dans SAS, la taille de cette mémoire tampon est définie par trois options : **INSERTBUFF**, **UPDATEBUFF** et **READBUFF**.

L'option **INSERTBUFF** spécifie le nombre de lignes à prendre en compte pour commande d'insertion dans ma base de données.

Sous Oracle par exemple, la valeur de cette option est positionnée à 1. Comme nous l'avons vu dans le chapitre 3, cela signifie que, durant une insertion en base de données, si nous importons 10 000 lignes, le module SAS/ACCESS exécute 10 000 requêtes de façon séquentielle.

Il est donc primordial d'ajuster ce paramètre afin d'optimiser l'efficacité de son traitement. La valeur optimale de cette option varie en fonction de facteurs tels que la mémoire disponible. Vous pourriez avoir besoin d'expérimenter des valeurs différentes afin de déterminer la meilleure.

Vous trouverez plus de détails sur ces options à l'adresse suivante :

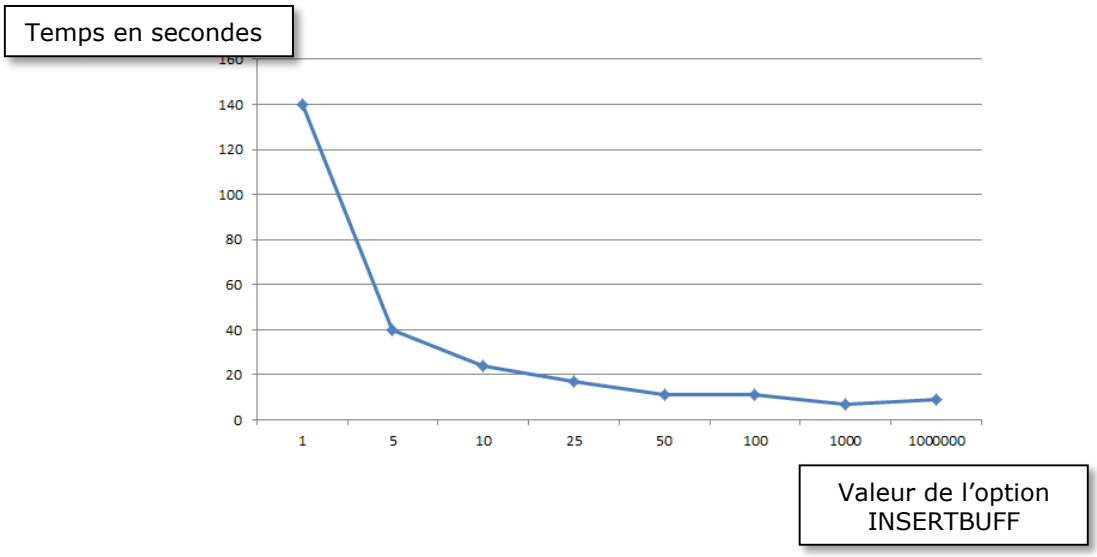
http://support.sas.com/documentation/cdl/en/acreldb/65053/HTML/default/viewer.htm#n1dsqxfjr3yjegn1lb5uqmmvbegu.htm

Pour démontrer le gain de temps, utilisons le code suivant en variant la valeur de l'option **INSERTBUFF** :

```
data oraloc1. Visiteurs_test(insertbuff=100);
      set oraloc1.visiteurs;
run;
```

INSERTBUFF est un nombre entier allant de 1 à 2 147 483 648. Différentes valeurs de **INSERTBUFF** ont été testées pour voir quel impact cela aurait sur l'étape DATA précédente :

Valeur de l'option INSERTBUFF	Temps d'exécution en secondes
1	**140 secondes**
5	**40 secondes**
10 (valeur par défaut)	**24 secondes**
25	**17 secondes**
50	**11 secondes**
100	**11 secondes**
1 000	**7 secondes**
1 000 000	**9 secondes**



Dans notre test, l'augmentation de la valeur de INSERTBUFF de 1 à 25 a eu un impact significatif sur les performances. Il est important de noter que les résultats obtenus illustrent une méthode de tests et non pas des résultats absolus, la valeur optimale d'un paramètre dépendant du code et des données.

Le gain de performance est également présent lors de la mise à jour des données de la table, via l'instruction SQL UPDATE et la configuration de l'option UPDATBUFF.

Valeur de l'option UPDATEBUFF	Temps d'exécution en secondes
1(valeur par défaut)	**42 secondes**
10 000	**12 secondes**
25 000	**6 secondes**

L'option UPDATEBUFF n'est disponible que pour les connexions aux bases de données Oracle.

3.8. L'option SASTRACE ? A utiliser en connaissance de cause !

Vous êtes maintenant convaincu de l'importance de l'option **SASTRACE**.

Faut-il alors positionner cette option, coûte que coûte, dans tous vos programmes ?

Non.

Cette option est à manier avec précaution.
Une utilisation abusive peut avoir des impacts non négligeables sur les performances, car le temps d'écriture des messages dans la log peut devenir plus long que l'exécution de la requête elle-même.

Exécutons le programme suivant :

```
options sastrace=off;

data oraloc1. Visiteurs_lot1;
set oraloc1.visiteurs(obs=10000);
run;
```

Les 10000 observations ont été insérées dans la table Oracle Visiteurs_lot1 en 1,24 seconde.

Maintenant, exécutons la même étape DATA mais en ayant au préalable positionné une option SASTRACE.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

Avec l'option SASTRACE activée, la création de la table a nécessité **6,49 secondes** !

Pourquoi ?

La réponse dans le journal SAS :

```
ORACLE_22: Executed: on connection 2
CREATE TABLE VISITEURS_LOT1(ID NUMBER (22),CODE_POSTAL NUMBER (13),ADRESSE_IP VARCHAR2
(50),NAVIGATEUR VARCHAR2 (50),RESOLUTION VARCHAR2 (50),OS VARCHAR2 (50),LANGUE VARCHAR2
(20),MOIS NUMBER (13),ANNEE NUMBER (13))


ORACLE_23: Executed: on connection 0
SELECT statement ORACLE_20


ORACLE_24: Prepared: on connection 2
INSERT INTO VISITEURS_LOT1
(ID,CODE_POSTAL,ADRESSE_IP,NAVIGATEUR,RESOLUTION,OS,LANGUE,MOIS,ANNEE) VALUES
(:ID,:CODE_POSTAL,:ADRESSE_IP,:NAVIGATEUR,:RESOLUTION,:OS,:LANGUE,:MOIS,:ANNEE)
```

Le journal SAS indique d'abord la requête de création de la table puis la requête d'insertion. Cependant, lors d'une insertion dans une table Oracle, les insertions sont réalisées, par défaut, par groupe de 10 observations. Toutes les 10 observations, les données à insérer sont envoyées à la base de données.

Le problème est qu'avec l'option SASTRACE activée le journal SAS est vite pollué par les traces d'insertion de ces paquets de 10 :

```
ORACLE_25: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_26: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_27: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_28: Executed: on connection 2
INSERT statement ORACLE_24
ORACLE: *-*-*-*-*-*-* COMMIT *-*-*-*-*-*-*
.
.
.
.

ORACLE_25: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_26: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_27: Executed: on connection 2
INSERT statement ORACLE_24

ORACLE_28: Executed: on connection 2
INSERT statement ORACLE_24
ORACLE: *-*-*-*-*-*-* COMMIT *-*-*-*-*-*-*
```

Ainsi, le journal SAS fait plusieurs milliers de lignes. **Nous avons des « EXECUTE » toutes les 10 observations et des « COMMIT » toutes les 1000 observations.**

Il est facile d'imaginer l'impact si nous travaillons sur une table contenant plusieurs millions d'enregistrement.

- Utilisez l'option **SASTRACE** lorsque vous êtes dans la phase de développement de votre programme. Si votre programme est destiné à être exécuté en mode batch ou que les performances ne sont pas au rendez-vous, vérifiez qu'une option SASTRACE ne s'est pas glissée dans votre code.
- Limitez, autant que vous le pouvez, le nombre d'observations lorsque vous développez ou débuggez avec l'option **SASTRACE** activée.

3.9. L'option DBIDIRECTEXEC pour soumettre directement votre requête SQL à la base de données.

Dernière option SAS qui nous intéresse, l'option DBIDIRECTEXEC. Avec cette option, les instructions DELETE et CREATE TABLE sont transmises directement au SGBD afin que le traitement soit beaucoup plus efficace.

Par défaut, cette option n'est pas activée.

```
options NODBIDIRECTEXEC;

data oraloc1.NODBIDIRECTEXEC;
      set oraloc1.visiteurs;
run;

proc sql;
create table oraloc1.NODBIDIRECTEXEC_SQL as
      SELECT *from oraloc1.visiteurs;
quit;
```

Dans les deux cas ci-dessus **le temps d'exécution est de 30 secondes.**
Le moteur SAS/ACCESS envoie les instructions suivantes à la base de données :

1. SELECT * FROM VISITEURS
2. CREATE TABLE NODBIDIRECTEXEC ….
3. INSERT INTO NODBIDIRECTEXEC …..

En positionnant l'option, les gains sont importants.

```
options DBIDIRECTEXEC;
```

En effet, avec une étape DATA, on ne constate aucune amélioration du temps de traitement :

```
options DBIDIRECTEXEC;

data oraloc1.DBIDIRECTEXEC_DATA;
   set oraloc1.visiteurs;
run;
NOTE: L'étape DATA used (Total process time):
   real time          29.43 secondes
```

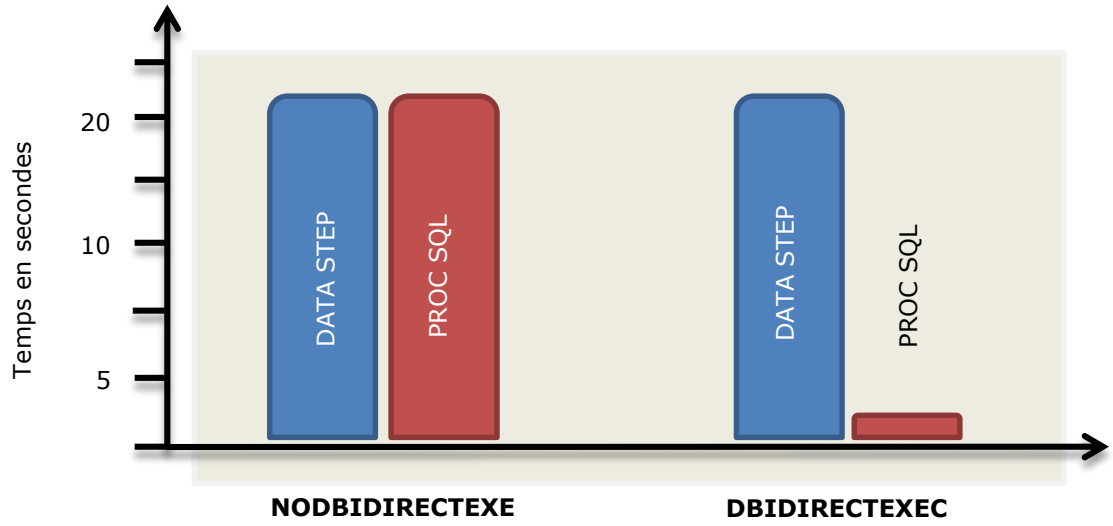
```
   user cpu time     4.83 secondes
   system cpu time   2.65 secondes
   memory       430.48k
   OS Memory      6768.00k
   Timestamp      18/07/2013 15 h 24
```

Cependant, avec une PROC SQL, le temps d'exécution passe à 1,5 seconde :

```
create table oraloc1.DBIDIRECTEXEC as
   SELECT *from oraloc1.visiteurs;
quit;
NOTE: PROCEDURE SQL used (Total process time):
   real time      1.64 secondes
   user cpu time     1.04 secondes
   system cpu time   1.01 secondes
   memory       370.22k
   OS Memory      6768.00k
   Timestamp      18/07/2013 15 h 26
```

Dans le cas de l'utilisation d'une PROC SQL associée à l'option DBDIRECTEXEC, le module SAS/ACCESS envoie une unique requête à la base de données :

CREATE TABLE DBIDIRECTEXEC as select * from VISITEURS



- L'option DBIDIRECTEXEC ne fonctionne pas avec les étapes DATA
- L'option DBIDIRECTEXEC fonctionne uniquement lorsque les instructions SQL ne concernent qu'une seule et même base de données.

3.10. Optimiser vos jointures

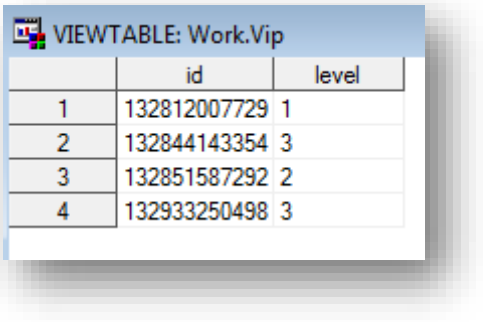
Toujours dans l'idée de limiter les données transitant entre la base de données et SAS, l'option **MULTI_DATASRC_OPT** apporte un gain significatif lors d'une jointure entre deux tables.

Lorsque vous réalisez cette jointure, le moteur SAS/ACCESS va déterminer la plus petite table des deux. Il utilisera les données de cette « petite » table pour filtrer directement la table plus

volumineuse. Cette opération a pour but de limiter directement le nombre de résultats retournés par la requête.

Prenons un exemple pour comprendre :

Nous avons une table VIP qui contient une liste de 4 visiteurs privilégiés.
Cette table n'est pas stockée dans la base de données mais il s'agit d'une table SAS stockée dans la WORK :



VIEWTABLE: Work.Vip

	id	level
1	132812007729	1
2	132844143354	3
3	132851587292	2
4	132933250498	3

Nous souhaitons extraire de notre table visiteurs Oracle les informations les concernant.

Après avoir positionné l'option dans le libname (**MULTI_DATASRC_OPT=in_clause**), la requête est la suivante :

```
proc sql;
      create table visiteurs_vip as
    select a.* from oraloc1.visiteurs a,vip b
       WHERE a.id=b.id;
quit;
```

Le moteur SAS/ACCESS exécute la requête suivante :

```
ORACLE_1: Prepared: on connection 1
SELECT * FROM SCOTT.VISITEURS

NOTE: Table WORK.VISITEURS_VIP created, with 8 rows and 9 columns.

76  quit;
NOTE: PROCEDURE SQL used (Total process time):
   real time       2.06 secondes
   user cpu time     0.48 secondes
   system cpu time   0.07 secondes
   memory          1697.43k
   OS Memory       11176.00k
   Timestamp       24/07/2013 03:51:03 PM
   Step Count      8 Switch Count 660
```

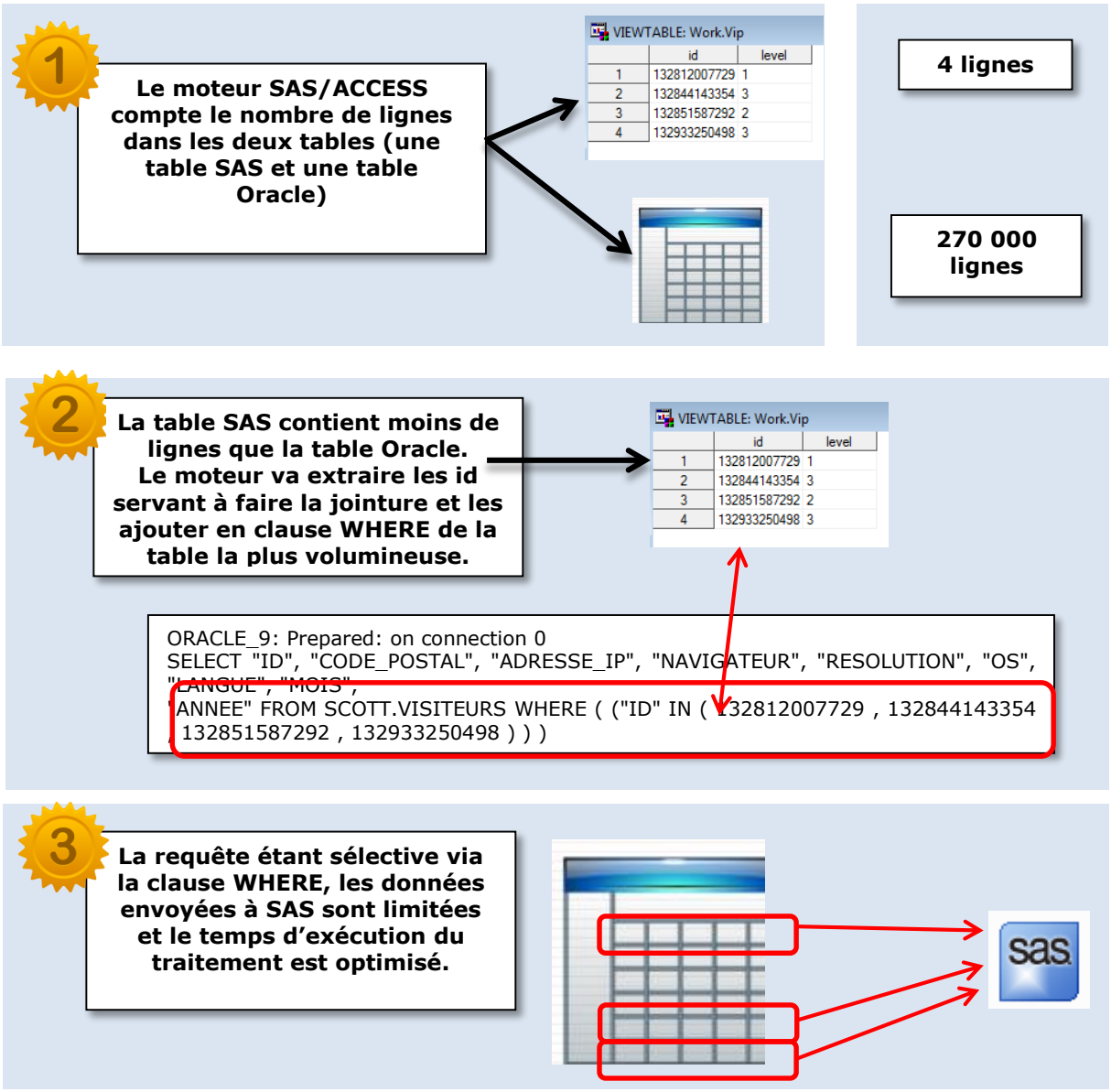
Avec l'option **MULTI_DATASRC_OPT** positionnée sur le LIBNAME, la requête envoyée à la base de données est toute autre :

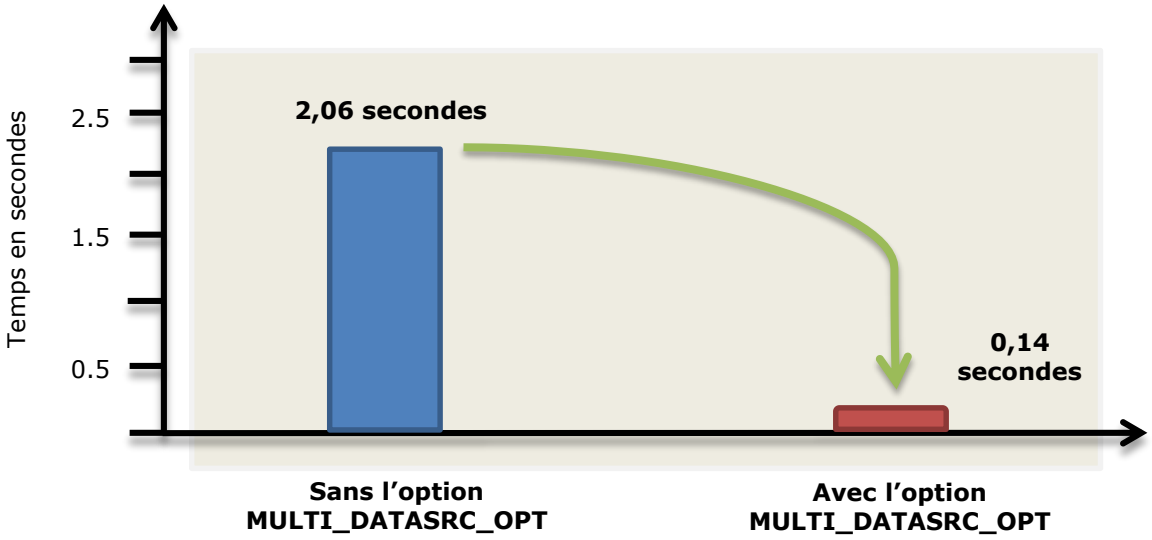
```
SELECT "ID", "CODE_POSTAL", "ADRESSE_IP", "NAVIGATEUR", "RESOLUTION", "OS", "LANGUE",
"MOIS",
```

```
"ANNEE" FROM SCOTT.VISITEURS WHERE ( ("ID" IN ( 132812007729 , 132844143354 , 132851587292
, 132933250498 ) ) )

NOTE: Table WORK.VISITEURS_VIP created, with 8 rows and 9 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.14 secondes
      user cpu time       0.00 secondes
      system cpu time     0.01 secondes
```

Le moteur SAS/ACCESS a modifié la clause WHERE de la requête envoyée afin de limiter le nombre de résultats retourné à SAS.

Le schéma ci-dessous présente en détail le fonctionnement lorsque cette option est positionnée :





- Le nombre d'éléments possible dans la clause « IN » est limité à 4500.
- Oracle limite la taille de la clause « IN » à 1000 éléments. Si la taille de la plus petite taille est supérieure à 1000, par exemple 4000, le moteur SAS/ACCESS va générer 4 clauses IN contenant chacune 1000 éléments.
- Dans le cadre d'une bibliothèque OLE DB cette limite est de 255.

3.11. Utiliser des tables volatiles

Lorsque vous effectuez des opérations complexes en base de données, le moteur de cette base stocke les résultats de certaines étapes intermédiaires et requêtes SQL dans autant de tables temporaires. Ces tables temporaires sont similaires à des tables classiques et utilisent le même espace de stockage. L'utilisation de tables volatiles permet d'optimiser les temps de traitement de certaines opérations.

Les tables volatiles sont des tables ayant une durée de vie très limitée. Elles sont créées en mémoire et n'existent que dans la session qui les a créées. Dès que la session se termine (déconnexion volontaire ou accidentelle), les tables volatiles sont supprimées.

Il est possible d'exécuter sur ces tables toutes les opérations que nous exécutons sur une table classique : insérer des données, les modifier, les supprimer, et bien sûr les sélectionner.

Dans une même session, si vous devez effectuer des calculs et/ou des requêtes plusieurs fois sur le même set de données, il est intéressant de stocker ce set de données dans une table volatile, pour travailler sur cette table. Une fois vos données de travail isolées dans cette table volatile, les requêtes vous servant à sélectionner les données qui vous intéressent seront simplifiées et plus rapides.

Pour connaitre la liste des SGBD compatibles avec les tables volatiles, veuillez-vous reporter à la documentation SAS 9.4

Afin d'illustrer l'utilisation des tables temporaires nous allons nous connecter à une base **TERADATA**.

La première étape consiste à initialiser la connexion à cette base de données via une instruction **LIBNAME**.

La première étape que nous allons mettre en place est de spécifier une option SASTRACE permettant d'avoir le plus d'informations possible dans le journal SAS :

```
options sastrace=(,,d,d) nostsuffix sastraceloc=saslog;
```

Exécutons ensuite l'instruction LIBNAME pour créer la bibliothèque :

```
LIBNAME test_vol TERADATA user="dbc" pw="dbc" server="192.168.72.130" connection=global
dbmstemp=yes;
```

Les SASTRACE indiquent la création d'une connexion à l'activation de la bibliothèque.

```
ACCESS ENGINE: Successful physical conn id 0
ACCESS ENGINE: Number of connections is 1

NOTE: Libref TEST_VOL was successfully assigned as follows:
      Engine:        TERADATA
      Physical Name: 192.168.72.130
```

Nous avons utilisé deux options nécessaires à l'utilisation des tables volatiles : **connexion**, positionnée à GLOBAL, et **dmbstemp** positionnée à YES.
Créons maintenant une table volatile dans la base de données en nous basant sur la table SAS **sashelp.pricedata:**

```

```

```
PROC APPEND DATA=sashelp.pricedata BASE=test_vol.volatile ;
RUN;
```

Les traces générées par l'option SASTRACE nous indiquent que les tables créées seront des tables volatiles. Nous avons également l'indication que la PROC APPEND partage la connexion créée lors de l'instruction LIBNAME :

```
ACCESS ENGINE: Entering DBIEXST with table name being volatile
ACCESS ENGINE: Successful SHARING existing connection id 0

ACCESS ENGINE: Entering dbrload with SQL Statement set to


ACCESS ENGINE: Entering dbrload with SQL Statement set to
          CREATE  VOLATILE MULTISET

ACCESS ENGINE: Entering dbrload with SQL Statement set to
          CREATE  VOLATILE MULTISET TABLE "volatile"
```

Enfin, nous créons une table test, dans la bibliothèque WORK, à partir de cette table volatile :

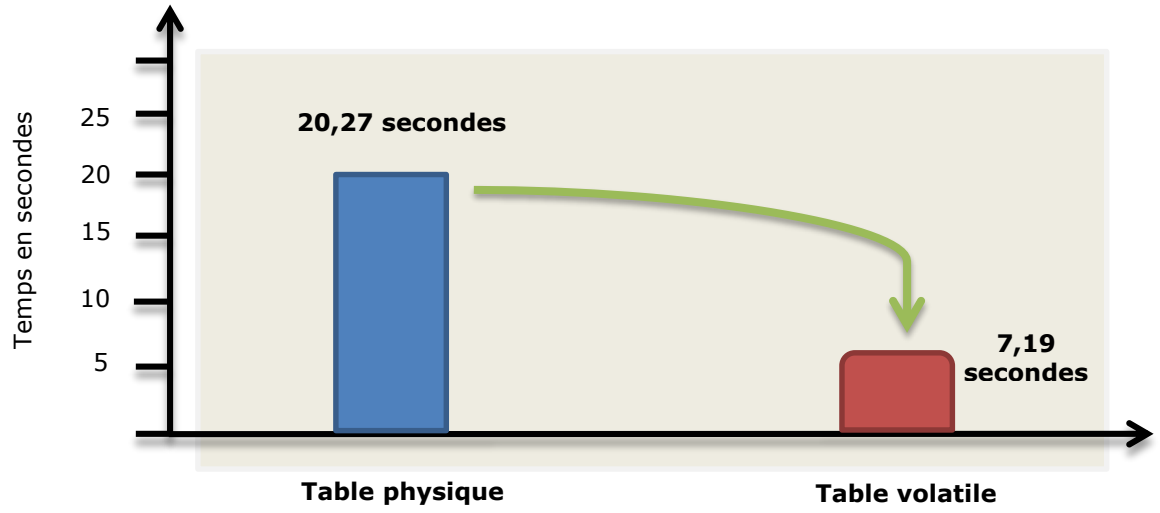
```
PROC SQL;
   create table test as SELECT * FROM test_vol.volatile;
QUIT;
```

Vous trouverez, ci-dessous, un extrait de SASTRACE correspondant à la PROC SQL. Comme pour la PROC APPEND, le moteur SAS/ACCESS réutilise la connexion « id 0 ».
A la fin de l'exécution de la PROC SQL, la connexion logique est bien terminée.

```
ACCESS ENGINE: Successful SHARING existing connection id 0
ACCESS ENGINE: Entering dbiopen

TERADATA_4: Prepared: on connection 0
SELECT * FROM "volatile"
TERADATA: trnvar()
TERADATA: trnvar(): Number of columns is 28
.
.
.
ACCESS ENGINE: Entering dbiclose
TERADATA: trclose()
TERADATA: trforc()
TERADATA: trforc: COMMIT WORK

DBMS_TIMER: summary statistics
DBMS_TIMER: total SQL execution seconds were: 0
DBMS_TIMER: total SQL prepare seconds were: 0
DBMS_TIMER: dbiopen/dbiclose timespan was 0.
ACCESS ENGINE: DBICLOSE open_id 0, connect_id 0
ACCESS ENGINE: Exiting dbiclos with rc=0X00000000
ACCESS ENGINE: Successful logical disconnect, id 0
```

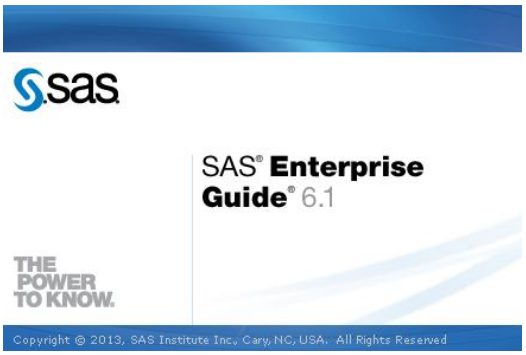
Comparons les temps de traitement pour ces deux étapes dans le cadre d'une utilisation avec et sans passer par une table volatile (en secondes) :

	Table physique	Table volatile
Etape 1 : PROC APPEND	**15,46**	**7,12**
Etape 2 : PROC SQL	**4,81**	**0,07**
TOTAL	**20,27**	**7,19**



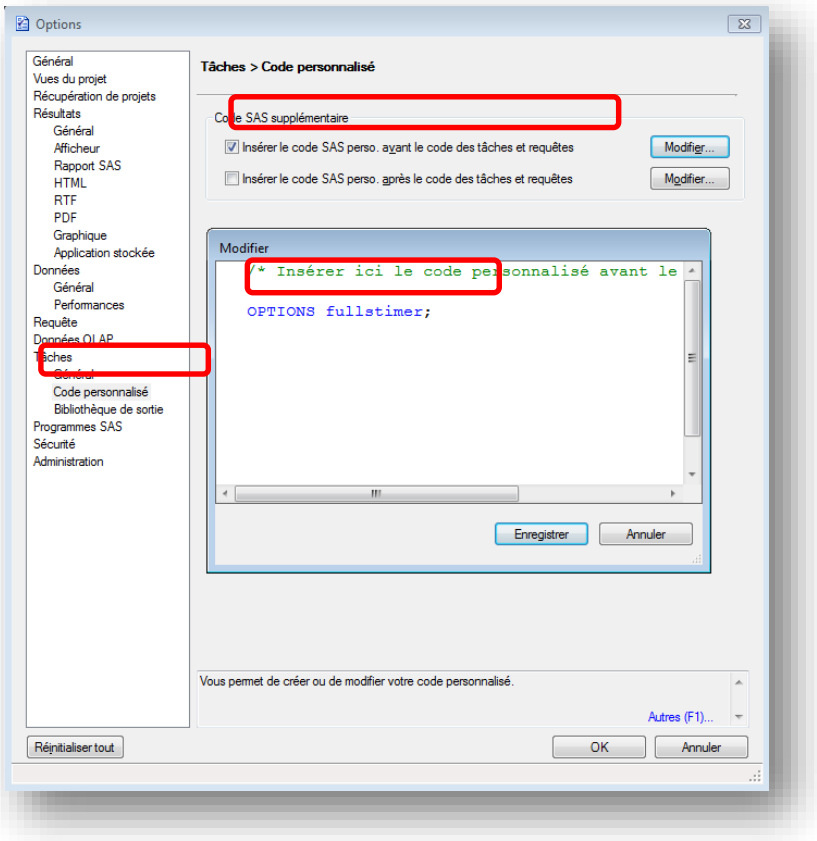
Lorsque vous utilisez des tables volatiles, il n'est pas nécessaire que l'utilisateur possède les droits d'écriture dans la base de données puisque la création de cette table s'effectue en mémoire.

4. COMMENT FORCER L'UTILISATION DU SQL PASSTHROUGH EXPLICIT DANS SAS ENTERPRISE GUIDE 6.1 ?



Les comportements et concepts décrits précédemment sont aussi applicables à SAS Enterprise Guide.

Par exemple, l'utilisateur peut utiliser plusieurs méthodes pour soumettre l'option SASTRACE ou l'option DBIDIRECTEXEC, notamment par le biais de l'option « code personnalisé », comme le montre la figure suivante :

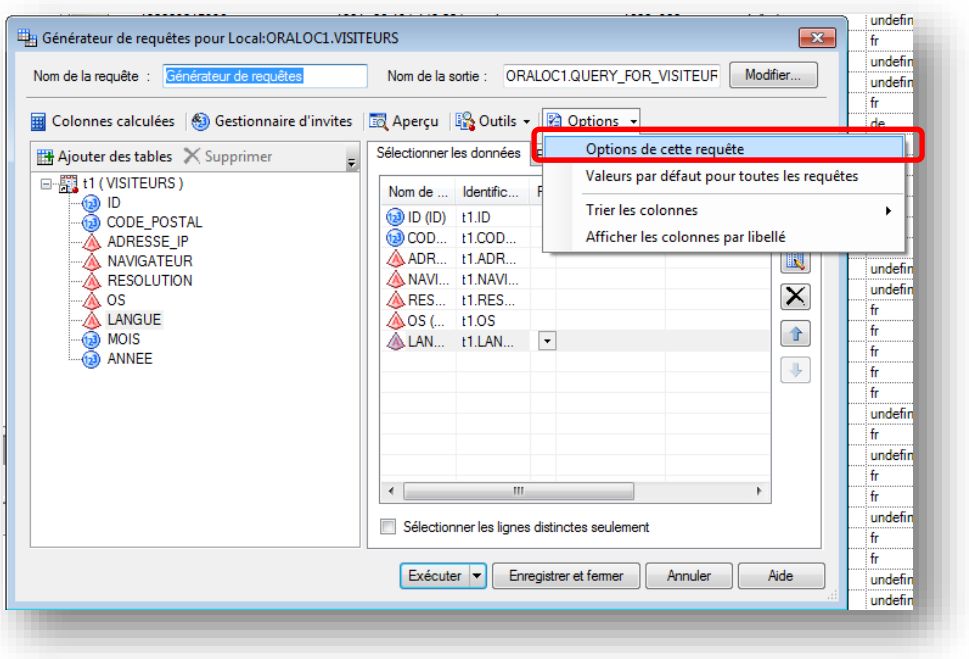


Dans SAS Enterprise Guide, **les requêtes SQL utilisent par défaut le SQL Pass-through implicite.**

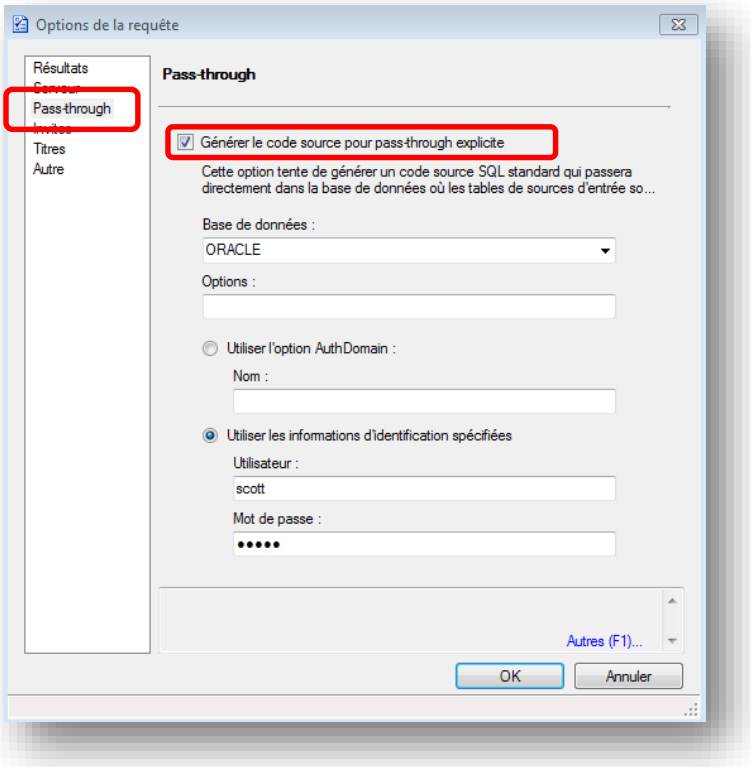
Le Générateur de requêtes permet de basculer vers du **SQL Pass-through explicite.**

Pour accéder à cette option, effectuez les manipulations suivantes :

- Dans le générateur de requêtes, cliquez sur le menu "Options" et selectionnez "Options de cette requête »



- Sélectionnez Pass-through dans le menu de gauche
- Cochez l'option « Générer le code source pour pass-through explicit ». Vous pouvez également saisir les options de votre choix, comme le nom de la base de données, l'utilisateur, le mot de passe….



5. LIENS UTILES ET REFERENCES

Pour ceux qui souhaitent approfondir le sujet, voici quelques références et liens utiles pour vous accompagner dans vos recherches :

- Levin, Louid. "Methods of Storing SAS® Data into Oracle Tables." Proceedings of SUGI 29, May 2004. http://www2.sas.com/proceedings/sugi29/106-29.pdf
- SAS® 9.4 SQL Procedure User's Guid : http://support.sas.com/documentation/cdl/en/sqlproc/65065/PDF/default/sqlproc.pdf
- Frank Capobianco, (Teradata Corporation) : "Explicit SQL Pass-Through: Is It Still :Useful ?" : http://support.sas.com/resources/papers/proceedings11/105-2011.pdf
- Liming, Douglas B. "Five Ways to Speed Up Your Data Loading Using SAS/ACCESS® for Relational Databases" : http://support.sas.com/resources/papers/proceedings11/103-2011.pdf
- Functions that are added via the SQL_FUNCTIONS= option might not be passed to the database management system (DBMS) : http://support.sas.com/kb/42/934.html
- SAS/ACCESS(R) 9.4 for Relational Databases: Maximizing Oracle Performance : http://support.sas.com/documentation/cdl/en/acreldb/65053/HTML/default/viewer.htm#p0fmgbpjwgbnvkn1sysefnr22fow.htm
- SAS/ACCESS(R) 9.4 for Relational Databases: Reference, Second Edition : Performance Considerations : http://support.sas.com/documentation/cdl/en/acreldb/66690/HTML/default/viewer.htm#n1v1cfazem7dejn1xiq60o8zd45g.htm
- Usage Note *23194:* When using SAS/ACCESS software, the PROC SQL queries that are included in my code do not always pass through to the DBMS engine. How can I make sure my queries are passed through? : http://support.sas.com/kb/23/194.html
- Usage Note *41616:* Sample of LIBNAME statement and SQL Pass-Through code to connect to Oracle database using SAS/ACCESS® Interface to Oracle : http://support.sas.com/kb/41/616.html
- Problem Note *47918:* The TODAY() and DATEPART() functions might return a DATETIME rather than a DATE variable when you query a database management system (DBMS) table : http://support.sas.com/kb/47/918.html
- Passing SAS Functions to Oracle : http://support.sas.com/documentation/cdl/en/acreldb/65247/HTML/default/viewer.htm#p0f64yzzxbsg8un1uwgstc6fivjd.htm
- A Guide to Efficient PROC SQL Coding : http://support.sas.com/resources/papers/sgf09/336-2009.pdf
- Problem Note 47859: READBUFF values that are set to more than 10,000 are truncated to 10,000 by DataDirect drivers : http://support.sas.com/kb/47/859.html
- Géraldine Cade-Deschamps (mars 2012) : SAS® In-Database : Quand le traitement SAS s'exécute dans le SGDB : http://www.sas.com/offices/europe/france/US2012_Q1_InDatabase.html

6. CONCLUSION

L'optimisation est un art mais aussi un combat de tous les jours. Vous ne serez jamais totalement formé. Mais même pour un débutant il y a juste quelques « *boutons à pousser* » pour rendre les traitements flux de données plus rapides.

Mais n'oubliez jamais que :

- SAS offre de nombreuses possibilités d'optimisation,
- Ces solutions sont simples à mettre en œuvre,
- Ces solutions sont bénéfiques avec peu d'effort,
- Assurez-vous d'une utilisation efficace des ressources,
- La mémoire tampon consomme de la mémoire,
- Il vous faudra un certain effort pour trouver les réglages optimaux en fonctionnement de votre environnement et de son utilisation,
- Utilisez les SASTRACE durant la phase de développement mais n'oubliez pas de la désactiver avant le passage en production,
- Et enfin, mais non des moindres, toujours tester les options sur de petits sous-ensembles de données.

Nicolas Housset
Consultant Support Clients SAS France