PRÉSENTATION DE LA PROCÉDURE FEDSQL

Dans une entreprise moderne, il est inévitable d'utiliser différents systèmes de gestion de bases de données réparties géographiquement, et décentralisées pour stocker et rechercher les données critiques. Pourtant, c'est seulement en combinant les informations provenant de ces différents systèmes que l'entreprise peut réaliser la pleine valeur des données qu'ils contiennent. Nouveauté SAS 9.4, le langage FEDSQL répond à cette problématique en permettant une vue uniforme et transparente des données stockées dans différents sources, tout en préservant leur autonomie locale mais offre également une gamme plus large de fonctionnalité ainsi qu'une plus grande souplesse d'utilisation.

Caractéristiques :

Catégories : Base SAS®

OS : Windows, Unix

Version : SAS® 9.4M3

Vérifié en Mai 2016

Sommaire

1. PRÉSENTATION DE LA FÉDÉRATION DE DONNÉES ET FEDSQL

Répondant à la norme ANSI SQL 1999 (appelé aussi SQL 3), le langage FEDSQL a pour objectif de faciliter et normaliser les jointures entre plusieurs tables et plusieurs sources de données. C'est de cet objectif que provient son nom FEDSQL, pour Federated SQL. Il s'agit d'une procédure du module Base SAS.

D'une façon générale, la notion de fédération de données et de base de données fédérée (Federated database system) renvoie à la combinaison de données provenant de plusieurs sources de données indépendantes. Ainsi, les données peuvent provenir de plusieurs bases de données.

On parle alors d'**architecture fédérée**. C'est-à-dire une architecture centrée sur les données.

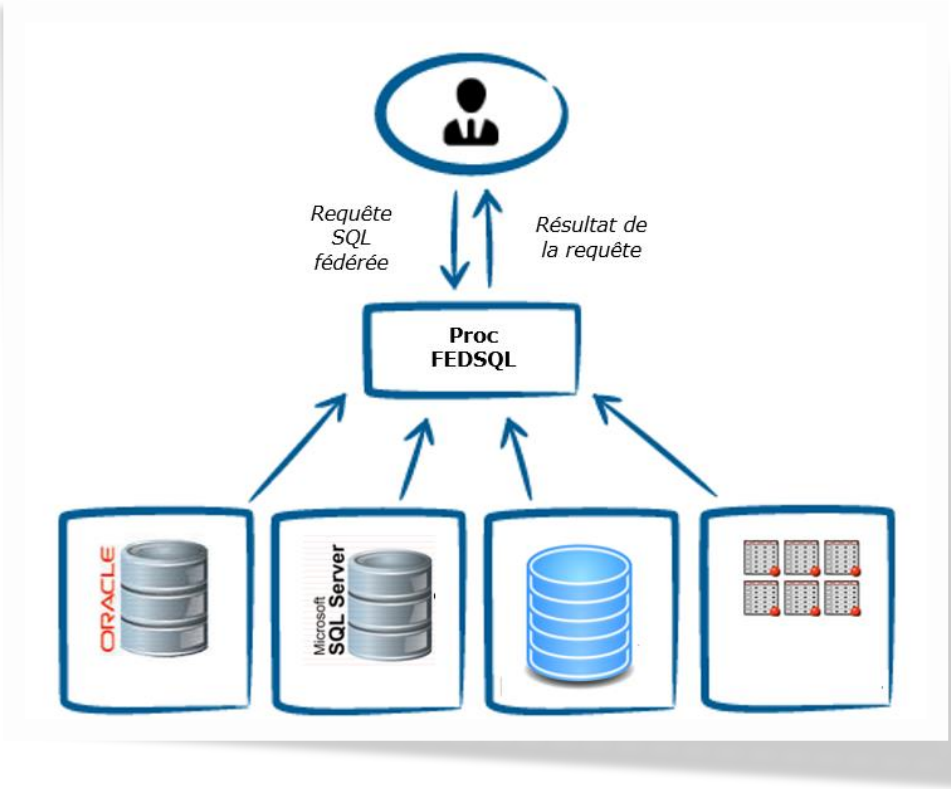
Dans l'entreprise, l'objectif de la création d'une base fédérée est de donner aux utilisateurs une vue unique des données présentes sur plusieurs systèmes a priori hétérogènes.

Aussi, la fédération de données répond à trois besoins :

- ☑ Donner aux utilisateurs une vue unique de données implémentées sur plusieurs systèmes à priori hétérogènes.
- ☑ Conserver les données où elles résident, plutôt que de les déplacer dans un magasin de données unique.
- ☑ Envoyer des requêtes réparties sur plusieurs sources de données dans une seule instruction SQL.

Derrière ce nom se cache une pratique courante que tout développeur utilisant le SQL connait. En effet, on appelle requête fédérée, une requête accédant à des données provenant de multiples sources de données et renvoyant un jeu de résultats uniques.

Le schéma ci-dessous explique le concept de requête fédérée :



Ainsi, comme la PROC SQL, la PROC FEDSQL apporte une plus grande richesse dans les opérations disponibles, permettant ainsi une meilleure modélisation et facilite l'interrogation des données stockées. La spécification des requêtes est indépendante du système de base de données, du système d'exploitation et de la disposition physique de données.

La syntaxe de base de la PROC FedSQL est similaire à celle de la PROC SQL :

```
PROC FEDSQL;
<Votre requête SQL>
QUIT;
```

Mais les avantages de FEDSQL ne se limitent pas à une plus grande souplesse dans la gestion des jointures entre plusieurs tables et plusieurs bases de données.

Le chapitre suivant présente les avantages et nouveautés offerts avec la proc et le langage FEDQL.

2. QUELS SONT LES AVANTAGES DE FEDSQL ?

Comme nous l'avons vu dans le chapitre d'introduction, la proc fedsql permet de réaliser, avec une plus grande souplesse des jointures entre plusieurs bases de données.

Mais ce n'est pas tout et le tableau ci-dessous présente les avantages de cette nouvelle procédure :

- ☑ FedSQL propose une syntaxe SQL commune à toutes les sources de données.
- ☑ Une seule requête FedSQL peut cibler des données dans plusieurs sources de données et renvoyer une seule table de résultat.
- ☑ FedSQL est une implémentation propriétaire des standards de la norme SQL ANSI: 1999.
- ☑ FedSQL supporte de nouveaux types de données tels que bigint, décimal, double, etc. FedSQL permet la traduction de et vers les types de données SAS existants tels que le numérique et le caractère SAS.
- ☑ FedSQL a la capacité d'effectuer des jointures hétérogènes dans le cas où les tables de la jointure sont réparties sur plusieurs bases de données, venant de SGBD hétérogènes.
- ☑ FedSQL peut exécuter des requêtes incluant une sous-requête imbriquée (Heterogeneous correlated subqueries)
- ☑ FedSQL peut générer les données résultantes dans n'importe laquelle des sources de données supportées indépendamment. Ceci permet aux utilisateurs d'adapter les sources de données en fonction des besoins.
- ☑ Le langage FEDSQL permet de créer des données dans n'importe quelle base de données supportées. Il est ainsi possible de stocker vos données au plus près de la source.

3. QUELLES SONT LES BASES DE DONNÉES SUPPORTÉES ?

DB2 for UNIX & PC Operating Environments	SAP HANA
Hadoop (Hive and HDMD)	Sybase IQ
Memory Data Store (MDS)	SAS data sets
MySQL	SAS Scalable Performance Data Engine (SPD Engine) data sets
Netezza	Teradata
ODBC databases (Comme Microsoft SQL Server)	Oracle
Aster	SAP (en lecture seule)

La liste de bases de données supportées évoluant à chaque maintenance de SAS, vous pouvez consulter la liste mise à jour sur le site du support SAS. L'accès aux différents SGBD est possible sous couvert d'avoir le module SAS/Access® Interface to adéquat sous licence et fonctionnel.

4. PREMIERS PAS AVEC LA PROC FEDSQL

Maintenant que les présentations sont faites, nous pouvons nous plonger dans le vif du sujet.
Dans les chapitres suivants nous allons voir comment nous connecter à une base et accéder aux données.

4.1. Se connecter aux données

Comme vous allez le voir, le mode de fonctionnement ne diffère pas du mode dans lequel vous avez l'habitude de travailler.

Démarrons avec un cas simple d'utilisation de la PROC FedSQL en utilisant le contenu de la bibliothèque WORK :

```
proc fedsql;
        SELECT * FROM work.test;
quit;
```

Dans un premier temps, nous allons voir comment créer simplement une connexion en illustrant chaque méthode d'un exemple simple de requête FedSQL (Uniquement des instructions SELECT). Puis dans le chapitre « Premiers pas avec la PROC FedSQL » il vous sera présenté différents exemples de PROC FedSQL. Aussi, ce chapitre étant focalisé sur le mécanisme de connexion il n'était pas nécessaire de complexifier pour la compréhension.

Avec l'ajout d'une déclaration LIBNAME, PROC FEDSQL est en mesure d'accéder à Oracle :

```
libname ora oracle user=xxxx pw=xxx path=xxxx;

proc fedsql;
select * FROM ORA.test;
quit;
```

Une bibliothèque supplémentaire pour base peut également être ajoutée :

```
libname base 'd:\base';
```

FedSQL a maintenant trois sources de données, que nous nommons « catalogues », à sa disposition, à savoir 1 bibliothèque SAS, une table de la WORK et une table Oracle.

Ces trois catalogues sont alors accessibles au sein de la PROC FEDSQL :

```
proc fedsql;
select * from base.test;
select * from WORK.test;
select * from ORA.test;
quit;;
```

4.2. Création, insertion et suppression

Ce nouvel exemple simple montre comment créer une table dans la WORK, y insérer des données puis supprimer cette table :

```
proc fedsql;
        create table work.a (x double);
        insert into work.a values (1.0);
        insert into work.a values (2.0);
        insert into work.a values (3.0);
```

```
        select * from work.a;
        drop table work.a;
quit;
```

4.3. Requête d'agrégation simple

Cet exemple montre une requête d'agrégation en utilisant la sashelp.class :

```
proc fedsql;
select sex, avg(height) as avg_height, avg(weight) as avg_weight from
sashelp.class group by sex;
quit;
```

4.4. Proc SQL versus Proc FEDSQL

4.4.1. 2 façons différentes d'accéder aux données

Les requêtes SQL soumises via la PROC FEDSQL sont exécutées en implicit pass through. Aussi, lorsque la requête est envoyée à la base, celle-ci est convertie dans le code spécifique à la source de données interrogée. Pour comprendre comment fonctionne la PROC FEDSQL, nous allons soumettre une requête en utilisant la PROC SQL, puis la PROC FEDQL.

Nous utiliserons également l'option _METHOD afin d'afficher, dans le journal, la méthode utilisée par SAS pour exécuter la requête.

La requête est la suivante :

```
select distinct(F.id_produit) FROM mysqllib.FACTURES F;
```

Le code est le suivant :

```
proc sql _method;
select distinct(F.id_produit) FROM mysqllib.FACTURES F;
quit;

proc fedsql _method;
select distinct(F.id_produit) FROM mysqllib.FACTURES F;
quit;
```

Regardons maintenant le journal SAS :

Proc SQL :

```
NOTE: SQL execution methods chosen are:
sqxslct
sqxextr( connection to SASIOMYL /* dbms=SASIOMYL, connect options=() */ ( select
distinct F.`ID_PRODUIT` from `FACTURES` F ) )
```

Proc FEDSQL :

```
Methods: Full query pushdown!
```

Cet exemple montre que le chemin pour exécuter la requête n'est pas le même en fonction de la procédure utilisée. Pour plus d'information sur l'option _METHOD, vous pouvez consulter le document [Exploring the Undocumented PROC SQL _METHOD Option]

A noter également, qu'avec des requêtes plus complexes exécutées via une PROC FEDSQL, les informations fournies par cette option _METHOD peuvent s'avérer intéressantes. L'exemple ci-dessous montre les informations lorsque nous exécutons une jointure entre deux bases de données :

```
Number of Sorts Performed is : 3
Number of Joins Performed is : 2
Number of Merge Joins Performed is : 1
        NestLoop (IN)
          SeqScan from MYSQLLIB.SAS.PRODUITS
          SubqueryScan
            Sort
              MergeJoin (INNER)
                Sort
                  SeqScan from NTZ.ADMIN.CLIENTS
                Sort
                  SeqScan from MYSQLLIB.SAS.FACTURES
```

4.4.2. Des résultats plus précis

Autre avantage : la Proc FEDSQL permet aussi de travailler de façon plus précise avec les BIGINT.

Le type de données bigint s'adresse aux situations dans lesquelles les valeurs entières sont susceptibles de ne pas appartenir à la plage prise en charge par le type de données INT.

Partons de la table MYSQL suivante :

#	Colonne	Type	Interclassement	Attributs	Null
1	ID	double			Oui
2	BIG_INTEGER	bigint(20)		UNSIGNED	Oui
3	INTEGER	int(11)			Non

Cette table contient les données suivantes :

ID	BIG_INTEGER	INTEGER
1	8446744073709000000	2147480000
2	135252500	147483600
3	170212047483650123	2040483647

Comparons maintenant les résultats de deux requetés SQL exécutés avec la PROC SQL et la PROC FESDQL :

```
Proc sql;
        TITLE 'Résultat PROC SQL - BIGINT';
        select BIG_INTEGER from mysqllib.T_BIGINTEGER;
        select sum(BIG_INTEGER) as TOTAL from mysqllib.T_BIGINTEGER;
quit;


Proc fedsql;
        TITLE 'Résultat PROC FEDSQL - BIGINT';
        select BIG_INTEGER from mysqllib.T_BIGINTEGER;
        select sum(BIG_INTEGER) as TOTAL from mysqllib.T_BIGINTEGER;
quit;
```

Résultat PROC SQL - BIGINT

BIG_INTEGER
8446744073708999680
135252500
170212047483650112

Résultat PROC SQL - BIGINT

TOTAL
8.617E18

Résultat PROC FEDSQL - BIGINT

BIG_INTEGER
8446744073709000000
135252500
170212047483650123

Résultat PROC FEDSQL - BIGINT

TOTAL
8616956121327902623

Nous constatons alors que les résultats sont plus fiables et plus précis avec la PROC FEDSQL.

5. PRÉSENTATION DES DIFFÉRENTS MODES DE CONNEXION ET DE LEUR FONCTIONNEMENT

Le chapitre précédent vous a permis de comprendre le fonctionnement la PROC FedSQL et d'en apprécier les avantages. Vous avez pu le constater, cette procédure fonctionne comme la PROC SQL et utilise les bibliothèques initialisées via l'instruction LIBNAME pour accéder aux données

Toutefois, il existe une autre façon de rendre accessible les données distantes à PROC FedSQL.

Aussi, ce chapitre présente les différentes possibilités pour vous créer des catalogues accessibles et utilisables par la PROC FedSQL. Nous abordons également, dans les chapitres 6.2 et 6.3, le piège à éviter et comment le contourner.

5.1. Afficher la chaîne de connexion

Soumettons d'abord une instruction LIBNAME pour accéder à des données Mysql, en utilisant la couche ODBC. Le code est le suivant :

```
libname my_sql odbc dsn=MYSQL user=<utilisateur> password=<mot_de_passe>;
```

Admettons maintenant que nous souhaitons connaître les bibliothèques disponibles, c'est-à-dire les catalogues qu'il est possible d'interroger et d'utiliser à l'intérieur de la PROC FedSQL.

Le code à soumettre est le suivant :

```
options msglevel=i;

proc fedsql _method;
quit;
```

Le résultat de cette commande est le suivant :

```
NOTE: Connection string:
NOTE: DRIVER=FEDSQL;CONOPTS= ( (DRIVER=ODBC;DB=MYSQL;UID=root;PWD=*;CATALOG=MY SQL);
(DRIVER=base;CATALOG=MAPS;SCHEMA=
      (NAME=MAPS;PRIMARYPATH={C:\Program Files\SASHome\SASFoundation\9.4\maps}));
(DRIVER=base;CATALOG=MAPSSAS;SCHEMA=
      (NAME=MAPSSAS;PRIMARYPATH={C:\Program Files\SASHome\SASFoundation\9.4\maps}));
      (DRIVER=base;CATALOG=MAPSGFK;SCHEMA= (NAME=MAPSGFK;PRIMARYPATH={C:\Program
      Files\SASHome\SASFoundation\9.4\mapsgfk})); (DRIVER=base;CATALOG=SASUSER;SCHEMA=
      (NAME=SASUSER;PRIMARYPATH={C:\Users\franih\Documents\My SAS Files\9.4}));
(DRIVER=base;CATALOG=WORK;SCHEMA=
      (NAME=WORK;PRIMARYPATH={C:\Users\franih\AppData\Local\Temp\SAS Temporary
Files\_TD1164_NBDEL068_})))
```

Nous constatons alors que notre bibliothèque MYSQL est bien présente :

```
NOTE: DRIVER=FEDSQL;CONOPTS= ( DRIVER=ODBC;DB=MYSQL;UID=root;PWD=*;CATALOG=MY_SQL);
```

La déclaration d'une bibliothèque, via l'instruction LIBNAME, n'est pas une obligation pour travailler avec la PROC FedSQL.

En effet, il est possible de « connecter » la PROC FedSQL directement à la base de données en utilisant les options DRIVER, via l'option CONN= :

```
proc fedsql conn="(DRIVER=ORACLE;UID=mdm41;PWD=xxxxx;PATH=MDM;CATALOG=ORA);";
quit;
```

Ainsi, et comme nous l'avons vu en introduction, il est possible de combiner les chaînes de connexion pour travailler avec plusieurs bases de données au sein de votre PROC FedSQL :

```
proc fedsql conn="(DRIVER=base;CATALOG=BASE;SCHEMA=
(NAME=BASE;PRIMARYPATH={d:\base}));(DRIVER=ORACLE;UID=mdm41;PWD=*;PATH=MDM;CATA
LOG=ORA);";
```

5.1. Combiner l'instruction LIBNAME et les chaînes de connexion avec la procédure FEDSQL

Par défaut, la PROC FedSQL combine les chaînes de connexion des bibliothèques disponibles avec toute chaîne de connexion fournie via l'option "conn ="

Aussi, admettons le code SAS ci-dessous :

```
libname ora oracle user=mdm41 pw=xxxx path=MDM;
libname base 'd:\base';
proc fedsql;
...
```

Nous savons que nous arriverons à un résultat similaire en initialisant nos connexions de la façon suivante :

```
libname ora oracle user=mdm41 pw=xxxx path= MDM;
proc fedsql
conn="(DRIVER=BASE;CATALOG=BASE;schema=(name=BASE;primarypath='d:\base'));";
```

Nous avons notre connexion à la base Oracle réalisée via l'instruction LIBNAME et notre connexion à notre bibliothèque BASE via l'option CONN= de la PROC FedSQL.

Au final, nos deux catalogues ORA et BASE sont bien accessibles et utilisables par la PROC FedSQL.

5.2. Problématique des doublons de catalogues

Mais maintenant, que se passe-t-il si je soumets un code en laissant, volontairement, une instruction LIBNAME de création de ma bibliothèque base et en définissant également cette bibliothèque base via l'option CONN ?

```
libname ora oracle user=mdm41 pw=xxxx path= MDM;
```

```
libname base 'd:\base';

proc fedsql
conn="(DRIVER=BASE;CATALOG=BASE;schema=(name=BASE;primarypath='d:\base'));";
```

Nous obtenons le message d'erreur :

```
ERROR: Duplicate catalog name, BASE, encountered in connection string, DSN or
file DSN
ERROR: TKTS initialization failed.
```

Soumettons le code vu au chapitre 5.1 pour afficher les catalogues en mémoire :

```
options msglevel=i;

proc fedsql _method;
quit;
```

Le résultat de cette commande est le suivant :

```
NOTE: Connection string:
NOTE: DRIVER FEDSQL,CONOPTS=( (DRIVER ORACLE,UID mdm41,PWD *,PATH MDM;CATALOG=ORA);
(DRIVER=base;CATALOG=BASE;SCHEMA=(NAME=BASE;PRIMARYPATH={d:\base}));
(DRIVER=base;CATALOG=MAPS;SCHEMA= (NAME=MAPS;PRIMARYPATH={C:\Program
      Files\SASHome\SASFoundation\9.4\maps})); (DRIVER=base;CATALOG=MAPSSAS;SCHEMA=
      (NAME=MAPSSAS;PRIMARYPATH={C:\Program Files\SASHome\SASFoundation\9.4\maps}));
      (DRIVER=base;CATALOG=MAPSGFK;SCHEMA= (NAME=MAPSGFK;PRIMARYPATH={C:\Program
      Files\SASHome\SASFoundation\9.4\mapsgfk})); (DRIVER=base;CATALOG=SASUSER;SCHEMA=
      (NAME=SASUSER;PRIMARYPATH={C:\Users\franih\Documents\My SAS Files\9.4}));
(DRIVER=base;CATALOG=WORK;SCHEMA=
      (NAME=WORK;PRIMARYPATH={C:\Users\franih\AppData\Local\Temp\SAS Temporary
Files\_TD6100_NBDEL068_})))
```

Nous constatons que la PROC FedSQL « a déjà connaissance » d'un catalogue nommé BASE. Nous savons que ce catalogue a été défini via l'instruction LIBNAME.
C'est pour cette raison qu'il n'est pas possible d'ajouter ce même catalogue, via l'option CONN=.
Pour résumer, La PROC FedSQL ne peux donc pas "dupliquer" les catalogues.

L'instruction LIBNAME a créé en mémoire un catalogue « BASE ». En ajoutant, en paramètre de la PROC FedSQL, une connexion à BASE, nous nous retrouvons avec un doublon et obtenons un message d'erreur.

Les noms des catalogues doivent être uniques pour que la PROC FEDSQL puisse y faire référence dans les requêtes soumises.

La PROC FedSQL ne peut pas faire référence à deux catalogues portant des noms identiques.

5.1. Une solution : L'option NOLIBS

Comme nous venons de le voir, le nom des catalogues doit être unique. Il n'est donc pas possible d'avoir le nom d'un catalogue en doublon.
Cela peut poser problème si vous n'avez pas la main sur la création des bibliothèques. En effet, elles peuvent avoir été créées via un fichier AUTOEXEC ou bien être définies dans les métadonnées.
Vous n'êtes donc pas au bout de vos surprises et vous pourriez vite être confronté à des doublons.
Il est alors possible d'utiliser l'option NOLIBS afin d'indiquer à la PROC FedSQL de n'utiliser QUE les catalogues définis avec l'option CONN= et, ainsi, d'ignorer toutes les connexions faites en amont de la PROC FedSQL :

```
proc fedsql nolibs
conn="(DRIVER=BASE;CATALOG=BASE;schema=(name=BASE;primarypath='d:\base'));";
quit;
```

Nous constatons alors que nous n'avons qu'un unique catalogue accessible dans la PROC FEDSQL :

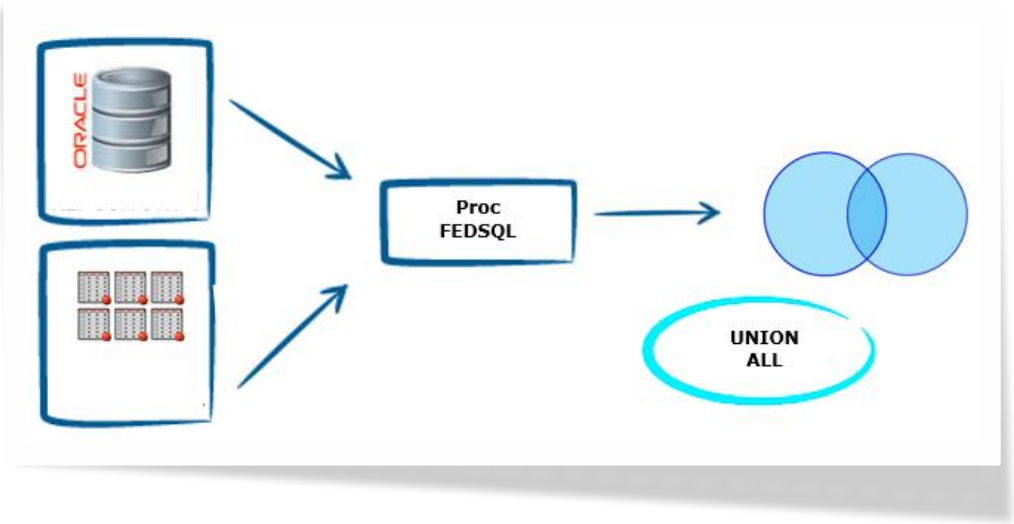
```
NOTE: Connection string:
NOTE: DRIVER=FEDSQL;CONOPTS= (
(DRIVER=BASE;CATALOG=BASE;SCHEMA=(NAME=BASE;PRIMARYPATH={d:\base})))
NOTE: Current catalog set to BASE
```

6. EXEMPLES D'UTILISATION DE LA PROC FEDSQL

Cette section présente des exemples d'utilisation de la PROC FEDSQL. Elle offre un panorama des fonctionnalités proposées par la procédure.

6.1. Accéder à plusieurs sources de données avec une requête fédérée

Comme nous l'avons vu en introduction, nous touchons là au « cœur » de la PROC FedSQL.
Dans l'exemple ci-dessous, nous accédons aux données d'une base SAS et aux données d'une base Oracle. La PROC FedSQL utilise deux bibliothèques pour se connecter aux deux sources de données. Nous pouvons ensuite créer, simplement, une jointure entre nos données :

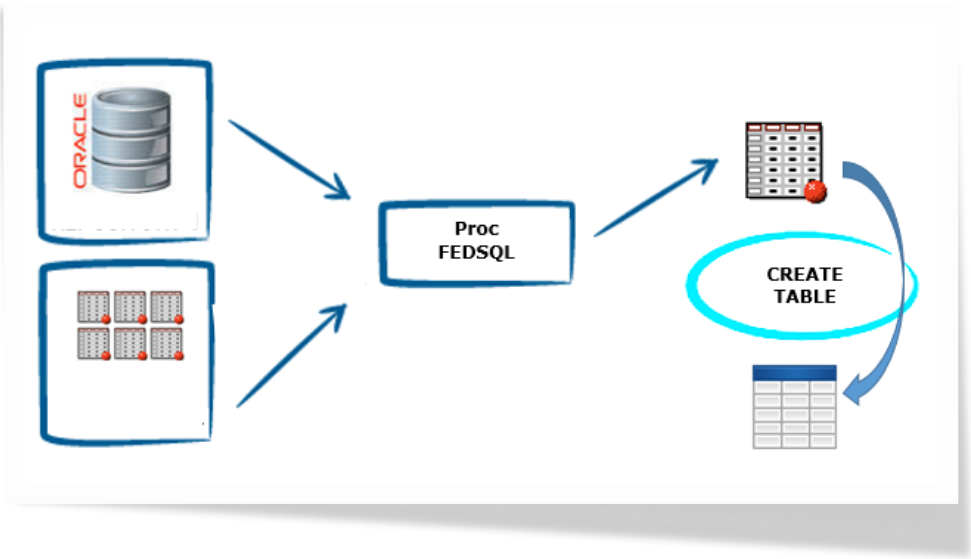


Le code est le suivant :

```
libname base 'd:\base';
libname ora oracle user=xxx pw=xxx path=xxx;
proc fedsql;
SELECT * FROM ora.test UNION ALL SELECT * FROM work.test;
quit;
```

6.2. Créer une table à partir d'une table existante

Cet exemple crée une table à partir de tables existantes en utilisant PROC FedSQL et l'instruction SQL CREATE TABLE. Via la requête, nous sélectionnons les lignes des tables existantes pour créer la nouvelle table.




```
libname mybase base 'C:\Base';
libname myspde spde 'C:\Spde';
libname myoracle oracle path=ora11g user=xxxxxx password=xxxxxx schema=xxxxxx;
proc fedsql;
   create table mybase.results as
      select product.prodid, product.product, customer.name,
         sales.totals, sales.country   from myspde.product, myoracle.sales
myoracle.customer where product.prodid = sales.prodid and
         customer.custid = sales.custid;
   select * from mybase.results;
quit;
```

6.3. Rapatrier des données avec une sous-requête corrélée

L'exemple suivant illustre l'interrogation des données en utilisant une sous-requête corrélée. Une sous-requête corrélée est une sous-requête qui fait référence à une table qui n'est pas définie dans sa clause FROM, mais bien ailleurs dans la requête dont elle fait partie. Ainsi, dans une sous-requête corrélée, la clause WHERE dans la sous-requête fait référence aux valeurs d'une table dans la requête externe.

La sous-requête corrélée est évaluée pour chaque ligne de la requête externe.

```
proc fedsql;
        SELECT * from mysqllib.produits WHERE mysqllib.produits.id in (
```

```
        select distinct(mysqllib.factures.id_produit) FROM
NTZ.clients,mysqllib.factures WHERE NTZ.clients.id=mysqllib.factures.id_client
);
quit;
```

ID	NOM_PRODUIT
17	TRAITEMENT MULTI USAGE XYLOPHENE 20ANS 30L
23	LOT 10 BOITES C.SECHE D.67 BATIBOX

6.4. Création et utilisation d'un index pour exécuter une jointure

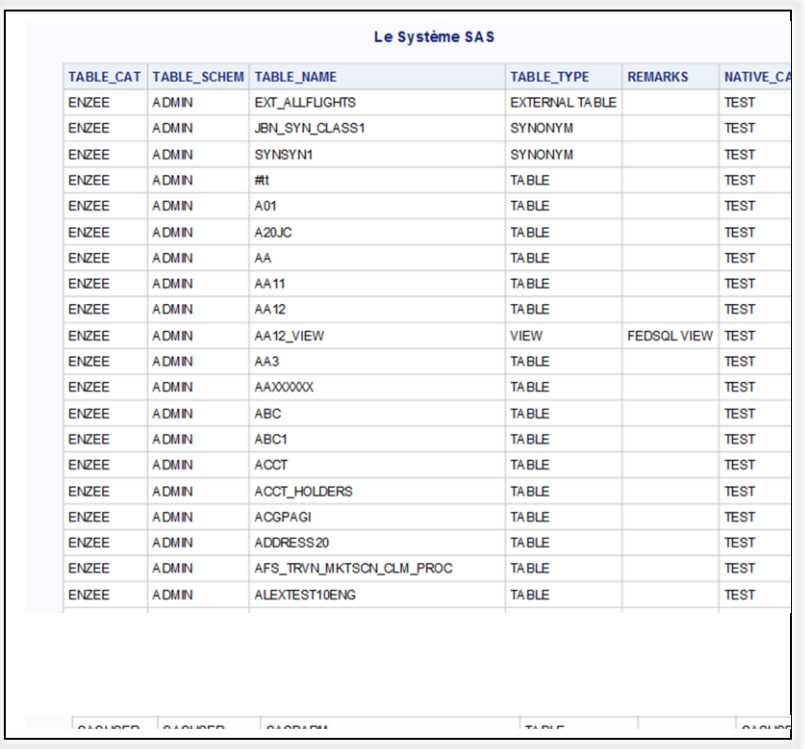
L'organisation des tables sous forme d'index est assez commune, les enregistrements sont stockés dans l'ordre de la clé primaire. On dit alors que la table est organisée en index. Dans ce nouvel exemple, vous verrez comment il est possible, dans une PROC FEDSQL, de créer un index pour une table Oracle, pour l'utiliser ensuite afin d'effectuer une jointure de la table Oracle et un ensemble de données provenant d'une autre base de données.

```
libname my_sql odbc dsn=MYSQL user=xxxxx password=xxxxxxx;

libname myoracle oracle path=ora11g user=xxxxxx password=xxxxxx schema=xxxxxx;

proc fedsql;
  create index prodid on myoracle.sales (prodid);

  select * from my_sql.product, myoracle.sales
  where product.prodid=sales.prodid;

quit;
```

6.5. Lister les tables disponibles et utilisables à l'intérieur de la proc FedSQL

Une requête sur la table DICTIONARY.TABLES permet de lister l'ensemble des tables utilisables dans votre Proc FedSQL :

```
PROC FedSQL;
        SELECT * FROM DICTIONARY.TABLES;
QUIT;
```



Le Système SAS

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	NATIVE_CA
ENZEE	ADMIN	EXT_ALLFLIGHTS	EXTERNAL TABLE		TEST
ENZEE	ADMIN	JBN_SYN_CLASS1	SYNONYM		TEST
ENZEE	ADMIN	SYNSYN1	SYNONYM		TEST
ENZEE	ADMIN	#t	TABLE		TEST
ENZEE	ADMIN	A01	TABLE		TEST
ENZEE	ADMIN	A20JC	TABLE		TEST
ENZEE	ADMIN	AA	TABLE		TEST
ENZEE	ADMIN	AA11	TABLE		TEST
ENZEE	ADMIN	AA12	TABLE		TEST
ENZEE	ADMIN	AA12_VIEW	VIEW	FEDSQL VIEW	TEST
ENZEE	ADMIN	AA3	TABLE		TEST
ENZEE	ADMIN	AAXXXXX	TABLE		TEST
ENZEE	ADMIN	ABC	TABLE		TEST
ENZEE	ADMIN	ABC1	TABLE		TEST
ENZEE	ADMIN	ACCT	TABLE		TEST
ENZEE	ADMIN	ACCT_HOLDERS	TABLE		TEST
ENZEE	ADMIN	ACGPAGI	TABLE		TEST
ENZEE	ADMIN	ADDRESS20	TABLE		TEST
ENZEE	ADMIN	AFS_TRVN_MKTSCN_CLM_PROC	TABLE		TEST
ENZEE	ADMIN	ALEXTEST10ENG	TABLE		TEST

6.6. Appliquer les options de tables

Lorsque vous manipulez des tables avec la PROC FedSQL, vous pouvez appliquer des options sur ces tables.

Une option de table permet de spécifier une action à réaliser sur une table, comme, par exemple, définir la valeur du "Buffer page size" (BUFSIZE=) ou encore spécifier un mot de passe.

Dans l'exemple ci-dessous créons une table en définissant une valeur personnalisée pour le "Buffer page size" :

```
libname Mabase base  'd:\base';
```

```
proc fedsql;

   create table Mabase.Table1 {options bufsize=16k}(x double) ;

   insert into Mabase.Table1 values (1.0);
   insert into Mabase.Table1 values (2.0);
   insert into Mabase.Table1 values (3.0);
quit;
```

6.7. Sécuriser ses tables

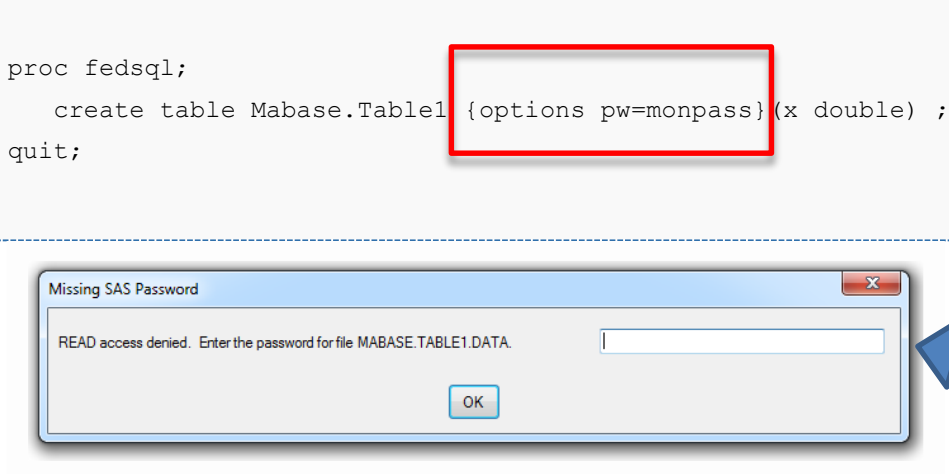
FedSQL permet également de sécuriser ses tables.
SAS vous permet de restreindre l'accès à des ensembles de données SAS en attribuant des mots de passe aux fichiers.

Vous pouvez spécifier trois niveaux de protection: lire, écrire, et modifier.

Ainsi, vous pouvez spécifier un mot de passe pour une source de données en utilisant les options de table FedSQL ALTER, PW, READ ou WRITE.

Le code suivant applique l'option de la table FedSQL PW afin d'attribuer un mot de passe pour les opérations READ, WRITE, et ALTER :

```
libname Mabase base  'd:\base';

proc fedsql;
   create table Mabase.Table1 {options pw=monpass}(x double) ;
quit;
```



6.8. Fonctionnement des transactions avec FEDSQL

FedSQL permettant, dans une même procédure d'effectuer plusieurs actions, il est intéressant de se poser la question du fonctionnement des transactions.
Pour rappel, les transactions sont un concept fondamental de tous les systèmes de bases de données. Une transaction rassemble plusieurs étapes en une seule opération que nous pouvons qualifier de "tout-ou-rien". En effet, les états intermédiaires entre les étapes ne sont pas visibles par les transactions concurrentes.

FedSQL supporte les instructions COMMIT et ROLLBACK fournissant une « protection » des données en veillant à ce que les mises à jour sont soient pleinement appliquées ou annulées si l'opération est interrompue. Il est en effet primordial de garantir que si « quelque chose se passe mal », aucune des étapes déjà exécutées n'est prise en compte.

Ainsi, une transaction est déclarée en entourant les commandes SQL de la transaction par les commandes BEGIN et COMMIT.

```
PROC FedSQL;
        BEGIN;
                <ma requete>
        COMMIT;
        BEGIN;
        <ma requete>
        ROLLBACK;
QUIT;
```

Attention, par défaut un mécanisme d'AUTOCOMMIT est positionné. Ce qui a pour conséquence d'annuler l'effet des commandes BEGIN, COMMIT et ROLLBACK et une transaction est effectivement commencée avec chaque mise à jour des données.

Pour désactiver l'AUTOCOMMIT, la syntaxe est la suivante :

```
LIBNAME LIBREF < ---------- > AUTOCOMMIT=NO;
```

A noter également que le support des transactions n'est pas disponible pour toutes les bases de données. Les transactions fonctionnent uniquement avec les bases Aster, DB2, Greenplum, Microsoft SQL Server, MySQL, ODBC databases, et Sybase IQ.

6.9. Créer et utiliser des vues avec FEDSQL

Les vues ont une importance croissante dans les bases de données. Dans la pratique, les vues permettent de définir des tables virtuelles correspondant aux besoins des programmes d'application en termes de données.

Pour illustrer le fonctionnement des vues FedSQL, commençons par créer une table « physique » en soumettant le code suivant :

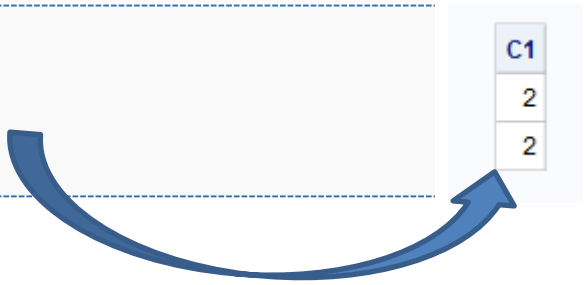
```
proc FedSQL;
        CREATE TABLE MaTable (C1 DOUBLE);
        INSERT INTO MaTable VALUES (1);
        INSERT INTO MaTable VALUES (2);
        INSERT INTO MaTable VALUES (3);
QUIT;
```

Puis nous créons maintenant une vue sur cette table en limitant les données retournées par cette vue :

```
PROC FedSQL;
        CREATE VIEW MaVue AS SELECT * FROM MaTable WHERE C1 > 1;
QUIT;
```

Du point de vue de l'utilisateur, l'interrogation au travers d'une vue s'effectue comme pour une table normale :

```
PROC FedSQL;
        SELECT C1 FROM MaVue;
QUIT;
```



Visualisons la vue dans le dictionnaire des tables SAS :

```
PROC FedSQL;
        SELECT * FROM DICTIONARY.TABLES WHERE TABLE_TYPE="VIEW";
QUIT;
```

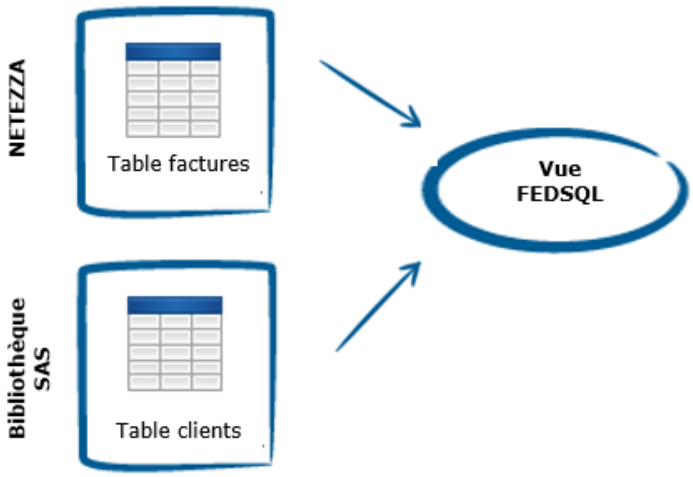
Le Système SAS

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	NATIVE_CAT
WORK	WORK	MAVUE	VIEW	FEDSQL VIEW	WORK

Si notre vue n'est plus utile, il est possible de la supprimer via une requête DROP VIEW :

```
PROC FedSQL;
        DROP VIEW MaVue;
QUIT;
```

Bien entendu, l'exemple présenté ci-dessus est très simple et ne reflète pas la puissance apportée par ces vues FedSQL. Les vues permettent, par exemple la définition d'une vue jointure de deux tables, ou résultant d'un calcul d'agrégats. Ainsi, les vues FedSQL permettent de réaliser par avance des cumuls ou synthèses sophistiqués :



```
PROC FedSQL;
        CREATE VIEW mavue AS
                SELECT clients.nom,NTZ.factures.nom_produit
                FROM clients,NTZ.factures
                WHERE clients.id=NTZ.factures.id_client;
quit;
```

6.10. Exemples de chaînes de connexion

Voici quelques exemples de chaînes de connexion adaptées à chaque type de base de données :s

Teradata

```
(DRIVER=TERADATA;UID=<uid>;PWD=<password>;server=<Teradata_server>;database=<base>;CATALOG=TERA;)
```

DB2

```
(CATALOG=DB2;DRIVER=DB2;DB=<base>;UID=<uid>;PWD=<password>)
```

SQL Server

```
(DRIVER=ODBC;UID=<uid>;PWD=<password>;CONOPTS=(DSN=<dsn sql server>);CATALOG=*)
```

Netezza

```
(DRIVER=NETEZZA;UID=<uid>;PWD=<password>;SERVER=<server netezza>;DATABASE=test;CATALOG=*)
```

ODBC Mysql

```
(driver=odbc;catalog=testdata; uid=<uid>;pwd=<password>;odbc_dsn=<dsn>)
```

7. ERREURS QUE VOUS POUVEZ RENCONTRER

```
Table "SASHELP.CLASS" does not exist or cannot be accessed
ERROR: [42S02]Table "SASHELP.CLASS" does not exist or cannot be accessed
(0x81bfc10b)
ERROR: [3F000]BASE driver, schema name SASHELP was not found for this
connection (0x81bfc8d1)
ERROR: Table "SASHELP.CLASS" does not exist or cannot be accessed
ERROR: BASE driver, schema name SASHELP was not found for this connection
```

⇨ La bibliothèque SASHELP est une bibliothèque concaténée et la PROC FEDSQL ne fonctionne pas avec les bibliothèques concaténées.

```
ERROR: Duplicate catalog name, BASE, encountered in connection string, DSN or
file DSN
ERROR: TKTS initialization failed.
```

⇨ Vérifier que vous n'avez pas deux fois la définition d'un catalogue en mémoire. Pour plus d'information sur cette erreur, dans le chapitre 5.2 de cet article.

```
ERROR: [42000]Function MD5(CHAR) does not exist (0x81bfc10b)
ERROR: Function MD5(CHAR) does not exist
```

8. LIENS UTILES

Quelques liens utiles pour aller plus loin et approfondir les notions abordées dans cet article :

- SAS® 9.4 Language Reference, Fourth Edition
- SAS® 9.4 FEDSQL Language Reference, Third Edition
- Lafler, Kirk Paul. 2013. "Exploring the PROC SQL _METHOD Option." *Proceedings of the SAS Global 2013 Conference*
- SAS PROC FEDSQL Documentation.
- Exploring the Undocumented PROC SQL _METHOD Option
- Working with PROC FEDSQL in SAS® 9.4

9. CONCLUSION

Comme nous avons pu le voir, cette nouveauté SAS 9.4, n'est pas le remplacement de la PROC SQL mais apporte de nouvelles fonctionnalités intéressantes en termes de gestion de base de données. Aussi, l'objectif premier de cette nouvelle procédure est de faciliter les jointures entre différentes sources de données et nécessite une bonne connaissance du langage SQL et du système informatiques que l'on souhaite interroger.

Nicolas HOUSSET
Consultant Support Clients SAS France