integrix

Composants & Tools

Nom du tools	uv ODBC
Statut	✔

Rédacteur(s)

- — *Manu Fernandes 27/05/2018 09:37* create

Généralités

Explique commet exposer correctement une db uv via ODBC alorsque l'app n'a pas été prévue pourcela (SB+,...)

Description

ODB permet d'exposer les tables et colonnes via le driver ODBC ; pour faire des appels SQL ; ou de subroutine

Compte tenu que l'accompte APP contient toute un série de chose que lsoit on ne désire pas exposer soit qui ne fonctionne pas sous ODBC; on ne va pas travailler dans l'app même.

Syntaxe

ODBC n'aime pas les éléments avec des . (dot) ou des espaces (space) et est case-sensitive !

- Les séparateurs seront des _ (underscore)
- Les éléments seront écrit en majuscule

Initialisation

Au shell,

1. dans UVDB
2. créer le directory xxSQL
   ```
   mkdir xxSQL
   ```
3. descendre dans ce directory
   ```
   cd xxSQL
   ```
4. lancer uv
   ```
   uv
   ```
5. accepter la création du système
   ```
   Would you like to set it up (Y/N)? Y
   ```
6. prendre le flavor
   ```
   3 Ideal
   ```
7. ajouter une PA LOGIN d'information
   ```
   INSERT INTO VOC (@ID,F1,F2) VALUES ('LOGIN','PA','DISPLAY bienvenue en xxSQL');
   ```
8. fixer un pointeur sur UV.ACCOUNT
   ```
   SET.FILE UV UV.ACCOUNT UV.ACCOUNT
   ```
9. ajouter xxSQL dans le UV.ACCOUNT as xxSQL
   ```
   INSERT INTO UV.ACCOUNT (@ID,PATH) VALUES ('xxSQL','path/xxSQL');
   ```
10. se rendre dans l'account HS.ADMIN pour activer les outils odbc
    1. lancer
       ```
       HS.ADMIN
       ```

2. Option

```
3 : Activate access to files in an account
```

3. donner le xxSQL

.

11. se rendre dans xxSQL
    1. fixer les droit par défaut à NONE

```
UPDATE HS_FILE_ACCESS SET ACCESS = 'NONE' ;
```

    2. compléter l'exposition de rien :

```
HS.UPDATE.FILEINFO
```

Créer un account 'vide' (appSQL)

```
cd UVDB
mkdir xxSQL
cd xxSQL
uv
This directory is not set up for uniVerse.
Would you like to set it up (Y/N)? "Y"
Which way do you wish to configure your VOC ? "0" "Ideal"
Your VOC is configured for Ideal UniVerse compatibility
Creating file VOC as Type 3, Modulo 23, Separation 4.
Creating file D_VOC as Type 3, Modulo 2, Separation 1.
Loading your VOC file.  (Each "*" = 10 records.)
**************************************************************************
Loading your D_VOC file.  (Each * = 10 records.)
Creating file &SAVEDLISTS& as Type 1.
Creating file D_&SAVEDLISTS& as Type 3, Modulo 1, Separation 2.
Added @ID, the default record for RetrieVe, to D_&SAVEDLISTS&.
Creating file VOCLIB as Type 2, Modulo 7, Separation 4.
Creating file D_VOCLIB as Type 3, Modulo 1, Separation 2.
Added @ID, the default record for RetrieVe, to D_VOCLIB.

SET.FILE UV UV.ACCOUNT UV.ACCOUNT

INSERT INTO UV.ACCOUNT (@ID,PATH) VALUES ('xxSQL','path/xxSQL');

INSERT INTO VOC (@ID,F1,F2) VALUES ('LOGIN','PA','DISPLAY bienvenue en xxSQL');
```

Activer ce compte pour odbc

```
LOGTO HS.ADMIN
HS.ADMIN

UniVerse Server Administration
1. List activated accounts
2. Show UniVerse ODBC Config configuration for an account
3. Activate access to files in an account
4. Deactivate access to files in an account
5. Run HS.SCRUB on a File/Table
6. Update File Information Cache in an account

Which would you like? ( 1 - 6 )  ? "3"

Enter the account name or path, or press Return for a list of accounts in
which file access has already been activated: "xxSQL"
Activating access to files in account xxSQL
"Le système crée un fichier HS_FILE_ACCESS "
Creating data portion of HS_FILE_ACCESS file.
Creating file HS_FILE_ACCESS as Type 30.
Creating dict portion of HS_FILE_ACCESS file.
Creating file D_HS_FILE_ACCESS as Type 3, Modulo 1, Separation 2.
Added @ID, the default record for RetrieVe, to D_HS_FILE_ACCESS.
Generating file information cache.
HS.FILEINFO errors saved in xxSQL/&COMO&/HS_FILE_ERRS
27 MAY 2018 (11:53:21)      ----------------------------------------
"message normal, &KEYSTORE& est protégé"
```

```
Could not open file &KEYSTORE&.  STATUS() = -2
&KEYSTORE& was not written into File Information Cache.
"messages normaux"
HELP.FILEs dictionary has no @ phrase or @SELECT phrase.
HELP.FILE was written into File Information Cache but may not behave as desired.
Could not create @EMPTY.NULL record in DICT HS_FILE_ACCESS - no permission.
HS_FILE_ACCESS was written into File Information Cache but may not behave as desired.
Updating UV.ACCOUNT file.
"Activation completed."
```

Tables

Définir un fichier exposé en tant que TABLE(s) exposées en ODBC ex : xxTABLE.

Chaque ASSOC multivaluée sera vu comme une table_assocname

Le principe veut que l'on définisse les DICT COLUMN indépendamment du DICT applicatif.

On crée un novueau DICT local à xxSQL et on fait pointer le DATA level vers la data réelle.

```
CREATE.FILE DICT xxTABLE 18 53 1
UPDATE VOC SET F2 = 'path/xxDATAFILENAME' WHERE @ID = 'xxTABLE';
INSERT INTO HS_FILE_ACCESS (FILENAME,ACCESS) VALUES('xxTABLE','READ');
INSERT INTO DICT xxTABLE (@ID,F1) VALUES('@EMPTY.NULL','X');
```

Init DICT local

1. créer un DICTIONNAIRE LOCAL vide
   ```
   CREATE.FILE DICT xxTABLE 18 53 1
   Creating file "D_xxTABLE" as Type 18, Modulo 53, Separation 1.
   Added "@ID", the default record for RetrieVe, to "D_xxTABLE".
   ```
2. le columnname @ID est créé d'office
3. fixer le path du datavalue du DICT
   ```
   UPDATE VOC SET F2 = 'path/xxDATAFILENAME' WHERE @ID = 'xxTABLE';
   UniVerse/SQL: 1 record updated.

   CT VOC xxTABLE
   xxTABLE
   0001 F
   0002 path/xxDATAFILENAME
   0003 D_xxTABLE
   ```

Fixe les droits

Insérer un READ dans HS_FILE_ACCESS xxTABLE

```
INSERT INTO HS_FILE_ACCESS (FILENAME,ACCESS) VALUES('xxTABLE','READ');
```

Special DICT keyword

@SELECT

Sera la PHrase utilisée lorsque l'on demande un SELECT * FROM xxTABLE;

La PH @SELECT contient la liste des DICt/ColumnName à présenter.

```
DICT xxTABLE @SELECT
0001 PH
0002 columnname columnname ...
```

@EMPTY.NULL

Veut dire que le NULL est accepté pour les chmps

```
INSERT INTO DICT xxTABLE (@ID,F1) VALUES('@EMPTY.NULL','X');
```

```
DICT xxTABLE @EMPTY.NULL
0001 X
```

ASSOCiation multivaluée

- Les associations décrivent la liste des columnname dont les multivalue sont dépendants.
- Les associations sont exposées comme des (sous)tables et sont nommées :**xxTABLE_*assocname***
- L'assoc name est défini dans le champs
  - 0006 m
  - 0007 *assocname*
- Tous les columnname d'une même *assocname* portent le meme assocname en attribut 7

ASSOC PHRASE

Une assoc_phrase décrit les column names exposé dans la table xxTABLE_*assocname*

```
DICT xxTABLE assocname
0001 PH
0002 columnname columnname columnname
```

@ASSOC_KEY.assocname

Le X-type spécial @ASSOC_KEY.assocname sera exploitée automatiquement pour créer la columnname comme champs clé automatique.

si on ne le déclare pas, se sera le nom `@ASSOC_ROW`.

DICT xxTABLE @ASSOC_KEY.*assocname* 00001 X

Colonnes

Structure

Les dictionnaires / représentent les colonnes exposées.

AMC	D-Type	I-Type
000	columnname : no special chars)	
001	D	I
002	n in @RECORD	I-type expression
003	Mask convertion	
004	Label	
005	nnL!R!T size to prsent the data at UV tcl	
006	S if single value M if Multivalue	
007	if 006=M : assocname if empty id = ASSOCNAME	
008	SqlDataType	

sql Data Types

Il faut respecter les SQL data type !

Les DICT doivent être de type D-type ou I-type.

Dictionary	remarque
CHAR [ACTER] [(n)]	n = number of characters
DEC [IMAL] [(p[,s])]	p = precision s = scale
FLOAT[(p)]	p = precision
NUMERIC[(p[,s])]	p = precision s = scale
VARCHAR[(n)]	n = number of characters
DATE	
DOUBLE PRECISION	

Dictionary	remarque
INTEGER	
REAL	
SMALLINT	
TIME	

Procédure de création

Paragraphe pour init d'une table

```
PA.TABLE.INIT tablename path.to.data

VOC PA.TABLE.INIT
PA
CREATE.FILE DICT <<I2,xxTABLE>> 18 53 1
UPDATE VOC SET F2 = '<<I3,PATH_DATAFILE>>' WHERE @ID = '<<I2,xxTABLE>>';
INSERT INTO HS_FILE_ACCESS (FILENAME,ACCESS) VALUES('<<I2,xxTABLE>>','READ');
INSERT INTO DICT <<I2,xxTABLE>> (@ID,F1) VALUES('@EMPTY.NULL','X');
INSERT INTO DICT <<I2,xxTABLE>> (@ID,F1) VALUES('@SELECT','PH');
```

Règle pour les colonnes

Pour toute création de columnname on procède tel que :

1. définition du DICT xxTABLE columnname

```
INSERT INTO DICT xxTABLE (ITEM.ID, CODE, LOC.L, CONV, NAME, FORMAT, SM, ASSOC, DATATYPE)
VALUES ('columnName', 'D!I', 'n!exp', '', 'columnName human', 'nR!L', 'S!M',
'empty!assocname', 'sqlDataType');

UniVerse/SQL: 1 record inserted.

CT DICT xxTABLE columnName

columnName
00001 D!I
00002 n!EXP
00003
00004 columnName human
00005 nR!L
00006 S!M
00007 empty!assocname
00008 sqlDataType
```

2. si c'est un I-type : compilation :

```
CD xxTABLE columname
```

3. si c'est une mv'ed : mise à jour de l'association DICT xxTABLE *assocname* pour y insérer en 002 le columnname à sa place ; l'ordre des columnName fait l'ordre des colonnes dans un `SELECT * FROM xxTABLE_assocname;`
4. mise à jour du DICT xxTABLE @SELECT pour y insérer le columnname à sa place. l'ordre des columnName fait l'ordre des colonnes dans un `SELECT * FROM xxTABLE;`

Il peut être util de vérifier si les données sont en phase avec les dict.

en lançant un HS.SCRUB xxTABLE

```
"HS.SCRUB xxTABLE  {AUTOFIX DICT}"
Running IN report mode.  No scrubbing will occur.

Installing TEMPORARY ON.ABORT record IN VOC
HS.SCRUB output will be saved IN /DATA/uvdbsb55/xxSQL/&COMO&/xxTABLE.SCRUB

Analyzing: xxTABLE

HS.SCRUB REPORT FOR xxTABLE         27 MAY 2018 (19:10:55)
SAL was analyzed USING VARCHAR(254) FOR a DATA TYPE
FNAME was analyzed USING VARCHAR(254) FOR a DATA TYPE
LNAME was analyzed USING VARCHAR(254) FOR a DATA TYPE
```

```
Deleting TEMPORARY ON.ABORT record FROM VOC
```

Lorsque les modifications sont terminées, on lance le

```
"HS.UPDATE.FILEINFO"

Updating file information cache.
This will have an effect ONLY IF access TO files IN this account has been activated.
```

qui va recréer le petit fichier .hs_fileinfo (caché) qui sera lu par les clients ODBC pour connaître les tableName et columnName.

Le client doit se déconnecter, reconnecter pour recevoir les mise à jour.

Méthodologie pour la mise à jour via ODBC

- Compte tenu que les db UV sont créés avant la notion de SQL, nous retrouvons souvent des structures et modèle physique complexe
  - Genre : les multivaleurs, les sous valeurs, les convertions, les records 'fusion' ...
  - → Il est très difficile à modéliser en mode Relationnel exploitable en SQL.
- Compte tenu que les db UV n'intègrent pas de contraintes fortes au niveau des tables (sauf à les mettre en palce, mais sur les app classiques ce n'est jamais fait)
  - → Il est dangereux d'exposer les fichiers en écriture (le pq il faut définir HS_FILE_ACCESS en mode READ(only)

Pour pallier aux problèmes de ce genre et réaliser une mise à jour via odbc, sql, la méthode idéale est la suivante :

1. créer un/plusieurs fichier/table 'plat', dédié à la réception des données depuis odbc, sans mv avec des dictionnaires/colonnes clairement typés pour odbc/sql et ouvrir ces fichiers/table en écriture.
   - le client odbc met à jour ces tables pour intégrer des données via
     ```
     INSERT INTO ...
     ```
2. créer une subroutine BASIC qui va lire les données de la table d'import pour réaliser les mise à jour vers les fichiersUV de production
   - cette routine ne contient pas de . (dot) dans sont nom et est cataloguée dans le compte ..._SQL.
3. le client ODBC, après avoir intégré les données via INSERT procède à un appel de la routine d'intégration tel que :
   - un Programme
     ```
     {CALL "routine arg arg... "}
     ```
     - le programme récupère les param via @SENTENCE ou GET(ARG.)
   - une Subroutine
     ```
     {CALL subroutine(parameter1, … parametern)}
     ```
     - la routine reçoit les parametres en ...paramètre
4. le prog/subr procède aux mise à jour, ... et efface les tuples de la table temporaire
5. les 'echo / PRINT / CRT ' du programme / subroutine sont les résultat reçus par le client odbc ; Des tuples, comme pour un SELECT.

Documents administratifs

- uv odbc ref gui de 7.241.0 oct2016