Programme de service SQL

- Les articles -

Date de mise en ligne : mardi 9 janvier 2007

Description :

Permet d'optimiser le codage des requêtes SQL dynamiques

Environnement iSeries

SGSRVSQ est un programme de service permettant de gérer la clause WHERE pour les requêtes dynamiques SQL.

SQL dynamique permet d'encapsuler une requête SQL dans une variable pour l'utiliser avec un "EXECUTE IMEDIATE" ou avec un curseur. Généralement, on préférera exécuter une instruction dynamique en deux temps : une étape de préparation de l'instruction permettant de vérifier la validité de la requête, puis l'exécution proprement dite.

Sql dynamique est très pratique pour le programmeur, attention toutefois au temps de traitement qui sera plus long qu'en sql statique (le compilateur ne connait ni les zones, ni le fichier au moment de la compilation). Il faut donc utiliser cette technique, quand le chargement statique n'est pas possible ou ingérable au niveau codding.

Si on propose un écran de sélection, comportant de nombreuses zones, l'utilisation de sql dynamique est judicieuse.

Prenons un exemple =>

On présente sur l'écran un sous fichier alimenté par une instruction "SELECT", par exemple "SELECT ZON01, ZON02 FROM FICHIER".

Une touche de fonction (c'est toujours un exemple, il peut aussi s'agir de zones de positionnements...) permet à l'utilisateur d'afficher un écran de sélection comportant 10 zones (ZON01 à ZON05 de type ALPHA et ZON06 à ZON10 de type NUMERIQUE) permettant d'affiner la sélection sur le fichier. *On suppose ici que ZONxx sont des zones de FICHIER et le nom des zones DDS de l'écran.* Quand l'utilisateur valide sa sélection il faut recharger le sous fichier en intégrant le choix de l'utilisateur. Si l'utilisateur choisis de renseigner ZON01, ZON02 et ZON07 on rechargera le sous fichier avec une requête de type "SELECT ZON01, ZON02 FROM FICHIER WHERE ZON01 LIKE 'A%' AND ZON02 = 'O' AND ZON07 > 10,5"

On voit dans cet exemple que la partie where de la requête à été alimenté en prenant en compte la sélection de l'utilisateur.

Le problème de ce type de codage est qu'il est très répétitif :

- il faut vérifier, pour chaque zones de sélection, si une valeur à été saisie.
- il faut pour les zones ALPHA doubler les quotes et détecter la présence éventuelle du caractère %.
- Pour les zones numérique, il faut assurer un formatage de la zone.

Exemple de codage classique :

```
W_WHERE = *blank ; // Traitements zones alphas if ZON01 <> *blank ;  // Doubler les cotes   // Détecter le caractère % (w_like = *on)  if W_WHERE = *blank ;   W_WHERE = 'WHERE' ;
 else ;   W_WHERE = %trim(W_WHERE) + ' and' ;  endif ;  W_WHERE = %trim(W_WHERE) + ' ZON01' ;  if w_like = *on ;   W_WHERE = %trim(W_WHERE) + ' LIKE' ;   else ;   W_WHERE =
%trim(W_WHERE) + ' =' ;   endif ;  W_WHERE = %trim(W_WHERE) + ' ' + ZON01 ; endif ; // il faut répéter l'opération pour toutes les zones ALPHA !!! //... // Traitement zones numériques if
ZON05 <> 0 ;  if W_WHERE = *blank ;   W_WHERE = 'WHERE' ;   else ;   W_WHERE = %trim(W_WHERE) + ' and' ;  endif ;   // Formater la zone numérique dans w_val         //(supprimer les
0 non significatis, gestion du signe...)  W_WHERE = %trim(W_WHERE) + ' > ' + w_val   endif // il faut répéter l'opération pour toutes les zones numériques !!! //...
```

Comme on le constate, une grosse partie du code est redondante, le programme de service SGSRVSQ va permettre

d'optimiser le codage.

Exemple de codage en utilisant le programme de service :

Avec SGSRVSQ le programmeur n'a pas à se soucier du formatage des zones numériques, de la détection des cotes ou du caractère "%" pour les zones ALPHA, le programme de service s'en charge. Il faut transmettre la zone saisie à l'écran, la zone du fichier correspondante, l'opérateur ainsi qu'un pointeur initialisé à l'adresse de la variable contenant la partie "WHERE" de la requête.

```
* Prototype pgm de service SRVSQL /COPY SERGE/PROTOTYPE,SGSRVSQ DPtr_Parm S * INZ(%ADDR(W_WHERE)) /free // Alimentation de la clause "WHERE" W_WHERE = *BLANK ; // gestion des zones alphas callp SGWHERA(ZON01 :'ZON01':Ptr_parm :'=') ; callp SGWHERA(ZON02 :'ZON02':Ptr_parm :'=') ; callp SGWHERA(ZON03 :'ZON03':Ptr_parm :'=') ; callp SGWHERA(ZON04 :'ZON04':Ptr_parm :'=') ; callp SGWHERA(ZON05 :'ZON05':Ptr_parm :'=') ; // gestion des zones numériques callp SGWHERN(ZON06 :'ZON06':Ptr_parm :'=') ; callp SGWHERN(ZON07 :'ZON07':Ptr_parm :'>=') ; callp SGWHERN(ZON08 :'ZON08':Ptr_parm :'=') ; callp SGWHERN(ZON09 :'ZON09':Ptr_parm :'=') ; callp SGWHERN(ZON10 :'ZON10':Ptr_parm :'=') ;
```

Le code est plus compact et facile à mettre à jour.

Voir un exemple d'utilisation complet de SGSRVSQ

Description technique de SGSRVSQ :

SGSRVSQ contient 2 procédures exportées SGWHERA pour gérer les saisies alpha et SGWHERN pour les saisies numériques.

Fonctionnalités :
— SGSRVSQ permet de compléter ou initialiser la partie WHERE d'une requête SQL.
— A chaque appel la variable contenant la clause WHERE de la requête se voit compléter avec une nouvelle valeur.
— Pour les zones alpha, les cotes (') seront doublées ('') et le caractère % détecté pour affecter « LIKE » comme opérateur pour la valeur en cours.
— Formatages des zones numériques en prenant en compte le signe (-xxx,xx) et en supprimant les zéros non significatifs.

Description des paramètres :

Les 2 procédures ont le même nombre de paramètres, seul le 1er paramètre a un type différent :
— P_ValEcr (132A ou 30P9) contient la valeur saisie par l'utilisateur, il s'agit soit d'une chaine de caractères pour SGWHERA, soit d'un nombre pour SGWHERN.
— Ptr_Whe (*) est un pointeur qui recouvre la variable ALPHA représentant la clause WHERE. Cela permet donc de mettre à jour la variable ALPHA contenant la clause WHERE dans le programme appelant. Vous pouvez lire l'article sur le passage de paramètres par pointeurs.
— P_SqlNam (255A) est le nom de la zone à l'intérieur de la requête, il s'agit la plupart du temps d'un nom de zone du fichier, mais cela peut être aussi une expression (par ex SUBSTR(ZONE, 1, 10).
— P_Sql_Opr (2A) correspond à l'opérateur à utiliser ('=' ou '<' ou '>' ou '<=' ou '>='), dans le cas d'une zone alpha, si P_Val_Ecr contient le caractère '%', l'opérateur est forcé à "LIKE".

Code source du programme de service SGSRVSQ

Source du liage (BIB/QSRVSRC) *Si le fichier source QSRVSRC n'existe pas il faut le créer.*

```
STRPGMEXP PGMLVL(*CURRENT) SIGNATURE('SGSRVSQ_V01')   EXPORT SYMBOL("SGWHERA")   EXPORT SYMBOL("SGWHERN") ENDPGMEXP
```

Prototype

Placez le prototype de SGSRVSQ dans le fichier source réservé à vos prototypes.

```
 D SGWHERA         PR      D  P_ValEcr                132A   VALUE      D  P_SqlNam                255A   VALUE      D  Ptr_Whe                  *   VALUE      D  P_SqlOpr                  2A
VALUE      D*----------------------------------------------------      DSGWHERN          PR      D P_ValEcr                 30P 9 VALUE      D P_SqlNam                 255    VALUE      D Ptr_whe
    *   VALUE      D P_SqlOpr                  2    VALUE
```

Code source de SGSRVSQ

Pensez à changer la directive copy

```
 *-------------------------------------------------------------------------------      * ® GOMES serge  Programme de service Traitement WHERE de requête SQL      *    CRTRPGMOD
MODULE(LIB/SGSRVSQ)      *    CRTSRVPGM SRVPGM(LIB/SGSRVSQ)      *-------------------------------------------------------------------------------      H NoMain  decedit(',')      H
COPYRIGHT('Serge GOMES')      /COPY SERGE/PROTOTYPE,SGSRVSQ          *-------------------------------------------------------------------------------      * Procedure name : SGWHERA      *
Purpose :       Alimente la partie WHERE d'une requête SQL à partir...      *                    d'une valeur ALPHA      * Returns :      * Parameter :      P_ValEcr => Valeur ALPHA transmise
(zone positionne...      *                    ment écran)      * Parameter :      P_SqlNam => Nom zone SQL (ex "SUBSTR(ZONE 1 10")      * Parameter :      Ptr_Whe => Pointeur sur la clause
WHERE (la clause ...      *                    WHERE du pgm appelant doit être " baséé" s...      *                    ur ce pointeur      * Parameter :      P_SqlOpr => Opérateur à appliquer si '%'
non détecté      *----------------------------------------------------      P SGWHERA         B                  EXPORT      D SGWHERA         PI      D  P_ValEcr                132A   VALUE      D  P_SqlNam
          255A   VALUE      D  Ptr_Whe                  *   VALUE      D  P_SqlOpr                  2A   VALUE      D                DS      DW_ValEcr                          LIKE(P_ValEcr)      DW_TabEcr
               1    DIM(132) OVERLAY(W_ValEcr:1)      D                DS      DW_ValRes                  255     DW_TabRes                  1    DIM(255) OVERLAY(W_ValRes:1)      DWK
 C                  ''''      DWS               C                  '%'      DWlike            S                N   INZ(*OFF)      DWC               S              1A   INZ      DI                S              3P 0      DJ                S
3P 0 INZ      DP_Rqt_Whe      S          32740    varying based(Ptr_whe)      /FREE      W_ValEcr = P_ValEcr ;      If W_ValEcr <> *blank ;      If P_Rqt_Whe = *blank ;      P_Rqt_Whe
= 'WHERE ' + %trim(P_SqlNam) ;      Else ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) + ' AND ' +      %trim(P_SqlNam) ;      EndIf ;      // Initialisation de W_TabRes
W_ValRes = *blank ;      For i = 1 to 132 ;      WC= W_TabEcr(i) ;      SELECT ;      When WC = WK ;      J=J+1 ;      W_TabRes(J) = WK ;      J=J+1 ;
W_TabRes(J) = WK ;      When WC = WS ;      J=J+1 ;      W_TabRes(J) = WC ;      Wlike = *ON ;      OTHER ;      J=J+1 ;      W_TabRes(J) = WC ;
ENDSL ;      EndFor ;      If Wlike ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) + ' LIKE' ;      Else ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ' ' + %trim(P_SqlOpr) ;      EndIf
;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ' ' + WK + %trim(W_ValRes) + WK ;      EndIf ;      P_Rqt_Whe = %trim(P_Rqt_Whe) ;      Return ;      /END-FREE      PSGWHERA
 E      PSGWHERN         B                  EXPORT      D                PI      D P_ValEcr                 30P 9 VALUE      D P_SqlNam                 255    VALUE      D Ptr_whe                  *
VALUE      D P_SqlOpr                  2    VALUE      D                DS      DWchain                    30    INZ      DWN9                       9  0 OVERLAY(Wchain:22)      D                DS
DWchain1                    30    INZ      DWENT                      21    OVERLAY(Wchain1:1)      DWDECI                     9    OVERLAY(Wchain1:22)      DW_ValEcr         S
LIKE(P_ValEcr)      DI                S              3P 0 INZ(1)      DWC               S              1A   INZ      DP_Rqt_Whe      S          32740    varying based(Ptr_whe)      /FREE      W_ValEcr =
P_ValEcr ;      If W_ValEcr <> 0 ;      If P_Rqt_Whe = *blank ;      P_Rqt_Whe = 'WHERE ' + %trim(P_SqlNam) ;      Else ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) + ' AND ' +
    %trim(P_SqlNam) ;      EndIf ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ' ' + %trim(P_SqlOpr) ;      If W_ValEcr < 0 ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ' -' ;
       W_ValEcr = -W_ValEcr ;      EndIf ;      Wchain=%trim(%editc(W_ValEcr :'X')) ;      Wchain1 = Wchain ;      // Suppression des 0 non significatifs      I = 1 ;      WC =
%subst(Wchain1:I:1) ;      Dow (%subst(Wchain1:I:1) = '0' OR      %subst(Wchain1:I:1) = *blank) AND      I <= %len(WENT) - 1 ;      %subst(Wchain1:I:1) = *blank ;
I=I+1 ;      EndDo ;      I=30 ;      Dow (%subst(Wchain1:I:1) = '0' OR      %subst(Wchain1:I:1) = *blank) AND      I>%len(WENT) ;      %subst(Wchain1:I:1) = *blank ;
 I=I-1 ;      EndDo ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ' ' + %trim(WENT) ;      If WN9 > 0 ;      P_Rqt_Whe  = %trim(P_Rqt_Whe) +      ',' + %trim(WDECI) ;
EndIf ;      EndIf ;      P_Rqt_Whe = %trim(P_Rqt_Whe) ;      Return ;      /END-FREE      PSGWHERN      E
```