Power Week

Université IBM i 2019

22 et 23 mai

IBM Client Center Paris

S49 – SQLRPG fonctions avancées

Nathanaël Bonnet
Gaia
nathanael.bonnet@gaia.fr

Gaia

- Conseil et formation IBM i depuis 1995
  - Inter et intra entreprise
- Base de connaissance en ligne
  - https://know400.gaia.fr
- Centre de services
  - TMA / TME sur mesure
  - Outillage spécifique

GAIA

https://www.gaia.fr

https://twitter.com/GaiaFrance

COMMON FRANCE

SQL dans RPG : mais pourquoi ?

- Pourquoi
  - Pour profiter pleinement de toute la puissance du langage RPG et du langage SQL !
  - Le RPG évolue, mais SQL est enrichie de nombreuses fonctionnalités à chaque TR ou version
    - Des registres
    - Des fonctions (manipulation des dates, expression régulière, encodage/décodage des URL, base64 …)
    - Des fonctions d'administration : services sous formes de tables, vues, fonctions ou procédures

```
exec sql
  set :msg = varchar_format(current date - 1 month + 2 days, 'Day, DD month YYYY');
dsply msg ;
```

```
DSPLY  Dimanche, 21 avril 2019
```

Expression régulière

- Par exemple tester la validité d'une adresse mail

```
dcl-s mail varchar(50) inz('nathanael.bonnet@gaia.fr') ;
dcl-s valide char(1) inz('N') ;

exec sql
  SELECT 'O'
    into :valide
    FROM sysibm.sysdummy1
    WHERE REGEXP_LIKE(:mail,
                      '^[a-z0-9._-]+@[a-z0-9._-]{2,}.[a-z]{2,4}$',
                      'i');
if valide = 'O' ;
  // -- traiter le mail
else ;
  // -- logger l'erreur
endif ;
```

Power Week

Université IBM i

22 et 23 mai 2019

Options de compilation

IBM

Possibilités

- CRTSQLRPGI : Une seule commande pour
  - *PGM, *MODULE ou *SRVPGM
    - Astuce RDi : vous pouvez créer plusieurs commandes de compilation correspondant à chaque type
- « carte » H / CTL-OPT
  - Pour les options RPG
- SET OPTION
  - Pour les options SQL
  - Doit être la 1ère instruction SQL dans l'ordre du source
  - Non exécutable (ne pas tester SQLSTATE/SQLCODE !), c'est une directive de compilation

Paramètres de compilation (1/2)

- Vous avez des paramètres de compilation spécifiques à SQL
  - **INCFILE** et **INCDIR** qui précisent d'où proviennent vos include SQL
  - **CLOSQLCSR** qui précise la portée de vos curseurs (groupe d'activation ou module)
  - **DATFMT, DATSEQ, TIMFMT, TIMSEQ** pour les formats de date et d'heure
  - **COMMIT** pour le contrôle de validation
  - **DYNUSRPRF** User pour les requêtes dynamiques
  - Etc…

Paramètres de compilation (2/2)

- Vous pouvez indiquer ces paramètres soit
  - Lorsque vous compilez

```
CRTSQLRPGI … COMMIT(*NONE) DATFMT(*ISO)
```

  - Dans le programme pour éviter de les oublier

```
C/Exec SQL
C+ Set Option
       Commit=*NONE,
       DatFmt=*ISO,
       DynUsrPrf=*Owner,
       DlyPrp=*YES
C/End-Exec
```

Principales options

- **SET OPTION**
  - **ALWCPYDTA** = Copie des données autorisées
    - ***OPTIMIZE** si cela permet d'aller plus vite
    - ***YES** copie autorisée lorsque nécessaire
    - ***NO** copie non autorisée
  - **CLOSQLCSR** = Fermeture implicite des curseurs SQL
    - ***ENDACTPGRP** Fin du groupe d'activation
    - ***ENDMOD** Fin du module
    - ***ENDPGM** Fin du programme
    - ***ENDSQL** Fin du 1er programme SQL dans la pile d'appel
    - ***ENDJOB** Fin du travail
  - **COMMIT** = Niveau de transaction implicite
    - ***CHG** Uncommitted Read
    - ***NONE** Pas de contrôle transactionnel implicite
    - ***CS** Cusror stability
    - ***ALL** Read stability
    - ***RR** Repeatable read

Principales options

- **COMPILEOPT** = Options de compilation (paramètre COMPILEOPT de CRTSQLRPGI)
  - ***NONE** Aucune option
  - **Valeur** Constante jusqu'à 5.000 caractères
- **DATFMT** = Format de date
  - ***JOB** Celui du travail (cf **DSPJOB**)
  - ***ISO** Format **yyyy-mm-dd**
  - ***EUR** Format **dd.mm.yyyy**
  - ***USA** Format **mm/dd/yyyy**
  - ***JIS** Format **yyyy-mm-dd**
  - ***MDY** Format **mm/dd/yy**
  - ***DMY** Format **dd/mm/yy**
  - ***YMD** Format **yy/mm/dd**
  - ***JUL** Format **yy/ddd**
- **DATSEP** = Séparateur de date
  - ***JOB** Celui du travail (cf **DSPJOB**)
  - ***SLASH** ou **'/'**
  - ***PERIOD** ou **'.'**
  - ***COMA** ou **','**
  - ***DASH** ou **'-'**
  - ***BLANK** ou **' '**

Principales options

- **DBGVIEW** = Vue de débogage
  - ***NONE** Aucune
  - ***SOURCE** Vue source
  - ***STMT** Vue instruction
  - ***LIST** Vue listing
- **DFTRDBCOL** = Base de données par défaut pour les éléments non qualifiés en SQL statique
  - ***NONE** Aucune
  - **Nom** Nom du schéma
- **DYNDFTCOL** = **DFTRDBCOL** pour SQL dynamique
  - ***NO** Ne pas utiliser **DFTRDBCOL** pour le SQL dynamique
  - ***YES** Utiliser **DFTRDBCOL** pour le SQL dynamique
- **DYNUSRPRF** = Utilisateur en cours pour SQL dynamique
  - ***USER** Utilisateur en cours du travail
  - ***OWNER** Propriétaire de l'objet *PGM en cours d'exécution
- **NAMING** = Convention d'appellation
  - ***SYS** Convention système **'/'**
  - ***SQL** Convention système **'.'**

Principales options

- **SQLPATH** = SQL Path (liste de bibliothèques pour SQL) pour routines dans du code SQL statique
  - ***LIBL** Utiliser la liste de bibliothèques du travail
  - **Valeur** Liste de bibliothèques séparées par des ,
- **TGTRLS** = Version cible
  - **VxRxMx** Version spécifique
- **TIMFMT** = Format d'heure
  - ***HMS** Format **hh:mm:ss**
  - ***ISO** Format **hh.mm.ss**
  - ***EUR** Format **hh.mm.ss**
  - ***USA** Format **hh:mm xx** avec **xx** = **AM** ou **PM**
  - ***JIS** Format **hh:mm:ss**
- **TIMSEP** = Séparateur d'heure
  - ***JOB** Celui du travail (cf **DSPJOB**)
  - ***COLON** ou **':'**
  - ***PERIOD** ou **'.'**
  - ***COMMA** ou **','**
  - ***BLANK** ou **' '**

Principales options

- **USRPRF** = Profil utilisateur pour l'exécution des instructions SQL statiques
  - ***USER** Utilisateur en cours
  - ***OWNER** Le propriétaire du programme (+ l'utilisateur en cours)
  - ***NAMING** En fonction de la convention
    - ***SQL -> USRPRF(*OWNER)**
    - ***SYS -> USRPRF(*USER)**

Particularités syntaxiques

- Commentaires
  - RPG
    - Fixe ***** (colonne 7)
    - Libre **//** ou **/* … */**
  - SQL
    - **-- commentaire ici**
- Inclusion d'un autre membre
  - RPG
    - **/include**
  - SQL
    - **C/EXEC SQL INCLUDE member-name**
    - **C/END-EXEC**
  - Pour les /include RPG imbriqués
    - **CRTSQLRPGI … RPGPPOPT(*LVL2)**
    - Valeurs possibles : ***NONE**, ***LVL1** et ***LVL2**

Power Week

Université IBM i

22 et 23 mai 2019

Outils

IBM

De multiples outils possibles

- RDi
  - Pour la programmation
  - Coloration, (auto)formatage
- Access Client Solutions
  - Exécution de scripts SQL
  - Visual Explain
  - Schéma et Outils de performance SQL
  - Cache des plans, génération des scripts SQL depuis les objets, …
- Lien entre RDi et ACS
  - Ouverture d'une instruction SQL sélectionnée dans RDi avec ACS
- D'autres produits
  - IBM Data Studio
    - Peut s'intègrer dans RDi
  - SqlDbx, Squirrel SQL, …. N'importe quel outil capable de communiquer avec DB2 (pilots JDBC, .Net …)

Power Week

Université IBM i

22 et 23 mai 2019

Gestion des erreurs

IBM

SQLCA

- **SQLCA** = Zone de communication SQL
  - C'est une data structure particulière qui est générée lors de la pré-compilation
  - Pour le RPG, elle est générée automatiquement
  - Pour le COBOL, il faut la déclarer dans le source
    - **`EXEC SQL`**
    - **`INCLUDE SQLCA`**
    - **`END-EXEC`**
  - C'est une zone dans laquelle SQL renvoie des informations relatives à l'exécution de chaque instruction SQL
  - Deux zones importantes dans cette DS:
    - **SQLCODE** ou **SQLCOD**
    - **SQLSTATE** ou **SQLSTT**

Contenu

```
D SQLCA           DS
D  SQLCAID                       8A   INZ(X'0000000000000000')
D  SQLAID                        8A   OVERLAY(SQLCAID)
D  SQLCABC                      10I 0
D  SQLABC                        9B 0 OVERLAY(SQLCABC)
D  SQLCODE                      10I 0
D  SQLCOD                        9B 0 OVERLAY(SQLCODE)
D  SQLERRML                      5I 0
D  SQLERL                        4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                     70A
D  SQLERM                       70A   OVERLAY(SQLERRMC)
D  SQLERRP                       8A
D  SQLERP                        8A   OVERLAY(SQLERRP)
D  SQLERR                       24A
D   SQLER1                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                     10I 0 DIM(6)  OVERLAY(SQLERR)
```

Présentation

```
D  SQLWRN                       11A
D   SQLWN0                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                       1A   OVERLAY(SQLWRN:*NEXT)
D  SQLWARN                       1A   DIM(11) OVERLAY(SQLWRN)
D  SQLSTATE                      5A
D  SQLSTT                        5A   OVERLAY(SQLSTATE)
D*  END OF SQLCA
```

SQLCODE

- La principale zone à connaître est la zone **SQLCODE** ou **SQLCOD** qui indique l'état de la dernière instruction SQL exécutée
  - **SQLCODE=0**
    - Exécution réussie de la requête
  - **SQLCODE=100**
    - Enregistrement non trouvé ou fin de fichier (pas de données dans l'ordre Fetch..)
  - **SQLCODE>0 and not=100**
    - Exécution ok avec avertissement (zones tronquées, valeurs nulles, etc.)
  - **SQLCODE<0**
    - Erreur grave dans l'ordre SQL

SQLSTATE(1/2)

- Le code d'état **SQLSTATE** est émis à chaque opération **SQL**
  - A un code **SQLSTATE** peuvent correspondre plusieurs codes **SQLCODE**
  - **SQLSTATE** définie des classes: Les principales
    - Code classe 00 : Exécution réussie

SQLSTATE	Signification	SQLCODE
00000	Exécution réussie	+000

    - Code classe 01: Avertissement

SQLSTATE	Signification	SQLCODE
01002	Erreur de déconnexion	+596
01004	Chaine tronquée	+000, +445
01006	Droit non révoqué	+569
// etc…		

SQLSTATE(2/2)

- Code classe 02 : Pas de données

SQLSTATE	Signification	SQLCODE
02000	Enregistrement non trouvé ou fin de fichier (Fetch non valide)	+100
02001	Pas d'autres résultats à envoyer	+387

- Code classe 07: Erreur SQL Dynamique

SQLSTATE	Signification	SQLCODE
07001	Nombre de variables hôtes différents que celui des variables indicateurs	-313
// etc…		

SQLERR*

- **SQLERRMC**
  - Texte du message SQL d'erreur si SQLCODE < 0
- **SQLERRML**
  - longueur significative de SQLERRMC
- **SQLERRD** ou **SQLERR** (ERREUR)
  - **SQLERRD(1)** ou **SQLER1** contient le N° message CPF si SQLCODE < à 0
  - **SQLERRD(2)** ou **SQLER2** contient le N° message CPD si SQLCODE < à 0
  - **SQLERRD(3)** ou **SQLER3** donne le nb de lignes affectées par un ordre UPDATE,DELETE, ou INSERT

SQLWARN*

- **SQLWARN** ou **SQLWRN** (WARNING)
  - Tableau de 8 caractères contenants `'  '` ou `'W'` (Warning)
  - **SQLWARN0** ou **SQLWN0** si une des valeurs **SQLWARN*** contient `'W'`
  - **SQLWARN1** ou **SQLWN1** contient `'W'` si une colonne a été tronquée
  - **SQLWARN3** ou **SQLWN3** contient `'W'` si le nb de variables HOST est invalide
  - **SQLWARN4** ou **SQLWN4** contient `'W'` si un ordre PREPARE pour UPDATE ou DELETE ne contient pas la clause WHERE

GET DIAGNOSTICS

- GET DIAGNOSTICS est une instruction SQL permettant de retrouver plus d'informations sur une erreur (condition), et ce plus facilement !

```
GET DIAGNOSTICS var1 = info, var2 = info … ;
```

- Les informations disponibles sont regroupées en 3 catégories
  - Informations sur l'instruction
  - Informations sur la connexion
  - Informations sur l'erreur
  - https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/db2/rbafzgetdiag.htm
- Attention
  - C'est une instruction SQL : cela modifie donc SQLCA/SQLCODE/SQLSTATE et les informations suivantes qui seraient à nouveau renvoyées par GET DIAGNOSTICS

GET DIAGNOSTICS

- Exemple
  - Connaitre le nombre d'enregistrements supprimés

```
dcl-s nbDel uns(10) ;

exec sql delete from nb.client where numclient > 100 ;
exec sql
  get diagnostics :nbDel = ROW_COUNT ;

dsply ( 'N lignes supprimées : ' + %char( nbDel ) ) ;
```

```
DSPLY  N lignes supprimées : 4
```

GET DIAGNOSTICS

- Retrouver la contrainte en erreur

```
CREATE TABLE NB.CLIENT19 (
  ID INTEGER NOT NULL NOT HIDDEN ,
  REFEXT CHARACTER(25) NOT HIDDEN ,
  NOM CHARACTER(25) NOT NULL NOT HIDDEN ,
  ETAT CHARACTER(1) DEFAULT 'O' NOT NULL NOT HIDDEN ,
  CONSTRAINT NB.CLIENT_PK PRIMARY KEY (ID) ,
  CONSTRAINT NB.CLIENT_CC_ETAT CHECK ( ETAT IN ('O', 'N'))
) ;
```

```
4 > call univ19_3
    Violation de la contrainte de vérification sur le membre CLIENT19
  ?
    Violation de la contrainte de vérification sur le membre CLIENT19
  ?
    INSERT, UPDATE ou MERGE non admis pour une contrainte CHECK.
    DSPLY  Err :contrainte CLIENT_CC_ETAT
```

```
dcl-s SqlStatePrv like(SqlState) ;
dcl-s contrainte char(128) ;
dcl-s msg char(52) ;

exec sql
  insert into nb.client19
    values (2, 'ref', 'Gaia', 'X')
    with nc;
select ;
when %subst(SqlState:1:2) = '00' or     // OK
     %subst(SqlState:1:2) = '01' ;      // Warning
  dsply 'insertion OK' ;
when %subst(SqlState:1:2) = '23' or     // Integrity Constraint Violation
     %subst(SqlState:1:2) = '27' ;      // Triggered Data Change Violation
  // conserver SQLSTATE précédent
  SqlStatePrv = SqlState ;
  exec sql
    get diagnostics condition 1 :contrainte = CONSTRAINT_NAME ;
  msg = 'Err :contrainte ' + contrainte ;
  dsply msg ;
endsl ;
```

```
La valeur indiquée est incorrecte car elle produirait une clé en double.
DSPLY  Err :contrainte CLIENT_PK
```

Power Week

Université IBM i

22 et 23 mai 2019

SELECT et INSERT multiples

IBM

SELECT multiples

- Via une DS DIM(xx)
  - Permet de meilleurs performance
  - Problème : combien de lignes avez-vous réellement lues ?

```
dcl-ds clients extname('CLIENT19') dim(100) qualified inz end-ds ;
dcl-s  clientsNb  uns(3) ;

exec sql
  declare c_clients cursor for
    select * from client19 ;
exec sql   open c_clients ;
exec sql
  fetch from c_clients for 100 rows into :clients ;

// Nombre de lignes réellement lues
clientsNb = SQLERRD(3) ;
exec sql
  get diagnostics :clientsNb = ROW_COUNT ;

exec sql
  close c_clients ;
```

INSERT MULTIPLES

- Toujours via une DS DIM(XX)
  - C'est vous qui indiquez le nombre de postes à insérer

```
dcl-ds clients extname('CLIENT19') dim(100) qualified inz end-ds ;
dcl-s  clientsNb  uns(3) ;

// Alimentaion des clients dans la DS
clients(1).ID = 10 ;
clients(1).REFEXT = 'Ref 10' ;
clients(1).NOM = 'Nom 10' ;
clients(1).ETAT = 'O' ;
clients(2).ID = 20 ;
clients(2).REFEXT = 'Ref 20' ;
clients(2).NOM = 'Nom 20' ;
clients(2).ETAT = 'O' ;
clientsNb = 2 ;

exec sql
  insert into client19 :clientsNb rows
    values (:clients) ;
```

Cas d’usage

- Peut simplifier certains traitements
- Alimentation de sous-fichiers
  - Lire le nombre d’enregistrements pour une page
- Problématiques liées aux performances
- En cas d’erreur, isoler l’enregistrement problématique est plus difficile

Power Week

Université IBM i

22 et 23 mai 2019

Valeurs nulles

IBM

Valeur nulle

- Les valeurs nulles sont directement supportées dans les tables
- Mais peuvent également être calculées
  - Jointure

- Il est possible de s'en passer
  - COALESCE(val1, val2, …)
  - Cela peut également poser d'autres problèmes
- Ou de les gérer

Gestion des valeurs nulles

- Malheureusement, avec SQL dans le RPG
  - Impossible d'utiliser %nullind
- SQL fournit un « indicateur » de nullité
  - À déclarer sous forme d'entier binaire signé sur 2 octets

Exemple : zone à zone

```
dcl-s id int(10) ;
dcl-s refext char(25) ;
dcl-s refextNull int(5) ;

exec sql
  declare c_client cursor for
    select id, refext
      from client19 ;
exec sql
  open c_client ;

exec sql
  fetch c_client into :id, :refext:refextNull ;
dow sqlcode >=0 and SqlCode < 100 ;
  // Référence externe nulle ?
  if refextNull = -1 ;
    dsply ( %char(id) + ', null' ) ;
  else ;
    dsply ( %char(id) + ', ' + refext ) ;
  endif ;
  // enreg suivant
  exec sql
    fetch c_client into :id, :refext:refextNull ;
```

```
4 > call univ19_8
    DSPLY  1, G2019
    DSPLY  10, Ref 10
    DSPLY  20, Ref 20
    DSPLY  101, null
    DSPLY  110, null
    DSPLY  120, null
```

Exemple : DS

```
dcl-ds client extname('CLIENT19') qualified end-ds  ;
dcl-s  clientNull int(5) dim(4) ;     // 4 colonnes dans CLIENT19

exec sql
  declare c_client cursor for
    select *
      from client19 ;
exec sql
  open c_client ;

exec sql
  fetch c_client into :client:clientNull ;
dow sqlcode >=0 and SqlCode < 100 ;
  // Référence externe nulle ?
  if clientNull(2) = -1 ;                // <<-- rang de la colonne
    dsply ( %char(client.id) + ', null' ) ;
  else ;
    dsply ( %char(client.id) + ', ' + client.refext ) ;
  endif ;
  // enreg suivant
  exec sql
    fetch c_client into :client:clientNull ;
enddo ;
```

```
4 > call univ19_9
    DSPLY  1, G2019
    DSPLY  10, Ref 10
    DSPLY  20, Ref 20
    DSPLY  101, null
    DSPLY  110, null
    DSPLY  120, null
```

Exemple : DS DIM

```
dcl-c MAX_CLIENTS const(100) ;

dcl-ds client     extname('CLIENT19') qualified dim(MAX_CLIENTS) end-ds  ;
dcl-ds clientNull dim(MAX_CLIENTS)    qualified ;
  nullInd int(5) dim(4) ;  // 4 colonnes dans CLIENT19
end-ds;

dcl-s i int(3) ;

...

exec sql
  fetch c_client for 15 rows into :client:clientNull ;
if sqlcode >=0 and SqlCode < 100 ;

  for i = 1 to Sqlerrd(3) ;
    // Référence externe nulle ?
    if clientNull(i).nullInd(2) = -1 ;        // <<-- rang de la colonne
      dsply ( %char(client(i).id) + ', null' ) ;
    else ;
      dsply ( %char(client(i).id) + ', ' + client(i).refext ) ;
    endif ;
  endfor ;

endif ;
```

```
4 > call univ19_10
    DSPLY  1, G2019
    DSPLY  10, Ref 10
    DSPLY  20, Ref 20
    DSPLY  101, null
    DSPLY  110, null
    DSPLY  120, null
```

Exemple : DS DIM, « indicateurs » nommés

```
dcl-ds client     extname('CLIENT19') qualified dim(MAX_CLIENTS) end-ds  ;
dcl-ds clientNull dim(MAX_CLIENTS)    qualified ;
  nullInd int(5) dim(4) ;  // 4 colonnes dans CLIENT19
end-ds;
dcl-ds nomNull dim(MAX_CLIENTS) qualified based(ptrClientNull);
  idNull  int(5)  ;
  refextNull  int(5)  ;
  nomNull  int(5)  ;
  etatNull  int(5) ;
end-ds;


ptrClientNull = %addr(clientNull) ;

exec sql
  fetch c_client for 15 rows into :client:clientNull ;
if sqlcode >=0 and SqlCode < 100 ;

  for i = 1 to Sqlerrd(3) ;
    // Référence externe nulle ?
    if nomNull(i).refextNull = -1 ;           // <<-- nom de la colonne
      dsply ( %char(client(i).id) + ', null' ) ;
    else ;
      dsply ( %char(client(i).id) + ', ' + client(i).refext ) ;
    endif ;
  endfor ;

endif ;
```

```
4 > call univ19_11
    DSPLY  1, G2019
    DSPLY  10, Ref 10
    DSPLY  20, Ref 20
    DSPLY  101, null
    DSPLY  110, null
    DSPLY  120, null
```

Power Week

Université IBM i

22 et 23 mai 2019

Accéder à l'IFS

IBM

LOB – Large OBject

- En SQL, il existe des types CLOB et BLOB
  - Caractère et binaire
  - Permettant de manipuler les données correspondantes
    - Exemple : pdf, photos directement dans la base de données
  - Non supporté en RPG

- RPG supporte la déclaration via
  - SQLTYPE(CLOB : taille)

```
dcl-s clob          sqltype( CLOB : 16700000 ) inz ;
```

  - Le compilateur déclare une DS RPG

```
//*DCL-S CLOB      SQLTYPE( CLOB : 16700000 ) INZ ;
DCL-DS CLOB;
  CLOB_LEN UNS(10);
  CLOB_DATA CHAR(16700000) CCSID(*JOBRUNMIX);
END-DS FILECLOB;
```

Lire un fichier

```
// Variables
dcl-s data          varchar(16700000)          inz ;
dcl-s clob          sqltype( CLOB : 16700000 ) inz ;
dcl-s isErr         ind                        inz ;

// GET_CLOB_FILE doit se faire sous contrôle transactionnel
exec sql
  set transaction isolation level CS ;
// accès au fichier
exec sql
  values get_clob_from_file('/home/NB/perimetre.csv', 1) into :clob ;
isErr = (%subst(sqlstate:1:2) <> '00' and %subst(sqlstate:1:2) <> '01') ;
// fin de la transaction
exec sql
  commit ;

// Données
if isErr ;
  dsply 'Erreur accès fichier' ;
else ;
  data = %subst(clob_data : 1 : clob_len ) ;
endif ;
```

Lire un fichier

- Attention
  - Encodage
  - Retour à la ligne (CRLF / LFCR / LF / CR)
  - Tabulation
  - Pour l'UTF-8 : avec / sans BOM
  - …
- Bref, à vous de traiter dans le programme RPG

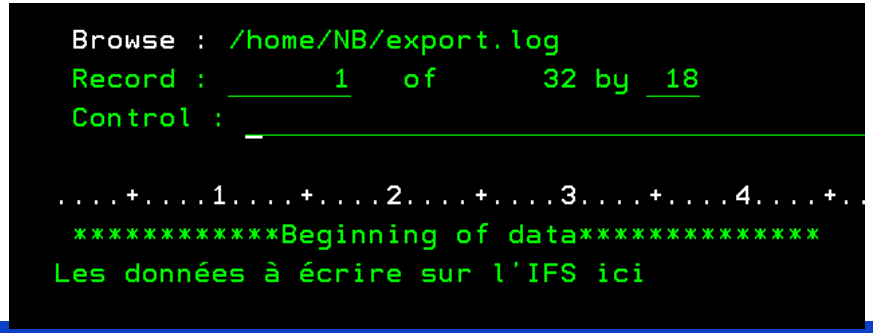
Ecrire un fichier

- Les définitions CLOB_FILE et BLOB_FILE sont également supportées

```
dcl-s fileName     char(128)   inz('/home/NB/export.log')  ;
dcl-s fileData     char(4096)  ;
dcl-s file         SQLTYPE(CLOB_FILE);

file_name = %trim( fileName ) ;
file_nl = %len( %trim( fileName ) ) ;
file_fo = SQFCRT ;   // création

fileData = 'Les données à écrire sur l''IFS ici' ;
exec sql
  values :fileData
    into :file;
if ( SqlCode < 0 or       // échec
     SqlCode = 100 ) ;    // pas de données
   dsply 'ERR' ;
endif ;
```



Ecrire un fichier

- A la compilation

```
//*DCL-S FILE          SQLTYPE(CLOB_FILE);
       DCL-DS FILE;
         FILE_NL UNS(10);
         FILE_DL UNS(10);
         FILE_FO UNS(10);
         FILE_NAME CHAR(255) CCSID(*JOBRUNMIX);
       END-DS FILE;
```

  - Valeurs possibles pour FILE_FO
    - SQFRD (2) : lecture seule
    - SQFCRT (8) : création (ne doit pas exister)
    - SQFOVR (16) : création ou remplacement
    - SQFAPP (32) : création ou ajout

Plus de possibilités

- D’autres types disponibles
  - XML_CLOB
  - XML_DBCLOB
  - XML_BLOB
  - XML_LOCATOR
  - XML_CLOB_FILE

- Il est possible de lire/écrire sur l’IFS en RPG via les API C
  - Nécessite moins de code avec SQL

Power Week

Université IBM i

22 et 23 mai 2019

Appel de procédure cataloguée/stockée

IBM

Cinématique 1/3

- Depuis la 7.1
  - Possibilité de lire le(s) result sets renvoyé(s) par une procédure cataloguée

- Déclaration d'un **RESULT_SET_LOCATOR**

```
D MyResultSet     s               SQLTYPE(RESULT_SET_LOCATOR)
```

  - Où
    - MyResustSet est une variable de type **RESULT_SET_LOCATOR**

- Appel procédure stockée

```
exec SQL call MYPROC(:Region);
```

  - Où `Region` est une variable transmise en paramètre

Cinématique 2/3

- Associer le **result-set** à un curseur

```
exec SQL associate result set locator (:MyResultSet)
         with procedure MYPROC;
exec SQL allocate C1 cursor for result set
         :MyResultSet;
```

  - Où
    - MyResultSet est la variable de type **RESULT_SET_LOCATOR**
    - C1 est le nom donné au curseur dans le programme RPG

Cinématique 3/3

- Parcours du curseur

```
// Lecture
exec SQL fetch next from C1 into :Row;
// Tant que la lecture est OK
dow %subst(sqlstt:1:2)='00' or
    %subst(sqlstt:1:2)='01';
   // Faire qq chose ici …
   // Lecture suivante
   exec SQL fetch next from C1 into :Row;
enddo;
// Fermeture du curseur
exec SQL close C1;
```

Plusieurs result-set

- Certaines procédures cataloguées renvoient plusieurs result-sets

```
// Déclaration des RESULT_SET_LOCATOR
dcl-s rsClient SQLTYPE(RESULT_SET_LOCATOR) ;
dcl-s rsAdress SQLTYPE(RESULT_SET_LOCATOR) ;

// Appel procédure
exec SQL call cli_adr_listInfo();
// Récupérer des result-sets
exec SQL associate result set locators (:rsClient,:rsAdress)
            with procedure cli_adr_listInfo;
// Associer les resul-sets avec des curseurs internes
exec SQL allocate client_list  cursor for result set :rsClient ;
exec SQL allocate adresse_list cursor for result set :rsAdress ;
```

Plusieurs result-set

- Remarques RPG
  - Les noms donnés aux curseurs sont des noms internes à RPG
  - C'est l'ordre de renvoi des curseurs dans la procédure cataloguée qui importe
- Remarques SQL
  - Pour plus de dynamisme dans les applications, il est possible d'utiliser les instructions SQL
    - **`DESCRIBE PROCEDURE`**
    - **`DESCRIE CURSOR`**
  - Qui permettent de retrouver des informations telles que le nombre de curseurs réellement retournés, l'estimation du nombre de lignes retournées …


MERCI