Power Systems

Modernisation IBM i – Nouveautés 2014-2015

19 et 20 mai 2015 – IBM Client Center, Bois-Colombes

S27 - Contrôle transactionnel en SQL embarqué

Mercredi 20 mai – 13h30-15h00

Pierre-Louis BERTHOIN – Gaia

Agenda

- Historique
- Transaction & journalisation
- Usage(s)
- Outils
- Un mot du PL/SQL (trigger, procédure, fonction)
- Bonnes pratiques

Power Systems

for Business

Historique

IBM

Un peu d'histoire

- Ted Codd a publié en 1979 les bases des SGBR, et parmi les définitions "La contrainte d'intégrité référentielle"
  - Qui consiste à garantir une dépendance des relations de données entre tables
- Exemple
  - Un détail de commande doit avoir un entête
  - Ce contrôle était prévu applicativement, dans les bases de données précédentes

Un peu d'histoire

- En plus du CIR est apparu la notion de « Transaction BD »
- Définition simple
  - Une transaction est un ensemble de mises à jour de tables ayant attrait à une même action, même événement
- Exemple
  - Une commande génèrera
    - Un paiement
    - Une modification de stock
    - Une demande de livraisons
    - …
- On parle
  - De validation quand la transaction est confirmée
  - D'invalidation dans le cas de problème

Un peu d'histoire

- Toutes les bases de données du marché ont implémenté ce mécanisme
- Le principe général est le suivant
  - On mémorise chaque modification d'une ou plusieurs tables dans des journaux (ou log ...)
  - On décide d'un début (point de synchro, de validation) et quand on finit cette transaction on valide ou on invalide
- Remarque
  - Beaucoup d'applications cachent derrières des briques applicatives ce mécanisme

Sur l'IBM i

- Encore AS400 à l'époque
- Pendant longtemps journaliser les bases de données n'était pas la norme
  - Les fichiers PF ne sont pas journalisés par défaut, et on avait confiance en nos applicatifs (le plus souvent maison)
  - L'espace utilisé par les journaux était vu comme un coût non justifié

Sur l'IBM i

- Plusieurs phénomènes ont changé cette donne, en voici deux principaux
  1. La HA : tous les outils de hautes disponibilité sont basés sur la réplication de postes de journaux
  2. La généralisation de l'utilisation de SQL : toutes les bases de données créées par SQL sont journalisées par défaut

Autres utilisations des journaux sur IBM i

- En voici 3 principales
  - La mise en œuvre de HA basée sur la réplication de postes de journaux, le plus souvent avec des journaux distants
  - L'audit de sécurité et d'utilisation des objets
    - Valeurs système **QAUDCTL**, **QAUDLVL**, **QAUDLVL2**
  - L'historique des distributions **SMTP**
    - Journal **QZMF** paramètre dans **CHGSMTPA**

Power Systems

for Business

Transaction et journalisation

IBM

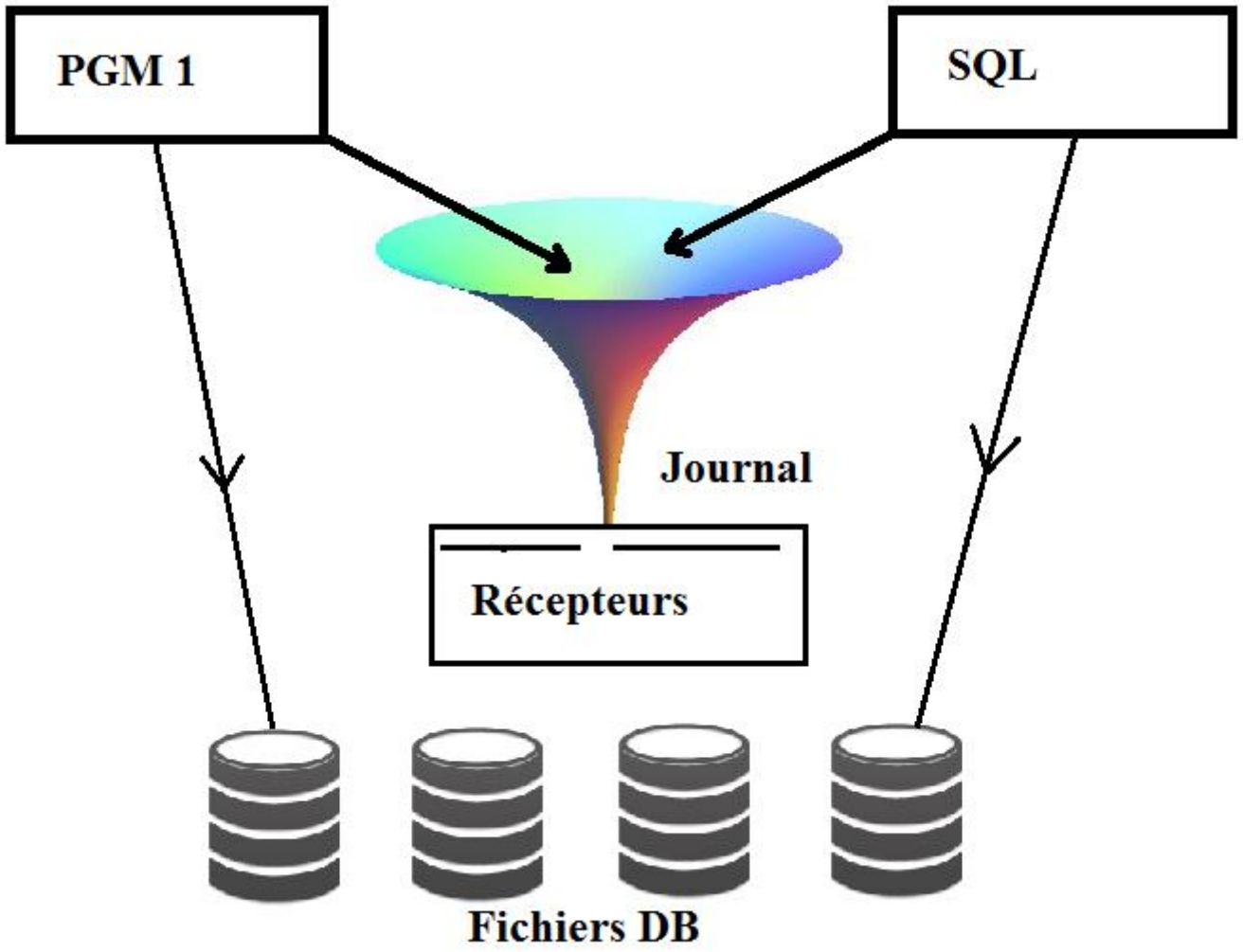
Propriétés d'une transaction

- Dans la littérature, une transaction est définie par les propriétés ACID, qui permettent de garantir la fiabilité d'une transaction
  - Atomique
    - Indivisible
  - Cohérente
    - Le système passe d'un état valide à un autre état valide, même si les états intermédiaires ne le sont pas
  - Isolée
    - Indépendance des transactions
  - Durable
    - Permanent après validation

Intérêt

- La transaction permet de s'assurer que les informations relationnelles stockées dans BD sont dans un état cohérent
  - Avant et après un traitement
  - Aussi bien en cas de réussite que d'échec
- Cela réduit donc la quantité de code à réaliser en RPG
  - Le code de gestion d'erreur qui permet de « défaire » les actions partiellement réalisées

Journaux sur l'IBM i



Mise en œuvre : création du journal

- Création d'un récepteur de journal (**CRTJRNRCV**)
  - L'objet qui contiendra les données
- Création du journal (**CRTJRN**)
  - L'objet qui permettra l'écriture des données
- L'environnement nécessaire est créé
  - Vous devez maintenant journaliser vos tables
- Remarques
  - En SQL, quand vous faites **CREATE COLLECTION**
    - un journal **QSQJRN** et un récepteur **QSQJRN0001** sont créés par défaut dans votre collection (bibliothèque)

Mise en œuvre : journalisation des tables

- Manuellement
  - **STRJRNPF**
- Automatiquement pour l'ensemble d'une bibliothèque
  - **STRJRNLIB**
- Vous pouvez choisir les images après ou les images avant et après
  - **IMAGES(*AFTER** ou ***BOTH)**
- Cela n'a aucun impact sur le contrôle de validation, par contre il existe une commande système
  - **RMVJRNCHG** qui permet d'invalider des mises à jour par rapport à des critères utilisateurs travail etc.…, et n'est utilisable qu'avec ***BOTH**
- A ce stade toutes vos modifications sont journalisées

Power Systems

for Business

Usage(s)

IBM

Usage

- La transaction BD est une opération purement technique
  - De manipulation de données
  - Très courte (ms – s)
- Elle est souvent utilisée à tort en tant que transaction métier
  - D'une durée plus longue (s – h)
  - Qui fait entrer en compte plusieurs transactions BD
  - En cas d'anomalie pour une transaction métier (annulation de l'achat d'un billet d'avion par exemple), c'est un mécanisme de compensation qui permet l'annulation métier

Mise en œuvre du « Contrôle transactionnel »

- Sur un applicatif traditionnel
  - CL et RPG

CL

```
STRCMTCTL
...
CALL RPG
...
COMMIT | ROLLBACK
ENDCMTCTL
```

RPG

```
dcl-f PRODUIT usage(*input *output)
      keyed commit ;
FPRODUIT   UF   E          K DISK    COMMIT
...
COMMIT ou ROLLBACK
```

- Commentaires
  - Le contrôle transactionnel doit être démarré lorsque le RPG est exécuté
  - Si le travail se termine anormalement, le système fait un **ROLLBACK** et un **ENDCMTCTL** implicite
    - On peut préciser un objet de notification qui permet de stocker des clés en cas de reprise

Mise en œuvre du « Contrôle transactionnel »

- Sur un applicatif RPG/SQL
  - Vous êtes par défaut en contrôle de validation
- Paramètre par défaut (implicite) **COMMIT(*CHG)** sur les commandes
  - **CRTSQL...**
  - **RUNSQLSTM**
  - **RUNSQL**
- Il s'applique sur l'ensemble des
  - Tables, vues, procédures, fonctions, index … utilisés dans les instructions SQL du programme
  - Aucun effet sur les cartes **F/dcl-f**

Gestion explicite

- Une solution pour éviter de commit sur des tables particulières
  - Compiler vos programmes avec **`COMMIT(*NONE)`**
  - Utiliser l'instruction **`SQL SET TRANSACTION`**
    - permet de démarrer le contrôle de validation à n'importe quel moment de l'exécution du programme

SET TRANSACTION

- Syntaxe
  - **SET TRANSACTION ISOLATION LEVEL**
    - **NO COMMIT**
    - **READ UNCOMMITTED**
    - **READ COMMITTED**
    - **REPEATABLE READ**
    - **SERIALIZABLE**
  - Ce sont les différents niveaux de transaction

Exemple

```
// code sql hors commit
 ....
exec sql
   update log
   set  loguser =:user,
        logtxt = :texte,
        logtime = current time,
        logdate = current date;
....
// démarrage contrôle transactionnel à  la demande
exec sql
  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
.....
exec sql
update
   tarif
   set prix = prix * :taux ;
// validation
exec sql commit;
```

Exemple

- Dans votre log le message SQL7981

```
ID message . . . . . :   SQL7981                                  00
Type de message  . . . :   Achèvement
Date d'envoi . . . . . :   12/05/15                               08:56:22

Message . . . . :   SET TRANSACTION exécutée.
Cause . . . . . :   La demande de modification du niveau de protection de NONE
  en RR et de définition du mode lecture écriture en READ WRITE a abouti pour
  le groupe d'activation 2.
```

```
Exec sql
 SET TRANSACTION ISOLATION
 LEVEL SERIALIZABLE ;
```

- Vous pouvez supprimer le contrôle de validation

```
ID message . . . . . :   SQL7981                                  00
Type de message  . . . :   Achèvement
Date d'envoi . . . . . :   12/05/15                               08:57:12

Message . . . . :   SET TRANSACTION exécutée.
Cause . . . . . :   La demande de modification du niveau de protection de LOCK
  en NONE et de définition du mode lecture écriture en READ WRITE a abouti
  pour le groupe d'activation 2.
```

```
Exec sql
 SET TRANSACTION ISOLATION
 LEVEL NO COMMIT    ;
```

Niveau d'isolation

- Equivalence IBM / SQL

SET TRANSACTION	IBM	SQL
READ UNCOMMITTED	*CHG	*UR (Uncommitted Read)
READ COMMITTED		*CS (Cursor Stability)
REPEATABLE READ	*ALL	*RS (Read Stability)
NO COMMIT	*NONE	*NC (No Commit)
SERIALIZABLE		*RR (Repeatable Read)

Gestion des options SQL

- Il est possible d'indiquer des options de compilation SQL via **SET OPTION**
- Cela permet de mémoriser les options de compilation dans le membre source
  - Au lieu des mots-clés de la commande CRTSQLRPGI

```
//
// Options par défaut SQL
//
exec sql Set Option
                Naming    = *SYS,
                Commit    = *CHG,
                UsrPrf    = *USER,
                DynUsrPrf = *USER,
                Datfmt    = *ISO,
                CloSqlCsr = *ENDMOD,
                Sqlpath   = *LIBL ;
```

Gestion des options SQL

- La principale option qui concerne le contrôle de validation est le paramètre **`COMMIT`**
  - Vous pouvez également préciser les utilisateurs d'exécutions
  - La portée d'un curseur
  - Le format de la date
  - Le chemin de recherche des tables, vues, procédures …
  - ...
- Vous pouvez utiliser l'instruction SQL **`INCLUDE`** afin de standardiser vos options de compilations
  - Créer un membre contenant uniquement
    - **`SET OPTION …`**

Gestion des options SQL

- Vous devez ensuite le déclarer comme votre première instruction SQL, dans l'ordre des n° de séquence, de votre programme
  - Le membre source est pris dans le fichier paramétré dans **INCFILE()** de la commande **CRT***

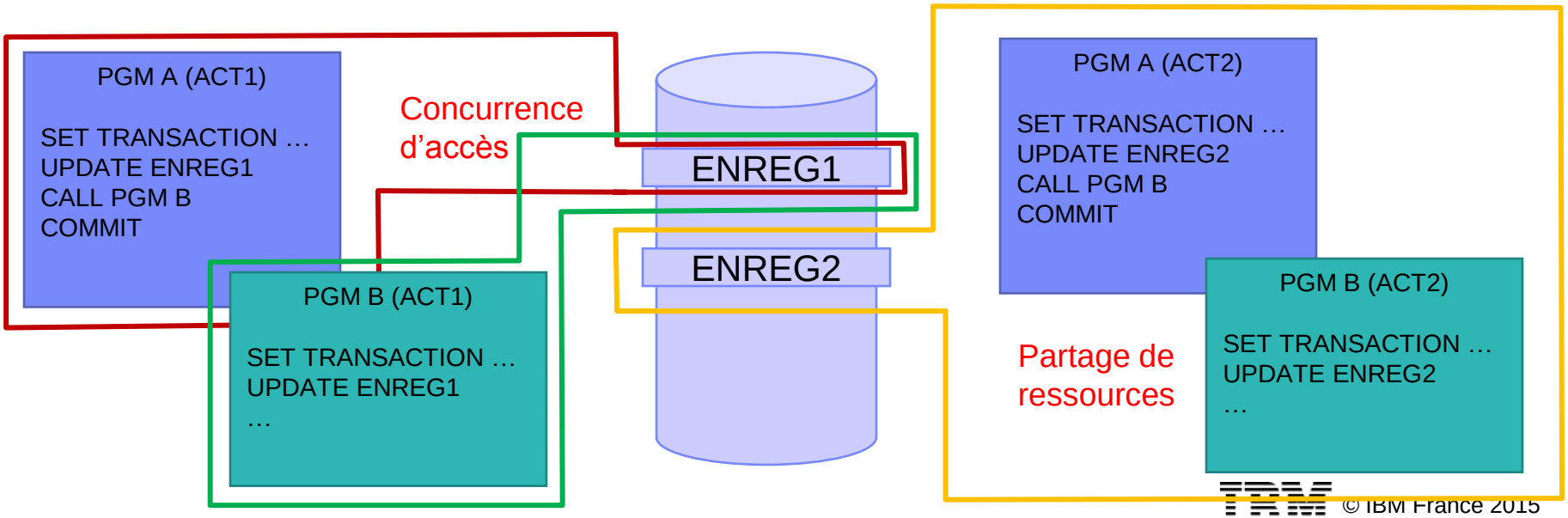
- Exemple :

```
exec sql
   include SQL_OPT  ;
```

```
//
// Options par défaut SQL
//
//*exec sql
//*         Set Option
//*             Naming    = *SYS,
//*             Commit    = *CHG,
//*             UsrPrf    = *USER,
//*             DynUsrPrf = *USER,
//*             Datfmt    = *ISO,
//*             CloSqlCsr = *ENDMOD;
```

SQLRPGLE et ILE

- En environnement ILE
  - L'ensemble des ressources utilisées et partagées par les programmes sont affectées au **groupe d'activation**, et non au programme lui-même
  - La portée d'une transaction est donc le groupe d'activation dans lequel elle a été initiée



Différence RPGLE / SQL

- Différences principales
  - Les maj peuvent porter sur plusieurs lignes
    - **`Update fichier set zone = 'O' where code = 'YES'`**
  - Les instructions SQL ne plantent pas
    - test du SQLCODE à traiter
    - **`If sqcode = xxx`**
    - **`Commit`**
    - **`endif`**
    - Voir annexes
  - Plus de comportements implicites
    - Déclaration paramètre **`COMMIT()`** ou **`SET OPTION COMMIT`**

Gestion du SQLCODE

- On peut gérer par **SQLCODE (10 I 0)** ou **SQLSTATE (5 A)**
  - SQLCODE est spécifique DB2
  - SQLSTATE est une norme du monde SQL
  - Le plus souvent on utilise le SQLCODE
    - 0 : tout c'est bien passé
- Le **SQLCODE** renvoie un statut sur la dernière instruction uniquement

```
dcl-s W_sqlcode like(sqlcode)
   ou
D W_sqlcode                S                like(sqlcode)

Exec sql Update table_1;
W_sqlcode = sqlcode;
Exec sql Update table_2;
If w_sqlcode = 0 and (sqlcode = 0 or sqlcode = 100) ;
   Commit;
Endif;
```

  - Méfiez notamment des boucles Fetch !

SQLCODE ou SQLSTATE

- Sur IBM i on utilise pratiquement tout le temps le SQLCODE
  - spécifique à DB2
- Avantages à utiliser SQLSTATE
  - Il existe sur toutes les base de données
  - Il est classé, les 2 premiers caractères ont une signification
  - Exemple :
    - 2D Invalid Transaction Termination
      - 2D522 COMMIT and ROLLBACK are not allowed in an ATOMIC Compound statement.
      - 2D528 Dynamic COMMIT or COMMIT ON RETURN procedure is invalid for the application execution environment
      - 2D529 Dynamic ROLLBACK is invalid for the application execution environment

GET DIAGNOSTICS

- Cette instruction SQL permet d'obtenir de nombreuses informations
  - L'environnement SQL en cours
    - Connexion, profil …
  - La dernière instruction
    - Nombre de lignes, type de curseur …
  - Les conditions, c'est-à-dire exceptions
    - Potentiellement plusieurs pour une instruction
    - Message d'erreur, SQLCODE, SQLSTATE, table concernée, contrainte violée, routine en erreur …

Exemple GET DIAGNOSTICS

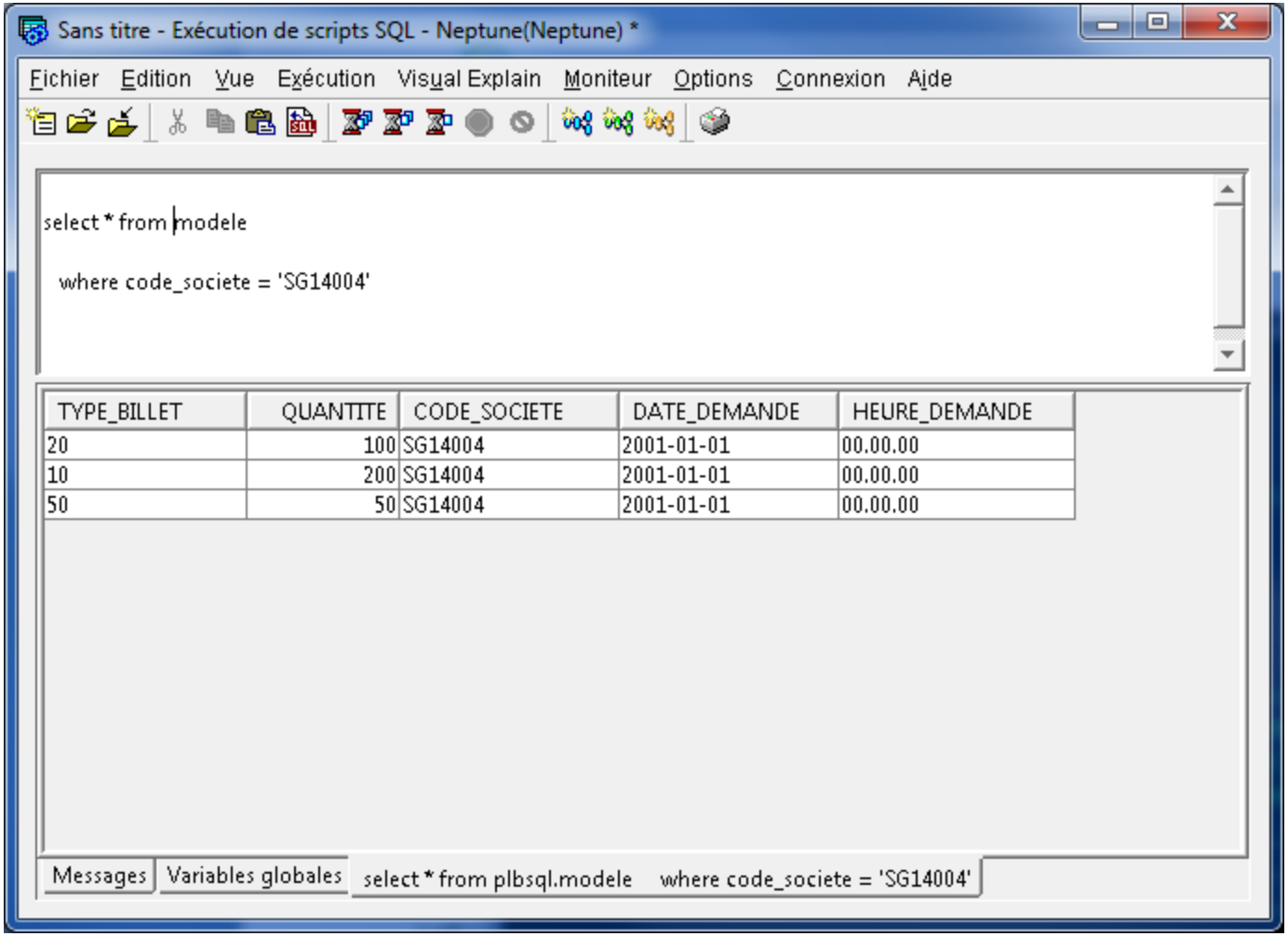
```
D                 DS
D  lastState                        5a
D  Class_SQL                        2a   overLay(lastState)

D W_SUCCESS       C                  '00'
D W_WARNING       C                  '01'


 .....
  exec SQL
    get diagnostics condition 1 :lastState = RETURNED_SQLSTATE ;
  select;
   when (Class_SQL = W_SUCCESS);
 .....
   when (Class_SQL = W_WARNING);
 .....
   Other;
 .....
  endsl ;
```

Exemple simplifié

- Une alimentation de distributeurs de billets de banques
  - Un fichier modèle contient les commandes par défaut d'une agence
  - Un fichier alimentation contient les commandes en cours pour la journée
- Ils ont la même structure
  - **TYPE_BILLET CHAR(5)**
  - **QUANTITE DEC(6)**
  - **CODE_SOCIETE CHAR(8)**
  - **DATE_DEMANDE DATE**
  - **HEURE_DEMANDE TIME**
- Notre exemple
  - Agence SG14004
  - Volontairement full SQL



Sans contrôle de validation

- Création d'un fichier temporaire dans QTEMP

```
create table qtemp/waliment as (
select * from modele where code_societe = 'SG14004'
) with data
```

- Horodatage de la demande

```
update qtemp/waliment
set date_demande  = current_date,
    heure_demande = current_time
```

- Module de mise à jour des quantités demandées

```
call majqte ('SG14004') ;
if sqlcode = 0 ;
```

  - Si tout était ok on recopiait le fichier

```
   insert into aliment
      select * from qtemp/waliment ;
endif ;
```

majqte, procédure ext. qui permet de modifier les quantités par minitel

Avec contrôle de validation

- -- Création de la demande dans le fichier aliment (journalisé)

```
include SQL_OPT ;
insert into aliment
   select * from modele where code_societe = 'SG14004' ;
```

- -- Horodatage de la demande

```
update aliment
set date_demande  = current_date,
    heure_demande = current_time
where code_societe = 'SG14004' ;
```

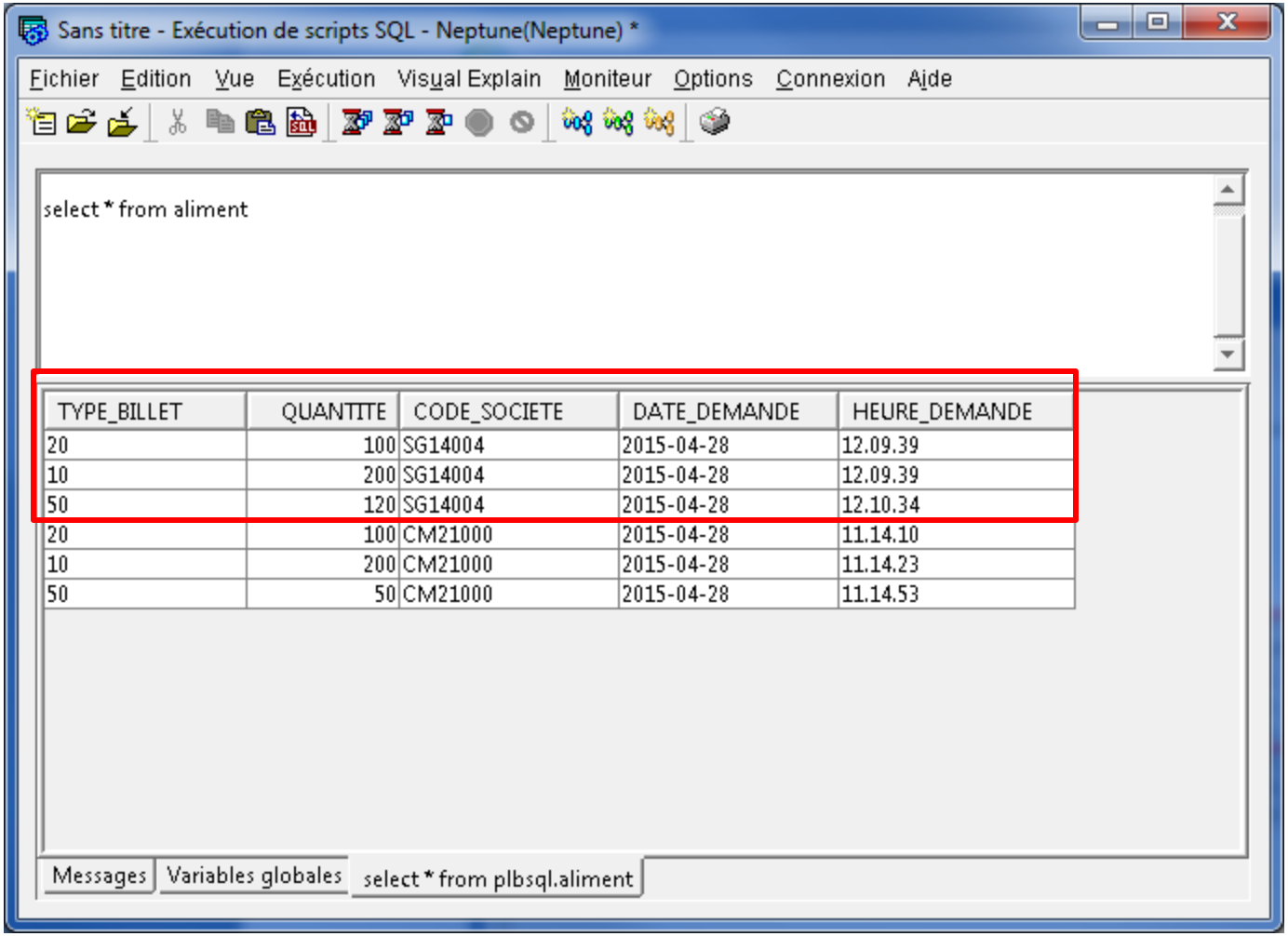
- -- Module de mise à jour des quantités demandées

```
call majqte( 'SG14004' ) ;
```

- -- Si la procédure de mise à jour ok

```
if sqlcode = 0;
   commit;
else;
  rollabck;
endif;
```

majqte, procédure ext. qui permet de modifier les quantités par minitel



Exemple de gestion de verrouillages

- Soit une table

```
CREATE TABLE MESSAGE (
  IDENT   CHAR (7 )  NOT NULL WITH DEFAULT,
  MESSAGE CHAR (80 ) NOT NULL WITH DEFAULT)
```

  - La table est journalisée
    - image après, option par défaut de SQL
  - Voici
    - Un programme de création
    - Un programme de mise à jour
    - Un programme de lecture
    - Un programme de lecture avec options

Programme de création

```
exec sql
  include SQL_OPT    ;

exec sql
   INSERT INTO messages
   VALUES('0000001', 'Création Réussie') ;

 if sqlcode <> 0;
    ...
 endif ;

 return ;
```

Programme de modification

```
exec sql
  include SQL_OPT ;

exec sql
   UPDATE messages
   SET MESSAGE = 'Modification Réussie'
   WHERE ident ='0000001' ;

select ;
when sqlcode =  100 ;     // non trouvé
   dsply 'Not found' ;
when sqlcode =  -913 ;    // Vérrouillé
   dsply 'Record locked' ;
endsl ;
```

Programme de lecture

```
dcl-s wMessage char(80) ;

exec sql
  include SQL_OPT ;

exec sql
   select message into :wMessage
   from messages
   Where ident = '0000001' ;

select ;
when sqlcode = 0 ;        // OK
   dsply %subst( wMessage : 1 : 50 ) ;
when sqlcode = 100 ;      // Non trouvé
   dsply 'Not found' ;
when sqlcode = -913 ;     // verrouillé
   dsply 'Record locked' ;
endsl ;
```

Programme de lecture avec option

```
dcl-s wMessage char(80) ;

exec sql
  include SQL_OPT ;

exec sql
   select message into :wMessage
   from messages
   Where ident = '0000001'
   skip locked data ;

select ;
when sqlcode =  0 ;       // OK
   dsply %subst( wMessage : 1 : 50 ) ;
when sqlcode =  100 ;     // Non trouvé
   dsply 'Not found or record locked' ;
endsl ;
```

Comportement avec validation en cours

- Les transactions permettent l'isolation des données
  - Entre les transactions
- C'est-à-dire
  - Il ne faut pas considérer les transactions que nécessaires à effectuer des modifications (au sens large)
  - Mais aussi des lectures sous contrôle transactionnel

Comportement avec validation en cours

Travail 1 premier lancé

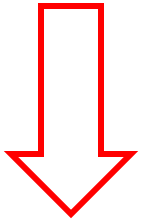
Programme de création
Ou
Programme de modification

COMMIT *ALL
Ou
COMMIT *CHG

Travail 2 deuxième lancé

Programme de Modification

COMMIT *ALL
Ou
COMMIT *CHG



SQLCODE 913
SQLSTATE = '57033'

Mise à jour impossible

Comportement avec validation en cours

Travail 1 premier lancé

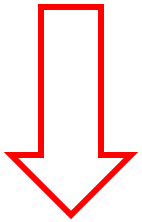
Programme de création
Ou
Programme de modification

COMMIT *ALL
Ou
COMMIT *CHG

Travail 2 deuxième lancé

Programme de Lecture Simple

COMMIT *CHG



SQLCODE 0
SQLSTATE = ' '

Lecture possible

Comportement avec validation en cours

Travail 1 premier lancé

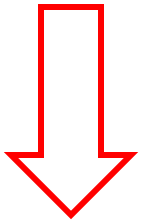
Programme de création
Ou
Programme de modification

COMMIT *ALL
Ou
COMMIT *CHG

Travail 2 deuxième lancé

Programme de Lecture Simple

COMMIT *ALL



SQLCODE 913
SQLSTATE = '57033'

Lecture Impossible + plantage

Comportement avec validation en cours

Travail 1 premier lancé

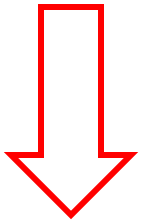
Programme de création
Ou
Programme de modification

COMMIT *ALL
Ou
COMMIT *CHG

Travail 2 deuxième lancé

Programme de Lecture
Avec option

COMMIT *ALL



SQLCODE 100
SQLSTATE = 'xxxxx'

Lecture Impossible +
Pas de plantage

Tableaux récapitulatif

- Règles de verrouillages

Actions	NC	UR	CS	RS	RR
Accès aux lignes en cours de modification?	O	O	N	N	N
Mise à jour des lignes en cours de transaction?	N	N	N	N	N
Le même ordre produit même résultat?	N	N	N	N	O
Une ligne mise à jour est modifiable ailleurs?	O	*	*	*	*
Une ligne lue est modifiable ailleurs?	**	**	**	N	N

  - * Non la ligne est verrouillée jusqu'au COMMIT
  - ** Non avec un curseur est FOR UPDATE

SELECT avec niveau de transaction

- Il est en général souhaitable de pouvoir effectuer certaines lectures
  - hors contrôle transactionnel alors qu'une transaction est en cours
  - Sous contrôle transactionnel alors qu'aucune transaction n'est en couts
- Pour ce faire :

  `SELECT …`

  `WITH NC|UR|CS|RS|RR`
  - Permet d'indiquer que le **`SELECT`** doit s'exécuter avec le niveau d'isolation correspondant

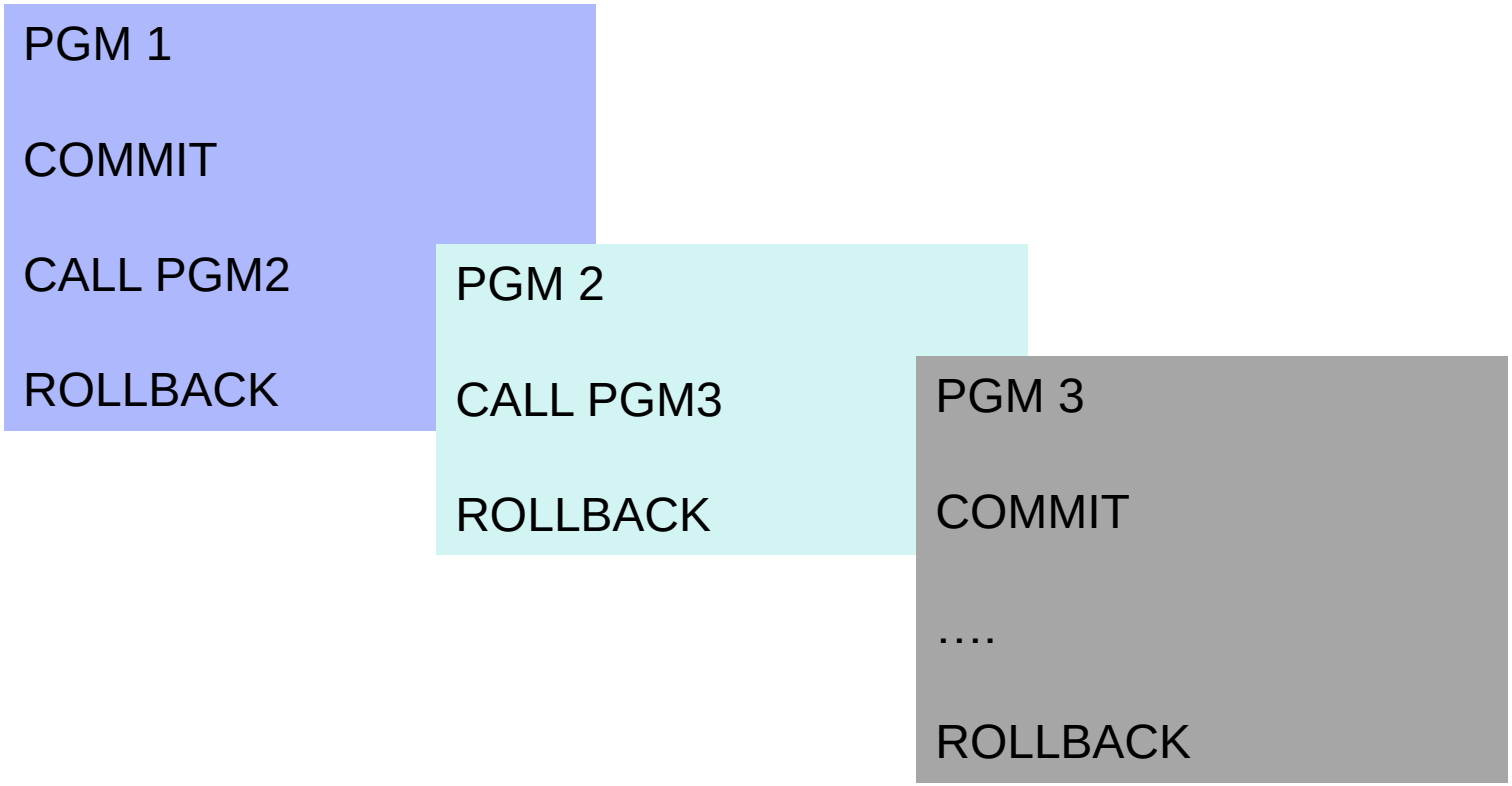
  `USE AND KEEP EXCLUSIVE LOCKS`
  - A utiliser avec précaution : toutes les lignes lues seront verrouillées pour tout accès avec les niveaux **`CS`**, **`RS`** ou **`RR`**. Les accès concurrents avec **`NC`** ou **`UR`** restent possibles

Options de lecture

- Nouveauté 6.1 / 7.1

Option	Résultat d'un select
`WAIT FOR OUTCOME`	Attendre que les lignes verrouillées soient libérées (CS ou RS)
`SKIP LOCKED DATA`	Les lignes verrouillées sont ignorées (NC, UR, CS, ou RS)
`USE CURRENTLY COMMITTED`	Utiliser les valeurs déjà validées (SELECT sous CS)

Qui gère la transaction ?



- On voit qu'à 1 ou 2 niveaux d'appels, on peut gérer mais au de la cela devient plus sportif

Qui gère la transaction ?

- Il y a deux écoles qui s'affrontent
  - Mettre en place une couche a qui incombe cette partie
    - On arrive vite à un **`autocommit`** qui peut faire perdre de l'intérêt à l'histoire
  - Faire du sur mesure
    - Il faudra se limiter à certaines transactions de vos développements
    - Avoir une vision globale des développements
  - Faire un mixte
    - Une couche par défaut
    - Et du sur mesure en utilisant les **`savepoints`** par exemple !
      - On parle des **`savepoints`** plus tard …

Qui gère la transaction ?

- Gestion de la transaction
  - Par l'appelant
    - Difficile à gérer en chaines d'appels
  - Par le programme lui-même
    - Permet de gérer la transaction au niveau le plus simple/bas
    - Permet de faire rentrer dans la transaction uniquement les opérations nécessaires
  - Groupe d'activation
    - Portée par défaut des transactions SQL
    - Pour **`STRCMTCTL`**
      - cf la commande

Le contrôle de validation s'applique à tout

- Jusque ici on choisissait les tables à mettre sous contrôle de validation
  - Un programme pouvait avoir deux tables sous contrôle de validation et deux autres sans
  - La logique du programme suffisait à imposer la validation des données
- Maintenant tous les fichiers sont sous contrôle de validation
  - certains peuvent poser des problèmes
- Exemple
  - Les tables de dépendance de clés (souvent compteurs)

Quid des données de clés dépendantes

```
// lecture du compteur
Exec sql
   select val into :newval
   From compteur ;
// mise à jour du compteur
Exec sql
   update Compteur
   set val = :Newval + 1 ;
// utilisation du nouvel identifiant
Exec sql
   insert into facture values( :Newval, ...) ;
// Validation
Exec sql
   commit;
```

- On parle souvent commit imposé
  - il vaut mieux un trou dans un compteur qu'un doublon

Point de sauvegarde

- C’est la possibilité de mettre en place des validations partielles
  - Pour pallier la rigidité de la transaction
- Exemple
  - Première partie indispensable
  - Deuxième partie optionnelle
  - En cas de plantage on fait un **ROLLBACK** jusqu’au point de sauvegarde
  - Ci-joint un exemple RPGLE

Point de sauvegarde

```
// Création d'un point de sauvegarde
EXEC SQL SAVEPOINT Savepoint_1 ;

if sqlcode = xxx ;
   // Invalidation partielle jusqu'au point de sauvegarde
   EXEC SQL ROLLBACK to SAVEPOINT Savepoint_1 ;
else ;
   // Validation complète
   EXEC SQL Commit ;
   EXEC SQL Release SAVEPOINT ;
endif ;
```

Power Systems

for Business

Outils

IBM

Poste de journaux

- Les postes de journaux sont divisés en Code et type
  - Les principaux codes
    - **D** - Opération sur un fichier base de données
    - **J** - Action sur un journal ou un récepteur
    - **R** - Indique une opération sur un record
    - **C** - Contrôle de validation (voir plus loin)
  - Les principaux type pour le code R
    - **BR** - Image avant d'un enregistrement suite à une opération d'annulation
    - **DL** - Enregistrement supprimé
    - **DR** - Enregistrement supprimé pour effectuer une opération de remise à l'état initial
    - **PT** - Enregistrement ajouté
    - **PX** - Enregistrement ajouté directement à un membre
    - **UB** - Image avant d'un enregistrement (si journalisé avec *Both)
    - **UP** - Image après d'un enregistrement
    - **UR** - Image après d'un enregistrement suite à une opération d'annulation

Poste de journaux sur les enregistrements

- Les principaux type pour le code C
  - **BC** - Démarrage du contrôle de validation
  - **SC** - Démarrage d'un cycle
  - **CM** - Commit
  - **RB** - Rollback
  - **LW** - Fin de transaction, écrit uniquement avec OMTJRNE(*NONE)
  - **EC** - Arrêt du contrôle de validation
- Optionnel

  ADDENVVAR ENVVAR(QTN_JRNSAVPT_*ALL_*ALL) VALUE(*YES) LEVEL(*SYS)
  - **SB** - Début d'un transaction imbriquée ou SAVEPOINT
  - **SQ** - Libération d'un SAVEPOINT ou commit d'une transaction imbriquée (Procédures cataloguées)
  - **SU** - Rollback d'un SAVEPOINT ou d'une transaction imbriquée

Affichage d'un journal

- En mode 5250
  - Par la commande **DSPJRN**
    - Possibilité de sortie fichier
- Depuis la version 6.1
  - par requête SQL
- Depuis la version 7.2
  - par System i Navigator

DSPJRN + filtres

Opt	Séquence	Code	Type	Objet	Bibliothèque	Travail	Heure
_	1	J	PR			SCPF	8:07:03
_	2	F	MS	FICHIER	QGPL	BEX933_EXP	22:38:17
_	3	D	DH	FICHIER	QGPL	BEX933_EXP	22:40:45
_	4	F	MS	T1	BERTHOIN	BEX933_EXP	23:04:16
_	5	F	MS	T2	BERTHOIN	BEX933_EXP	23:04:16
_	6	F	MS	T3	BERTHOIN	BEX933_EXP	23:04:16
_	7	J	RS	TJRN000001	BERTHOIN	BEX933_EXP	23:04:51
_	8	J	RS	TJRN000002	BERTHOIN	BEX933_EXP	23:04:51
_	9	J	RS	TJRN000003	BERTHOIN	BEX933_EXP	23:04:51
_	10	D	DH	T1	BERTHOIN	BEX933_EXP	23:04:51
_	11	D	DH	T2	BERTHOIN	BEX933_EXP	23:04:51
_	12	D	DH	T3	BERTHOIN	BEX933_EXP	23:04:51

- Exemple
  - **F - MS** : Fichier sauvegardé

DSPJRN pour contrôle transactionnel

- Affichage des postes d'un journal

```
66    F      OP    FICHIER      QGPL          QPADEV009G    8:10:02
67    C      BC                               QPADEV009G    8:10:02
68    C      SC                               QPADEV009G    8:10:02
70    R      PX    FICHIER      QGPL          QPADEV009G    8:10:02
71    F      CL    FICHIER      QGPL          QPADEV009G    8:10:02
```

- Exemple
  - C / BC début de contrôle de validation
  - C / SC Début de cycle de validation
  - R / PX Enregistrement ajouté

DSPJRN pour contrôle transactionnel

- Après validation

```
C      CM                                  QPADEV009G    8:42:17
```

- Exemple
  - **C / CM -** Validation

- Après invalidation

```
C      RB                                  QPADEV009G    8:45:56
```

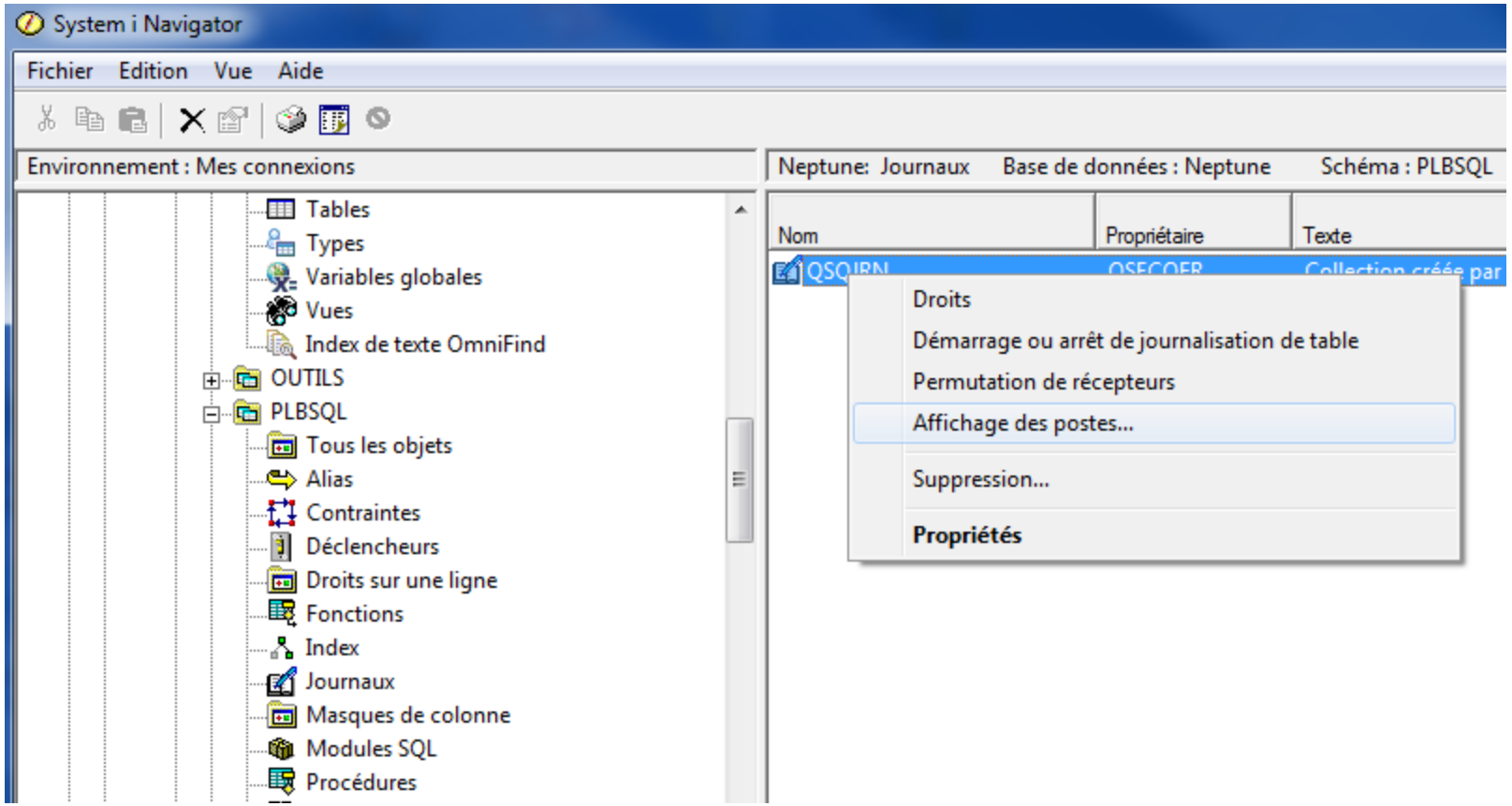
- Exemple
  - **C / RB -** Invalidation

Utilisation UDTF QSYS2/Display_Journal

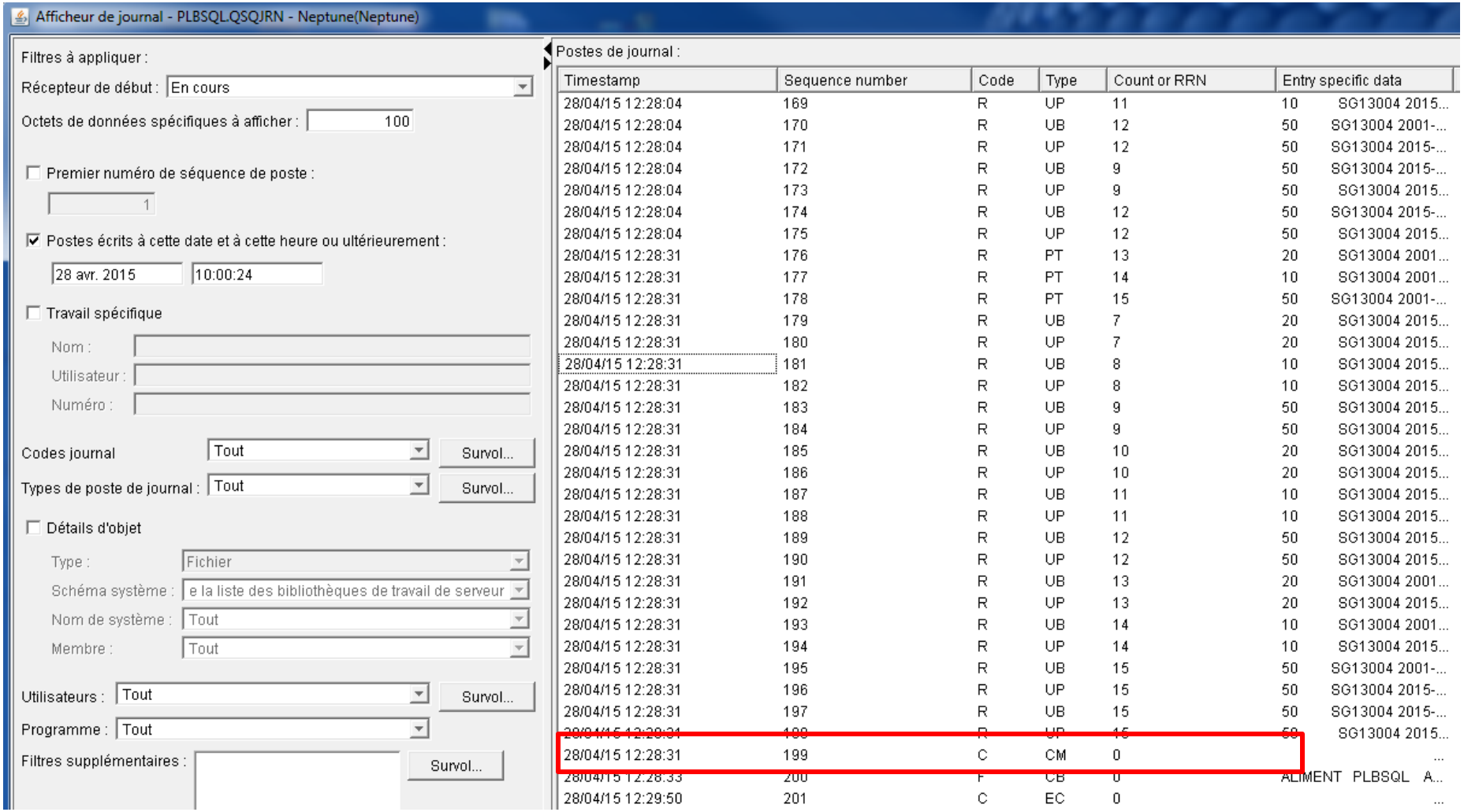
```
select * from table (Display_Journal(
-- journal
   'Library',
   'Journal',
   ' ',
   '*CURCHAIN',
-- format
   CAST('2010-03-30-19.01.15.000000' as TIMESTAMP),
   CAST(null as DECIMAL(21, 0)),
   '',
   '',
   '',
   '',
   '',
   '',
   '',
   '',
   ''
) ) as journ
```

- La donnée Entry_Data est un **BLOB** (Binary Large Object)
  - vous pouvez utiliser **CAST(ENTRY_DATA AS CHAR(nnn))** pour voir les données

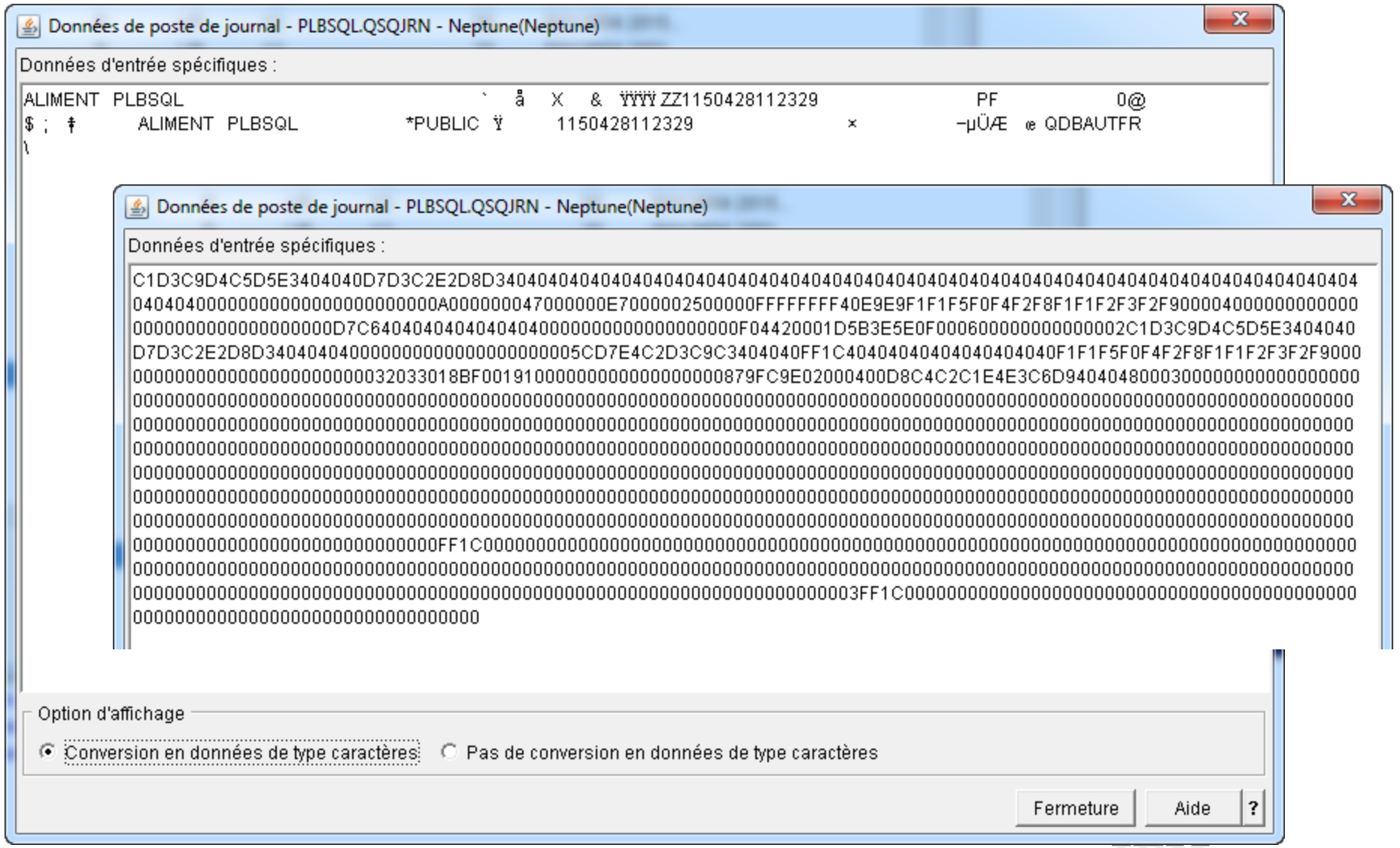
Via IBM i Access for Windows



Via IBM i Access for Windows



Via IBM i Access for Windows



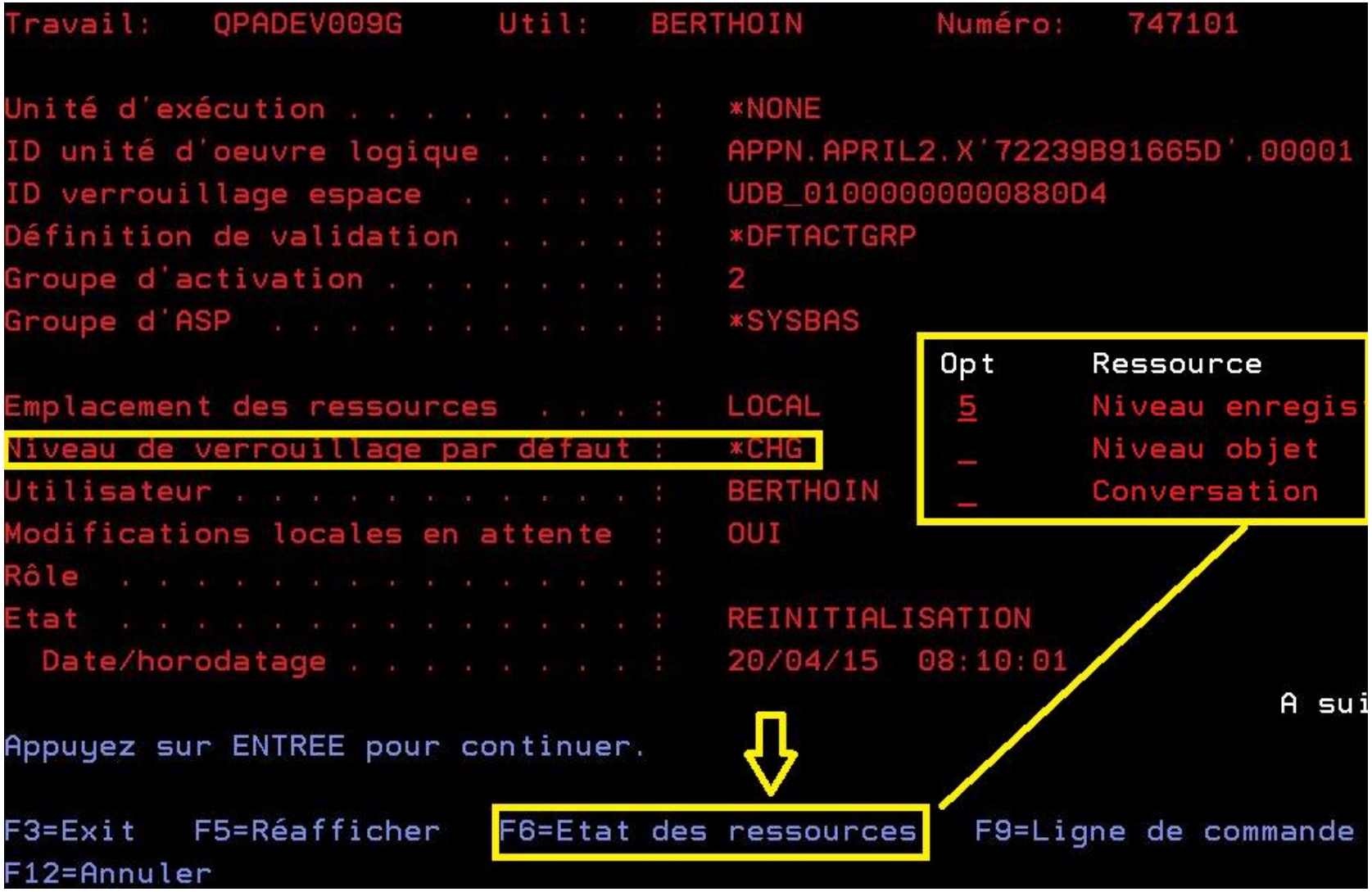
WRKCMTDFN

- Gérer les définitions de validation

```
     Définition  No      Util        Nom         Unité
Opt  validation  trav    travail     travail     exécut    Util
__   *DFTACTGRP  747101  BERTHOIN    QPADEV009G  *NONE     BERTHOIN
```

- Par exemple
  - Contrôle de validation dans le groupe d'activation par défaut
  - Option 5 pour voir le détail

WRCMTDFN



WRCMTDFN

- Affichage des validations

```
Définition de validation . . . . . :   *DFTACTGRP

                                    ----------Modifications----------
                                                                   En
Fichier     Biblio      Membre      Validation   Invalidat     instance
FICHIER     QGPL        FICHIER             0           0            1
```

- Par exemple
  - On a une Validation en instance que vous pouvez voir par un **DSPJRN**

Sauvegarde d'un fichier en cours de validation

- **SAV… SAVACT(*SYNCLIB)**
  - Indique que l'on peut sauvegarder en cours d'utilisation
- Vous pouvez également utiliser des points de synchronisation **SYNCID(nom ...)** que vous aurez définis au préalable par la commande **STRSAVSYNC**
- Par défaut vous obtenez un CPD836A

  - Option ***NOCMTBDY** pour ne pas attendre les validations en cours

Fichier temporaires

- Attention
  - Les fichiers dans QTEMP ne sont pas journalisés par défaut, donc impossible de travailler sous CV, on peut démarrer unitairement la journalisation sur la table, pour une application existante par exemple.
  - Les tables crées par CPYF / CRTDUPOBJ dans une bibliothèque, sont journalisées uniquement si on l'a demandé sur la bibliothèque par STRJRNLIB, On peut également démarrer la journalisation unitairement.
  -
  - Les tables générées pas SQL ont le même comportement
    - Declare global tempory table …
    - Create table … like,
    - Create table … as (select …)
- Dans certains cas on intérêt à gérer ses tables dans un module séparé sans contrôle de validation

Power Systems

i for Business

PL/SQL

IBM

PL/SQL : Procedural Langage

- C’est le langage SQL qui permet d’écrire
  - Des procédures
  - Des fonctions
  - Des triggers
- Particularité de DB2 for i
  - Il est possible pour chacun de ces types d’utiliser un programme RPG
    - Et donc SQL RPG !
- Pour les routines avec programmes externes RPG
  - Portée de la transaction = groupe d’activation

Généralités

- Fonction SQL
  - De façon générale, une fonction effectue le plus souvent des lectures
  - Il n'est pas possible d'appeler une procédure qui effectue COMMIT, ROLLBACK ou SET TRANSACTION
- Procédure SQL
  - Elle sert aussi bien à restituer des informations qu'à effectuer des traitements
- Trigger
  - Il est appelé à chaque modification d'un enregistrement, qui peut déjà se faire sous contrôle transactionnel
  - Il n'est pas possible d'appeler une procédure qui effectue COMMIT, ROLLBACK ou SET TRANSACTION
- L'ensemble de ces routines peut être appelé sous contrôle transactionnel ou non …

Procédure

```
CREATE PROCEDURE ...
COMMIT ON RETURN [YES|NO]
CONCURRENT ACCESS RESOLUTION [DEFAULT|USE CURRENTLY COMMITTED|WAIT FOR OUTCOME]
```

- Détails
  - **COMMIT ON RETURN**
    - **NO** (défaut) : DB2 n'effectue aucun **COMMIT** ni **ROLLBACK** automatique
    - **YES** : DB2 effectue un **COMMIT** automatiquement si la procédure s'exécute sans anomalie. Dans le cas contraire, **ROLLBACK** n'est pas exécuté
  - **CONCURRENT ACCESS RESOLUTION**
    - **USE CURRENTLY COMMITTED** : DB2 utilise les données actuellement validées et non les données en attente de validation
    - **WAIT FOR OUTCOME** : DB2 attend que les données soient disponibles (libérées)
      - Permet les lectures en niveau de transaction **CS (Cursor Stability)**

Quid des triggers

- On doit limiter aux maximum les mises à jours dans les triggers
  - Trigger avant
    - De contrôle (donne une autorisation)
    - De complémentation (ajoute des informations avant écriture de l'enregistrement)
  - Trigger après
    - De log (historise une information, un comportement)
    - Action (déclenche une action)

- Attention
  - Il faut éviter le déclenchement en boucle du trigger
    - Ne pas mettre à jour d'enregistrement du même fichier

Quid des triggers

- Deux solutions utilisées
  - Soit pas de commit
  - Soit un commit spécifique dans son groupe d'activation
  - Le choix se fait au cas par cas, le trigger n'est pas là pour faire des mises à jours en masse
- Pour être au plus fin possible
  - Le trigger peut vérifier qu'il s'exécute sous contrôle transactionnel et adapter son comportement en fonction

Power Systems

for Business

Bonnes pratiques

IBM

Bonnes pratiques

- Toujours être le plus explicite possible
  - Plus on donne d'informations à DB2 et plus il est capable de gérer les verrous avec finesse
  - **SELECT** dispose de nombreuses clauses
    - **WITH NC | UR | CS | RS | RR**
    - **USE AND KEEP EXCLUSIVE LOCKS**
    - **WAIT FOR OUTCOME | SKIP LOCKED DATA | USE CURRENTLY COMMITTED**
    - **FOR READ ONLY | FOR FETCH ONLY**
    - **FOR UPDATE OF *col1, col2, ...***

Bonnes pratiques

- On y va pas à moitié, on peut choisir certaines transactions mais il faut les faire à fond
- Un temps de validation de transaction doit être court
  - **`WRKCMTDFN JOB(*ALL) STATUS(*PENDING)`**
- Si vous utilisez des curseurs en mises à jours attention à l'augmentation de conflits (for update of)
- Ayez au maximum la maitrise des validations / invalidations
  - Afin d'éviter les invalidations implicites sur des événements anormaux
  - Le vrai seul cas qu'on ne peut pas gérer c'est l'arrêt anormal du travail

Bonnes pratiques ODBC/JBDC

- Par défaut vous êtes sous contrôle de validation
  - en cas de réutilisation de code (procédure externe) vous risquez des invalidations intempestives
- Journaliser vos tables
  - Afin d'éviter le plantage sur les éléments records
- Par contre le système peut utiliser des journaux à lui pour gérer des opérations sur les tables
  - par exemple **`create table as`** ..
- Attention au progiciel, boite noire

Bonnes pratiques

- Ponctuellement, ou suite à un problème système, vous pouvez faire le contrôle suivant
  - En règle générale le système gère bien, vous pouvez quand même regarder si vous avez des validations en attente indéfinie

```
WRKCMTDFN JOB(*ALL) STATUS(*UNDECIDED)
```

- Si vous en avez des cas vous devrez faire
  - Une validation forcée option 14
  - Une invalidation forcée option 16
- Bien sûr
  - Vous devrez peut être vérifier les journaux avant d'agir
  - Ce n'est pas un mode de fonctionnement
    - si vous en avez régulièrement il faudra ouvrir un incident chez IBM

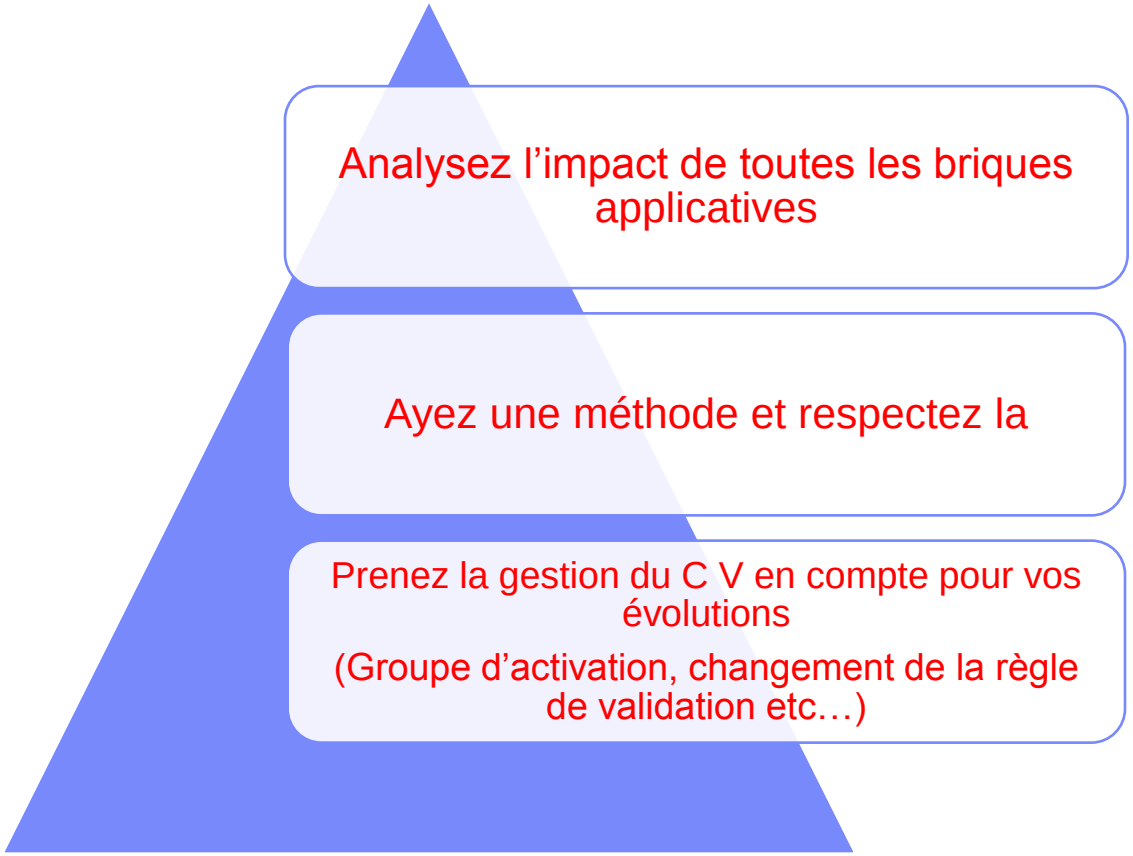
Bonnes pratiques

- Privilégier SQLSTATE au SQLCODE comme évoquer plus haut
- Privilégier GET DIAGNOSTICS pour des informations détaillées plutôt que SQLCA
- Tester et/ou mémoriser SQLCODE/STATE/GET DIAGNOSTICS après chaque instruction SQL
  - Se rapporte toujours à la dernière instruction
  - Sauf instruction de déclaration, par exemple PREPARE qui n'alimente pas la zone

Bonnes pratiques

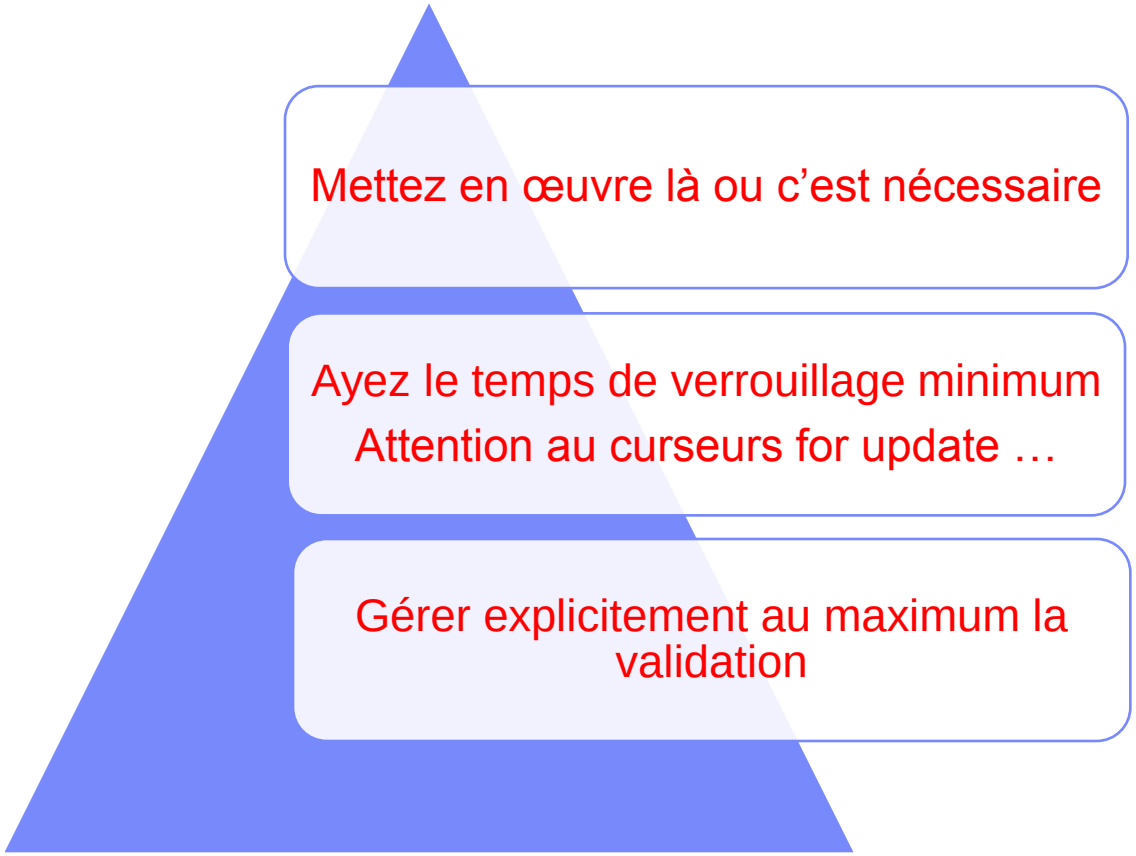
- Être particulièrement attentif au SQL dynamique
  - Il est plus difficile d'anticiper les erreurs
- La transaction génère des verrouillages
  - Attention tables compteurs, logs, …
  - Ne pas mettre 1 millions de lignes dans la transaction, ni mille tables

Synthèse (pour votre applicatif)



- Soyez Global et Permanent

Synthèse (pour vos développements)



- Soyez Précis et rapide

URLographie

- Pour la doc IBM version 7.2
  - http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzahg/ic-homepage.htm?lang=fr
- Pour les SQLCODE
  - http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzala/rzalakickoff.htm?lang=fr
  - http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzas2/rzas2finder.htm?lang=fr (pour accéder directement au finder)
- Pour le SQL Embarqué
  - http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzajp/rzajpkickoff.htm?lang=fr

Power Systems

for Business

MERCI

IBM

www.gaia.fr