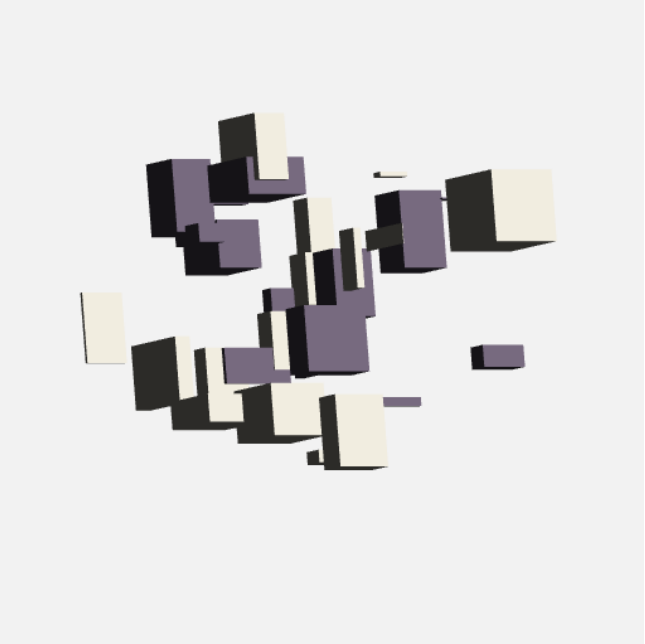
Masterclass DB2 SQL
Pour développeurs IBMi



Sommaire

1. Préambule

J'ai rédigé ce document d'abord pour mes propres besoins. J'ai démarré sa rédaction aux alentours de 2006 (ou peut être 2007) et l'ai enrichi au fil de mes découvertes et des missions de développement et de conseil dont j'ai eu la charge. Au début, c'était une collection de bonnes pratiques jetées dans un document Word, et je l'ai utilisé comme un gros « pense-bête ». Puis, constatant qu'il répondait bien à mes besoins, mais qu'il pouvait aussi servir à d'autres personnes, je l'ai restructuré pour qu'il soit plus facilement utilisable par d'autres.

Durant une longue période, j'ai assuré de nombreuses formations en entreprise, autour de la plateforme IBM i, mais aussi autour du développement Web. J'ai eu l'occasion d'utiliser le présent document comme support d'une masterclass (SQL DB2) que je proposais pendant un moment. J'ai eu à l'époque d'excellents retours de la part des apprenants, et j'ai la prétention de croire que ce n'était pas juste pour me faire plaisir.

En ce début d'année 2021, je ne m'occupe plus de formation, mais il y a encore beaucoup de développeurs qui travaillent sur système IBM i, et à qui ce document pourrait être utile. Et notamment de jeunes développeurs, qui ont peut être des difficultés pour trouver des ressources pouvant les aider dans leur travail. J'espère que ce document pourra répondre à certains de leurs besoins.

Attention, ce document n'est pas adapté pour des personnes débutant en SQL, car il présente le plus souvent des techniques SQL relativement complexes. Ces techniques sont abordées au travers d'exemples concrets issus pour une large part de cas réels auxquels j'ai été confronté sur le terrain.

Certaines techniques présentées ici intéresseront plus particulièrement les développeurs d'applications de gestion. D'autres techniques intéresseront plutôt les administrateurs de bases de données, car elles leur permettront de superviser plus efficacement le déploiement et l'administration de bases DB2 (et s'ils le souhaitent, de développeur leurs propres outils).

Pour bénéficier pleinement des techniques présentées dans ce document, on recommandera de travailler sur un serveur IBM i, à minima en V6Rx, ou mieux encore, en V7Rx.

Ce document ne couvre que très partiellement les nouveautés apparues sur serveur IBM i à partir de la V7R2. Pour une présentation plus complète des nouveautés apparues à partir de la V7, on se reportera de préférence sur le document « SQL_DBTwo_NewsV7 » qui se trouve dans le même dépôt Github.
Si vous débutez sur DB2 SQL, en particulier sur serveur IBM i, je vous recommande de lire d'abord le document « SQL_DBTwo_Quickstart » qui se trouve aussi dans le même dépôt.

Liens utiles :
https://developer.ibm.com/
https://www.foothing.net/

2. Techniques SQL avancées

2.1 Bonnes et mauvaises pratiques

Il est souhaitable de bannir - dans les programmes et les procédures stockées - les requêtes de type :

```
SELECT * FROM ...
```

Il faut toujours déclarer explicitement la liste des colonnes que l'on souhaite utiliser, et ce pour plusieurs raisons :
- d'abord des raisons de performance : il est vraiment très rare d'avoir besoin de toutes les colonnes d'une table, il est donc préférable de limiter la portée du result set aux seules colonnes réellement nécessaires.
- la structure de la table peut évoluer dans le temps, et renvoyer à terme plus ou moins de colonnes que ce qu'elle contenait à l'origine, pouvant entraîner des plantages (dans le cas de colonnes ajoutées ou supprimées ne correspondant plus, du coup, aux variables destinataires) ou des problèmes de performances

Dans un INSERT, plutôt que d'écrire ceci :

```
INSERT INTO tablex
SELECT col1, col2, col3 from tabley ;
```

Il est vivement recommandé d'écrire ceci :

```
INSERT INTO tablex (col1, col2, col3)
SELECT col1, col2, col3 from tabley ;
```

En ce cas de modification ultérieure de la structure de la table "tablex", les risques de plantage sur la requête d'insertion seront minimisés par le fait que les colonnes destinataires ont été déclarées explicitement dans la requête d'insertion (cf. la liste des colonnes en rouge).

Toujours dans le cas d'un INSERT, si on a plusieurs séries de valeurs à insérer dans une table, plutôt que d'écrire ceci :

```
INSERT INTO tablex (col1, col2, col3) VALUES (1a, 2a, 3a) ;
INSERT INTO tablex (col1, col2, col3) VALUES (1b, 2b, 3b) ;
INSERT INTO tablex (col1, col2, col3) VALUES (1c, 2c, 3c) ;
```

... on préfèrera utiliser l'écriture suivante (plus performante) :

```
INSERT INTO tablex (col1, col2, col3) VALUES
(1a, 2a, 3a) ,
(1b, 2b, 3b) ,
(1c, 2c, 3c) ;
```

Le type des données renvoyées par des calculs en SQL est directement déduit du type des colonnes en entrée. Ce point est particulièrement important sur les divisions :

```
select 20/100 from sysibm.sysdummy1; -- 0  (0.2 arrondi à 0)
select 20/100.0 from sysibm.sysdummy1; -- 0.2
select 20.0/100 from sysibm.sysdummy1; -- 0.2
select 20.0/100.0 from sysibm.sysdummy1; -- 0.2
```

Si les 2 opérateurs de la division sont de type "integer", alors SQL considère qu'il doit renvoyer un résultat du même type, d'où le problème que l'on constate sur la première requête ci-dessus.

Si on veut forcer une colonne de type integer à passer au type decimal, on peut utiliser la fonction CAST :

```
select CAST(20 as DECIMAL(5, 2))/100 from sysibm.sysdummy1; -- 0.2
```

mais on peut tout aussi bien faire ce passage par une simple addition avec 0.0 :

```
select (20+0.0)/100 from sysibm.sysdummy1; -- 0.2
```

Attention : le CAST ci-dessous, appliqué sur le résultat de la division, arrive trop tard, et le résultat final est zéro au lieu de 0,2 :

```
select cast( 20/100  as decimal(15, 2)) as diverror from sysibm.sysdummy1;
```

Le fait de placer un calcul dans une clause WHERE empêche l'optimiseur SQL de fonctionner, si le calcul s'applique directement sur une colonne de la table considéréeDB2 ne peut dans ce cas déterminer de chemin d'accès adapté, et se trouve obligé de constituer une table temporaire en pré-calculant les valeurs pour toutes les lignes de la table, même si elle est très volumineuse. Exemple de requête que DB2 ne peut optimiser :

```
select * from matable where matable.macolonne * 100 = 10000 ;
```

Requête équivalente optimisable par DB2 :

```
select * from matable where matable.macolonne = 100 ;
```

2.2 Tables temporaires

On a parfois besoin de générer un jeu de données temporaire, pour exploiter une information dans une autre requête, on pourrait dès lors être tenté de recourir systématiquement aux tables temporaires mais ce n'est pas toujours nécessaire. Par exemple, si on a besoin d'identifier des doublons relatifs à une colonne X d'une table Y, on peut recourir à la technique suivante qui est très pratique et performante :

```
select x.* from (
select codart, count(*) as comptage from produit group by codart
) x where x.comptage > 1 ;
```

Dans l'exemple ci-dessus, on crée une requête permettant de compter le nombre d'occurrences de la colonne "codart" sur la table "produit". On encapsule cette requête - qui devient dès lors une sous-requête - pour en extraire tous les cas où le comptage effectué dans la sous-requête est supérieur à 1.

Mais dans certains cas, la création de table temporaire peut se révéler nécessaire. C'est notamment le cas si on a besoin d'exploiter le jeu de données renvoyé par une requête dans plusieurs requêtes exécutées indépendamment, au sein d'un même travail.

Exemple de création de table temporaire (avec ou sans données) :

- **Mauvaise pratique** : utilisation de CREATE TABLE avec QTEMP comme bibliothèque cible

```
create table qtemp.toto2 as (
select empid, salary from gjarrige3/my_emp fetch first 5 rows only
) with data;

create table qtemp.toto2 as (
select empid, salary from gjarrige3/my_emp fetch first 5 rows only
) definition only;
```

- **Bonne pratique** : utilisation de DECLARE GLOBAL TEMPORARY TABLE

```
declare global temporary table toto3 as (
select empid, salary from gjarrige3/my_emp fetch first 5 rows only
) with data with replace ;

declare global temporary table toto3 as (
select empid, salary from gjarrige3/my_emp fetch first 5 rows only
) definition only with replace ;
```

Avantages de la seconde méthode : la clause WITH REPLACE permet de ne pas se poser la question de savoir si l'objet temporaire existe déjà ou pas.

On peut créer une table temporaire de la même façon qu'avec CREATE TABLE :

```
DECLARE GLOBAL TEMPORARY TABLE MY_EMP (
```

```
EMPID INTEGER,
NAME VARCHAR(30),
DEPTID INTEGER,
SALARY DECIMAL(11, 2),
MGRID INTEGER
) WITH REPLACE ;
```

Attention : s'il est possible d'écrire une requête de ce type dans un client SQL quelconque :

```
declare global temporary table toto3 as (
select empid, salary
from gjarrige3/my_emp
where salary > 10000
) with data with replace ;
```

... à l'intérieur d'une procédure stockée, il sera nécessaire de procéder en 2 temps :

1 - création de l'enveloppe de la table temporaire (pas de clause WHERE)

```
declare global temporary table toto3 as (
select empid, salary
from gjarrige3/my_emp
) definition only with replace ;
```

2 - insertion des données dans la table temporaire

```
insert into qtemp/toto3
select empid, salary
from gjarrige3/my_emp
where salary > 10000 ;
```

et pour être plus rigoureux, penser à déclarer la liste des colonnes destinataires (cf. chapitre "bonnes pratiques") :

```
insert into qtemp/toto3 (empid, salary)
select empid, salary
from gjarrige3/my_emp
where salary > 10000 ;
```

Les tables temporaires sont intéressantes quand on a besoin de disposer d'une table temporaire devant être utilisée pour produire plusieurs tables distinctes. Mais si cette table temporaire est destinée à produire une seule table, il peut être plus judicieux d'utiliser les CTE (cf. chapitre suivant).

A NOTER :
Une autre technique peut être utilisée pour créer des données temporaires, elle consiste à créer un jeu de données à la volée, sans passer par la création de table temporaire. On retrouvera cette technique dans le chapitre sur les UDTF, c'est la raison pour laquelle on ne s'attardera pas dessus pour l'instant. On notera que cette technique peut être aussi être utilisée dans les CTE (étudiées au chapitre suivant).

Exemple de jeu de données créé à la volée :

```
Select X.SALCODE, X.SALNAME
 From (Values
         ('ALL','All Customers'),
         ('WEB','Web Customers')
      ) X(SALCODE, SALNAME)
 Where X.SALCODE = UPPER('All');
```

2.3 Introduction aux Statistiques

Ce chapitre introductif aux statistiques va nous servir à constituer un petit jeu de données, que nous réutiliserons dans différents chapitres.

Ce jeu de données est constitué à partir d'un exemple fourni par IBM sur cette page :

Commençons par créer une table des employés avec leurs salaires :

```
CREATE TABLE FORMATION/MY_EMP(
EMPID INTEGER NOT NULL PRIMARY KEY,
 NAME VARCHAR(10),
 DEPTID INTEGER NOT NULL,
 SALARY DECIMAL(9, 2),
 MGRID INTEGER);


INSERT INTO FORMATION/MY_EMP
(EMPID, NAME, DEPTID, SALARY, MGRID)
VALUES
(1, 'Jones', 10, 30000, 10),
(2, 'Hall', 10, 35000, 10),
(3, 'Kim', 10, 40000, 10),
(4, 'Lindsay', 20, 38000, 10),
(5, 'McKeough', 10, 42000, 11),
(6, 'Barnes', 20, 41000, 11),
(7, 'O''Neil', 30, 36000, 12),
(8, 'Smith', 20, 34000, 12),
(9, 'Shoeman', 30, 33000, 12),
(10, 'Monroe', 40, 50000, 15),
(11, 'Zander', 20, 52000, 16),
(12, 'Henry', 30, 51000, 16),
(13, 'Aaron', 10, 54000, 15),
(14, 'Scott', 40, 53000, 16),
(15, 'Mills',  30, 70000, 17),
(16, 'Goyal', 20, 80000, 17),
(17, 'Urbassek', 10, 95000, NULL);
```

Exemple de report contenant une liste d'employés avec leurs salaires, suivi d'une ligne "total" (ajoutée au moyen de la clause UNION) :

```
select empid, name, salary from FORMATION/MY_EMP
union
select 0 as empid, 'TOTAL -> ', sum(salary) from FORMATION/MY_EMP ;
```

Résultat obtenu :

EMPID	NAME	SALARY
1	Jones	30000.00
2	Hall	35000.00
3	Kim	40000.00
4	Lindsay	38000.00
5	McKeough	42000.00
6	Barnes	41000.00
7	O'Neil	36000.00
8	Smith	34000.00
9	Shoeman	33000.00
10	Monroe	50000.00
11	Zander	52000.00
12	Henry	51000.00
13	Aaron	54000.00
14	Scott	53000.00
15	Mills	70000.00
16	Goyal	80000.00
17	Urbassek	95000.00
0	TOTAL ->	834000.00

Supposons maintenant que nous souhaitons obtenir une liste avec le niveau de détail ci-dessus, mais complété avec des sous-totaux par département insérés après chaque département (on retrouve ici la notion de "rupture" chère aux développeurs IBMi). On pourrait se rapprocher du résultat souhaité en combinant une série de CTE, comme dans l'exemple suivant :

```
with cte_total_par_employe as (
  select deptid, empid, name, sum(salary) as total from FORMATION/MY_EMP
  group by deptid, empid, name
),
cte_total_par_dept as (
  select deptid, 0 as empid, '' as name, sum(salary) as total from FORMATION/MY_EMP
  group by deptid, 0, ''
),
cte_total_general as (
  select 0 as deptid, 0 as empid, 'TOTAL -> ' as name, sum(salary) from
FORMATION/MY_EMP
),
cte_consolidation as (
   select * from cte_total_par_employe
  union
   select * from cte_total_par_dept
),
cte_consolidation2 as (
  select * from cte_consolidation order by deptid
)
select * from cte_consolidation2
union
select * from cte_total_general
;
```

Mais le résultat obtenu n'est pas tout à fait conforme à notre attente car le tableau final se présente de la façon suivante :

DEPTID	EMPID	NAME	SALARY
10	1	Jones	30000.00
10	2	Hall	35000.00
10	3	Kim	40000.00
20	4	Lindsay	38000.00
10	5	McKeough	42000.00
20	6	Barnes	41000.00
30	7	O'Neil	36000.00
20	8	Smith	34000.00
30	9	Shoeman	33000.00
40	10	Monroe	50000.00
20	11	Zander	52000.00
30	12	Henry	51000.00
10	13	Aaron	54000.00
40	14	Scott	53000.00
30	15	Mills	70000.00
20	16	Goyal	80000.00
10	17	Urbassek	95000.00
10	0		296000.00
20	0		245000.00
30	0		190000.00
40	0		103000.00
0	0	TOTAL ->	834000.00

On voit que les sous-totaux par département ne sont pas placés comme on le souhaitait.

Exemple de report permettant d'obtenir un sous-total par employé et département, plus une rupture par département, et un total général :

```
SELECT deptid, empid, name, sum(salary) as total
FROM FORMATION/MY_EMP
GROUP BY GROUPING SETS((deptid, empid, name), (DEPTID), ())
;
```

Résultat obtenu :

DEPTID	EMPID	NAME	SALARY
10	1	Jones	30000.00
10	2	Hall	35000.00
10	3	Kim	40000.00
10	5	McKeough	42000.00
10	13	Aaron	54000.00
10	17	Urbassek	95000.00
10	null	null	296000.00
20	4	Lindsay	38000.00
20	6	Barnes	41000.00
20	8	Smith	34000.00
20	11	Zander	52000.00
20	16	Goyal	80000.00
20	null	null	245000.00
30	7	O'Neil	36000.00
30	9	Shoeman	33000.00
30	12	Henry	51000.00
30	15	Mills	70000.00
30	null	null	190000.00
40	10	Monroe	50000.00
40	14	Scott	53000.00
40	null	null	103000.00
null	null	null	834000.00

On voit qu'en utilisant la clause GROUPING SETS, on peut assez facilement aboutir au résultat souhaité. Cette technique sera étudiée plus en détail dans le chapitre "CUBE et Grouping Sets".

2.4 Common Table Expression (CTE)

2.4.1 CTE

Les CTE (Common Table Expression) permettent de définir des requêtes relativement sophistiquées, tout en conservant un bon niveau de lisibilité et de maintenabilité. Elles permettent en effet de segmenter, au sein d'une même requête, les différentes étapes permettant d'aboutir au résultat final souhaité.

On peut définir un nombre quasiment illimité de CTE à l'intérieur d'une requête SQL. Voici un exemple théorique :

```
with
temp1 as ( select * from xxx ),
temp2 as ( select * from yyy where ...) ,
temp3 as ( select * from temp1 where ... ) ,
temp4 as ( select * from temp3 where ... ) ,
temp5 as ( select * from temp4 left outer join temp2 on ... where ...)
select * from temp5 ;
```

Appliquons ce principe au dernier exemple du chapitre précédent, qui pour rappel correspond à la requête suivante :

```
select  deptid, empid, name, sum(salary) from  MY_EMP
GROUP BY GROUPING SETS((deptid, empid, name), (DEPTID), ())
union
select  0, 0, 'TOTAL ', sum(salary) from  MY_EMP
;
```

La requête ci-dessus peut être découpée en plusieurs CTE :

```
WITH
-- 1ère CTE permettant d'agréger les salaires sur plusieurs niveaux
cte_1 (deptid, empid, name, sum_salary) as (
      select deptid, empid, name, sum(salary) from  MY_EMP
      GROUP BY GROUPING SETS((deptid, empid, name), (DEPTID), ())
),
-- 2ème CTE permettant d'obtenir le total des salaires sans distinction de niveau
cte_2 (deptid, empid, name, sum_salary) as (
      select  0, 0, 'TOTAL ', sum(salary) from  MY_EMP
),
-- 3ème CTE consolidant le résultat des 2 premières CTE
cte_3 as (
      select deptid, empid, name, sum_salary from cte_1
```

```
      union
      select deptid, empid, name, sum_salary from cte_2
)
-- Requête produisant le résultat final
select * from cte_3
;
```

Quelques remarques :

- Les commentaires fournis dans l'exemple sont donnés à titre indicatif, pour montrer qu'il est possible de documenter une requête et d'en améliorer la lisibilité.

- Dans les 2 premières CTE, le nom des colonnes des jeux de données renvoyés par ces 2 CTE sont renommés au moyen des listes de colonnes indiquées avant le mot clé "as" :

```
      cte_1 (deptid, empid, name, sum_salary) as (...
```

Ce n'est pas obligatoire, comme on peut le voir avec la 3ème CTE, mais l'utilisation de cette technique peut dans certains cas contribuer à améliorer la lisibilité.

Voici un exemple de requête utilisant une CTE (Common Table Expression) pour ajouter un id unique sur chaque ligne (via la fonction rrn()) l'objectif est d'identifier les lignes pour lesquels il y a un doublon sur la colonne globalempcode. Or cette colonne alpha contient des valeurs numériques qui ne sont pas toutes saisies de la même façon (certaines avec un zéro devant, d'autres sans le zéro), d'où application de la fonction integer() pour homogénéïser le format des valeurs, avant de les comparer pour détecter d'éventuels doublons seuls les doublons contenant "ADP" au début de la colonne "localempcode" nous intéressent :

```
with temp1 as (
  select rrn(a) as rrn, integer(a.globalempcode) as int_globalempcode ,
     substr(a.localempcode, 1, 3) as sub_localempcode , a.localempcode,
     (trim(a."LAST NAME") concat ' ' concat trim(a."FIRST NAME")) as name
  from My_LIBRARY.EMPLOYEE a
)
select x.*
from temp1 x
inner join temp1 y
  on  x.int_globalempcode = y.int_globalempcode and x.rrn <> y.rrn
where x.sub_localempcode = 'ADP'
order by x.int_globalempcode , x.rrn
;
```

Voici un autre exemple de requête, plus complexe, utilisant plusieurs CTE pour, dans un premier temps, définir des listes de données à analyser, et ensuite exploiter ces données. En l'occurrence, il s'agit d'une requête qui a été utilisée sur un environnement de production réel, pour identifier des dépendances entre des tables, des vues et des procédures stockées. On ne rentrera pas dans le détail fonctionnel de ce que fait cette requête, on souhaite simplement présenter ici différents usages possibles des CTE :

```
with
-- liste des libraries Base de donnnées de référence à prendre en compte dans
l'analyse
tmplibrary as (
  select x.library  From
       (Values('ZZDWHPRD'), ('ZZDWHTMP')
       ) X(library)
) ,
-- objets complémentaires (tables et vues) à prendre en compte dans l'analyse
tmpcomplements as (
  select x.item_name, x.item_library, x.item_type From
  (Values
       ('ZZFLMT2Y', 'ZZDWHHTH', 'V'),
       ('SUIVI_DT_DERN_CHGT', 'ZZDWHHTH', 'V'),
       ('MITMAS_01', 'ZZFPHSAW', 'V'),
       ('MF_STK_FRN', 'ZZDWHHTH', 'T'),
       ('DIM_ARTICLE', 'ZZDWHHTH', 'T'),
       ('MF_PRI_STD', 'ZZDWHHTH', 'T'),
       ('', '', 'X')
  ) X(item_name, item_library, item_type)
) ,
-- identification des tables utilisées par les procédures stockées
tmptables as (
select item_name as table_name, item_library as table_schema, item_type as table_type
       from tmpcomplements where item_type = 'T'
) ,
-- identification des vues utilisées par les procédures stockées
tmpviews as (
select item_name as table_name, item_library as table_schema, item_type as table_type
       from tmpcomplements where item_type = 'V'
) ,
-- traitement récursif pour l'analyse des dépendances sur les vues DB2
tmpviewtree as (
  SELECT LEVEL, view_name, view_schema,  object_name, object_schema,
      substr(object_type, 1, 1) as object_type
  FROM qsys2.sysviewdep
  where view_schema in ( select library from tmplibrary )
    and object_schema in ( select library from tmplibrary )
  START WITH view_name in ( select table_name from tmpviews )
  CONNECT BY NOCYCLE PRIOR view_name = object_name
) ,
-- liste des vues finalisée
tmptreefinal as (
  select max(level) as max_level ,
  object_name, object_schema, object_type
  from tmpviewtree
  group by object_name, object_schema, object_type
) ,
-- Dernière CTE effectuant la synthèse de l'ensemble des données collectées
synthese as (
  select x.table_name, x.table_schema, x.table_type from (
    select object_name as table_name, object_schema as table_schema,
```

```
      object_type as table_type
    from tmptreefinal
  ) x
  union
    select A1.table_name, A1.table_schema, A1.table_type from tmpviews A1
  union
    select A2.table_name, A2.table_schema, A2.table_type from tmptables A2
  union
    select item_name as table_name, item_library as table_schema, item_type as
table_type
       from tmpcomplements where item_type = 'X'
)
select * from synthese ;
```

On trouvera dans la partie "Etudes de cas" plusieurs exemples de CTE (cf. chapitres "amortissement d'immobilisation" et "détection des périodes d'inactivité").

2.4.2 RCTE (CTE récursive)

Pour information, ce chapitre a été publié sur foothing.net en juin 2014.

Il peut être parfois utile, sous SQL DB2, de générer une série de valeurs (par exemple de 1 à 10), valeurs qui serviront de point d'appui pour effectuer des jointures et récupérer - ou calculer - des données secondaires.

A quoi cela peut-il servir ? Imaginez que vous ayiez besoin d'afficher une statistique de chiffre d'affaires (CA) pour un produit, et cela sur 52 semaines. Supposons que vous n'ayiez pas du tout vendu ce produit une dizaine de semaines dans l'année. Il y a dans ce cas toutes les chances pour que vous n'ayiez aucune ligne dans votre base de données pour ce produit et les semaines en question. Vous aurez donc un trou dans votre stat, ce qui peut être gênant si vous souhaitez voir apparaître toutes les semaines dans votre tableau statistique, y compris celles pour lesquelles vous n'avez pas de CA. Vous me rétorquerez peut être que vous pouvez gérer ça avec une boucle dans un programme RPG ou PHP. Oui certes, mais ce serait tellement plus confortable de générer vos 52 lignes - correspondant à vos 52 semaines - directement via SQL, sans passer par l'utilisation d'un langage complémentaire.

Eh bien, c'est possible, et nous allons voir 2 manières de procéder :

- via une UDTF (User Data Table Function)
- via l'utilisation d'une requête SQL récursive

Voyons tout de suite la 1ère solution, avec l'UDTF suivante :

```
CREATE OR REPLACE FUNCTION MYLIBRARY.GETINCAUTO(VAL_MAX INTEGER)
RETURNS TABLE (VAL_INC INTEGER)
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN
  DECLARE V_NUM INTEGER DEFAULT 1;
  DECLARE GLOBAL TEMPORARY TABLE TMPINCAUTO
     ( VAL_INC INTEGER )
  WITH REPLACE ;
  WHILE V_NUM <= VAL_MAX DO
      INSERT INTO SESSION.TMPINCAUTO (VAL_INC)
      SELECT V_NUM FROM  SYSIBM.SYSDUMMY1 ;
      SET V_NUM = V_NUM + 1;
  END WHILE ;
  RETURN
     SELECT VAL_INC FROM SESSION.TMPINCAUTO ;
END ;
```

Si on veut obtenir une liste de valeurs allant de 1 à 10, on appellera l'UDTF de la façon suivante :

```
SELECT * FROM TABLE (MYLIBRARY.GETINCAUTO(10) ) MYUDTF ;
```

Cette approche consistant à utiliser une UDTF est sans aucun doute la plus facile à appréhender. Après tout, il s'agit ni plus ni moins d'alimenter une table temporaire, et de la renvoyer sous forme de "result set" à la requête appelante.

Attention, il est impératif de donner un nom à votre UDTF dans la requête appelante, comme je l'ai fait dans l'exemple ci-dessus en la nommant "MYUDTF". Mettez le nom que vous voulez, mais si vous n'en mettez pas, ça ne fonctionnera pas.

Voyons maintenant la seconde approche consistant à utiliser une CTE récursive (en anglais : RCTE pour Recursive Common Table Expression).

```
WITH GEN_IDS(NX) AS (
 SELECT 1 as N1 FROM SYSIBM.SYSDUMMY1
UNION ALL
 SELECT NX+1 as N2 FROM GEN_IDS WHERE NX < 10
)
SELECT NX as N FROM GEN_IDS ;
```

Avec la récursivité, la technique est toujours la même : elle consiste à déclarer une CTE (Common Table Expression), qui va utiliser une clause "UNION ALL" entre une première requête qui sert de point de départ, et une seconde requête qui fait appel à la CTE en cours d'exécution (d'où la récursivité). Dans cette seconde requête, il ne faut pas oublier de définir une condition de terminaison (ici : NX < 10), sinon vous risquez de partir dans une boucle sans fin.

Vous pouvez bien sûr encapsuler cette technique récursive dans une UDTF, si vous le jugez utile.

Voici deux exemples d'utilisation que je vous laisse le soin de décortiquer :

```
-- génération d'un calendrier :
WITH
GEN_IDS(NX) AS (
 SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
UNION ALL
 SELECT NX+1 AS N2 FROM GEN_IDS WHERE NX < 30
)
SELECT (CURRENT DATE + NX DAYS) AS N,
  DAYOFWEEK_ISO(CURRENT DATE + NX DAYS) AS DAY_OF_WEEK,
  DAYOFYEAR( CURRENT DATE + NX DAYS ) AS DAY_OF_YEAR,
  WEEK_ISO(CURRENT DATE + NX DAYS) AS WEEK
FROM GEN_IDS
;

-- génération d'identifiants aléatoires (pour constituer un jeu d'essai par exemple)
WITH
GEN_IDS(NX) AS (
 SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
UNION ALL
 SELECT NX+1 AS N2 FROM GEN_IDS WHERE NX < 10
)
SELECT NX AS NUMERO,
```

```
  1000+INT(RAND() * 1000) AS RANDOMID,
  CAST(RAND() * 100 AS DEC(5, 2)) AS RANDOM_PRICE,
  GENERATE_UNIQUE() AS RANDOM_UNIQ_ID ,
  CURRENT_TIMESTAMP + (NX*1000) MICROSECOND AS RANDOM_TSTAMP
FROM GEN_IDS ;
```

Qu'on utilise la technique récursive, ou la technique plus classique faisant appel à l'UDTF présentée au début de l'article, je trouve extrêmement intéressante cette possibilité de générer à la volée via SQL, des séries de valeurs n'existant pas initialement dans la base de données.

On trouvera un exemple d'utilisation de l'UDTF GENINCAUTO() dans la partie "Etudes de cas", chapitre "Amortissement d'immobilisation".

2.5 Pagination

Dans les applications web, on a besoin de savoir gérer un mécanisme de pagination pour afficher des listes de données "par page".

En supposant qu'un jeu de données renvoyé par une requête contienne 100 lignes, et que le système d'affichage gère des pages de 10 lignes maximum, ce sont donc 10 pages qu'il faut savoir afficher alternativement.

Ce type d'affichage avec pagination implique que l'on exécute 2 requêtes SQL :

- un première requête pour comptabiliser le nombre de lignes total du jeu de données considéré
- une seconde requête dédiée à l'affichage page par page, dont à effectuer une pagination à l'intérieur du jeu de données global

L'écran ci-dessous est un exemple de pagination effectuée sur les tables systèmes DB2.

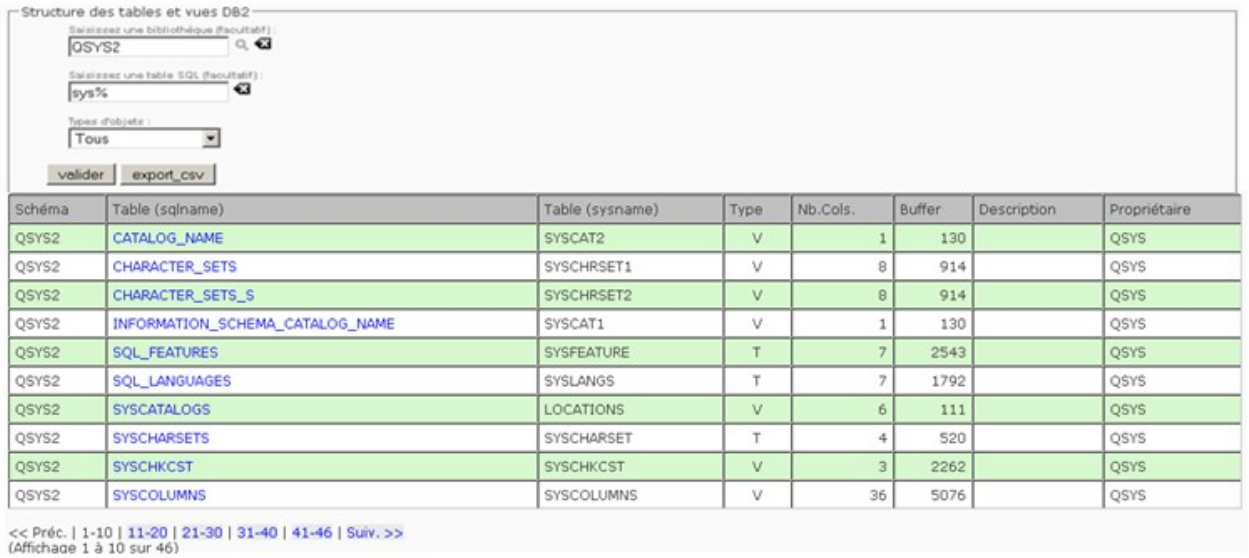


Schéma	Table (sqlname)	Table (sysname)	Type	Nb.Cols.	Buffer	Description	Propriétaire
QSYS2	CATALOG_NAME	SYSCAT2	V	1	130		QSYS
QSYS2	CHARACTER_SETS	SYSCHRSET1	V	8	914		QSYS
QSYS2	CHARACTER_SETS_S	SYSCHRSET2	V	8	914		QSYS
QSYS2	INFORMATION_SCHEMA_CATALOG_NAME	SYSCAT1	V	1	130		QSYS
QSYS2	SQL_FEATURES	SYSFEATURE	T	7	2543		QSYS
QSYS2	SQL_LANGUAGES	SYSLANGS	T	7	1792		QSYS
QSYS2	SYSCATALOGS	LOCATIONS	V	6	111		QSYS
QSYS2	SYSCHARSETS	SYSCHARSET	T	4	520		QSYS
QSYS2	SYSCHKCST	SYSCHKCST	V	3	2262		QSYS
QSYS2	SYSCOLUMNS	SYSCOLUMNS	V	36	5076		QSYS

<< Préc. | 1-10 | 11-20 | 21-30 | 31-40 | 41-46 | Suiv. >>
(Affichage 1 à 10 sur 46)

Cet affichage peut s'effectuer en utilisant une technique « full SQL » ou une technique de type « scroll cursor ».
Nous allons étudier ces deux techniques dans ce chapitre.

ATTENTION : le code suivant ne permet pas de gérer une pagination

```
SELECT * FROM table FETCH FIRST 10 ROWS ONLY
```

Le code SQL ci-dessus renvoie les 10 premières lignes du jeu de données identifié par le SGBD. Cette technique est totalement inadaptée à la gestion de listes avec pagination.

Avec MySQL, vous pouvez gérer facilement une pagination au moyen du code SQL suivant :

```
SELECT * FROM table LIMIT 10, 20
```

(renvoie les lignes 10 à 20 du jeu de données identifié par le SGBD)

Avec DB2, c'est un peu plus compliqué :

- Les données DB2 « chargées » dans la copie d'écran précédente le sont (chargées) via la requête SQL suivante :

```
SELECT A.*
FROM QSYS2/SYSTABLES A
WHERE A.TABLE_SCHEMA = ? AND (A.TABLE_NAME LIKE ? OR A.SYSTEM_TABLE_NAME LIKE ?)
```

- Cette requête ne peut gérer à elle seule la pagination, qui est effectuée ici par plages de 10 lignes. Pour gérer la pagination avec une technique « full SQL », la requête ci-dessus doit être légèrement modifiée, et encapsulée dans une autre requête :

```
SELECT foo.* FROM (
   SELECT row_number() over (ORDER BY TABLE_NAME) as rn,
   A.*
   FROM QSYS2/SYSTABLES A
   WHERE A.TABLE_SCHEMA = ? AND (A.TABLE_NAME LIKE ? OR A.SYSTEM_TABLE_NAME
LIKE ?)
) AS foo
WHERE foo.rn BETWEEN ? AND ?
```

La technique « full SQL » présentée ci-dessus donne de bons résultats sur des tables de taille raisonnable (difficile de donner un chiffre précis car cela dépend beaucoup de la puissance du (des) processeurs(s) de votre serveur IBM i). Mais elle présente quelques défauts :

- Elle est « intrusive » dans le sens où elle nécessite de modifier la requête SQL pour y insérer un certain nombre d'éléments (modification du début du SELECT, inclusion du tri dans la clause OVER…).
- Avec cette technique, DB2 a tendance à s'effondrer sur les tables de très grande taille, donc si vous rencontrez des difficultés avec cette technique, je vous recommande de recourir à la technique du curseur scrollable qui donne de bons résultats dans la plupart des cas.

L'exemple de requête ci-dessous, trouvée dans un framework PHP, utilise la technique de

pagination "full SQL", mais elle présente de grosses lacunes en termes de performances :

```
$limit_sql = "SELECT z2.*
FROM (
SELECT ROW_NUMBER() OVER() AS \"ZEND_DB_ROWNUM\", z1.*
FROM (
" . $votre_requete_sql_ici . "
) z1
) z2
WHERE z2.zend_db_rownum BETWEEN " . ($offset+1) . " AND " . ($offset+$count);
```

Le code ci-dessus présente 2 défauts :

- le tri incorrect, car avec la syntaxe utilisée par le composant base de données du framework, il est impossible d'insérer les colonnes à trier dans la clause OVER
- La pagination s'applique sur le jeu de données renvoyé par la requête encapsulée dans la variable $votre_requete_sql_ici. Elle ne s'applique pas directement sur votre requête, donc attention aux performances sur les tables de grande taille !!!

La technique du curseur scrollable, qui est plus performante sur les tables de grande taille, est relativement verbeuse. Les exemples suivants sont donnés pour le langage PHP. Pour des exemples dans d'autres langages, prière de vous reporter à la documentation officielle de ces langages.

=> Technique du curseur scrollable pour la librairie « ibm_db2 »

```
try {
    $st = db2_prepare($db, $sql, array('cursor' => DB2_SCROLLABLE));
    if (!$st) {
        self::MyDBError($db, 'selectBlock/db2_prepare', $sql, $args);
        $rows = null;
    } else {
        if (!db2_execute($st, $args)) {
            self::MyDBError($db, 'selectBlock/db2_execute', $sql, $args);
            $rows = null;
        } else {
            for (
            $tofetch = $nbl_by_page,
            $row = db2_fetch_assoc($st, $offset);
            $row !== false && $tofetch-- > 0;
            $row = db2_fetch_assoc($st)
            ) {
                $rows [] = $row;
                if ($profiler_status) {
                    $profiler_nb_rows++;
                }
            }
        }
        db2_free_stmt($st);
    }
```

```
            unset($st);
            return $rows;
        } catch (Exception $e) {
            self::MyDBException($db, $e, $sql, $args);
        }
```

=> Technique du curseur scrollable pour la librarie standard PDO (Portable Data Object) :

```
        try {
            $st = $db->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
            $st->execute($args);

            if ($offset > 0) {
                /*
                 * Un bug d'origine inconnu oblige à effectuer un premier
                 * positionnement
                 * sur la ligne n° 0, quand on affiche les offsets > 0
                 * Dans le cas où on affiche l'offset 0, il ne faut surtout pas faire
                 * ce premier positionnement, car il interfère avec celui qui est
                 * effecuté par la boucle d'affichage (for).
                 */
                $lost = $st->fetch(PDO::FETCH_ASSOC, PDO::FETCH_ORI_REL, 0);
            }

            for (
            $tofetch = $nbl_by_page,
            $row = $st->fetch(PDO::FETCH_ASSOC, PDO::FETCH_ORI_REL, $offset);
            $row !== false && $tofetch-- > 0;
            $row = $st->fetch(PDO::FETCH_ASSOC)
            ) {
                $rows [] = $row;
                if ($profiler_status) {
                    $profiler_nb_rows++;
                }
            }

            unset($st);

            return $rows;
        } catch (Exception $e) {
            self::MyDBException($db, $e, $sql, $args);
        }
```

Quelques remarques :

- En plus d'être performante sur les tables de grande taille, la technique du curseur scrollable présente l'avantage de ne pas être intrusive par rapport au code de la requête SQL à exécuter (votre requête SQL ne subit aucune modification).
- A noter : la technique du curseur scrollable fonctionne très bien en PHP sous PDO avec les bases DB2 pour IBM i et DB2 Express C. Mais j'ai détecté un bug récemment, peut être lié à

ma version d'ODBC pour Windows. Après pas mal de tests, j'ai trouvé une solution de contournement qui consiste à effectuer un premier positionnement sur l'offset zéro, quand on est sur l'affichage d'offset de valeur supérieure à zéro.

Essayons d'appliquer le mécanisme de pagination "full SQL" à la requête statistique utilisée dans le chapitre "Common Table Expression -> Statistiques".

Pour gérer une pagination, on rappelle que l'on a besoin dans un premier temps de connaître le nombre de lignes total de la liste à afficher. On peut l'obtenir au moyen de la requête suivante :

```
select count(*) from (
select  DEPTID, empid, name, sum(salary) from  MY_EMP
GROUP BY GROUPING SETS((DEPTID, empid, name), (DEPTID))
union
select  0, 0, 'TOTAL ', sum(salary) from  MY_EMP
) x ;
```

On peut obtenir le même résultat en utilisant une CTE (plus lisible quand la requête atteint un certain niveau de complexité) :

```
with tmpstat as (
select  DEPTID, empid, name, sum(salary) from  MY_EMP
GROUP BY GROUPING SETS((DEPTID, empid, name), (DEPTID))
union
select  0, 0, 'TOTAL ', sum(salary) from  MY_EMP
)
select count(*) from tmpstat ;
```

Pour une pagination « full SQL » sur la base de la requête précédente, on veut dans l'exemple ci-dessous afficher les lignes 11 à 23. A noter qu'il a été nécessaire d'utiliser 2 CTE, du fait que la première requête effectue des agrégations :

```
with tmpstat as (
SELECT
DEPTID, empid, name, sum(salary) from  MY_EMP
GROUP BY GROUPING SETS((DEPTID, empid, name), (DEPTID))
union
select  0, 0, 'TOTAL ', sum(salary) from  MY_EMP
),
tmpstat2 as (
select row_number() over () as rn, tmpstat.* from tmpstat
)
select * from tmpstat2 x where x.rn between 11 and 23
;
```

La pagination "full SQL" offre l'avantage de fonctionner, quel que soit le langage de développement utilisé pour exploiter le jeu de donnnées renvoyé par la requête. Mais elle a pour inconvénient de complexifier les requêtes SQL.

2.6 Identifiants uniques

Pour la gestion d'identifiants uniques dans des tables SQL, on peut recourir à plusieurs techniques :

- identifiant interne en incrémentation automatique :

```
CREATE TABLE MABIB/XSERVICE
( IDMANU INTEGER NOT NULL
CONSTRAINT RQ_IDMANU_PK
PRIMARY KEY,
LIBELLE CHAR(30) NOT NULL WITH DEFAULT,
IDAUTO INTEGER NOT NULL  -- identifiant automatique “interne”
GENERATED ALWAYS AS IDENTITY
( START WITH 1 , INCREMENT BY 1 , CACHE 10 )
UNIQUE
)
INSERT INTO MABIB/XSERVICE (idmanu, libelle) VALUES(1, 'TEST ID 1') ;
INSERT INTO MABIB/XSERVICE VALUES(28, 'TEST ID 28', DEFAULT) ;
INSERT INTO MABIB/XSERVICE VALUES(3, 'TEST ID 3', DEFAULT) ;
INSERT INTO MABIB/XSERVICE VALUES(15, 'TEST ID 15', DEFAULT) ;
INSERT INTO MABIB/XSERVICE (IDMANU) VALUES(4),(33),(10),(9) ;
```

A noter : la colonne IDMANU est un identifiant manuel (avec contrainte de type "clé primaire") destiné à montrer que l’on peut scinder identifiant "métier" et identifiant interne (bonne pratique à utiliser autant que possible).

Technique pour récupérer le dernier ID “généré », à utiliser avec réserves, car ne permet pas de préciser la table dont on souhaite récupérer le dernier identifiant :

```
SELECT IDENTITY_VAL_LOCAL() AS DERNIERID FROM SYSIBM/SYSDUMMY1 ;
```

Cette solution ne permet pas de préciser la table dont on souhaite récupérer le dernier identifiant incrémenté. Elle renvoie le derrnier ID automatique incrémenté pour la dernière insertion effectuée. Il convient donc de l'utiliser avec prudence.

Il faut souligner que, si plusieurs travaux font des insertions en parallèle dans la table, la fonction IDENTITY_VAL_LOCAL() renvoie le dernier ID généré pour le travail en cours.

On peut utiliser une autre technique, qui fonctionne depuis la V6R1. Pour l'illustrer, commençons par créer une table des pays :

```
CREATE TABLE MABIB.LSTPAYS (
```

```
  CODFRA CHAR (3 ) NOT NULL WITH DEFAULT,
  CODISO CHAR (2 ) NOT NULL WITH DEFAULT,
  LIBELLE CHAR (50 ) NOT NULL WITH DEFAULT,
  IDAUTO INTEGER NOT NULL
      GENERATED ALWAYS AS IDENTITY
      ( START WITH 1 , INCREMENT BY 1 , CACHE 2 )
) ;
```

Dans cette technique, il n'y a pas d'ambiguïté concernant l'origine de l'identifiant incrémenté :

```
SELECT X.IDAUTO into :DERNIERID FROM NEW TABLE (
   INSERT INTO MABIB.LSTPAYS (CODFRA, CODISO, LIBELLE)
   VALUES ('FRA', 'FR', 'France')
) X ; -- renvoie 2 après la seconde insertion
```

On peut même effectuer plusieurs insertions et récupérer les valeurs mini et maxi de l'identifiant incrémenté :

```
SELECT min(X.IDAUTO) as min_id, max(X.IDAUTO) as max_id FROM NEW TABLE (
   INSERT INTO MABIB.LSTPAYS (CODFRA, CODISO, LIBELLE)
   VALUES ('FRA', 'FR', 'France'), ('UK', 'UK', 'Grande Bretagne'), ('DE', 'DE',
'Allemagne')
) X ; -- renvoie les valeurs mini et maxi de l'identifiant IDAUTO pour les 3
insertions effectuées
```

Pour la gestion d'identifiants uniques, on peut aussi s'appuyer sur les objets de type "séquence" présentés au chapitre suivant.

2.7 Sequence

La technique utilisant des séquences est intéressante, mais les performances des objets de type séquence (sur DB2 for i) sont décevantes car il s'agit d'objets de type DATA AREA (qui entraînent des problèmes de verrouillages, particulièrement problématiques sur des traitements traitant de gros volumes de données).

```
CREATE SEQUENCE My_Library.seq_item_num
AS INTEGER
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 2000
NO CYCLE
CACHE 20
ORDER
;

SELECT SEQSCHEMA, SEQNAME, INCREMENT, START, CACHE,
NEXTCACHEFIRSTVALUE FROM SYSCAT.SEQUENCES WHERE OWNERTYPE='U';

-- Modification de la valeur minimale acceptée par une séquence :
ALTER SEQUENCE seq_item_num MINVALUE 1000;
-- Redémarrage d'une séquence à la valeur 1001 :
ALTER SEQUENCE seq_item_num RESTART WITH 1001;



-- technique de base pour l'incrémentation d'une séquence DB2
select next value for My_Library.seq_item_num AS LAST_INSERT_ID from
sysibm.sysdummy1;

-- technique de base pour l'incrémentation d'une séquence DB2 à l'intérieur d'un
INSERT
insert into MY_LIBRARY.TABL_METIER (id, libelle)
VALUES
(next value for My_LIBRARY.KSQ_IDTOUR, 'toto') ;

-- technique pour l'incrémentation lors de l'insertion de plusieurs lignes (récupère
le dernier ID généré)
SELECT MAX(ID) FROM FINAL TABLE (
INSERT INTO MABASE.PRXVTE
(ID, DAT_DEB, DAT_FIN, PRX_VTE)
VALUES
(next value for MABASE.SQ_ID_PXVT, '2011-03-01', '2011-12-31', 10.50),
(next value for MABASE.SQ_ID_PXVT, '2011-03-01', '2011-12-31', 12.50)
);


-- techniques "synonymes" pouvant être utilisées dans des programmes ou des
```

```
procédures stockées :
VALUES NEXT VALUE FOR seq_item_num;
VALUES seq_item_num NEXTVAL;

-- syntaxe ci-dessus mise "en situation" dans un INSERT
INSERT INTO item_tbl (seq_item_num) VALUES (NEXT VALUE FOR seq_item_num);
INSERT INTO item_tbl (seq_item_num) VALUES (seq_item_num NEXTVAL);

-- We can use the PREVIOUS VALUE command to get the previous value of the sequence
(intérêt ???):
VALUES PREVIOUS VALUE FOR seq_item_num;
VALUES seq_item_num CURRVAL;
```

2.8 OLAP

La documentation ci-dessous est extraite de la page suivante :

http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.db2_olap.doc/cmdsolapfcn.htm

RANK

Classe les lignes en affectant à chacune un numéro d'ordre. Ce numéro est défini par l'addition du chiffre 1 au nombre de lignes distinctes précédant la
ligne concernée dans le tri. S'il est impossible de déterminer l'ordre relatif de deux lignes ou plus contenant des valeurs de ligne identiques, le même numéro d'ordre leur est affecté. Dans ce cas, la numérotation du classement peut être discontinue. Le Tableau 1 donne un exemple de classement effectué par la fonction RANK sur un ensemble de valeurs de ligne fournies.

La syntaxe de la fonction RANK est généralement la suivante :

```
RANK ( ) OVER (ORDER BY expression-clé-tri  ordre-expression)
```

où expression-clé-tri est l'ensemble des données à classer et ordre-expression est un mot clé, ASC ou DESC, permettant de classer les valeurs d'expression-clé-tri par ordre croissant ou décroissant.

DENSERANK

Classe les lignes en affectant à chacune un numéro d'ordre. Ce numéro est défini par l'addition du chiffre 1 au nombre total de lignes précédant la ligne concernée dans le classement. En conséquence, le classement sera séquentiel, sans discontinuités dans la numérotation. Le Tableau 1 donne un exemple de classement effectué par la fonction DENSERANK sur un ensemble de valeurs de ligne fournies.
La syntaxe de la fonction DENSERANK est généralement la suivante :

```
DENSERANK ( ) OVER (ORDER BY expression-clé-tri ordre-expression)
```

où expression-clé-tri est l'ensemble des données à classer et ordre-expression est un mot clé, ASC ou DESC, permettant de classer les valeurs d'expression-clé-tri par ordre croissant ou décroissant.

ROWNUMBER

Calcule le numéro séquentiel de la ligne en fonction de l'ordre, la première ligne portant le numéro 1. Si la clause ORDER BY n'est pas spécifiée, les lignes sont numérotées arbitrairement.
La syntaxe de la fonction ROWNUMBER est généralement la suivante :

```
ROWNUMBER ( ) OVER ([ORDER BY expression-clé-tri  ordre-expression)
```

où expression-clé-tri est l'ensemble des données à classer et ordre-expression est un mot clé, ASC ou DESC, permettant de classer les valeurs d'expression-clé-tri par ordre croissant ou décroissant. Dans DB2 Cube Views, la source de données de cette fonction doit être une mesure existante, et non une colonne ou un attribut.

```
-- Create sample table
CREATE TABLE MY_LIBRARY.T_SALES
(COUNTRY VARCHAR(20),
CITY VARCHAR(20),
AMOUNT DECIMAL(10,2),
DATE DATE);

-- Create a view that lists total sales for each city.
CREATE VIEW MY_LIBRARY.V_TOTAL_SALES AS
SELECT COUNTRY,
CITY,
SUM(AMOUNT) AS SALES_AMT
FROM MY_LIBRARY.T_SALES GROUP BY COUNTRY, CITY;

-- Insert sample MY_LIBRARY.T_SALES data
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'New York', 200, '2011-10-01');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'New York', 800, '2011-09-01');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'New York', 190, '2011-08-01');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'New York', 230, '2011-10-03');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'California', 200, '2011-10-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'California', 390, '2011-09-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'California', 720, '2011-08-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'California', 110, '2011-10-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Los Angeles', 160, '2011-08-03');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Los Angeles', 500, '2011-10-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Los Angeles', 330, '2011-09-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Los Angeles', 120, '2011-08-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Alaska', 360, '2011-08-03');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Alaska', 600, '2011-10-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Alaska', 450, '2011-09-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('USA', 'Alaska', 720, '2011-08-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'London', 450, '2011-10-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'London', 530, '2011-09-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'London', 790, '2011-08-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'London', 330, '2011-07-05');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'Manchester', 200, '2011-08-01');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'Manchester', 330, '2011-07-01');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'Manchester', 120, '2011-10-03');
INSERT INTO MY_LIBRARY.T_SALES VALUES ('UK', 'Manchester', 640, '2011-09-03');


SELECT CITY, SALES_AMT,
   RANK() OVER (ORDER BY SALES_AMT DESC) AS RANK
FROM MY_LIBRARY.V_TOTAL_SALES;
```

```
SELECT COUNTRY, CITY, SALES_AMT,
   RANK() OVER (PARTITION BY COUNTRY ORDER BY SALES_AMT DESC) AS RANK,
   DENSE_RANK() OVER (PARTITION BY COUNTRY ORDER BY SALES_AMT DESC) AS DENSE_RANK
FROM MY_LIBRARY.V_TOTAL_SALES;

SELECT COUNTRY, CITY, SALES_AMT,
   RANK() OVER (PARTITION BY COUNTRY ORDER BY SALES_AMT DESC) AS RANK
FROM MY_LIBRARY.V_TOTAL_SALES;

SELECT CITY, SALES_AMT,
   RANK() OVER (ORDER BY SALES_AMT DESC) AS RANK
FROM MY_LIBRARY.V_TOTAL_SALES
ORDER BY CITY FETCH FIRST 3 ROWS ONLY;

WITH TOP_CITIES AS
(SELECT CITY, SALES_AMT, RANK() OVER (ORDER BY SALES_AMT) AS RANK
FROM MY_LIBRARY.V_TOTAL_SALES)
SELECT * FROM TOP_CITIES
WHERE RANK <= 3;
```

Requête exploitant la fonction ROWNUMBER() pour identifier précisément chaque ligne, et disposer ces lignes en colonne selon un regroupement sur la colonne COUNTRY :

```
WITH
TEMP_SALES AS (
  SELECT COUNTRY, CITY, SALES_AMT,
  ROWNUMBER() OVER (PARTITION BY COUNTRY ORDER BY SALES_AMT DESC) AS RANK
  FROM MY_LIBRARY.V_TOTAL_SALES
),
TEMP_COUNTRY AS (
  SELECT DISTINCT COUNTRY AS COUNTRY FROM TEMP_SALES
)
SELECT A.COUNTRY, B1.SALES_AMT, B2.SALES_AMT, B3.SALES_AMT, B4.SALES_AMT

FROM TEMP_COUNTRY A
LEFT OUTER JOIN TEMP_SALES B1 ON A.COUNTRY = B1.COUNTRY AND B1.RANK = 1
LEFT OUTER JOIN TEMP_SALES B2 ON A.COUNTRY = B2.COUNTRY AND B2.RANK = 2
LEFT OUTER JOIN TEMP_SALES B3 ON A.COUNTRY = B3.COUNTRY AND B3.RANK = 3
LEFT OUTER JOIN TEMP_SALES B4 ON A.COUNTRY = B4.COUNTRY AND B4.RANK = 4
;
```

2.9 CUBE et Grouping Sets

Nous allons dans ce chapitre étudier plus en détail les techniques DB2 de consolidation statistiques, que nous avions juste survolé dans le chapitre "Introduction aux statistiques". Nous allons réutiliser le jeu de données que nous avions créé dans le chapitre d'introduction.

```
-- agrégations simples sur 1 seule dimension
SELECT DEPTID, SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY DEPTID ;

SELECT MGRID, DEPTID, SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY  MGRID, DEPTID ;


-- GROUPING SETS sur la clause GROUP BY retourne les lignes récapitulatives
-- selon les groupes indiqués dans la clause GROUPING SETS

SELECT DEPTID, MGRID, SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY GROUPING SETS((DEPTID, MGRID ), (DEPTID) )
;

-- la clause ROLLUP clause fournit une ligne récapitulative
-- pour toutes les hiérarchies supérieures données par la clause GROUP BY
-- dans l'exemple, elle fournira une ligne pour :
--  (DEPTID, MGRID)
--  (DEPTID)
--  ( ) cumul général

SELECT DEPTID, MGRID,
SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY ROLLUP(DEPTID, MGRID )
;


-- la clause CUBE fournit en plus une ligne récapitulative pour :
--  (DEPTID, MGRID)
--  (DEPTID)
--  (MGRID)
--  ( ) cumul général

SELECT DEPTID, MGRID,
SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY CUBE(DEPTID, MGRID )
;
```

```
-- La fonction GROUPING peut être utilisée pour différencier une vraie valeur
-- null d'une colonne d'une valeur nulle générée par ROLLUP, CUBE ou
-- GROUPING SETS
-- * La fonction retourne 0 si vraie valeur nulle pour la colonne
-- * La fonction retourne 1 si valeur null générée par ROLLUP, CUBE ou GROUPING SETS

SELECT DEPTID, MGRID, GROUPING(MGRID) as group_mgrid, SUM(SALARY) as sum_salary
FROM MY_LIBRARY.MY_EMP
GROUP BY CUBE(DEPTID, MGRID )
;
```

DEPTID	MGRID	GROUP_MGRID	SUM_SALARY
10	10	0	105000
10	11	0	42000
10	15	0	54000
10	null	0	95000
10	null	1	296000
20	10	0	38000
20	11	0	41000
20	12	0	34000
20	16	0	52000
20	17	0	80000
20	null	1	245000
30	12	0	69000
30	16	0	51000
30	17	0	70000
30	null	1	190000
40	15	0	50000
40	16	0	53000
40	null	1	103000
null	null	1	834000
null	null	0	95000
null	10	0	143000
null	11	0	83000
null	12	0	103000
null	15	0	104000
null	16	0	156000

```
ATTENTION : on peut avoir des sous-totalisations parasites déclenchées par la
présence de NULL dans certaines colonnes (comme MGRID par exemple). Pour contourner
le problème, on pourrait faire une sous-requête préalable remplaçant les NULL par une
valeur par défaut (comme zéro), comme dans l'exemple ci-dessous, mais le GROUP BY
ROLLUP permet d'obtenir le même résultat plus efficacement (voir exemple plus haut) :

SELECT x.DEPTID, x.MGRID, SUM(x.SALARY) as sum_salary
```

```
FROM
(
SELECT DEPTID, ifnull(MGRID, 0) as MGRID,SALARY
FROM GJARRIGE3.MY_EMP
) x
GROUP BY CUBE(x.DEPTID, x.MGRID )
;
```

DEPTID	MGRID	SUM_SALARY
10	0	95000
10	10	105000
10	11	42000
10	15	54000
10	null	296000
20	10	38000
20	11	41000
20	12	34000
20	16	52000
20	17	80000
20	null	245000
30	12	69000
30	16	51000
30	17	70000
30	null	190000
40	15	50000
40	16	53000
40	null	103000
null	null	834000
null	0	95000
null	10	143000
null	11	83000
null	12	103000
null	15	104000
null	16	156000
null	17	150000

2.10 Cryptage de données

Création d'une table contenant une colonne cryptée :

```
CREATE TABLE MY_LIBRARY.TEST_ENCRYPTION
(ID INTEGER,
NAME VARCHAR(30),
CONFIDENTIAL_INFO VARCHAR(100) FOR BIT DATA);
```

Cette déclaration définit le mot de passe de cryptage dans un registre spécial lié à des fonctions de chiffrement et de déchiffrement. Cette déclaration n'est pas sous le contrôle de transaction et n'est pas lié à l'authentification DB2.

```
SET ENCRYPTION PASSWORD 'myPassword';
```

La déclaration suivante utilise un mot de passe de cryptage du registre spécial, qui dans notre cas doit être "myPassWord".

```
INSERT INTO MY_LIBRARY.TEST_ENCRYPTION VALUES (101, 'Daniel K', ENCRYPT('ACSX1001'));
```

Nous pouvons également fournir le mot de passe explicitement tel que :

```
INSERT INTO MY_LIBRARY.TEST_ENCRYPTION VALUES (102, 'Daniel K', ENCRYPT('ACSX1001',
'Daniel K'));
```

La fonction ENCRYPT fournit également le moyen de définir un "conseil de mot de passe", qui pourra être utile si le propriétaire de la donnée oublie le mot de passe :

```
INSERT INTO MY_LIBRARY.TEST_ENCRYPTION VALUES (103, 'Daniel K', ENCRYPT('ACSX1001',
'Daniel K', 'Name'));

SELECT * FROM MY_LIBRARY.TEST_ENCRYPTION;


SELECT ID, NAME, DECRYPT_CHAR(CONFIDENTIAL_INFO) FROM MY_LIBRARY.TEST_ENCRYPTION
WHERE ID = 101;

SELECT ID, NAME, DECRYPT_CHAR(CONFIDENTIAL_INFO, 'Daniel K') FROM
MY_LIBRARY.TEST_ENCRYPTION WHERE ID IN (102, 103);

SELECT GETHINT(CONFIDENTIAL_INFO) FROM MY_LIBRARY.TEST_ENCRYPTION;
```

2.11 Hiérarchie récursive

Exemples trouvés sur la documentation IBM :
http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/index.jsp?topic=%2Fcom.ibm.db2.luw.apdv.porting.doc%2Fdoc%2Fr0053104.html

```
CREATE TABLE MY_EMP(
  EMPID  INTEGER NOT NULL PRIMARY KEY,
  NAME   VARCHAR(10),
  SALARY DECIMAL(9, 2),
  MGRID  INTEGER);

INSERT INTO MY_EMP VALUES ( 1, 'Jones',    30000, 10);
INSERT INTO MY_EMP VALUES ( 2, 'Hall',     35000, 10);
INSERT INTO MY_EMP VALUES ( 3, 'Kim',      40000, 10);
INSERT INTO MY_EMP VALUES ( 4, 'Lindsay',  38000, 10);
INSERT INTO MY_EMP VALUES ( 5, 'McKeough', 42000, 11);
INSERT INTO MY_EMP VALUES ( 6, 'Barnes',   41000, 11);
INSERT INTO MY_EMP VALUES ( 7, 'O''Neil',  36000, 12);
INSERT INTO MY_EMP VALUES ( 8, 'Smith',    34000, 12);
INSERT INTO MY_EMP VALUES ( 9, 'Shoeman',  33000, 12);
INSERT INTO MY_EMP VALUES (10, 'Monroe',   50000, 15);
INSERT INTO MY_EMP VALUES (11, 'Zander',   52000, 16);
INSERT INTO MY_EMP VALUES (12, 'Henry',    51000, 16);
INSERT INTO MY_EMP VALUES (13, 'Aaron',    54000, 15);
INSERT INTO MY_EMP VALUES (14, 'Scott',    53000, 16);
INSERT INTO MY_EMP VALUES (15, 'Mills',    70000, 17);
INSERT INTO MY_EMP VALUES (16, 'Goyal',    80000, 17);
INSERT INTO MY_EMP VALUES (17, 'Urbassek', 95000, NULL);


-- 1er exemple : La requête suivante retourne tous les employés travaillant pour
Goyal, ainsi que des informations supplémentaires, notamment la hiérarchie de chaque
employé

SELECT NAME,
        LEVEL,
        SALARY,
        CONNECT_BY_ROOT NAME AS ROOT,
        SUBSTR(SYS_CONNECT_BY_PATH(NAME, ':'), 1, 25) AS CHAIN
   FROM MY_EMP
   START WITH NAME = 'Goyal'
   CONNECT BY PRIOR EMPID = MGRID
   ORDER SIBLINGS BY SALARY;
;

-- résultat obtenu

NAME       LEVEL       SALARY      ROOT  CHAIN
  ---------- ----------- ----------- ----- ---------------
  Goyal               1    80000.00 Goyal :Goyal
```

```
  Henry                2    51000.00 Goyal :Goyal:Henry
  Shoeman              3    33000.00 Goyal :Goyal:Henry:Shoeman
  Smith                3    34000.00 Goyal :Goyal:Henry:Smith
  O Neil               3    36000.00 Goyal :Goyal:Henry:O Neil
  Zander               2    52000.00 Goyal :Goyal:Zander
  Barnes               3    41000.00 Goyal :Goyal:Zander:Barnes
  McKeough             3    42000.00 Goyal :Goyal:Zander:McKeough
  Scott                2    53000.00 Goyal :Goyal:Scott
```

```
-- 2ème exemple : Retour de la structure organisationnelle de la table DEPARTMENT, à
partir du code département transmis dans la clause WHERE.

TODO : exemple ci-dessous à revoir (compléter avec derniers tests effectués)

SELECT LEVEL, CAST(SPACE((LEVEL - 1) * 4) || '/' || DEPTNAME
       AS VARCHAR(40)) AS DEPTNAME
     FROM DEPARTMENT
     START WITH DEPTNO = 'A00'
     CONNECT BY NOCYCLE PRIOR DEPTNO = ADMRDEPT
;

-- résultat obtenu

LEVEL       DEPTNAME
----------- ----------------------------------------
          1 /SPIFFY COMPUTER SERVICE DIV.
          2     /PLANNING
          2     /INFORMATION CENTER
          2     /DEVELOPMENT CENTER
          3         /MANUFACTURING SYSTEMS
          3         /ADMINISTRATION SYSTEMS
          2     /SUPPORT SERVICES
          3         /OPERATIONS
          3         /SOFTWARE SUPPORT
          3         /BRANCH OFFICE F2
          3         /BRANCH OFFICE G2
          3         /BRANCH OFFICE H2
          3         /BRANCH OFFICE I2
          3         /BRANCH OFFICE J2
```

On verra au chapitre relatif aux tables systèmes, et en particulier dans le chapitre décrivant la table SYSVIEWDEP, qu'il est possible de décliner le principe de la recherche récursive pour identifier toutes les dépendances d'une vue DB2.

2.12 Table pivot

On peut utiliser la table pivot SYSIBM.SYSDUMMY1 pour différents usages :

```
-- dans une CTE pour calculer une date une seule fois
with TMPDATSYS AS (
    SELECT YEAR(CURRENT_DATE)*10000 + MONTH(CURRENT_DATE)*100 + DAY(CURRENT_DATE) AS
DATSYS FROM SYSIBM/SYSDUMMY1
),
-- et ainsi pouvoir la réutiliser en plusieurs endroits :
WHERE CUDATE <= (SELECT DATSYS FROM TMPDATSYS) ;

-- pour une insertion de variable programme dans une table
insert into MF_MITTRA_DAT (DATTIMTRT)
select
  V_DATTIMTRT
from SYSIBM/SYSDUMMY1 ;

-- dans un MERGE, quand on n'a pas besoin d'une véritable table d'origine
MERGE INTO My_LIBRARY.testmerge A
 USING (SELECT * FROM SYSIBM.SYSDUMMY1) B
 ON A.macle = :VARPGM1
 WHEN MATCHED THEN
 UPDATE SET
  a.codea = :VARPGM1 ,
  a.coden = :VARPGM2 + 1
 WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES(:VARPGM1, :VARPGM2 , 1 )
;
```

On peut aussi utiliser la table système QSQPTABL comme table pivot, mais son usage n'est pas recommandé car il s'agit d'une table spécifique à l'IBMi, tandis que SYSDUMMY1 est une table standard de DB2 (toutes plateformes confondues). Pour une bonne portabilité du code SQL sur les différentes plateformes DB2, on recommande donc d'utiliser SYSDUMMY1.

2.13 MERGE

L'ordre MERGE est une avancée majeure du SQL DB2, apparu avec la V7R1 :

```
-- drop table My_LIBRARY.testmerge ;
create table My_LIBRARY.testmerge (
macle char(10) default null,
codea char(10) default null,
coden integer default null
);

-- drop table My_LIBRARY.testmerge2 ;
create table My_LIBRARY.testmerge2 (
macle char(10) default null,
codea char(10) default null,
coden integer default null
);
```

Plutôt que d'alimenter la table testmerge via un INSERT, autant le faire directement avec un premier MERGE. Dans ce cas de figure, les données ne viennent pas d'une table source mais de valeurs fixées "en dur", du coup la clause USING ne sert à rien, mais comme elle n'est pas optionnelle, on l'alimente avec la table pivot SYSDUMMY1 :

```
MERGE INTO My_LIBRARY.testmerge A
 USING (SELECT * FROM SYSIBM.SYSDUMMY1) B
 ON A.macle = 'CLE1'
 WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES ( 'CLE1' , 'A1' , 1 )
;
```

Insérons une seconde ligne dans la table testmerge avec la même technique. On sait que l'on ne passera pas dans le bloc WHEN MATCHED, mais on l'a mis ici à titre de premier exemple de cette syntaxe :

```
MERGE INTO My_LIBRARY.testmerge A
 USING (SELECT * FROM SYSIBM.SYSDUMMY1) B
 ON A.macle = 'CLE2'
 WHEN MATCHED THEN
 UPDATE SET
  a.codea = 'A2' ,
  a.coden = a.coden + 1
 WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES ( 'CLE2' , 'A2' , 2 )
;
```

Vérifions le contenu de la table testmerge :

```
select * from My_LIBRARY.testmerge ;
```

Vous pouvez vous amuser à réexécuter les 2 requêtes ci-dessus, et notamment la seconde en modifiant les valeurs fixées dans l'UPDATE, pour voir si vos modifications sont bien prises en compte.

Alimentons maintenant la table testmerge2 à partir du contenu de la table testmerge :

```
MERGE INTO My_LIBRARY.testmerge2 a
 USING (SELECT macle , codea , coden FROM My_LIBRARY.testmerge ) b
 ON a.macle = b.macle
 WHEN MATCHED THEN
 UPDATE SET
  a.codea = b.codea ,
  a.coden = a.coden + b.coden
 WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES ( b.macle , b.codea , b.coden )
;
```

Vérifions le contenu de testmerge2 :

```
select * from My_LIBRARY.testmerge2 ;
```

Exemple avec des conditions dans le WHEN MATCHED, pour effectuer soit une suppression, comme dans l'exemple ci-dessous, où l'on supprime la ligne dans la table « testmerge2 » si elle existe déjà et a pour valeur « A1 » dans la colonne « codea ».

Pour compléter l'exemple, j'ai fait en sorte que des valeurs « en dur » soient affectées à la ligne mise à jour dans le cas où une ligne a la colonne « codea » fixée à « A2 ».

```
MERGE INTO My_LIBRARY.testmerge2 a
 USING (SELECT macle , codea , coden FROM My_LIBRARY.testmerge ) b
 ON a.macle = b.macle
 WHEN MATCHED and a.codea = 'A1'  THEN
    DELETE
 WHEN MATCHED and a.codea = 'A2'  THEN
 UPDATE SET
  a.codea = 'X2' ,
  a.coden = 9999
 WHEN MATCHED and a.codea <> 'A1'  and a.codea <> 'A2'    THEN
 UPDATE SET
  a.codea = b.codea ,
  a.coden = a.coden + b.coden
WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES ( b.macle , b.codea , b.coden )
```

```
;
```

Autre cas de figure qui peut être utilisé dans une procédure stockée (PL/SQL), un script PHP ou un programme RPG : on a des données de variables du programme (ou de la procédure stockée) et on veut insérer ces données dans une table sans passer par une table « source », on peut dans ce cas utiliser la table pivot SYSDUMMY1 comme table source, et écrire ceci (les variables programmes sont reconnaissables au fait qu'elles sont préfixées par « : ») :

```
MERGE INTO My_LIBRARY.testmerge A
 USING (SELECT * FROM SYSIBM.SYSDUMMY1) B
 ON A.macle = :VARPGM1
 WHEN MATCHED THEN
 UPDATE SET
  a.codea = :VARPGM1 ,
  a.coden = :VARPGM2 + 1
 WHEN NOT MATCHED THEN
 INSERT ( a.macle , a.codea , a.coden )
 VALUES (:VARPGM1, :VARPGM2 , 1 )
;
```

Si l'on souhaite développer en SQL dynamique (technique utilisable en RPG, PL/SQL et PHP), on peut écrire ceci :

```
sql = "MERGE INTO qtemp.testmerge A
USING (SELECT * FROM SYSIBM.SYSDUMMY1) B
ON A.macle1 = ?
WHEN MATCHED THEN
UPDATE SET
 a.codea = ? ,
 a.coden = ?
WHEN NOT MATCHED THEN
INSERT ( a.macle , a.codea , a.coden )
VALUES ( ? , ? , ? )" ;

 EXEC SQL PREPARE s1 FROM :sql;
 EXEC SQL EXECUTE s1 USING :VARPGM1 , :VARPGM1 , :VARPGM2 :VARPGM1 , :VARPGM2 , 1
;
```

Mais la technique ci-dessus présente l'inconvénient d'obliger le développeur à disséminer les points d'interrogation dans les différents blocs WHEN MATCHED, WHEN NOT MATCHED, en les démultipliant.

Une approche plus pragmatique consiste à déclarer les variables programmes au sein d'une table virtuelle, déclarée au sein de la clause USING, comme dans l'exemple ci-dessous, pris dans la documentation officielle de DB2 pour Z/OS :

```
sql = "MERGE INTO employee AS t
 USING TABLE(VALUES(
  CAST (? AS CHAR(6)),
```

```
  CAST (? AS VARCHAR(12)),
  CAST (? AS CHAR(1)),
  CAST (? AS VARCHAR(15)),
  CAST (? AS SMALLINT),
  CAST (? AS INTEGER))
 ) s (empno, firstnme, midinit, lastname, edlevel, salary)
 ON t.empno = s.empno
 WHEN MATCHED THEN
    UPDATE SET
      salary = s.salary
 WHEN NOT MATCHED THEN
    INSERT
       (empno, firstnme, midinit, lastname, edlevel, salary)
       VALUES (s.empno, s.firstnme, s.midinit, s.lastname, s.edlevel, s.salary)";
 EXEC SQL PREPARE s1 FROM :sql;
 EXEC SQL EXECUTE s1 USING '000420', 'SERGE', 'K', 'FIELDING', 18, 39580
;
```

Cette approche évite de disséminer, et surtout de démultiplier, les points d'interrogation au sein de la requête. On obtient ainsi une requête plus lisible et plus maintenable, que dans l'exemple précédent.

Le seul inconvénient que l'on pourrait trouver à l'utilisation du MERGE, c'est l'impossibilité de récupérer un « result set » des données impactées par une instruction de mise à jour. En effet, avec une instruction de type DELETE, INSERT ou UPDATE, il est possible de récupérer le "result set" relatif aux données mises à jour, en utilisant la syntaxe SQL DB2 suivante :

```
SELECT * FROM FINAL TABLE (INSERT ...)
```

Cette technique utilisant la clause « FINAL TABLE » peut être très utile pour récupérer la liste des identifiants créés par un INSERT SQL, ou tout simplement le dernier identifiant généré par une série d'INSERTs, on peut dans ce cas écrire une requête du genre :

```
SELECT MAX(ID) FROM FINAL TABLE (INSERT ...)
```

Le MERGE SQL ne peut pas être combiné avec la clause « FINAL TABLE », donc on ne peut pas récupérer le result set résultant d'un MERGE. C'est néanmoins un inconvénient mineur au vu des possibilités qu'apporte le MERGE.

2.14 Dates d'effet

La manipulation de données soumises à une date d'effet (appelée aussi "date de valeur" ou "date d'application") est une opération un peu compliquée en SQL. Démonstration par l'exemple :

```
-- Création d'une table des articles
DROP TABLE My_LIBRARY.PRXART ;

CREATE TABLE My_LIBRARY.PRXART (
CODSOC CHAR ( 3) NOT NULL WITH DEFAULT,
CODART DEC (8 , 0) NOT NULL WITH DEFAULT,
LIBART CHAR(30) NOT NULL WITH DEFAULT
)
;

INSERT INTO My_LIBRARY.PRXART (CODSOC, CODART, LIBART)
VALUES
('001', 1, 'ARTICLE 1'),
('001', 2, 'ARTICLE 2'),
('001', 3, 'ARTICLE 3')
;

-- Création d'une table des prix de vente
DROP TABLE My_LIBRARY.PRXVTE ;

CREATE TABLE My_LIBRARY.PRXVTE (
CODSOC CHAR ( 3) NOT NULL WITH DEFAULT,
CODART DEC (8 , 0) NOT NULL WITH DEFAULT,
DATDEB DATE NOT NULL WITH DEFAULT,
DATFIN DATE NOT NULL WITH DEFAULT,
PXVENT DEC (11 , 2) NOT NULL WITH DEFAULT);


CREATE INDEX My_LIBRARY.PRXVTEL1 ON My_LIBRARY.PRXVTE
(CODSOC ASC, CODART ASC, DATDEB ASC) ;


INSERT INTO My_LIBRARY.PRXVTE (CODSOC, CODART, DATDEB, DATFIN, PXVENT)
VALUES
('001', 1, '2009-01-01', '2009-01-31', 1.5),
('001', 1, '2009-01-01', '2009-02-28', 1.8),
('001', 1, '2009-03-01', '2009-03-31', 2.5),
('001', 1, '2009-03-01', '2009-05-31', 2.8),
('001', 1, '2009-06-01', '2009-06-01', 3),
('001', 2, '2009-01-01', '2009-12-31', 2.5),
('001', 3, '2009-01-01', '2009-02-28', 2.9),
('001', 3, '2009-03-01', '2009-03-31', 3.5),
('001', 3, '2009-03-01', '2009-05-31', 3.9),
('001', 3, '2009-06-01', '2009-06-01', 4)
;
```

```
-- En théorie, récupérer une donnée avec date d'effet consiste à exécuter une sous-
requête
--  qui permettra d'identifier la ligne de tarif valide pour la période considérée

select art.*, tar_deb.DATDEB, tar_deb.DATFIN, tar_deb.PXVENT, rrn(tar_deb) as
rrn_tar_deb
from My_LIBRARY.PRXART art
left outer join My_LIBRARY.PRXVTE tar_deb
on art.CODSOC = tar_deb.CODSOC and art.CODART = tar_deb.CODART and tar_deb.DATDEB =
(   SELECT max(x.DATDEB) FROM My_LIBRARY.PRXVTE x
     WHERE art.CODSOC = x.CODSOC and art.CODART = x.CODART
       and x.DATDEB <= (select datref from cte_datref) and x.DATFIN >=
             (select datref from cte_datref)
)
```

CODSOC	CODART	LIBART	DATDEB	DATFIN	PXVENT	RRN_TAR_DEB
1	1	ARTICLE 1	01/03/2009	31/03/2009	2,5	3
1	1	ARTICLE 1	01/03/2009	31/05/2009	2,8	4
1	2	ARTICLE 2	01/01/2009	31/12/2009	2,5	6
1	3	ARTICLE 3	01/04/2009	30/07/2009	3,7	10

```
-- Mais on voit qu'en réalité on peut obtenir des doublons si, comme c'est le cas
avec notre jeu
-- d'essai, on a plusieurs lignes de tarif avec la même date de début pour le même
article.
-- Si la base tarifaire autorise ce cas de figure, il faut travailler à la fois sur
la date de
-- début et sur la date de fin. On pourrait dès lors penser que 2 sous-requêtes
permettraient
-- de régler le problème, mais on va voir que ce n'est pas si simple :

-- Technique tentante mais qui ne fonctionne pas
with datref as (select '2009-04-01' as datref from sysibm.sysdummy1)
select art.*, tar_deb.DATDEB, tar_deb.DATFIN, tar_deb.PXVENT, rrn(tar_deb) as
rrn_tar_deb, rrn(tar_fin) as rrn_tar_fin
from My_LIBRARY.PRXART art
left outer join My_LIBRARY.PRXVTE tar_deb
on art.CODSOC = tar_deb.CODSOC and art.CODART = tar_deb.CODART and tar_deb.DATDEB =
(   SELECT max(x.DATDEB) FROM My_LIBRARY.PRXVTE x
     WHERE art.CODSOC = x.CODSOC and art.CODART = x.CODART
       and x.DATDEB <= (select datref from datref) and x.DATFIN >= (select datref
from datref)
)
left outer join My_LIBRARY.PRXVTE tar_fin
on art.CODSOC = tar_fin.CODSOC and art.CODART = tar_fin.CODART and tar_fin.DATFIN =
(   SELECT max(x.DATFIN) FROM My_LIBRARY.PRXVTE x
     WHERE art.CODSOC = x.CODSOC and art.CODART = x.CODART
       and x.DATDEB <= (select datref from datref) and x.DATFIN >= (select datref
from datref)
```

```
)
;

-- Fonctionne correctement sous DB2 for i (utilise le numéro relatif
d'enregistrement)
with datref as (select '2009-04-01' as datref from sysibm.sysdummy1)
select art.*, tar_deb.DATDEB, tar_deb.DATFIN, tar_deb.PXVENT
from My_LIBRARY.PRXART art
left outer join My_LIBRARY.PRXVTE tar_deb
on rrn(tar_deb) =
(   SELECT rrn(x) FROM My_LIBRARY.PRXVTE x
     WHERE art.CODSOC = x.CODSOC and art.CODART = x.CODART
       and x.DATDEB <= (select datref from datref) and x.DATFIN >= (select datref
from datref)
     ORDER BY x.DATDEB DESC, x.DATFIN ASC
     FETCH FIRST 1 ROW ONLY
)
;

-- Résultat obtenu :
```

CODSOC	CODART	LIBART	DATDEB	DATFIN	PXVENT	RRN_TAR_DEB
1	1	ARTICLE 1	01/03/2009	31/05/2009	2,8	4
1	2	ARTICLE 2	01/01/2009	31/12/2009	2,5	6
1	3	ARTICLE 3	01/04/2009	30/07/2009	3,7	10

```
-- Fonctionne également sous DB2 for i (sans passer par rrn)
with datref as (select '2009-04-01' as datref from sysibm.sysdummy1)
select art.*, tar_deb.DATDEB, tar_deb.DATFIN, tar_deb.PXVENT, rrn(tar_deb) as
rrn_tar_deb
from My_LIBRARY.PRXART art
left outer join My_LIBRARY.PRXVTE tar_deb
on art.CODSOC = tar_deb.CODSOC and art.CODART = tar_deb.CODART and (tar_deb.DATDEB,
tar_deb.DATFIN) =
(   SELECT x.DATDEB, x.DATFIN FROM My_LIBRARY.PRXVTE x
     WHERE art.CODSOC = x.CODSOC and art.CODART = x.CODART
       and x.DATDEB <= (select datref from datref) and x.DATFIN >= (select datref
from datref)
     ORDER BY x.DATDEB DESC, x.DATFIN ASC
     FETCH FIRST 1 ROW ONLY
)
;
```

2.15 UDF

UDF (User Defined Function) :

Exemples de fonctions de conversion de date pouvant être réutilisées dans une application "métier" :

```
-- Conversion d'une date alpha en véritable date SQL

CREATE OR REPLACE FUNCTION My_LIBRARY.CVT_ALP_2_DATE(DATE_ENT CHAR(10))
RETURNS DATE
LANGUAGE SQL
SPECIFIC  MY_LIBRARY/CVTALP2DAT
DETERMINISTIC
CONTAINS SQL
RETURN
CASE WHEN DATE_ENT is null or trim(DATE_ENT) = ''
     THEN CURDATE()
     ELSE DATE(TO_DATE(DATE_ENT, 'YYYY-MM-DD'))
END
;

COMMENT ON SPECIFIC FUNCTION MY_LIBRARY.CVTALP2DAT
      IS 'Conversion de date au format Alpha en date SQL' ;

-- Conversion d'une date SQL en date numérique (SAMJ)

CREATE OR REPLACE FUNCTION My_LIBRARY.CVT_DATE_2_NUM (
    DAT_ENT DATE )
RETURNS decimal(8, 0)
LANGUAGE SQL
SPECIFIC MY_LIBRARY/CVTDAT2NUM
DETERMINISTIC
CONTAINS SQL
CALLED ON NULL INPUT
RETURN
CASE WHEN DATE_ENT is null
     THEN 0
     ELSE YEAR(DAT_ENT) * 10000 + MONTH(DAT_ENT) * 100 + DAY(DAT_ENT)
END
;

COMMENT ON SPECIFIC FUNCTION MY_LIBRARY.CVTDAT2NUM
      IS 'Conversion de date SQL en décimal(8.0) ' ;

-- Conversion d'une date numérique (SAMJ) en date SQL

CREATE FUNCTION My_LIBRARY.CVT_NUM_2_DATE(
    DATE_ENT DECIMAL(8, 0)
)
RETURNS DATE
LANGUAGE SQL
```

```
SPECIFIC  MY_LIBRARY/CVTNUM2DAT
DETERMINISTIC
CONTAINS SQL
RETURN
CASE WHEN DATE_ENT IS NULL OR DATE_ENT = 0
THEN NULL
ELSE
  DATE(SUBSTR(CHAR(DATE_ENT) , 1, 4) CONCAT '-' CONCAT
       SUBSTR(CHAR(DATE_ENT) , 5, 2) CONCAT '-' CONCAT
       SUBSTR(CHAR(DATE_ENT) , 7, 2) )
END
;

COMMENT ON SPECIFIC FUNCTION MY_LIBRARY.CVTNUM2DAT
      IS 'Conversion de décimal(8.0) en date SQL' ;
```

Attention : un des paramètres importants à noter dans la définition des fonctions est le mot clé DETERMINISTIC qui indique que, si une fonction est appelée plusieurs fois avec les mêmes valeurs de paramètres en entrée, alors elle doit renvoyer la valeur calculée précédemment sans réexécuter le code de la fonction. Les appels suivants d'une fonction sont donc plus rapides. Si DETERMINISTIC est pertinent pour la conversion de dates, il ne l'est plus lorsqu'il s'applique à des données susceptibles d'évoluer dans le temps. Dans ce cas de figure, il faut déclarer la fonction en "NOT DETERMINISTIC", de manière à ce qu'elle s'exécute intégralement à chaque appel.

Exemples d'utilisation :

```
-- utilisation dans une requête SQL
select CVT_NUM_2_DAT2(UCORDT) as DAT_UCORDT) from matable ;

-- utilisation dans une procédure stockée (PL/SQL)
SET V_DATE = CVT_NUM_2_DAT2(20140709) ;
```

2.16 UDTF

UDTF (User Data Table Function)

objectif d'une UDTF : renvoyer sous forme d'une table "virtuelle" des données qui peuvent être de provenance diverses (valeurs en dur, données au format XML, etc...). Une UDTF peut être utilisée directement dans une clause SELECT, ou en jointure avec d'autres tables.

```
-- requête utilisée à l'intérieur de l'UDTF pour générer une table "à la volée"
Select X.SALCODE, X.SALNAME
 From (Values
         ('ALL','All Customers'),
         ('WEB','Web Customers')
      ) X(SALCODE, SALNAME)
 Where X.SALCODE = UPPER('All');


-- Création d'un UDTF encapsulant la requête ci-dessus
Create or replace Function MY_LIBRARY.TstSales( INPUTCODE varchar(3) )
Returns Table (SALCODE varchar(3),
               SALName varchar(20))
Language SQL
Begin
    Return
Select X.SALCODE, X.SALNAME
 From table (Values('ALL', 'All Customers'),
       ('WEB', 'Web Customers')) X(SALCODE, SALNAME)
       Where X.SALCODE = UPPER(INPUTCODE)
;
End
;

-- fonctionne dès lors qu'on n'oublie pas d'attribuer un nom à la table générée
-- par l'appel de la fonction (comme "myudtf" par exemple) :
select * from table (MY_LIBRARY.TSTSALES('All') ) myudtf ;


-- code obtenu après regénération via System i Navigator :

CREATE OR REPLACE FUNCTION MY_LIBRARY/TSTSALES (
       INPUTCODE VARCHAR(3) )
       RETURNS TABLE (
       SALCODE VARCHAR(3) ,
       SALNAME VARCHAR(20) )
       LANGUAGE SQL
       SPECIFIC MY_LIBRARY/TSTSALES
       NOT DETERMINISTIC
       READS SQL DATA
       CALLED ON NULL INPUT
       SET OPTION  ALWBLK = *ALLREAD ,
```

```
       ALWCPYDTA = *OPTIMIZE ,
       COMMIT = *NONE ,
       DECRESULT = (31, 31, 00) ,
       DFTRDBCOL = *NONE ,
       DYNDFTCOL = *NO ,
       DYNUSRPRF = *USER ,
       SRTSEQ = *HEX
       BEGIN
RETURN
SELECT X . SALCODE , X . SALNAME
FROM TABLE ( VALUES ( 'ALL' , 'All Customers' ) ,
( 'WEB' , 'Web Customers' ) ) X ( SALCODE , SALNAME )
WHERE X . SALCODE = UPPER ( INPUTCODE )
;
END  ;

GRANT ALTER , EXECUTE
ON SPECIFIC FUNCTION MY_LIBRARY/TSTSALES
TO CENTRAL ;

GRANT EXECUTE
ON SPECIFIC FUNCTION MY_LIBRARY/TSTSALES
TO PUBLIC ;
```

2.17 XML

2.17.1 SQL vers XML

Exemples de génération de données au format XML, à partir de données SQL.

```
-- Création d'une table exemple :

CREATE TABLE MY_LIBRARY.HQEMPLOYEE (
EMPID INTEGER NOT NULL PRIMARY KEY,
FIRSTNAME VARCHAR (10),
LASTNAME VARCHAR (10),
SALARY DECIMAL (8, 2),
MGRID INTEGER);

INSERT INTO MY_LIBRARY.HQEMPLOYEE VALUES
(1, 'John', 'Brett', 66000, 6),
(2, 'Peter', 'Robert', 35000, 5),
(3, 'Kim', 'Reynolds', 40000, 5),
(4, 'Lindsey', 'Bowen', 80000, NULL),
(5, 'Paul', 'Taylor', 80000, 4),
(6, 'Tim', 'Johnson', 41000, 5),
(7, 'Lauren', 'Brook', 36000, 5),
(8, 'Smith', 'Wright', 34000, 4),
(9, 'Mohan', 'Kumar', 50000, 5);



-- 1er exemple

SELECT XMLAGG(XMLROW(
empid,
FIRSTNAME concat ' ' concat LASTNAME as name_emp
OPTION ROW "employe" as ATTRIBUTES ) ) AS XML_DATA
FROM MY_LIBRARY.HQEMPLOYEE e;

=> résultat obtenu :
<employe EMPID="1" NAME_EMP="John Brett"/><employe EMPID="2" NAME_EMP="Peter
Robert"/><employe EMPID="3" NAME_EMP="Kim Reynolds"/><employe EMPID="4"
NAME_EMP="Lindsey Bowen"/><employe EMPID="5" NAME_EMP="Paul Taylor"/><employe
EMPID="6" NAME_EMP="Tim Johnson"/><employe EMPID="7" NAME_EMP="Lauren
Brook"/><employe EMPID="8" NAME_EMP="Smith Wright"/><employe EMPID="9"
NAME_EMP="Mohan Kumar"/>
```

```
-- 2ème exemple

SELECT
XMLROW (
empid,
FIRSTNAME concat ' ' concat LASTNAME as name_emp
OPTION ROW "employe" as ATTRIBUTES   )
as XML_DATA
FROM MY_LIBRARY.HQEMPLOYEE e;

=> résultat obtenu :
<employe EMPID="1" NAME_EMP="John Brett"/>
<employe EMPID="2" NAME_EMP="Peter Robert"/>
<employe EMPID="3" NAME_EMP="Kim Reynolds"/>
<employe EMPID="4" NAME_EMP="Lindsey Bowen"/>
<employe EMPID="5" NAME_EMP="Paul Taylor"/>
<employe EMPID="6" NAME_EMP="Tim Johnson"/>
<employe EMPID="7" NAME_EMP="Lauren Brook"/>
<employe EMPID="8" NAME_EMP="Smith Wright"/>
<employe EMPID="9" NAME_EMP="Mohan Kumar"/>



-- 3ème exemple : renvoie le même résultat que la requête précédente

SELECT XMLSERIALIZE(
XMLROW (
empid,
FIRSTNAME concat ' ' concat LASTNAME as name_emp
OPTION ROW "employe" as ATTRIBUTES   )
AS varchar(32000)
VERSION '1.0'  -- paramètre optionnel
) as XML_DATA
FROM MY_LIBRARY.HQEMPLOYEE e;
```

2.17.2 XML vers SQL

On peut aussi utiliser des données au format XML stockées dans l'IFS et les mettre au format SQL :

```
-- exemples pris sur Volubis : http://www.volubis.fr/Pausecaf/PAUSECAF65.html

-- exemple 1 :

SELECT * FROM XMLTABLE (
'$r/producteurs/PRODUCTEUR' PASSING XMLPARSE(DOCUMENT
GET_XML_FILE('/formation/prod.xml') ) as "r"
COLUMNS
code CHAR(5) PATH "CODE_PRODUCTEUR",
nom CHAR(25) PATH "NOM_PRODUCTEUR"
) as txml

;

-- exemple 2 :

SELECT * FROM XMLTABLE ('$result/document/data/element'
PASSING XMLPARSE(DOCUMENT
SYSTOOLS.HTTPGETBLOB('http://data.nantes.fr/api/publication/LOC_AIRES_COV_NM/
LOC_AIRES_COV_NM_STBL/content/?format=xml', '')
) as "result"
COLUMNS
nom CHAR(50) PATH "geo/name",
cdpst CHAR(5) PATH "CODE_POSTAL",
places INT PATH "CAPACITE_VOITURE"
) as RESULT
;
```

2.17.3 XSLT

```
-- Transformation de flux XML avec XSLT
-- Exemple pris sur le site suivant (et légèrement adapté) :
-- http://pic.dhe.ibm.com/infocenter/iseries/v7r1m0/index.jsp?topic=%2Fsqlp
%2Frbafyxml3610.htm
-- voir aussi : http://www.volubis.fr/news/liens/courshtm/XML/SQLXML.htm#level22
-- Nécessite pour fonctionner d'avoir au préalable installé le XML Toolkit d'IBM


DECLARE GLOBAL TEMPORARY TABLE XML_TAB (
DOCID INTEGER,
XML_DOC XML  CCSID 1208,
XSL_DOC  CLOB(1M) CCSID 1208
) WITH REPLACE ;


INSERT INTO QTEMP.XML_TAB VALUES
     (1,
      '<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <student studentID="1" firstName="Steffen" lastName="Siegmund"
        age="23" university="Rostock"/>
       </students>',

     '<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
     <head/>
     <body>
     <h1><xsl:value-of select="$headline"/></h1>
     <table border="1">
     <th>
     <tr>
     <td width="80">StudentID</td>
     <td width="200">First Name</td>
     <td width="200">Last Name</td>
     <td width="50">Age</td>
     <xsl:choose>
          <xsl:when test="$showUniversity =''true''">
               <td width="200">University</td>
          </xsl:when>
     </xsl:choose>
     </tr>
     </th>
     <xsl:apply-templates/>
     </table>
     </body>
```

```
     </html>
     </xsl:template>
          <xsl:template match="student">
               <tr>
               <td><xsl:value-of select="@studentID"/></td>
               <td><xsl:value-of select="@firstName"/></td>
               <td><xsl:value-of select="@lastName"/></td>
               <td><xsl:value-of select="@age"/></td>
               <xsl:choose>
                    <xsl:when test="$showUniversity = ''true'' ">
                         <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
               </xsl:choose>
               </tr>
     </xsl:template>
</xsl:stylesheet>'
);

SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM QTEMP.XML_TAB;
```

2.18 Création de Table

2.18.1 Généralités

Quelques particularités de DB2 for i, en ce qui concerne la création de tables :

Pour forcer un nom court sur les colonnes :

```
CRE_DATE for column CREDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
```

Pour forcer un nom court sur les tables elles-mêmes, 2 méthodes :

- Soit on le fait au moment de la création en indiquant le nom court via l'instruction FOR SYSTEM NAME :
  - `CREATE TABLE TAB_AVEC_NOM_LONG FOR SYSTEM NAME TABNOMCOUR ;`
- Soit on le fait après la création, via l'ordre RENAME :
  - `RENAME TABLE TAB_AVEC_NOM_LONG TO SYSTEM NAME TABNOMCOUR ;`

Attention : des conflits surviennent si on essaie d'affecter un nom court à un objet DB2, alors que ce nom court est déjà affecté à un autre objet. Dans ce cas, on peut permuter les noms courts des objets en conflit, après les avoir identifiés (cf. le chapitre consacré aux "conflits sur noms courts").

Autre particularité de DB2 for i, la possibilité de forcer un format sur une table :

```
CREATE TABLE MABIB.MATABLE (
...
)  RCDFMT MONFORMAT ;
```

2.18.2 Exemples de tables

Exemple de création de table sur DB2 for i :

```
CREATE TABLE My_LIBRARY.ENCRMCLITB (
       ID INTEGER
              GENERATED ALWAYS AS IDENTITY
              (START WITH 1, INCREMENT BY 1),
       CODEAGC   CHAR(3) CCSID 1147 NOT NULL ,
       DATEPAY   DATE ,
       NOMVEND   CHAR(30) CCSID 1147 NOT NULL default '',
       TYPEPAY   CHAR(4)  CCSID 1147 NOT NULL ,
       REFRPAY   CHAR(20) CCSID 1147 NOT NULL default '',
       INFOPAY   CHAR(30) CCSID 1147 NOT NULL default '',
       BKGPAY    CHAR(7)  CCSID 1147 NOT NULL default '',
       ACCPAY    CHAR(5)  CCSID 1147 NOT NULL default '',
       MONTPAY   DEC (11, 2) NOT NULL default 0,
       VALIDPAY  CHAR(1) CCSID 1147 NOT NULL default '',
       KBQPAY    CHAR(6) CCSID 1147 NOT NULL default '',
       DATEEDTR  DATE ,
       DATEREMI  DATE ,
       TIMEREMI  TIME ,
       CRE_DATE for column CREDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
       CRE_TIME for column CRETIME TIME NOT NULL DEFAULT CURRENT_TIME ,
       CRE_USID for column CREUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
       UPD_DATE for column UPDDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
       UPD_TIME for column UPDTIME TIME NOT NULL DEFAULT CURRENT_TIME ,
       UPD_USID for column UPDUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
       STATUT CHAR (1 ) NOT NULL WITH DEFAULT ' ' ,
       DEL_DATE for column DELDATE DATE DEFAULT null,
       DEL_TIME for column DELTIME TIME DEFAULT null ,
       DEL_USID for column DELUSID CHAR(20) CCSID 1147 DEFAULT null ,
       CONSTRAINT My_LIBRARY.Q_kuoni_ENCRMCLITB_ID_00001 PRIMARY KEY( ID )
) VOLATILE ;


CREATE INDEX My_LIBRARY.ENCRMCLI01 ON My_LIBRARY.ENCRMCLITB(ID) ;
CREATE INDEX My_LIBRARY.ENCRMCLI02 ON My_LIBRARY.ENCRMCLITB(STATUT, ID) ;
CREATE INDEX My_LIBRARY.ENCRMCLI03 ON My_LIBRARY.ENCRMCLITB(STATUT, CODEAGC, ID) ;
CREATE INDEX My_LIBRARY.ENCRMCLI04 ON My_LIBRARY.ENCRMCLITB(STATUT, CODEAGC, DATEPAY,
ID) ;
CREATE INDEX My_LIBRARY.ENCRMCLI05 ON My_LIBRARY.ENCRMCLITB(STATUT, CODEAGC, BKGPAY,
ID) ;
CREATE INDEX My_LIBRARY.ENCRMCLI06 ON My_LIBRARY.ENCRMCLITB(DATEREMI, CODEAGC,
TYPEPAY, ID) ;


LABEL ON TABLE My_LIBRARY.ENCRMCLITB IS 'Retailing - lignes de transactions' ;
```

```
COMMENT ON TABLE My_LIBRARY.ENCRMCLITB IS 'Retailing - lignes de transactions' ;

LABEL ON COLUMN My_LIBRARY.ENCRMCLITB
(
       ID        IS 'Id de transaction' ,
       CODEAGC   IS 'Code agence' ,
       DATEPAY   IS 'Date réception paiement' ,
       NOMVEND   IS 'Nom du vendeur' ,
       TYPEPAY   IS 'Type paiement' ,
       REFRPAY   IS 'Réf. paiement' ,
       INFOPAY   IS 'Info paiement' ,
       BKGPAY    IS 'N° du booking' ,
       ACCPAY    IS 'Code clients PII' ,
       MONTPAY   IS 'Montant reçu' ,
       VALIDPAY  IS 'Statut ligne' ,
       KBQPAY    IS 'Compte banque Kuoni' ,
       DATEEDTR  IS 'Date édition reçu' ,
       DATEREMI  IS 'Date remise' ,
       TIMEREMI  IS 'Heure remise' ,
       CRE_DATE  IS 'Date création        ' ,
       CRE_TIME  IS 'Heure création       ' ,
       CRE_USID  IS 'ID user création     ' ,
       UPD_DATE  IS 'Date modif           ' ,
       UPD_TIME  IS 'Heure modif          ' ,
       UPD_USID  IS 'ID user modif        ' ,
    STATUT    IS 'Code statut      ' ,
       DEL_DATE  IS 'Date suppression        ' ,
       DEL_TIME  IS 'Heure suppression       ' ,
       DEL_USID  IS 'ID user suppression     '
       ) ;

LABEL ON COLUMN My_LIBRARY.ENCRMCLITB
(
       ID        TEXT IS 'Id de transaction' ,
       CODEAGC   TEXT IS 'Code agence' ,
       DATEPAY   TEXT IS 'Date réception paiement' ,
       NOMVEND   TEXT IS 'Nom du vendeur' ,
       TYPEPAY   TEXT IS 'Type paiement' ,
       REFRPAY   TEXT IS 'Réf. paiement' ,
       INFOPAY   TEXT IS 'Info paiement' ,
       BKGPAY    TEXT IS 'N° du booking' ,
       ACCPAY    TEXT IS 'Code clients PII' ,
       MONTPAY   TEXT IS 'Montant reçu' ,
       VALIDPAY  TEXT IS 'Statut ligne' ,
       KBQPAY    TEXT IS 'Compte banque Kuoni' ,
       DATEEDTR  TEXT IS 'Date édition reçu' ,
       DATEREMI  TEXT IS 'Date remise' ,
       TIMEREMI  TEXT IS 'Heure remise' ,
       CRE_DATE  TEXT IS 'Date création        ' ,
       CRE_TIME  TEXT IS 'Heure création       ' ,
       CRE_USID  TEXT IS 'ID user création     ' ,
       UPD_DATE  TEXT IS 'Date modif           ' ,
       UPD_TIME  TEXT IS 'Heure modif          ' ,
```

```
      UPD_USID  TEXT IS 'ID user modif        ' ,
    STATUT    TEXT IS 'Code statut       ' ,
      DEL_DATE  TEXT IS 'Date suppression        ' ,
      DEL_TIME  TEXT IS 'Heure suppression       ' ,
      DEL_USID  TEXT IS 'ID user suppression     '
      ) ;


CREATE TABLE My_LIBRARY.ENCBANQUTB (
      KBQPAY    CHAR(6) CCSID 1147 NOT NULL ,
      KBQNAME   CHAR(30) CCSID 1147 NOT NULL ,
      KBQROUT   CHAR(20) CCSID 1147 NOT NULL ,
      KBQNUM    CHAR(20) CCSID 1147 NOT NULL ,
      KBQSWIF   CHAR(11) CCSID 1147 NOT NULL ,
      KBQIBAN   CHAR(30) CCSID 1147 NOT NULL ,
      KBQADDR   CHAR(80) CCSID 1147 NOT NULL ,
      CRE_DATE for column CREDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
      CRE_TIME for column CRETIME TIME NOT NULL DEFAULT CURRENT_TIME ,
      CRE_USID for column CREUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
      UPD_DATE for column UPDDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
      UPD_TIME for column UPDTIME TIME NOT NULL DEFAULT CURRENT_TIME ,
      UPD_USID for column UPDUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
      CONSTRAINT My_LIBRARY.Q_kuoni_ENCBANQUTB_KBQPAY_00001 UNIQUE (KBQPAY)
) VOLATILE ;

CREATE INDEX My_LIBRARY.ENCBANQU01 ON My_LIBRARY.ENCBANQUTB(KBQPAY) ;
CREATE INDEX My_LIBRARY.ENCBANQU02 ON My_LIBRARY.ENCBANQUTB(KBQNAME, KBQPAY) ;

LABEL ON TABLE My_LIBRARY.ENCBANQUTB  IS 'Retailing - lignes des ENCBANQUTB' ;

COMMENT ON TABLE My_LIBRARY.ENCBANQUTB IS 'Retailing - lignes des ENCBANQUTB' ;

LABEL ON COLUMN My_LIBRARY.ENCBANQUTB
(
      KBQPAY    IS 'Bank account' ,
      KBQNAME   IS 'Bank name' ,
      KBQROUT   IS 'Routing number' ,
      KBQNUM    IS 'Bank account number' ,
      KBQSWIF   IS 'SWIFT Code' ,
      KBQIBAN   IS 'IBAN' ,
      KBQADDR   IS 'Adresse' ,
      CRE_DATE IS 'Date création        ' ,
      CRE_TIME IS 'Heure création       ' ,
      CRE_USID IS 'ID user création     ' ,
      UPD_DATE IS 'Date modif           ' ,
      UPD_TIME IS 'Heure modif          ' ,
      UPD_USID IS 'ID user modif        '
) ;

LABEL ON COLUMN My_LIBRARY.ENCBANQUTB
(
      KBQPAY   TEXT IS 'Bank account' ,
```

```
       KBQNAME  TEXT IS 'Bank name' ,
       KBQROUT  TEXT IS 'Routing number' ,
       KBQNUM   TEXT IS 'Bank account number' ,
       KBQSWIF  TEXT IS 'SWIFT Code' ,
       KBQIBAN  TEXT IS 'IBAN' ,
       KBQADDR  TEXT IS 'Adresse' ,
       CRE_DATE TEXT IS 'Date création' ,
       CRE_TIME TEXT IS 'Heure création' ,
       CRE_USID TEXT IS 'ID user création' ,
       UPD_DATE TEXT IS 'Date modif' ,
       UPD_TIME TEXT IS 'Heure modif' ,
       UPD_USID TEXT IS 'ID user modif'
) ;


CREATE TABLE My_LIBRARY.ENCPAYMTTB (
       TYPEPAY   CHAR(4) CCSID 1147 NOT NULL ,
       TYPEDESCR CHAR(30) CCSID 1147 NOT NULL ,
       TYPEAX    CHAR(1) CCSID 1147 NOT NULL ,
       CPTEBQE   CHAR(1) CCSID 1147 NOT NULL ,
       TYPEOFFC  CHAR(10) CCSID 1147 NOT NULL DEFAULT '',
       TYPEOFFT  CHAR(1) CCSID 1147 NOT NULL DEFAULT '',
       TYPECOMM  DEC(5, 2) NOT NULL DEFAULT 0,
       TYPECOMA  CHAR(10) NOT NULL DEFAULT '',
       CRE_DATE for column CREDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
       CRE_TIME for column CRETIME TIME NOT NULL DEFAULT CURRENT_TIME ,
       CRE_USID for column CREUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
       UPD_DATE for column UPDDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
       UPD_TIME for column UPDTIME TIME NOT NULL DEFAULT CURRENT_TIME ,
       UPD_USID for column UPDUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
       CONSTRAINT My_LIBRARY.Q_kuoni_ENCPAYMTTB_TYPEPAY_00001 UNIQUE (TYPEPAY)
) VOLATILE ;

CREATE INDEX My_LIBRARY.ENCPAYMT01 ON My_LIBRARY.ENCPAYMTTB(TYPEPAY) ;
CREATE INDEX My_LIBRARY.ENCPAYMT02 ON My_LIBRARY.ENCPAYMTTB(TYPEDESCR, TYPEPAY) ;

LABEL ON TABLE My_LIBRARY.ENCPAYMTTB  IS 'Retailing - Moyens de paiement' ;

COMMENT ON TABLE My_LIBRARY.ENCPAYMTTB IS 'Retailing - Moyens de paiement' ;

LABEL ON COLUMN My_LIBRARY.ENCPAYMTTB
(
       TYPEPAY   IS 'Type paiement' ,
       TYPEDESCR IS 'Description' ,
       TYPEAX    IS 'Ecriture AX' ,
       CPTEBQE   IS 'Compte banque' ,
       TYPEOFFC IS 'Offset account' ,
       TYPEOFFT IS 'Offset account type' ,
       TYPECOMM IS 'Commission %' ,
       TYPECOMA IS 'Compte commission' ,
       CRE_DATE IS 'Date création        ' ,
       CRE_TIME IS 'Heure création       ' ,
       CRE_USID IS 'ID user création     ' ,
```

```
      UPD_DATE IS 'Date modif           ' ,
      UPD_TIME IS 'Heure modif          ' ,
      UPD_USID IS 'ID user modif        '
) ;

LABEL ON COLUMN My_LIBRARY.ENCPAYMTTB
(
      TYPEPAY   TEXT IS 'Type paiement' ,
      TYPEDESCR TEXT IS 'Description' ,
      TYPEAX    TEXT IS 'Ecriture AX' ,
      CPTEBQE   TEXT IS 'Compte banque' ,
      TYPEOFFC TEXT IS 'Offset account' ,
      TYPEOFFT TEXT IS 'Offset account type' ,
      TYPECOMM TEXT IS 'Commission %' ,
      TYPECOMA TEXT IS 'Compte commission' ,
      CRE_DATE TEXT IS 'Date création' ,
      CRE_TIME TEXT IS 'Heure création' ,
      CRE_USID TEXT IS 'ID user création' ,
      UPD_DATE TEXT IS 'Date modif' ,
      UPD_TIME TEXT IS 'Heure modif' ,
      UPD_USID TEXT IS 'ID user modif'
) ;


CREATE TABLE My_LIBRARY.ENCBASEPTB (
      ID NUMERIC(9, 0) NOT NULL ,
      BKGLONG    integer NOT NULL default 0 ,
      CLILONG    integer NOT NULL default 0 ,
      KUOMAIL    CHAR(80) CCSID 1147 NOT NULL ,
      KUONAME    CHAR(30) CCSID 1147 NOT NULL ,
      KUOADD     CHAR(320) CCSID 1147 NOT NULL DEFAULT '',
      KUOPATH    CHAR(128) CCSID 1147 NOT NULL DEFAULT '',
      CRE_DATE for column CREDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
      CRE_TIME for column CRETIME TIME NOT NULL DEFAULT CURRENT_TIME ,
      CRE_USID for column CREUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
      UPD_DATE for column UPDDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
      UPD_TIME for column UPDTIME TIME NOT NULL DEFAULT CURRENT_TIME ,
      UPD_USID for column UPDUSID CHAR(20) CCSID 1147 NOT NULL DEFAULT USER ,
      CONSTRAINT My_LIBRARY.Q_kuoni_ENCBASEPTB_ID_00001 UNIQUE (ID)
) VOLATILE ;

CREATE INDEX My_LIBRARY.ENCBASEP01 ON My_LIBRARY.ENCBASEPTB(ID) ;

LABEL ON TABLE My_LIBRARY.ENCBASEPTB  IS 'Retailing - Paramétrage de base' ;

COMMENT ON TABLE My_LIBRARY.ENCBASEPTB IS 'Retailing - Paramétrage de base' ;

LABEL ON COLUMN My_LIBRARY.ENCBASEPTB
(
      ID       IS 'ID paramétrage' ,
      BKGLONG  IS 'Longueur N° Booking' ,
      CLILONG  IS 'Longueur Code Clients (Code ABTA PII)' ,
      KUOMAIL  IS 'Adresse Mail compta siège' ,
```

```
	KUONAME  IS 'Nom de la société' ,
	KUOADD   IS 'Adresse de la société' ,
	KUOPATH   IS 'Chemin d''accès fichier compta' ,
	CRE_DATE IS 'Date création        ' ,
	CRE_TIME IS 'Heure création       ' ,
	CRE_USID IS 'ID user création     ' ,
	UPD_DATE IS 'Date modif           ' ,
	UPD_TIME IS 'Heure modif          ' ,
	UPD_USID IS 'ID user modif        '
) ;

LABEL ON COLUMN My_LIBRARY.ENCBASEPTB
(
	ID       TEXT IS 'ID paramétrage' ,
	BKGLONG  TEXT IS 'Longueur N° Booking' ,
	CLILONG  TEXT IS 'Longueur Code Clients (Code ABTA PII)' ,
	KUOMAIL  TEXT IS 'Adresse Mail compta siège' ,
	KUONAME  TEXT IS 'Nom de la société' ,
	KUOADD   TEXT IS 'Adresse de la société' ,
	KUOPATH  TEXT IS 'Chemin d''accès fichier compta' ,
	CRE_DATE TEXT IS 'Date création' ,
	CRE_TIME TEXT IS 'Heure création' ,
	CRE_USID TEXT IS 'ID user création' ,
	UPD_DATE TEXT IS 'Date modif' ,
	UPD_TIME TEXT IS 'Heure modif' ,
	UPD_USID TEXT IS 'ID user modif'
) ;


ALTER TABLE My_LIBRARY.ENCRMCLITB
	ADD CONSTRAINT My_LIBRARY.Q_kuoni_ENCRMCLITB_CODEAGC_00002
	FOREIGN KEY( CODEAGC )
	REFERENCES My_LIBRARY.ENCAGENCTB ( CODEAGC )
	ON DELETE CASCADE
	ON UPDATE NO ACTION ;

ALTER TABLE My_LIBRARY.ENCRMCLITB
	ADD CONSTRAINT My_LIBRARY.Q_kuoni_ENCRMCLITB_TYPEPAY_00002
	FOREIGN KEY( TYPEPAY )
	REFERENCES My_LIBRARY.ENCPAYMTTB ( TYPEPAY )
	ON DELETE CASCADE
	ON UPDATE NO ACTION ;

ALTER TABLE My_LIBRARY.ENCRMCLITB
	ADD CONSTRAINT My_LIBRARY.Q_kuoni_ENCRMCLITB_KBQPAY_00002
	FOREIGN KEY( KBQPAY )
	REFERENCES My_LIBRARY.ENCBANQUTB ( KBQPAY )
	ON DELETE CASCADE
	ON UPDATE NO ACTION ;
```

2.18.3 MQT

Exemple de création de MQT (Materialized Query Table) :

```
CREATE TABLE MABIB.MAMQT (
   col1, col2, col3
) AS (
   SELECT col1, col2, col3 from MABIB.MATABLE
)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER
ENABLE QUERY OPTIMIZATION
;

LABEL ON TABLE MABIB.MAMQT
             IS 'ceci est un exemple de MQT' ;
```

Le rafraîchissement du contenu de la MQT se fait au moyen de la commande suivante :

```
REFRESH TABLE MABIB.MAMQT ;
```

2.18.4 Renommer une colonne

Comment renommer une colonne de table :
Il n'existe pas d'instruction SQL permettant de renommer explicitement une colonne de table, mais on peut contourner le problème en procédant d e la façon suivante :

```
ALTER TABLE my_table ADD COLUMN new_name ... BEFORE old_name ;
UPDATE my_table SET new_name = old_name;
ALTER TABLE my_table DROP COLUMN old_name;
```

Explication : on commence par ajouter une colonne en précisant qu'on souhaite l'insérer avant la colonne à renomer. On copie ensuite le contenu de la colonne à renommer dans la nouvelle colonne. Et enfin, on supprime l'ancienne colonne dont on n'a plus besoin.

Attention : certaines opérations de maintenance SQL, telles que les "DROP COLUMN", entraînent sur IBMi le déclenchement automatique de messages systèmes nécessitant une réponse de l'utilisateur. S'il est facile de répondre à ce type de message quand on travaille en mode 5250, cela devient problématique quand on travaille avec un client SQL comme System i Navigator.

Exemple de message renvoyé par SQL, dans System i Navigator, lors d'une tentative de DROP COLUMN :

Etat SQL : 57014
Code fournisseur : -952
Message : [SQL0952] Traitement de l'instruction SQL arrêté. **Code raison : 10**. Cause . . . . . : L'opération SQL a pris fin avant son aboutissement normal. Code raison : 10. Les codes raison et leur signification sont les suivants : 1 - Une demande API SQLCancel a été traitée (à partir d'ODBC, par exemple). 2 - Le traitement SQL a pris fin en envoyant une exception. 3 - Fin anormale. 4 - Fin du groupe d'activation. 5 - Récupération du groupe d'activation ou des ressources. 6 - Fin du traitement. 7 - Une fonction EXIT a été appelée. 8 - Exception non gérée. 9 - Une fonction Long Jump a été traitée. **10 - Une réponse d'annulation à un message d'interrogation a été reçue.** 11 - Programme d'exit d'ouverture de fichier base de données (QIBM_QDB_OPEN). 0 - Cause inconnue. Que faire . . . : Si le code raison est 1, une demande client a été faite pour annuler le traitement SQL. Pour tous les autres codes raison, reportez-vous aux messages précédents pour connaître les raisons de l'arrêt du traitement SQL.

Pour régler le problème, il faut ajouter - si ce n'est pas déjà fait - une réponse automatique sur le message CPA32B2 avec la commande système WRKRPYLE (qui permet de consulter la liste des réponses existantes).
L'ajout de la réponse peut se faire par F6, ou en saisissant la commande suivante :

```
ADDRPYLE SEQNBR(xxxx) MSGID(CPA32B2) RPY('I')
```

(remplacer "xxxx" par un numéro de message vacant)

A partir de là, il devient possible d'exécuter la commande SQL suivante dans le client SQL de son choix (ou dans une procédure stockée DB2), mais au préalable il faut avoir exécuté la commande suivante :

```
call qcmdexc ('CHGJOB INQMSGRPY(*SYSRPYL)') ;
```

La documentation pour la valeur *SYSRPYL indique ceci :

« Le système vérifie, dans la liste des réponses système, si un poste existe pour tout message d'interrogation émis par ce travail. Si c'est le cas, il utilise la réponse de ce poste. Sinon, une réponse est obligatoire. »

Le message renvoyé par l'ALTER TABLE est le CPA32B2. Comme il se trouve que ce message a déjà une réponse automatique définie sur l'IBM i, la commande CHGJOB permet d'en bénéficier au sein du travail relatif au code SQL en cours d'exécution.

2.18.5 Données volatiles

Données volatiles :

La notion de « donnée volatile » est apparue sur DB2/LUW en V8, et sur DB2 for i en V6.

```
CREATE TABLE MABASE.T1 (
...
) NOT VOLATILE ;

CREATE TABLE MABASE.T2 (
...
) VOLATILE ;
```

Lorsque vous spécifiez le mot clé VOLATILE sur une table, BIND devient favorable à l'utilisation des chemins d'accès indexé, même si la table était vide au moment où RUNSTATS a été exécuté. Il faut savoir que la collecte de statistiques sur une table vide remplit le catalogue avec des statistiques indiquant que la table ne contient aucune donnée. Et, bien sûr, lorsque les statistiques de DB2 indiquent qu'une table est petite ou vide, DB2 utilisera de préférence un « hash » plutôt qu'un chemin d'accès s'appuyant sur un index. Mais le contenu de certaines tables est fluctuant, pouvant passer de « vide » à plusieurs dizaines de milliers de lignes. Le mot clé « VOLATILE » obligera DB2 à rechercher systématiquement un chemin d'accès adapté au volume de la table considérée.

2.18.6 Conflits sur noms courts

L'utilisation des noms longs (supérieurs à 10 caractères) pour la création d'objets sur une base DB2 for i ne va pas sans poser des problèmes si l'on a besoin de maintenir une correspondance avec les noms courts pour diverses raisons.
Parmi les raisons justifiant de continuer à utiliser des noms courts, on peut citer :
- l'utilisation de la commande CLRPFM sur certains tables (généralement plus rapide qu'un DELETE SQL)
- l'utilisation des objets dans de vieux programmes RPG dans lesquels les noms longs ne peuvent être utilisés

Le déploiement sur différentes plateformes peut présenter des difficultés, car selon l'ordre dans lequel les objets sont créés à l'intérieur d'une bibliothèque, les noms courts attribués automatiquement par SQL lors de la création des objets sont susceptibles d'être affectés à des objets différents de ceux de l'environnement d'origine.

Par exemple :
- sur le serveur 1, dans la bibliothèque XAFVHSAT, le nom court TMP_R00004 est affecté à la table TMP_RAP_CONT
- sur le serveur 2, dans la bibliothèque XAFVHSAT, le nom court TMP_R00004 est affecté à la vue VIW_RAP_CONT

On voit donc que dans ce cas, un CLRPFM sur l'objet TMP_R00004 n'aurait pas le même effet sur les 2 serveurs, et surtout il n'impacterait pas du tout le même objet (on a pris ici pour exemple une vue, mais ce pourrait être une autre table ou une MQT).

On peut être tenté d'utiliser l'ordre SQL RENAME mais dans certains cas cela ne fonctionnera pas car le nom court que l'on souhaite récupérer est affecté à un autre objet. Il faut dans ce cas "détacher" ce nom court de l'objet en le remplaçant par un autre nom court, avant de pouvoir réaffecter le nom court qui nous intéresse à un autre objet. On peut dans ce cas envisager de faire une permutation des noms courts, mais comme c'est un processus laborieux, autant le faire faire par une petite procédure stockée, que nous appellerons CHGNOMCOUR.

Une fois qu'elle sera opérationnelle, on pourra corriger notre problème de nom court sur l'exemple précédent au moyen d'une seule ligne de code SQL, telle que :

```
call MABIBPROC.CHGNOMCOUR ('XAFVHSAT', 'TMP_RAP_CONT', 'TMP_R00004') ;
```

Le code source complet de la procédure stockée CHGNOMCOUR est présenté dans le chapitre "Etudes de cas".

2.19 Requêtes SQL paramétrées

A chaque fois que l'on a besoin d'effectuer des opérations de mise à jour, d'insertion ou de suppression multiples, à partir d'un jeu de données parcouru à l'aide d'une boucle, comme dans un curseur SQL, il est vivement recommandé de préparer la requête de modification des données en amont de la boucle traitant le curseur. On obtient dès lors un "statement" que l'on va pouvoir exploiter à l'intérieur de la boucle. Ce "statement" est en fait un pointeur vers un emplacement mémoire de DB2 dans lequel il a stocké le chemin d'accès précalculé nécessaire pour effectuer l'opération de mise à jour (ou d'insertion, ou de suppression) de manière optimale.

Cette technique offre plusieurs avantages :
- calcul du chemin d'accès une seule fois en amont de la boucle, ce qui donne de très bonnes performances
- simplicité d'écriture : on n'a pas besoin de se préoccuper du format (numérique ou alphabétique) des données mises à jour, c'est DB2 qui s'en charge
- excellente protection contre les attaques dites "par injection SQL", car DB2 applique de puissants algorithmes de filtrage sur les données envoyées dans chacune des colonnes concernée par l'opération de mise à jour.

Dans l'exemple ci-dessous, extrait de la procédure que l'on retrouvera dans la partie consacrée aux procédures stockées (cf. chapitre "Techniques avancées"), la requête d'insertion est définie en amont d'un curseur SQL, via la syntaxe suivante :

```
SET V_DYNSQL1 = 'INSERT INTO ' concat trim(V_NOM_TABL) concat ' (ODCONO, ODDIVI,
ODITNO, ODSTYN, ODHDPR, ODORTA, ODCUCD, ODPRRF, ODSAPR, ODFVDT, ODLVDT, ODFVDT_DAT,
ODLVDT_DAT, ODITNO_8) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';

PREPARE V_DYNSTM1 FROM V_DYNSQL1;
```

L'exécution de la requête d'insertion est déclenchée dans le curseur, via la syntaxe suivante :

```
EXECUTE V_DYNSTM1 USING VALCONO, FXDIVI, ZXITNO, ZXSTYN, ZXHDPR, V_ORI_TAR, FXCUCD,
FXPRRF, FXSAPR, FXFVDT, FXLVDT, FXFVDT_DAT, FXLVDT_DAT, FXITNO_8;
```

A noter que ce principe est valable aussi bien dans les procédures stockées, qu'en SQLRPGLE et en PHP.

2.20 Verrouillage optimiste

Description de la technique dite du "verrouillage optimiste"

- Dans le cadre d'un développement en RPG, il est facile de mettre en œuvre la technique du verrouillage physique des enregistrements bases de données, en utilisant les ordres de lecture de base de données natifs.
- Dans le cadre d'un développement utilisant les techniques du web, il est impossible de mettre en œuvre un verrouillage physique des enregistrements bases de données, du fait notamment que les transactions HTTP fonctionnent en mode « stateless » (sans état).
- Il existe une technique relativement simple et élégante pour pallier l'absence de verrouillage physique, et qui n'est pas propre à DB2, c'est la technique dite du « verrouillage optimiste ».
- Pour que la technique du verrouillage optimiste puisse être mise oeuvre, il faut que les tables de la base de données respectent une certaine normalisation. Par exemple, il est indispensable que chaque table possède dans ses colonnes, soit un numéro de version, soit des zones mouchards dédiées au stockage des informations suivantes : « qui a mis à jour cette donnée, et quand ? »
- Le principe est finalement très simple et se décompose en plusieurs étapes (prenons pour exemple l'écran de mise à jour d'un produit) :
    - L'utilisateur affiche l'écran de mise à jour d'un produit. La requête SQL d'extraction de la fiche produit va récupérer les informations définissant le produit, ainsi que le contenu des zones mouchards de dernière mise à jour de cette fiche produit.
    - L'utilisateur modifie des informations de la fiche produit dans un formulaire HTML, puis il valide ce formulaire ce qui a pour effet de déclencher la série d'opérations suivantes :
        - Le script PHP côté serveur contrôle la validité des informations saisies. Si des anomalies sont détectées, le formulaire est réaffiché avec des messages d'erreur. Si aucune anomalie n'est détectée, le script passe à l'étape suivante
        - Le script PHP déclenche l'exécution d'une requête SQL de type UPDATE qui aura pour éléments de clé (dans la clause WHERE) l'identifiant de l'enregistrement modifié, ET les colonnes « mouchards » de la dernière modification connue. Si la requête échoue, cela signifie que la ligne dans la table SQL n'existe plus, OU, que cette ligne a subi une modification par un autre utilisateur (ou un autre travail) entre le moment où la ligne a été extraite de la base et le moment où on a tenté de la mettre à jour. Si cela se produit, on informe l'utilisateur qu'il a été pris de vitesse par quelqu'un d'autre, et on peut lui proposer plusieurs possibilités : soit abandonner la transaction, soit la réactualiser avec les dernières informations en base avant de procéder à une nouvelle tentative de mise à jour. Si aucune anomalie ne s'est produite, alors la ligne a bien été modifiée en base, et l'utilisateur peut passer à autre chose.

2.21 Verrouillage physique et niveaux d'isolement

Présentation des niveaux d'isolement (en anglais : Isolation Levels)

NB : on retrouve certains des éléments décrits ci-dessous dans la documentation de la commande IBM i RUNSQLSTM.

Des niveaux d'isolement sont employés par DB2 pour définir le niveau de protection que l'on souhaite assurer aux données lues. L'augmentation du niveau d'isolement au niveau d'un traitement réduit la capacité d'accéder aux données, pour les traitements et applications concurrents. En contrepartie, cela permet de fiabiliser la qualité des données retournées par les requêtes. DB2 offre 4 niveaux d'isolement standards, et 1 niveau d'isolement spécifique à la plateforme IBMi. Ces niveaux d'isolement sont les suivants :

- UR (Uncommitted Read)
- NC (No Commit), spécifique à la plateforme IBMi
- CS (Cursor Stability), valeur par défaut
- RS (Read Stability)
- RR (Repeatable Read)

Deux manières de déclarer le niveau d'isolement :

1 - La déclaration du niveau d'isolement peut se faire sur chaque requête au moyen de l'instruction WITH xx (où xx est le niveau d'isolement souhaité). Par exemple, la requête ci-dessous fonctionne selon le mode d'isolement NC :

```
SELECT * FROM table WITH NC ;
```

2 – La déclaration du niveau d'isolement peut être faite en début de programme, au moyen de la clause SET OPTION. Dans ce cas, le niveau d'isolement s'appliquera de manière homogène à toutes les requêtes exécutées à l'intérieur du programme (cela évite de coder « en dur » le niveau d'isolement sur chaque requête). Exemple ci-dessous correspondant au niveau d'isolement NC :

```
SET OPTION COMMIT = *NONE ;
```

On recommandera d'utiliser de préférence la clause SET OPTION, plutôt que la clause WITH xx, car la clause SET OPTION permet de définir un niveau d'isolement homogène pour l'ensemble des requêtes exécutées par un programme.

Les options possibles pour le paramètre COMMIT sont les suivantes :

*NONE	correspond au niveau d'isolement NC (No Commit).
*CHG	correspond au niveau d'isolement UR (Uncommitted Read).
*CS	correspond au niveau d'isolement CS (Cursor Stability).
*ALL	correspond au niveau d'isolement RS (Read Stability).
*RR	correspond au niveau d'isolement RR (Repeatable Read).

Description détaillée des niveaux d'isolement :

Uncommitted Read (UR)
Le niveau d'isolement UR indique que les objets désignés dans les instructions SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME et REVOKE, ainsi que toutes les lignes mises à jour, supprimées ou insérées sont verrouillés jusqu'à la fin de l'unité d'oeuvre (transaction). Les modifications invalidées dans les autres travaux peuvent être visualisées.

No Commit (NC) – mode spécifique à la plateforme IBMi
Le niveau d'isolement NC fonctionne de la même façon que le niveau d'isolement d'UR, excepté le fait que les COMMIT et ROLLBACK n'ont aucun effet sur les instructions SQL exécutées dans ce mode. Cela signifie que, quand un COMMIT ou un ROLLBACK est exécuté, les curseurs ne sont pas fermés et les verrous tenus ne sont pas libérés (même ceux acquis avec l'instruction LOCK TABLE). De plus, toutes les mises à jour (ou suppressions) sont appliquées immédiatement, et sont visibles immédiatement pour les autres applications. Avec ce niveau d'isolement, aucun verrou n'est acquis pour des opérations de lecture. Pour les opérations de MISE À JOUR, la durée du verrou est réduite au minimum (par exemple, une ligne est verrouillée seulement au moment de sa mise à jour).
On recommandera l'usage systématique du niveau d'isolement NC sur les requêtes « attaquant » des tables IBMi non journalisées, pour lesquelles l'utilisation de COMMIT et ROLLBACK n'est pas possible. Ce mode est celui qui se rapproche le plus de la gestion des verrouillages la plus couramment pratiquée dans les applications natives IBMi (Adelia et RPG).

Cursor Stability (CS)
Le niveau d'isolement CS est le niveau d'isolement appliqué par défaut par DB2. Il protège la ligne en cours de lecture ou de mise à jour, de toute tentative de modification qui pourrait être effectuée par d'autres travaux ou d'autres applications. De même, il est impossible d'accéder aux données lues et modifiées par d'autres processus, jusqu'à ce que ces données aient fait l'objet d'un COMMIT. Tant que le COMMIT n'est pas exécuté, un verrou de partage (share) est appliqué sur la ligne en cours de lecture, et un verrou d'exclusion (X) est appliqué à toutes les lignes mises à jour et supprimées (jusqu'au prochain COMMIT). Le niveau d'isolement CS est le plus couramment utilisé car il interdit l'accès à des données sales, et il réduit le verrouillage des données au strict minimum (le verrouillage s'applique uniquement à la ligne en cours de FETCH, et aux lignes mises à jour et supprimées non COMMITées).

Read Stability (RS)
Le niveau d'isolement RS fonctionne comme le niveau d'isolement CS, mais au lieu de verrouiller uniquement la ligne en cours de traitement, le niveau d'isolement RS a pour effet d'appliquer le verrou approprié à toutes les lignes concernées par le curseur. Ceci permet de s'assurer que dans la même transaction (en anglais : « Unit of Work »), des lignes qui ont été précédemment lues par un curseur ne peuvent pas être modifiées par d'autres applications.

Le niveau d'isolement RS pourrait s'appliquer par exemple pour le traitement d'un ensemble de données interdépendantes, comme par exemple les lignes « détail » d'une commande.

Repeatable Read (RR)

Le niveau d'isolement RR est le niveau d'isolement le plus fort. Dans ce mode, DB2 verrouille toutes les lignes correspondant au jeu de données retourné par le curseur, ainsi que toutes les lignes qui sont utilisées pour établir ce jeu de données. Si une ligne est lue par l'application en utilisant ce niveau d'isolement, aucune autre application ne peut la modifier, jusqu'à ce que la transaction soit terminée. Ceci permet de s'assurer que le jeu de données résultat est conforme en tous points. En règle générale, l'exécution répétée d'une même requête dans ce mode retournera le même jeu de données en sortie, d'où l'idée de « lecture répétable » (en anglais : « repeatable read » ou « RR »). Ce mode peut considérablement pénaliser les applications concurrentes, du fait du grand nombre de verrous établis à l'intérieur d'une même transaction.

Le tableau ci-dessous donne un aperçu des limitations propres à chaque mode :

Niveau d'isolement	Accès à des données non commitées	Lectures non répétables	Lecture de données fantômes
RR	Impossible	Impossible	Impossible
RS	Impossible	Impossible	Possible
CS	Impossible	Possible	Possible
UR et NC	Possible	Possible	Possible

Verrouillage pessimiste et gestion des verrouillages

Apparu en V6R1, la clause "SKIP LOCKED DATA" est une nouvelle option utilisable sur les requêtes de type SELECT, DELETE et UPDATE.
Elle permet de "sauter" les lignes de bases de données qui sont verrouillées par d'autres travaux. Elle permet donc de répondre aux besoins des applications pour lesquelles un verrouillage pessimiste est requis.
La clause "SKIP LOCKED DATA" est ignorée si elle est spécifiée lorsque le niveau d'isolement en vigueur est de type "lecture reproductible" (RR) ou de type "lecture non validée" (UR).

```
DELETE FROM matable
WHERE date_effet <= '2007-01-01'
  WITH CS
  SKIP LOCKED DATA ;
```

2.22 Variables globales

Apparues avec la V7, les variables globales permettent de stocker dans l'emplacement de son choix des données qui peuvent être exploitées par les requêtes d'une base de données particulière.

Cette approche peut être particulièrement intéressante pour pouvoir distinguer plusieurs environnements d'exécution (test, recette, préproduction, production) se trouvant sur un même serveur et une même partition.

Exemple de variables globales définissant un environnement d'exécution :

```
CREATE VARIABLE MABIB.APP_TYP_ENV CHAR(3)  DEFAULT 'R';
LABEL ON VARIABLE MABIB.APP_TYP_ENV IS 'environnement de recette';

CREATE VARIABLE MABIB.APP_COD_SOC CHAR(3)  DEFAULT '010';
LABEL ON VARIABLE MABIB.APP_COD_SOC IS 'code société 010';
```

La lecture des variables globales se fait très simplement :

```
SELECT MABIB.APP_TYP_ENV FROM SYSIBM.SYSDUMMY1; -- Renvoie "R"
SELECT MABIB.APP_COD_SOC FROM SYSIBM.SYSDUMMY1; -- Renvoie "010"
```

Si on travaille avec des listes de bibliothèques (donc en syntaxe « système » au lieu de « SQL »), on doit faire abstraction de la bibliothèque de stockage des variables globales :

```
SELECT APP_TYP_ENV FROM SYSIBM.SYSDUMMY1; -- renvoie "R"
SELECT APP_COD_SOC FROM SYSIBM.SYSDUMMY1; -- renvoie "010"
```

Autre manière d'arriver au même résultat :

```
SELECT * FROM (VALUES(MABIB.APP_TYP_ENV)) VARIABLES(TYP_ENV) ;
```

La table QSYS2.SYSVARIABLES contient la liste des variables globales qui ont été créées sur le système. Il est dès lors facile de retrouver toutes les bases de données dans lesquelles une variable est déclarée, au moyen d'une requête du type :

```
SELECT * FROM QSYS2.SYSVARS WHERE VARIABLE_NAME = 'APP_COD_SOC' ;
```

A noter : la valeur associée à chaque variable est stockée dans la table SYSVARIABLES sous forme d'un pointeur. On ne peut donc pas visualiser cette information directement à partir de cette table.

La mise à jour d'une variable globale se fait de la façon suivante :

```
SET MABIB.APP_TYP_ENV = 'P' ;
```

On peut aussi alimenter le contenu d'une variable globale via une sous-requête scalaire.

```
SET MABIB.APP_TYP_ENV = (SELECT TYP_ENV FROM TABENV FETCH FIRST 1 ROW ONLY) ;
```

2.23 Registres DB2

DB2 fournit un certain nombre de registres intéressants à connaître.

Les registres pouvant être exploités avec DB2 pour IBM i sont les suivants :

- CURRENT SERVER : pour récupérer l'identifiant du serveur IBM i
- CURRENT SCHEMA : pour récupérer la bibliothèque courante (*LIBL par défaut)
- CURRENT PATH : pour récupérer le chemin d'accès courant (*LIBL par défaut)
- CURRENT DATE
- CURRENT TIME
- CURRENT TIMESTAMP
- USER (sans CURRENT) : profil utilisateur stocké dans une chaîne de 128 caractères

Il existe aussi quelques registres DB2 qui ne sont pas supportés par DB2 pour IBM i, tels que :

- CURRENT QUERY OPTIMIZATION
- CURRENT EXPLAIN MODE
- CURRENT EXPLAIN SNAPSHOT

Il est possible de récupérer un registre DB2 au moyen de l'une des requêtes suivantes :

```
SELECT CURRENT SCHEMA FROM QSQPTABL;
VALUES CURRENT SCHEMA INTO :VAR;
```

Sur la plateforme IBM i, ces requêtes retournent toutes deux la valeur suivante : *LIBL

Il est possible de modifier les valeurs de certains registres. Ces techniques sont à manipuler avec beaucoup de précaution. Par exemple :

- la bibliothèque courante peut être forcée au moyen de la requête suivante :

```
SET SCHEMA = 'mabib'
```

- le PATH courant peut être forcé avec une liste de bibliothèques spécifiques :

```
SET PATH = "bib1", "bib2", "bib3"
```

- le PATH courant peut être forcé avec le PATH courant auquel on ajoute une liste de bibliothèques (il semble que cette dernière technique ne fonctionne pas sur IBM i) :

```
SET PATH = CURRENT PATH, "bib1", "bib2"
```

Pour de plus amples précisions sur les registres :

De nouveaux registres sont apparus avec la V6 de l'OS/400 :

SQL Special Register Name	Register Name	Datatype	Database Monitor Column (where QQRID=1000)
CURRENT CLIENT_APPLNAME	Client Application Name	VARCHAR(255)	QVC3001
CURRENT CLIENT_ACCTNG	Client Accounting	VARCHAR(255)	QVC3005
CURRENT CLIENT_PROGRAMID	Client Program ID	VARCHAR(255)	QVC3006
CURRENT CLIENT_USERID	Client User ID	VARCHAR(255)	QVC3002
CURRENT CLIENT_WRKSTNNAME	Client Workstation	VARCHAR(255)	QVC3003

Pour voir le contenu de ces registres :

```
SELECT CURRENT CLIENT_APPLNAME,
CURRENT CLIENT_ACCTNG,
CURRENT CLIENT_PROGRAMID,
CURRENT CLIENT_USERID,
CURRENT CLIENT_WRKSTNNAME
FROM SYSIBM.SYSDUMMY1
```

Il devient donc possible d'écrire ceci à l'intérieur d'un programme :

```
exec sql
SELECT CURRENT CLIENT_PROGRAMID INTO :pgmID FROM sysibm.sysdummy1;
if (pgmID <> 'DB2WBQRY');
return;
endif;
```

Ces nouveaux registres sont récupérables dans les tables de logs de performances générées par l'une ou l'autre des requêtes ci-dessous :

En V6R1:

```
STRDBMON OUTFILE(QGPL/DB2WQFILE) OUTMBR(*FIRST *REPLACE)
JOB(*ALL/*ALL/*ALL) TYPE(*DETAIL)
COMMENT('FTRCLTPGM(DB2WBQRY)')
```

En V7R1:

```
STRDBMON OUTFILE(QGPL/DB2WQFILE) OUTMBR(*FIRST *REPLACE)
JOB(*ALL/*ALL/*ALL) TYPE(*DETAIL)FTRCLTPGM(DB2WBQRY)
```

Requête d'analyse :

```
SELECT
  qqtime AS time ,
  qvc3002 AS client_userid ,
  qvc3003 AS client_wkstnname ,
  qvc3001 AS client_applname ,
  qvc3005 AS client_acctng,
```

```
  qvc3006 AS client_programid,
  qq1000 AS statement
FROM db2wqfile
WHERE qqrid = 1000 AND qvc3006 = 'DB2WBQRY'
ORDER BY qqtime;
```

Question récurrente dans les forums : comment récupérer le numéro de version de l'instance DB2 courante ?

- sur DB2 pour Windows : plusieurs possibilités

-> avec db2level :

-> avec db2licm -l :

-> avec les fonctions env_get_inst_info(), env_get_prod_info() and env_get_sys_info() :

```
SELECT service_level, fixpack_num, bld_level
FROM TABLE (sysproc.env_get_inst_info()) as A ;

service_level          fixpack_num bld_level
---------------------- ----------- ----------
DB2 v9.1.100.129       1           s061104
```

- sur DB2 pour IBM i, à partir de la V7, on dispose d'une nouvelle vue DB2, ENV_SYS_INFO, qui se trouve dans la bibliothèque SYSIBMADM.

```
SELECT OS_NAME, OS_VERSION, OS_RELEASE, HOST_NAME, TOTAL_CPUS, CONFIGURED_CPUS,
        TOTAL_MEMORY
FROM SYSIBMADM.ENV_SYS_INFO
```

Exemple de valeurs renvoyées :

OS_NAME	IBM i
OS_VERSION	7
OS_RELEASE	1

On peut obtenir les mêmes infos sans passer par la vue DB2 :

```
SELECT
CHAR(OS_NAME, 20) AS OS_NAME,
CHAR(OS_VERSION, 20) AS OS_VERSION,
CHAR(OS_RELEASE, 20) AS OS_RELEASE,
HOST_NAME,
TOTAL_CPUS,
CONFIGURED_CPUS,
TOTAL_MEMORY
FROM (
SELECT A.* FROM TABLE(SYSIBMADM.ENV_SYS_INFO ( )) AS A
) X ;
```

2.24 Tables systèmes

Les tables systèmes de la base de données DB2 for i recèlent des trésors insoupçonnés.
On peut assez facilement, à partir de leur contenu, se constituer des outils de supervision ou d'administration de bases de données.

Le tableau ci-dessous, sans prétendre être exhaustif, présente un panel de tables systèmes qu'il est intéressant de connaître :

Nom de la vue	***Contenu de la vue***
SYSCHKCST	Contraintes de vérification
SYSCOLUMNS	Colonnes des tables/vues SQL, Pfs/LFs
SYSCST	Contraintes
SYSCSTCOL	Colonnes impliquées dans les contraintes
SYSCSTDEP	Dépendances dûes aux contraintes
SYSINDEXES	Index SQL
SYSKEYCST	Clés utilisées par les contraintes
SYSKEYS	Clés des index SQL
SYSPACKAGE	Modules SQL
SYSREFCST	Contraintes d'intégrité référentielle
SYSTABLES	Tables et vues SQL, Pfs et LFs
SYSTRIGCOL	Colonnes utilisées dans déclencheur SQL
SYSTRIGDEP	Objets utilisés par un déclencheur SQL
SYSTRIGGER	Déclencheurs
SYSTRIGUPD	Colonnes testées dans la clause WHEN
SYSVIEWDEP	Dépendances dûes aux vues SQL
SYSVIEWS	Vues SQL

Exemple de requêtes permettant d'extraire des informations d'une base de données, avec leurs équivalents MySQL et DB2 :

MySQL	DB2
Show databases	SELECT TABLE_SCHEM FROM SYSIBM.SQLSCHEMAS
Show tables	SELECT TABLE_NAME FROM SYSTABLES WHERE TABLE_SCHEMA = schema_name
Show index from table_name	SELECT INDEX_NAME FROM SYSINDEXES

	WHERE TABLE_NAME = table_name AND INDEX_SCHEMA = schema_name

Les fichiers systèmes situés dans QSYS offrent aussi un certain nombre d'informations précieuses :

QADBIFLD	Toutes les zones de tous les fichiers de votre système = référentiel "physique" complet
QADBKFLD	Toutes les clés de tous les fichiers de votre système = le référentiel "logique" le plus complet (Nom des zones, position dans la clé, ...)
QADBFDEP	Les dépendances physiques / logiques de votre Base de Données
QADBXREF	Tous les fichiers de votre AS400 (nombre de champs, de clés, longueur d'enregistrement, etc...)
QADBCCST	Toutes les contraintes de votre AS400
QADBFCST	idem

Par exemple, pour trouver les fichiers logiques se trouvant dans des bibliothèques différentes de leurs physiques de rattachement :

```
SELECT substr( DBFFIL, 1, 10) as fic1, DBFLIB,
       substr( DBFFDP, 1, 10) as fic2, DBFLDP
 FROM QSYS/QADBFDEP
 WHERE DBFLIB = 'AZTOUDTA' and DBFLIB <> DBFLDP ;
```

Autre exemple : extraction de la liste des tables utilisant un type de donnée particulier (par exemple le type VARCHAR) :

```
select distinct table_name
from qsys2.syscolumns
where table_schema = 'yyyyy'
  and data_type = 'VARCHAR';
```

Dernier exemple de ce chapitre : extraction de la liste des tables d'une bibliothèque :

```
select
table_name,
system_table_name,
table_type,
column_count,
row_length,
table_text,
is_insertable_into,
table_owner
from qsys2.systables
```

```
where table_schema = 'DWHPRD' ;
```

Dans les chapitres qui suivent, nous verrons d'autres exemples de requêtes permettant d'extraire des informations intéressantes.

2.24.1 SYSTABLES, SYSCOLUMNS, SYSINDEXS

Les tables SYSTABLES et SYSCOLUMNS permettent de récupérer facilement la structure des tables SQL.
On peut exploiter ces informations pour générer le code SQL de recréation des tables au format DB2, ou dans le format spécifiique à une autre base de données (MySQL par exemple). On peut aussi s'appuyer sur ces informations pour générer des fichiers XML ou des classes (PHP ou Java) répondant aux spécifications attendues par une équipe de développement.

Structure de la table SYSTABLES :

N°	Nom de colonne (long)	Nom court	Type	Longueur	Déc.
1	TABLE_NAME	NAME	VARCHAR	128	
2	TABLE_OWNER	CREATOR	VARCHAR	128	
3	TABLE_TYPE	TYPE	CHAR	1	
4	COLUMN_COUNT	COLCOUNT	INTEGER	9	0
5	ROW_LENGTH	RECLENGTH	INTEGER	9	0
6	TABLE_TEXT	LABEL	VARGRAPHIC	50	
7	LONG_COMMENT	REMARKS	VARGRAPHIC	2000	
8	TABLE_SCHEMA	DBNAME	VARCHAR	128	
9	LAST_ALTERED_TIMESTAMP	ALTEREDTS	TIMESTAMP	10	
10	SYSTEM_TABLE_NAME	SYS_TNAME	CHAR	10	
11	SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR	10	
12	FILE_TYPE	FILETYPE	CHAR	1	
13	BASE_TABLE_CATALOG	LOCATION	VARCHAR	18	
14	BASE_TABLE_SCHEMA	TBDBNAME	VARCHAR	128	
15	BASE_TABLE_NAME	TBNAME	VARCHAR	128	
16	BASE_TABLE_MEMBER	TBMEMBER	VARCHAR	10	
17	SYSTEM_TABLE	SYSTABLE	CHAR	1	
18	SELECT_OMIT	SELECTOMIT	CHAR	1	
19	IS_INSERTABLE_INTO	INSERTABLE	VARCHAR	3	
20	IASP_NUMBER	IASPNUMBER	SMALLINT	4	0
21	ENABLED	ENABLED	VARCHAR	3	
22	MAINTENANCE	MAINTAIN	VARCHAR	6	
23	REFRESH	REFRESH	VARCHAR	9	

24	REFRESH_TIME	REFRESHDTS	TIMESTAMP	10	
25	MQT_DEFINITION	MQTDEF	DBCLOB	2097152	
26	ISOLATION	ISOLATION	CHAR	2	
27	PARTITION_TABLE	PART_TABLE	VARCHAR	11	
28	TABLE_DEFINER	DEFINER	VARCHAR	128	
29	MQT_RESTORE_DEFERRED	MQTRSTDFR	CHAR	1	
30	ROUNDING_MODE	DECFLTRND	CHAR	1	

Structure de la table SYSCOLUMNS :

N°	Nom de colonne (long)	Nom court	Type	Longueur	Déc.
1	COLUMN_NAME	NAME	VARCHAR	128	
2	TABLE_NAME	TBNAME	VARCHAR	128	
3	TABLE_OWNER	TBCREATOR	VARCHAR	128	
4	ORDINAL_POSITION	COLNO	INTEGER	9	0
5	DATA_TYPE	COLTYPE	VARCHAR	8	
6	LENGTH	LENGTH	INTEGER	9	0
7	NUMERIC_SCALE	SCALE	INTEGER	9	0
8	IS_NULLABLE	NULLS	CHAR	1	
9	IS_UPDATABLE	UPDATES	CHAR	1	
10	LONG_COMMENT	REMARKS	VARGRAPHIC	2000	
11	HAS_DEFAULT	DEFAULT	CHAR	1	
12	COLUMN_HEADING	LABEL	VARGRAPHIC	60	
13	STORAGE	STORAGE	INTEGER	9	0
14	NUMERIC_PRECISION	PRECISION	INTEGER	9	0
15	CCSID	CCSID	INTEGER	9	0
16	TABLE_SCHEMA	DBNAME	VARCHAR	128	
17	COLUMN_DEFAULT	DFTVALUE	VARGRAPHIC	2000	
18	CHARACTER_MAXIMUM_LENGTH	CHARLEN	INTEGER	9	0
19	CHARACTER_OCTET_LENGTH	CHARBYTE	INTEGER	9	0
20	NUMERIC_PRECISION_RADIX	RADIX	INTEGER	9	0
21	DATETIME_PRECISION	DATPRC	INTEGER	9	0
22	COLUMN_TEXT	LABELTEXT	VARGRAPHIC	50	
23	SYSTEM_COLUMN_NAME	SYS_CNAME	CHAR	10	
24	SYSTEM_TABLE_NAME	SYS_TNAME	CHAR	10	
25	SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR	10	
26	USER_DEFINED_TYPE_SCHEMA	TYPESCHEMA	VARCHAR	128	
27	USER_DEFINED_TYPE_NAME	TYPENAME	VARCHAR	128	
28	IS_IDENTITY	IDENTITY	VARCHAR	3	
29	IDENTITY_GENERATION	GENERATED	VARCHAR	10	
30	IDENTITY_START	START	DECIMAL	31	0

31	IDENTITY_INCREMENT	INCREMENT	DECIMAL	31	0
32	IDENTITY_MINIMUM	MINVALUE	DECIMAL	31	0
33	IDENTITY_MAXIMUM	MAXVALUE	DECIMAL	31	0
34	IDENTITY_CYCLE	CYCLE	VARCHAR	3	
35	IDENTITY_CACHE	CACHE	INTEGER	9	0
36	IDENTITY_ORDER	ORDER	VARCHAR	3	
37	COLUMN_EXPRESSION	EXPRESSION	DBCLOB	2097152	
38	HIDDEN	HIDDEN	VARCHAR	1	
39	HAS_FLDPROC	FLDPROC	VARCHAR	1	

A noter que DB2 for i fournit une API très pratique pour récupérer le contenu de la table SYSCOLUMNS, l'API QSQSYSCOL2

```
SELECT *
FROM QSYS2.systables A,
     TABLE (QSYS2.QSQSYSCOL2(A.system_table_schema, A.system_table_name) ) AS c
WHERE a.TABLE_SCHEMA = 'MABIB' AND a.TABLE_NAME = 'MATABLE' ;
```

Exemple de requête permettant de préformater les informations pour une présentation dans une page web par exemple :

```
SELECT
 a.TABLE_SCHEMA,
 a.TABLE_NAME,
 c.ORDINAL_POSITION,
 c.COLUMN_NAME AS FIELD,
 case when c.DATA_TYPE = 'TIMESTMP' then 'TIMESTAMP' else (
   case when c.DATA_TYPE = 'VARC' then 'VARCHAR' else  (
       case when c.DATA_TYPE = 'VARG' then 'VARGRAPHIC' else c.DATA_TYPE  end
 ) end ) end as DATA_TYPE,
 c.DATA_TYPE_LENGTH as LENGTH,
 c.NUMERIC_SCALE as SCALE,
 c.NUMERIC_PRECISION,
 c.IS_NULLABLE AS COLUMN_NULLABLE,
 c."CCSID" as COLUMN_CCSID,
 c.SYSTEM_COLUMN_NAME,
 c.COLUMN_HEADING,
 c.COLUMN_TEXT,
 c.HAS_DEFAULT,
 c.COLUMN_DEFAULT,
 c.ALLOCATE,
 c.IS_IDENTITY, c.IDENTITY_GENERATION, c.IDENTITY_START,
 c.IDENTITY_INCREMENT, c.IDENTITY_MINIMUM, c.IDENTITY_MAXIMUM,
 c.IDENTITY_CYCLE, c.IDENTITY_CACHE, c.IDENTITY_ORDER
FROM QSYS2.systables A,
   TABLE (QSYS2.QSQSYSCOL2(A.system_table_schema, A.system_table_name) ) AS c
WHERE a.TABLE_SCHEMA = 'MABIB' AND a.TABLE_NAME = 'MATABLE' ;
```

IMPORTANT : la requête ci-dessus fonctionne aussi bien sur des tables que sur des vues DB2, dès lors que l'on souhaite simplement connaître la liste et la structure des colonnes renvoyée par une vue.

Récupérer la liste des indexs liés à une table se fait très simplement au moyen de la requête suivante :

```
SELECT X.INDEX_NAME, X.INDEX_SCHEMA,
X.SYSTEM_INDEX_NAME, X.SYSTEM_INDEX_SCHEMA,
'YES' as INDEX_SQL,
IS_UNIQUE AS INDEX_TYPE,
0 AS EVI_DISTINCT_VALUES
FROM QSYS2.SYSINDEXES X
WHERE X.TABLE_SCHEMA = 'MABIB' AND X.TABLE_NAME = 'MATABLE' ;
```

On peut récupérer le détail du tri défini par un index au moyen de la requête suivante :

```
SELECT SUBSTR(COLUMN_NAME, 1, 30) AS COLUMN_NAME, ORDERING FROM QSYS2.SYSKEYS
WHERE INDEX_SCHEMA = 'MABIB' AND INDEX_NAME = 'MATABLE_INDEX_xx'
ORDER BY ORDINAL_POSITION
;
```

2.24.2 SYSVIEWS et SYSVIEWDEP

La table système QSYS2.SYSVIEWS contient la plupart des éléments permettant de recréer une vue si besoin, à commencer par la colonne VIEW_DEFINITION qui contient le code source de création de la vue :

N°	Nom de colonne (long)	Nom court	Type	Longueur	Déc.
1	TABLE_NAME	NAME	VARCHAR	128	
2	VIEW_OWNER	CREATOR	VARCHAR	128	
3	SEQNO	SEQNO	INTEGER	9	0
4	CHECK_OPTION	CHECK	CHAR	1	
5	VIEW_DEFINITION	TEXT	VARGRAPHIC	5000	
6	IS_UPDATABLE	UPDATES	CHAR	1	
7	TABLE_SCHEMA	DBNAME	VARCHAR	128	
8	SYSTEM_VIEW_NAME	SYS_VNAME	CHAR	10	
9	SYSTEM_VIEW_SCHEMA	SYS_VDNAME	CHAR	10	
10	IS_INSERTABLE_INTO	INSERTABLE	VARCHAR	3	
11	IASP_NUMBER	IASPNUMBER	SMALLINT	4	0
12	IS_DELETABLE	DELETES	CHAR	1	
13	VIEW_DEFINER	DEFINER	VARCHAR	128	
14	ROUNDING_MODE	DECFLTRND	CHAR	1	

On peut donc afficher le contenu d'une vue au moyen de la requête suivante :

```
SELECT TABLE_NAME, VIEW_DEFINITION
FROM QSYS2.SYSVIEWS
WHERE TABLE_SCHEMA = 'MABIB' AND TABLE_NAME = 'MAVUE' ;
```

A noter que la requête ci-dessous, présentée également dans le chapitre précédent, fonctionne aussi bien pour les vues que pour les tables, pour l'affichage des colonnes renvoyées par une table ou une vue :

```
SELECT
 a.TABLE_SCHEMA,
 a.TABLE_NAME,
 c.ORDINAL_POSITION,
 c.COLUMN_NAME AS FIELD,
 case when c.DATA_TYPE = 'TIMESTMP' then 'TIMESTAMP' else (
   case when c.DATA_TYPE = 'VARC' then 'VARCHAR' else  (
       case when c.DATA_TYPE = 'VARG' then 'VARGRAPHIC' else c.DATA_TYPE  end
 ) end ) end as DATA_TYPE,
 c.DATA_TYPE_LENGTH as LENGTH,
 c.NUMERIC_SCALE as SCALE,
 c.NUMERIC_PRECISION,
 c.IS_NULLABLE AS COLUMN_NULLABLE,
```

```
 c."CCSID" as COLUMN_CCSID,
 c.SYSTEM_COLUMN_NAME,
 c.COLUMN_HEADING,
 c.COLUMN_TEXT,
 c.HAS_DEFAULT,
 c.COLUMN_DEFAULT,
 c.ALLOCATE,
 c.IS_IDENTITY, c.IDENTITY_GENERATION, c.IDENTITY_START,
 c.IDENTITY_INCREMENT, c.IDENTITY_MINIMUM, c.IDENTITY_MAXIMUM,
 c.IDENTITY_CYCLE, c.IDENTITY_CACHE, c.IDENTITY_ORDER
FROM QSYS2.systables A,
   TABLE (QSYS2.QSQSYSCOL2(A.system_table_schema, A.system_table_name) ) AS c
WHERE a.TABLE_SCHEMA = 'MABIB' AND a.TABLE_NAME = 'MATABLE' ;
```

La table SYSVIEWDEP permet de connaître la liste des objets DB2 dépendants de chacune des vues :

N°	Nom de colonne (long)	Nom court	Type	Longueur	Déc.
1	VIEW_NAME	DNAME	VARCHAR	128	
2	VIEW_OWNER	DCREATOR	VARCHAR	128	
3	OBJECT_NAME	ONAME	VARCHAR	128	
4	OBJECT_SCHEMA	OSCHEMA	VARCHAR	128	
5	OBJECT_TYPE	OTYPE	CHAR	24	
6	VIEW_SCHEMA	DDBNAME	VARCHAR	128	
7	SYSTEM_VIEW_NAME	SYS_VNAME	CHAR	10	
8	SYSTEM_VIEW_SCHEMA	SYS_VDNAME	CHAR	10	
9	SYSTEM_TABLE_NAME	SYS_TNAME	CHAR	10	
10	SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR	10	
11	TABLE_NAME	BNAME	VARCHAR	128	
12	TABLE_OWNER	BCREATOR	VARCHAR	128	
13	TABLE_SCHEMA	BDBNAME	VARCHAR	128	
14	TABLE_TYPE	BTYPE	CHAR	1	
15	IASP_NUMBER	IASPNUMBER	SMALLINT	4	0
16	PARM_SIGNATURE	SIGNATURE	VARCHAR	10000	

La requête ci-dessous permet d'identifier la liste des objets DB2 qui sont directement dépendants d'une vue :

```
SELECT *
FROM QSYS2.SYSVIEWDEP ;
WHERE VIEW_SCHEMA = 'MABIB' AND VIEW_NAME = 'MAVUE'
;
```

Dans le chapitre qui s'intitule "hiérarchie récursive", on a vu de quelle manière il était possible d'identifier les liens hiérarchiques entre différents lignes d'une table. On peut décliner cette technique pour récupérer l'exhaustivité des dépendances liées à une vue, quel que soit le

niveau de profondeur de ces dépendances :

```
SELECT LEVEL, VIEW_NAME, VIEW_SCHEMA,  OBJECT_NAME, OBJECT_SCHEMA,
      SUBSTR(OBJECT_TYPE, 1, 1) AS OBJECT_TYPE
FROM QSYS2.SYSVIEWDEP
WHERE VIEW_SCHEMA = 'MABIB'
START WITH VIEW_NAME = 'MAVUE'
CONNECT BY NOCYCLE PRIOR VIEW_NAME = OBJECT_NAME
;
```

2.24.3 SYSROUTINES et SYSROUTINEDEP

La table système QSYS2.SYSROUTINES contient la plupart des éléments permettant de recréer une UDF (user defined function) ou une procédure stockée.

N°	Nom de colonne (long)	Nom court	Type	Longueur	Déc.
1	SPECIFIC_SCHEMA	SPECSCHEMA	VARCHAR	128	
2	SPECIFIC_NAME	SPECNAME	VARCHAR	128	
3	ROUTINE_SCHEMA	RTNSCHEMA	VARCHAR	128	
4	ROUTINE_NAME	RTNNAME	VARCHAR	128	
5	ROUTINE_TYPE	RTNTYPE	VARCHAR	9	
6	ROUTINE_CREATED	RTNCREATE	TIMESTAMP	10	
7	ROUTINE_DEFINER	DEFINER	VARCHAR	128	
8	ROUTINE_BODY	BODY	VARCHAR	8	
9	EXTERNAL_NAME	EXTNAME	VARCHAR	279	
10	EXTERNAL_LANGUAGE	LANGUAGE	VARCHAR	8	
11	PARAMETER_STYLE	PARM_STYLE	VARCHAR	7	
12	IS_DETERMINISTIC	DETERMINE	VARCHAR	3	
13	SQL_DATA_ACCESS	DATAACCESS	VARCHAR	8	
14	SQL_PATH	SQL_PATH	VARCHAR	3483	
15	PARM_SIGNATURE	SIGNATURE	VARCHAR	2048	
16	NUMBER_OF_RESULTS	NUMRESULTS	SMALLINT	4	0
17	MAX_DYNAMIC_RESULT_SETS	RESULTS	SMALLINT	4	0
18	IN_PARMS	IN_PARMS	SMALLINT	4	0
19	OUT_PARMS	OUT_PARMS	SMALLINT	4	0
20	INOUT_PARMS	INOUT_PARM	SMALLINT	4	0
21	PARSE_TREE	PARSE_TREE	VARCHAR	1024	
22	PARM_ARRAY	PARM_ARRAY	BLOB	320000	
23	LONG_COMMENT	REMARKS	VARGRAPHIC	2000	
24	ROUTINE_DEFINITION	ROUTINEDEF	DBCLOB	2097152	
25	FUNCTION_ORIGIN	ORIGIN	CHAR	1	
26	FUNCTION_TYPE	TYPE	CHAR	1	
27	EXTERNAL_ACTION	EXT_ACTION	CHAR	1	
28	IS_NULL_CALL	NULL_CALL	VARCHAR	3	
29	SCRATCH_PAD	SCRATCHPAD	INTEGER	9	0
30	FINAL_CALL	FINAL_CALL	VARCHAR	3	
31	PARALLELIZABLE	PARALLEL	VARCHAR	3	
32	DBINFO	DBINFO	VARCHAR	3	
33	SOURCE_SPECIFIC_SCHEMA	SRCSCHEMA	VARCHAR	128	
34	SOURCE_SPECIFIC_NAME	SRCNAME	VARCHAR	128	
35	IS_USER_DEFINED_CAST	CAST_FUNC	VARCHAR	3	
36	CARDINALITY	CARD	BIGINT	18	0

37	FENCED	FENCED	VARCHAR	3	
38	COMMIT_ON_RETURN	CMTONRET	VARCHAR	3	
39	IASP_NUMBER	IASPNUMBER	SMALLINT	4	0
40	NEW_SAVEPOINT_LEVEL	NEWSAVEPTL	VARCHAR	3	
41	LAST_ALTERED	ALTEREDTS	TIMESTAMP	10	
42	DEBUG_MODE	DEBUG_MODE	CHAR	1	
43	DEBUG_DATA	DEBUG_DATA	CLOB	1048576	
44	ROUNDING_MODE	DECFLTRND	CHAR	1	
45	ROUTINE_TEXT	LABEL	VARGRAPHIC	50	
46	ROUTINE_ENVIRONMENT	RTN_ENV	BLOB	16777216	
47	ROUTINE_DEFAULT_QDT	RTNDFTQDT	BLOB	1048576	

La requête suivante permet de connaître la liste des objets DB2 utilisés par une procédure stockée (ou une UDF) :

```
SELECT DISTINCT OBJECT_NAME, OBJECT_TYPE, OBJECT_SCHEMA
FROM QSYS2.SYSROUTINEDEP
WHERE SPECIFIC_SCHEMA = 'MABIB' and SPECIFIC_NAME = 'MAPROCEDURE'
ORDER BY OBJECT_TYPE DESC, OBJECT_NAME
;
```

On peut utiliser la technique récursive vue au chapitre précédent, pour identifier l'exhaustivité des dépendances d'une procédure stockée (incluant les dépendances des vues utilisées s'il y en a) :

```
WITH CTE_ROUTINEDEP_ALL AS
(
SELECT DISTINCT OBJECT_NAME, OBJECT_TYPE
FROM QSYS2.SYSROUTINEDEP
WHERE SPECIFIC_SCHEMA = 'MABIB_PROC' AND SPECIFIC_NAME = 'MAPROCEDURE'
ORDER BY OBJECT_NAME
) ,
CTE_ROUTINEDEP_TAB AS (
SELECT DISTINCT OBJECT_NAME, OBJECT_TYPE
FROM CTE_ROUTINEDEP_ALL
WHERE OBJECT_TYPE = 'TABLE'
ORDER BY OBJECT_NAME
) ,
CTE_DEPENDANCES AS (
SELECT LEVEL, VIEW_NAME, VIEW_SCHEMA,  OBJECT_NAME, OBJECT_SCHEMA,
      SUBSTR(OBJECT_TYPE, 1, 1) AS OBJECT_TYPE
FROM QSYS2.SYSVIEWDEP
WHERE VIEW_SCHEMA = 'MABIB_DATA'
START WITH VIEW_NAME IN (SELECT OBJECT_NAME FROM CTE_ROUTINEDEP_TAB)
CONNECT BY NOCYCLE PRIOR VIEW_NAME = OBJECT_NAME
)
SELECT * FROM CTE_DEPENDANCES ;
```

2.24.4 SYSTABLESTAT

La table système SYSTABLESTAT est très pratique pour connaître le nombre de lignes présentes et/ou supprimées à l'intérieur d'une table.

Exemple de requête permettant d'identifier les écarts - en termes de nombre de lignes - entre 2 bibliothèques (MA_BIB_1 et MA_BIB_2), pour toutes les tables dont le nom est préfixé en "DIM":

```
WITH TMPSTAT AS (
    SELECT A.TABLE_SCHEMA, A.TABLE_NAME, A.NUMBER_ROWS AS NUMBER_ROWS_APPBIB2,
      (SELECT B.NUMBER_ROWS FROM QSYS2.SYSTABLESTAT B
         WHERE A.TABLE_NAME = B.TABLE_NAME AND B.TABLE_SCHEMA = 'MA_BIB_1'
      ) AS NUMBER_ROWS_APPBIB1
    FROM QSYS2.SYSTABLESTAT A
    WHERE
    A.TABLE_SCHEMA = 'MA_BIB_2'
    AND SUBSTR(A.TABLE_NAME, 1, 3) = 'DIM'
)
SELECT * FROM TMPSTAT
WHERE NUMBER_ROWS_APPBIB2 > 0 AND NUMBER_ROWS_APPBIB2 <> NUMBER_ROWS_APPBIB1
ORDER BY TABLE_NAME
;
```

Structure de la table SYSTABLESTAT en V7R1 :

N°	Nom de colonne	Type	Longueur	Déc.
1	TABLE_SCHEMA	VARCHAR	128	
2	TABLE_NAME	VARCHAR	128	
3	PARTITION_TYPE	CHAR	1	
4	NUMBER_PARTITIONS	INTEGER	9	0
5	NUMBER_DISTRIBUTED_PARTITIONS	INTEGER	9	0
6	NUMBER_ROWS	BIGINT	18	0
7	NUMBER_ROW_PAGES	BIGINT	18	0
8	NUMBER_PAGES	BIGINT	18	0
9	OVERFLOW	BIGINT	18	0
10	CLUSTERED	CHAR	1	
11	ACTIVE_BLOCKS	BIGINT	18	0
12	AVGCOMPRESSEDROWSIZE	BIGINT	18	0
13	AVGROWCOMPRESSIONRATIO	FLOAT	29	
14	AVGROWSIZE	BIGINT	18	0
15	PCTROWSCOMPRESSED	FLOAT	29	
16	PCTPAGESSAVED	SMALLINT	4	0

17	NUMBER_DELETED_ROWS	BIGINT	18	0
18	DATA_SIZE	BIGINT	18	0
19	VARIABLE_LENGTH_SIZE	BIGINT	18	0
20	FIXED_LENGTH_EXTENTS	BIGINT	18	0
21	VARIABLE_LENGTH_EXTENTS	BIGINT	18	0
22	COLUMN_STATS_SIZE	BIGINT	18	0
23	MAINTAINED_TEMPORARY_INDEX_SIZE	BIGINT	18	0
24	NUMBER_DISTINCT_INDEXES	INTEGER	9	0
25	OPEN_OPERATIONS	BIGINT	18	0
26	CLOSE_OPERATIONS	BIGINT	18	0
27	INSERT_OPERATIONS	BIGINT	18	0
28	UPDATE_OPERATIONS	BIGINT	18	0
29	DELETE_OPERATIONS	BIGINT	18	0
30	CLEAR_OPERATIONS	BIGINT	18	0
31	COPY_OPERATIONS	BIGINT	18	0
32	REORGANIZE_OPERATIONS	BIGINT	18	0
33	INDEX_BUILDS	BIGINT	18	0
34	LOGICAL_READS	BIGINT	18	0
35	PHYSICAL_READS	BIGINT	18	0
36	SEQUENTIAL_READS	BIGINT	18	0
37	RANDOM_READS	BIGINT	18	0
38	LAST_CHANGE_TIMESTAMP	TIMESTAMP	10	
39	LAST_SAVE_TIMESTAMP	TIMESTAMP	10	
40	LAST_RESTORE_TIMESTAMP	TIMESTAMP	10	
41	LAST_USED_TIMESTAMP	TIMESTAMP	10	
42	DAYS_USED_COUNT	INTEGER	9	0
43	LAST_RESET_TIMESTAMP	TIMESTAMP	10	
44	NUMBER_PARTITIONING_KEYS	INTEGER	9	0
45	PARTITIONING_KEYS	VARCHAR	2880	
46	SYSTEM_TABLE_SCHEMA	CHAR	10	
47	SYSTEM_TABLE_NAME	CHAR	10	

3. Procédures stockées

3.1 compilation et exécution

Principe de compilation des procédures stockées :

Le principe suivant est valable également pour les UDF et UDTF :

Si vous êtes en mode 5250, et en supposant que le source de la procédure se trouve dans l'IFS, charger au préalable la liste des bibliothèques par EDTLIBL, puis lancer la commande de compilation :

```
RUNSQLSTM SRCSTMF('/M3DHSADSRC/P00AA00PR1.SQL') COMMIT(*NONE)
```

(dans l'exemple ci-dessus, le source est stocké dans un fichier de l'IFS)

Si vous exécutez votre requête dans System i Navigator, il faut procéder par étapes :

1 - sélectionner le type connexion *SYSTEME, et non pas *SQL

2 - charger la liste des bibliothèques en mode "ligne de commande" de la façon suivante :
cl: addlible LIBRARY1 ;
cl: addlible LIBRARY2 ;
cl: addlible LIBRARY3 ;

3 - exécution du code de compilation

Il est également possible d'encapsuler l'appel d'une procédure stockée à l'intérieur d'un programme RPG, comme dans l'exemple ci-dessous, où le nom de la procédure et son unique paramètre sont transmis au programme RPG qui les utilise pour effectuer le CALL :

```
Dcono             s              3  0
Dprocstocke       s             10
c     *entry        plist
c                   parm                    procstocke
c                   parm                    cono
 /free
    exec sql call :procstocke  (:cono) ;
    *inlr = *on;
```

On peut aussi faire un CALL de procédure stockée l'intérieur d'un programme CL via l'ordre RUNSQL ou via QSH. Dans ce cas de figure, on devra concaténer les éventuels paramètres dans la chaîne de caractères contenant le CALL, ce qui est une opération généralement assez pénible.

3.2 Premier exemple

L'exemple de procédure "full SQL" ci-dessous reçoit un pourcentage d'augmentation de salaires, calcule le total des salaires avant et après augmentation, et renvoie la différence entre ces 2 totaux en sortie :

```
CREATE OR REPLACE PROCEDURE DB2SAMPL.SP_Update_Salary
  (IN percentage DECIMAL(5,2), OUT extraCost DECIMAL(11,2))
LANGUAGE SQL
BEGIN
  -- Declare variables
  DECLARE v_sumOldSalaries DECIMAL(11,2) DEFAULT 0;
  DECLARE v_sumNewSalaries DECIMAL(11,2) DEFAULT 0;
  DECLARE reqsql VARCHAR(1000);

  -- Select the total value of current salaries
  SELECT SUM(salary) INTO v_sumOldSalaries FROM DB2SAMPL.HQEMPLOYEE;

  -- Update the salary with given input parameter
  SET reqsql = 'UPDATE DB2SAMPL.HQEMPLOYEE SET SALARY = SALARY * (100 + ?)/100';
  PREPARE stmt1 FROM reqsql;
  EXECUTE stmt1 USING percentage;

  -- Select the total value of new salaries
  SELECT SUM(salary) INTO v_sumNewSalaries FROM DB2SAMPL.HQEMPLOYEE;
  -- pour information, la requête ci-dessous est strictement équivalente à la requête
précédente
  -- SET v_sumNewSalaries = (SELECT SUM(salary) FROM FORMATION/MY_EMP);

  -- Set the OUT paramater as the difference of two salaries.
  SET extraCost = v_sumNewSalaries - v_sumOldSalaries;
  ROLLBACK;
END ;

-- Test :
CALL SP_Update_Salary(20.00, ?);
```

Points à noter :
A partir de la V7R1, il devient possible d'utiliser la syntaxe "CREATE OR REPLACE", telle qu'elle est présentée dans l'exemple ci-dessus.
Toujours à partir de la V7R1, il devient possible de définir des valeurs par défaut pour certains paramètres d'une procédure stockée, comme dans l'exemple suivant :

```
CREATE OR REPLACE PROCEDURE FORMATION/MA_PROC (
       IN VCODSOC DECIMAL(3, 0) ,
       IN VPERIOD CHAR(1) DEFAULT 'J' )
...
```

3.3 Procédure stockée DB2 "full SQL"

Canevas de procédure stockée "full SQL" pouvant être utilisé en environnement IBM i :

```
CREATE PROCEDURE FORMATION/PROC_MODL (
     IN VALCONO DECIMAL(3, 0) )
     LANGUAGE SQL
     SPECIFIC FORMATION/PROC_MODL
     NOT DETERMINISTIC
     MODIFIES SQL DATA
     CALLED ON NULL INPUT
     SET OPTION ALWBLK = *ALLREAD ,
     -- TGTRLS = *CURRENT ,
     ALWCPYDTA = *OPTIMIZE ,
     COMMIT = *NONE ,
     CLOSQLCSR = *ENDMOD ,
     DATFMT = *ISO ,
     TIMFMT = *ISO ,
     DECMPT = *JOB ,
     DECRESULT = (31, 31, 00) ,
     DFTRDBCOL = *NONE ,
     DYNDFTCOL = *NO ,
     DYNUSRPRF = *USER ,
     SRTSEQ = *HEX ,
     OUTPUT = *PRINT ,
     DBGVIEW = *SOURCE
     BEGIN
       -- Déclaration du nom de la procédure courante (pour gestion des logs)
       DECLARE V_NOM_PROC CHAR(20) DEFAULT 'PROC_MODL';
       -- Déclaration du contexte applicatif (pour gestion des logs)
       DECLARE V_CONTEXT VARCHAR(30) DEFAULT '';
       -- Déclaration de la variable définissant l'étape courante (pour log)
       DECLARE V_STEP_NUM INTEGER DEFAULT 0;
       DECLARE V_STEP_DEB TIMESTAMP ;
       -- Déclaration de la variable servant à stocker le numéro de job courant
       DECLARE V_JOB_NUM INTEGER DEFAULT 0;
       -- Déclaration nb d'enregistrements retournés par GET DIAGNOSTICS (ROWCOUNT)
       DECLARE V_NBR_ENR INTEGER DEFAULT 0;
       -- Déclaration des "SQL return codes"
       DECLARE SQLCODE INTEGER DEFAULT 0;
       DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
        -- Déclaration des variables destinées à alimenter la log si erreurs SQL
       DECLARE V_SQL_STATE CHAR(5) DEFAULT '00000';
       DECLARE V_SQL_CODE INT DEFAULT 0;
       DECLARE V_MSG_TXT VARCHAR(100) DEFAULT '';
       DECLARE V_MSG_TXTLEN INTEGER DEFAULT 0;
       -- Déclaration de variables utilisables sur des requêtes dynamiques
       DECLARE SQL_STMT VARCHAR(2048) DEFAULT '';
       DECLARE V_PARAM1 CHAR(10) DEFAULT '';
       DECLARE V_PARAM2 CHAR(10) DEFAULT '';
```

```
       DECLARE V_PARAM3 CHAR(10) DEFAULT '';
       DECLARE V_PARNU1 INTEGER DEFAULT 0;
       DECLARE V_PARNU2 INTEGER DEFAULT 0;
       DECLARE V_PARNU3 INTEGER DEFAULT 0;
       -- Déclaration de jeux de 2 jeux de date prévus pour divers usages (chaque
       --    date dans 2 formats)
       -- Date de valeur
       DECLARE V_DATVAL DATE ;
       DECLARE V_DATVAL8 DEC(8, 0) DEFAULT 0 ;
       -- Date de période (utile pour traitements hebdo et mensuels notamment)
       DECLARE V_DATPER DATE ;
       DECLARE V_DATPER8 DEC(8, 0) DEFAULT 0 ;

       -- Pavé de gestion des Données non trouvées
       DECLARE CONTINUE HANDLER FOR NOT FOUND
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
         CALL PRCTRACLOG
           (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
SQLSTATE, SQLCODE, 'NOTFOUND', '', V_CONTEXT ) ;
       END ;

        -- Pavé de gestion des Avertissements
       DECLARE CONTINUE HANDLER FOR SQLWARNING
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
         CALL PRCTRACLOG
           (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
SQLSTATE, SQLCODE, 'WARNING', '', V_CONTEXT ) ;
       END ;

       -- Pavé de gestion des Erreurs
       DECLARE EXIT HANDLER FOR SQLEXCEPTION
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
         GET DIAGNOSTICS EXCEPTION 1
            V_MSG_TXT    = MESSAGE_TEXT,
            V_MSG_TXTLEN = MESSAGE_LENGTH;
            IF V_MSG_TXTLEN > 100 THEN
                SET V_MSG_TXT = SUBSTR(V_MSG_TXT, 1, 100) ;
            END IF ;
         CALL PRCTRACLOG
            (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
V_SQL_STATE, V_SQL_CODE, 'ERROR', V_MSG_TXT, V_CONTEXT );
       END ;

       -- Valeur par défaut indispensable pour V_STEP_DEB (en cas d'erreur avant
       --    l'ouverture de la 1ère step)
       SET V_STEP_DEB = CURRENT TIMESTAMP ;
```

```
       -- Stockage du contexte applicatif
       SET V_CONTEXT = VALCONO CONCAT '/' CONCAT '*' ;

       -- incrémentation du compteur alimentant le numéro de job (indispensable pour
       --   les logs)
       CALL PRCTRACJOB ( V_JOB_NUM ) ;

        -- Traitement des "règles métiers" - Début

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

-----------------------------------------------------
-- Exemple de requête "métier"
-----------------------------------------------------
Delete from ma_table  ;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

-----------------------------------------------------
-- Exemple de requête "métier"
-----------------------------------------------------
INSERT INTO ma_table (col1, col2, ...)
SELECT col1a, col2a, ... FROM ... ;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

        -- Traitement des "règles métiers" - Fin
```

```
END  ;
COMMENT ON SPECIFIC PROCEDURE FORMATION/PROC_MODL
     IS 'Procédure exemple' ;
```

3.4 PRCTRACLOG

Dans le canevas de procédure du chapitre précédent, on utilise une procédure PRCTRACLOG pour "monitorer" différents évènements (avertissements et anomalies) et conserver une trace de chaque requête SQL (en termes de durée et de nombre de lignes impactées). Nous allons voir comment créer la table de log PRC_TRACE et sa procédure d'alimentation PRCTRACLOG.

Code souce de création de la table PRC_TRACE :

```
CREATE TABLE FORMATION/PRC_TRACE (
T_NOM_SERV CHAR(20) CCSID 297 NOT NULL,
T_NUM_JOB INTEGER NOT NULL,
T_NOM_PROC CHAR(20) CCSID 297 NOT NULL,
T_STEP_NUM INTEGER NOT NULL,
T_STEP_DEB TIMESTAMP NOT NULL,
T_STEP_FIN TIMESTAMP NOT NULL,
T_NBR_ENR BIGINT NOT NULL,
T_SQL_CODE INTEGER NOT NULL,
T_SQL_STATE CHAR(5) CCSID 297 NOT NULL,
T_MSG_TYP CHAR(10) CCSID 297 NOT NULL DEFAULT '',
T_MSG_TXT VARCHAR(100) CCSID 297 NOT NULL DEFAULT '',
T_USER CHAR(20) CCSID 297 NOT NULL,
T_CRE_LOG TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
T_CONTEXT VARCHAR(30) CCSID 297 NOT NULL DEFAULT '',
T_CONO DECIMAL(3, 0) NOT NULL DEFAULT 0
) ;
COMMENT ON TABLE FORMATION/PRC_TRACE
IS 'Log sur procédures stockées SQL';

LABEL ON TABLE FORMATION/PRC_TRACE
IS 'Log sur procédures stockées SQL';

LABEL ON COLUMN FORMATION/PRC_TRACE (
T_NOM_SERV IS 'Nom seveur',
T_NUM_JOB IS 'Numéro job SQL',
T_NOM_PROC IS 'Nom procédure',
T_STEP_NUM IS 'N° Etape',
T_STEP_DEB IS 'Début Etape',
T_STEP_FIN IS 'Fin Etape',
T_NBR_ENR IS 'Nbre enreg.',
T_SQL_CODE IS 'SQL Code',
T_SQL_STATE IS 'SQL Status',
T_MSG_TYP IS 'Type Message SQL',
T_MSG_TXT IS 'Message SQL',
T_USER IS 'Utilisateur',
T_CRE_LOG IS 'Date création log.' ) ;

LABEL ON COLUMN FORMATION/PRC_TRACE (
```

```
T_NOM_SERV TEXT IS 'Nom seveur',
T_NUM_JOB TEXT IS 'Numéro job SQL',
T_NOM_PROC TEXT IS 'Nom procédure',
T_STEP_NUM TEXT IS 'N° Etape',
T_STEP_DEB TEXT IS 'Début Etape',
T_STEP_FIN TEXT IS 'Fin Etape',
T_NBR_ENR TEXT IS 'Nbre enreg.',
T_SQL_CODE TEXT IS 'SQL Code',
T_SQL_STATE TEXT IS 'SQL Status',
T_MSG_TYP TEXT IS 'Type Message SQL',
T_MSG_TXT TEXT IS 'Message SQL',
T_USER TEXT IS 'Utilisateur',
T_CRE_LOG TEXT IS 'Date création log.' ) ;

CREATE INDEX FORMATION/PRC_TRACE_L01 ON FORMATION/PRC_TRACE(T_NUM_JOB, T_STEP_NUM ) ;
CREATE INDEX FORMATION/PRC_TRACE_L02 ON FORMATION/PRC_TRACE(T_NOM_PROC, T_NUM_JOB,
T_STEP_NUM ) ;
CREATE INDEX FORMATION/PRC_TRACE_L03 ON FORMATION/PRC_TRACE(T_NOM_PROC,
T_STEP_DEB ) ;
CREATE INDEX FORMATION/PRC_TRACE_L04 ON FORMATION/PRC_TRACE(T_NUM_JOB, T_STEP_DEB ) ;
```

Le code source de la procédure PRCTRACLOG est présenté ci-dessous :

On notera dans le code source l'utilisation de SQL dynamique pour le traitement de la requête d'insertion dans la table PRC_TRACE. Les points d'interrogation situés dans la requête sont remplacés lors du "EXECUTE" par les valeurs transmises à la suite du mot clé "USING" :

```
-- ligne à exécuter avant compilation
cl: addlible formation ;

-- DROP PROCEDURE FORMATION/PRCTRACLOG ;
CREATE PROCEDURE FORMATION/PRCTRACLOG
    (
      IN P_NUM_JOB   integer ,
      IN P_NOM_PROC  char(20) ,
      IN P_STEP_NUM  integer ,
      IN P_STEP_DEB  timestamp ,
      IN P_STEP_FIN  timestamp ,
      IN P_NBR_ENR   bigint ,
      IN P_SQL_STATE char(5) ,
      IN P_SQL_CODE  integer ,
      IN P_MSG_TYP   char(10) ,
      IN P_MSG_TXT   varchar(100),
      IN P_CONTEXT   varchar(30)
      )
      LANGUAGE SQL
      SPECIFIC FORMATION/PRCTRACLOG
      NOT DETERMINISTIC
      MODIFIES SQL DATA
      CALLED ON NULL INPUT
```

```
       SET OPTION ALWBLK = *ALLREAD ,
       -- TGTRLS = *CURRENT ,
       ALWCPYDTA = *OPTIMIZE ,
       COMMIT = *NONE ,
       CLOSQLCSR = *ENDMOD ,
       DATFMT = *ISO ,
       TIMFMT = *ISO ,
       DECMPT = *JOB ,
       DECRESULT = (31, 31, 00) ,
       DFTRDBCOL = *NONE ,
       DYNDFTCOL = *NO ,
       DYNUSRPRF = *USER ,
       SRTSEQ = *HEX ,
       OUTPUT = *PRINT ,
       DBGVIEW = *SOURCE ,
       ALWCPYDTA = *OPTIMIZE ,
       ALWBLK = *ALLREAD
       BEGIN

      -- Déclaration de registres DB2 utilisables dans les logs
      DECLARE V_USER CHAR(20) DEFAULT '' ;
      DECLARE V_SERV CHAR(20) DEFAULT '' ;
      DECLARE V_CONO INTEGER DEFAULT 0 ;
      DECLARE stmt VARCHAR(512);

     SET V_CONO = CASE WHEN SUBSTR(P_CONTEXT, 3, 1) = ' ' OR SUBSTR(P_CONTEXT, 3, 1)
= '/'
          THEN   CAST(SUBSTR(P_CONTEXT, 1, 2) AS INTEGER)
          ELSE    CAST(SUBSTR(P_CONTEXT, 1, 3) AS INTEGER)
      END ;

      SET V_USER = USER ;
      SET V_SERV = CURRENT SERVER ;

      SET stmt = 'INSERT INTO PRC_TRACE (T_NOM_SERV, T_NUM_JOB, T_NOM_PROC,
T_STEP_NUM, T_STEP_DEB, T_STEP_FIN, T_NBR_ENR, T_SQL_STATE, T_SQL_CODE, T_MSG_TYP,
T_MSG_TXT, T_USER, T_CONTEXT, T_CONO) VALUES
(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';

      PREPARE s1 FROM stmt;
      EXECUTE s1 USING V_SERV, P_NUM_JOB, P_NOM_PROC, P_STEP_NUM, P_STEP_DEB,
P_STEP_FIN, P_NBR_ENR, P_SQL_STATE, P_SQL_CODE, P_MSG_TYP, P_MSG_TXT, V_USER,
P_CONTEXT, V_CONO ;

END ;

COMMENT ON SPECIFIC PROCEDURE FORMATION/PRCTRACLOG
IS 'Enregistrement log des procédures stockées' ;
```

3.5 PRCTRACJOB

En complément de l'exemple de procédure "full SQL", voici le code de la procédure PRCTRACJOB qui alimente un compteur permettant d'identifier chaque exécution d'une procédure stockée.

```
CREATE PROCEDURE FORMATION/PRCTRACJOB (
   INOUT V_JOB_NUM integer
)
       LANGUAGE SQL
       SPECIFIC FORMATION/PRCTRACJOB
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       SET OPTION ALWBLK = *ALLREAD ,
       -- TGTRLS = V5R4M0 ,
       ALWCPYDTA = *OPTIMIZE ,
       COMMIT = *NONE ,
       CLOSQLCSR = *ENDMOD ,
       DATFMT = *ISO ,
       TIMFMT = *ISO ,
       DECMPT = *JOB ,
       DECRESULT = (31, 31, 00) ,
       DFTRDBCOL = *NONE ,
       DYNDFTCOL = *NO ,
       DYNUSRPRF = *USER ,
       SRTSEQ = *HEX ,
       OUTPUT = *PRINT ,
       DBGVIEW = *SOURCE ,
       ALWCPYDTA = *OPTIMIZE ,
       ALWBLK = *ALLREAD
       BEGIN
        -- incrémentation du compteur alimentant le numéro de job (indispensable pour
les logs)
        SELECT NEXTVAL FOR SEQ_NJOBPR INTO V_JOB_NUM FROM SYSIBM/SYSDUMMY1 ;
       END ;

       COMMENT ON SPECIFIC PROCEDURE FORMATION/PRCTRACJOB
       IS 'Incr. compteur de job des procédures stockées' ;
```

Le compteur de travail est stocké dans une séquence DB2, dont voici le code de création :

```
-- drop sequence FORMATION/SEQ_NJOBPR ;
CREATE SEQUENCE FORMATION/SEQ_NJOBPR
       AS INTEGER
       START WITH 1
       INCREMENT BY 1
       MINVALUE 1
       MAXVALUE 2147483647
```

```
       NO CYCLE NO CACHE NO ORDER ;

COMMENT ON SEQUENCE FORMATION/SEQ_NJOBPR
       IS 'Compteur de job pour proc. stockées DB2' ;

LABEL ON SEQUENCE FORMATION/SEQ_NJOBPR
       IS 'Compteur de job pour proc. stockées DB2' ;
```

3.6 Result Set en PL/SQL

La production de "result set" à l'intérieur d'une procédure stockée, à destination d'un programme client (PHP, RPG, etc...) :

```
CREATE or replace PROCEDURE MY_LIBRARY.SP_Resultset_ToClient()
DYNAMIC RESULT SETS 1
LANGUAGE SQL
BEGIN
DECLARE c1 CURSOR
WITH RETURN
-- TO CLIENT    -- cette directive empêche la consommation du resultset par une autre
procédure stockée DB2
FOR SELECT EMPNO, SALARY
FROM MY_LIBRARY.EMPLOYEE ;
OPEN c1;
RETURN;
END
;

CALL MY_LIBRARY.SP_Resultset_ToClient() ;


-- la technique présentée dans la procédure ci-dessous ne fonctionne que si
-- la directive WITH RETURN TO CLIENT est activée dans la procédure appelée
CREATE or replace PROCEDURE MY_LIBRARY.SP_Resultset_OuterClient()
RESULT SETS 1
LANGUAGE SQL
BEGIN
CALL MY_LIBRARY.SP_Resultset_ToClient();
END
;

call MY_LIBRARY.SP_Resultset_OuterClient();


-- La procédure ci-dessous ne fonctionne que si la directive WITH RETURN TO CLIENT
-- est désactivée dans la procédure stockée à l'origine du resultset
CREATE or replace PROCEDURE MY_LIBRARY.SP_Resultset_Exploit()

LANGUAGE SQL
-- SET OPTION DBGVIEW=*SOURCE   (à activer pour le débogage)
BEGIN
DECLARE V_RSLOC1 RESULT_SET_LOCATOR VARYING;
DECLARE V_EMPID VARCHAR(6) ;
DECLARE V_SALARY DECIMAL(9, 2) ;
DECLARE V_ROWNOTFOUND INTEGER DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET V_ROWNOTFOUND = 1;

CALL MY_LIBRARY.SP_Resultset_ToClient() ;
```

```
ASSOCIATE RESULT SET LOCATORS(V_RSLOC1) WITH PROCEDURE
MY_LIBRARY.SP_Resultset_ToClient;

ALLOCATE C1 CURSOR FOR RESULT SET V_RSLOC1;

WHILE V_ROWNOTFOUND = 0 DO
   -- code "métier" à insérer ici
   FETCH C1 INTO V_EMPID , V_SALARY ;
END WHILE;

CLOSE C1 ;
END
;


call MY_LIBRARY.SP_Resultset_Exploit();
```

3.7 Result Set en SQLRPGLE

La production de "result set" à l'intérieur de programmes RPG, encapsulés dans des procédures stockées de type externe, est strictement identique au principe utilisé dans des procédures stockées DB2.

Deux études de cas en particulier proposent des exemple d'implémentation d'un "result set" (jeu de données) SQL produit par un programme RPG. Ce sont les études de cas suivantes :

- WrkObjLck version SQL
- WrkJobScde version SQL

Voici à titre d'information l'extrait de code RPG de l'étude de cas "WrkObjLck version SQL", dans lequel le "result set" est généré :

```
//*********************************************
// Si demandé par le programme appelant,
// génération d'un result set à partir de la
// table temporaire
//*********************************************
If ( Resultset = 'YES' ) ;
  sql3 = 'SELECT distinct Job_name, Job_user_name, Job_number, ' +
           'Lock_state, Lock_status, Lock_type, Member_name, ' +
           'Share, Lock_scope ' +
           'FROM QTEMP/OBJL0100 FOR FETCH ONLY ' ;
  EXEC SQL
    PREPARE REQ1 FROM :sql3 ;
  EXEC SQL
    DECLARE C1 CURSOR FOR REQ1 ;
  EXEC SQL
    OPEN C1 ;
  EXEC SQL
    SET RESULT SETS CURSOR C1 ;

Endif ;
```

3.8 Techniques avancées

Exemple de procédure stockée combinant plusieurs techniques telles que des curseurs et du SQL dynamique :

```
--  Alimentation des tables MF_PRIX_COUR et MF_PRIX_PREC selon contexte transmis via VALCONTEXT
--  si VALCONTEXT = 'STD' alors INSERT INTO MF_PRIX_COUR        (STD est la valeur par défaut)
--  si VALCONTEXT = 'N-1' alors INSERT INTO MF_PRIX_PREC
--  L'appel de cette procédure avec VALCONTEXT N-1 est assur? par la procédure J00GC09PR1
--   qui a été créée dans le but exclusif de simplifier l'appel de la procédure MY_PROCEDURE,
--   au sein de l'existant.

SET PATH *LIBL ;
CREATE OR REPLACE PROCEDURE MY_LIBRARY/J00GC08PR1 (
     IN VALCONO DECIMAL(3, 0),
     IN VALPERI CHAR(1) DEFAULT 'J',
     VALCONTEXT CHAR(10) DEFAULT 'STD' )
     LANGUAGE SQL
     SPECIFIC MY_LIBRARY/MY_PROCEDURE
     NOT DETERMINISTIC
     MODIFIES SQL DATA
     CALLED ON NULL INPUT
     SET OPTION ALWBLK = *ALLREAD ,
     -- TGTRLS = *CURRENT ,
     ALWCPYDTA = *OPTIMIZE ,
     COMMIT = *NONE ,
     CLOSQLCSR = *ENDMOD ,
     DATFMT = *ISO ,
     TIMFMT = *ISO ,
     DECMPT = *JOB ,
     DECRESULT = (31, 31, 00) ,
     DFTRDBCOL = *NONE ,
     DYNDFTCOL = *NO ,
     DYNUSRPRF = *USER ,
     SRTSEQ = *HEX ,
     OUTPUT = *PRINT ,
     DBGVIEW = *SOURCE
     BEGIN
       -- Déclaration du nom de la procédure courante (pour gestion des logs)
       DECLARE V_NOM_PROC CHAR(20) DEFAULT 'MY_PROCEDURE';
       -- Déclaration du nom de la table en cours de traitement
       DECLARE V_NOM_TABL CHAR(20) DEFAULT '';
       -- Déclaration du contexte applicatif (pour gestion des logs)
       DECLARE V_CONTEXT VARCHAR(30) DEFAULT '';
       -- Déclaration de la variable d?finissant l'étape courante (pour log)
       DECLARE V_STEP_NUM INTEGER DEFAULT 0;
       DECLARE V_STEP_DEB TIMESTAMP ;
       -- Déclaration de la variable servant à stocker le numéro de job courant
       DECLARE V_JOB_NUM INTEGER DEFAULT 0;
       -- Déclaration nombre d'enregistrements retournés par GET DIAGNOSTICS
(ROWCOUNT)
```

```
	DECLARE V_NBR_ENR INTEGER DEFAULT 0;
	-- Déclaration des "SQL return codes"
	DECLARE SQLCODE INTEGER DEFAULT 0;
	DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
	 -- Déclaration des variables destinées à alimenter la log en cas d'erreurs
SQL
	DECLARE V_SQL_STATE CHAR(5) DEFAULT '00000';
	DECLARE V_SQL_CODE INT DEFAULT 0;
	DECLARE V_MSG_TXT VARCHAR(100) DEFAULT '';
	DECLARE V_MSG_TXTLEN INTEGER DEFAULT 0;
	-- Déclaration de variables susceptibles d'être utilisées sur des requêtes
dynamiques
	DECLARE SQL_STMT VARCHAR(2048) DEFAULT '';
	DECLARE V_PARAM1 CHAR(10) DEFAULT '';
	DECLARE V_PARAM2 CHAR(10) DEFAULT '';
	DECLARE V_PARAM3 CHAR(10) DEFAULT '';
	DECLARE V_PARNU1 INTEGER DEFAULT 0;
	DECLARE V_PARNU2 INTEGER DEFAULT 0;
	DECLARE V_PARNU3 INTEGER DEFAULT 0;
	-- Déclaration de jeux de 2 jeux de date prévus pour divers usages (chaque
date dans 2 formats)
	-- Date de valeur
	DECLARE V_DATVAL DATE ;
	DECLARE V_DATVAL8 DEC(8, 0) DEFAULT 0 ;
	-- Date de période (utile pour traitements hebdo et mensuels notamment)
	DECLARE V_DATPER DATE ;
	DECLARE V_DATPER8 DEC(8, 0) DEFAULT 0 ;
	-- Variables pour curseur SQL dynamique
	DECLARE V_DYNSQL_CUR VARCHAR(500);
	DECLARE V_DYNSQL1   VARCHAR(500);
	DECLARE V_DYNSQL2   VARCHAR(500);
	DECLARE V_DYNSQL3   VARCHAR(500);

	DECLARE V_NBR_INS INTEGER DEFAULT 0;
	DECLARE V_ORI_TAR INTEGER DEFAULT 0;

	DECLARE ZXITNO CHAR(15) DEFAULT '';
	DECLARE ZXSTYN CHAR(15) DEFAULT '';
	DECLARE ZXHDPR CHAR(15) DEFAULT '';
	DECLARE FXDIVI CHAR(3) DEFAULT '' ;
	DECLARE FXITNO CHAR(15) DEFAULT '';
	DECLARE FXCUCD CHAR(3) DEFAULT '';
	DECLARE FXPRRF CHAR(2) DEFAULT '';
	DECLARE FXSAPR DECIMAL(17, 6) DEFAULT 0;
	DECLARE FXFVDT DECIMAL(8, 0) DEFAULT 0 ;
	DECLARE FXLVDT DECIMAL(8, 0) DEFAULT 0 ;
	DECLARE FXFVDT_DAT DATE ;
	DECLARE FXLVDT_DAT DATE ;
	DECLARE FXITNO_8 CHAR(8) DEFAULT '';

	-- Déclaration du curseur avant le corps de la procédure (sinon ne passe pas à
la compilation)
	DECLARE CUR1 CURSOR FOR V_DYNSTM_CUR;
```

```
       DECLARE CTAR CURSOR FOR V_DYNSTM2;

       -- Pavé de gestion des "NOT FOUND" supprimé pour éviter de saturer
       --  la table PRC_TRACE avec des messages inutiles générés par les
       --  curseurs de recherche de tarif
       -- (même technique appliquée dans J00GC10PR1)

       -- Pavé de gestion des Avertissements
       DECLARE CONTINUE HANDLER FOR SQLWARNING
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
         CALL PRCTRACLOG
           (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
SQLSTATE, SQLCODE, 'WARNING', '', V_CONTEXT ) ;
       END ;

       -- Pavé de gestion des Erreurs
       DECLARE EXIT HANDLER FOR SQLEXCEPTION
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
         GET DIAGNOSTICS EXCEPTION 1
            V_MSG_TXT    = MESSAGE_TEXT,
            V_MSG_TXTLEN = MESSAGE_LENGTH;
            IF V_MSG_TXTLEN > 100 THEN
                SET V_MSG_TXT = SUBSTR(V_MSG_TXT, 1, 100) ;
            END IF ;
         CALL PRCTRACLOG
            (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
V_SQL_STATE, V_SQL_CODE, 'ERROR', V_MSG_TXT, V_CONTEXT ) ;
       END ;

       -- Valeur par défaut indispensable pour V_STEP_DEB (en cas d'erreur avant
l'ouverture de la 1ère step)
       SET V_STEP_DEB = CURRENT TIMESTAMP ;

       -- Stockage du contexte applicatif
       SET V_CONTEXT = VALCONO ;

       -- incrémentation du compteur alimentant le numéro de job (indispensable pour
les logs)
       CALL PRCTRACJOB ( V_JOB_NUM ) ;

       -- suppression des blancs inutiles pour faciliter les tests ult?rieurs
       SET VALCONTEXT = TRIM(VALCONTEXT) ;

       IF (VALCONTEXT = 'N-1') THEN
          SET V_NOM_TABL = 'MF_PRIX_PREC' ;
       ELSE
          SET V_NOM_TABL = 'MF_PRIX_COUR' ;
       END IF ;
```

```
        -- Traitement des "règles métiers" - Début

-------------------------------------------------
--Debut Chargement
-------------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
-- création de l'enveloppe de la table temporaire
declare global temporary table SAX_XFRPRF as (
SELECT
  X.ODCONO,
  X.ODDIVI,
  X.ODITNO,
  X.ODCUCD,
  X.ODPRRF,
  X.ODSAPR,
  X.ODFVDT,
  X.ODLVDT
FROM XFRPRF X
FETCH FIRST 1 ROW ONLY
) DEFINITION ONLY WITH REPLACE  ;

-- insertion des lignes utiles pour le traitement pour l'année N-1
IF (VALCONTEXT = 'N-1') THEN
       INSERT INTO QTEMP/SAX_XFRPRF (
       SELECT
         X.ODCONO,
         X.ODDIVI,
         X.ODITNO,
         X.ODCUCD,
         X.ODPRRF,
         X.ODSAPR,
         X.ODFVDT,
         X.ODLVDT
       FROM XFRPRF X
       WHERE X.ODCONO = VALCONO
         AND X.ODCUCD = 'EUR'
      AND X.ODPRRF = 'PR'
      AND X.ODFVDT = (
             select max(Y.ODFVDT)
             from XFRPRF Y
             where   X.ODCONO = Y.ODCONO
                 AND X.ODDIVI = Y.ODDIVI
                 AND X.ODITNO = Y.ODITNO
                    AND X.ODCUCD = Y.ODCUCD
                    AND X.ODPRRF = Y.ODPRRF
                    AND Y.ODFVDT <= year(current_date - 1 year)*10000 +
month(current_date - 1 year)*100 + day(current_date - 1 year)
      )
```

```
      ) ;
ELSE
    -- insertion des lignes utiles pour le traitement pour l'année en cours
      INSERT INTO QTEMP/SAX_XFRPRF (
      SELECT
        X.ODCONO,
        X.ODDIVI,
        X.ODITNO,
        X.ODCUCD,
        X.ODPRRF,
        X.ODSAPR,
        X.ODFVDT,
        X.ODLVDT
      FROM XFRPRF X
      WHERE X.ODCONO = VALCONO
        AND X.ODCUCD = 'EUR'
      AND X.ODPRRF = 'PR'
      AND X.ODFVDT = (
            select max(Y.ODFVDT)
            from XFRPRF Y
            where   X.ODCONO = Y.ODCONO
                AND X.ODDIVI = Y.ODDIVI
                AND X.ODITNO = Y.ODITNO
                   AND X.ODCUCD = Y.ODCUCD
                   AND X.ODPRRF = Y.ODPRRF
                   AND Y.ODFVDT <= year(current_date)*10000 + month(current_date)*100
+ day(current_date)
      )
       ) ;
END IF ;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
CREATE INDEX QTEMP/SAX_XFRPRF_L1 ON QTEMP/SAX_XFRPRF (ODCONO, ODITNO );
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------
```

```
-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
SET V_DYNSQL1 = 'INSERT INTO ' concat trim(V_NOM_TABL) concat ' (ODCONO, ODDIVI,
ODITNO, ODSTYN, ODHDPR, ODORTA, ODCUCD, ODPRRF, ODSAPR, ODFVDT, ODLVDT, ODFVDT_DAT,
ODLVDT_DAT, ODITNO_8) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';

PREPARE V_DYNSTM1 FROM V_DYNSQL1;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
SET V_DYNSQL2 = 'SELECT X.ODDIVI, X.ODITNO, X.ODCUCD, X.ODPRRF, X.ODSAPR, X.ODFVDT,
X.ODLVDT
 FROM QTEMP/SAX_XFRPRF X WHERE X.ODCONO = ? AND X.ODITNO = ? FETCH FIRST 1 ROW ONLY';

PREPARE V_DYNSTM2 FROM V_DYNSQL2;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
SET V_DYNSQL_CUR = 'SELECT A.MMITNO, A.HMSTYN, A.MMHDPR,
  B1.ODDIVI AS XXDIVI, B1.ODITNO AS XXITNO, B1.ODCUCD AS XXCUCD,
  B1.ODPRRF AS XXPRRF, B1.ODSAPR AS XXSAPR, B1.ODFVDT AS XXFVDT,
  B1.ODLVDT AS XXLVDT
FROM (
       SELECT S.MMCONO, S.MMITNO, H.HMSTYN, S.MMHDPR
       FROM MITMAS S
       LEFT OUTER JOIN MITMAH H
         ON S.MMCONO = H.HMCONO
```

```
        AND S.MMITNO = H.HMITNO
      WHERE S.MMCONO = ?
) A
LEFT OUTER JOIN QTEMP/SAX_XFRPRF B1
      ON B1.ODCONO = A.MMCONO
      AND B1.ODITNO = A.MMITNO'
;

PREPARE V_DYNSTM_CUR FROM V_DYNSQL_CUR;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
SET V_DYNSQL3 = 'Delete from ' concat V_NOM_TABL ;
EXECUTE IMMEDIATE V_DYNSQL3 ;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

OPEN CUR1 USING VALCONO ;

FETCH CUR1 INTO ZXITNO, ZXSTYN, ZXHDPR,
   FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT, FXLVDT ;

WHILE ( SQLSTATE = '00000' ) DO

    SET V_ORI_TAR = '1' ;

      -- SI TARIF NON TROUVE SUR MMITNO, ALORS RECHERCHE TARIF SUR MITMAH.HMSTYN
      IF (FXSAPR IS NULL AND ZXSTYN IS NOT NULL AND ZXSTYN <> '') THEN
     SET V_STEP_NUM = 8 ;
     SET V_ORI_TAR = '2' ;
```

```
      -- EXECUTE V_DYNSTM2 INTO FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT USING
VALCONO, ZXSTYN;
      OPEN CTAR USING VALCONO, ZXSTYN;
      FETCH CTAR INTO FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT, FXLVDT;
      CLOSE CTAR;
    END IF ;

    -- SI TARIF NON TROUVE SUR MITMAH.HMSTYN, ALORS RECHERCHE TARIF SUR MITMAS.MMHDPR
    IF (FXSAPR IS NULL AND ZXHDPR IS NOT NULL AND ZXHDPR <> '') THEN
      SET V_STEP_NUM = 9 ;
      SET V_ORI_TAR = '3' ;
      -- EXECUTE V_DYNSTM2 INTO FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT USING
VALCONO, ZXHDPR;
      OPEN CTAR USING VALCONO, ZXHDPR;
      FETCH CTAR INTO FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT, FXLVDT;
      CLOSE CTAR;
    END IF ;

    -- Ecriture dans MF_PRIX_COUR avec le code ITNO/STYN/HDPR trouvé en amont
       IF (FXSAPR IS NOT NULL) THEN
         if (FXFVDT IS NOT NULL) THEN
            SET FXFVDT_DAT = CVT_NUM_2_DATE(FXFVDT);
         ELSE
            SET FXFVDT_DAT = NULL ;
         END IF;
         if (FXLVDT IS NOT NULL) THEN
            SET FXLVDT_DAT = CVT_NUM_2_DATE(FXLVDT);
         ELSE
            SET FXLVDT_DAT = NULL ;
         END IF;
      SET V_STEP_NUM = 10 ;
      SET FXITNO_8 = substring(ZXITNO, 1, 8) ;
      EXECUTE V_DYNSTM1 USING VALCONO, FXDIVI, ZXITNO, ZXSTYN, ZXHDPR, V_ORI_TAR,
FXCUCD, FXPRRF, FXSAPR, FXFVDT, FXLVDT, FXFVDT_DAT, FXLVDT_DAT, FXITNO_8;
      SET V_NBR_INS = V_NBR_INS + 1 ;
       END IF ;

       -- Retour au numéro de step 7 pour faciliter le suivi en cas d'anomalie
       SET V_STEP_NUM = 7 ;
    FETCH CUR1 INTO ZXITNO, ZXSTYN, ZXHDPR,
       FXDIVI, FXITNO, FXCUCD, FXPRRF, FXSAPR, FXFVDT, FXLVDT ;

END WHILE;

CLOSE CUR1;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
-- GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
-- nombre de lignes traitées remplacées exceptionnellement par le nombre de lignes
insérées (variable V_NBR_INS)
```

```
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_INS,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------


IF (VALCONTEXT <> 'N-1') THEN

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------
delete from MF_PRIX_COUR_TAR ;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
-- GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
-- nombre de lignes traitées remplacées exceptionnellement par le nombre de lignes
insérées (variable V_NBR_INS)
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_INS,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

insert into MF_PRIX_COUR_TAR (ODCONO, ODITNO_8, ODSAPR_MIN, ODSAPR_MAX, ODSAPR_AVG)
select x.ODCONO, x.ODITNO_8, min(x.ODSAPR) as odsapr_min, max(x.ODSAPR) as
odsapr_max, avg(x.ODSAPR) as ODSAPR_AVG
from MF_PRIX_COUR x
where x.ODSTYN IS NOT NULL
group by x.ODCONO, x.ODITNO_8
;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
-- GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
-- nombre de lignes traitées remplacées exceptionnellement par le nombre de lignes
insérées (variable V_NBR_INS)
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_INS,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

END IF ;
```

```
        -- Traitement des "règles métiers" - Fin

END  ;
COMMENT ON SPECIFIC PROCEDURE MY_LIBRARY/MY_PROCEDURE
     IS 'Chargement table MF_PRIX_COUR, MF_PRIX_PREC et MF_PRIX_COUR_TAR (selon
VALCONTEXT)' ;
```

3.10 Références croisées

On peut utiliser la technique du DSPPGMREF sur les procédures stockées, mais on n'obtiendra que les noms courts des tables et autres objets utilisés par la procédure, ce qui peut être un problème si les noms longs sont très utilisés.

On peut obtenir ces même références croisées via les tables systèmes DB2, cette fois avec les noms longs des objets dépendants, au moyen de la requête suivante :

```
SELECT distinct OBJECT_NAME, OBJECT_SCHEMA, OBJECT_TYPE
FROM QSYS2.SYSROUTINEDEP
WHERE SPECIFIC_SCHEMA = 'MY_LIBRARY' AND SPECIFIC_NAME = 'MY_PROCEDURE'
ORDER BY OBJECT_TYPE DESC, OBJECT_NAME;
```

Et à l'inverse, on peut obtenir la liste des procédures stockées utilisant une table au moyen de la requête suivante :

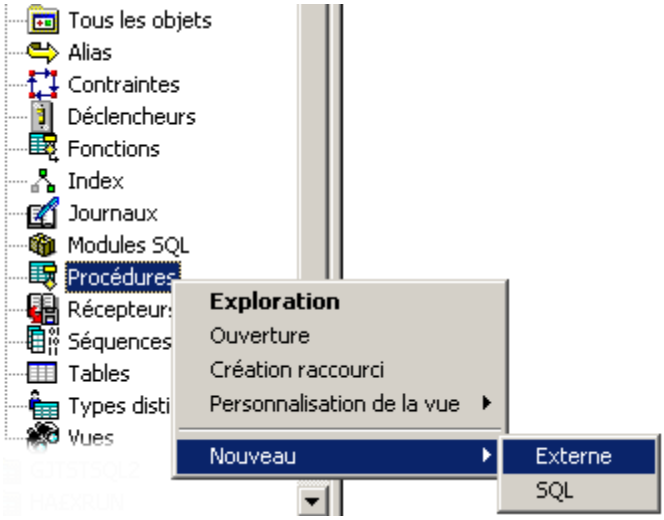
```
SELECT distinct SPECIFIC_SCHEMA, SPECIFIC_NAME, OBJECT_SCHEMA
FROM  QSYS2.SYSROUTINEDEP
WHERE OBJECT_NAME = 'MY_TABLE'
ORDER BY SPECIFIC_SCHEMA, SPECIFIC_NAME ;
```

On notera dans l'exemple ci-dessus que l'on a pris soin de ne pas préciser dans le WHERE de sélection sur la colonne OBJECT_SCHEMA, car si à l'intérieur des procédures stockées on n'a pas déclaré de noms de bibliothèques en face des objets DB2 utilisés, cette colonne contiendra des NULL au niveau de la table SYSROUTINDEP.

3.11 Procédure stockée DB2 « externe »

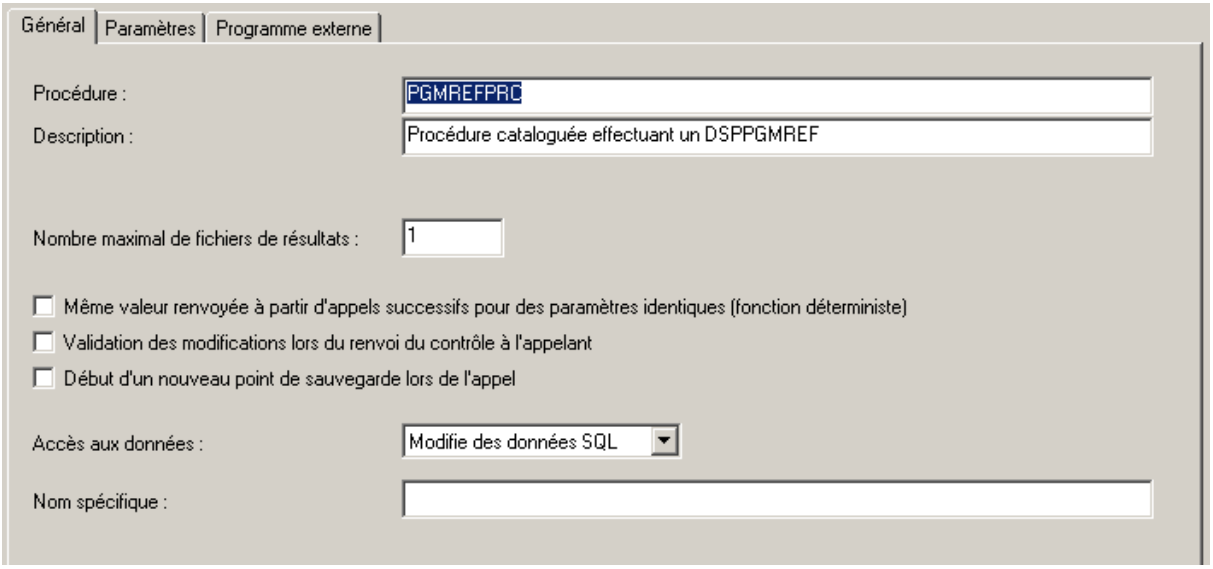
Le logiciel System i Navigator d'IBM offre de très bons assistants permettant de faciliter la création de procédures stockées. On peut y choisir de créer une procédure de type "SQL" (on s'appuie intégralement sur le langage normalisé PL/SQL qui est intégré à DB2), ou une procédure de type "externe" (on s'appuie alors sur un programme développé dans un langage autre que SQL, tel que le RPGLE, le C, le Cobol, etc…).

Création d'une procédure stockée externe sous IBM i Navigator :

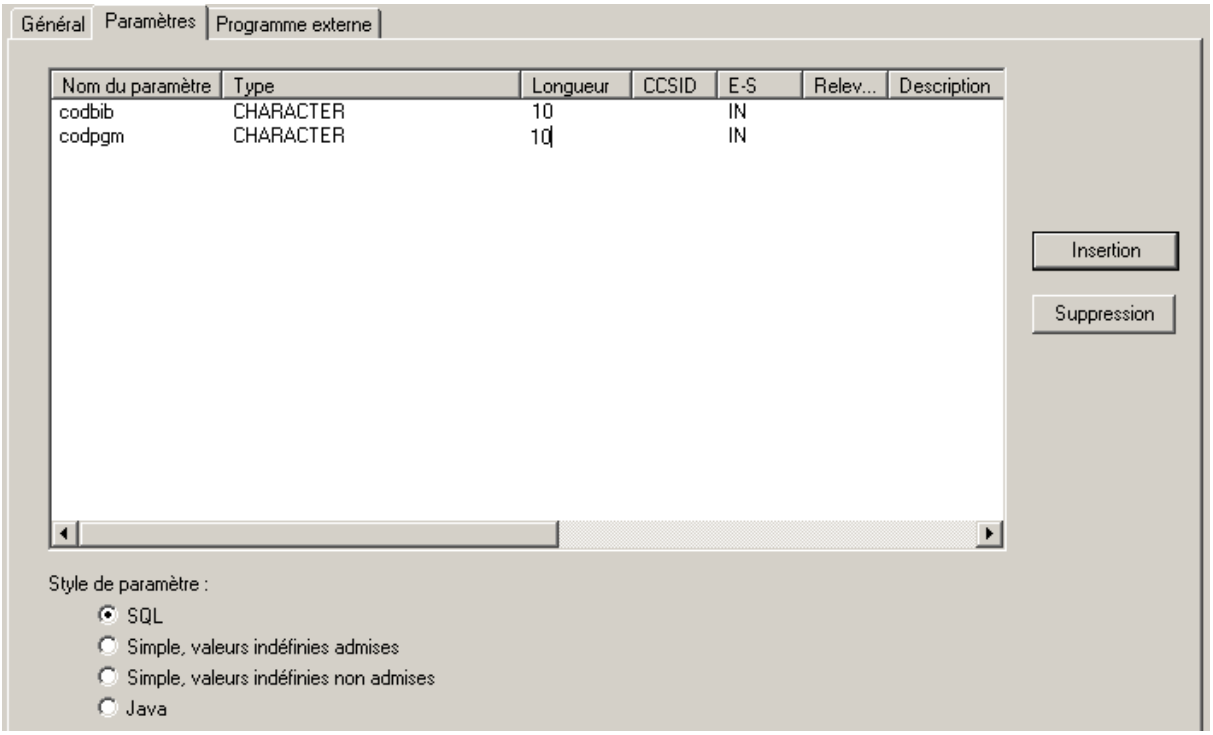


Dans les copies d'écran suivantes, on crée une procédure stockée de type "externe", s'appuyant sur le programme SQLRPGLE PGMREFPROC.

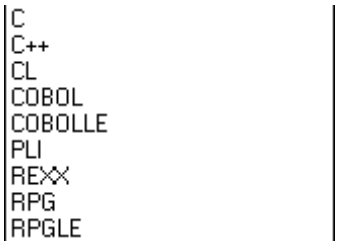
Premier onglet : la procédure stockée va renvoyer 1 jeu de données (result set), ce qui explique qu'on ait saisi "1" dans le nombre maximal de fichiers de résultats. Les autres options sont laissées avec leurs valeurs par défaut.



Deuxième onglet : définition des paramètres d'entrée/sortie du programme SQLRPGLE utilisé par la procédure stockée.



Troisième onglet : on y précise simplement que le programme encapsulé sera de type « RPGLE ». A titre d'information, la liste des types de programmes pouvant être encapsulés dans une procédure stockée externe est la suivante :



La validation du 3ème onglet a pour effet de générer automatiquement la procédure stockée sur le IBM i. Attention : l'objet créé est un objet de type DB2, et non pas un objet de type OS/400, donc ne vous étonnez pas de ne pas le voir par un simple WRKOBJ. Si on souhaite le supprimer, on peut le faire en mode SQL avec la requête suivante : DROP PROCEDURE PGMREFPRC

On peut également générer le source SQL de création de la procédure stockée, par exemple pour pouvoir le réexécuter sur un autre IBM i. Pour faire cela sous System i Navigator, il suffit de faire un clic-droit sur la procédure, et de sélectionner l'option "Génération d'instructions SQL…". On peut choisir de générer le source dans un fichier texte (PC) ou dans un membre de fichier source IBM i.

Une fois généré, le source peut être retravaillé sur l'IBM i par exemple via PDM, il peut également être transféré sur une autre machine, il peut également être exécuté via la commande OS/400 RUNSQLSTM, ce qui aura pour effet de recréer la procédure (attention, si elle existe déjà, il faut la supprimer au préalable via un "DROP PROCEDURE", ou mieux, utiliser l'instruction "CREATE OR REPLACE maprocedure").

Il est important de souligner que l'on peut tester sous IBM i Navigator le bon fonctionnement de la procédure stockée. Par exemple, en passant par l'option « Exécution de scripts SQL » de System i Navigator, je peux saisir la requête suivante (en supposant que la procédure se trouve dans la bibliothèque MABIB) :

```
CALL MABIB.PGMREFPRC ('MABIB', 'MONPGM')
```

Si la procédure stockée fonctionne bien, une fenêtre doit s'afficher dans IBM i Navigator, renvoyant le contenu du « Result Set », soit les références croisées du programme « MONPGM » de la bibliothèque « MABIB ».

Voici le code source final de la procédure stockée PGMREFPRC :

```
--  Procédure stockée générée via IBM i  Navigator,
--    dont le principe est d'effectuer un DSPPGMREF d'un programme
--    reçu en paramètre, et de renvoyer un RESULT SET correspondant
--    au contenu du fichier généré par le DSPPGMREF
--
--  Procédure créée avec la commande suivante :
--     RUNSQLSTM SRCFILE(MABIB/QSQLSRC)
--       SRCMBR(PGMREFPRC) COMMIT(*NONE) NAMING(*SQL)
```

```
--
--  Commande d'exécution de la procédure :
--     CALL MABIB.PGMREFPRC ('xxMABIBxx', 'xxMONPGMxx');
--
CREATE PROCEDURE MABIB.PGMREFPRC (
       IN CODBIB CHAR(10) ,
       IN CODPGM CHAR(10) )
       DYNAMIC RESULT SETS 1
       LANGUAGE RPGLE
       SPECIFIC MABIB.PGMREFPRC
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'MABIB/PGMREFPROC'
       PARAMETER STYLE SQL ;
--
COMMENT ON SPECIFIC PROCEDURE MABIB.PGMREFPRC
       IS 'Procédure encapsulant un DSPPGMREF' ;
```

A partir de maintenant, nous sommes en mesure d'invoquer cette procédure stockée, et de récupérer son « Result Set » à l'intérieur d'un script PHP. C'est exactement ce que fait le script suivant.

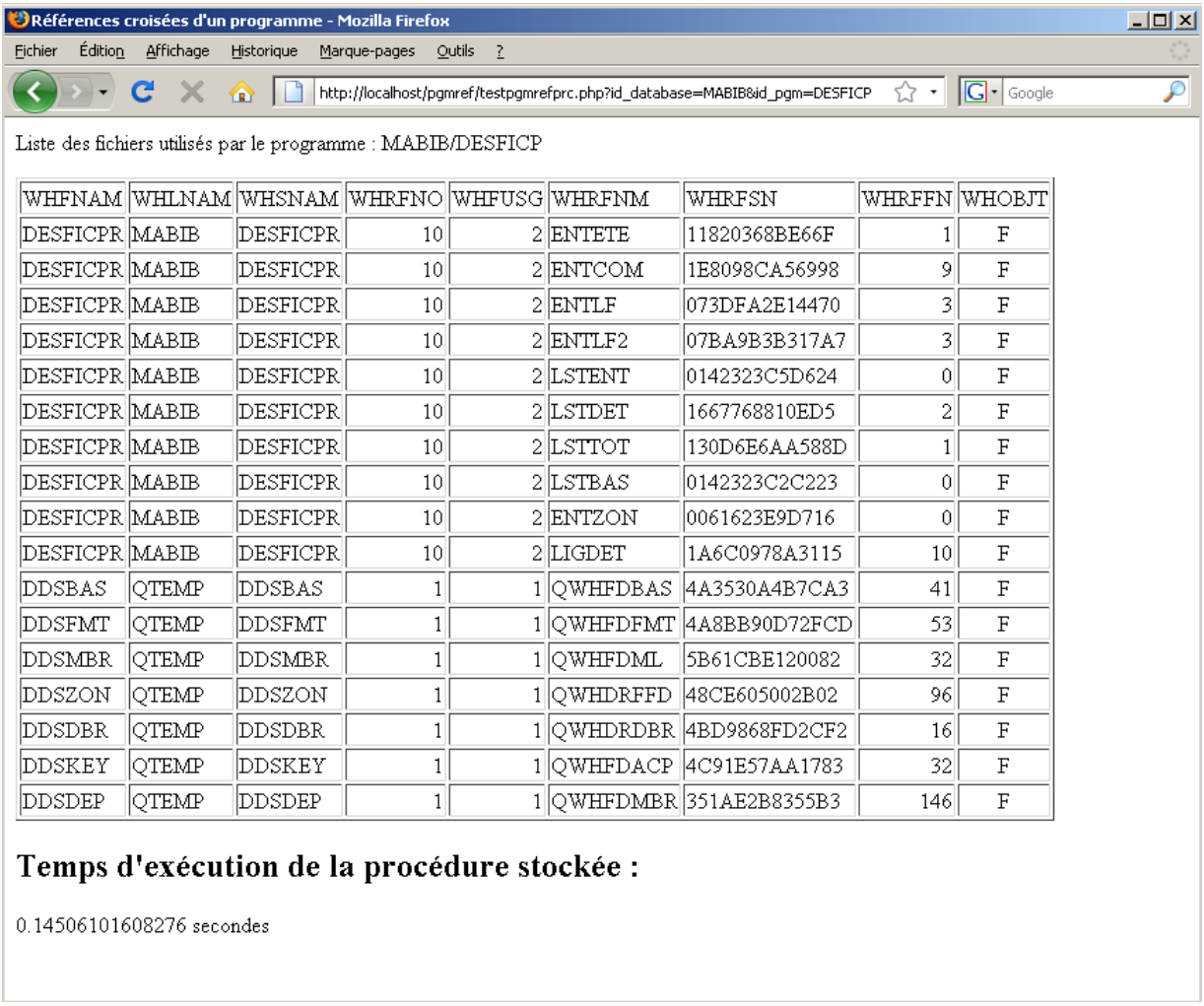
Code source du script PHP testpgmrefprc.php (utilisant PDO pour la connexion à la base de données) :

```
<html>
<head>
    <title>R&eacute;f&eacute;rences crois&eacute;es d'un programme</title>
</head>
<body>
<?php

// configuration de la connexion à la base de données
require_once 'config.php' ;

// chargement de la fonction getmicrotime
require_once 'fonctions.php' ;

// récupération des paramètres du $_GET
if (isset($_GET)) {
    $params = $_GET ;
} else {
    $params = array() ;
} ;

// paramètres d'appel de la procédure cataloguée

$codbib = trim($params['id_database']) ;
$codpgm = trim($params['id_pgm']) ;

echo 'Liste des fichiers utilis&eacute;s par le programme : ' . trim($codbib) . '/' .
trim($codpgm) . '<p/>' ;
```

```
echo <<<EOM
  <table border='1'>
  <tr>
    <td>WHFNAM</td>
    <td>WHLNAM</td>
    <td>WHSNAM</td>
    <td>WHRFNO</td>
    <td>WHFUSG</td>
    <td>WHRFNM</td>
    <td>WHRFSN</td>
    <td>WHRFFN</td>
    <td>WHOBJT</td>
  </tr>
EOM;

$time_start = getmicrotime() ;

// requête d'appel de la procédure stockée
$sql = "CALL MABIB.PGMREFPRC (?, ?)" ;
try {
    $st = $db->prepare($sql);
    $st->bindParam(1, $codbib);
    $st->bindParam(2, $codpgm);
    $st->execute() ;
    if ($st) {
        do {
            $row_data = $st->fetchAll(PDO::FETCH_ASSOC);
            if ($row_data) {
                for($yIndex = 0; $yIndex < count($row_data); $yIndex++) {
                    echo <<<EOM
                    <tr>
                    <td>{$row_data[$yIndex]['WHFNAM']}</td>
                    <td>{$row_data[$yIndex]['WHLNAM']}</td>
                    <td>{$row_data[$yIndex]['WHSNAM']}</td>
                    <td align="right">{$row_data[$yIndex]['WHRFNO']}</td>
                    <td align="right">{$row_data[$yIndex]['WHFUSG']}</td>
                    <td>{$row_data[$yIndex]['WHRFNM']}</td>
                    <td>{$row_data[$yIndex]['WHRFSN']}</td>
                    <td align="right">{$row_data[$yIndex]['WHRFFN']}</td>
                    <td align="center">{$row_data[$yIndex]['WHOBJT']}</td>
                    </tr>
EOM;
                }
            }
        } while ($st->nextRowset());
    }
} catch (PDOException $e) {
    if ($e->getCode() == 24000) {
        echo 'Aucune donn&eacute;e disponible pour le programme indiqu&eacute;<br/>'
;
    } else {
        echo 'Erreur grave autre que 24000 :<br/>' ;
```

```
        echo 'Error : ' . $e->getMessage() . '<br/>';
        echo 'Code  : ' . $e->getCode() . '<br/>';
        echo 'File  : ' . $e->getFile() . '<br/>';
        echo 'Line  : ' . $e->getLine() . '<br/>';
        echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
    }
}

echo <<<EOTABLE
  </table>
EOTABLE;

$time_stop = getmicrotime() ;
$time_ecart = $time_stop - $time_start ;

echo "<p><h2>Temps d'ex&eacute;cution de la proc&eacute;dure stock&eacute;e
:</h2></p>";
echo "{$time_ecart} secondes </p>" ;

?>
```

Le résultat obtenu dans le navigateur internet pour le programme DESFICP est le suivant :



WHFNAM	WHLNAM	WHSNAM	WHRFNO	WHFUSG	WHRFNM	WHRFSN	WHRFFN	WHOBJT
DESFICPR	MABIB	DESFICPR	10	2	ENTETE	11820368BE66F	1	F
DESFICPR	MABIB	DESFICPR	10	2	ENTCOM	1E8098CA56998	9	F
DESFICPR	MABIB	DESFICPR	10	2	ENTLF	073DFA2E14470	3	F
DESFICPR	MABIB	DESFICPR	10	2	ENTLF2	07BA9B3B317A7	3	F
DESFICPR	MABIB	DESFICPR	10	2	LSTENT	0142323C5D624	0	F
DESFICPR	MABIB	DESFICPR	10	2	LSTDET	1667768810ED5	2	F
DESFICPR	MABIB	DESFICPR	10	2	LSTTOT	130D6E6AA588D	1	F
DESFICPR	MABIB	DESFICPR	10	2	LSTBAS	0142323C2C223	0	F
DESFICPR	MABIB	DESFICPR	10	2	ENTZON	0061623E9D716	0	F
DESFICPR	MABIB	DESFICPR	10	2	LIGDET	1A6C0978A3115	10	F
DDSBAS	QTEMP	DDSBAS	1	1	QWHFDBAS	4A3530A4B7CA3	41	F
DDSFMT	QTEMP	DDSFMT	1	1	QWHFDFMT	4A8BB90D72FCD	53	F
DDSMBR	QTEMP	DDSMBR	1	1	QWHFDML	5B61CBE120082	32	F
DDSZON	QTEMP	DDSZON	1	1	QWHDRFFD	48CE605002B02	96	F
DDSDBR	QTEMP	DDSDBR	1	1	QWHDRDBR	4BD9868FD2CF2	16	F
DDSKEY	QTEMP	DDSKEY	1	1	QWHFDACP	4C91E57AA1783	32	F
DDSDEP	QTEMP	DDSDEP	1	1	QWHFDMBR	351AE2B8355B3	146	F

Points importants :

Le temps d'exécution est calculé au moyen de la fonction getmicrotime() que nous avons vue dans un précédent chapitre.

Vous noterez que pour la première fois, j'ai utilisé la méthode bindParam() de PDO pour encapsuler les paramètres passés à la requête SQL :

```
$sql = "CALL MABIB.PGMREFPRC (?, ?)" ;
...
$st = $db->prepare($sql);
$st->bindParam(1, $codbib);
$st->bindParam(2, $codpgm);
$st->execute() ;
...
```

Ce n'était pas indispensable, mais je voulais profiter de l'occasion pour vous montrer cette technique. On aurait tout aussi bien pu écrire la variante suivante :

```
$sql = "CALL MABIB.PGMREFPRC (:bib, :pgm)" ;
...
$st = $db->prepare($sql);
$st->bindParam(':bib', $codbib);
$st->bindParam(':pgm', $codpgm);
$st->execute() ;
...
```

Et bien sûr la version plus simple, sans bindParam(), qui donne également le même résultat :

```
$sql = "CALL MABIB.PGMREFPRC (?, ?)" ;
...
$st = $db->prepare($sql);
$st->execute(array($codbib, $codpgm)) ;
...
```

La boucle principale du script PHP consiste à parcourir la liste des « Result Set » renvoyés par la procédure stockée. Comme le programme RPG utilisé ici ne renvoie qu'un seul « Result Set », cette boucle ne s'exécutera qu'une seule fois.

```
do { … } while ($stmt->nextRowset());
```

Le « Result Set » récupéré par le script PHP est un tableau à deux dimensions (stocké dans la variable $row_data). Si on affiche le contenu de ce tableau avec la fonction PHP print_r(), on obtient le résultat suivant :

```
[0] => Array (
        [WHFNAM] => DESFICPR
        [WHLNAM] => MABIB
        [WHSNAM] => DESFICPR
        [WHRFNO] => 10
        [WHFUSG] => 2
        [WHRFNM] => ENTETE
        [WHRFSN] => 11820368BE66F
        [WHRFFN] => 1
        [WHOBJT] => F )
[1] => Array (
        [WHFNAM] => DESFICPR
        [WHLNAM] => MABIB
        [WHSNAM] => DESFICPR
        [WHRFNO] => 10
        [WHFUSG] => 2
        [WHRFNM] => ENTCOM
        [WHRFSN] => 1E8098CA56998
        [WHRFFN] => 9
        [WHOBJT] => F )
etc….
```

On pourrait donc parcourir le tableau au moyen de 2 boucles imbriquées, la première boucle parcourant les lignes (0, 1, etc…), la seconde parcourant les colonnes. Comme les colonnes sont nommées explicitement (et non pas seulement numérotées), et que je souhaite contrôler l'ordre d'affichage des colonnes, j'ai opté pour une approche plus « rigide », en utilisant une syntaxe du type :

```
row_data[<numéro de ligne>][<nom de colonne>]
```

A noter que c'est la fonction fetchAll(), associée au paramètre PDO::FETCH_ASSOC, qui permet de récupérer les noms des colonnes renvoyées par le « Result Set ».

On aboutit donc à la boucle ci-dessous, qui parcourt les lignes du « Result Set », et nous permet d'afficher le tableau présenté page précédente :

```
for($yIndex = 0; $yIndex < count($row_data); $yIndex++) {
	echo <<<EOM
		<tr>
		<td>{$row_data[$yIndex]['WHFNAM']}</td>
		<td>{$row_data[$yIndex]['WHLNAM']}</td>
		<td>{$row_data[$yIndex]['WHSNAM']}</td>
		<td align="right">{$row_data[$yIndex]['WHRFNO']}</td>
		<td align="right">{$row_data[$yIndex]['WHFUSG']}</td>
		<td>{$row_data[$yIndex]['WHRFNM']}</td>
		<td>{$row_data[$yIndex]['WHRFSN']}</td>
		<td align="right">{$row_data[$yIndex]['WHRFFN']}</td>
		<td align="center">{$row_data[$yIndex]['WHOBJT']}</td>
		</tr>
EOM;
}
```

4. Compléments

4.1 Faire pivoter les données

Nous allons étudier dans ce chapitre des techniques d'inversion de données, permettant de mettre des lignes en colonnes, ou des colonnes en lignes.

Ils sont inspirés d'exemples tirés - et adaptés à DB2 - du livre suivant :

"SQL Hacks", d'Andrew Cumming et Gordon Russell, éd. O'Reilly

Les techniques proposées dans ce très bon livre sont transposées dans la syntaxe spécifique à plusieurs bases de données (sauf DB2), mais la plupart des techniques présentées dans la syntaxe MySQL peuvent être adaptées assez facilement à DB2.

Les références bibliographiques sont rappelées en fin de documentation.

4.1.1 Lignes en colonnes

Technique permettant de mettre des lignes en colonnes

Pour commencer, créons une table des notes.

```
create table notes
(ETUDIANT CHAR (20 )    NOT NULL WITH DEFAULT,
 COURS    CHAR (20 )    NOT NULL WITH DEFAULT,
 NOTE     INTEGER       NOT NULL WITH DEFAULT);

insert into notes (etudiant, cours, note) values
('Gao Gong', 'Java   ', 80),
('Gao Gong', 'BD     ', 77),
('Gao Gong', 'Algèbre', 50),
('Zang Yi ', 'Java   ', 62),
('Zang Yi ', 'BD     ', 95),
('Zang Yi ', 'Algèbre', 63);

select * from notes ;
```

ETUDIANT	COURS	NOTE
Gao Gong	Java	80
Gao Gong	BD	77
Gao Gong	Algèbre	50
Zang Yi	Java	62
Zang Yi	BD	95
Zang Yi	Algèbre	63

Pour la suite de l'exercice, on a besoin d'une table contenant la liste des étudiants. Comme on n'en a pas, on va en créer une artificiellement par l'intermédiaire d'une vue sur la table « notes » :

```
CREATE VIEW ETUDIANTS AS SELECT DISTINCT ETUDIANT FROM NOTES ;

select * from etudiants ;
```

ETUDIANT
Gao Gong
Zang Yi

1ère solution : utiliser 3 jointures externes entre la vue « etudiants » et la table « notes » :

```
select a.etudiant, jav.note as java, bas.note as bd, alg.note as algebre
  from etudiants a
  left outer join notes jav
     on (a.etudiant = jav.etudiant and jav.cours = 'Java')
  left outer join notes bas
```

```
   on (a.etudiant = bas.etudiant and bas.cours = 'BD')
left outer join notes alg
   on (a.etudiant = alg.etudiant and alg.cours = 'Algèbre');
```

```
ETUDIANT                       JAVA          BD          ALGEBRE
Gao Gong                         80          77               50
Zang Yi                          62          95               63
```

```
Rappel : on peut utiliser une CTE en remplacement de la vue "etudiants"
```

```
with tmpetudiants as (
SELECT DISTINCT ETUDIANT FROM qtemp.NOTES
)
select a.etudiant, jav.note as java, bd.note, alg.note
  from tmpetudiants a
  left outer join qtemp.notes jav
     on (a.etudiant = jav.etudiant and jav.cours = 'Java')
  left outer join qtemp.notes bd
     on (a.etudiant = bd.etudiant and bd.cours = 'BD')
  left outer join qtemp.notes alg
     on (a.etudiant = alg.etudiant and alg.cours = 'Algèbre')
;
```

2ème solution : utiliser une seule jointure avec l'instruction CASE, et un GROUP BY sur le nom de l'étudiant.

```
select a.etudiant,
  max(case when b.cours = 'Java'    then b.note else 0 end) as Java,
  max(case when b.cours = 'BD'      then b.note else 0 end) as BD,
  max(case when b.cours = 'Algèbre' then b.note else 0 end) as Algebre
from etudiants a
left outer join notes b  on (a.etudiant = b.etudiant)
group by a.etudiant  ;
```

```
ETUDIANT                       JAVA          BD          ALGEBRE
Gao Gong                         80          77               50
Zang Yi                          62          95               63
```

N.B. : sans la clause GROUP BY, on n'aurait pas pu utiliser la clause MAX sur les colonnes calculées « Java », « BD » et « Algebre ». On aurait donc eu une requête telle que celle-ci-dessous :

```
select a.etudiant,
  case when b.cours = 'Java'    then b.note else 0 end  as Java,
  case when b.cours = 'BD'      then b.note else 0 end  as BD,
  case when b.cours = 'Algèbre' then b.note else 0 end  as Algebre
from etudiants a
left outer join notes b  on (a.etudiant = b.etudiant);
```

qui aurait abouti au résultat suivant :

```
ETUDIANT                       JAVA          BD          ALGEBRE
```

Gao Gong	80	0	0
Gao Gong	0	77	0
Gao Gong	0	0	50
Zang Yi	62	0	0
Zang Yi	0	95	0
Zang Yi	0	0	63

4.1.2 Colonnes en lignes

Technique permettant de mettre des colonnes en lignes

Pour aller plus vite, nous allons repartir du chapitre précédent, en créant une table “etudnotes” qui recevra le résultat de la requête créée précédemment.

```
create table etudnotes
 (ETUDIANT CHAR (20 )    NOT NULL WITH DEFAULT,
  JAVA     INTEGER       NOT NULL WITH DEFAULT,
  BD       INTEGER       NOT NULL WITH DEFAULT,
  ALGEBRE  INTEGER       NOT NULL WITH DEFAULT);

insert into etudnotes
  select a.etudiant,
    max(case when b.cours = 'Java'    then b.note else 0 end) as Java,
    max(case when b.cours = 'BD'      then b.note else 0 end) as BD,
    max(case when b.cours = 'Algèbre' then b.note else 0 end) as Algebre
  from etudiants a
  left outer join notes b  on (a.etudiant = b.etudiant)
  group by a.etudiant ;
```

Nous obtenons donc la table suivante avec une présentation en colonnes :

ETUDIANT	JAVA	BD	ALGEBRE
Gao Gong	80	77	50
Zang Yi	62	95	63

… à partir de laquelle nous pouvons afficher les colonnes en ligne au moyen de la requête suivante :

```
select etudiant, 'Java' as matiere, Java as note from etudnotes
union
select etudiant, 'BD' as matiere, BD from etudnotes
union
select etudiant, 'Algèbre' as matiere, Algebre from etudnotes ;
```

ETUDIANT	MATIERE	NOTE
Gao Gong	Java	80
Zang Yi	Java	62
Gao Gong	BD	77
Zang Yi	BD	95
Gao Gong	Algèbre	50
Zang Yi	Algèbre	63

Pour obtenir un tri par étudiant et matière :

```
select x.* from (
select etudiant, 'Java' as matiere, Java as note from etudnotes
```

```
union
select etudiant, 'BD' as matiere, BD from etudnotes
union
select etudiant, 'Algèbre' as matiere, Algebre from etudnotes
) x order by x.etudiant, x.matiere
;
```

4.2 QCMDEXC

Depuis la V5R4, vous pouvez lancer des commandes systèmes IBM i en utilisant DB2 (et donc PHP).

Par exemple, si vous souhaitez récupérer dans une table DB2 la liste des profils utilisateurs IBM i, vous pouvez procéder ainsi :

```
function cmdsys_rtvusrprf_all () {
   $cmd = 'DSPUSRPRF USRPRF(*ALL) OUTPUT(*OUTFILE)
   OUTFILE(QTEMP/TMPPROFILE) OUTMBR(*FIRST *REPLACE)';
   $cmd_length = strlen($cmd);
   return "CALL QCMDEXC ('{$cmd}', {$cmd_length})" ;
}
$cmd_sys = cmdsys_rtvusrprf_all();
echo $cmd_sys , '<br />' . PHP_EOL;
$result = db2_exec($conn, $cmd_sys);
if ($result) {
   print "La table a été créée correctement.<br />" . PHP_EOL;
} else {
   print "La table n'a pas été créée.<br />" . PHP_EOL;
}
```

ATTENTION : avec l'arrivée de la TR7 (Technology Refresh) qui s'applique à la V7R1, il n'est plus nécessaire de transmettre la longueur de la chaîne à l'API QCMDEXC.

- Cette manière d'exécuter des commandes systèmes IBM i fonctionne aussi bien avec db2_connect() qu'avec PDO, et elle ne nécessite l'emploi d'aucune boîte à outils complémentaires
- Autre exemple d'utilisation possible, vous voulez détecter des déphasages entre des environnements de recette, de préproduction et de production… simplement en récupérant la liste de objets IBMi et/ou DB2 des bibliothèques de chaque environnement, puis en les comparant via un bon vieil algorithme de « matching », comme dans l'exemple ci-dessous :

La méthode utilisée pour extraire les objets IBM i d'une bibliothèque est la suivante:

```
function extract_ibmi_objects_from_lib($library, $tmp_table) {
      $tmp_table = strtoupper(trim($tmp_table)) ;
      $library = strtoupper(trim($library));
      $cmd = 'DSPOBJD OBJ('.$library.'/*ALL) OBJTYPE(*ALL) DETAIL(*BASIC)
OUTPUT(*OUTFILE)  OUTFILE(QTEMP/'.$tmp_table.') OUTMBR(*FIRST *REPLACE)' ;
      $cmd_length = strlen($cmd);
      return "CALL QCMDEXC ('{$cmd}', {$cmd_length})" ;
}
```

On peut dès lors exécuter cette commande via DB2 sur 2 bibliothèques différentes d'un même serveur IBM i, ou encore sur 2 bibliothèques de 2 serveurs IBM i différents. Il ne reste plus qu'à comparer les 2 jeux de données en les parcourant séquentiellement et parallèlement, pour

détecter les écarts (technique de « matching »).

ATTENTION : la technique présentée ici pour l'exécution de commandes système est très pratique mais elle présente un inconvénient qui est que - en cas de problème - DB2 récupère un message d'erreur générique, mais n'est pas en mesure de connaître la cause de l'échec.

Pour un meilleur contrôle des opérations, il est préférable d'exécuter les commandes dans un CL, puis d'encapsuler ce CL dans une procédure stockée DB2, qui sera en mesure de récupérer et de retransmettre un code retour fourni par le CL, si une anomalie est détectée.

4.3 Classement

Attention, la comparaison du contenu de 2 bibliothèques par la technique du « matching » (présentée au chapitre précédent) réserve quelques surprises si on n'y prend pas garde, du fait de la manière dont PHP et DB2 pour IBM i trient alphabétiquement les chaînes de caractères :

Une différence aussi minime rend inopérante la technique du « matching » par comparaison alphabétique, sauf si vous laissez à DB2 l'entière responsabilité de déterminer si les données sont bien triées ou pas. Donc, plutôt que de comparer les chaînes de caractères via PHP, créez une fonction PHP qui recevra les chaînes de caractères 1 et 2 à comparer. A l'intérieur de cette fonction, tronquez les 2 chaînes à comparer à la longueur de la plus courte des deux, et exécutez le code SQL ci-dessous via votre « wrapper » préféré :

```
$sql = <<<BLOC_SQL
 SELECT CASE WHEN '{$chaine1}' > '{$chaine2}' THEN 1 ELSE
          CASE WHEN '{$chaine1}' < '{$chaine2}' THEN -1 ELSE 0
END
        END as RESULT
 FROM SYSIBM.SYSDUMMY1
BLOC_SQL;
```

La technique ci-dessus peut aisément s'encapsuler dans une UDF (User Defined Function).

4.4 Migration

Certains frameworks (PHP ou autres) proposent une fonction de migration de base de données, permettant de déployer de nouvelles tables dans une base de données, ou encore de procéder à des altérations de table (pour l'ajout ou la suppression de colonnes, ou encore pour redimensionner des colonnes).

L'approche proposée par ces frameworks consiste généralement à écrire des classes de "migration".
Dans cette approche, une migration consiste à installer une nouvelle table, ou à appliquer des modifications sur une table existante.
Une migration peut aussi intervenir sur plusieurs tables, mais il peut être plus facile de superviser des migrations intervenant sur un très petit nombre de tables (quitte à écrire plusieurs classes de migration pour le déploiement d'une nouvelle version de logiciel).

Une migration consisterait donc à gérer :
- un numéro de migration (incrément) associé à chaque nouvelle migration
- les actions SQL à exécuter dans le cas de l'installation d'une migration
- les actions SQL à exécuter dans le cas d'une désinstallation d'une migration (si on souhaite faire machine arriière après avoir constaté une anomalie consécutive à une migration installée

On peut s'inspirer de cette approche pour écrire des procédures stockées dédiées à la migration.
Exemple ci-dessous de procédure stockée (expérimentale) pouvant être envisagée dans cette optique.
Dans cet exemple, la procédure peut être appelée avec 2 valeurs qui sont "UP" pour une montée de version et "DOWN" dans le cas d'un retour en arrière (cette version non finalisée ne gère pas de notion de numéro de migration) :

```
CREATE OR REPLACE PROCEDURE M3DHSADPGM/UPG1110001 (
     IN VSENS VARCHAR(4) )
     LANGUAGE SQL
     SPECIFIC M3DHSADPGM/UPG1110001
     NOT DETERMINISTIC
     MODIFIES SQL DATA
     CALLED ON NULL INPUT
     SET OPTION ALWBLK = *ALLREAD ,
     TGTRLS = V5R4M0 ,
     ALWCPYDTA = *OPTIMIZE ,
     COMMIT = *NONE ,
     CLOSQLCSR = *ENDMOD ,
     DATFMT = *ISO ,
     TIMFMT = *ISO ,
```

```
     DECMPT = *JOB ,
     DECRESULT = (31, 31, 00) ,
     DFTRDBCOL = *NONE ,
     DYNDFTCOL = *NO ,
     DYNUSRPRF = *USER ,
     SRTSEQ = *HEX ,
     OUTPUT = *PRINT ,
     DBGVIEW = *NONE
     BEGIN
       -- Déclaration du nom de la procédure courante (pour gestion des logs)
       DECLARE V_NOM_PROC CHAR(20) DEFAULT 'UPG1110001';
       -- Déclaration du contexte applicatif (pour gestion des logs)
       DECLARE V_CONTEXT VARCHAR(30) DEFAULT '';
       -- Déclaration de la variable définissant l'étape courante (pour log)
       DECLARE V_STEP_NUM INTEGER DEFAULT 0;
       DECLARE V_STEP_DEB TIMESTAMP ;
       -- Déclaration de la variable servant à stocker le numéro de job courant
       DECLARE V_JOB_NUM INTEGER DEFAULT 0;
       -- Déclaration nombre d'enregistrements retournés par GET DIAGNOSTICS
(ROWCOUNT)
       DECLARE V_NBR_ENR INTEGER DEFAULT 0;
       -- Déclaration des "SQL return codes"
       DECLARE SQLCODE INTEGER DEFAULT 0;
       DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
        -- Déclaration des variables destinées à alimenter la log en cas d'erreurs
SQL
       DECLARE V_SQL_STATE CHAR(5) DEFAULT '00000';
       DECLARE V_SQL_CODE INT DEFAULT 0;
       DECLARE V_MSG_TXT VARCHAR(100) DEFAULT '';
       DECLARE V_MSG_TXTLEN INTEGER DEFAULT 0;
       -- Déclaration de variables susceptibles d'être utilisées sur des requêtes
dynamiques
       DECLARE SQL_STMT VARCHAR(2048) DEFAULT '';
       DECLARE V_PARAM1 CHAR(10) DEFAULT '';
       DECLARE V_PARAM2 CHAR(10) DEFAULT '';
       DECLARE V_PARAM3 CHAR(10) DEFAULT '';
       DECLARE V_PARNU1 INTEGER DEFAULT 0;
       DECLARE V_PARNU2 INTEGER DEFAULT 0;
       DECLARE V_PARNU3 INTEGER DEFAULT 0;
       -- Déclaration de jeux de 2 jeux de date prévus pour divers usages (chaque
date dans 2 formats)
       -- Date de valeur
       DECLARE V_DATVAL DATE ;
       DECLARE V_DATVAL8 DEC(8, 0) DEFAULT 0 ;
       -- Date de période (utile pour traitements hebdo et mensuels notamment)
       DECLARE V_DATPER DATE ;
       DECLARE V_DATPER8 DEC(8, 0) DEFAULT 0 ;

       -- Pavé de gestion des Données non trouvées
       DECLARE CONTINUE HANDLER FOR NOT FOUND
       BEGIN
         SET V_SQL_STATE = SQLSTATE;
         SET V_SQL_CODE  = SQLCODE;
```

```
          CALL PRCTRACLOG
            (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
SQLSTATE, SQLCODE, 'NOTFOUND', '', V_CONTEXT ) ;
        END ;

         -- Pavé de gestion des Avertissements
        DECLARE CONTINUE HANDLER FOR SQLWARNING
        BEGIN
          SET V_SQL_STATE = SQLSTATE;
          SET V_SQL_CODE  = SQLCODE;
          CALL PRCTRACLOG
            (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
SQLSTATE, SQLCODE, 'WARNING', '', V_CONTEXT ) ;
        END ;

        -- Pavé de gestion des Erreurs
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
          SET V_SQL_STATE = SQLSTATE;
          SET V_SQL_CODE  = SQLCODE;
          GET DIAGNOSTICS EXCEPTION 1
             V_MSG_TXT    = MESSAGE_TEXT,
             V_MSG_TXTLEN = MESSAGE_LENGTH;
             IF V_MSG_TXTLEN > 100 THEN
                 SET V_MSG_TXT = SUBSTR(V_MSG_TXT, 1, 100) ;
             END IF ;
          CALL PRCTRACLOG
             (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, 0,
V_SQL_STATE, V_SQL_CODE, 'ERROR', V_MSG_TXT, V_CONTEXT ) ;
        END ;

        -- Interception des réponses de messages automatiques
       call qcmdexc ('CHGJOB INQMSGRPY(*SYSRPYL)') ;

        -- Valeur par défaut indispensable pour V_STEP_DEB (en cas d'erreur avant
l'ouverture de la 1ère step)
        SET V_STEP_DEB = CURRENT TIMESTAMP ;

        -- Stockage du contexte applicatif
        SET V_CONTEXT = VSENS ;

        -- incrémentation du compteur alimentant le numéro de job (indispensable pour
les logs)
        CALL PRCTRACJOB ( V_JOB_NUM ) ;

         -- Traitement des "règles métiers" - Début

          IF VSENS = 'UP' THEN
-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
```

```
-----------------------------------------------

SET SQL_STMT = 'ALTER TABLE M3DWHPRD/MF_STOCKS ADD COLUMN MBDIGR CHARACTER (2)
DEFAULT NULL';
EXECUTE IMMEDIATE SQL_STMT ;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

SET SQL_STMT = 'LABEL ON COLUMN M3DWHPRD/MF_STOCKS ( MBDIGR TEXT IS ''Groupe de
distribution'') ';
EXECUTE IMMEDIATE SQL_STMT ;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------

          END IF ;

          IF VSENS = 'DOWN' THEN
-----------------------------------------------
-- Initialisation d'une nouvelle étape
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

SET SQL_STMT = 'ALTER TABLE M3DWHPRD/MF_STOCKS DROP COLUMN MBDIGR';
EXECUTE IMMEDIATE SQL_STMT ;
-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT;
-- Ajout d'une trace dans la log
CALL PRCTRACLOG
    (V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB, CURRENT TIMESTAMP, V_NBR_ENR,
SQLSTATE, SQLCODE, 'TRACE', '', V_CONTEXT  ) ;
-----------------------------------------------
```

```
          END IF ;

        -- Traitement des "règles métiers" - Fin

END  ;
COMMENT ON SPECIFIC PROCEDURE M3DHSADPGM/UPG1110001
     IS 'Proc Upgrade DB 2011-10 numéro 001' ;
```

Attention : on rappelle ici que toute altération de table SQL déclenche automatiquement un message système nécessitant une réponse. Il est nécessaire d'ajouter la ligne ci-dessous au début de la procédure de migration, pour lui permettre d'utiliser le mécanisme de réponse automatique :

```
call qcmdexc ('CHGJOB INQMSGRPY(*SYSRPYL)') ;
```

On rappelle que ce point a été discuté en détail dans le chapitre "Création de table -> Renommer une colonne".

4.5 Encodage

Les encodages UTF sont disponibles depuis la V5R3 :

```
UTF-8 : name CHAR(10) CCSID ( 1208) ;
UTF-16 : name GRAPHIC(10) CCSID ( 1200) ;
UCS-2 : name GRAPHIC (10) CCSID ( 13488) ;
```

Affichage des charsets définis sur le système :

```
SELECT character_set_name from sysibm.syscharsets
```

Vu sur Pausecafé 62 :
http://www.volubis.fr/Pausecaf/PAUSECAF62.html

Unicode ?

A l'origine était le CCSID.

Pour un jeu de caractères donné (, , par ex.) définition d'une grille de codification de tous les caractères pour un pays donné :

Nous avons le même jeu de caractère que les espagnols, nous n'avons pas le même code page (codification).

Le CCSID est la codification du jeu de caractères et du code page d'origine d'une donnée.

"Cette données est Française ! " (CCSID au niveau zone, venant du CCSID du fichier, lui même venant du job de création)

si le CCSID du job (qui lit le fichier) est différent (il vient de la langue de l'utilisateur, sinon de QCCSID), il faut modifier la valeur héxa afin que le "é" Français s'affiche é pour l'utilisateur Espagnol ou Danois et non "{" (par exemple).

La codification est une codification sur plusieurs octets permettant dans une même grille, de coder **tous les caractères du monde** (y compris chinois, bengali, braille, symboles mathématiques et notes de musique !)

- UCS-2 projet d'origine , CCSID 13488
- UTF-16 sur-ensemble à UCS-2, normalisé par l'ISO, CCSID 1200
- UTF-8 proche d'UTF-16, mais stocke sur 1 octet les caractères occidentaux, sur 2 ou 4 les autres

Avec UCS-2 et UTF-16, une zone base de données de 20 caractères = 40 Octets

Avec UTF-8 une zone 20 cacactères = 20 Octets, donc potentiellement trop courte, à réserver au VARCHAR, CLOB et fichiers IFS.

```
A         UCS2       10G    CCSID(13488)
A* Lg de stockage = 20 octets
A         UTF16      10G    CCSID(1200)
A* Lg de stockage = 20 octets
A         UTF8       10A    CCSID(1208)
A* Lg de stockage = 10 octets
```

(voir aussi la , lors des journées "Modernisation des *apllications" d'Avril 2012)*

Vu sur la présentation de Christian Grière :

```
CREATE TABLE CLIENT1
(UCS2 GRAPHIC (10) CCSID 13488 NOT NULL WITH DEFAULT,
UTF16 GRAPHIC (10) CCSID 1200 NOT NULL WITH DEFAULT,
UTF8 CHAR (10) CCSID 1208 NOT NULL WITH DEFAULT);
```

A partir de V6R1 :

```
CREATE TABLE CLIENT2
(UTF16 NCHAR NOT NULL WITH DEFAULT);
```

Quelques articles intéressants sur les problèmes d'encodage et de conversion sous DB2 :

http://www.ibm.com/developerworks/data/library/techarticle/dm-0506chong/

http://www.youngiprofessionals.com/wiki/FastCGI

5. Etudes de Cas

5.1 Amortissement d'immobilisation

La requête suivante implémente un exemple de tableau d'amortissement, selon le principe comptable d'amortissement dit "linéaire".
Le tableau d'amortissement est généré intégralement sans l'aide d'aucune table externe, simplement en s'appuyant sur un enchaînement de CTE.
Pour constituer les lignes du tableau correspondant aux annuités d'amortissement, la requête a besoin de s'appuyer sur l'UDTF GENINCAUTO que nous avons étudiée dans le chapitre "Récursivité et RCTE".

Dans l'exemple ci-dessous, le bien à amortir a une valeur initiale de 100000 euros, et il est amorti sur 5 ans avec taux d'amortissement de 20%.
L'amortissement démarre dans le courant de l'année 2014, plus exactement en mars, donc la première annuité est calculée sur 10 mois seulement, au lieu de 12. La première annuité étant incomplète, l'amortissement effectif se fera sur 6 années au lieu de 5.

Attention : il faut se méfier des divisions en SQL. Par exemple, pour la division 20/100, DB2 SQL ne renverra pas 0,2, mais 0, car pour DB2, la division d'un entier par un autre entier doit renvoyer un entier. Pour que DB2 SQL renvoie 0,2, il faut que l'un des facteurs de l'opération soit de type "décimal". Par exemple, dans la requête ci-dessous, div2, div3 et div4 contiennent 0,2, tandis que div1 contient 0 :

```
select 20/100 as div1, 20.0/100 as div2, 20/100.0 as div3, (20+0.0)/100 as div4 from
sysibm.sysdummy1;
```

Pour bien comprendre le fonctionnement de la requête ci-dessous, on recommande d'exécuter chacune des CTE individuellement, de manière à bien comprendre le rôle de chacune.

```
-- Requête de calcul d'un tableau d'amortissement d'immobilisation selon le mode
"linéaire"

WITH
TMP_VALINITIALES (CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1, NB_ANNUITES) as  (
  SELECT CAST(100000 AS DEC(11, 0)), -- capital initial
    CAST((20+0.0) / 100 AS DEC(5, 2)), -- taux d'amortissment
    CAST(2014 AS INTEGER), -- année de départ
    CAST(10 AS INTEGER), -- nbre mois pour calcul 1ère annuitée
    CAST(5 AS INTEGER) -- nombre d'années d'amortissement
  FROM SYSIBM.SYSDUMMY1
) ,
```

```
-- Conversion de certaines valeurs initiales en format décimal et précalcul de
certaines données
TMP_VALDEPART (CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1, PRORATA, NB_ANNUITES) AS (
  SELECT CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1,
    CAST( CAST(NB_MOIS_AN1 AS DEC(2, 0)) / 12.0 AS DEC(5, 4)) as PRORATA, -- prorata
de la première annuité au format décimal
    CASE WHEN NB_MOIS_AN1 = 12 THEN NB_ANNUITES ELSE NB_ANNUITES + 1 END as
NB_ANNUITES -- nombre d'années d'amortissement de type entier
  FROM TMP_VALINITIALES
) ,
-- Calcul d'un premier tableau d'annuités théoriques
TMP_TABLEAU1 AS (
  SELECT MYUDTF .VAL_INC, (SELECT TAUX FROM TMP_VALDEPART) as taux,
    (SELECT CAPITAL FROM TMP_VALDEPART) AS CAPITAL_INITIAL,
    CASE WHEN MYUDTF.VAL_INC = 1 THEN
      -- la première année n'est pas forcément une année pleine, d'où application
d'un prorata temporis sur la mensualité
      (SELECT CAPITAL FROM TMP_VALDEPART) * (SELECT TAUX FROM TMP_VALDEPART) *
(SELECT PRORATA FROM TMP_VALDEPART)
    ELSE
      -- mensualité théorique pour une année pleine
      (SELECT CAPITAL FROM TMP_VALDEPART) * (SELECT TAUX FROM TMP_VALDEPART)
    END AS ANNUITES
  FROM TABLE (MYLIBRARY.GETINCAUTO2((SELECT NB_ANNUITES FROM  TMP_VALDEPART)) )
MYUDTF
),
-- Second tableau théorique incluant le calcul du CRD
TMP_TABLEAU2 AS (
  SELECT A.VAL_INC, A.CAPITAL_INITIAL, A.ANNUITES,
    (SELECT SUM(ANNUITES) FROM TMP_TABLEAU1 X WHERE X.VAL_INC <= A.VAL_INC) as
AMORT_CUMULE,
     A.CAPITAL_INITIAL - (SELECT SUM(ANNUITES) FROM TMP_TABLEAU1 X WHERE X.VAL_INC <=
A.VAL_INC) AS CRD
  FROM TMP_TABLEAU1 A
),
-- Rattrapage de la dernière annuité si CRD négatif sur la dernière année
TMP_TABLEAU3 AS (
  SELECT A.VAL_INC + (SELECT ANNEE_DEPART FROM TMP_VALDEPART) - 1 AS ANNEE,
     A.CAPITAL_INITIAL,
     CASE WHEN CRD < 0 THEN
        A.ANNUITES + CRD
     ELSE
        A.ANNUITES
     END AS AMORTISSEMENT,
     AMORT_CUMULE,
     CASE WHEN CRD < 0 THEN
        0
     ELSE
        A.CRD
     END AS CRD
     FROM TMP_TABLEAU2 A
)
SELECT * FROM TMP_TABLEAU3
```

;

5.2 Détection de périodes d'inactivité

Cette étude de cas présente une technique de détection de périodes d'inactivité, au sein d'une table des activités délimitée par une notion de collaborateur et de date de début et de fin d'activité.

Dans un cas de ce type, la difficulté principale consiste à faire "matcher", pour un collaborateur donné, la date de fin d'activité d'une période d'activité, qui se trouve sur une ligne de la table, avec la date de début d'activité d'une autre période d'activité, qui se trouve sur une autre ligne de la même table. Ce sont donc les "trous" entre date de fin d'activité précédente et date de début d'activité suivante que l'on cherche à détecter. Quelquefois, il n'y a pas de seconde ligne définissant une nouvelle activité, il y a dans ce cas un trou plus important délimité par la date de fin d'activité précédente et la date du jour, ou encore entre la date du jour et la date de début d'activité.

L'étude de cas présentée ici est "tirée" d'un cas réel dans lequel les dates n'étaient pas de véritables colonnes de type date, mais des colonnes de type numérique (8.0). Une fonction de conversion de date était donc nécessaire, ce point est couvert par la fonction CVT_ALP_2_DATE présentée ci-dessous.

La requête d'identification des périodes d'inactivité fonctionne aussi bien sur DB2 for i que sur DB2 Express C (version 9.7).

```
-- fonction de conversion de date utilisée dans le cadre du projet (pour DB2 for i)
CREATE OR REPLACE FUNCTION MABASETEST.CVT_ALP_2_DATE (
       DATE_ENT CHAR(10) )
       RETURNS DATE
       LANGUAGE SQL
       SPECIFIC MABASETEST/CVTALP2DAT
       DETERMINISTIC
       CONTAINS SQL
       CALLED ON NULL INPUT
       SET OPTION  ALWBLK = *ALLREAD ,
       ALWCPYDTA = *OPTIMIZE ,
       COMMIT = *NONE ,
       DECRESULT = (31, 31, 00) ,
       DFTRDBCOL = *NONE ,
       DYNDFTCOL = *NO ,
       DYNUSRPRF = *USER ,
       SRTSEQ = *HEX
       RETURN
```

```
CASE WHEN DATE_ENT IS NULL OR TRIM ( DATE_ENT ) = ''
THEN NULL
-- la fonction TO_DATE() renvoie un timestamp DB2, tranformé en type date par la
fonction DATE()
ELSE DATE ( TO_DATE ( DATE_ENT , 'YYYY-MM-DD' ) )
END  ;

-- Même fonction compatible avec DB2 Express C :
CREATE OR REPLACE FUNCTION MABASETEST.CVT_ALP_2_DATE (
       DATE_ENT CHAR(10) )
       RETURNS DATE
       LANGUAGE SQL
       DETERMINISTIC
       CONTAINS SQL
       CALLED ON NULL INPUT
       RETURN
CASE WHEN DATE_ENT IS NULL OR TRIM ( DATE_ENT ) = ''
THEN NULL
ELSE DATE ( TO_DATE ( DATE_ENT , 'YYYY-MM-DD' ) )
END  ;
```

Pour les besoins de la démonstration, nous avons besoin d'une table que j'ai appelée TSTACTIVITE.
Pour faciliter l'analyse du résultat de la requête, nous décidons d'alimenter une table TSTVACANCE - de structure identique à TSTACTIVITE - avec le périodes d'inactivité des collaborateurs.

```
-- Structure des tables d'origine et de destination
-- Attention : pour DB2 Express C, penser à supprimer la notion de CCSID

CREATE TABLE MABASETEST.TSTACTIVITE (
NO_ID CHAR(30) CCSID 297 DEFAULT NULL,
SYS_ORIG CHAR(6) CCSID 297 DEFAULT NULL,
LIB_PRENOM_USUEL CHAR(50) CCSID 297 DEFAULT NULL,
LIB_NOM_USUEL CHAR(50) CCSID 297 DEFAULT NULL,
EMPLOI CHAR(120) CCSID 297 DEFAULT NULL,
DAT_DEB_EMPLOI CHAR(10) CCSID 297 DEFAULT NULL,
DAT_FIN_EMPLOI CHAR(10) CCSID 297 DEFAULT NULL,
COD_OPER CHAR(8) CCSID 297 DEFAULT NULL,
TMP_CRE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
TMP_MAJ TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ;


CREATE TABLE MABASETEST.TSTVACANCE (
NO_ID CHAR(30) CCSID 297 DEFAULT NULL,
SYS_ORIG CHAR(6) CCSID 297 DEFAULT NULL,
LIB_PRENOM_USUEL CHAR(50) CCSID 297 DEFAULT NULL,
LIB_NOM_USUEL CHAR(50) CCSID 297 DEFAULT NULL,
EMPLOI CHAR(120) CCSID 297 DEFAULT NULL,
DAT_DEB_EMPLOI CHAR(10) CCSID 297 DEFAULT NULL,
```

```
DAT_FIN_EMPLOI CHAR(10) CCSID 297 DEFAULT NULL,
COD_OPER CHAR(8) CCSID 297 DEFAULT NULL,
TMP_CRE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
TMP_MAJ TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ;
```

Les 2 tables étant créées, on pourra injecter dans la table TSTACTIVITE le jeu de données fourni à la fin de ce chapitre.

La requête d'identification des périodes d'inactivité ci-dessous a été testée avec succès sur DB2 for i et sur DB2 Express C (version 9.7).
Elle permet d'insérer dans la table TSTVACANCE la liste des périodes d'inactivité de chacun des collaborateurs.
Pour bien comprendre son fonctionnement, il est recommandé d'exécuter individuellement chacune des CTE

```
-- requête d'insertion dans la table TSTVACANCE des périodes d'inactivité
-- identifiées dans les requêtes qui suivent :
insert into MABASETEST.TSTVACANCE (no_id, sys_orig, lib_prenom_usuel, lib_nom_usuel,
emploi, dat_deb_emploi, dat_fin_emploi, cod_oper, tmp_cre, tmp_maj)

-- première requête temporaire pour sélectionner les seule lignes qui nous
intéressent, on en profite pour convertir les dates alpha en vraies dates DB2
with temp1 as (
select no_id, MABASETEST.CVT_ALP_2_DATE(dat_deb_emploi) as dat_deb_emploi,
       MABASETEST.CVT_ALP_2_DATE(dat_fin_emploi) as  dat_fin_emploi
from MABASETEST.TSTACTIVITE
where no_id is not null and no_id <> '' and emploi is not null and trim(emploi) <> ''
  and dat_deb_emploi is not null and dat_fin_emploi is not null
group by no_id, dat_deb_emploi, dat_fin_emploi
) ,
-- création d'une rupture sur no_id et dat_deb_emploi
temp2 as (
select no_id, dat_deb_emploi, dat_fin_emploi,
   row_number() over(partition by no_id order by no_id, dat_deb_emploi)  as rupture
from (select * from temp1
       where dat_deb_emploi is not null and dat_fin_emploi is not null
       order by no_id, dat_deb_emploi) a
order by no_id, dat_deb_emploi
) ,
-- création d'un lien artificiel vers une "rupture suivante" (si la rupture suivante
n'existe pas, ce n'est pas grave, car "rupture" et "rupture suivante" seront liées
par un INNER JOIN)
temp3 as (
select a.*, a.rupture+1 as rupture_suivante from temp2 a
) ,
-- sélection finale des périodes de vacance (on ne prend que les cas où
-- x.vacance > 0 car les autres cas correspondent à des chevauchements de dates
temp4 as (
```

```
select * from (
select a.no_id, a.dat_fin_emploi + 1 day as dat_deb_inactif,
       b.dat_deb_emploi - 1 day as dat_fin_inactif,
       days(b.dat_deb_emploi) - days(a.dat_fin_emploi) - 1 as vacance
from (select * from temp3 order by no_id, dat_deb_emploi) a
inner join (select * from temp3 order by no_id, dat_deb_emploi) b
on a.no_id = b.no_id and a.rupture_suivante = b.rupture
) x
where x.vacance > 0
)

select no_id, 'INACTIF' as code,
  (select x.lib_prenom_usuel from MABASETEST.TSTACTIVITE x where a.no_id = x.no_id
fetch first 1 row only) as prenom ,
  (select y.lib_nom_usuel from MABASETEST.TSTACTIVITE y where a.no_id = y.no_id fetch
first 1 row only) as nom ,
  'INACTIVITE' as situation,
  char( dat_deb_inactif, iso ) as dat_deb_inactif,
  char( dat_fin_inactif, iso ) as dat_fin_inactif,
  'MYUSERCODE' as cod_oper, current timestamp as tmp_cre, current timestamp as
tmp_maj
from temp4 a
;
```

Pour vérifier le bon fonctionnement de la requête ci-dessus, il nous faut un jeu d'essai, que voici :

```
-- Jeu de données

DELETE FROM MABASETEST.TSTACTIVITE ;
INSERT INTO MABASETEST.TSTACTIVITE
( NO_ID, SYS_ORIG, LIB_PRENOM_USUEL, LIB_NOM_USUEL, EMPLOI, DAT_DEB_EMPLOI,
DAT_FIN_EMPLOI, COD_OPER, TMP_CRE, TMP_MAJ )
VALUES
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2008-07-01', '2008-07-
31', '', '2012-08-17 13:21:44.480870', '2012-08-17 13:21:44.480870' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2008-02-01', '2008-02-
29', '', '2012-08-17 13:21:44.481767', '2012-08-17 13:21:44.481767' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2007-07-01', '2007-07-
31', '', '2012-08-17 13:21:44.482747', '2012-08-17 13:21:44.482747' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2007-02-01', '2007-02-
28', '', '2012-08-17 13:21:44.483532', '2012-08-17 13:21:44.483532' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2006-07-01', '2006-07-
31', '', '2012-08-17 13:21:44.484518', '2012-08-17 13:21:44.484518' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2006-02-01', '2006-02-
28', '', '2012-08-17 13:21:44.485299', '2012-08-17 13:21:44.485299' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2005-07-01', '2005-07-
31', '', '2012-08-17 13:21:44.486100', '2012-08-17 13:21:44.486100' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2005-02-01', '2005-02-
28', '', '2012-08-17 13:21:44.486874', '2012-08-17 13:21:44.486874' ),
```

```
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2004-07-01', '2004-07-
31', '', '2012-08-17 13:21:44.487667', '2012-08-17 13:21:44.487667' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2004-01-01', '2004-02-
29', '', '2012-08-17 13:21:44.488450', '2012-08-17 13:21:44.488450' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2003-07-16', '2003-07-
16', '', '2012-08-17 13:21:44.489437', '2012-08-17 13:21:44.489437' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2003-06-30', '2003-07-
15', '', '2012-08-17 13:21:44.490216', '2012-08-17 13:21:44.490216' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2003-02-03', '2003-02-
03', '', '2012-08-17 13:21:44.491207', '2012-08-17 13:21:44.491207' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2002-07-01', '2002-07-
18', '', '2012-08-17 13:21:44.491995', '2012-08-17 13:21:44.491995' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2002-02-04', '2002-02-
16', '', '2012-08-17 13:21:44.492779', '2012-08-17 13:21:44.492779' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2001-07-02', '2001-07-
17', '', '2012-08-17 13:21:44.493566', '2012-08-17 13:21:44.493566' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2001-02-05', '2001-02-
20', '', '2012-08-17 13:21:44.494364', '2012-08-17 13:21:44.494364' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2000-07-03', '2000-07-
19', '', '2012-08-17 13:21:44.495163', '2012-08-17 13:21:44.495163' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '2000-02-07', '2000-02-
19', '', '2012-08-17 13:21:44.495931', '2012-08-17 13:21:44.495931' ),
( '11492', '', 'ISABELLE', 'ROBERT', 'VENDEUR(EUSE) COLL', '1999-07-06', '1999-07-
14', '', '2012-08-17 13:21:44.496899', '2012-08-17 13:21:44.496899' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', '', '2012-04-09', '', '', '2012-08-17
13:21:42.121245', '2012-08-17 13:21:42.121245' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'DECORATEUR(TRICE)', '2009-01-01', '2012-04-
08', '', '2012-08-17 13:21:42.124000', '2012-08-17 13:21:42.124000' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'DECORATEUR(TRICE)', '2009-01-01', '2012-04-
08', '', '2012-08-17 13:21:42.125170', '2012-08-17 13:21:42.125170' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2005-01-01', '2008-12-31', '',
'2012-08-17 13:21:42.126143', '2012-08-17 13:21:42.126143' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2005-01-01', '2008-12-31', '',
'2012-08-17 13:21:42.127119', '2012-08-17 13:21:42.127119' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2005-01-01', '2008-12-31', '',
'2012-08-17 13:21:42.128100', '2012-08-17 13:21:42.128100' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2005-01-01', '2008-12-31', '',
'2012-08-17 13:21:42.129097', '2012-08-17 13:21:42.129097' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2005-01-01', '2008-12-31', '',
'2012-08-17 13:21:42.130264', '2012-08-17 13:21:42.130264' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2004-01-01', '2004-12-31', '',
'2012-08-17 13:21:42.131247', '2012-08-17 13:21:42.131247' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2003-09-01', '2003-12-31', '',
'2012-08-17 13:21:42.132229', '2012-08-17 13:21:42.132229' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2003-08-01', '2003-08-31', '',
'2012-08-17 13:21:42.133035', '2012-08-17 13:21:42.133035' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '2003-08-01', '2003-08-31', '',
'2012-08-17 13:21:42.134010', '2012-08-17 13:21:42.134010' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '1996-06-01', '2003-07-31', '',
'2012-08-17 13:21:42.134979', '2012-08-17 13:21:42.134979' ),
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '1996-06-01', '2003-07-31', '',
'2012-08-17 13:21:42.135982', '2012-08-17 13:21:42.135982' ),
```

```
( '8083', '', 'MARIE FRANCE', 'CHAN', 'OUV QUALIFIE', '1996-01-01', '1996-05-31', '',
'2012-08-17 13:21:42.136950', '2012-08-17 13:21:42.136950' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'ASSISTANT(E)', '2011-01-01', '', '', '2012-08-17
13:21:42.137930', '2012-08-17 13:21:42.137930' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'ASSISTANT(E)', '2011-01-01', '', '', '2012-08-17
13:21:42.138918', '2012-08-17 13:21:42.138918' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'ASSISTANT(E)', '2011-01-01', '', '', '2012-08-17
13:21:42.139902', '2012-08-17 13:21:42.139902' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'ASSISTANT(E)', '2010-01-01', '2010-12-31', '',
'2012-08-17 13:21:42.140879', '2012-08-17 13:21:42.140879' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'SECRETAIRE', '2007-02-01', '2009-12-31', '',
'2012-08-17 13:21:42.141869', '2012-08-17 13:21:42.141869' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'SECRETAIRE', '2007-02-01', '2009-12-31', '',
'2012-08-17 13:21:42.143042', '2012-08-17 13:21:42.143042' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'SECRETAIRE', '2007-02-01', '2009-12-31', '',
'2012-08-17 13:21:42.143920', '2012-08-17 13:21:42.143920' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2006-01-01', '2007-01-31', '',
'2012-08-17 13:21:42.144818', '2012-08-17 13:21:42.144818' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2006-01-01', '2007-01-31', '',
'2012-08-17 13:21:42.145610', '2012-08-17 13:21:42.145610' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2006-01-01', '2007-01-31', '',
'2012-08-17 13:21:42.146394', '2012-08-17 13:21:42.146394' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2004-01-01', '2005-12-31', '',
'2012-08-17 13:21:42.147179', '2012-08-17 13:21:42.147179' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2003-09-01', '2003-12-31', '',
'2012-08-17 13:21:42.147961', '2012-08-17 13:21:42.147961' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2003-08-01', '2003-08-31', '',
'2012-08-17 13:21:42.148749', '2012-08-17 13:21:42.148749' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '2003-08-01', '2003-08-31', '',
'2012-08-17 13:21:42.149538', '2012-08-17 13:21:42.149538' ),
( '8084', '', 'CAROLINE', 'MARTIN', 'EMPL. ADMIN.', '1998-03-02', '2003-07-31', '',
'2012-08-17 13:21:42.150514', '2012-08-17 13:21:42.150514' ),
;
```

5.3 ChgNomCour

La procédure stockée CHGNOMCOUR vise à résoudre le problème des conflits entre noms longs et courts, tel qu'il a été présenté dans le chapitre "Création de Table -> Conflits sur noms courts".

Petite explication préliminaire : dans la procédure CHGNOMCOUR, on a fait en sorte que toutes les manipulations d'objets SQL effectuées soient tracées dans une table temporaire QTEMP.TMP_TRACE. On pourra après chaque appel analyser le contenu de cette table pour s'assurer qu'aucune anomalie ne s'est produite. On pourrait également faire évoluer la procédure pour qu'elle renvoie en sortie un code retour indiquant si tout s'est bien passé, ou pas.

L'étude de cette procédure est intéressante car elle utilise massivement les tables systèmes étudiées dans les chapitres précédents.

Code source de la procédure CHGNOMCOUR :

```
CREATE OR REPLACE PROCEDURE MABIBPROC/CHGNOMCOUR (
     IN OBJBIB VARCHAR(10),
     IN OBJNOMLONG VARCHAR(80),
     IN OBJNOMCOURT VARCHAR(10) )
     LANGUAGE SQL
     SPECIFIC MABIBPROC/CHGNOMCOUR
     NOT DETERMINISTIC
     MODIFIES SQL DATA
     CALLED ON NULL INPUT
     SET OPTION ALWBLK = *ALLREAD ,
     -- TGTRLS = *CURRENT ,
     ALWCPYDTA = *OPTIMIZE ,
     COMMIT = *NONE ,
     CLOSQLCSR = *ENDMOD ,
     DATFMT = *ISO ,
     TIMFMT = *ISO ,
     DECMPT = *JOB ,
     DECRESULT = (31, 31, 00) ,
     DFTRDBCOL = *NONE ,
     DYNDFTCOL = *NO ,
     DYNUSRPRF = *USER ,
     SRTSEQ = *HEX ,
     OUTPUT = *PRINT ,
     DBGVIEW = *NONE
     BEGIN
       -- Déclaration du nom de la procédure courante (pour gestion des logs)
       DECLARE V_NOM_PROC CHAR(20) DEFAULT 'CHGNOMCOUR';
       -- Déclaration du contexte applicatif (pour gestion des logs)
       DECLARE V_CONTEXT VARCHAR(30) DEFAULT '';
```

```
        -- Déclaration de la variable définissant l'étape courante (pour log)
        DECLARE V_STEP_NUM INTEGER DEFAULT 0;
        DECLARE V_STEP_DEB TIMESTAMP ;
        DECLARE V_STEP_FIN TIMESTAMP ;
        DECLARE V_SERVER VARCHAR(10) ;
        DECLARE V_USER VARCHAR(10) ;
        DECLARE V_TYP_MSG VARCHAR(10);

        -- Déclaration de la variable servant à stocker le numéro de job courant
        DECLARE V_JOB_NUM INTEGER DEFAULT 0;
        -- Déclaration nombre d'enregistrements retournés par GET DIAGNOSTICS
(ROWCOUNT)
        DECLARE V_NBR_ENR INTEGER DEFAULT 0;
        -- Déclaration des "SQL return codes"
        DECLARE SQLCODE INTEGER DEFAULT 0;
        DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
         -- Déclaration des variables destinées à alimenter la log en cas d'erreurs
SQL
        DECLARE V_SQL_STATE CHAR(5) DEFAULT '00000';
        DECLARE V_SQL_CODE INT DEFAULT 0;
        DECLARE V_MSG_TXT VARCHAR(100) DEFAULT '';
        DECLARE V_MSG_TXTLEN INTEGER DEFAULT 0;
        -- Déclaration de variables susceptibles d'être utilisées sur des requêtes
dynamiques
        DECLARE SQL_STMT VARCHAR(2048) DEFAULT '';
        DECLARE V_PARAM1 CHAR(10) DEFAULT '';
        DECLARE V_PARAM2 CHAR(10) DEFAULT '';
        DECLARE V_PARAM3 CHAR(10) DEFAULT '';
        DECLARE V_PARNU1 INTEGER DEFAULT 0;
        DECLARE V_PARNU2 INTEGER DEFAULT 0;
        DECLARE V_PARNU3 INTEGER DEFAULT 0;
        -- Déclaration de jeux de 2 jeux de date prévus pour divers usages (chaque
date dans 2 formats)
        -- Date de valeur
        DECLARE V_DATVAL DATE ;
        DECLARE V_DATVAL8 DEC(8, 0) DEFAULT 0 ;
        -- Date de période (utile pour traitements hebdo et mensuels notamment)
        DECLARE V_DATPER DATE ;
        DECLARE V_DATPER8 DEC(8, 0) DEFAULT 0 ;

        DECLARE V_NOM_LONG VARCHAR(50) ;
        DECLARE V_NOM_COURT CHAR(10) ;
        DECLARE V_NBR_OBJ INTEGER DEFAULT 0;


        DECLARE TMP_STMT VARCHAR(512) DEFAULT
          'INSERT INTO QTEMP.TMP_TRACE (T_NOM_SERV, T_NUM_JOB, T_NOM_PROC, T_STEP_NUM,
T_STEP_DEB, T_STEP_FIN, T_NBR_ENR, T_SQL_STATE, T_SQL_CODE, T_MSG_TYP, T_MSG_TXT,
T_USER) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';

        -- Pavé de gestion des Données non trouvées
        DECLARE CONTINUE HANDLER FOR NOT FOUND
        BEGIN
```

```
          SET V_SQL_STATE = SQLSTATE;
          SET V_SQL_CODE  = SQLCODE;
          SET V_NBR_ENR = 0 ;
          SET V_TYP_MSG = 'NOTFOUND';
          SET V_MSG_TXT = '' ;
          EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
       END ;

        -- Pavé de gestion des Avertissements
       DECLARE CONTINUE HANDLER FOR SQLWARNING
       BEGIN
          SET V_SQL_STATE = SQLSTATE;
          SET V_SQL_CODE  = SQLCODE;
          SET V_NBR_ENR = 0 ;
          SET V_TYP_MSG = 'WARNING';
          SET V_MSG_TXT = '';
          EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
       END ;

       -- Pavé de gestion des Erreurs
       DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
       BEGIN
          SET V_SQL_STATE = SQLSTATE;
          SET V_SQL_CODE  = SQLCODE;
          GET DIAGNOSTICS EXCEPTION 1
             V_MSG_TXT    = MESSAGE_TEXT,
             V_MSG_TXTLEN = MESSAGE_LENGTH;
             SET V_NBR_ENR = 0 ;
             IF V_MSG_TXTLEN > 100 THEN
                 SET V_MSG_TXT = SUBSTR(V_MSG_TXT, 1, 100) ;
             END IF ;
             SET V_TYP_MSG = 'ERROR';
             EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM,
V_STEP_DEB, V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
       END ;

       -- Valeur par défaut indispensable pour V_STEP_DEB (en cas d'erreur avant
l'ouverture de la 1ère step)
       SET V_STEP_DEB = CURRENT TIMESTAMP ;
       SET V_STEP_FIN = CURRENT TIMESTAMP ;
       SET V_SERVER = CURRENT SERVER ;
       SET V_USER = USER ;

        -- Traitement des "règles métiers" - Début
-----------------------------------------------
-- Initialisation de l'étape 1
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

-- Création du statement pour l'insertion dans TMP_TRACE
```

```
PREPARE s1d FROM TMP_STMT;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
-- Ajout d'une trace dans la log
SET V_TYP_MSG = 'TRACE';
SET V_MSG_TXT = 'Préparation statement s1d';
SET V_STEP_FIN = CURRENT TIMESTAMP ;
EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
-----------------------------------------------


-----------------------------------------------
-- Initialisation de l'étape 1
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
-----------------------------------------------

-- Création de la table temporaire de log
DECLARE GLOBAL TEMPORARY TABLE TMP_TRACE (
    T_NOM_SERV CHAR(20) CCSID 297 NOT NULL,
    T_NUM_JOB INTEGER NOT NULL,
    T_NOM_PROC CHAR(20) CCSID 297 NOT NULL,
    T_STEP_NUM INTEGER NOT NULL,
    T_STEP_DEB TIMESTAMP NOT NULL,
    T_STEP_FIN TIMESTAMP NOT NULL,
    T_NBR_ENR BIGINT NOT NULL,
    T_SQL_CODE INTEGER NOT NULL,
    T_SQL_STATE CHAR(5) CCSID 297 NOT NULL,
    T_MSG_TYP CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
    T_MSG_TXT VARCHAR(200) ALLOCATE(10) CCSID 297 NOT NULL DEFAULT '' ,
    T_USER CHAR(20) CCSID 297 NOT NULL,
    T_CRE_LOG TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
    T_CONTEXT VARCHAR(30) ALLOCATE(10) CCSID 297 NOT NULL DEFAULT '' ,
    T_CONO DECIMAL(3, 0) NOT NULL DEFAULT 0
) WITH REPLACE ;

-----------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
-- Ajout d'une trace dans la log
SET V_TYP_MSG = 'TRACE';
SET V_MSG_TXT = 'Création table log';
SET V_STEP_FIN = CURRENT TIMESTAMP ;
EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
-----------------------------------------------



-----------------------------------------------
```

```
-- Initialisation de l'étape 1
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
---------------------------------------------

---------------------------------------------
-- Cas où le nom court est monopolisé par un index (si ça arrive, on le supprime)
SET SQL_STMT = 'DROP INDEX ' concat trim(OBJBIB) concat '.' concat trim(OBJNOMCOURT);
EXECUTE IMMEDIATE SQL_STMT ;

---------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
-- Ajout d'une trace dans la log
SET V_TYP_MSG = 'TRACE';
SET V_MSG_TXT = 'Suppression index';
SET V_STEP_FIN = CURRENT TIMESTAMP ;
EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
---------------------------------------------


---------------------------------------------
-- Initialisation de l'étape 1
SET V_STEP_NUM = V_STEP_NUM + 1 ;
SET V_STEP_DEB = CURRENT TIMESTAMP ;
---------------------------------------------

-- Requête pour vérifier si nom court et longs sont déjà concordants
select sum(comptage) into V_NBR_OBJ  from (
    select count(*) as comptage
    from qsys2.systables
    where system_table_schema = trim(OBJBIB)
      and table_name = trim(OBJNOMLONG)
      and system_table_name = trim(OBJNOMCOURT)
  union
    select count(*) as comptage
    from qsys2.sysviews
    where system_view_schema = trim(OBJBIB)
      and table_name = trim(OBJNOMLONG)
      and system_view_name = trim(OBJNOMCOURT)
) x ;

---------------------------------------------
-- Diagnostic de la dernière requête exécutée
GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
-- Ajout d'une trace dans la log
SET V_TYP_MSG = 'TRACE';
SET V_MSG_TXT = 'Détection concordance';
SET V_STEP_FIN = CURRENT TIMESTAMP ;
EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_OBJ, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
```

```
-----------------------------------------------

IF (V_NBR_OBJ = 1) THEN
   -- Nom court et nom long concordants, donc il n'y a rien à faire
   SET V_TYP_MSG = 'TRACE';
   SET V_MSG_TXT = 'Noms concordants, traitement stoppé';
   SET V_STEP_FIN = CURRENT TIMESTAMP ;
   EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_OBJ, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
END IF ;

IF (V_NBR_OBJ = 0) THEN
    -- Nom court et nom long non concordants, on doit "renommer" l'objet cible, avec
    -- éventuellement permutation de noms courts si 2 objets sont en conflit

    -----------------------------------------------
    -- Initialisation de l'étape 1
    SET V_STEP_NUM = V_STEP_NUM + 1 ;
    SET V_STEP_DEB = CURRENT TIMESTAMP ;
    -----------------------------------------------

    -- On contrôle si le nom court est "préempté" par un autre objet
    select sum(comptage) into V_NBR_OBJ  from (
        select count(*) as comptage
        from qsys2.systables
        where system_table_schema = trim(OBJBIB)
          and system_table_name = trim(OBJNOMCOURT)
      union
        select count(*) as comptage
        from qsys2.sysviews
        where system_view_schema = trim(OBJBIB)
          and system_view_name = trim(OBJNOMCOURT)
    ) x ;


    -----------------------------------------------
    -- Diagnostic de la dernière requête exécutée
    GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
    -- Ajout d'une trace dans la log
    SET V_TYP_MSG = 'TRACE';
    SET V_MSG_TXT = 'Comptage nb.objets (2)';
    SET V_STEP_FIN = CURRENT TIMESTAMP ;
    EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
    -----------------------------------------------

    -- Le nom court n'est monopolisé par "personne", on fait le "rename" et c'est
fini
    IF (V_NBR_OBJ = 0) THEN
        -----------------------------------------------
        -- Initialisation de l'étape 1
        SET V_STEP_NUM = V_STEP_NUM + 1 ;
        SET V_STEP_DEB = CURRENT TIMESTAMP ;
```

```
        ------------------------------------------------

        SET SQL_STMT = 'RENAME TABLE ' concat trim(OBJBIB) concat '.' concat
            trim(OBJNOMLONG) concat ' TO SYSTEM NAME '  concat trim(OBJNOMCOURT);
        EXECUTE IMMEDIATE SQL_STMT ;
        ------------------------------------------------
        -- Diagnostic de la dernière requête exécutée
        GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
        -- Ajout d'une trace dans la log
        SET V_TYP_MSG = 'TRACE';
        SET V_MSG_TXT = 'Renommage sans permutation';
        SET V_STEP_FIN = CURRENT TIMESTAMP ;
        EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
        ------------------------------------------------

    END IF ;

    IF (V_NBR_OBJ = 1) THEN
        -- On récupère le nom long de l'objet qui monopolise le nom court qui nous
intéresse
        select x.table_name into V_NOM_LONG from (
            select table_name
            from qsys2.systables
            where system_table_schema = trim(OBJBIB)
              and system_table_name = trim(OBJNOMCOURT)
            union
            select table_name
            from qsys2.sysviews
            where system_view_schema = trim(OBJBIB)
              and system_view_name = trim(OBJNOMCOURT)
        ) x
        where x.table_name is not null
        fetch first 1 row only
        ;
    END IF ;
    IF (LENGTH(RTRIM(V_NOM_LONG)) < 11) THEN
       -- Ajout d'une trace dans la log
       SET V_TYP_MSG = 'TRACE';
       SET V_MSG_TXT = 'Nom long < 10 c, traitement stoppé';
       SET V_STEP_FIN = CURRENT TIMESTAMP ;
       EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;

       -- on positionne V_NBR_OBJ à zéro pour stopper le traitement
       SET V_NBR_OBJ = 0 ;
    END IF ;

    IF (V_NBR_OBJ = 1) THEN

        -- On récupère le nom court de l'objet que l'on souhaite "renommer"
        select x.system_table_name into V_NOM_COURT from (
            select system_table_name
```

```
            from qsys2.systables
            where system_table_schema = trim(OBJBIB)
              and table_name = trim(OBJNOMLONG)
            union
            select system_view_name as system_table_name
            from qsys2.sysviews
            where system_view_schema = trim(OBJBIB)
              and table_name = trim(OBJNOMLONG)
        ) x
        where x.system_table_name is not null
        fetch first 1 row only
        ;

        -- Permutation des noms courts en 3 étapes :

        ---------------------------------------------
        -- Initialisation de l'étape 1
        SET V_STEP_NUM = V_STEP_NUM + 1 ;
        SET V_STEP_DEB = CURRENT TIMESTAMP ;
        ---------------------------------------------

        -- étape 1 : on commence par mettre un nom court bidon sur l'objet qui
        --    tient le nom court qui nous intéresse
        SET SQL_STMT = 'RENAME TABLE ' concat trim(OBJBIB) concat '.' concat
            trim(V_NOM_LONG) concat ' TO SYSTEM NAME TMPXXXXXX';
        EXECUTE IMMEDIATE SQL_STMT ;
        ---------------------------------------------
        -- Diagnostic de la dernière requête exécutée
        GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
        -- Ajout d'une trace dans la log
        SET V_TYP_MSG = 'TRACE';
        SET V_MSG_TXT =  'Permutation (1) : ' concat SQL_STMT;
        SET V_STEP_FIN = CURRENT TIMESTAMP ;
        EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
        ---------------------------------------------

        ---------------------------------------------
        -- Initialisation de l'étape 1
        SET V_STEP_NUM = V_STEP_NUM + 1 ;
        SET V_STEP_DEB = CURRENT TIMESTAMP ;
        ---------------------------------------------

        -- étape 2 : on attribue le nom court à l'objet qui nous intéresse
        SET SQL_STMT = 'RENAME TABLE ' concat trim(OBJBIB) concat '.' concat
            trim(OBJNOMLONG) concat ' TO SYSTEM NAME '  concat trim(OBJNOMCOURT);
        EXECUTE IMMEDIATE SQL_STMT ;
        ---------------------------------------------
        -- Diagnostic de la dernière requête exécutée
        GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
        -- Ajout d'une trace dans la log
        SET V_TYP_MSG = 'TRACE';
        SET V_MSG_TXT =  'Permutation (2) : ' concat SQL_STMT;
```

```
        SET V_STEP_FIN = CURRENT TIMESTAMP ;
        EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
        ------------------------------------------------

        ------------------------------------------------
        -- Initialisation de l'étape 1
        SET V_STEP_NUM = V_STEP_NUM + 1 ;
        SET V_STEP_DEB = CURRENT TIMESTAMP ;
        ------------------------------------------------

        -- étape 3 : on attribue le nom court de l'objet qui nous intéresse
        --            à l'objet qui "tenait" le nom court que l'on souhaitait
        --            récupérer
        SET SQL_STMT = 'RENAME TABLE ' concat trim(OBJBIB) concat '.' concat
            trim(V_NOM_LONG) concat ' TO SYSTEM NAME '  concat trim(V_NOM_COURT);
        EXECUTE IMMEDIATE SQL_STMT ;
        ------------------------------------------------
        -- Diagnostic de la dernière requête exécutée
        GET DIAGNOSTICS V_NBR_ENR = ROW_COUNT ;
        -- Ajout d'une trace dans la log
         SET V_TYP_MSG = 'TRACE';
        SET V_MSG_TXT =  'Permutation (3) : ' concat SQL_STMT;
        SET V_STEP_FIN = CURRENT TIMESTAMP ;
        EXECUTE s1d USING V_SERVER, V_JOB_NUM, V_NOM_PROC, V_STEP_NUM, V_STEP_DEB,
V_STEP_FIN, V_NBR_ENR, SQLSTATE, SQLCODE, V_TYP_MSG, V_MSG_TXT, V_USER ;
        ------------------------------------------------

   END IF ;

END IF ;

        -- Traitement des "règles métiers" - Fin

END  ;
COMMENT ON SPECIFIC PROCEDURE MABIBPROC/CHGNOMCOUR
     IS 'Proc pour changer le nom court d''un objet DB2' ;
```

5.4 Retrouver le nom court d'un objet

Retrouver le nom court d'un objet DB2 n'est pas un problème quand on sait dans quelle bibliothèque cet objet se trouve.
Par exemple, pour retrouver le nom court d'une table ARTICLE_VALORISE dont on sait qu'elle se trouve dans la bibliothèque MABIBL, on peut écrire la requête suivante :

```
SELECT TRIM(SYSTEM_TABLE_SCHEMA) CONCAT '.' CONCAT SYSTEM_TABLE_NAME AS SHORT_NAME
FROM QSYS2.SYSTABLES
WHERE TABLE_SCHEMA = 'MABIBL' AND TABLE_NAME = 'ARTICLE_VALORISE' ;
```

En revanche, retrouver le nom court d'un objet DB2 quand on travaille en utilisant le principe des listes de bibliothèques, principe cher aux développeurs IBMi, cela devient problématique. Mais on peut contourner la difficulté en s'appuyant un peu sur CL et beaucoup sur SQL.

En premier lieu, on a besoin d'un CL qui nous permettra de récupérer la liste des bibliothèques du travail en cours d'exécution. Cette information est facilement accessible via la commande IBMi RTVJOBA. On en profite pour écrire un CL capable de récupérer la liste des bibliothèques utilisateurs, la liste des bibliothèque système, et la bibliothèque courante par défaut si elle est définie. Ces 3 informations sont renvoyées dans 3 paramètres distincts.

Code source du CL :

```
/* Source du programme CLP MABIBPGM/DWHSRC MBR(RTVJOBALIB)                  */
/* Récupération de la liste des bibliothèques d'un travail IBMi à partir de */
/* la commande système RTVJOBA.                                             */
/* Le programme récupère 3 types de listes de bibliothèques :               */
/* - liste des bibliothèques utilisateur                                    */
/* - liste des bibliothèques système                                        */
/* - bibliothèque par défaut (CURLIB)                                       */
             PGM        PARM(&USRLIBL &SYSLIBL &CURLIB)
             DCL        VAR(&SYSLIBL) TYPE(*CHAR) LEN(165)
             DCL        VAR(&CURLIB) TYPE(*CHAR) LEN(10)
             DCL        VAR(&USRLIBL) TYPE(*CHAR) LEN(2750)
             RTVJOBA    SYSLIBL(&SYSLIBL) CURLIB(&CURLIB) +
                          USRLIBL(&USRLIBL)
 FIN:        ENDPGM
```

Une fois le CL créé, on va l'encapsuler dans une procédure stockée de type externe pour pouvoir l'utiliser en SQL :

```
-- Procédure externe encapsulant le CLP RTVJOBALIB pour récupérer les
--  listes de bibliothèque utilisateur, système, et curlib pour un travail IBMi
```

```
CREATE OR REPLACE PROCEDURE MABIBPGM/RTVJOBASQL (
    INOUT USRLIBL CHAR(2750) ,
    INOUT SYSLIBL CHAR(165) ,
    INOUT CURLIB CHAR(10)
)
    LANGUAGE CL
    SPECIFIC MABIBPGM/RTVJOBASQL
    NOT DETERMINISTIC
    NO SQL
    CALLED ON NULL INPUT
    EXTERNAL NAME 'MABIBPGM/RTVJOBALIB'
    PARAMETER STYLE GENERAL ;

LABEL ON SPECIFIC PROCEDURE MABIBPGM/RTVJOBASQL
      IS 'Récupération des listes de bibliothèques du job' ;
```

Exemples d'utilisation en SQL :

```
call MABIBPGM/RTVJOBASQL ('', '', '') ;
```

Si la procédure fonctionne correctement, son appel au travers de System i Navigator doit faire apparaître le contenu des 3 paramètres en sortie, contenant la "User Library List", la "System Library List" et la "Current Library". On constate que les "library list" sont récupérées sous forme de chaînes de caractères, ce qui est inutilisable d'un point de vue SQL. Il nous faut donc convertir ces chaînes de caractères en vraies listes SQL, travail que va effectuer l'UDTF ci-dessous.

Par exemple, pour récupérer la "User Library List" sous forme de véritable liste, au sens SQL du terme, on utilisera un RCTE (CTE récursive), selon le principe suivant :

```
WITH GEN_IDS(NX) AS (
    SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
UNION ALL
    SELECT NX+11 AS N2 FROM GEN_IDS WHERE NX < 2750
)
SELECT
    SUBSTR(USRLIBL, NX, 10) AS LIBLIST, 'USRLIBL' AS TYPLIB
FROM GEN_IDS ;
```

On utilisera le même principe pour obtenir la liste correspondant à la "System Library List".

Le code source complet de l'UDTF est le suivant :

```
-- UDTF développée par Grégory Jarrige le 2/06/2014
-- Fonction destinée à récupérer sous forme de liste SQL (soit une ligne par
-- bibliothèque), la liste des bibliothèques d'un travail IBMi
-- Le paramètre d'appel permet de filtrer le résultat en sortie, via les valeurs
-- suivantes :
--   'USRLIBL' pour obtenir exclusivement la liste des bibliothèques utilisateur
--   'SYSLIBL' pour obtenir exclusivement la liste des bibliothèques système
--   'CURLIB'  pour obtenir exclusivement la bibliothèque définie comme "CURLIB"
--   ''        pour obtenir toutes les bibliothèques dans l'ordre suivant
--               (bibliothèques utilisateurs, puis bibliothèques système, puis
curlib)
-- Exemples d'appel :
--  SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('') ) MYUDTF  ;
--  SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('USRLIBL') )
MYUDTF ;

CREATE OR REPLACE FUNCTION MABIBPGM.RTVLIBLIST(
    TYPLISTRET VARCHAR(10)
)
RETURNS TABLE (ORDRE INTEGER, LIBNAME CHAR(10), LIBTYPE CHAR(10))
LANGUAGE SQL
SPECIFIC MABIBPGM/RTVLIBLIST
NOT DETERMINISTIC
MODIFIES SQL DATA
BEGIN
    DECLARE USRLIBL CHAR(2750) DEFAULT '' ;
    DECLARE SYSLIBL CHAR(165) DEFAULT '' ;
    DECLARE CURLIB CHAR(10) DEFAULT '' ;
    DECLARE GLOBAL TEMPORARY TABLE TMPLIBLIST
       ( LIBNAME CHAR(10), LIBTYPE CHAR(10))
    WITH REPLACE ;
    CALL MABIBPGM.RTVJOBASQL ( USRLIBL , SYSLIBL , CURLIB ) ;

    -- USER LIBRARY LIST
    IF TYPLISTRET = '' OR TYPLISTRET = 'USRLIBL' THEN
        INSERT INTO SESSION.TMPLIBLIST (LIBNAME, LIBTYPE)
        WITH GEN_IDS(NX) AS (
            SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
        UNION ALL
            SELECT NX+11 AS N2 FROM GEN_IDS WHERE NX < 2750
        )
        SELECT
            SUBSTR(USRLIBL, NX, 10) AS LIBLIST, 'USRLIBL' AS TYPLIB
        FROM GEN_IDS ;
    END IF ;

    -- SYSTEM LIBRARY LIST
    IF TYPLISTRET = '' OR TYPLISTRET = 'SYSLIBL' THEN
        INSERT INTO SESSION.TMPLIBLIST (LIBNAME, LIBTYPE)
        WITH GEN_IDS(NX) AS (
            SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
```

```
        UNION ALL
            SELECT NX+11 AS N2 FROM GEN_IDS WHERE NX < 165
        )
        SELECT
        SUBSTR(SYSLIBL, NX, 10) AS LIBLIST, 'SYSLIBL' AS TYPLIB
        FROM GEN_IDS ;
    END IF ;

    -- CURLIB
    IF TYPLISTRET = '' OR TYPLISTRET = 'CURLIB' THEN
        IF (TRIM(CURLIB) <> '*NONE' AND TRIM(CURLIB)  <> '') THEN
            INSERT INTO SESSION.TMPLIBLIST (LIBNAME, LIBTYPE)
            SELECT CURLIB, 'CURLIB' FROM SYSIBM.SYSDUMMY1 ;
        END IF ;
    END IF ;

    RETURN
        WITH TMP AS (
            SELECT LIBNAME, LIBTYPE FROM SESSION.TMPLIBLIST WHERE TRIM(LIBNAME) <> ''
        )
        SELECT rrn(a) AS ORDRE, a.LIBNAME, a.LIBTYPE FROM TMP a ;
END ;

LABEL ON SPECIFIC FUNCTION MABIBPGM/RTVLIBLIST
      IS 'Récupération des listes de bibliothèques' ;
```

Comme indiqué dans le code source de l'UDTF, il est possible de vérifier son bon fonctionnement via les requêtes suivantes :

```
SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('') ) MYUDTF  ;
SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('USRLIBL') ) MYUDTF ;
SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('SYSLIBL') ) MYUDTF ;
SELECT ORDRE, LIBNAME, LIBTYPE FROM TABLE (MABIBPGM.RTVLIBLIST ('CURLIB') ) MYUDTF ;
```

Il ne nous reste plus qu'à développer l'UDTF finale RTVNOMCOURT. Elle s'appuiera sur l'UDTF RTVLIBLIST pour récupérer la liste des bibliothèques utilisateur, dans le cas où l'on a précisé "*LIBL" en paramètre d'appel. Si on a précisé une bibliothèque spécifique au lieu de "*LIBL" alors la recherche du nom court se fera sur cette bibliothèque exclusivement. A partir d'une bibliothèque, ou d'une liste de bibliothèques, on recherche dans la table QSYS2.SYSTABLES le nom court du (ou des) objet(s) portant le nom indiqué dans le première paramètre d'appel. Si plusieurs objets portent le même nom, c'est le premier (par rapport à l'ordre des bibliothèques) qui est retenu et renvoyé en sortie de l'UDTF RTVNOMCOURT :

```
CREATE OR REPLACE FUNCTION MABIBPGM.RTVNOMCOURT(
    NOM_LONG VARCHAR(256),
    BIBL VARCHAR(10)
)
RETURNS CHAR(21)
```

```
LANGUAGE SQL
SPECIFIC MABIBPGM/RTVNOMCOUR
NOT DETERMINISTIC
MODIFIES SQL DATA

BEGIN
   DECLARE GLOBAL TEMPORARY TABLE TMPLIBLIST2 (
        ORDRE INTEGER,
        LIBNAME CHAR(10)
   ) WITH REPLACE ;
   IF TRIM(BIBL) <> '' AND UPPER(TRIM(BIBL)) <> '*LIBL' THEN
        INSERT INTO SESSION.TMPLIBLIST2 (ORDRE, LIBNAME) (
            SELECT 1, UPPER(TRIM(BIBL)) FROM SYSIBM.SYSDUMMY1
        ) ;
   ELSE
        IF UPPER(TRIM(BIBL)) = '*LIBL' THEN
            INSERT INTO SESSION.TMPLIBLIST2 (ORDRE, LIBNAME) (
                SELECT ORDRE, LIBNAME FROM TABLE (MABIBPGM.RTVLIBLIST ('USRLIBL') )
MYUDTF
            ) ;
        END IF ;
   END IF ;

RETURN
    WITH TMPLIST AS (
        SELECT TRIM(a.SYSTEM_TABLE_SCHEMA) CONCAT '.' CONCAT
            TRIM(a.SYSTEM_TABLE_NAME) AS NOM_COURT,
            b.ORDRE AS ORDRE, a.SYSTEM_TABLE_SCHEMA , a.SYSTEM_TABLE_NAME
        FROM QSYS2.SYSTABLES a
        INNER JOIN SESSION.TMPLIBLIST2 b ON a.SYSTEM_TABLE_SCHEMA = b.LIBNAME
        WHERE a.TABLE_NAME = TRIM(NOM_LONG)
    ) ,
    TMPLIST2 AS (
        SELECT ORDRE, NOM_COURT FROM TMPLIST ORDER BY ORDRE
    ) ,
    TMPLIST3 AS (
        SELECT NOM_COURT FROM TMPLIST2 FETCH FIRST 1 ROW ONLY
    )
    SELECT NOM_COURT FROM TMPLIST3 ;
END ;

LABEL ON SPECIFIC FUNCTION MABIBPGM/RTVNOMCOUR
      IS 'Récupération du nom court d''un objet DB2' ;


-- Exemples d'appel :
SELECT MABIBPGM.RTVNOMCOURT('ARTICLE_VALORISE' , 'MABIBL1' ) FROM SYSIBM.SYSDUMMY1 ;
SELECT MABIBPGM.RTVNOMCOURT('ARTICLE_VALORISE' , '' ) FROM SYSIBM.SYSDUMMY1 ;
SELECT MABIBPGM.RTVNOMCOURT('ARTICLE_VALORISE' , '*LIBL' ) FROM SYSIBM.SYSDUMMY1 ;
```

Nous allons exploiter cet outil dans le chapitre qui suit et qui s'intitule "CLRPFM sur nom long".

5.5 CLRPFM sur nom long

La V7R2 du système d'exploitation IBMi va s'accompagner de l'ordre SQL TRUNCATE, qui sera strictement équivalent au CLRPFM en termes de performances, mais offrira l'avantage de travailler avec le nom long des tables à vider.

Pour les systèmes antérieurs à la V7R2, en revanche, le vidage de table peut se faire avec DELETE (qui peut dans certains cas se révéler lent, voire très lent), ou le CLRPFM, nettement plus rapide, mais qui a l'énorme défaut de n'accepter que le nom court des tables SQL à vider. Or ce nom court n'est pas forcément connu, et surtout il n'est pas forcément homogène d'un système à l'autre (le risque est alors grand de vider une table qui n'est pas celle souhaitée).

Pour éviter ce problème, on peut créer une procédure stockée offrant un CLRPFM amélioré, capable d'aller recherche le nom court associé au nom long de la table à vider. Pour récupérer le nom court de l'objet considéré, on s'appuiera sur le travail effectué au chapitre précédent (cf. chapitre "Retrouver le nom court d'un objet").

Voici le code source de la procédure stockée :

```
CREATE OR REPLACE PROCEDURE MABIBPGM/CLRPFM_NOMLONG (
    IN NOM_LONG VARCHAR(256) ,
    IN BIBL VARCHAR(10),
    INOUT CODRET CHAR(10) DEFAULT ''
)
LANGUAGE SQL
SPECIFIC MABIBPGM/CLRPFMLONG
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT

BEGIN
    DECLARE V_NOM_COURT CHAR(21) ;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
    -- Possibilité de tracer les anomalies (non implémenté)
    SET CODRET = 'ER' ;
    -- SET V_SQL_STATE = SQLSTATE ;
    -- SET V_SQL_CODE = SQLCODE ;
    -- GET DIAGNOSTICS EXCEPTION 1
    END ;

    SET CODRET = '' ;

    SELECT MABIBPGM.RTVNOMCOURT(NOM_LONG , BIBL ) INTO V_NOM_COURT FROM
SYSIBM.SYSDUMMY1 ;
```

```
    IF V_NOM_COURT IS NOT NULL AND TRIM(V_NOM_COURT) <> '' THEN
        SET V_NOM_COURT = REPLACE(V_NOM_COURT, '.', '/') ;
        CALL QCMDEXC ('CLRPFM ' CONCAT V_NOM_COURT) ;
        SET CODRET = 'OK' ;
    END IF ;

END ;

COMMENT ON SPECIFIC PROCEDURE MABIBPGM/CLRPFMLONG
    IS 'CLRPRM sur nom long de table DB2' ;


-- Exemples d'appel :
CALL MABIBPGM.CLRPFM_NOMLONG ( 'DIM_ARTICLE_GJA_01' ,  'MY_LIBRARY' ) ;
CALL MABIBPGM.CLRPFM_NOMLONG ( 'DIM_ARTICLE_GJA_01' ,  '*LIBL' ) ;
```

5.6 ChkObj en SQL

Il peut être utile de diposer, en SQL, d'une fonction permettant de vérifier la présence d'un objet IBMi dans une bibliothèque, et ce quel que soit le type d'objet considéré.

Pour disposer de ce type de fonction en SQL, il nous faut créer un CL et 2 fonctions SQL :

```
/* Source du CLP MABIBPGM/DWHSRC MBR(CHKOBJCLP) :                    */
/* Paramètres en entrée :                                            */
/*  Nom objet                      10                                */
/*  Nom biblliothèque              10                                */
/*  Type d'objet                   10                                */
/* Paramètres en sortie :                                            */
/*  Trouve (1=oui ; 0=non)          1                                */

PGM        PARM(&OBJET &BIBL &TYPE &TROUVE)
DCL        VAR(&OBJET) TYPE(*CHAR) LEN(10)
DCL        VAR(&BIBL) TYPE(*CHAR) LEN(10)
DCL        VAR(&TYPE) TYPE(*CHAR) LEN(10)
DCL        VAR(&TROUVE) TYPE(*CHAR) LEN(1)

/* Positionnement à "trouvé" par défaut   */
CHGVAR     VAR(&TROUVE) VALUE('1')

/* Vérification présence programme   */
CHKOBJ     OBJ(&BIBL/&OBJET) OBJTYPE(&TYPE)
MONMSG     MSGID(CPF9801) EXEC(DO)
CHGVAR     VAR(&TROUVE) VALUE('0')
ENDDO
ENDPGM
```

```
-- Fonction externe encapsulant le CL CHKOBJCLP pour vérification de
--  l'existence d'un objet (dont le type IBMi est transmis en 3ème paramètre)

CREATE OR REPLACE FUNCTION MABIBPGM/CHKOBJSQF (
    OBJET CHAR(10) ,
    BIBL CHAR(10) ,
    TYPOBJ CHAR(10)
)
RETURNS CHAR(1)
LANGUAGE CL
SPECIFIC MABIBPGM/CHKOBJSQF
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
EXTERNAL NAME 'MABIBPGM/CHKOBJCLP'
PARAMETER STYLE SQL ;
```

```
LABEL ON SPECIFIC FUNCTION MABIBPGM/CHKOBJSQF
       IS 'Contrôle existence d''un objet IBMi' ;


-- Fonction SQL destinée à faciliter l'appel de la fonction CHKOBJSQF,
--  par l'utilisation de paramètre en entrée de type VARCHAR, alors que la
--  fonction CHKOBJSQF ne tolère que le type CHAR, du fait qu'il s'agit d'une
--  fonction externe encapsulant un CLP

CREATE OR REPLACE FUNCTION MABIBPGM/CHKOBJSQL (
    OBJET VARCHAR(10) ,
    BIBL VARCHAR(10) ,
    TYPOBJ VARCHAR(10)
)
RETURNS CHAR(1)
LANGUAGE SQL
SPECIFIC MABIBPGM/CHKOBJSQL
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
SET OPTION  ALWBLK = *ALLREAD ,
ALWCPYDTA = *OPTIMIZE ,
COMMIT = *NONE ,
DECRESULT = (31, 31, 00) ,
DFTRDBCOL = *NONE ,
DYNDFTCOL = *NO ,
DYNUSRPRF = *USER ,
SRTSEQ = *HEX
BEGIN
    DECLARE TOBJET CHAR ( 10 ) ;
    DECLARE TBIBL CHAR ( 10 ) ;
    DECLARE TTYPOBJ CHAR ( 10 ) ;
    SET TOBJET = OBJET ;
    SET TBIBL = BIBL ;
    SET TTYPOBJ = TYPOBJ ;

    RETURN
        WITH TMP AS (
            SELECT CHKOBJSQF ( TOBJET , TBIBL , TTYPOBJ ) AS TROUVE FROM SYSIBM /
SYSDUMMY1
        )
        SELECT TROUVE FROM TMP ;
END ;

LABEL ON SPECIFIC FUNCTION MABIBPGM/CHKOBJSQL
       IS 'Contrôle existence d''un objet IBMi' ;
```

A noter : on fera appel à la fonction CHKOBJSQL dans le chapitre suivant (Contrôle de Jobd).

5.7 Contrôle de Jobd

Les objets IBMi de type Jobd (Job Description) sont très pratiques, et donc très utilisés. Rattachés à des profils utilisateurs et/ou à des travaux IBMi (batchs ou interactifs), ils permettent de définir la liste des bibliothèques que chaque travail est habilité à utiliser. D'un point de vue SQL, le fait d'utiliser la convention d'appellation système (*SYS) permet de s'appuyer sur cette notion de liste de bibliothèque. On peut dès lors s'affranchir de la notion de bibliothèque dans les programmes de type SQLRPGLE, ainsi que dans les procédures stockées SQL. Dans ce cas de figure, si un objet DB2 sur lequel on exécute une requête SQL n'a pas de bibliothèque définie, DB2 recherchera dans la liste des bibliothèques le premier objet portant le nom de l'objet considéré.

Ce mécanisme est très pratique, mais les Jobd ne sont pas des objets SQL, on ne peut donc pas exécuter de requête SQL permettant - par exemple - de retrouver toutes les Jobd utilisant une bibliothèque particulière. Si on souhaite renommer ou supprimer une bibliothèque, et si l'on a beaucoup de Jobd à contrôler, on court le risque de rendre certaines Jobd inopérantes à cause d'une bibliothèque manquante, et de voir certains traitements se planter lors de leur exécution.

En s'appuyant sur une API IBM, et un peu de CL et de SQL, on peut pallier ce manque, et développer un outil de contrôle fiable et performant.

5.7.1 API QWDRJOBD en RPG

On a besoin dans un premier temps d'un programme permettant d'extraire la liste des bibliothèques d'une jobd. Ce programme sera écrit en RPG en utilisant l'API IBM QWDRJOBD. Le source du programme RPG sera donné à la fin de ce chapitre.

Ce programme sera appelé au travers d'un programme CL, lui même associé à une UDF externe RTVJOBDSQF. Cette fonction sera appelée par l'UDTF RTVJOBDLIBL, cette dernière étant en mesure de renvoyer la liste des bibliothèques de la Jobd sous forme de liste SQL, en s'appuyant sur le mécanisme de la récursivité que nous avons vue dans le chapitre "Retrouver le nom court d'un objet".

```
/* Source du CL MABIBPGM/QCLSRC MBR(RTVJOBDSQC) qui fait appel au   */
/* programme RPGLE RTVJOBDPGM pour l'extraction de la liste des     */
/* bibliothèques                                                    */
PGM         PARM(&JOBDNAME &JOBDLIB &LIBL)

DCL         VAR(&JOBDNAME) TYPE(*CHAR) LEN(10)
DCL         VAR(&JOBDLIB) TYPE(*CHAR) LEN(10)
DCL         VAR(&VAR) TYPE(*CHAR) LEN(2750)
DCL         VAR(&LIBL) TYPE(*CHAR) LEN(2750)
DCL         VAR(&LIBLCNT) TYPE(*INT) LEN(2)
DCL         VAR(&JOBDL) TYPE(*CHAR) LEN(20)
CHGVAR      VAR(&JOBDL) VALUE(&JOBDNAME *CAT &JOBDLIB)

CALL        PGM(RTVJOBDPGM) PARM(&JOBDL &LIBL +
              &LIBLCNT)

ENDPGM
```

```
-- Fonction externe encapsulant le CL RTVJOBDSQC pour appel du programme
--  RPGLE RTVJOBDPGM (ce dernier faisant appel à l'API QWDRJOBD

CREATE OR REPLACE FUNCTION MABIBPGM/RTVJOBDSQF (
       JOBDNAME CHAR(10) ,
       JOBDLIB CHAR(10) )
       RETURNS CHAR(2750)
       LANGUAGE CL
       SPECIFIC MABIBPGM/RTVJOBDSQF
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'MABIBPGM/RTVJOBDSQC'
       PARAMETER STYLE SQL ;

LABEL ON SPECIFIC FUNCTION MABIBPGM/RTVJOBDSQF
       IS 'Extraction liste des bibl. d''une Jobd' ;
```

```
-- Fonction permettant d'extraire la liste des bibliothèques d'une Jobd sous
--  forme d'une table temporaire (ou table fonction).

CREATE OR REPLACE FUNCTION MABIBPGM/RTVJOBDLIBL (
    JOBDNAME VARCHAR(10) ,
    JOBDLIB VARCHAR(10) )
    RETURNS TABLE (
    JOBDNOUT CHAR(10) ,
    JOBDLOUT CHAR(10) ,
    LIBNAME CHAR(10)
)
LANGUAGE SQL
SPECIFIC MABIBPGM/RTVJBDLIBL
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
SET OPTION  ALWBLK = *ALLREAD ,
ALWCPYDTA = *OPTIMIZE ,
COMMIT = *NONE ,
DECRESULT = (31, 31, 00) ,
DFTRDBCOL = *NONE ,
DYNDFTCOL = *NO ,
DYNUSRPRF = *USER ,
SRTSEQ = *HEX
BEGIN

DECLARE LIBL CHAR ( 2750 ) DEFAULT '' ;
DECLARE JOBDNTMP CHAR ( 10 ) ;
DECLARE JOBDLTMP CHAR ( 10 ) ;

-- Passage des paramètre de type VARCHAR en CHAR pour faciliter l'utilisation
--  de la fonction RTVJOBDSQF qui ne tolère pas les VARCHAR
SET JOBDNTMP = JOBDNAME ;
SET JOBDLTMP = JOBDLIB ;

-- Table temporaire pour extraire la liste des bibliothèques
DECLARE GLOBAL TEMPORARY TABLE TMPLIBLIST
( LIBNAME CHAR ( 10 ) )
WITH REPLACE ;

-- Récupération de la liste des bibliothèques de la jobd au format CHAR ( 2750 )
SET LIBL = ( SELECT RTVJOBDSQF ( JOBDNTMP , JOBDLTMP ) FROM SYSIBM / SYSDUMMY1 ) ;

-- Utilisation de la récursivité pour extraire la liste des bibliothèques
INSERT INTO SESSION / TMPLIBLIST ( LIBNAME )
WITH GEN_IDS ( NX ) AS (
    SELECT 1 AS N1 FROM SYSIBM / SYSDUMMY1
UNION ALL
    SELECT NX + 11 AS N2 FROM GEN_IDS WHERE NX < 2750
)
```

```
SELECT
SUBSTR ( LIBL , NX , 10 ) AS LIBLIST
FROM GEN_IDS ;

RETURN
    WITH TMP AS (
        SELECT JOBDNAME , JOBDLIB , LIBNAME FROM SESSION / TMPLIBLIST
    )
    SELECT JOBDNAME , JOBDLIB , LIBNAME FROM TMP ;
END ;

LABEL ON SPECIFIC FUNCTION MABIBPGM/RTVJBDLIBL
      IS 'Extraction liste des bibl. d''une Jobd' ;
```

Source du programme RPG, écrit par Robert Cozzi en 2006, et partiellement réécrit en RPG Free par l'auteur de ce cours :

```
     H DFTACTGRP(*NO) OPTION(*SRCSTMT : *NODEBUGIO)
     H COPYRIGHT('(c) 2006 - Robert Cozzi, Jr. - All rights reserved.')

      ************************************************************
      **  RTVJOBD - Retrieve Job Description Command Proc Pgm.
      **            This program returns the library list
      **            of the specified Job Description ("jobd").
      **            In addition, the number of library names
      **            in the jobd's library lis is also returned.
      **            See the associated RTVJOBD CMD source for use
      **            in CL.  TIP: The return variables in your CL
      **            program should be defined as follows:
      **               DCL &LIBL    TYPE(*CHAR) LEN(2750)
      **               DCL &LIBLCNT TYPE(*INT)  LEN(2)
      **  Partially rewritten in RPG Free by Gregory Jarrige (2014-06-04)

     D RtvJobD         PR
     D  szJobD                       20A   Const
     D  rtnLIBL                    2750A
     D  rtnLIBLCount                  5I 0

     D RtvJobD         PI
     D  szJobD                       20A   Const
     D  rtnLIBL                    2750A
     D  rtnLIBLCount                  5I 0

      /COPY QSYSINC/QRPGLESRC,QWDRJOBD
      /COPY QSYSINC/QRPGLESRC,QUSEC

      **  Retrieve Job Description
     D*QWDRJOBD        PR                  ExtPgm('QWDRJOBD')
     D RtvJobDAPI      PR                  ExtPgm('QWDRJOBD')
     D  szRtnBuffer               65535A   OPTIONS(*VARSIZE)
     D  nRtnBufLen                   10I 0 Const
```

```
D**  Specify 'JOBD0100'
D  apiFormat                     8A   Const
D  JobD                         20A   Const
D  api_error                          LikeDS(QUSEC)

D JobD            DS                  LikeDS(QWDD0100)
D                                     Based(pJobD)

D JobDInfo        DS                  LikeDS(QWDD0100)

D LibList         S             11A   Based(pLIBL) DIM(250)
D LibL            S           2750A   Based(pLIBL)
D APIErrDS        DS                  LikeDS(QUSEC)
 /free
      *INLR = *ON ;

       //  Sadly, with this API, we need to call it twice when
       //  the LIBL is needed.
       //  First call: Get the length of the data to be returned.
       APIErrDS = *ALLX'00' ;
       APIErrDS.QUSBPRV = %size(APIErrDS) ;
       JobDInfo = *ALLX'00' ;
       callp RtvJobDAPI(JobDInfo : %size(JobDInfo):
               'JOBD0100': szJOBD : APIErrDS) ;

       if APIErrDS.QUSBAVL = 0 ;
           pJobD = %Alloc(JobDInfo.QWDBAVL) ;
           JOBD = *ALLX'00' ;
           //  Second call: Get the library list.
           callp RtvJobDAPI(JOBD : JobDInfo.QWDBAVL :
                   'JOBD0100': szJOBD : APIErrDS) ;
           if %Parms >= 3 ;
               rtnLIBLCount = JobD.QWDNLILL ;
           endif ;
           if %Parms >= 2 ;
               pLibl = pJobD + JobD.QWDOILL;
               rtnLibl = %subst(LIBL:1:JobD.QWDNLILL*%size(LibList));
           endif ;
           deAlloc pJobD ;
       endif ;
 /end-free
```

5.7.2 Contrôle de toutes les Jobd

Il nous reste à écrire une UDTF, que nous appellerons CHKALLJOBD. Cette UDTF effectuera un DSPJOBD *ALL vers une table temporaire, ce qui nous permettra ensuite de lire le contenu de cette table temporaire, et pour chaque occurrence de la table, d'appeler notre fonction RTVJOBDLIBL permettant d'extraire la liste des bibliothèques déclarée au sein de la Jobd. Pour chacune des bibliothèques appartenant à la liste obtenue, nous ferons appel à la fonction CHKOBJSQL pour vérifier si elle existe bien. Dans le cas contraire, elle fera partie des données renvoyées dans le result set produit par l'UDTF CHKALLJOBD.

L'utilisation de l'UDTF se fera au moyen de la requête suivante :

```
select * from table ( MABIBPGM.CHKALLJOBD  () ) myudtf ;
```

Voici le code soure de la fonction CHKALLJOBD :

```
-- Fonction destinée à contrôler la liste des bibliothèques pour l'ensemble
-- des Jobd d'un système

CREATE OR REPLACE FUNCTION MABIBPGM/CHKALLJOBD ( )
       RETURNS TABLE (
       JOBDNOUT CHAR(10) ,
       JOBDLOUT CHAR(10) ,
       LIBNAME CHAR(10) )
       LANGUAGE SQL
       SPECIFIC MABIBPGM/CHKALLJOBD
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       SET OPTION  ALWBLK = *ALLREAD ,
       ALWCPYDTA = *OPTIMIZE ,
       COMMIT = *NONE ,
       DECRESULT = (31, 31, 00) ,
       DFTRDBCOL = *NONE ,
       DYNDFTCOL = *NO ,
       DYNUSRPRF = *USER ,
       SRTSEQ = *HEX
       BEGIN
DECLARE SQLSTATE CHAR ( 5 ) DEFAULT '00000' ;
DECLARE VSQL VARCHAR ( 256 ) ;
DECLARE VCMD VARCHAR ( 256 ) ;
DECLARE VBIBL CHAR ( 10 ) ;
DECLARE VOBJET CHAR ( 10 ) ;

DECLARE CUR1 CURSOR FOR V_DYNSTM ;
DECLARE GLOBAL TEMPORARY TABLE TMPLIBLIS3 (
    JOBDN CHAR ( 10 ) ,
```

```
    JOBDL CHAR ( 10 ) ,
    LIBNAME CHAR ( 10 ) ,
    TROUVE CHAR ( 1 )
) WITH REPLACE ;

SET VCMD = 'DSPOBJD OBJ(*ALL/*ALL) OBJTYPE(*JOBD) OUTPUT(*OUTFILE)
OUTFILE(QTEMP/TMPJOBDLST) OUTMBR(*FIRST *REPLACE)' ;
CALL QCMDEXC ( VCMD , LENGTH ( VCMD ) ) ;

SET VSQL = 'SELECT ODLBNM as bibl, ODOBNM as objet FROM QTEMP.TMPJOBDLST' ;
PREPARE V_DYNSTM FROM VSQL ;

OPEN CUR1 ;
FETCH CUR1 INTO VBIBL , VOBJET ;

WHILE ( SQLSTATE = '00000' ) DO

    INSERT INTO SESSION / TMPLIBLIS3 ( JOBDN , JOBDL , LIBNAME , TROUVE )
    WITH TMPLIST AS (
        SELECT * FROM TABLE ( RTVJOBDLIBL ( VOBJET , VBIBL ) ) MYUDTF
    )
    SELECT JOBDNOUT , JOBDLOUT , LIBNAME ,
            CHKOBJSQL ( LIBNAME , '*LIBL' , '*LIB' ) AS TROUVE
    FROM TMPLIST WHERE TRIM ( LIBNAME ) <> '' AND SUBSTR ( LIBNAME , 1 , 1 ) <> '*' ;

    FETCH CUR1 INTO VBIBL , VOBJET ;

END WHILE ;

CLOSE CUR1 ;

DELETE FROM SESSION / TMPLIBLIS3 WHERE TROUVE = '1' ;

RETURN
    WITH TMP AS (
        SELECT JOBDN , JOBDL , LIBNAME FROM SESSION / TMPLIBLIS3
    )
    SELECT JOBDN , JOBDL , LIBNAME FROM TMP ;
END
;

LABEL ON SPECIFIC FUNCTION MABIBPGM/CHKALLJOBD
      IS 'Contrôle liste de bibl. / toutes les Jobd' ;
```

5.8 WrkObjLck version SQL

Ce chapitre propose une utilisation en RPG de l'API QWCLOBJL, API qui permet de récupérer la liste des travaux verrouillant un objet DB2.

Cet exemple de programme RPG est particulièrement intéressant par le fait qu'il produit un "result set" SQL en sortie, "result set" qui peut être exploité par le programme appelant, qu'il s'agisse d'un script PHP ou d'un autre langage (PL/SQL, Java, RPG, etc...). La partie du code RPG définissant le "result set" est indiquée en rouge dans le code source du programme RPG ci-dessous.

A noter que cet exemple a fait l'objet d'une publication sur foothing.net :

Exemple de procédure stockée "externe" encapsulant le programme RPG :

```
CREATE OR REPLACE PROCEDURE votre_librarie/APIOBJLCKP (
        IN OBJNAME CHAR(10) ,
        IN OBJLIB CHAR(10) ,
        IN OBJTYPE CHAR(10) ,
        IN MEMBER CHAR(10) ,
       IN RESULTSET CHAR(3) )
       DYNAMIC RESULT SETS 1
        LANGUAGE RPGLE
        SPECIFIC votre_librarie/APIOBJLCKP
        NOT DETERMINISTIC
        MODIFIES SQL DATA
        CALLED ON NULL INPUT
        EXTERNAL NAME 'votre_librarie/APIOBJLCK'
        PARAMETER STYLE SQL ;

COMMENT ON SPECIFIC PROCEDURE votre_librarie/APIOBJLCKP
        IS 'API QWCLOBJL -> QTEMP/OBJL0100 ou Result Set' ;
```

Code source du programme RPG appelé par la procédure stockée :

```
     ****************************************************************
     * Exemple d'utilisation de l'API QWCLOBJL (équivalent de WRKOBJLCK)
     * Adaptation d'un exemple fourni par Scott Klement sur :
     *  http://archive.midrange.com/rpg400-l/200202/msg00344.html
     * Adapté par Gregory Jarrige le 22/04/2013
     * Modifications par rapport à la version de Scott Klement :
     * - réécriture du code en RPG Free
     * - génération de la liste des "locks" dans une table temporaire DB2
     * - renvoi optionnel de la liste des "locks" en result set
     * - attention : ne surtout pas utiliser les directives de compilation
     *   proposées par Scott Klement (DFTACTGRP(*NO) ACTGRP(*NEW)), car
     *   elles ont pour effet d'empêcher le renvoi du "result set" vers
     *   la procédure stockée DB2 qui encapsule le programme RPG.
```

```
 *****************************************************************
H usrprf(*owner) datfmt(*iso)
D CrtUsrSpc       PR                  ExtPgm('QUSCRTUS')
D   UsrSpc                      20A   CONST
D   ExtAttr                     10A   CONST
D   InitSize                    10I 0 CONST
D   InitVal                      1A   CONST
D   PublicAuth                  10A   CONST
D   Text                        50A   CONST
D   Replace                     10A   CONST
D   ErrorCode                 32766A   options(*varsize)

D RtvPtrUS        PR                  ExtPgm('QUSPTRUS')
D   UsrSpc                      20A   CONST
D   Pointer                       *

D LstObjLck       PR                  ExtPgm('QWCLOBJL')
D   UsrSpc                      20A   const
D   Format                       8A   const
D   Object                      20A   const
D   ObjType                     10A   const
D   Member                      10A   const
D   ErrorCode                 32766A   options(*varsize)

D*************************************************
D* API error code data structure
D*************************************************
D dsEC            DS
D*                                    Bytes Provided (size of struct)
D  dsECBytesP             1      4I 0 INZ(256)
D*                                    Bytes Available (returned by API)
D  dsECBytesA             5      8I 0 INZ(0)
D*                                    Msg ID of Error Msg Returned
D  dsECMsgID              9     15
D*                                    Reserved
D  dsECReserv            16     16
D*                                    Msg Data of Error Msg Returned
D  dsECMsgDta            17    256

D*************************************************
D* List API generic header data structure
D*************************************************
D dsLH            DS                   BASED(p_UsrSpc)
D*                                     Filler
D   dsLHFill1                  103A
D*                                     Status (I=Incomplete,C=Complete
D*                                             F=Partially Complete)
D   dsLHStatus                   1A
D*                                     Filler
D   dsLHFill2                   12A
D*                                     Header Offset
D   dsLHHdrOff                  10I 0
D*                                     Header Size
D   dsLHHdrSiz                  10I 0
D*                                     List Offset
D   dsLHLstOff                  10I 0
D*                                     List Size
D   dsLHLstSiz                  10I 0
D*                                     Count of Entries in List
D   dsLHEntCnt                  10I 0
D*                                     Size of a single entry
```

```
     D   dsLHEntSiz                 10I 0

     D*************************************************
     D*  List Object Locks API format OBJL0100
     D*************************************************
     D* http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Fapis
%2Fqwclobjl.htm
     D dsOL            DS                  based(p_Entry)
     D*                                     Job Name
     D  dsOL_JobName                10A
     D*                                     Job User Name
     D  dsOL_UserName               10A
     D*                                     Job Number
     D  dsOL_JobNbr                  6A
     D*                                     Lock State
     D  dsOL_LckState               10A
     D*                                     Lock Status
     D  dsOL_LckSts                 10i 0
     D*                                     Lock Type
     D  dsOL_LckType                10i 0
     D*                                     Member (or *BLANK)
     D  dsOL_Member                 10A
     D*                                     1=Shared File, 0=Not Shared
     D  dsOL_Share                   1A
     D*                                     Lock Scope
     D  dsOL_LckScope                1A
     D*                                     Thread identifier
     D  dsOL_ThreadID                8A

     D p_UsrSpc        S               *
     D p_Entry         S               *
     D Msg             S              50A
     D MsgLockStatus   S              10A
     D MsgLockType     S              11A
     D MsgShare        S               4A
     D MsgScope        S              10A
     D Sql1            S             450A
     D Sql2            S             100A
     D Sql3            S             450A
     D x               S              10I 0

     C     *entry        plist
     C                   parm                    ObjName          10
     C                   parm                    ObjLib           10
     C                   parm                    ObjType          10
     C                   parm                    Member           10
     C                   parm                    Resultset         3

      /free
          If ( %parms < 5 ) ;
             *inlr = *on ;
          Endif ;

          // ****************************************
          // Create a user space to store output of
          // the list object locks API
          // ****************************************
          callp     CrtUsrSpc('OBJLOCKS  QTEMP': 'USRSPC':
                    1: x'00': '*ALL': 'Output of List ' +
                    'Object Locks API': '*YES': dsEC) ;
          If ( dsECBytesA > 0 ) ;
```

```
    *inlr = *on ;
Endif ;

// ****************************************
// Dump the Object Locks to the user space
// ****************************************
callp     LstObjLck('OBJLOCKS  QTEMP': 'OBJL0100':
          ObjName+ObjLib: ObjType: Member: dsEC) ;

If ( dsECBytesA > 0 ) ;
   *inlr = *on ;
Endif ;

// ****************************************
// Get a pointer to the user space
// ****************************************

callp     RtvPtrUS('OBJLOCKS  QTEMP': p_UsrSpc) ;

//*********************************************
// Création d'une table DB2 temporaire destinée
// à stocker les entrées de la liste renvoyée
// par l'API
//*********************************************
sql1 = 'declare global temporary table OBJL0100 ( ' +
       'Job_name        CHAR(10) , ' +
       'Job_user_name   CHAR(10) , ' +
       'Job_number      CHAR(6)  , ' +
       'Lock_state      CHAR(10) , ' +
       'Lock_status     CHAR(10) , ' +
       'Lock_type       CHAR(11) , ' +
       'Member_name     CHAR(10) , ' +
       'Share           CHAR(4)  , ' +
       'Lock_scope      CHAR(10) , ' +
       'Thread_id       CHAR(8) ' +
       ') with replace ' ;

EXEC SQL EXECUTE IMMEDIATE :sql1 ;

sql2 = 'INSERT INTO QTEMP/OBJL0100 ' +
      'VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)' ;

EXEC SQL PREPARE cur1 FROM :sql2 ;

for x = 0 to (dsLHEntCnt-1) ;
      p_Entry = p_UsrSpc + (dsLHLstOff + (dsLHEntSiz*x));

      MsgLockStatus = '' ;
      select ;
      when      dsOL_LckSts = 1 ;
          MsgLockStatus = 'HELD' ;
      when      dsOL_LckSts = 2 ;
          MsgLockStatus = 'WAIT' ;
      when      dsOL_LckSts = 2 ;
          MsgLockStatus = 'REQ' ;
      endsl;

      MsgLockType = '' ;
      select ;
      when      dsOL_LckType = 1 ;
         MsgLockType = 'OBJECT';
```

```
            when      dsOL_LckType = 2 ;
               MsgLockType = 'MBR CTL BLK';
            when      dsOL_LckType = 3 ;
               MsgLockType = 'MBR ACC PTH';
            when      dsOL_LckType = 3 ;
               MsgLockType = 'MBR DATA' ;
            endsl;

            If ( dsOL_Share = '1' ) ;
               MsgShare = 'YES' ;
            Else ;
               MsgShare = 'NO' ;
            Endif ;

            If ( dsOL_LckScope = '1' ) ;
               MsgScope = 'THREAD' ;
            Else ;
               If ( dsOL_LckScope = '2' ) ;
                  MsgScope = 'LOCK SCAPE' ;
               Else ;
                  MsgScope = 'JOB' ;
               Endif ;
            Endif ;

            EXEC SQL EXECUTE cur1 USING :dsOL_JobName ,
               :dsOL_UserName , :dsOL_JobNbr , :dsOL_LckState ,
               :MsgLockStatus , :MsgLockType , :dsOL_Member ,
               :MsgShare , :MsgScope , :dsOL_ThreadID ;

      Endfor ;

      //*********************************************
      // Si demandé par le programme appelant,
      // génération d'un result set à partir de la
      // table temporaire
      //*********************************************
      If ( Resultset = 'YES' ) ;
        sql3 = 'SELECT distinct Job_name, Job_user_name, Job_number, ' +
                  'Lock_state, Lock_status, Lock_type, Member_name, ' +
                  'Share, Lock_scope ' +
                  'FROM QTEMP/OBJL0100 FOR FETCH ONLY ' ;
        EXEC SQL
          PREPARE REQ1 FROM :sql3 ;
        EXEC SQL
          DECLARE C1 CURSOR FOR REQ1 ;
        EXEC SQL
          OPEN C1 ;
        EXEC SQL
          SET RESULT SETS CURSOR C1 ;

      Endif ;

      *inlr = *on ;

  /end-free
```

Exemple d'appel de la procédure stockée en PHP (avec l'extension ibm_db2) :

```
$conn = db2_connect('*LOCAL', $user, $password);

$sql = 'CALL votre_librarie/APIOBJLCKP (?, ?, ?, ?, ?)';
$stmt = db2_prepare($conn, $sql);

$objname = "ARTFRS";
$objlib  = "GJARRIGE3";
$objtype = "*FILE";
$member  = "ARTFRS";
$resulset = "YES";

db2_bind_param($stmt, 1, "objname", DB2_PARAM_IN);
db2_bind_param($stmt, 2, "objlib", DB2_PARAM_IN);
db2_bind_param($stmt, 3, "objtype", DB2_PARAM_IN);
db2_bind_param($stmt, 4, "member", DB2_PARAM_IN);
db2_bind_param($stmt, 5, "resulset", DB2_PARAM_IN);

if (db2_execute($stmt)) {

      while ($row = db2_fetch_assoc($stmt)) {
            echo $row['JOB_NAME'], '|' , $row['MEMBER_NAME'], '|' ,
$row['LOCK_STATE'] , PHP_EOL ;
      }
}
```

A noter : l'extension "ibm_db2" offre la possibilité d'exploiter plusieurs result sets produits par le composant appelé (cf. exemple sur php.net) :

http://us1.php.net/manual/fr/function.db2-next-result.php

5.9 WrkJobScde version SQL

Le programme RPG présenté dans ce chapitre est une réécriture en RPG Free d'un programme proposé par David Malle pour le site foothing.net.

Le but de ce programme est d'extraire, en utilisant une API IBM, l'intégralité des lignes définies dans le planning de travaux de l'IBMi (traditionnellement accessible au travers de la commande WRKJOBSCDE).

J'ai choisi d'intégrer ce programme dans le cadre de ce cours SQL, car c'est un programme de type SQLRPGLE, qui utilise de nombreuses techniques SQL, et constitue de ce fait un bon outil pédagogique.

Il faut souligner que cette version génère en sortie un "result set" SQL, qui peut être réutilisé par le programme appelant. On recommandera d'encapsuler ce programme dans une procédure stockée externe, de manière à pouvoir l'exploiter directement via un CALL SQL (cf. le chapitre précédent pour un exemple de procédure stockée externe).

Le code source du programme est abondamment documenté :

```
 *-------------------------------------------------------------
 *  Ecrit par David Malle le 21/10/2004 pour foothing.net
 *  Reecriture par Gregory Jarrige le 9/06/2014
 *  avec quelques amenagements :
 *  - reecriture des cartes C en RPG Free
 *  - utilisation de prototypes sur l'appel des API,
 *  - possibilite de recuperer un result set en sortie (mode par defaut)
 *  - prise en compte de l'integralite des donnees renvoyees par l'API
 *    (la version initiale n'en traitait qu'une partie)
 *  -------------------------------------------------------------
 *  Objet : Lecture des informations du scheduler
 *          pour insertion dans le fichier RTVSCDE dans QTEMP
 *  -------------------------------------------------------------
 *  Type  : SQLRPGLE
 *  Compil: CRTSQLRPGI OBJ(BIBOBJ/RTVSCDE)
 *                     SRCFILE(BIBSRC/QRPGLESRC)
 *                     SRCMBR(RTVSCDEPGM)
 *                     OBJTYPE(*PGM)
 *  -------------------------------------------------------------
 *  Comment appeler le programme
 *  Appel : CALL RTVSCDE PARM('N') => cree un fichier QTEMP/RTVSCDE
 *  Appel : CALL RTVSCDE PARM('O') => genere un result set SQL
 *  Appel : CALL RTVSCDE           => idem appel precedent
 *-------------------------------------------------------------
H option(*noxref:*nodebugio)
```

```
 *
 * Retrieve des infos du scheduler
 *
 * Les information sont trouvees dans l'objet via l'API QWCLSCDE
 * QDFTJOBSCD  *JOBSCD   QUSRSYS
 *
 *API error data structure.
DErrorDS          DS           116
D BytesProvd              1      4B 0
D BytesAvail              5      8B 0
D MessageId               9     15A
D Errn                   16     16A
D MessageDta             17    116A
 *
D SqlCur          S            900A
D SqlIns          S            250A
D CurrentEnt      S              5P 0
 *
 * Structure externe correspondant a l'API QWCLSCDE
DSCDL0100         ds
D dsINFSTA                       1A
D dsJOBNAM                      10A
D dsENTNUM                      10A
D dsSCHDAT                      10A
D dsSCHDAY                      70A
D dsSCHTIM                       6A
D dsFREQUE                      10A
D dsFRELDY                      50A
D dsFRECAC                      10A
D dsFNEXSB                      10A
D dsSTATUS                      10A
D dsJBQUNM                      10A
D dsJBQLNM                      10A
D dsUSPREA                      10A
D dsLASSBD                      10A
D dsLASSBT                       6A
D dsTEXT                        50A
D dsRES1                        23A
D dsJQSTATUS                    10A
D dsDATOMI                     200A
D dsJDNAME                      10A
D dsJDLIBRARY                   10A
D dsUPRSBM                      10A
D dsMSQNAME                     10A
D dsMSQLIBR                     10A
D dsSAVENTRY                    10A
D dsLSBMJNAM                    10A
D dsLSBMJUSR                    10A
D dsLSBMJNUM                     6A
D dsLATTSBMD                    10A
D dsSATTSBMT                     6A
D dsTATTSTAT                    10A
D dsRESVDTA                      2A
```

```
D dsLENCMD                       10I 0
D dsCMDSTRG                     512A
 *
 * Nom du USERSPACE
DInputDS          DS                  Inz
D UserSpace                      20
D  SpaceName                     10    OverLay(UserSpace:1)
D                                      Inz('RTVSCDE   ')
D  SpaceLib                      10    OverLay(UserSpace:11)
D                                      Inz('QTEMP     ')
 *
 * Parametres pour gestion du USERSPACE
DFormatName       S                8
DFormType         S               10
DUserData         S               10
DUserName         S               10    Inz('*ALL')
DJobName          S               10    Inz('*ALL')
DHdlContinue      S               16    Inz('    ')
 *
 *
DSpaceSize        S                9B 0 inz(65536)
DSpaceAttr        S               10
DSpaceValue       S                1
DSpaceAuth        S               10    Inz('*CHANGE')
DSpaceText        S               50
DSpaceReplc       S               10    Inz('*YES')
 *
 * DS pour premiere lecture du USERSPACE
DGeneralDS        DS             140    Inz
D InputSize              113    116B 0
D ListOffset             125    128B 0
D NumberOfList           133    136B 0
D EntrySize              137    140B 0
 *
 * Variables pour positionnement pour les lectures du USERSPACE
D StartPosit      S                9B 0
D StartLen        S                9B 0
 *
 * Gestion des parametres en entree du programme.
DResultSetOn      S                 N   inz
DCreateTab        S               1
 ****************************************************
 * Create User Space (QUSCRTUS) API
 ****************************************************
D QUSCRTUS        PR                  ExtPgm('QUSCRTUS')
D   UserSpace                    20A   const
D   Attrib                       10A   const
D   InitSize                     10I 0 const
D   InitVal                       1A   const
D   PubAuth                      10A   const
D   Text                         50A   const
 * optional group 1:
D   Replace                      10A   const options(*nopass)
```

```
D   ErrorCode                 8000A   options(*varsize: *nopass)
 * optional group 2:
D   Domain                      10A   const options(*nopass)
 * optional group 3:
D   XferSizeReq                 10I 0 const options(*nopass)
D   OptAlign                     1A   const options(*nopass)
 ****************************************************
 * List Job Schedule Entries (QWCLSCDE) API
 ****************************************************
D QWCLSCDE        PR                  ExtPgm('QWCLSCDE')
D   UserSpace                   20A   const
D   FormatName                   8A   const
D   JobName                     10A   const
D   HdlContinue                 16A   const
D   ErrorCode                 8000A   options(*varsize: *nopass)
 ****************************************************
 * Retrieve User Space (QUSRTVUS) API
 ****************************************************
D QUSRTVUS        PR                  ExtPgm('QUSRTVUS')
D   UserSpace                   20A   const
D   StartPosit                   9B 0 const
D   StartLen                     9B 0 const
D   GeneralDS                 8000A   options(*varsize)
D   ErrorCode                 8000A   options(*varsize: *nopass)
 ****************************************************
 * Delete User Space (QUSDLTUS) API
 ****************************************************
D QUSDLTUS        PR                  ExtPgm('QUSDLTUS')
D   UserSpace                   20A   const
D   ErrorCode                 8000A   options(*varsize: *nopass)
 ****************************************************
 * Reception parametres du programme
 ****************************************************
 * Reception parametres
C     *ENTRY        PLIST
C                   PARM                    CreateTab
 /FREE
       // -------------------------------------------------------
       // generation d'un result set par defaut, sauf si le 1er parametre
       //   est renseigne et different de 'O'
       // -------------------------------------------------------
       ResultSetOn = *on ;
       IF %PARMS = 1;
          IF CreateTab <> 'O' ;
            ResultSetOn = *off ;
          ENDIF ;
       ENDIF ;
       // ---------------> Soit le traitement de lecture du Scheduler
       exsr      SchedulerT ;
       // ---------------> Fin du programme
       *inlr = *on ;
       // -------------------------------------------------------
       // Lecture des infos du scheduler
```

```
// ---------------------------------------------------
begsr SchedulerT ;
    exsr CommitS ;
    exsr TableC ;
    exsr UserSpaceS ;
    exsr UserSpaceC ;
    exsr SchedulerR ;
    exsr UserSpace1 ;
    IF  NumberOfList > 0 ;
       CurrentEnt = 1 ;
       DoW CurrentEnt <= NumberOfList ;
         exsr UserSpace2 ;
         CurrentEnt += 1 ;
       Enddo ;
    Endif ;
    exsr UserSpaceS ;
    //*****************************************
    // Si demande par le programme appelant,
    // generation d'un result set a partir de la
    // table temporaire
    //*****************************************
    If ( ResultSetOn = *on ) ;
      exsr GenResultSet ;
    EndIF ;
endsr ;
// ---------------------------------------------------
// Deconn. du commitment control pour eviter les pbms de journalisation
//  du fichier ds QT
// ---------------------------------------------------
begsr CommitS ;
    Exec SQL
    Set option commit = *none ;
endsr ;
// ---------------------------------------------------
// Creation de la table RTVSCDE dans QTEMP
// ---------------------------------------------------
begsr TableC ;
    Exec SQL
    DECLARE GLOBAL TEMPORARY TABLE RTVSCDE (
      RTINFSTA CHAR(1) CCSID 297 NOT NULL DEFAULT '' ,
      RTJOBNAM CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTENTNUM CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTSCHDAT CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTSCHDAY CHAR(70) CCSID 297 NOT NULL DEFAULT '' ,
      RTSCHTIM CHAR(6) CCSID 297 NOT NULL DEFAULT '' ,
      RTFREQUE CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTFRELDY CHAR(50) CCSID 297 NOT NULL DEFAULT '' ,
      RTFRECAC CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTFNEXSB CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTSTATUS CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTJBQUNM CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTJBQLNM CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTUSPREA CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
```

```
      RTLASSBD CHAR(10) CCSID 297 NOT NULL DEFAULT '' ,
      RTLASSBT CHAR(6) CCSID 297 NOT NULL DEFAULT '' ,
      RTTEXT CHAR(50) CCSID 297 NOT NULL DEFAULT '' ,
      RTRES1 CHAR(23) CCSID 297 NOT NULL DEFAULT '' ,
      JQSTATUS CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      DATOMI CHAR(200)  CCSID 297 NOT NULL DEFAULT '',
      JDNAME CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      JDLIBRARY CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      UPRSBM CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      MSQNAME CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      MSQLIBR CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      SAVENTRY CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      LSTSBMJNAM CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      LSTSBMJUSR CHAR(10)  CCSID 297 NOT NULL DEFAULT '' ,
      LSTSBMJNUM CHAR(6)  CCSID 297 NOT NULL DEFAULT '' ,
      LSTATTSBMD CHAR(10)  CCSID 297 NOT NULL DEFAULT '',
      LSTATTSBMT CHAR(6)  CCSID 297 NOT NULL DEFAULT '' ,
      LSTATTSTAT CHAR(10)  CCSID 297 NOT NULL DEFAULT '',
      RESERVDTA CHAR(2)  CCSID 297 NOT NULL DEFAULT '' ,
      CMDSTRING CHAR(512)  CCSID 297 NOT NULL DEFAULT ''
    ) WITH REPLACE ;

    // Preparation de la requete d'insertion
    sqlIns = 'insert into QTEMP/RTVSCDE ' +
         ' values (? , ? , ? , ? , ? , ? , ? , ? , ? , ' +
         ' ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ?,' +
         ' ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? , ? ) ' ;

  Exec SQL
    PREPARE REQINS FROM :sqlIns ;

endsr ;
// ------------------------------------------------------
// Supression du USERSPACE
// ------------------------------------------------------
begsr UserSpaceS ;
    callP QUSDLTUS ( UserSpace : ErrorDS ) ;
endsr ;
// ------------------------------------------------------
// Creation du USERSPACE
// ------------------------------------------------------
begsr UserSpaceC ;
    CallP QUSCRTUS ( UserSpace : *blanks : SpaceSize :
        *blanks : '*ALL' : *blanks : '*YES' : ErrorDS) ;
endsr ;
// ---------------------------------------
// Lecture des infos du scheduler dans le USERSPACE
// ---------------------------------------
begsr SchedulerR ;
    CallP QWCLSCDE ( UserSpace:'SCDL0200':JobName:
        HdlContinue : ErrorDS ) ;
endsr ;
// ---------------------------------------
```

```
// Lecture 1er poste du USERSPACE
// -------------------------------------
begsr UserSpace1 ;
    CallP QUSRTVUS (UserSpace:1:140:GeneralDS:ErrorDS) ;

    StartPosit = (ListOffset + 1) ;
    StartLen = EntrySize ;
endsr ;
// -------------------------------------
// Lecture de tous les postes du USERSPACE
// -------------------------------------
begsr UserSpace2 ;
    CallP QUSRTVUS ( UserSpace : StartPosit : StartLen :
        SCDL0100 : ErrorDS ) ;

    StartPosit = (StartPosit + EntrySize) ;
    // ------------------------------------------------------------
    // Insertion des donnees du scheduler dans le fichier RTVSCDE
    // (sauf la longueur de la commande qui, avec son format
    // binaire, posait des problemes de recuperation en SQL)
    // ------------------------------------------------------------
    Exec SQL
     EXECUTE REQINS USING :SCDL0100.dsINFSTA ,
                          :SCDL0100.dsJOBNAM ,
                          :SCDL0100.dsENTNUM ,
                          :SCDL0100.dsSCHDAT ,
                          :SCDL0100.dsSCHDAY ,
                          :SCDL0100.dsSCHTIM ,
                          :SCDL0100.dsFREQUE ,
                          :SCDL0100.dsFRELDY ,
                          :SCDL0100.dsFRECAC ,
                          :SCDL0100.dsFNEXSB ,
                          :SCDL0100.dsSTATUS ,
                          :SCDL0100.dsJBQUNM ,
                          :SCDL0100.dsJBQLNM ,
                          :SCDL0100.dsUSPREA ,
                          :SCDL0100.dsLASSBD ,
                          :SCDL0100.dsLASSBT ,
                          :SCDL0100.dsTEXT ,
                          :SCDL0100.dsRES1 ,
                          :SCDL0100.dsJQSTATUS ,
                          :SCDL0100.dsDATOMI ,
                          :SCDL0100.dsJDNAME ,
                          :SCDL0100.dsJDLIBRARY ,
                          :SCDL0100.dsUPRSBM ,
                          :SCDL0100.dsMSQNAME ,
                          :SCDL0100.dsMSQLIBR ,
                          :SCDL0100.dsSAVENTRY ,
                          :SCDL0100.dsLSBMJNAM ,
                          :SCDL0100.dsLSBMJUSR ,
                          :SCDL0100.dsLSBMJNUM ,
                          :SCDL0100.dsLATTSBMD ,
                          :SCDL0100.dsSATTSBMT ,
```

```
                            :SCDL0100.dsTATTSTAT ,
                            :SCDL0100.dsRESVDTA ,
                            :SCDL0100.dsCMDSTRG ;
endsr ;
 // ---------------------------------------
 // Generation d'un result set en sortie
 // certaines colonnes sont "nettoyees" car
 // elles contiennent des caracteres parasites
 // renvoyes par l'API
 // ---------------------------------------
 begsr GenResultSet ;
   sqlCur = 'select ' +
    'RTINFSTA, RTJOBNAM, RTENTNUM, RTSCHDAT, RTSCHDAY, ' +
    'RTSCHTIM, RTFREQUE, RTFRELDY, RTFRECAC, ' +
    'replace(RTFNEXSB, x''00'', '''') as RTFNEXSB, ' +
    'RTSTATUS, RTJBQUNM, RTJBQLNM, RTUSPREA, ' +
    'replace(RTLASSBD, x''00'', '''') as RTLASSBD, ' +
    'replace(RTLASSBT, x''00'', '''') as RTLASSBT, ' +
    'RTTEXT, RTRES1, JQSTATUS, ' +
    'replace(datomi, x''00'', '''') as datomi, ' +
    'JDNAME, JDLIBRARY, UPRSBM, MSQNAME, MSQLIBR, SAVENTRY, ' +
    'LSTSBMJNAM, LSTSBMJUSR, LSTSBMJNUM, ' +
    'replace(LSTATTSBMD, x''00'', '''') as LSTATTSBMD, ' +
    'replace(LSTATTSBMT, x''00'', '''') as LSTATTSBMT, ' +
    'LSTATTSTAT, RESERVDTA, CMDSTRING ' +
    'from qtemp/rtvscde ' ;

   Exec SQL
     PREPARE REQCUR FROM :sqlCur ;
   EXEC SQL
     DECLARE CUR1 CURSOR FOR REQCUR ;
   EXEC SQL
     OPEN CUR1 ;
   EXEC SQL
     SET RESULT SETS CURSOR CUR1 ;
endsr ;
```

5.10 Comparaison de procédures stockées

La TR7 (Technology Refresh 7) qui accompagne la V7R1 contient une procédure stockée intéressante, CHECK_SYSROUTINE, qui est fournie gracieusement par IBM, à titre expérimental.

Cette procédure permet de comparer une procédure stockée se trouvant sur le système courant, par rapport à une procédure stockée de même nom se trouvant dans la même bibliothèque mais sur un serveur distant.

La version spécifique proposée dans ce chapitre offre plus de souplesse que la version IBM car elle permet de préciser des noms de bibliothèques différents et des serveurs différents (pouvant tous deux être différents du serveur d'exécution de la procédure).

Exemples d'utilisation :
CALL SYSTOOLS.CHECK_SYSROUTINE2('TEST', 'MABIBREF' , 'PREPROD', 'MABIBREF', 0);
CALL SYSTOOLS.CHECK_SYSROUTINE2('PREPROD', 'MABIBREF1', 'TEST', 'MABIBREF2' , 0);

```
CREATE OR REPLACE PROCEDURE SYSTOOLS.CHECK_SYSROUTINE2 (
    IN P_DB_NAME1 VARCHAR(10) ,
    IN P_SCHEMA_NAME1 VARCHAR(128) ,
    IN P_DB_NAME2 VARCHAR(10) ,
    IN P_SCHEMA_NAME2 VARCHAR(128) ,
    IN P_AVOID_RESULT_SET INTEGER DEFAULT  0
)
DYNAMIC RESULT SETS 1
LANGUAGE SQL
SPECIFIC SYSTOOLS.CHECK_SYSROUTINE2
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
SET OPTION  ALWBLK = *ALLREAD ,
ALWCPYDTA = *OPTIMIZE ,
COMMIT = *NONE ,
DECRESULT = (31, 31, 00) ,
DFTRDBCOL = QSYS2 ,
DLYPRP = *NO ,
DYNDFTCOL = *NO ,
DYNUSRPRF = *OWNER ,
SRTSEQ = *HEX
BEGIN
-- PROCEDURE SYSTOOLS.CHECK_SYSROUTINE2
--
-- Created on December 4, 2012 by Scott Forstie (forstie@us.ibm.com)
-- Adaptée par Grégory Jarrige le 4/04/2014

-- Cette version spécifique offre plus de souplesse que la version IBM car
```

```
--  elle permet de préciser des noms de bibliothèques différents et des
--  serveurs différents (pouvant tous deux être de type "remote").
-- Exemple :
-- CALL SYSTOOLS.CHECK_SYSROUTINE2('TEST', 'M3FEHSAP' , 'PREPROD', 'MABIBPGM', 0);
-- CALL SYSTOOLS.CHECK_SYSROUTINE2('PREPROD', 'MABIBPGM', 'TEST', 'M3FEHSAP' , 0);

--
-- Dependency:
--
-- This procedure will fail as shown below if the remote-system-name isn't configured
for *IP
-- in the RDB Directory Entry.  Use the WRKRDBDIRE command to review the RDB
configuration.
--
-- Example RDB setup: ADDRDBDIRE RDB(<remote-system-name>) RMTLOCNAME(<remote-system-
name> *IP)
--
DECLARE ROW_COUNT INTEGER ;
DECLARE SQLCODE INTEGER ;
DECLARE SQLSTATE CHAR ( 5 ) ;
DECLARE LOCAL_SQLCODE INTEGER ;
DECLARE LOCAL_SQLSTATE CHAR ( 5 ) ;
DECLARE V_STMT1 VARCHAR ( 1000 ) ;
DECLARE V_STMT2 VARCHAR ( 1000 ) ;
DECLARE PREPARE_ATTRIBUTES VARCHAR ( 1000 ) ;
DECLARE V_STEP_NUM INTEGER DEFAULT 0;
DECLARE V_TMP_DBNAME VARCHAR(11) ;

DECLARE C_SYSROUTINE_DIFF CURSOR FOR
SELECT ROUTINE_NAME,
A_SERVER_NAME , A_ROUTINE_CREATED , A_ROUTINE_DEFINER , A_LAST_ALTERED ,
A_SPECIFIC_SCHEMA , A_SPECIFIC_NAME , A_ROUTINE_SCHEMA ,
A_ROUTINE_NAME , A_ROUTINE_TYPE , A_ROUTINE_BODY , A_EXTERNAL_NAME , A_IN_PARMS ,
A_OUT_PARMS , A_INOUT_PARMS , HEX ( A_PARM_SIGNATURE ) AS PARM_SIGNATURE ,
B_SERVER_NAME , B_ROUTINE_CREATED , B_ROUTINE_DEFINER , B_LAST_ALTERED ,
B_SPECIFIC_SCHEMA , B_SPECIFIC_NAME , B_ROUTINE_SCHEMA ,
B_ROUTINE_NAME , B_ROUTINE_TYPE , B_ROUTINE_BODY , B_EXTERNAL_NAME , B_IN_PARMS ,
B_OUT_PARMS , B_INOUT_PARMS , HEX ( B_PARM_SIGNATURE ) AS PARM_SIGNATURE ,
CASE ERROR_CODE
    WHEN '1' THEN 'Attributs différents'
    WHEN '2' THEN 'Objet absent sur ' CONCAT TRIM(P_DB_NAME2)
    WHEN '3' THEN 'Objet absent sur ' CONCAT TRIM(P_DB_NAME1)
END as ERROR
FROM SESSION . SYSRTNDIFF
ORDER BY 1
;

DECLARE EXIT HANDLER FOR SQLEXCEPTION
GENERAL_HANDLER : BEGIN
  DECLARE FAILURE_TEXT VARCHAR ( 1000 ) ;
  GET DIAGNOSTICS CONDITION 1
    LOCAL_SQLCODE = DB2_RETURNED_SQLCODE ,
    LOCAL_SQLSTATE = RETURNED_SQLSTATE;
```

```
  SET FAILURE_TEXT =
         'CHECK_SYSROUTINE2 Failure: SQLSTATE=' CONCAT LOCAL_SQLSTATE CONCAT
',SQLCODE=' CONCAT
         RTRIM ( CHAR ( LOCAL_SQLCODE ) ) CONCAT ',STEP_NUM=' CONCAT CHAR(V_STEP_NUM)
;
  RESIGNAL SQLSTATE LOCAL_SQLSTATE SET MESSAGE_TEXT = FAILURE_TEXT ;

END GENERAL_HANDLER ;


SET V_STEP_NUM = V_STEP_NUM + 1 ;

DECLARE GLOBAL TEMPORARY TABLE SESSION . SYSRTNDIFF (
ROUTINE_NAME VARCHAR ( 128 ) ALLOCATE ( 18 ) DEFAULT NULL , -- For Final ORDER
A_SERVER_NAME VARCHAR ( 18 ) DEFAULT NULL ,
A_ROUTINE_CREATED TIMESTAMP DEFAULT NULL ,
A_ROUTINE_DEFINER VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL,
A_LAST_ALTERED TIMESTAMP DEFAULT NULL,
A_SPECIFIC_SCHEMA VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL ,
A_SPECIFIC_NAME VARCHAR ( 128 ) ALLOCATE ( 18 ) DEFAULT NULL ,
A_ROUTINE_SCHEMA VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL ,
A_ROUTINE_NAME VARCHAR ( 128 ) ALLOCATE ( 18 ) DEFAULT NULL ,
A_ROUTINE_TYPE VARCHAR ( 9 ) DEFAULT NULL ,
A_ROUTINE_BODY VARCHAR ( 8 ) DEFAULT NULL ,
A_EXTERNAL_NAME VARCHAR ( 279 ) ALLOCATE ( 21 ) DEFAULT NULL ,
A_IN_PARMS SMALLINT DEFAULT NULL ,
A_OUT_PARMS SMALLINT DEFAULT NULL ,
A_INOUT_PARMS SMALLINT DEFAULT NULL ,
A_SQL_DATA_ACCESS VARCHAR ( 8 ) DEFAULT NULL,
A_PARM_SIGNATURE VARCHAR ( 2048 ) ALLOCATE ( 50 ) DEFAULT NULL,
B_SERVER_NAME VARCHAR ( 18 ) DEFAULT NULL ,
B_ROUTINE_CREATED TIMESTAMP DEFAULT NULL,
B_ROUTINE_DEFINER VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL,
B_LAST_ALTERED TIMESTAMP DEFAULT NULL,
B_SPECIFIC_SCHEMA VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL ,
B_SPECIFIC_NAME VARCHAR ( 128 ) ALLOCATE ( 18 ) DEFAULT NULL ,
B_ROUTINE_SCHEMA VARCHAR ( 128 ) ALLOCATE ( 10 ) DEFAULT NULL ,
B_ROUTINE_NAME VARCHAR ( 128 ) ALLOCATE ( 18 ) DEFAULT NULL ,
B_ROUTINE_TYPE VARCHAR ( 9 ) DEFAULT NULL ,
B_ROUTINE_BODY VARCHAR ( 8 ) DEFAULT NULL ,
B_EXTERNAL_NAME VARCHAR ( 279 ) ALLOCATE ( 21 ) DEFAULT NULL ,
B_IN_PARMS SMALLINT DEFAULT NULL ,
B_OUT_PARMS SMALLINT DEFAULT NULL ,
B_INOUT_PARMS SMALLINT DEFAULT NULL ,
B_SQL_DATA_ACCESS VARCHAR ( 8 ) DEFAULT NULL,
B_PARM_SIGNATURE VARCHAR ( 2048 ) ALLOCATE ( 50 ) DEFAULT NULL,
ERROR_CODE CHAR(1)
) WITH REPLACE ON COMMIT PRESERVE ROWS ;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

DECLARE GLOBAL TEMPORARY TABLE SESSION . CHECKIT1 AS (
SELECT CURRENT SERVER as SERVER_DEF, ROUTINE_CREATED , ROUTINE_DEFINER , LAST_ALTERED
```

```
, SPECIFIC_SCHEMA ,
SPECIFIC_NAME , ROUTINE_SCHEMA , ROUTINE_NAME , ROUTINE_TYPE ,
ROUTINE_BODY , EXTERNAL_NAME , IN_PARMS ,
OUT_PARMS , INOUT_PARMS , SQL_DATA_ACCESS , PARM_SIGNATURE , ROUTINE_DEFINITION
FROM QSYS2 . SYSROUTINE ) WITH NO DATA
WITH REPLACE ON COMMIT PRESERVE ROWS ;


SET V_STEP_NUM = V_STEP_NUM + 1 ;

DECLARE GLOBAL TEMPORARY TABLE SESSION . CHECKIT2 AS (
SELECT * FROM SESSION . CHECKIT1 ) WITH NO DATA
WITH REPLACE ON COMMIT PRESERVE ROWS ;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

SET PREPARE_ATTRIBUTES = ' FOR READ ONLY OPTIMIZE   FOR ALL ROWS   WITH NC ' ;

IF P_DB_NAME2 = CURRENT SERVER THEN
    SET V_TMP_DBNAME = '' ;
ELSE
    SET V_TMP_DBNAME = TRIM(P_DB_NAME2) CONCAT '.' ;
END IF ;

SET V_STMT2 = ' INSERT INTO SESSION.CHECKIT2
    SELECT ''' CONCAT TRIM(P_DB_NAME2) CONCAT ''' AS SERVER_2, ROUTINE_CREATED,
ROUTINE_DEFINER, LAST_ALTERED, SPECIFIC_SCHEMA, SPECIFIC_NAME,
    ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_BODY, EXTERNAL_NAME,
IN_PARMS,
    OUT_PARMS, INOUT_PARMS, SQL_DATA_ACCESS, PARM_SIGNATURE, ROUTINE_DEFINITION
    FROM ' CONCAT TRIM(V_TMP_DBNAME) CONCAT 'QSYS2.SYSROUTINE WHERE ROUTINE_SCHEMA =
''' concat (P_SCHEMA_NAME2) concat '''' ;
PREPARE PULL_REMOTE_DATA2 ATTRIBUTES PREPARE_ATTRIBUTES FROM V_STMT2 ;
EXECUTE PULL_REMOTE_DATA2  ;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

IF P_DB_NAME1 = CURRENT SERVER THEN
    SET V_TMP_DBNAME = '' ;
ELSE
    SET V_TMP_DBNAME = TRIM(P_DB_NAME1) CONCAT '.' ;
END IF ;

SET V_STMT1 = ' INSERT INTO SESSION.CHECKIT1
    SELECT ''' CONCAT TRIM(P_DB_NAME1) CONCAT ''' AS SERVER_1, ROUTINE_CREATED,
ROUTINE_DEFINER, LAST_ALTERED, SPECIFIC_SCHEMA, SPECIFIC_NAME,
    ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_BODY, EXTERNAL_NAME,
IN_PARMS,
    OUT_PARMS, INOUT_PARMS, SQL_DATA_ACCESS, PARM_SIGNATURE, ROUTINE_DEFINITION
    FROM ' CONCAT TRIM(V_TMP_DBNAME) CONCAT 'QSYS2.SYSROUTINE WHERE ROUTINE_SCHEMA =
''' concat (P_SCHEMA_NAME1) concat '''' ;

SET V_STEP_NUM = V_STEP_NUM + 1 ;
```

```
PREPARE PULL_REMOTE_DATA1 ATTRIBUTES PREPARE_ATTRIBUTES FROM V_STMT1 ;
--EXECUTE PULL_REMOTE_DATA1 USING P_SCHEMA_NAME1 ;
EXECUTE PULL_REMOTE_DATA1  ;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

INSERT INTO SESSION . SYSRTNDIFF

SELECT
A.ROUTINE_NAME as ROUTINE_NAME_ORDER,
A.SERVER_DEF , A.ROUTINE_CREATED , A.ROUTINE_DEFINER , A.LAST_ALTERED ,
A.SPECIFIC_SCHEMA , A.SPECIFIC_NAME , A.ROUTINE_SCHEMA , A.ROUTINE_NAME ,
A.ROUTINE_TYPE , A.ROUTINE_BODY , A.EXTERNAL_NAME , A.IN_PARMS , A.OUT_PARMS ,
A.INOUT_PARMS , A.SQL_DATA_ACCESS , A.PARM_SIGNATURE ,
B.SERVER_DEF , B . ROUTINE_CREATED , B . ROUTINE_DEFINER , B . LAST_ALTERED ,
B . SPECIFIC_SCHEMA , B . SPECIFIC_NAME , B . ROUTINE_SCHEMA , B . ROUTINE_NAME ,
B . ROUTINE_TYPE , B . ROUTINE_BODY , B . EXTERNAL_NAME , B . IN_PARMS , B .
OUT_PARMS ,
B . INOUT_PARMS , B . SQL_DATA_ACCESS , B . PARM_SIGNATURE ,
'1'
FROM SESSION . CHECKIT1 A
INNER JOIN SESSION.CHECKIT2 B
ON A . ROUTINE_NAME = B . ROUTINE_NAME
WHERE
      A . SPECIFIC_NAME <> B . SPECIFIC_NAME
 OR   A . ROUTINE_TYPE <> B . ROUTINE_TYPE
 OR   A . ROUTINE_BODY <> B . ROUTINE_BODY
 OR   A . IN_PARMS <> B . IN_PARMS
 OR   A . OUT_PARMS <> B . OUT_PARMS
 OR   A . INOUT_PARMS <> B . INOUT_PARMS
 OR   A . SQL_DATA_ACCESS <> B . SQL_DATA_ACCESS
 OR   A . PARM_SIGNATURE <> B . PARM_SIGNATURE
 OR  A. ROUTINE_DEFINITION <> B. ROUTINE_DEFINITION
;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

INSERT INTO SESSION . SYSRTNDIFF

SELECT
B.ROUTINE_NAME as ROUTINE_NAME_ORDER,
B.SERVER_DEF , B . ROUTINE_CREATED , B . ROUTINE_DEFINER , B . LAST_ALTERED ,
B . SPECIFIC_SCHEMA , B . SPECIFIC_NAME , B . ROUTINE_SCHEMA , B . ROUTINE_NAME ,
B . ROUTINE_TYPE , B . ROUTINE_BODY , B . EXTERNAL_NAME , B . IN_PARMS , B .
OUT_PARMS ,
B . INOUT_PARMS , B . SQL_DATA_ACCESS , B . PARM_SIGNATURE ,
 '', current timestamp, '', current timestamp, '', '', '', '', '', '', '', 0, 0, 0,
'', '' ,
'2'
FROM SESSION . CHECKIT1 A
RIGHT EXCEPTION JOIN SESSION.CHECKIT2 B
ON A . ROUTINE_NAME = B . ROUTINE_NAME
```

```
;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

INSERT INTO SESSION . SYSRTNDIFF

SELECT
A.ROUTINE_NAME as ROUTINE_NAME_ORDER,
'', current timestamp, '', current timestamp, '', '', '', '', '', '', '', 0, 0, 0,
'', '' ,
A.SERVER_DEF , A . ROUTINE_CREATED , A . ROUTINE_DEFINER , A . LAST_ALTERED ,
A . SPECIFIC_SCHEMA , A . SPECIFIC_NAME , A . ROUTINE_SCHEMA , A . ROUTINE_NAME ,
A . ROUTINE_TYPE , A . ROUTINE_BODY , A . EXTERNAL_NAME , A . IN_PARMS , A .
OUT_PARMS ,
A . INOUT_PARMS , A . SQL_DATA_ACCESS , A . PARM_SIGNATURE ,
'3'
FROM SESSION . CHECKIT1 A
LEFT EXCEPTION JOIN SESSION.CHECKIT2 B
ON A . ROUTINE_NAME = B . ROUTINE_NAME
;

SET V_STEP_NUM = V_STEP_NUM + 1 ;

SELECT COUNT ( * ) INTO ROW_COUNT FROM SESSION . SYSRTNDIFF ;
IF ROW_COUNT <> 0 THEN

SET V_STEP_NUM = V_STEP_NUM + 1 ;

-- If the caller wants to directly query
-- SESSION.SYSRTNDIFF, don't return a result set
IF P_AVOID_RESULT_SET = 0 THEN
-- Return the SYSROUTINE mismatches to the caller
OPEN C_SYSROUTINE_DIFF ;
END IF ;
END IF ;

END ;

COMMENT ON SPECIFIC PROCEDURE SYSTOOLS.CHECK_SYSROUTINE2
      IS 'Procédure de comparaison de procédures stockées entre 2 bases ' ;
```

6. Bibliographie

Le site Developerworks d'IBM propose d'excellents dossiers sur SQL DB2, et sur tout un tas d'autres sujets :

https://developer.ibm.com/

Deux très bons ouvrages (en anglais), très complémentaires, pour découvrir de nombreuses techniques SQL avancées. Ces deux ouvrages sont disponibles chez O'Reilly.com :

- *SQL Cookbook*, par Anthony Molinaro (couvre MySQL, SQL Server, Oracle, PostgreSQL et DB2)
- *SQL Hacks*, par Andrew Cumming et Gordon Russell (couvre MySQL, SQL Server, Oracle, PostgreSQL et Access)

Dans ces ouvrages, chaque technique est présentée dans différentes versions (une pour chaque SGBD). Contrairement au premier, le second ouvrage ne donne pas d'exemples pour DB2, mais de nombreuses techniques présentées pour MySQL peuvent être facilement adaptées à DB2.

On arrive à trouver ces deux ouvrages, dans des versions traduites en français, sur le marché de l'occasion (éventuellement en version numérique), sous les titres suivants :

- *SQL, le livre de recette*s (pour le livre de Anthony Molinaro), mais il a été édité aussi sous le titre « *SQL par l'exemple* ».
- *SQL à 200 %* (pour le livre Andrew Cumming et Gordon Russell)

Un autre très bon livre (en anglais) pour apprendre à reconnaître les mauvaises pratiques SQL, et différentes manières de les corriger :

- *SQL Antipatterns: Avoiding the Pitfalls of Database Programming*, par Bill Karwin https://pragprog.com/titles/bksqla/sql-antipatterns/