100 DB2/SQL & Enterprise COBOL Performance and Tuning TIPS

Author: Erol Technologies

Version: 1
Status: Validation
Date: octobre 2013

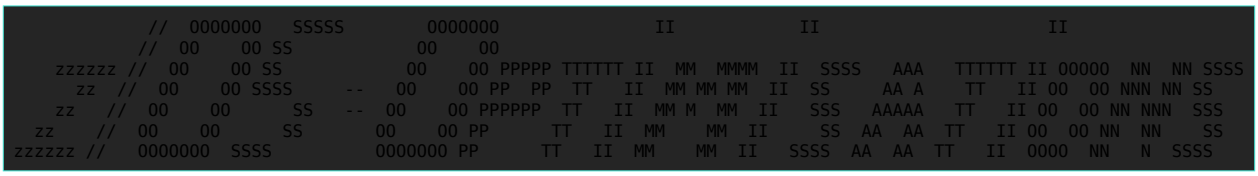
Contact: denis.wallerich@datim.fr

Dedication

For z/OS DB2 & Cobol developers.

Abstract

This document presents a set of rules and simple tips to improve the performance of Cobol/DB2 programs
It contains many examples of SQL queries and COBOL constructs and presents some advanced concepts of DB2 (V9 and V10)



Contents

1 SQL TIPS

1.1 Ne pas rapatrier plus de lignes et de colonnes que nécessaires

Il s'agit d'une première règle de base très importante

Toutes les colonnes présentes dans la `SELECT-liste` doivent être manipulées par DB2 avant d'être restituées au programme applicatif, monopolisant les ressources du système et alourdissant les tris.

En codant des prédicats appropriés, on doit limiter le nombre de lignes retournées au programme au strict minimum.

La présence de colonnes inutiles peut modifier le chemin d'accès dans une jointure ou empêcher un accès performant de type « IndexOnly »

1.2 Eviter le SELECT *

En règle générale, le SQL compilé ne devrait jamais comporter de `SELECT *`. Comme déjà indiqué précédemment, il ne faut accéder qu'aux données nécessaires au programme. En respectant cette règle, on obtient un maximum d'efficacité et de flexibilité dans l'écriture des requêtes :

- La présence de `SELECT *` augmente l'effort de maintenance dans les programmes en cas d'ajout, de modification ou de destruction de colonne
- La présence de `SELECT *` augmente la charge dans DB2 pour rapatrier et trier les colonnes
- Un `SELECT *` peut contraindre DB2 à ne pas pouvoir faire le choix d'un accès IndexOnly

> ***Note***
>
> De nombreux `SELECT *` figurent dans les exemples de ce document, simplement par soucis de concision

1.3 Ne pas lire les données déjà connues du programme

Cela peut paraître évident, mais de nombreux programmeurs violent cette règle un jour ou l'autre.

Ici, par exemple, le surcoût induit par la lecture inutile de la colonne `CRP_CODE` peut devenir significatif si la requête est exécutée de nombreuses fois:

```
.           Exec Sql
              SELECT
                CRP_CODE,
                CRP_NOM
              INTO
                :DCLVLWACRSP.CRP-CODE,
                :DCLVLWACRSP.CRP-NOM
              FROM VLWACRSP
              WHERE CRP_CODE = :W-CRP-CODE
            End-Exec
```

1.4 Coder explicitement les valeurs littérales

Quand c'est possible, il est préférable de coder explicitement les valeurs littérales dans les requêtes SQL plutôt que de passer par un `MOVE` dans une host variable intermédiaire. Cela permet à l'optimiseur DB2 de calculer un chemin d'accès optimal.

```
.      * Move 'EUR' To CURRENCY-CODE-REF Of DCLVLWBRCOURSDEV
         Exec Sql
            SELECT RATE
            INTO :DCLVLWBRCOURSDEV.RATE
            FROM VLWBRCOURSDEV
            WHERE
                TPS_DT_ARRT       = :DCLVLWBRCOURSDEV.TPS-DT-ARRT
            AND CURRENCY_CODE     = :DCLVLWBRCOURSDEV.CURRENCY-CODE
       *    AND CURRENCY_CODE_REF = :DCLVLWBRCOURSDEV.CURRENCY-CODE-REF
            AND CURRENCY_CODE_REF = 'EUR'
         End-exec
```

1.5 Déporter le maximum de la charge de travail dans SQL

C'est peut-être la règle la plus importante à connaître par les développeurs DB2 : laisser SQL se charger d'accéder et filtrer les données.

Il faut ainsi encourager le codage de jointures SQL plutôt que de joindre les données dans le programme applicatif à la suite de multiples accès dans DB2. Il s'agit de loin du plus gros problème chez beaucoup de programmeurs ayant l'habitude de coder des accès fichiers plutôt que des requêtes relationnelles.

Une jointure SQL surpasse toujours une jointure programmée pour diverses raisons :

- L'optimiseur DB2 génère un code plus efficace que celui écrit par la plupart des programmeurs.
- Les données peuvent êtres jointes et filtrées très tôt par DB2 avant d'être transmises au programme applicatif.
- Le langage SQL simplifie les tests et la détection de bugs : une jointure peut rapidement être testée sous SPUFI contrairement à de multiples requêtes qui seraient réparties dans un programme.

Par ailleurs, en filtrant au plus tôt les données dans DB2 par le codage de prédicats appropriés dans les clauses WHERE, on évite le surcoût occasionné par le rapatriement de données inutiles qui doivent ensuite encore être filtrées par le programme applicatif.

Il existe évidemment des contres-exemples à cette règle, en particulier avec l'utilisation de procédures stockées ou de fonctions DB2 (UDFs) pour lesquelles le coût peut se révéler plus important qu'avec un codage équivalent réalisé dans un langage de programmation classique.

1.6 Faire un minimum d'appels à DB2 (EXEC SQL)

Chaque `EXEC SQL` représente un coût fixe incompressible (communication cross-memory, thread DB2,...) Mais certaines techniques existent pour réduire le nombre d'appels à DB2:

- Le `SELECT` singleton (avec `FETCH FIRST 1 ROW ONLY`) permet de remplacer un curseur (3 `Exec SQL`)
- Le `MERGE` (ou Upsert = update/insert intégrés)
- Le `SELECT from INSERT`, `UPDATE`, `DELETE` ou `MERGE` (insert, update, merge ou delete + lecture simultanés d'un ligne)
- Le Multi-Row `FETCH` (MRF, avec la possibilité de faire `UPDATE` ou `DELETE` dans le curseur)

- Le Multi-Row `INSERT` (MRI)
- Le Multi-Row `MERGE` (MRM)

1.7 Préférer le singleton `SELECT` au curseur

Pour lire une seule ligne, un programme peut utiliser un curseur ou un `SELECT` singleton.

Un curseur nécessite un `OPEN`, un `FETCH` puis un `CLOSE`, pour ne retourner au final qu'un seule ligne.

En revanche, le singleton `SELECT` n'a besoin que d'un seul appel à DB2 `SELECT...INTO`.

Le singleton `SELECT` est toujours plus économique qu'un curseur, cela est d'autant plus vrai avec la clause `FETCH FIRST 1 ROW ONLY` qui assure toujours le retour d'une unique ligne et stoppe au plus tôt la recherche.

Historiquement, le problème du singleton `SELECT` provient de la gestion du `SQLCODE -811` lorsque le résultat comporte plusieurs lignes. Avec l'ajout de la clause `FETCH FIRST 1 ROW ONLY`, ce problème disparaît.

1.8 Eviter le Singleton `SELECT` quand des données doivent être modifiées

Lorsqu'une ligne doit être modifiée après lecture, il est recommandé d'utiliser un curseur avec la clause `FOR UPDATE OF` afin de garantir l'intégrité des données. DB2 maintient alors un verrou exclusif empêchant ainsi la ligne d'être modifiée par une autre tâche entre le moment de sa lecture et de sa mise à jour.

1.9 Utiliser `DISTINCT` judicieusement

Le DISTINCT élimine les doublons dans le résultat d'une requête. Si la présence de doublons ne pose pas de problème dans le programme applicatif, ne pas le coder, car si un index adéquat n'est pas disponible, il induit dans DB2 une charge supplémentaire de tri

> ***Note***
>
> Avec DB2V9 `DISTINCT` peut dorénavant utiliser un index "non unique" pour éliminer les doublons et éviter ainsi un tri (de la même façon que `GROUP BY`)

1.10 Utiliser `ORDER BY` seulement (mais toujours) si c'est indispensable

L'ordre dans lequel les lignes sont retournées par DB2 n'ai jamais garanti sans la présence d'un `ORDER BY`. Il y a quelques fois confusion lorsque DB2 utilise un index pour satisfaire une requête et que le l'ordre des lignes retournées est conforme au résultat attendu sans avoir eut besoin de coder un `ORDER BY`. Cela n'est en fait que le reflet du choix effectué par l'optimiseur à un instant donné. Ce choix d'index pouvant changer à tout instant (après un rebind par exemple)

1.11 Limiter les colonnes spécifiées dans ORDER BY

Quand ORDER BY est utilisé, DB2 s'assure que les données soient triées en séquence pour chacune des colonne spécifiées. Pour ce faire DB2 invoque un tri (à moins qu'un index approprié existe). Plus le nombre de colonnes à trier est important moins performante sera la requête. Il ne faut donc spécifier dans le tri que les colonnes strictement nécessaires.

1.12 Eviter les tris internes provoqués par ORDER BY

En utilisant un index, DB2 ne déclenchera pas de tri supplémentaire pour satisfaire un ORDER BY. Pour cela, il faut que :

- les premières colonnes de l'index correspondent exactement à la séquence de tri spécifiée dans un ORDER BY
- ou que les premières colonnes de l'index possèdent un prédicat d'égalité dans la clause WHERE et que les suivantes correspondent exactement à la séquence de l'ORDER BY
- l'index est utilisé pour accéder à une seule table
- les premières colonnes de l'index portent sur la première table d'un jointure

Depuis peu, DB2 est capable de parcourir un index en sens inverse (REVERSE SCAN), permettant ainsi de satisfaire à la fois un ORDER BY ascendant et descendant. Pour cela il est nécessaire que la séquence de tri spécifiée dans l'ORDER BY soit en totale opposition à l'index.

Par exemple, l'index suivant:

```
.               CREATE INDEX Ix (C1 ASC, C2 DESC, C3 ASC)
```

est capable de prendre en charge ces 2 tris :

```
.              ORDER BY C1 ASC,  C2 DESC, C1 ASC

               ORDER BY C1 DESC, C2 ASC,  C1 DESC
```

1.13 Lire les données sur l'index CLUSTER pour les accès batch de masse.

Lorsque qu'une table doit être parcourue en totalité par un programme batch, il est fortement recommandé de l'accéder par l'index «cluster» afin de bénéficier du «Sequentiel Prefetch»

L'utilisation d'un ORDER BY sur une colonne avec un fort CLUSTERRATIO encourage DB2 à accéder aux données en "sequential prefetch"

1.14 Promouvoir les accès «Index Only » autant que possible

1.15 Favoriser les prédicats indexables et «Stage 1»

Pendant la phase de développement d'une requête il faut toujours s'interroger de savoir dans quel stage seront appliqués les prédicats : «Stage 1» ou «Stage 2».

Un prédicat qui peut être satisfait en «Stage 1» est évalué par la couche Data Manager de DB2, alors qu'un prédicat «Stage 2» est évalué par la couche Relational Data System de DB2. Le composant Data Manager de DB2 est plus proche des données physiques que le composant Relational Data System.

Un prédicat «Stage 1» est évalué à un niveau très proche des données physiques, il diminue donc le volume des données transmises vers la couche supérieure relationnelle de DB2.

Les prédicats «Stage 1» sont donc plus performants que les prédicats «Stage 2».

Note

Quand c'est le programme applicatif qui est chargé du filtrage des données retournées par DB2 on parle quelquefois de «Stage 3»

Un prédicat dit «indexable» ne garantie pas l'utilisation d'un index par DB2, cela signifie simplement qu'un index pourrait être utilisé pour satisfaire le prédicat.

En revanche, DB2 n'utilisera jamais un index pour satisfaire un prédicat non indexable

Classement des différents prédicats:

Type de prédicat	Indexable ?	Stage 1 ?
COL=value	Y	Y
COL=noncol expr	Y	Y
COL IS NULL	Y	Y
COL op value	Y	Y
COL op noncol expr	Y	Y
COL BETWEEN value1 AND value2	Y	Y
COL BETWEEN noncol expr1 AND noncol expr2	Y	Y
Value BETWEEN COL1 AND COL2	N	N
COL BETWEEN COL1 AND COL2	N	N
COL BETWEEN expression1 AND expression2	N	N
COL LIKE 'pattern'	Y	Y
COL IN(list)	Y	Y
COL <> value	N	Y
COL <> noncol expr	N	Y
COL IS NOT NULL	N	Y
COL NOT BETWEEN value1 AND value2	N	Y
COL NOT BETWEEN noncol expr1 AND noncol expr2	N	Y
Value NOT BETWEEN COL1 AND COL2	N	N
COL NOT IN(list)	N	Y
COL NOT LIKE 'char'	N	Y
COL LIKE '%char'	N	Y
COL LIKE '_char'	N	Y

COL LIKE hostvariable	Y	Y
T1.COL = T2.COL	Y	Y
T1.COL op T2.COL	Y	Y
T1.COL <> T2.COL	N	Y
T1.COL1 = T1.COL2	N	N
T1.COL1 op T1.COL2	N	N
T1.COL1 <> T1.COL2	N	N
COL = (nonsubq)	Y	Y
COL = ANY(nonsubq)	N	N
COL = ALL(nonsubq)	N	N
COL op (nonsubq)	Y	Y
COL op ANY(nonsubq)	Y	Y
COL op ALL(nonsubq)	Y	Y
COL <> (nonsubq)	N	Y
COL <> ANY(nonsubq)	N	N
COL <> ALL(nonsubq)	N	N
COL IN(nonsubq)	Y	Y
(COL1,...COLn) IN(nonsubq)	Y	Y
COL NOT IN(nonsubq)	N	N
(COL1,...COLn) NOT IN(nonsubq)	N	N
COL = (corsubq)	N	N
COL = ANY(corsubq)	N	N
COL = ALL(corsubq)	N	N
COL op(corsubq)	N	N
COL op ANY(corsubq)	N	N
COL op ALL(corsubq)	N	N
COL <> (corsubq)	N	N
COL <> ANY(corsubq)	N	N
COL <> ALL(corsubq)	N	N
COL IN(corsubq)	N	N
(COL1,...COLn) IN (corsubq)	N	N
COL NOT IN(corsubq)	N	N
(COL1,...COLn) NOT IN (corsubq)	N	N
EXISTS(subq)	N	N
NOT EXISTS(subq)	N	N
COL = expression	Y	Y
Expression = value	N	N
Expression <> value	N	N

Expression op value	N	N
Expression op(subquery)	N	N

Une “noncol expr” est une expression dans laquelle aucune colonne de table n'est spécifiée.

Exemples de "noncol expr" :

```
.               CURRENT TIMESTAMP – 10 DAYS
                :HOST + 100
                FLOAT(10.5)
```

Les prédicats Stage 1 combinés avec `AND`, `OR` ou précédés par `NOT` sont des prédicats «Stage 1». Tous les autres sont «Stage 2».

Les prédicats indexable combinés avec `AND` ou `OR` sont indexables, en revanche ils ne sont plus indexables s'ils sont précédés de `NOT`

> ***Note***
>
> Tous les prédicats indexables sont «Stage 1». L'inverse n'étant pas vrai : tous les prédicats «Stage 1» ne sont pas nécessairement indexables.
>
> Un prédicat bien que conforme syntaxiquement «Stage 1» peut se retrouver rétrogradé par DB2 en «Stage 2» (cela se produisait avant DB2 V8 par exemple lorsqu'un type ou une longueur de donnée ne correspondait pas)
>
> Les critères d'indexabilité et de Stage pour un type de prédicat donné peuvent changer d'une version de DB2 à l'autre

Certains prédicats «Stage 2» peuvent être réécrits en «Stage 1», et certains prédicats non-indexables peuvent être rendus indexables

L'indexabilité et l'évaluation d'un prédicat en «Stage 1» sont deux aspects importants concernant l'optimisation des requêtes, mais ne garantissent pas toujours forcément la meilleure performance possible

Il faut parfois s'autoriser l'écriture de prédicats «Stage 2», cela est toujours plus efficace que d'opérer un filtrage des données dans le programme applicatif («Stage 3»)

1.16 Reformuler les requêtes pour accéder au Stage 1

Tenter de réécrire un prédicat Stage 2 en Stage 1 est une excellente démarche, et cela est souvent rendu possible grâce à la flexibilité et aux nombreuses possibilités offertes par le langage SQL

Voici 2 exemples de prédicats Stage 2 transformés en Stage 1 :

```
.               1) WHERE YEAR(COL_DATE) = 2013

                   => WHERE COL_DATE BETWEEN '2013-01-01' AND '2013-12-31'

                2) WHERE SUBSTR(LASTNAME,1,1) = 'M'

                   => WHERE LASTNAME LIKE 'M%'
```

Prédicat non indexable rendu indexable:

```
.               WHERE COL NOT BETWEEN 'A' AND 'G'

                => WHERE COL >= 'H'
          OU    => WHERE COL BETWEEN 'H' AND 'Z'
```

Ce dernier cas suppose une bonne connaissance des données présentes en table afin de s'assurer que la colonne ne contiendra jamais une valeur `inférieure à 'A'`

1.17 Eviter les comparaisons entre colonnes d'une même table

Ce genre de prédicat et toujours «Stage 2»

1.18 Eviter l'utilisation de `NOT` (sauf avec `EXISTS`)

Les prédicats utilisant `NOT` ne sont pas indexables, donc, autant que possible, réécrire les requêtes en évitant `NOT` (^=, <>). En s'appuyant sur une bonne connaissance des domaines de valeurs accédés, il est souvent possible de transformer un `NOT` en inégalité.

Par exemple, si on est certain que la colonne `CODE` ne contient que des données alphanumériques :

```
.               WHERE CODE <> ' '
```

peut être réécrit ainsi :

```
.               WHERE CODE > ' '
```

1.19 Ne pas comparer des données de type et/ou longueur différentes

1.20 Utiliser des types de donnée compatibles

Utiliser des données de type et de longueur équivalents pour comparer des valeurs de colonne avec des host variables ou des littéraux, sinon une conversion doit être appliquée, risquant de rétrograder ainsi l'évaluation en STAGE 2 alors qu'il serait éligible en STAGE1 si les longueurs et les type correspondaient

Si DB2 doit opérer une conversion de type, les index disponibles deviennent inutilisables

On peut utiliser une fonction `CAST` afin de de faire correspondre des types de données dépareillés.

Par exemple, pour faire correspondre une colonne définie en `SMALLINT` avec une colonne définie en `DECIMAL`

```
.               WHERE DECIMAL(SMALLINT_COL,9,2) = DECIMAL_COL
```

1.21 Quand utiliser `BETWEEN` et < = `AND` > =

Un prédicat `BETWEEN` est plus facile à lire que son équivalent codé à l'aide de combinaisons de `inférieur ou égal` et `supérieur ou égal`.

Dans les précédentes versions de DB2, le `BETWEEN` était plus performant, mais à présent, l'optimiseur reconnaît les 2 formulations de façon équivalente, sans différence de performance entre l'une ou l'autre. Un prédicat «BETWEEN reste plus facile à comprendre et à maintenir que de multiples `<= and >=`, il faut donc favoriser l'usage du `BETWEEN`.

Il existe cependant une exception à cette règle dans le cas particulier de la comparaison d'une host variable avec 2 colonnes :

```
.               WHERE :HOST BETWEEN COL1 AND COL2
```

doit être réécrit ainsi :

```
.               WHERE :HOST >= COL1 AND :HOST <= COL2
```

La raison de cette exception est qu'une formulation `BETWEEN` avec comparaison d'une host variable avec 2 colonnes est un prédicat STAGE 2, alors que la seconde formulation est STAGE 1.

1.22 Préférer le `IN (liste)` au `LIKE`

Quand c'est faisable, il faut préférer l'emploi d'un `IN` ou d'un `BETWEEN` plutôt qu'un `LIKE` dans la clause `WHERE` d'un `SELECT`.

Par exemple, si le nombre exact d'occurrences différentes retourné par un `LIKE` est connu :

```
.               WHERE COL LIKE 'VALUE%'
```

peut être réécrit de manière plus performante ainsi :

```
.               WHERE COL IN ('VALUE1','VALUE2','VALUE3')
```

Les fonctionnalités d'un `LIKE` peuvent être obtenues avec un `BETWEEN`:

```
.               WHERE COL LIKE 'K%'
```

est plus performant écrit ainsi :

```
.              WHERE COL BETWEEN 'KAAAAAAA' AND 'KZZZZZZZ'
```

1.23 Ne pas utiliser `LIKE` avec `%` ou `_` en début de host variable

La présence de l'un de ces caractère en début de recherche empêche DB2 d'utiliser un matching d'index.

A l'intérieur d'une host variable, l'optimiseur ne peut pas déterminer à l'avance à quelle position se trouvera le caractère `%` ou `_`, celui-ci ne peut donc pas supposer que l'index ne sera utilisé. C'est à l'exécution, après analyse du premier caractère, que DB2 décidera ou pas de faire un matching index", un non matching index scan ou un tablespace scan

1.24 Coder les prédicats les plus restrictifs en premier

Certaines personnes affirment que l'ordre des prédicats n'a pas d'importance dans une requête car l'optimiseur est assez intelligent pour les réorganiser correctement.

Cela est en partie vrai, l'optimiseur s'améliorant constamment d'une version de DB2 à l'autre.

Toutefois, si l'on comprend pourquoi cet ordre est important pour DB2 lors de l'évaluation de la requête et quelles sont les règles appliquées par l'optimiseur pour ordonner les prédicats, on pourra être certain que les requêtes fonctionneront toujours de façon optimale même dans les situations où l'optimiseur ne saura pas les réécrire

DB2 utilise un ordre prédéfini pour l'évaluation de chacun des prédicats. Cette séquence d'évaluation dépendant de 4 facteurs différents :

- les index utilisés
- l'éligibilité du prédicat en Stage 1 ou Stage 2
- le type de prédicat
- l'ordre dans lequel est codé physiquement le prédicat dans la requête

---

Deux ensembles de règles définissent précisément l'ordre d'évaluation:

- **Première série de règles:**
  1. les prédicats indexables sont appliqués en premier sur les index définis dans le chemin d'accès dans la même séquence que les colonnes de l'index (matching index), ensuite sont traités les prédicats Stage1 référençant des colonnes d'index « non matching » (screening index)
  2. Ensuite les prédicats Stage 1 ne référençant pas une colonne d'index sont appliqués après accès aux données
  3. Enfin, les autres prédicats Stage2 sont appliqués sur toutes les lignes déjà qualifiées
- **Deuxième série de règles**, dans chaque étape d'évaluation, les prédicats sont traités dans cet ordre:
  1. tous les prédicats de type égalité (`=, >, < , BETWEEN, IN` à 1 élément, etc)
  2. tous les prédicats de type "RANGE" (`IN, LIKE`) et les prédicats `IS NOT NULL`
  3. tous les autres prédicats (les sous-requêtes non corrélées avant les corrélées)
- **Enfin, lorsque ces 2 ensembles de règles sont épuisées**, c'est l'ordre physique dans la requête qui est pris en compte.

Il subsiste de ce fait, un certain niveau de contrôle pour l'ordre d'évaluation des prédicats.

Il faut donc toujours placer les prédicats les plus discriminants en premier (dans chaque groupe d'évaluation et pour un même type de prédicat)

Le gain obtenu par déplacement de prédicats est généralement minime, mais peut parfois être significatif

> ***Note***
>
> Cette règle s'applique uniquement pour des prédicats de même type

1.25 Minimiser les calculs dans les «Colonne Expressions»

Un index standard ne peut pas être utilisé avec une colonne participant à une expression arithmétique.

Par exemple, ce prédicat est non-indexable :

```
.               SELECT * FROM TABLE
                WHERE COL_DATE - 10 DAYS = :HV-DATE
```

En déplaçant les calculs dans la partie droite de l'égalité, on obtient un prédicat indexable (car DB2 sait indexer une «noncol expression») :

```
.               SELECT * FROM TABLE
                WHERE COL_DATE = DATE(:HV-DATE) + 10 DAYS
```

On peut également effectuer le calcul dans le programme avant la requête :

```
.          ADD +10 TO   HV-DATE
           Exec sql
             SELECT * FROM TABLE
             WHERE COL_DATE = :HV-DATE
           End-exec
```

En règle générale, il faut éviter les calculs dans SQL, excepté pour les fonctions DATE/TIME qu'il est souvent difficile de reproduire en cobol

Moins une requête contient d'expressions arithmétiques, plus il est facile de la comprendre et de juger rapidement si elle est indexable et Stage 1

> ***Note***
>
> DB2 V9 offre maintenant la possibilité de créer des index «sur expressions»

1.26 Utiliser `SYSDUMMYx` pour des données inexistantes en table

On a quelques fois besoin de sélectionner des données qui ne figurent pas dans les tables accédées. Cela représente une difficulté sans l'aide des tables `SYSIBM.SYSDUMMYx`

Par exemple, on peut utiliser la fonction `RAND` qui ne requiert pas de table DB2 :

```
.               SELECT RAND(:HOST-RAND-RANGE)
                FROM SYSIBM.SYSDUMMY1
```

Note

SYSIBM.SYSDUMMY1 : version originale ebcdic

SYSIBM.SYSDUMMYA : version ascii

SYSIBM.SYSDUMMYE : version ebcdic (=SYSIBM.SYSDUMMY1)

SYSIBM.SYSDUMMYU : version unicode.

1.27 Limiter les fonctions scalaires dans les clauses `WHERE`

Pour des raisons de performance, il faut éviter d'utiliser des fonctions scalaires dans les clauses `WHERE`. Ces fonctions ne sont pas indexables, sauf à créer spécifiquement un «Index on Expression » pour le prédicat (DB2 V9). En revanche, leur utilisation dans les `SELECT-list` n'a que très peu d'impact sur les performances

1.28 Désactiver le «List Prefetch» avec `OPTIMIZE FOR 1 ROW`

Dans certains cas de figure, quand un «list prefetch» dégrade les performances, on peut ajouter la clause `OPTIMIZE FOR 1 ROW`. Cela va décourager l'optimiseur de choisir un chemin d'accès avec «list prefetch», ce qui peut être intéressant dans un environnement TP où des données sont affichées à l'écran une page à la fois.

Sequential Prefetch :

- Décidé par l'optimiseur au moment du bind
- (+) Chaque lecture de page coûte 1.6 msec
- (+) Les pages sont lues par groupes de 32

Dynamic Prefetch :

- (+) Décidé au moment de l'exécution
- (+) Chaque lecture de page coûte un accès disque (~ 1ms)
- (+) Les pages sont lues par groupes de 32 (certaines pouvant êtres inutiles)

List Prefetch :

- (+) Choisi par l'optimiseur pour un accès indexé dont le CLUSTERRATIO est < 80%
- (+) Lecture des seules pages utiles
- (-) Requiert un tri des record identifiers (RIDs) de l'index avant lecture des pages
- (-) Données non triées sur l'index (un `ORDER BY` nécessitera donc un tri supplémentaire)
- (-) Chaque lecture de page coûte plusieurs accès disque (1 à 20 ms)
- (-) Si une application ferme prématurément un curseur avant d'avoir balayé l'ensemble des lignes, les ressources utilisées par le list prefetch pour construire la `sorted RID list` sont perdues

1.29 Désactiver un accès par un index

Pendant la phase de mise au point d'une requête, on peut ajouter la condition `OR 0 = 1` à un prédicat pour désactiver l'utilisation d'un index particulier.

`OR 0 = 1` a pour effet de "dégrader" le prédicat en Stage 2, forçant DB2 à utiliser un autre index ou un tablespace scan.

De la même manière on peut ajouter 0 à une colonne numérique ou concaténer une chaîne vide à la fin d'une colonne caractère. Cette dernière technique ayant l'avantage de simplement désactiver l'accès «matching index» sans pour autant transformer le prédicat en Stage 2

1.30 Explorer d'autres chemins d'accès

`OPTIMIZE FOR 1 ROW` et `(.. OR 0=1)` sont 2 méthodes possibles pour encourager DB2 à choisir un chemin d'accès différent.

On peut encore utiliser d'autres techniques:

- Déclarer une table `VOLATIL` désactive le list prefech et certaines techniques de l'optimiseur
- Coder un `ORDER BY` sur les colonnes d'un index dans la séquence de l'index pour favoriser le choix de cet index
- Ajouter un prédicat:

```
.             ... AND DATE_NAISSANCE >= '0001-01-01'
```

- Coder une égalité dans une sous-requête décourage l'utilisation d'un index par "transitive closure":

```
.             WHERE COL = (SELECT 'ABCDEF' FROM SYSIBM.SYSDUMMY1)
```

- `OPTIMIZE FOR n ROWS` (n pouvant avoir n'importe quelle valeur)
- `FETCH FIRST n ROWS ONLY`.
- « No Operation » : `+0, -0, /1, *1, CONCAT ''` (chaîne vide).

  Ajouter ou soustraire zéro, diviser ou multiplier par 1, ou concaténer une chaîne vide ne modifient pas le résultat d'une requête mais peuvent faire changer de décision l'optimiseur.

  Bien que ces prédicats soient théoriquement indexables, IBM a décider de faire une exception pour les « no-operation colonne expression » suivantes : +0, -0, /1, *1, et CONCAT, considérant qu'elles étaient couramment utilisées depuis des années par les programmeurs comme des «astuces» pour désactiver un accès par index en «trompant» l'optimiseur.

  Dans l'exemple suivant, un «tablespace scan» est forcé car une chaîne vide est concaténée à la host variable, et il n'existe pas d'autres prédicats pour un accès indexé. Cependant le prédicat reste Stage 1 :

```
.               SELECT * FROM TABLE
                WHERE COL < :HOST-VAR CONCAT ''
```

Ces techniques sont réservées normalement pour des tests, mais il est acceptable d'en tolérer quelques unes en production (plutôt que de devoir modifier manuellement les statistiques par exemple)

1.31 Favoriser le `NOT EXISTS` plutôt que le `NOT IN`

Avec une sous-requête utilisée en négation, il faut essayer de coder `NOT EXISTS` plutôt que `NOT IN`

`NOT EXISTS` se contente de tester la non existence d'une valeur, tandis que `NOT IN` doit d'abord matérialiser puis trier l'ensemble des résultats de la sous-requête.

1.32 Ordonner les éléments dans les listes `IN ( . . . )`

L'ordre des éléments dans une liste `IN (...)` peut impacter les performances dans le cas ou un index n'est pas utilisé. DB2 parcours dans ce cas la liste de gauche à droite jusqu'à trouver une valeur qui corresponde.

Pour cette raison, il convient donc de placer en début de liste les valeurs les plus souvent rencontrées, indépendamment de tout classement alphanumérique

1.33 Eliminer les doublons dans les listes `IN`

Lorsqu'on utilise un prédicat `IN` avec une liste de valeurs, DB2 trie la liste dans l'ordre ascendant pour éliminer les doublons si la colonne spécifiée est indexée.

Si la colonne n'est pas indexée, le tri et l'élimination des doublons n'a pas lieu.

Il y a donc un intérêt à ordonner cette liste à partir de la valeur la moins restrictive (la plus souvent rencontrée en table) à la plus restrictive et d'éliminer les doublons dans le cas d'un accès non indexé

1.34 Connaître les règles de «transitive closure» appliquées par DB2

La «transitive closure» dans DB2 est la capacité de celui-ci à utiliser des règles de transitivité sur les prédicats (`Si A=B et B=C alors A=C`) de manière à calculer un chemin optimal pour l'accès aux données lors des opérations de jointure

Par exemple, sur cette requête, DB2 détermine automatiquement un prédicat supplémentaire à ajouter à la requête:

```
.               SELECT * FROM A, B
                WHERE A.COL1 = B.COL1
                  AND B.COL1 =:HOST
      * => prédicat ajouté automatiquement par l'optimiseur:
                  AND A.COL1 =:HOST
```

Il n'est donc pas nécessaire de coder un prédicats redondant sur les égalités et les «RANGE» prédicats.

Cependant, la «transitive closure» n'est pas appliquée avec les sous-requêtes et les prédicats `LIKE` (ainsi que `IN list` jusqu'en DB2 V10).

Ainsi dans l'exemple suivant, pour obtenir une requête plus efficace il est nécessaire d'indiquer explicitement à DB2 un prédicat supplémentaire:

```
.               SELECT * FROM A, B
                WHERE A.COL1 = B.COL1
                  AND B.COL1 LIKE 'VALUE%'
      * => prédicat à ajouter manuellement:
                  AND A.COL1 LIKE 'VALUE%'
```

Note

Avant DB2 V10, la transitivité ne s'appliquait pas sur les `IN list`, ce n'est plus le cas aujourd'hui

Lors d'un `EXPLAIN`, l'optimiseur indique dans la colonne `ADDED_PRED` de la table `DSN_PREDICATE_TABLE` si un prédicat a été généré automatiquement par application d'une règle de « transitive closure »

1.35 Utiliser `FETCH FIRST n ROWS ONLY` pour limiter certains résultats

La clause `FETCH FIRST N ROWS ONLY` limite le nombre de lignes qualifiées pour n'importe quel ordre `SELECT`.

`SQLCODE` est positionné à `+100` après la Nième ligne lue, ou avant si le nombre de lignes qualifiées est inférieur à N.

`FETCH FIRST N ROWS ONLY` ne fonctionne pas comme `OPTIMIZE FOR N ROWS`. La clause `FETCH FIRST` stop le résultat après que soient retournées les N premières lignes, alors que la clause `OPTIMIZE FOR` non (elle ne fait qu'orienter le choix de l'optimiseur vers un chemin favorisant l'accès aux N premières lignes). Cependant en spécifiant `FETCH FIRST N ROWS ONLY`, DB2 considère implicitement la présence d'un `OPTIMIZE FOR` pour le même nombre de lignes, il n'est donc pas nécessaire de coder `OPTIMIZE FOR` avec une clause `FETCH FIRST`

Si vous n'indiquez pas un `ORDER BY` dans une requête avec `FETCH FIRST N ROWS ONLY`, le résultat reste imprévisible. Il n'y a pas d'ordre prédéterminé dans une table DB2, le résultat d'une requête est uniquement basé sur le chemin d'accès choisi par l'optimiseur DB2. Ainsi `FETCH FIRST N ROWS ONLY` limite le résultat à N lignes mais ne garantie jamais des lignes identiques à chaque exécution.

Note

Depuis DB2V9 la clause `FETCH FIRST n ROWS ONLY` peut être spécifiée dans une sous-requête ou une fullselect:

```
.             SELECT * FROM DEPT
              WHERE DEPTNO IN
                           (SELECT WORKDEPT FROM EMP
                            ORDER BY SALARY DESC
                            FETCH FIRST 1 ROW ONLY)
```

1.36 Coder des tests d'existence appropriés et performants

Les programmes ont quelquefois besoin de connaître si au moins une donnée existe, et sans nécessairement avoir besoin d'en connaître la valeur.

Avant DB2 V7, la meilleur méthode consistait à passer par la table `SYSIBM.SYSDUMMY`

```
.               SELECT 1 FROM SYSIBM.SYSDUMMY1
                WHERE EXISTS
                     (SELECT 1 FROM TABLE
                      WHERE NAME = 'VALUE')
```

Aujourd'hui on peut coder plus simplement avec `FETCH FIRST 1 ROW ONLY`

```
.               SELECT 1
                INTO :W-NUM
                FROM TABLE
                WHERE NAME = 'VALUE'
                FETCH FIRST 1 ROW ONLY
```

Si `SQLCODE = ZERO`, la donnée existe, sinon elle n'existe pas.

On est souvent tenté de selectionner la colonne sur laquelle porte le test, cela n'est absolument pas nécessaire (puisque c'est `SQLCODE` détermine le test et non le contenu de la colonne) et significativement moins performant:

```
.               SELECT COL
                INTO :COL
                FROM TABLE
                WHERE COL = :COL
```

L'ordre suivant peut être jusqu'à 20% plus performant que le précédent:

```
.               SELECT 1
                INTO :W-NUM
                FROM TABLE
                WHERE COL = :COL
```

Autre exemple fréquemment rencontré:

```
.               SELECT VALUE INTO :W-VALUE
                FROM TABLE
                WHERE ID = :W-ID
                  AND T_TIMESTAMP IN (SELECT MAX(T_TIMESTAMP)
                                      FROM TABLE
                                      WHERE ID = :W-ID)
```

L'ordre suivant est plus simple et plus performant:

```
.               SELECT VALUE INTO :W-VALUE
                WHERE ID = :W-ID
                FETCH FIRST 1 ROW ONLY
                ORDER BY VALUE DESC
```

1.37 Coder des jointures SQL plutôt que des «jointures programmées»

Une jointure codée avec SQL est en général toujours plus performante que programmée dans un langage évolué comme cobol.

L'optimiseur DB2 dispose d'un arsenal de techniques et de ressources qui surpassent les algorithmes qui peuvent être employés par un programmeur cobol.

Il ne faut programmer soi-même une jointure qu'en dernier ressort, après que toutes les autres possibilités aient été explorées.

Tests comparatifs réalisés avec 3 tables de 100 000 lignes:

jointure programmée en cobol	jointure SQL	jointure SQL + MRF
29 sec cpu	17 sec cpu	13 sec cpu
	gain: 40%	**gain: 55%**

1.38 Transformer une sous-requête `IN` en `EXISTS` (ou l'inverse)

Les 2 requêtes ci-dessous produisent exactement le même résultat, mais opèrent de façon différente dans DB2, l'une étant plus performante que l'autre en fonction de la distribution des données dans les tables

Par exemple, cette sous-requête non corrélée :

```
.               SELECT * FROM T1
                WHERE T1.C1 IN
                            (SELECT T2.C1
                             FROM T2
                             WHERE T2.C2 LIKE 'D%')
```

peut aussi être ré-écrite ainsi :

```
.              SELECT * FROM T1
               WHERE EXISTS
                     (SELECT 1
                      FROM T2
                      WHERE T2.C1 = T1.C1
                        AND T2.C2 LIKE 'D%')
```

Cette technique d'optimisation appelée **«correlation and de-correlation»** est mise en œuvre par l'optimiseur DB2 V9 qui transforme parfois, sous certaines conditions, une sous-requête en jointure ou en sous-requête avec `EXISTS`.

Il n'en demeure pas moins intéressant de coder directement les requêtes sous leur forme optimisée:

- afin de pérenniser les performances (une maintenance évolutive pourrait neutraliser le choix de l'optimiseur).
- certaines requêtes complexes ne pouvant pas bénéficier de la technique de «correlation and de-correlation» de l'optimiseur

1.39 Favoriser les jointures plutôt que les sous-requêtes

Une jointure peut être plus performante qu'une sous-requête corrélée ou une sous-requête

```
.               SELECT COL1, COL2
                FROM T1 X
                WHERE COL1 IN
                          (SELECT COL1 FROM T2
                           WHERE   COL2 = X.COL2)
```

la requête précédente est équivalente mais généralement moins performante que celle-ci :

```
.               SELECT T1.COL1, T1.COL2
                FROM T1, T2
                WHERE T1.COL1 = T2.COL1
                  AND T1.COL2 = T2.COL2
```

Bien que les performances des sous-requête corrélée et non corrélées s'améliorent dans les nouvelles versions de DB2, il est préférable d'écrire une jointure (quand c'est possible) plutôt qu'une sous-requête car la maintenabilité et l'évolutivité de la requête s'en trouvent améliorées.

Par ailleurs, dans certains cas, l'optimiseur transforme de lui-même une sous-requête en jointure.

1.40 Joindre les tables sur des colonnes «INDEXED»

La performance des jointures est directement liée à la présence d'index sur les prédicats.

Il est parfois nécessaire de créer des index spécialisés sur l'ensemble des colonnes figurant dans les prédicats de jointure

1.41 Joindre les tables volumineuses sur des colonnes «CLUSTERED»

Lors de jointures sur des tables volumineuses, il faut autant que possible utiliser des colonnes CLUSTERED comme critères de jointure, cela évite certains tris intermédiaires.

1.42 Attention à l'utilisation de `ORDER BY` dans une jointure

Lorsque le résultat d'un jointure doit être trié, il faut si possible limiter l'ORDER BY aux colonnes d'une seule table afin de permettre à DB2 d'éviter un tri.

Chaque fois que des colonnes de plusieurs tables sont spécifiées pour un ORDER BY dans une jointure, DB2 invoque une procédure de tri

1.43 Fournir des critères de recherche appropriés

Lorsque cela est possible, fournir des critères de recherche supplémentaires dans la clause WHERE de chaque table dans une jointure. Ces critères sont, en plus des critères de jointure, qui sont obligatoires pour éviter des produits cartésiens. Ces informations fournies à l'optimiseur DB2, permet un meilleur classement des tables à joindres (en réduisant la taille des tables résultantes intermédiaires). En général, plus les informations fournies à DB2 sont nombreuses pour une requête, meilleures sont les chances de l'exécuter de manière optimale

1.44 Préférer un codage explicite pour les INNER JOIN

Plutôt que de spécifier une jointure en utilisant un liste de tables séparée par des virgules dans la clause FROM, il est préférable d'utiliser INNER JOIN avec une clause ON.

La syntaxe d'un INNER JOIN explicite contient le mot clé JOIN favorisant la compréhension de la jointure.

De la même façon, les prédicats de jointure doivent être isolés dans la clause ON, facilitant ainsi la lecture, la maintenance et l'évolution future du code SQL.

1.45 Préférer un codage explicite pour les OUTER JOIN

Eviter l'ancien style de codage des jointures externes utilisant un simple SELECT, une UNION et une sous-requête corrélée.

Une jointure externe LEFT OUTER JOIN est plus facile à coder, à lire et à maintenir.

La jointure explicite est plus performante car elle tente d'effectuer le traitement de toutes les tables en une seule passe.

1.46 Coder des LEFT OUTER JOIN plutôt que des RIGHT OUTER JOIN

Une jointure externe peut indifféremment être codée LEFT OUTER ou RIGHT OUTER, à condition de bien choisir la table gauche (outer) et la table droite (inner) dans l'opération de jointure, et ceci, sans incidence sur les performances.

Cependant, les LEFT OUTER JOINS sont plus facile à lire et appréhender :

- La «table gauche» (outer) étant toujours codée en premier, les autres tables jointes avec LEFT OUTER arrivant ensuite (en liste) dans la requête.
- L'EXPLAIN renseigne la colonne JOIN_TYPE dans la PLAN_TABLE pour décrire la méthode de jointure (FULL, RIGHT or LEFT).

  Cette colonne contient F pour FULL , L pour LEFT ou RIGHT et blanc pour pour INNER ou NO JOIN.

  Db2 convertit donc toujours un RIGHT en LEFT. Il devient donc plus difficile de décrypter un RIGHT OUTER JOIN dans la PLAN_TABLE car la valeur R pour RIGHT n'existe pas

DB2 représente donc en interne tous les RIGTH JOIN en LEFT JOIN

1.47 Utiliser COALESCE avec les jointures externes

Si on veut éviter d'avoir à mettre en oeuvre des variables indicateurs de NULL pour les colonnes pouvant retourner NULL, on peut utiliser COALESCE

Par exemple, dans un OUTER JOIN

```
.               SELECT COALESCE(T1.COL1,'**COL1 is NULL in table T1**'),
                       COALESCE(T2.COL1,'**COL1 is NULL in table T2**')
                FROM T1
                FULL OUTER JOIN T2
                  ON T1.COL1 = T2.COL1
```

Dans un `MAX(COL)`

```
.               SELECT COALESCE(MAX(SALAIRE), 0) FROM TABLE WHERE …
```

> ***Note***
>
> La fonction `VALUE` est un synonyme pour `COALESCE`. Cependant `COALESCE` respecte mieux le standard ANSI. La fonction `IFNOTNULL`, parfois employée, offre moins de souplesse que `COALESCE`

1.48 Limiter les colonnes dans un `GROUP BY`

Lorsqu'on utilise un `GROUP BY` pour obtenir une agrégation de données, il ne faut spécifier que les colonnes strictement nécessaires.

DB2 doit opérer un tri pour grouper les données, le nombre de colonnes participant au tri influe donc directement sur les performances de la requête.

1.49 `GROUP BY` et `ORDER BY` ne sont pas équivalents

La clause GROUP BY trie des données dans l'unique but de les agréger, le résultat n'est pas nécessairement dans l'ordre induit par le `GROUP BY`.

Si l'on veut s'assurer que le résultat soit trié dans un ordre spécifique, il faut ajouter la clause `ORDER BY`.

Si `GROUP BY` et `ORDER BY` sont spécifiés ensemble de manière compatible, DB2 peut éliminer un tri redondant.

1.50 `ORDER BY` et colonnes «SELECTEES»

DB2 permet de spécifier une colonne dans une clause `ORDER BY` même si celle-ci n'est pas présente dans la `SELECT-liste`.

En revanche, vous ne pouvez pas éliminer une colonne sur laquelle porte une «fonction colonne», ou avec un `UNION`, `UNION ALL`, `GROUP BY`, `DISTINCT`.

1.51 Ajouter `ALL` aux opérateurs ensemblistes pour éviter le tri

Les opérateurs `UNION`, `INTERSECT` et `EXCEPT` provoquent un tri final des valeurs résultantes afin d'en retirer les doublons.

On peut éliminer le processus de tri en ajoutant la spécification `ALL` à ces opérateurs, dans la mesure ou l'on sait d'avance que le résultat ne peut pas contenir de doublons (par la connaissance des données manipulées), ou que les doublons sont tolérés par le programme applicatif

1.52 Créer des «Index on Expression»

```
.           WHERE (SALAIRE + COMMISSION)  > 1000


            CREATE INDEX remuneration ON employee(SALAIRE + COMMISSION)
```

1.53 Ajouter des colonnes «non clé» à un Index Unique (V10)

```
.           CREATE UNIQUE INDEX ix ON t1(c1,c2,c3)
                   INCLUDE (c4,c5)
```

1.54 Utiliser des «Table Expressions» plutôt que des vues

Les «Table Expressions», parfois appelées «Vues Intégrées», permettent à une clause `FROM` d'un `SELECT` de contenir un autre `SELECT`

- En regroupant ainsi tout le code SQL dans le programme, l'intention du développeur devient plus claire, il devient plus facile de maintenir et de déboguer le code.
- Lorsqu'on est confronté à un problème avec du code utilisant une vue, à moins de disposer d'outils spécifiques, il faut interroger le catalogue (SYSIBM.SYSVIEWS), mais le code SQL stocké en table est difficile à lire car il n'est pas mis en forme
- Une «Table Expression» peut servir à forcer un ordre de traitement spécifique dans une requête, à "pré-filtrer" une jointure ou à isoler un processus de `GROUP BY` en dehors d'une jointure pour la rendre plus efficace

  Par exemple, cette première requête qui comporte une «Table Expression» est équivalente mais plus performante que la seconde car les 2 fonctions d'agrégation `MIN()` traitées après la jointure portent sur plus de lignes :

```
.           SELECT D.DEPTNO, D.DEPTNAME, D.LOCATION, TOTAL_SALARY
            FROM DEPT D,
                 (SELECT WORKDEPT, SUM(SALARY) AS TOTAL_SALARY
                  FROM EMP E
                  WHERE E.BONUS BETWEEN 0.00 and 1000.00
                  GROUP BY E.WORKDEPT) AS E
            WHERE D.DEPTNO = E.WORKDEPT;
```

```
.           SELECT D.DEPTNO, MIN(D.DEPTNAME) AS DEPT_NAME, MIN(D.LOCATION) AS DEPT_LOCATION,
                   SUM(E.SALARY) AS TOTAL_SALARY
            FROM DEPT D, EMP E
            WHERE D.DEPTNO = E.WORKDEPT
              AND E.BONUS BETWEEN 0.00 AND 1000.00
            GROUP BY D.DEPTNO
```

1.55 Utiliser les «Row Expressions»

Une «Row Expression» permet de coder plusieurs comparaisons à la fois sur un ensemble de colonnes dans un prédicat utilisant une sous-requête.

Par exemple :

```
.               SELECT * FROM T1
                WHERE (COL1, COL2) IN (SELECT COLX, COLY FROM T2)
```

Un nombre quelconque de colonnes peut être spécifié.

Outre l'avantage de simplifier considérablement l'écriture de certaines requêtes, les «Row Expressions» peuvent procurer des gains de performance significatifs.

Ainsi, en codant avec l'ancienne méthode la même requête que dans l'exemple précédent, on obtenait un double scan de la table T2 :

```
.               SELECT * FROM T1
                WHERE COL1 IN (SELECT COLX FROM T2)
                  AND COL2 IN (SELECT COLY FROM T2)
```

1.56 Utiliser des «Scalar Fullselects»

Les «Scalar Fullselects» ont été introduites par DB2 V8, elles permettent d'écrire des requêtes plus intelligentes et plus performantes.

Une «Scalar Fullselect» utilisée dans une expression est une «Fullselect» entourée de parenthèses, elle doit retourner une seule ligne ne comportant qu'une seule valeur.

Elles permettent de coder un `SELECT` n'importe où :

- dans un `SELECT`
- dans une clause `WHERE`
- dans un `CASE`

Elles permettent également de sélectionner des combinaisons de données détaillées et agrégées.

«Scalar Fullselects» dans une clause WHERE:

```
.               SELECT * FROM T1
                WHERE COL1 BETWEEN 2.0 * (SELECT MIN(COL1) FROM T1)
                               AND 0.5 * (SELECT MAX(COL1) FROM T1)
```

«Scalar Fullselects» dans une SELECT-liste:

```
.               SELECT T1.COL1
                      ,(SELECT MAX(COL2)
                        FROM T2
                        WHERE T2.COL1 = T1.COL1)
                FROM T1
```

1.57 Utiliser les «Common Table Expressions» et la récursion

Les «Common Table Expressions» (CTE) permettent de créer des table «inline» construites uniquement pour la durée et les besoins d'une requête, sans avoir recours à une table temporaire.

La possibilité de coder du SQL récursif avec les CTE offre de nouvelles fonctionnalités intéressantes, comme par exemple celles de créer des générateurs de données à l'intérieur d'une requête.

Voici une CTE récursive calculant `factorielle(n)` jusqu'à 10 :

```
.               WITH FACTORIELLE(N, FACT) AS
                     (SELECT 0, 1 FROM SYSIBM.SYSDUMMY1
                      UNION ALL
                      SELECT N + 1, (N + 1) * FACT
                      FROM FACTORIELLE
                      WHERE N < 10 )

                SELECT * FROM FACTORIELLE

      *       ---------+---------+---------+---------+-
      *                N         FACT
      *       ---------+---------+---------+---------+-
      *                0            1
      *                1            1
      *                2            2
      *                3            6
      *                4           24
      *                5          120
      *                6          720
      *                7         5040
      *                8        40320
      *                9       362880
      *               10      3628800
```

Dans l'exemple suivant, une table est peuplée avec un identifiant unique et des données aléatoires en une seule passe :

```
.               INSERT INTO TABLE
                WITH ID_GENERATOR(ID) AS
                    (SELECT 1 FROM SYSIBM.SYSDUMMY1
                     UNION ALL
                     SELECT ID + 1
                     FROM ID_GENERATOR
                     WHERE ID < 10 )
                SELECT
                      ID
                     ,SUBSTR('BCDFGHJKLMNPRSTVWZ',INTEGER(ROUND(RAND()*17,0)) +1,1)
                      CONCAT
                      SUBSTR('AEIOUY',INTEGER(ROUND(RAND()*5,0))+1,1)
                      CONCAT
                      SUBSTR('BCDFGHJKLMNPRSTVWZ',INTEGER(ROUND(RAND()*17,0)) +1,1)
                      AS NAME
                     ,CURRENT DATE - INTEGER(20*365*RAND()) DAYS AS DATE
                FROM ID_GENERATOR
```

la CTE `ID_GENERATOR` est une instruction récursive qui «compte jusqu'à 10» puis s'arrête, créant ainsi une table «inline» :

```
.       *  ---------+---------+---------+-----
        *  ID NAME DATE
        *  ---------+---------+---------+-----
        *  1 PAW 2002-12-28
        *  2 WIW 2005-07-10
        *  3 DAP 2006-06-26
        *  4 JYT 1999-06-25
        *  5 TOT 2008-02-29
```

```
      *  6 DIC 2002-01-10
      *  7 MUZ 1998-07-16
      *  8 COC 1995-10-17
      *  9 CUJ 1996-07-07
      * 10 FOS 1996-05-30
```

1.58 Utiliser le «Multi-Row» FETCH (MRF)

Un «Multi-Row» FETCH retourne plusieurs lignes au programme applicatif en une seule fois dans un tableau de colonnes.

En rapatriant de nombreuses lignes en une seule fois, le nombre de CALL à DB2 diminue, rendant le programme plus performant

Un exemple complet de mise en oeuvre du MRF en cobol est présenté dans la section: Exemple de Multi Rows Fetch (MRF).

1.59 Utiliser le «Multi-Row» INSERT (MRI)

Deux modes de fonctionnement:

- mode ATOMIC : les lignes sont insérées en table en "tout ou rien"
- mode NOT ATOMIC CONTINUE ON SQLEXCEPTION : seules les lignes en erreur sont rejetées

  Le mode NOT ATOMIC est moins performant car il nécessite une gestion plus complexe de la «Log DB2» afin d'être en mesure d'effectuer correctement un ROLLBACK

> ***Attention!***
>
> Il faut veiller à toujours bien vider les HostVariable Areas en **fin** de programme et **avant** chaque COMMIT (par un Perform EXEC-MRI comme dans l'exemple ci-dessous)

```
.      *==========*
        INSERT-MRI.
       *==========*
           Add 1 To ind-row
           Move ALIAS-ID of DCLTLWAAALE
             To ALIAS-ID of HVA-DCLTLWAAALE(ind-row)

           ...

           if ind-row = 100
             Perform EXEC-MRI
           end-if
           .
        EXEC-MRI.
       *========*
           Exec Sql
               INSERT INTO tlwaaale
               VALUES (:HVA-DCLTLWAAALE)
```

```
            FOR :ind-row ROWS
            ATOMIC
        End-exec
        If SQLCODE Not = 0
          Perform ERROR-SQL-GET-DIAGNOSTICS
        Else
          Move 0 To ind-row
        End-if
        .
     COMMIT-PROG.
    *===========*
        ...
        Perform EXEC-MRI
        Exec SQL COMMIT WORK End-exec
        ...

     END-PROG.
    *========*
        ...
        Perform EXEC-MRI
        ...
        Stop Run.
```

Gains mesurés en insertion MRI sur une table de 5 colonnes (90 octets) avec de 0 à 2 index:

nb index	100 000 inserts normaux	100 000 inserts MRI NOT ATOMIC	Gain	100 000 inserts MRI ATOMIC	Gain
2	22 sec cpu	20 sec cpu	**9%**	17 sec cpu	**22%**
1	16 sec cpu	14 sec cpu	**12%**	12 sec cpu	**25%**
0	9 sec cpu	7 sec cpu	**22%**	5 sec cpu	**44%**

1.60 Appeler `GET DIAGNOSTICS` seulement en cas d'erreur

Dans le cas d'un multi-row `FETCH` on peut avoir un `SQLCODE = +354` en retour, indiquant qu'il s'est produit une ou plusieurs erreurs

Dans le cas d'un `NON ATOMIC` multi-row `INSERT` on peut avoir un `SQLCODE = -253` si certaines insertions sont en erreur ou un `SQLCODE = -254` si toutes les insertions ont échoué

Dans le cas d'un `ATOMIC` multi-row `INSERT`, le `SQLCODE` est la valeur réelle de la première erreur rencontrée (`ATOMIC` s'arrête dès la première erreur), mais il faut cependant faire appel à `GET DIAGNOSTICS` pour connaître la ligne concernée par l'erreur.

`GET DIAGNOSTICS` est très couteux, il convient donc d'abord de tester la valeur du `SQLCODE` et de ne l'appeler qu'en cas d'erreur.

Si N erreurs sont rapportées par `GET DIAGNOSTICS (:error-count = Number)`, N appels à `GET DIAGNOSTICS CONDITION` renvoient:

- une première valeur de `DB2_RETURNED_SQLCODE` qui est un SQLCODE générique
- les valeurs suivantes de `DB2_RETURNED_SQLCODE` concernent chaque ligne en erreur (dans l'ordre inverse de la host variable area)

```
.      *-------------------------*
        ERROR-SQL-GET-DIAGNOSTICS.
       *-------------------------*
          Exec Sql
               GET DIAGNOSTICS
               :rows-returned = Row_Count
              ,:error-count = Number
          End-exec
          Display "rows returned=" rows-returned
          Display "   error-count=" error-count
          Perform Varying i-error From 1 By 1
              Until i-error > error-count
               Exec Sql
                GET DIAGNOSTICS CONDITION :i-error
                :RETURNED-SQLCODE = DB2_RETURNED_SQLCODE
               ,:ERROR-ROW-NUMBER = DB2_ROW_NUMBER
             End-exec
             Display '    - row num '    error-row-number
                     ' sqlcode=' RETURNED-SQLCODE
          End-perform
          Move +8 To RETURN-CODE
          Goback
          .
```

1.61 Gérer soi-même tous les SQLCODE et éviter les SQL WHENEVER

L'instruction `EXEC SQL WHENEVER ...` est une directive pour le précompilateur DB2 (et non pas une instruction SQL exécutable) qui spécifie des actions à effectuer en fonction de la valeur du `SQLCODE` pour toutes les autres instructions SQL qui suivent cette directive dans le programme. Elle permet par exemple de débrancher systématiquement le traitement par `GO TO` vers une routine de gestion d'erreur en cas de `SQLCODE` négatif.

Les sauts inconditionnels générés par `WHENEVER` sont incompatibles avec une approche de programmation structurée, il faut donc l'éviter et coder soi-même la gestion des `SQLCODE` après chaque `EXEC SQL`.

Certains SQLCODE sont sans gravité pour la poursuite du programme, d'autres doivent déclencher une gestion d'erreur, il est donc important de ne jamais passer sous silence aucune des valeurs d'un SQLCODE.

L'utilisation d'un `EVALUATE` accompagné d'un `WHEN OTHER` (plutôt qu'un `IF`) est une manière élégante, évolutive et robuste de gérer tous les cas possibles à l'issue d'un `EXEC SQL`

Attention!

Ne jamais mélanger différentes méthodes de gestion du SQLCODE à l'intérieur d'un même programme (confusion possible lors de maintenances)

`EXEC SQL WHENEVER` n'est pas une instruction SQL exécutable, elle ne peut donc pas être conditionnée par un `IF` cobol, elle est "activée" à la lecture du source par le précompilateur de "haut en bas"

1.62 Utiliser le SELECT from INSERT, UPDATE ou DELETE

Permettent la mise à jour et la lecture simultanées d'une table à l'intérieur d'un seul EXEC SQL

3 options de lecture :

- FROM OLD TABLE: lecture des données avant que les changements soient opérés en table
- FROM NEW TABLE: lecture des données après que les changements soient opérés en table (mais avant l'évaluation des contraintes d'intégrités et le déclenchement de triggers)
- FROM FINAL TABLE: lecture des données après que les changements soient opérés en table (mais après l'évaluation des contraintes d'intégrités et le déclenchement de triggers)

Exemple de SELECT from an INSERT

```
.               SELECT COL1 INTO :H1
                FROM FINAL TABLE
                    (INSERT (COL1, COL2, COL3) INTO T1
                     VALUES('JONES', 'CHARLES', CURRENT DATE))
```

Exemple de SELECT from an UPDATE:

```
.               SELECT SUM(salaire) INTO :SALAIRE-HV
                FROM FINAL TABLE
                     (UPDATE EMPLOYES
                      SET SALAIRE = SALARY * 1.1
                      WHERE DEPT = 'DBA')
```

Exemple de SELECT from an DELETE:

```
.               SELECT *
                FROM OLD TABLE
                    (DELETE FROM EMPLOYES
                     WHERE LOCALISATION = 'FLORANGE')
```

1.63 Utiliser le MERGE (update ou insert = Upsert)

```
.         Exec Sql
               MERGE INTO tlwaaale T
               USING (VALUES
                (:ALIAS-ID
                ,:ALIAS-TYPE
                ,:GLOBAL-CPY-ID
                ,:CPH-GRP-CD
                ,:DESC-ALIAS))
               AS N
               (ALIAS_ID
                ,ALIAS_TYPE
                ,GLOBAL_CPY_ID
                ,CPH_GRP_CD
                ,DESC_ALIAS)
              ON (T.ALIAS_ID   = N.ALIAS_ID and
                  T.ALIAS_TYPE = N.ALIAS_TYPE )
              WHEN MATCHED THEN
                UPDATE
                  Set DESC_ALIAS = N.DESC_ALIAS
              WHEN NOT MATCHED THEN
                INSERT
                 (ALIAS_ID
                 ,ALIAS_TYPE
                 ,GLOBAL_CPY_ID
                 ,CPH_GRP_CD
                 ,DESC_ALIAS)
                VALUES
                (N.ALIAS_ID
                ,N.ALIAS_TYPE
                ,N.GLOBAL_CPY_ID
                ,N.CPH_GRP_CD
                ,N.DESC_ALIAS)
         End-exec
         If sqlcode = 0
           ...
```

> ***Note***
>
> `NOT ATOMIC CONTINUE ON SQLEXCEPTION` est la seule option possible avec le Multi Row MERGE (MRM)

1.64 Utiliser la mémoïsation

Une fonction **mémoïsée** stocke les résultats de ses appels précédents dans une table et, lorsqu'elle est appelée à nouveau avec les mêmes paramètres, renvoie la valeur stockée au lieu de la recalculer.

Le terme anglais «memoization» est dérivé du mot latin « memorandum » signifiant « qui doit être rappelé ».

Bien que le terme mémoïsation évoque « mémorisation », il a une signification particulière ici.

Un exemple de mise en oeuvre complète de tables DB2 «mémoïsées» en cobol est présenté dans la section: Chargement à la volée en table hashcodée avec accès direct (mémoïsation).

2 COBOL TIPS

2.1 Compiler avec le coprocesseur intégré

- Les COPY books cobol peuvent contenir des `EXEC CICS`, `DLI` et `SQL`

  → il n'est plus nécessaire d'avoir 2 bibliothèques séparées pour les copy et includes

- Compilation en un seul step

  → débogage de la compilation sur l'unique source d'origine (diagnostics cobol, cics et sql réunis dans le même listing)

- Les directives `COPY REPLACING` s'appliquent aux `EXEC SQL` et `EXEC CICS`

- Simplification de la programmation avec les «nested programs»

  → `DFHEIBLK`, `DFHCOMMAREA` et `SQLCODE` sont générées comme `GLOBAL` dans le programme principal

  → il n'est plus nécessaire de les passer en paramètre dans les `CALL nested programs`

- Les zones binaires internes à CICS sont générées en `COMP-5`

  → les programmes CICS peuvent être compilés `TRUNC(OPT)`

2.2 Soigner l'ordre dans les expressions conditionnelles.

Dans une expression conditionnelles avec de multiples `OR`, il faut toujours veiller à coder le plus à gauche les conditions pour lesquelles la valeur VRAIE a le plus de chance d'être rencontrée, le compilateur pourra ainsi stopper au plus tôt l'évaluation globale de l'expression.

2.3 Factoriser les expressions

En factorisant des expressions dans les programmes, on arrive souvent à éliminer certains calculs inutiles

Par exemple, ce premier bloc de code :

```
.           MOVE ZERO TO TOTAL
            PERFORM VARYING I FROM 1 BY 1 UNTIL I = 1000
              COMPUTE TOTAL = TOTAL + ITEM(I)
            END-PERFORM
            COMPUTE TOTAL = TOTAL * REMISE
```

est plus efficace que celui-ci :

```
.           MOVE ZERO TO TOTAL
            PERFORM VARYING I FROM 1 BY 1 UNTIL I = 1000
              COMPUTE TOTAL = TOTAL + ITEM(I) * REMISE
            END-PERFORM
```

2.4 Utiliser des variables symboliques

Afin de permettre à l'optimiseur de reconnaître plus facilement les constantes, il faut initialiser celles-ci en `WORKING-STORAGE` avec une clause `VALUE` et ne plus changer sa valeur en cours de programme.

Lorsqu'une variable est passée à un sous-programme `BY REFERENCE` (défaut), cette variable ne sera jamais considérée par le compilateur comme une constante à moins d'effectuer le passage de paramètre `BY VALUE`

Si un littéral est chargé dans une variable en cours de programme, cette variable sera considérée comme une constante seulement dans une partie limitée du code (principe de l'optimiseur à «lucarne»)

2.5 Utiliser le `PACKED DECIMAL` pour calculer sur plus de 8 digits

Au delà de 8 digits, le code généré par le compilateur pour l'arithmétique décimale (`COMP-3`) devient en général plus efficace que celui de l'arithmétique binaire.

Cela est particulièrement vrai lorsque l'expression est compliquée ou comporte la clause `ROUNDED`.

> ***Note***
>
> Au delà de 18 digits, le compilateur utilise systématiquement l'arithmétique décimale
>
> De 9 à 18 digits, il utilise l'arithmétique binaire ou décimale suivant le cas

2.6 Eviter les conversions de type entre variables inconsistentes

Dans une opération mathématique avec plusieurs opérandes, s'ils sont de types différents, l'un des opérandes doit être au préalable converti dans le même type que l'autre.

Il faut s'assurer par exemple que tous les opérandes possèdent non seulement le même `USAGE`, mais aussi le même nombre de décimales

Les opérations en virgule flottante (`COMP-1`, `COMP-2`), souffrent en particulier des conversions de type.

Si des données binaires ou packed-decimal participent à une opération en virgule flottante, il faut veiller à ne pas dépasser 9 digits pour ces dernières afin d'accélérer leur conversion vers du flottant

2.7 Rendre plus efficace les grands exposants

En utilisant des virgules flottantes pour les grands exposants, on obtient une évaluation plus efficace et d'une plus grande précision.

Un exposant en virgule flottante force les calculs en arithmétique flottante.

Par exemple, ceci force un calcul en virgule flottante:

```
.         COMPUTE fixed-point1 = fixed-point2 ** 100000.E+00
```

et est plus efficace que cela (le gain pouvant aller jusqu'à 100%):

```
.         COMPUTE fixed-point1 = fixed-point2 ** 100000
```

2.8 Utiliser le PACKED DECIMAL signé avec un nombre impair de digits

Une zone comp-3 avec un nombre impair de digits (plus son signe) occupera toujours un nombre entier d'octets, alors qu'avec un nombre pair de digits, il subsiste toujours un demi-octet inutilisable devant être initialisé à zéro

Ainsi, si N est impair: S9(N) COMP-3 est 5 à 30% plus rapide que S9(N - 1) COMP-3

Si c'est possible, utiliser au plus 15 digits, afin d'éviter l'appel à des routines de calcul externes pour la multiplication et la division

Lorsque le signe est conforme à cobol ('C' positif, 'D' négatif), compiler NUMPROC(PFD) (en éliminant l'extra-code de rectification du signe) procure un gain de 20% (à éviter si ces données packed sont générées par du PL/1 ou de l'assembleur)

2.9 Optimiser les indices

Utiliser des variables binaires (COMP, COMP-4 ou COMP-5) pour tous les indices

Lorsque plusieurs indices interviennent simultanément dans l'adressage d'une table, essayer de placer le plus à droite les indices variant le plus souvent. Les indices variant peu ou les constantes d'indice devant être placés le plus à gauche.

Faire ses propres tests de validité d'indice et de débordement de table plutôt que de s'en remettre à l'option SSRANGE du compilateur. Cette option est d'une aide précieuse en phase de mise au point, mais n'a pas vocation à être activée en production.

Utiliser l'adressage relatif

Utiliser des index

Alors qu'un indice doit toujours être multiplié par la taille d'un élément de la table avant de pouvoir désigner l'emplacement physique d'une donnée, l'index procède par déplacement (addition) calculé à partir de l'adresse du début de la table. Un index est donc une adresse directement utilisable par le programme pour désigner un élément dans une table

Un index ne comporte pas de limite autre que la mémoire adressable par le compilateur (123 Mo et 2Go avec la prochaîne version cobol 5).

A la différence des indices binaires, il est insensible à l'option de compilation TRUNC( )

Cependant, les index sont plus difficiles à manipuler et à comprendre par le développeur :

- ils ne sont pas utilisables avec les opérateurs arithmétiques standards (+, -, *, /)
- ils n'ont pas de valeur « affichable » par DISPLAY en phase de débogage

2.10 Eviter le USAGE DISPLAY et COMP-3 pour les variables de boucles

A chaque fois qu'un indice usage display (encore appelé External Decimal) est utilisé par un programme, il est au préalable converti en packed decimal, manipulé ensuite en arithmétique décimale, puis enfin retraduit en usage display.

En cas d'usage intensif dans une boucle, le surcoût occasionné peut être très significatif

Performances comparées d'une variable de contrôle dans une boucle:

- le packed decimal (`COMP-3`) est 250% moins rapide que le `Binary`
- l' `USAGE DISPLAY` est 550% moins rapide que le `Binary`

..note

Mais le packed decimal reste cependant plus efficace pour les très grands entiers : `S9(18) COMP-3` est plus rapide que le `S9(18) Binary`

2.11 Compiler `NOSSR`, `OPT(STD)`, `TRUNC(OPT)`, `NUMPROC(PFD)`

Ces 4 options utilisées **conjointement** procurent le code le plus performant En effet, l'optimiseur peut faire des choix encore plus économiques après que TRUNC(OPT) ou NUMPROC(PFD) ait simplifié certaines instructions

2.12 Utiliser des zones binaires signées

Tout particulièrement si le programme est compilé `TRUNC(BIN)` ou contient des variables définies en `COMP-5`.

Dans ces 2 cas, les opérations sur des binaires signés sont 2 à 6 fois plus rapides que sur des binaires non signés

2.13 Coder un `DEPENDING ON` pour des tables «partiellement chargées»

Les `SEARCH` et `SEARCH ALL` sont plus performants sur les tables définies `OCCURS DEPENDING ON`. Cela est d'autant plus vrai avec des tables volumineuses se retrouvant que partiellement chargées dans le déroulement d'un programme.

Pour les SEARCH ALL, il devient alors inutile d'initialiser la table à HIGH-VALUE (avec `ASCENDING KEY`) ou `LOW-VALUE` (avec `DESCENDING KEY`) avant le chargement de la table

2.14 Economiser les appels aux routines systèmes date/time

On rencontre encore assez couramment des traitements batch qui accèdent à la date système à l'intérieur d'une boucle.

Les appels aux routines systèmes de date/time ne sont pas neutres et sans conséquences pour la machine et les autres traitements. Le code déroulé est plus lourd qu'il n'y paraît, il génère par exemple toujours une "interruption système". Il est donc indispensable de les limiter au strict minimum.

On a ici 6 appels consécutifs et inutiles à la date système :

```
.       MOVE Function Current-Date(5:2)   To MOIS-DJ
        MOVE Function Current-Date(1:4)   To ANNEE-DJ
        MOVE Function Current-Date(7:2)   To JOUR-DJ
        MOVE Function Current-Date(9:2)   To HEURE-DJ
        MOVE Function Current-Date(11:2)  To MINUTE-DJ
        MOVE Function Current-Date(13:2)  To SECONDE-DJ
```

qu'il conviendrait plutôt d'écrire ainsi :

```
.         Move Function Current-Date  To W-CURRENT-DATE
          Move W-CURRENT-DATE(1:4)    To ANNEE-DJ
          Move W-CURRENT-DATE(5:2)    To MOIS-DJ
          Move W-CURRENT-DATE(7:2)    To JOUR-DJ
          Move W-CURRENT-DATE(9:2)    To HEURE-DJ
          Move W-CURRENT-DATE(11:2)   To MINUTE-DJ
          Move W-CURRENT-DATE(13:2)   To SECONDE-DJ
```

2.15 Faciliter le travail de l'optimiseur cobol

L'optimiseur cobol est activé lorsque l'option `OPT(STD)` ou `OPT(FULL)` est positionnée.

- Elimine certains transferts de contrôle inutiles et certaines branches inefficientes (y-compris ceux générés par le compilateur lui-même)
- Simplifie les `PERFORM` et les `CALL nested program`
- Elimine les calculs redondants d'indices et de constantes
- Elimine certaines expressions conditionnelles
- Détruit le code mort

Afin d'augmenter les chances de réussite de l'optimiseur, respecter ces quelques principes

- programmer structuré à l'aide de `PERFORM`, `EVALUATE` et `nested PROGRAM`

  Les `nested PROGRAMS` sont transformés en cobol « inline » éliminant du coup le code de gestion des paramètres en « linkage
- Utiliser des variables symboliques
- regrouper tous les facteurs constants le plus à gauche des expressions arithmétiques et conditionnelles: `compute S = 1 + 2 + 10 ** (-x)`

  L'optimiseur parcours les expressions de gauche à droite à la recherche de constantes calculables une fois pour toute à la compilation
- Placer les moves de zones contiguës les uns à la suite des autres : Ils seront transformés en un move unique (`move CORRESPONDING`)
- Eviter les `PERFORM THRU`

  cela peut inhiber totalement l'optimiseur sur une partie du code :

  ```
  IGYOP3094-W There may be a loop from the "PERFORM" statement at "PERFORM (line 40.
  "PERFORM" statement optimization was not attempted.
  ```

- Redéfinir les variables binaires `S9(9)` en `S9(8)` quand c'est possible

  le `S9(8)` compilé avec `TRUNC(OPT)`, en générant du code plus simple que `S9(9)`, ouvre la voie à des optimisations supplémentaires (également valable pour le `packed-decimal` compilé `NUMPROC(PFD)`)

> ***Attention!***
>
> L'option `OPT(FULL)` détruit du programme toutes les variables non référencées dans la `PROCEDURE DIVISION`. Cela peut avoir un impact sur les zones servant de «Eyes catcher» recherchées par les utilitaires de contrôle/scan des loadmodules, ou lors de la

visualisation d'un dump

2.16 Déterminer le type de call le plus efficace

Performance des différents types de CALLs (coût du CALL seul):

- le CALL nested program est 60% plus rapide que la CALL statique
- le CALL statique est 30 à 45% plus rapide que le CALL dynamique «littéral» (CALL statique compilé `DYNAM`).
- Le CALL statique est 45 à 55% plus rapide que le CALL dynamique «identifier».
- le CALL dynamique «littéral» (compilé `DYNAM`) est 20 à 25% plus rapide le CALL dynamique «identifier».

2.17 Evitez la clause `IS INITIAL`

La clause `IS INITIAL` codée après le `PROGRAM-ID` spécifie qu'à chaque appel du programme (et de tous ceux qu'il contient), celui-ci est placé dans son statut initial ou de «premier appel» , cela se traduit par une réinitialisation complète de toutes les variables de la `WORKING-STORAGE` (application de toutes les clauses `VALUE` aux variables).

L'appel d'un programme contenant la clause `IS INITIAL` est de 600 à 1000% plus coûteux (coût de l'appel) que sans clause (dépendant de la taille de la WORKING).

2.18 Ne pas coder `BLOCK CONTAINS n RECORD` dans les `FD`

Coder `BLOCK CONTAINS 0` dans les programme afin de laisser au JCL le soin de déterminer le blocksize optimal pour chaque fichier.

On peut également activer l'option globale de compilation `BLOCK0` pour tous les programmes

2.19 Compiler `RENT`, `DATA(31)` et link-editer `AMODE(31)`, `RMODE(ANY)`

Et vérifier que les attributs des loadmodules soient conformes aux paramètres d'exécution du LE (Langage Environment) :

- `ALL31(ON)` et `HEAP(,,ANYWHERE)`

> ***Note***
>
> Pour connaître le paramètrage LE par défaut en vigueur sur la partition, il suffit d'exécuter un programme en lui passant le paramètre `/RPTOPTS(ON)` :

```
.        //STEP0001   EXEC PGM=PGMNAME,PARM='/RPTOPTS(ON)'
```

2.20 Compiler `RENT`, `DATA(31)` et link-editer `AMODE(31)`, `RMODE(ANY)`

2.21 A propos du paramètre de run time LE `CBLPSHPOP`

If this option is set ON, when program A CALLs program B then LE adds a "PUSH HANDLE" before the CALL to program B and does a "POP HANDLE" upon return to program A. Why? To save the current state of any HANDLE ABEND or HANDLE CONDITIONs that program A may have done. If program B resets those conditions, the POP HANDLE will restore A's conditions upon return. That process happens for every execution of every CALL statement.

That's pretty handy if program B contains EXEC CICS commands. However, way back when I was an application programmer we had to use EXEC CICS LINK to call programs that were going to do CICS commands. That may not be the best performing strategy today, but we have a lot of COBOL CICS code that still adheres to that philosophy. For such applications, if there is a CALL to program B, program B wouldn't be expected to have any CICS commands in it, so the PUSH/POP pair is unnecessary. In fact, it turns out that those PUSH/POP pairs can consume a fair bit of CPU, and CBLPSHPOP(ON) is known to cause potentially significant CPU consumption.1 Of course if a transaction doesn't do any CALLs then the state of CBLPSHPOP is moot.

2.22 Accéder aux fichiers VSAM de manière optimale

- `ACCESS IS DYNAMIC` est le mode optimal lorsque les enregistrements sont lus à la fois de façon aléatoire et de façon séquentielle
- `ACCESS IS RANDOM` est le mode optimal lorsque les enregistrements sont lus uniquement de façon aléatoire
- `ACCESS IS SEQUENTIAL` est le mode optimal lorsque les enregistrements sont lus uniquement de façon séquentielle
- Pour améliorer les performances des accès séquentiels on peut augmenter le nombre de buffers data (`BUFND`)
- Pour améliorer les performances des accès aléatoires on peut augmenter le nombre de buffers index (`BUFNI`)

```
.        //VSAM001 DD DISP=SHR,DSN=VSAM.FIC,AMP=('BUFNI=10,BUFND=20')
```

2.23 Compiler `TRUNC(OPT)` et favoriser le binary `S9(8)`

Format bin	Code machine généré avec TRUNC(OPT)	Performance relative

S9(1)→S9(4)	le compilateur génère des instructions pour un demi-mot (2 bytes) si possible	Le plus performant
S9(5)→S9(8)	le compilateur génère des instructions pour un mot machine (4 bytes) si possible	
S9(9)	le compilateur convertit la valeur en double-mot puis génère des instructions pour un double-mot (8 bytes)	40% moins performant que S9(8)
S9(10)→S9(17)	le compilateur génère des instructions pour un double-mot machine (8 bytes)	25% moins performant que S9(8)
S9(18)	le compilateur convertit la valeur en decimal packed puis génère des instructions pour du decimal packed	1000% moins performant que S9(8)

Note

En raison des conversions induites avec TRUNC(OPT) :

-S9(9) est moins performant que S9(10)

-S9(18) binary est moins performant que S9(18) comp-3

Le classement n'est pas identique avec TRUNC(BIN):

-S9(1) → S9(4) est le plus performant

-S9(5) → S9(9) est 45% plus lent que S9(4)

-S9(10) → S9(18) est 3000% plus lent que S9(4)

2.24 Utiliser des host variables binaires cobol adaptées

DB2 utilise les types `SMALLINT` et `INTEGER` sur la totalité de la longueur de leur représentation interne (2 ou 4 octets) avec les mêmes règles que Cobol quand l'option `TRUNC(BIN)` est positionnée. Les valeurs ainsi stockées peuvent donc largement dépasser le nombre de digits spécifié dans les PICTURE Cobol des programmes manipulant ces données.

Dans cet exemple, une colonne de type integer est mise à jour dans les limites imposées par DB2 qui peuvent dépasser les limites d'une zone binaire cobol compilée TRUNC(OPT) :

```
.              UPDATE TABLE
                  SET COL_int = COL_int * 1000
```

Si le programme est compilé `TRUNC(OPT)` ou `TRUNC(STD)`, il faut donc s'assurer que la taille des host-variables binaires destinées à stocker des colonnes de nombre entiers soit adaptée aux valeurs pouvant être transmises par DB2 :

- Pour les entiers `SMALLINT` pouvant dépasser 9999, utiliser du `S9(4) COMP-5`
- Pour les entiers `INTEGER` pouvant excéder 999 999 999, utiliser du `S9(9) COMP-5`

2.25 Options et paramètres qui peuvent affecter les performances

Options de compilation :

`ARITH, AWO, BLOCK0, DATA, DYNAM, FASTSRT, NUMPROC, OPTIMIZE, RENT, RMODE, SSRANGE, TEST,`

> ***Note***
>
> - Valeurs par défaut soulignées
> - Valeurs recommandées en rouge

- **ARITH(COMPAT/EXTEND) - AR(C/E)**

  Limitation des nombres décimaux à 18 ou 31 chiffres. Surcharge due à ARITH(EXTEND) environ égale à 1% mais dépendante de la quantité de données décimales

- **AWO/NOAWO**

  Passage au bloc suivant des fichiers de format VB selon la taille courante ou maximale de l'enregistrement. Surcharge due à NOAWO d'autant plus forte que le LRECL est proche du BLKSIZE

- **BLOCK0/NOBLOCK0**

  Blocage par défaut ou non des fichiers séquentiels. Surcharge due à NOBLOCK0 d'autant plus forte que le nombre de fichiers non explicitement bloqués est élevé

- **DYNAM/NODYNAM - DYN/NODYN**

  Chargement ou édition des liens des sous-programmes. Allongement des temps d'appel imputable à DYNAM allant de 40% à 100% de la durée initiale

- **FASTSRT/NOFASTSRT - FSRT/NOFSRT**

  Usage de DFSORT ou du tri COBOL. Réduction des temps due à FASTSRT atteignant 45% en cas de prépondérance des tris

- **NUMPROC(MIG/NOPFD/PFD)**

  Correction des signes des nombres décimaux en entrée ou en totalité ou pas du tout. Economie due à NUMPROC(PFD) environ égale à 1% mais exigeant des signes parfaitement normalisés

- **OPTIMIZE(FULL/STD)/NOOPTIMIZE - OPT(FULL/STD)/NOOPT**

  Optimisation plus ou moins avancée par suppression ou non des données sans référence ou pas d'optimisation. Economie due à OPTIMIZE(FULL) environ égale à 1% mais guère supérieure à celle due à OPTIMIZE(STD)

- **SSRANGE/NOSSRANGE - SSR/NOSSR**

  Vérification ou non du rang des indices. Surcharge due à SSRANGE pouvant atteindre 20% si la majorité des structures sont indicées

- **TEST(HOOK/NOHOOK,SEPARATE/NOSEPARATE - SEP/NOSEP,EJPD/NOEJPD)/NOTEST**

  Inclusion plus ou moins complète de facilités de détermination de problèmes ou non. Surcharge due à TEST pouvant largement dépasser 20% et à proscrire dans un environnement de production

- **THREAD/NOTHREAD**

  Capacité ou non à cohabiter dans une enclave avec des tâches PL/I ou POSIX. Surcharge des appels due à THREAD pouvant dépasser 45% et à proscrire sauf absolue nécessité

- **TRUNC(BIN/OPT/STD)**

  Troncature des données binaires en base 2 ou selon leur taille ou en base 10. Surcharge due à TRUNC(BIN) et TRUNC(STD) respectivement égales à 10% et 5% en moyenne

- **XMLPARSE(COMPAT/XMLSS) - XP(C/X)**

  Analyse XML par COBOL ou z/OS System Services. Surcharge due à XMLPARSE(COMPAT) supérieure à 20% pour de moindres fonctionnalités

Options de link-edit :

`AMODE, RMODE, RENT, REUS`

Options du Langage Environment :

`AIXBLD, ALL31, CBLPSHPOP, CHECK, DEBUG, INTERRUPT, RPTOPTS, RPTSTG, RTEREUS, SIMVRD`

`STORAGE, TEST, TRAP, VCTRSAVE`

`HEAP, ANYHEAP, BELOWHEAP, STACK, and LIBSTACK`

3 Mise en oeuvre du «Multi Rows» en cobol

3.1 Exemple de Multi Rows Fetch (MRF)

Action n° 1 : Adaptation du Declare Cursor

Ajout de la clause `WITH ROWSET POSITIONNING` dans le `DECLARE CURSOR`

```
.        Exec Sql
          DECLARE CUR-TIERS CURSOR WITH ROWSET POSITIONING
            FOR
            SELECT
              TIERS.GLOBAL_CPY_ID,
              TIERS.CPY_SHNM,
              ...
              C.CPY_DAT_CLOT_NOT,
              C.CPY_CD_NAF_REV
            FROM VLWAAKON TIERS
                 LEFT OUTER JOIN VLWB1FNN A
                  ON (A.IDSIRIS = TIERS.GLOBAL_CPY_ID)
                 LEFT OUTER JOIN VLWAANOT B
                  ON (B.GLOBAL_CPY_ID = TIERS.GLOBAL_CPY_ID)
                 LEFT OUTER JOIN VLWAACPY C
                  ON (C.GLOBAL_CPY_ID = TIERS.GLOBAL_CPY_ID)
            WITH UR
         End-exec
```

Action n° 2 : Déclaration des variables de gestion MRF

Déclarer un tableau de host-variables pour chaque colonne accédée

- HVA-CUR-TIERS

Déclarer un tableau de host-variables pour chaque indicateur de NULL, si besoin

- HVA-NULL-INDICATORS

Déclarer les variables de gestion MRF:

- RWS-COUNT-CUR-TIERS : compteur du nombre de lignes lues à chaque fetch
- RWS-IDX-CUR-TIERS : indice utilisé pour le parcours du tableau de host-variables
- RWS-MAX-CUR-TIERS : constante, taille maximale d'un rowset
- RWS-SIZE-CUR-TIERS : variable, taille du rowset choisie par le programme

```
.      *-----------------------------------------------------------------+
       *   Host Variable Array Null Indicators pour MRF:
       *-----------------------------------------------------------------+
        01 HVA-NULL-INDICATORS.
          10 HVA-STCRCA-IND-NULL       Pic S9(4) Binary Occurs 200.
          10 HVA-IND2                  Pic S9(4) Binary Occurs 200.
          ...
          10 HVA-IND39                 Pic S9(4) Binary Occurs 200.
          10 HVA-IND40                 Pic S9(4) Binary Occurs 200.
          10 HVA-IND41                 Pic S9(4) Binary Occurs 200.
```

```
      *-----------------------------------------------------------------+
      *   Host Variables Array définitions pour fetch MRF:
      *-----------------------------------------------------------------+
       01 HVA-CUR-TIERS.
         10 HVA-GLOBAL-CPY-ID      Pic S9(9) binary       Occurs 200.
         10 HVA-CPY-SHNM           Pic X(20)              Occurs 200.
         ...
         10 HVA-CPY-DAT-CLOT-NOT   Pic X(10)              Occurs 200.
         10 HVA-CPY-CD-NAF-REV     Pic X(6)               Occurs 200.
      *-----------------------------------------------------------------+
      *   MRF variables de gestion
      *-----------------------------------------------------------------+
       01 RWS-COUNT-CUR-TIERS   Pic S9(4) Binary Value 0.
       01 RWS-IDX-CUR-TIERS     Pic S9(4) Binary Value 0.
       01 RWS-MAX-CUR-TIERS     Pic S9(4) Binary Value 200.
       01 RWS-SIZE-CUR-TIERS    Pic S9(4) Binary Value 100.
```

Action n° 3 : Substitution de l'ordre `Exec Sql Fetch ...` par un `Perform Fetch-MRF`

```
.     *-----------------------------------------------------------------+
      * 2013/05/13 : Optimisations cpu (EROL Technologies)
      *   Fetch curseur CUR-TIERS en "MultiRowset Fetch"
      *-----------------------------------------------------------------+
            Perform FETCH-MRF-CUR-TIERS
            Evaluate SQLCODE
              When 0
            ...
```

Action n° 4 : Codage du paragraph `Fetch-MRF`

Ce paragraphe contient l'ordre `FETCH` MRF ainsi que toute la logique de gestion des `HVAs` et du `SQLCODE`

```
.      FETCH-MRF-CUR-TIERS.
      *====================*
      *-----------------------------------------------------------------+
      *   Gestion du MultiRowsetFetch curseur CUR-TIERS :
      *-----------------------------------------------------------------+
           If RWS-SIZE-CUR-TIERS > RWS-MAX-CUR-TIERS
             Move RWS-MAX-CUR-TIERS To RWS-SIZE-CUR-TIERS
           End-if
           If RWS-IDX-CUR-TIERS < RWS-COUNT-CUR-TIERS
             Add 1 To RWS-IDX-CUR-TIERS
             Move 0 To SQLCODE
            Else
             If RWS-IDX-CUR-TIERS = 0 Or
                RWS-COUNT-CUR-TIERS = RWS-SIZE-CUR-TIERS
               Exec Sql
                Fetch Next Rowset From CUR-TIERS
                For :RWS-SIZE-CUR-TIERS Rows
                Into
                  :HVA-GLOBAL-CPY-ID
                 ,:HVA-CPY-SHNM
                  ...
```

```
                ,:HVA-CPY-DAT-CLOT-NOT:HVA-IND40
                ,:HVA-CPY-CD-NAF-REV:HVA-IND41
             End-exec
              Move SQLERRD(3) To RWS-COUNT-CUR-TIERS
              If SQLCODE = 0 Or
                (SQLCODE = +100 And RWS-COUNT-CUR-TIERS > 0)
                Move 1 To RWS-IDX-CUR-TIERS
                Move 0 To SQLCODE
              End-if
            Else
              Move +100 To SQLCODE
            End-if
          End-if
          If SQLCODE = 0
            Move HVA-GLOBAL-CPY-ID(RWS-IDX-CUR-TIERS)
                 To GLOBAL-CPY-ID Of DCLVLWAAKON
            Move HVA-CPY-SHNM(RWS-IDX-CUR-TIERS)
                 To CPY-SHNM Of DCLVLWAAKON
            ...
            Move HVA-CPY-DAT-CLOT-NOT(RWS-IDX-CUR-TIERS)
                 To CPY-DAT-CLOT-NOT Of DCLVLWAACPY
            Move HVA-IND40(RWS-IDX-CUR-TIERS)
                 To IND40
            Move HVA-CPY-CD-NAF-REV(RWS-IDX-CUR-TIERS)
                 To CPY-CD-NAF-REV Of DCLVLWAACPY
            Move HVA-IND41(RWS-IDX-CUR-TIERS)
                 To IND41
          End-if
           .
```

Action n° 5 : Adaptation de l'Open Curseur (si réutilisé plusieurs fois)

Afin de permettre la réutilisation d'un curseur MRF en cours de programme, il est **indispensable** de réinitialiser la variable de parcours du tableau HVA (ici `RWS-IDX-CUR-TIERS`)

```
.         Move Zero To RWS-IDX-CUR-TIERS
          Exec Sql
           OPEN CUR-TIERS
          End-exec
```

4 Mise en oeuvre de tables «préloadées» et «mémoïsées» en cobol

4.1 Performances comparées de diverses méthodes d'accès en table

Une table DB2 contenant 10 000 lignes est accédée par DB2 suivant 3 méthodes SQL différentes

Cette même table est ensuite chargée en mémoire et accédée suivant 3 méthodes de recherche. Les temps de chargement de la table en mémoire ne sont pas pris en compte et toutes les lignes de la table sont lues chaque fois :

méthode	cpu mesurée pour 10 000 accès
Exec Sql SELECT	10
Exec Sql FETCH	5
Exec Sql FETCH MRF (rowset de 100 lignes)	3
SEARCH COBOL (recherche séquentielle)	35
SEARCH ALL COBOL (recherche dichotomique)	0,4
HASHCODE (fonction hash écrite en ASM)	0,05

4.2 Préchargement en table cobol avec recherche dichotomique (préload)

Cette technique utilise une méthode de recherche dichotomique en table en s'appuyant sur l'instruction cobol `SEARCH ALL` Il ne s'agit pas à proprement parlé de mémoïsation, mais plutôt d'une anticipation des données qui vont être accédées par le programme

- La totalité des données doit être au préalable chargée dans la table en mémoire
- Les données doivent être triées dans la table suivant la clé servant de recherche
- La table doit être accédée au moins pour 50 % du nombre de ses lignes afin d'amortir le coût du préchargement
- Le temps de recherche n'augmente que faiblement avec la taille de la table (ordre de grandeur : O(log n))

Exemple : Remplacement d'un accès VSAM KSDS par un accès en table avec recherche dichotomique

```
.      *-------------------------------------*
         L600-LECTURE-VSAM-MTA   SECTION.
       *-------------------------------------*
            MOVE 123456 TO MTA-ALTYNA OF MTA-ENR
            READ KV9124L
              INVALID KEY DISPLAY 'TYPE ALIAS INCONNU DANS RICOS   '
                          DISPLAY ' ALTYNA :' MTA-ALTYNA
              PERFORM ARRET-PROG
            END-READ.
```

Action n° 1 : Substitution de l'ordre READ dans le programme principal

Le `READ` est remplacé par un `CALL` "nested program" avec passage de 3 paramètres: un code fonction de lecture (READ), la description de l'enregistrement du fichier ainsi qu'un code retour

La clé d'accès doit préalablement être chargée dans la variable correspondante de l'enregistrement (`MTA-ALTYNA`)

```
.           Move 123456 To MTA-ALTYNA Of MTA-ENR
            CALL 'SKV9124L' using by content    'READ'
                                  by reference MTA-ENR
                                  by reference RC-SKV9124L
            IF RC-SKV9124L   NOT = ZERO
               DISPLAY 'TYPE ALIAS INCONNU DANS RICOS   '
               DISPLAY ' ALTYNA :' MTA-ALTYNA
               PERFORM ARRET-PROG
            END-IF.
```

Action n° 2 : Construction du "nested program" par ``COPY Replacing``

1. Le module doit être déclaré à l'intérieur du programme principal et à la fin de celui-ci, il est construit à l'aide de 3 copy book
2. **Program-id xxxxxxxx.** → le nom du module « nested program »
3. **Copy SEARCVSF Replacing** → copy book déclaration de la cobol `FILE SECTION`:
   - **:VSAMFILE:** → nom cobol du fichier VSAM
   - **:VSAMKEY:** → nom cobol de la zone clé de l'enregistrement
   - **:VSAMRECORD:** → nom cobol de l'enregistrement
4. Description complète de l'enregistrement du fichier
5. **Copy SEARCVSW Replacing** → copy book déclaration de la working cobol:
   - **:RESSOURCE:** → nom de la ressource VSAM gérée (pour affichage uniquement)
   - **:MAX-SIZE:** → taille de la table (à déterminer manuellement)
   - **:PICTURE-KEY:** → picture cobol cobol de la clé d'accès
6. **Linkage Section** → compléter la description de `PARM-RECORD` avec la description de l'enregistrement
7. **HASH-TABLE-KEYS** → cette zone groupe contient les sous-zones clé de la table hascodée, il faut décrire chacune des colonnes clé à l'identique des zones du DCLGEN
8. **HASH-TABLE-DATA** → cette zone groupe contient les sous-zones de données de la table hascodée, il faut y décrire chacune des colonnes à stocker à l'identique des zones du DCLGEN
9. **Copy SEARCVSP Replacing** → renseigner les pseudo-variables suivantes :
   - **:VSAMFILE:** → nom cobol du fichier VSAM
   - **:VSAMKEY:** → nom cobol de la zone clé de l'enregistrement
   - **:STARTKEY:** → valeur de début de clé pour le `START` de positionnement de la première lecture de chargement
   - **:VSAMRECORD:** → nom cobol de l'enregistrement

- **:MOVESIN:** → l'ensemble des `MOVE` cobol des zones de l'enregistrement vers la table cobol (pour le chargement des données de la table)
- **:MOVESOUT:** → l'ensemble des `MOVE` cobol des zones de données de la table vers l'enregistrement lu (lecture des données de la table)

10. Ajouter un `End Program "Program principal."` si celui-ci n'en possède pas

```
.       Identification Division.
        Program-id. SKV9124L.
       *file section template:
        COPY SEARCVSF REPLACING
             ==:VSAMFILE:==   BY ==KV9124L==
             ==:VSAMKEY:==    BY ==MTA-CLE==
             ==:VSAMRECORD:== BY ==MTA-ENR==.
        01 MTA-ENR.
          03 MTA-CLE.
            05 MTA-ALTYNA               PIC X(12).
          03 MTA-DATA.
            05 MTA-ALTYDE               PIC X(50).

       *working-storage template:
        COPY SEARCVSW REPLACING
             ==:RESSOURCE:==     BY =="KV9124L"==
             ==:MAX-SIZE:==      BY ==1000==
             ==:PICTURE-KEY:==   BY ==Pic X(12)==.

        LINKAGE SECTION.
        01 SEARCH-FUNC           PIC X(8).
        01 PARM-RECORD.
          03 MTA-CLE.
            05 MTA-ALTYNA        PIC X(12).
          03 MTA-DATA.
            05 MTA-ALTYDE        PIC X(50).
        01 RC                    PIC S9(9) comp.
        01 SEARCH-TAB.
          02 SEARCH-LINE OCCURS 10000 DEPENDING ON VSAM-LINES
             ASCENDING KEY MTA-CLE INDEXED BY INDX.
            03 MTA-CLE.
              05 MTA-ALTYNA                PIC X(12).
            03 MTA-DATA.
              05 MTA-ALTYDE                PIC X(50).

       *procedure division template:
        COPY SEARCVSP REPLACING
             ==:VSAMFILE:==    BY ==KV9124L==
             ==:VSAMKEY:==     BY ==MTA-CLE==
             ==:STARTKEY:==    BY ==Low-Value==
             ==:VSAMRECORD:==   BY ==MTA-ENR==
             ==:MOVESIN:==     BY ==
                MOVE MTA-CLE OF MTA-ENR
                  TO MTA-CLE OF SEARCH-LINE(INDX)
                MOVE MTA-DATA   OF MTA-ENR
                  TO MTA-DATA   OF SEARCH-LINE(INDX)==

             ==:MOVESOUT:==   BY ==
                MOVE MTA-CLE OF SEARCH-LINE(INDX)
```

```
                TO MTA-CLE OF PARM-RECORD
             MOVE MTA-DATA   OF SEARCH-LINE(INDX)
                TO MTA-DATA   OF PARM-RECORD==.
       End Program SKV9124L.
       END PROGRAM KV9120.
```

4.3 Chargement à la volée en table hashcodée avec accès direct (mémoïsation)

Cette technique utilise une méthode d'accès direct en table hashcodée. Une fonction de hashage appliquée sur chaque clé calcule un indice permettant un accès au données stockées en table.

En cas de collision entre 2 clés (la fonction renvoit le même indice pour 2 clés différentes) , un "double hash" est effectué sur la clé jusqu'à déterminer un nouvel emplacement libre dans la table.

Les performances sont essentiellement liées à la rapidité de la fonction de hashage et de ses facultés à bien répartir les clés dans la table (bon "effet d'avalanche" tout en minimisant les collisions)

La table est gérée à la façon d'une mémoire cache :

- Les données non déjà présentes en mémoire sont stockées en table au fur et à mesure de leur lecture dans DB2
- La taille de la table mémoire ne constitue pas un frein au fonctionnement car une fraction seulement des données DB2 peut être chargée en mémoire, si la table est saturée, les accès sont alors effectués "normalement" dans DB2 (sans gain, mais malgé tout sans dégradation)
- Pas de surcoût lié à un préchargement des données en table, le gain est acquis dès la "seconde lecture" d'une même donnée
- Un surcoût en occupation mémoire: la table doit être définie "légèrement plus grande" que l'ensemble des données à y stocker
- Le temps de recherche reste constant quelle que soit la taille de la table
- Il faut au préalable estimer une taille optimale pour la table (de 1 à 1,5 fois)

Exemple : Remplacement d'un accès DB2 par un accès en table hashcodée (memoïsée)

```
.       Exec sql
           SELECT  CODE_ENTITE_CRR,
                   CODE_APPLI_CRR
           INTO :DCLVLWBRCON.CODE-ENTITE-CRR,
                :DCLVLWBRCON.CODE-APPLI-CRR
           FROM   VLWBRCONAPSI
           WHERE  BAAPID = :DCLVLWBRCON.BAAPID
           WITH UR
        End-exec
```

Action n° 1 : Substitution de l'ordre DB2 dans le programme principal

`L'EXEC SQL` est remplacé par un `CALL` "nested program" avec passage de 2 paramètres: un code fonction de lecture (READ) et la description de la table (DCLGEN)

La clé d'accès doit préalablement être chargée dans la variable correspondante du `DCLGEN` (ici `BAAPID`)

```
.           Move '0123456' To BAAPID Of DCLVLWBRCON
            Call "SLWBAKON" Using By content    "READ"
                                By reference DCLVLWBRCON
            EVALUATE SQLCODE
            WHEN 0 ...
```

Au retour du module, le `SQLCODE` continue à être testé comme auparavant (le programme n'a subit d'autre modification que le remplacement de `l'EXEC SQL` par un `CALL`)

> ***Attention!***
>
> Vérifier la présence d'éventuelles directives de précompilation `SQL WHENEVER` en amont du programme, car elles n'agissent plus à cet endroit du code. Il est impératif dans ce cas d'ajouter les tests manquants sur le `SQLCODE`

Action n° 2 : Construction du "nested program" par ``COPY Replacing``

1. Le module doit être déclaré à l'intérieur du programme principal et à la fin de celui-ci, il est construit à l'aide de 2 copy book
2. **Program-id xxxxxxxx.** → le nom du module « nested program »
3. **Copy HASHTB2W.** → copy book déclaration du début de la `WORKING-STORAGE`
4. **RESSOURCE-NAME** → nom de la ressource DB2 gérée (pour affichage uniquement)
5. **HASH-PRIME-SIZE** → taille de la table (il faut choisir un nombre premier pour des raisons de performance, sinon le module refuse de fonctionner)
6. **HASH-LOOP-MAX** → nombre max de tentatives de « double hash » en cas de collision de clé. Plus ce nombre est élevé, plus la table sera remplie de façon optimale, mais avec une dégradation des temps d'accès. Si la table est très volumineuse, et l'encombrement mémoire une contrainte, cette valeur peut être portée jusqu'à 100 (pour atteindre un taux de remplissage > à 90%), sinon prendre une valeur < 10 et une taille égale à 1,5 fois le nombre de lignes estimées
7. **Linkage Section** → ajouter le copy du `DCLGEN` de la table (si ce `DCLGEN` est déjà utilisé dans le programme principal et s'il contient un `DECLARE table`, il est nécessaire de substituer le nom de la table par un nom inexistant par ailleurs, sinon le précompilateur ne l'accepte pas). Le reste de la linkage ne doit pas être modifié (`HASH-FUNC` et entête de `HASH-TABLE`)
8. **HASH-TABLE-KEYS** → cette zone groupe contient les sous-zones clé de la table hascodée, il faut décrire chacune des colonnes clé à l'identique des zones du DCLGEN
9. **HASH-TABLE-DATA** → cette zone groupe contient les sous-zones de données de la table hascodée, il faut y décrire chacune des colonnes à stocker à l'identique des zones du DCLGEN

10. **Copy HASH1B2P Replacing** → renseigner les pseudo-variables suivantes :
    - **:DCLGEN:** → nom cobol du DCLGEN de la table DB2
    - **:LOADKEYS:** → l'ensemble des `MOVE` cobol des zones clés du `DCLGEN` vers la clé de la table hashcodée (chargement de la clé)
    - **:DATATOTABLE:** → l'ensemble des `MOVE` cobol des zones de données du `DCLGEN` vers la table hashcodée (chargement des données de la table)
    - **:DATAFROMTABLE:** → l'ensemble des `MOVE` cobol des zones de données de la table vers le `DCLGEN` (lecture des données de la table)
11. Avant le `End Program "nested module".` → `l'EXEC SQL` de lecture de la table DB2 qui a été substitué dans le programme principal
12. Ajouter un `End Program "Program principal."` si celui-ci n'en possède pas

> ***Note***
>
> Si les 2 pseudo-variables `:DATATOTABLE:` et `:DATAFROMTABLE:` sont substituées par " " (SPACES), aucune donnée ne sera stockée en table, à l'exception de la clé et du SQLCODE. Dans ce cas, le module se renvoi uniquement un `SQLCODE` (fonctionnement en mode "test d'existence" d'une valeur en table)
>
> `HASH-LINE Occurs 100000` est une limite arbitraire qui peut être augmentée (ou diminuée) au besoin. Il faut savoir que cette valeur est celle testée (calcul de `size * longueur`) par le compilateur pour vérifier si la limite des 123 Mo adressables par cobol est atteinte ou non (génère un message d'erreur à la compilation en cas de dépassement de capacité).

```
.       Identification Division.
        Program-id. SLWBRCON.
       *Working-Storage copy book:
        Copy HASHTB2W.
        01 RESSOURCE-NAME      PIC X(8) Value "TLWBRCON".
        01 HASH-PRIME-SIZE     PIC S9(8) binary Value 4001.
        01 HASH-LOOP-MAX       PIC S9(8) binary Value +10.
        Linkage Section.
        01 HASH-FUNC           PIC X(4).
        copy LWBRCON Replacing VLWBRCONAPSI By DUMMY02.
        01 HASH-TABLE.
          02 HASH-LINE Occurs 100000 Depending On HASH-PRIME-SIZE.
            03   HASH-TABLE-KEYS.
              05 BAAPID              PIC X(12).
            03 HASH-TABLE-DATA.
               05 CODE-ENTITE-CRR     PIC X(5).
              05 CODE-APPLI-CRR      PIC X(12).
       *Procedure Division copy book:

            Copy HASH1B2P Replacing
             ==:DCLGEN:==          By ==DCLVLWBRCON==
             ==:LOADKEYS:==        By ==
                Move BAAPID Of DCLVLWBRCON
                To BAAPID Of HASH-KEYS==
```

```
           ==:DATATOTABLE:==     By ==
              Move CODE-ENTITE-CRR OF DCLVLWBRCON
                To CODE-ENTITE-CRR OF HASH-TABLE-DATA(HASH)
              Move CODE-APPLI-CRR OF DCLVLWBRCON
                To CODE-APPLI-CRR OF HASH-TABLE-DATA(HASH)==

           ==:DATAFROMTABLE:==   By ==
              Move CODE-ENTITE-CRR OF HASH-TABLE-DATA(HASH)
                To CODE-ENTITE-CRR OF DCLVLWBRCON
              Move CODE-APPLI-CRR OF HASH-TABLE-DATA(HASH)
                To CODE-APPLI-CRR OF DCLVLWBRCON==.
          Exec sql
            SELECT   CODE_ENTITE_CRR,
                     CODE_APPLI_CRR
            INTO :DCLVLWBRCON.CODE-ENTITE-CRR,
                 :DCLVLWBRCON.CODE-APPLI-CRR
            FROM   VLWBRCONAPSI
            WHERE   TPS_DT_ARRT = :DCLVLWBRCON.TPS-DT-ARRT
              AND   BAAPID      = :DCLVLWBRCON.BAAPID
            WITH UR
         End-exec
                .
      End Program SLWBRCON.
      End Program LWEMIEXM.
```

Le module dispose en interne de 3 autres fonctionnalités :

Une fonction statistique, permettant de surveiller le bon fonctionnement du module

```
.         Call "modulexx" Using By content   "STAT"
                                 By reference DCLxxx
```

Elle produit un certains nombre d'informations utiles sur le bon fonctionnement du module (nb d'appels, taux de remplissage, collisions,...) :

```
.      * Ressource name       :HASH0002
       * Hash table lines     :00000811
       * Taux de remplissage  :97%
       * Hash table size      :000089210 (bytes)
       * Hash factor limite   :00000027
       * Key len              :00000008
       * Module Call          :00002407
       * Exec SQL Call        :00000833
       * clés hashcodées      :00000787
       * clés rejetées        :00000046
       * Sqlcode rejetés      :00000000
       * Hash zero value      :00000000
       * collisions (hash-loop 00000000) :00000916
       * collisions (hash-loop 00000001) :00000637
       * collisions (hash-loop 00000002) :00000487
```

Une fonction de «fin d'utilisation» du module, permettant de libérer au plus tôt la mémoire occupée par la table (si le programme principal le nécessite, pour d'autres traitements s'enchaînant à la suite par exemple). Par défaut, la mémoire est libérée automatiquement à l'arrêt du programme principal.

```
.        Call "modulexx" Using By content   "CLOS"
                             By reference DCLxxx
```

Une fonction de «visualisation» du contenu de la table (DUMP), permettant de déboguer éventuellement les accès effectués par le module

```
.       Call "modulexx" Using By content   "DUMP"
                             By reference DCLxxx
```

5 Annexes

5.1 Interprétation du contenu de la PLAN_TABLE DB2

METHOD:

- **0:** First table accessed, continuation of previous table accessed or not used
- **1:** Nested Loop Join
- **2:** Merge Scan Join
- **3:** Sorts required by ORDER BY, GROUP BY, SELECT DISTINCT, UNION
- **4:** Hybrid Join

ACCESSTYPE:

- **I:** By an Index
- **I1:** One fetch Index Scan
- **M:** Multiple index scan
- **MX:** By Index mentioned in ACCESSNAME
- **MI:** Intersection of Multiple indexes
- **MU:** Union of multiple indexes
- **N:** Index scan when matching predicated in IN keyword
- **R:** Tablespace scan
- **RW:** Work file scan of a materialized user defined table funtion
- **T:** By a spare index ; Star join work files
- **V:** By buffers for an INSERT statement within a SELECT
- **Blank:** NA

INDEXONLY:

- **Y/N:** Whether access to an Index alone is enough to carry out the step

---

TSLOCKMODE:

IS:	Intent Share Lock
IX:	Intent Exclusive lock
S:	Share Lock
U:	Update Lock
X:	Exclusive Lock
SIX:	Share with Intent Exclusive lock
N:	UR Isolation: No Lock
NS:	For CS, RS, RR an S Lock
NIS:	For CS, RS, RR an IS Lock
NSS:	For CS, RS an IS Lock and for RR an S Lock
SS:	For UR, CS, RS an IS Lock and for RR an S Lock

PREFETCH:

S:	Pure Sequential
L:	thru a page List
D:	Optimizer expects dynamic prefetch
Blank:	Unknown at bind time or NA

ACCESS_DEGREE:

n:	Number of Parallel tasks or operations activated by a Query
0:	if there is a host variable

PARALLELISM_MODE:

I:	Query I/O parallelism
C:	Query CP parallelism
X:	Sysplex query parallelism

PAGE_RANGE:

Y:	Yes;Whether the table qualifies for page-range(scan only the partitions those are needed)
Blank:	No

JOIN_TYPE:

F:	Full Outer Join
L:	Left Outer Join
S:	Star Join
Blank:	Inner Join or No join
Note:	Right Outer Join always converted to Left Outer Join

WHEN_OPTIMIZE:

Blank:	At bind time, using a default filter factor for any host variables, parameter markers or special registers
B:	Above facts + Bind option REOPT (ALWAYS) or REOPT (ONCE) must be specified.
R:	At Runtime, using input variables, parameter markers or special registers. Bind option REOPT (ALWAYS) or REOPT (ONCE) must be specified.

QBLOCK_TYPE:

SELECT:	Select
INSERT:	Insert
UPDATE:	Update
DELETE:	Delete
SELUPD:	Select with FOR UPDATE OF
DELCUR:	DELETE WHERE CURRENT OF CURSOR
UPDCUR:	UPDATE WHERE CURRENT OF CURSOR
CORSUB:	Correlated Subquery
NCOSUB:	NonCorrelated Subquery
TABLEX:	Table Expression
TRIGGER:	WHEN caluse on CREATE TRIGGER
UNION:	Union
UNIONA:	Union All

PRIMARY-ACCESSTYPE:

D:	Direct Row access
Blank:	No Direct Row access

TABLE_TYPE:

B:	Buffers for an INSERT stament within a SELECT
C:	Common Table Expression
F:	Table Function
M:	Materialized Qery Table
Q:	Temp intermediate table(Not Materialized), name of the viewor nested table expression, Contains a UNION ALL where materialization was vertual not actual.
RB:	Recursive Common Table Expression
T:	Table
W:	Work file (Materialized)

TABLE_ENCODE:

- **A:** ASCII
- **E:** EBCDIC
- **U:** Unicode
- **M:** when multiple CCSID is in one table

5.2 Recherche d'index *obsolètes ou inutiles*

5.2.1 Recherche d'index non utilisés depuis 1 an

```
.         Exec Sql
               SELECT
                 DBNAME
                ,LASTUSED
                ,SUBSTR(NAME,1,18) NAME18
                ,SUBSTR(CREATOR,1,18) CREATOR18
                ,REORGLASTTIME
                ,TOTALENTRIES
                ,SPACE
                ,REORGINSERTS
                ,REORGDELETES
                ,(SELECT SUBSTR(TBNAME,1,18)
                   FROM SYSIBM.SYSINDEXES
                       WHERE NAME = IXS.NAME
                         AND CREATOR = IXS.CREATOR
                  ) TBNAME
                ,COALESCE (
                    (SELECT SUBSTR(DNAME,1,8)
                     FROM SYSIBM.SYSPACKDEP
                     WHERE BNAME = IXS.NAME
                     FETCH FIRST 1 ROW ONLY
                    ), '*NO PACKAGE*'
                   ) ONE_PACKAGE
                 ,(SELECT UNIQUERULE
                    FROM SYSIBM.SYSINDEXES IX
                     WHERE IX.NAME = IXS.NAME
                         AND IX.CREATOR = IXS.CREATOR
                    ) UNIQUERULE
                 ,PARTITION
                    FROM SYSIBM.SYSINDEXSPACESTATS IXS
                    WHERE (LASTUSED IS NULL OR
                           LASTUSED <= CURRENT DATE - 1 YEAR)
                     AND TOTALENTRIES > 0
               ORDER BY DBNAME,NAME,PARTITION
               WITH UR;
.         End-exec
```

5.2.2 Recherche d'index dupliqués

```
.          Exec Sql
              SELECT * FROM SYSIBM.SYSINDEXES A
                 INNER JOIN SYSIBM.SYSINDEXES B
                  ON (    A.TBNAME = B.TBNAME
                      AND A.TBCREATOR = B.TBCREATOR
                      AND A.COLCOUNT = B.COLCOUNT
                      AND NOT ( A.NAME = B.NAME AND A.CREATOR = B.CREATOR )
                     )
              WHERE A.COLCOUNT =
                (SELECT COUNT(*) FROM
                 SYSIBM.SYSKEYS C
                 WHERE C.IXNAME = B.NAME
                  AND C.IXCREATOR = B.CREATOR
                  AND COLNAME CONCAT CAST(COLSEQ AS CHAR(3))
                              CONCAT ORDERING
                  IN (SELECT COLNAME CONCAT CAST(COLSEQ AS CHAR(3))
                                     CONCAT ORDERING
                      FROM SYSIBM.SYSKEYS D
                      WHERE D.IXNAME = A.NAME
                      AND D.IXCREATOR = A.CREATOR )
                     )
              GROUP BY A.DBNAME,A.CREATOR,A.TBNAME,A.NAME
              ORDER BY A.DBNAME ,A.TBNAME,A.FULLKEYCARD
.          End-exec
```

5.3 Mesure Cpu en CICS

```
.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DFHMNTDS-POINTER POINTER.
       01  WORKUSED PIC S9(9) BINARY.
       01  LASTUSED PIC S9(9) BINARY.
       01  CPUUSAGE PIC S9(9) BINARY.
       LINKAGE SECTION.
       01  DFHMNTDS.
         02  FILLER PIC X(1264).
         02  CPUTIME PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       ...
       MOVE 0 TO LASTUSED
       EXEC CICS
         COLLECT STATISTICS SET(DFHMNTDS-POINTER)
         MONITOR(EIBTASKN) NOHANDLE
       END-EXEC
       IF EIBRESP = 0
         SET ADDRESS OF DFHMNTDS TO DFHMNTDS-POINTER
         COMPUTE WORKUSED = CPUTIME / 62.5
         COMPUTE CPUUSAGE = WORKUSED - LASTUSED
         MOVE WORKUSED TO LASTUSED
       ELSE
         MOVE 0 TO WORKUSED
         MOVE 999999999 TO CPUUSAGE
       END-IF
       ... ALL THE WORK I WANT TO MEASURE
       EXEC CICS
         COLLECT STATISTICS SET(DFHMNTDS-POINTER)
         MONITOR(EIBTASKN) NOHANDLE
       END-EXEC
       IF EIBRESP = 0
         SET ADDRESS OF DFHMNTDS TO DFHMNTDS-POINTER
         COMPUTE WORKUSED = CPUTIME / 62.5
         COMPUTE CPUUSAGE = WORKUSED - LASTUSED
         MOVE WORKUSED TO LASTUSED
       ELSE
         MOVE 0 TO WORKUSED
         MOVE 999999999 TO CPUUSAGE
       END-IF
       DISPLAY 'MEASURED CPU CONSUMPTION IN 1/1000 SECONDS = ' CPUUSAGE
       ...
```

5.4 Module HASHDB2 (mémoïsation DB2)

Copy book HASDB2W0 (working storage template)

```
      *================================================================*
      *    ,--------------------,                                      *
      *   /  EROL Technologies /                                       *
      *  '--------------------'                                        *
      *                                                                *
      *================================================================*
```

```
       Data Division.
       Working-storage Section.
       01 HEAPID                  Pic S9(9) Binary Value Zero.
       01 MSIZE                   Pic S9(9) Binary.
       01 ADDR-STORAGE            Pointer          Value Null.
       01 FC.
         03 CONDITION-TOKEN-VALUE.
           88 CEE000          Value X'0000000000000000'.
           04 CASE-1-CONDITION-ID.
             05 SEVERITY          Pic S9(4) Binary.
             05 MSG-NO            Pic S9(4) Binary.
           04 CASE-2-CONDITION-ID Redefines CASE-1-CONDITION-ID.
             05 CLASS-CODE        Pic S9(4) Binary.
             05 CAUSE-CODE        Pic S9(4) Binary.
           04 CASE-SEV-CTL        Pic X.
           04 FACILITY-ID         Pic XXX.
         03 I-S-INFO              Pic S9(9) Binary.
       01  ABDCODE                Pic S9(9) Binary Value +16.
       01  TIMING                 Pic S9(9) Binary Value 1.

       01 TABLE-STATUS            Pic X Value Spaces.
         88  NOT-ALLOCATED  Value " ".
         88  ALLOCATED      Value "A".

       01 LOOKUP-STATUS           Pic X Value Spaces.
         88  START-LOOKUP   Value " ".
         88  END-LOOKUP     Value "E".

       01 PRIME-TEST              Pic X Value Spaces.
         88  IS-PRIME       Value " ".
         88  IS-NOT-PRIME   Value "N".

       01 MESSAGE-PLAY            Pic X Value Spaces.
         88 MESSAGE-WARNING Value "Y".

       01 NUM                     PIC S9(8) binary.
       01 N1                      PIC S9(8) binary.
       01 R1                      PIC S9(8) binary.
       01 I                       Pic S9(8) Binary.
       01 I3                      Pic S9(8) Binary.

       01 HASH                    Pic S9(8) Binary.
       01 KEY-LEN                 Pic S9(8) Binary.
       01 HASH-FACTOR             Pic S9(8) Binary.

       01 KEY-CPT                 PIC S9(8) binary Value 0.
       01 MODULE-CALL             Pic S9(8) Binary value 0.
       01 EXEC-SQL-CPT            Pic S9(8) Binary value 0.
       01 HASH-ECHEC-CPT          PIC S9(8) binary Value 0.
       01 HASH-ZERO-CPT           PIC S9(8) binary Value 0.
       01 SQLCODEX                PIC +999.
       01 TAUX                    PIC S9(3)V99 Comp-3.
       01 TAUX-X                  PIC Z99.
       01 KEYNUMERIC              PIC 9(18).
       01 SQLCOD-000              PIC S9(8) binary Value 0.
```

```
       01 SQLCOD-100          PIC S9(8) binary Value 0.
       01 SQLCOD-811          PIC S9(8) binary Value 0.
       01 SQLCOD-XXX          PIC S9(8) binary Value 0.
       01 COLLISIONS-TAB.
         05 COLLISIONS-CPT Occurs 1000 Pic S9(8) binary Value 0.
```

Copy book HASDB2P0 (procedure division template)

```
           03 HASH-TABLE-SQLCODE  Pic S9(4) comp.
           03 HASH-SLOT redefines HASH-TABLE-SQLCODE Pic X(2).

      *================================================================*
      *   ,--------------------,
      *  /  EROL Technologies /  Octobre 2013
      * '--------------------'
      *
      *  Nested modules de chargement à la volée de tables DB2
      *
      *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
      *
      *  -  Les clés non présentes en table hascodée sont lues dans
      *     DB2 puis stockées en table (ainsi que le SQLCODE)
      *     Les données accédées une Nième fois sont restituées à
      *     partir de la table hashcodée (+sqlcode) par un accès direct
      *     grace à l'indice calculé par une fonction de hashage
      *
      *  - HASH-TABLE-SIZE détermine la taille (lignes) de la table,
      *    cette valeur est augmentée de 50% et ajustée à un nombre
      *    premier, la table est allouée dynamiquement dès le 1er accès
      *    Si cette valeur est trop petite, certains accès seront
      *    systématiquement réorientés vers DB2
      *    Le taux de remplissage varie en fonction de la nature des
      *    clés
      *
      *  - HASH-LOOP-MAX détermine le nombre de tentatives d'insertions
      *    en table en cas de collision de clé. Plus ce chiffre est
      *    élevé, meilleur sera le taux de remplissage de la table,
      *    mais avec une dégradation perceptible des performances
      *
      *  - RESSOURCE-NAME sert uniquement lors de l'émission de
      *    messages d'anomalie (personnalisable)
      *----------------------------------------------------------------+
       Procedure Division Using HASH-FUNC
                                :DCLGEN:.
           Evaluate HASH-FUNC
           When "READ"
             If NOT-ALLOCATED
               Move Length Of HASH-KEYS
                 To KEY-LEN
               Perform ALLOCATION-TABLE
             End-if
             Add 1 To MODULE-CALL
             Perform HASH-TABLE-LOOKUP

           When "STAT"
             If ALLOCATED
```

```
            Perform STATISTIQUES-TABLE
          Else
            Display "<ERROR> HASH-TABLE Not allocated"
            Display " Ressource name :"  RESSOURCE-NAME
            Call "CEE3ABD" Using abdcode timing
          end-if
        When "CLOS"
          If ALLOCATED
            Perform DESALLOCATION-TABLE
          end-if

        When "DUMP"
          If ALLOCATED
            Perform DUMP-HASHTABLE
          Else
            Display "<ERROR> HASH-TABLE Not allocated"
            Display " Ressource name :"  RESSOURCE-NAME
            Call "CEE3ABD" Using abdcode timing
          end-if

        When OTHER
          Display "<ERROR> Invalide hash function:" HASH-FUNC
          Display " Ressource name :"  RESSOURCE-NAME
          Display " Valide function : READ | STAT | CLOS | DUMP"
          Call "CEE3ABD" Using abdcode timing
        end-evaluate
        Goback
        .

       HASH-TABLE-LOOKUP.
      *=================*
           :LOADKEYS:
           Set START-LOOKUP To True                                        00
           Perform Varying HASH-FACTOR From 0 By 1 Until END-LOOKUP        00
             Call "HASHSUM2" Using  HASH-KEYS
                                    KEY-LEN
                                    HASH-TABLE-SIZE
                                    HASH
                                    HASH-FACTOR
             Evaluate True
             When HASH = 0
      *      garde-fou (la fonction hash ne devrait jamais rendre 0)
               Add 1 to HASH-ECHEC-CPT HASH-ZERO-CPT
               Perform EXEC-DB2-REQUEST
               set END-LOOKUP To True

             When HASH-SLOT(HASH) = High-Value
      *      l'emplacement est libre: Exec Sql + memoization
               Perform EXEC-DB2-REQUEST
      *        seuls les sqlcode 0 +100 et -811  sont memoizés, pour les
      *        autres (-904,...), ils seront à nouveau demandés à DB2
               Evaluate SQLCODE
                 When Zero
                   Add 1 To  KEY-CPT SQLCOD-000
                   Move HASH-KEYS To HASH-TABLE-KEYS(HASH)
                   Move SQLCODE   To HASH-TABLE-SQLCODE(HASH)
```

```
                :DATATOTABLE:
      *         Move :INDICATORS: To ind-:TABLENAME:(hash)
              When -811
                Add 1 To  KEY-CPT SQLCOD-811
                Move HASH-KEYS To HASH-TABLE-KEYS(HASH)
                Move SQLCODE   To HASH-TABLE-SQLCODE(HASH)
                :DATATOTABLE:
      *         Move :INDICATORS: To ind-:TABLENAME:(hash)
              When +100
                Add 1 To  KEY-CPT SQLCOD-100
                Move HASH-KEYS To HASH-TABLE-KEYS(HASH)
                Move SQLCODE   To HASH-TABLE-SQLCODE(HASH)
              When Other
                Add 1 To SQLCOD-XXX
            End-Evaluate
            set END-LOOKUP To True

          When HASH-TABLE-KEYS(HASH) = HASH-KEYS
      *     clé trouvée en table : renvoyer données + sqlcode
            :DATAFROMTABLE:
            Move HASH-TABLE-SQLCODE(HASH) To SQLCODE
            set END-LOOKUP To True

          When HASH-FACTOR >= HASH-LOOP-MAX
      *     échec hashcode (nb d'essais épuisé):
            If Not MESSAGE-WARNING
              divide HASH-ECHEC-CPT by 100 giving N1 Remainder R1
              If R1 = 0
                Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
                If Taux > 85
                  Move Taux To Taux-x
                  Display "<WARNING> HASH-TABLE ("
                         RESSOURCE-NAME ") "  "is " Taux-x "% Full"
                  Set MESSAGE-WARNING To True
                End-if
              End-if
            End-if
            Add 1 to HASH-ECHEC-CPT
            Perform EXEC-DB2-REQUEST
            set END-LOOKUP To True

          When Other
      *     HASH-TABLE-KEYS(HASH) <> KEYS ==> collision de clés
      *     => calculer un nouveau hash en incrémentant hash-factor
            Add 1 to COLLISIONS-CPT(HASH-FACTOR + 1)
          End-evaluate
        End-perform
        .

       DESALLOCATION-TABLE.
      *====================*
        Call 'CEEFRST' Using ADDR-STORAGE FC
        If CEE000 Of FC
          Set Address Of HASH-TABLE To Null
          Set NOT-ALLOCATED To True
        Else
```

```
           Display "Erreur libération mémoire (Heap):"
           Display " heapid=" HEAPID " heapsize=" MSIZE
           Display " Ressource name :"  RESSOURCE-NAME
           Display " Severity=" SEVERITY " Msg-No=" MSG-NO
           Call "CEE3ABD" Using ABDCODE TIMING
         End-if
          .

       ALLOCATION-TABLE.
      *================*
         Perform SET-PRIME-NUMBER-SIZE
         Compute MSIZE = Length Of HASH-LINE * HASH-TABLE-SIZE
         Call 'CEEGTST' Using HEAPID
                              MSIZE
                              ADDR-STORAGE
                              FC
         If CEE000 Of FC
           Set Address Of HASH-TABLE To ADDR-STORAGE
           Move High-Value To HASH-TABLE
           Set ALLOCATED To True
         Else
           Display "Erreur allocation mémoire (Heap):"
           Display " heapid=" heapid " heapsize=" msize
           Display " Ressource name :"  RESSOURCE-NAME
           Display " Severity=" severity " Msg-No=" msg-no
           Call "CEE3ABD" Using ABDCODE TIMING
         End-if
          .

       SET-PRIME-NUMBER-SIZE.
      *======================*
      * table size * 1.5  (et doit etre impair)
         Compute HASH-TABLE-SIZE = (HASH-TABLE-SIZE * 3) / 2
         Divide HASH-TABLE-SIZE BY 2 Giving N1 Remainder R1
         If R1 = 0
            Add 1 to HASH-TABLE-SIZE
         End-if
         Set IS-NOT-PRIME To True
         Perform Until IS-PRIME
      *    Display "Check primality of:" HASH-TABLE-SIZE
           Set IS-PRIME To True
           Compute NUM =
              Function Integer(Function Sqrt(HASH-TABLE-SIZE)) + 1
           Perform Varying I From 2 BY 1
                   Until I >= NUM Or IS-NOT-PRIME
             Divide HASH-TABLE-SIZE BY I Giving N1 Remainder R1
             If R1 = 0
               Set IS-NOT-PRIME To True
             End-if
           End-Perform
           If IS-NOT-PRIME
             Add 2 To HASH-TABLE-SIZE
           Else
             Display "<" RESSOURCE-NAME "> "
                     "HashTable size set to "
                     HASH-TABLE-SIZE
```

```
          End-if
        End-Perform
         .

       STATISTIQUES-TABLE.
      *==================*
           Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
           Move    Taux To Taux-x
           Display " Ressource name       :" RESSOURCE-NAME
           Display " Taux de remplissage  :" TAUX-X "%"
           Display " Hash table size      :" HASH-TABLE-SIZE
                                          "( " MSIZE " octets)"
           Display " Re-hash factor limite:" HASH-LOOP-MAX
           Display " Key len              :" KEY-LEN
           Display " Module Call          :" MODULE-CALL
           Display " Exec SQL Call        :" EXEC-SQL-CPT
           Display " Clés rangées         :" KEY-CPT
           Display " Clés en échec        :" HASH-ECHEC-CPT
           Display " Sqlcode Zero         :" SQLCOD-000
           Display " Sqlcode +100         :" SQLCOD-100
           Display " Sqlcode -811         :" SQLCOD-811
           Display " Sqlcode autres       :" SQLCOD-XXX
           Display " Hash zero value      :" HASH-ZERO-CPT
           If COLLISIONS-CPT(1) = 0
             Display " Rehashing            : 0 (optimal)"
           Else
             Display " Niveaux de rehashing :"
             Perform Varying I from 0 by 1
                     Until I >= HASH-LOOP-MAX
                        Or COLLISIONS-CPT(I + 1) = 0
               Display "   collisions (rehash " I ") :"
                       COLLISIONS-CPT(I + 1)
             End-perform
           End-if
            .
       DUMP-HASHTABLE.
      *==============*
           Display " Dump Hash table      :" RESSOURCE-NAME
           Perform Varying I From 1 By 1
                   Until I > HASH-TABLE-SIZE
             If HASH-SLOT(I) = High-Value
                 Display "(SLOT " I ") "
                         " *** empty slot ***"
             Else
               Move HASH-TABLE-SQLCODE(I) To SQLCODEX
               Display "(SLOT " I ") "
                       "SQLCODE :" SQLCODEX
                       ";KEYS:" HASH-TABLE-KEYS(I)
                       ";DATA:" HASH-TABLE-DATA(I)
             End-if
           End-perform
            .
       EXEC-DB2-REQUEST.
      *================*
           Add 1 to EXEC-SQL-CPT
      * ici Exec Sql :
```

5.5 Module HASHCALL (mémoïsation CALL sous-programme)

Copy book HASCALW0 (working storage template)

```
      *================================================================*
      *   ,--------------------,                                       *
      *  /  EROL Technologies /                                        *
      * '--------------------'                                         *
      *                                                                *
      *================================================================*
       Data Division.
       Working-storage Section.
       01 HEAPID                  Pic S9(9) Binary Value Zero.
       01 MSIZE                   Pic S9(9) Binary.
       01 ADDR-STORAGE            Pointer          Value Null.
       01 FC.
         03 CONDITION-TOKEN-VALUE.
           88 CEE000          Value X'0000000000000000'.
           04 CASE-1-CONDITION-ID.
             05 SEVERITY          Pic S9(4) Binary.
             05 MSG-NO            Pic S9(4) Binary.
           04 CASE-2-CONDITION-ID Redefines CASE-1-CONDITION-ID.
             05 CLASS-CODE        Pic S9(4) Binary.
             05 CAUSE-CODE        Pic S9(4) Binary.
           04 CASE-SEV-CTL        Pic X.
           04 FACILITY-ID         Pic XXX.
         03 I-S-INFO              Pic S9(9) Binary.
       01  ABDCODE                Pic S9(9) Binary Value +16.
       01  TIMING                 Pic S9(9) Binary Value 1.

       01 TABLE-STATUS            Pic X Value Spaces.
         88  NOT-ALLOCATED  Value " ".
         88  ALLOCATED      Value "A".

       01 LOOKUP-STATUS           Pic X Value Spaces.
         88  START-LOOKUP   Value " ".
         88  END-LOOKUP     Value "E".

       01 PRIME-TEST              Pic X Value Spaces.
         88  IS-PRIME       Value " ".
         88  IS-NOT-PRIME   Value "N".

       01 MESSAGE-PLAY            Pic X Value Spaces.
         88 MESSAGE-WARNING Value "Y".

       01 NUM                     PIC S9(8) binary.
       01 N1                      PIC S9(8) binary.
       01 R1                      PIC S9(8) binary.
       01 I                       Pic S9(8) Binary.
       01 I3                      Pic S9(8) Binary.

       01 HASH                    Pic S9(8) Binary.
       01 KEY-LEN                 Pic S9(8) Binary.
       01 HASH-FACTOR             Pic S9(8) Binary.

       01 KEY-CPT                 PIC S9(8) binary Value 0.
```

```
       01 MODULE-CALL          Pic S9(8) Binary value 0.
       01 EXEC-SQL-CPT         Pic S9(8) Binary value 0.
       01 HASH-ECHEC-CPT       PIC S9(8) binary Value 0.
       01 HASH-ZERO-CPT        PIC S9(8) binary Value 0.
       01 SQLCODEX             PIC +999.
       01 TAUX                 PIC S9(3)V99 Comp-3.
       01 TAUX-X               PIC Z99.
       01 KEYNUMERIC           PIC 9(18).
       01 SQLCOD-000           PIC S9(8) binary Value 0.
       01 SQLCOD-100           PIC S9(8) binary Value 0.
       01 SQLCOD-811           PIC S9(8) binary Value 0.
       01 SQLCOD-XXX           PIC S9(8) binary Value 0.
       01 COLLISIONS-TAB.
         05 COLLISIONS-CPT Occurs 1000 Pic S9(8) binary Value 0.
```

Copy book HASCALP0 (procedure division template)

```
           03 HASH-SLOT Pic X(1).

       Procedure Division Using :DCLGEN:.
           Perform ENTRY-CALL-FILTER
           If NOT-ALLOCATED
             Move Length Of HASH-KEYS
               To KEY-LEN
             Perform ALLOCATION-TABLE
           End-if
           Add 1 To MODULE-CALL
           Perform HASH-TABLE-LOOKUP
           Goback
           .

       HASH-TABLE-LOOKUP.
      *=================*
      *    constitution d'une zone clé unique "concaténée":
           :LOADKEYS:
           Set START-LOOKUP To True
           Perform Varying HASH-FACTOR From 0 By 1 Until END-LOOKUP
             Call "HASHSUM2" Using  HASH-KEYS
                                    KEY-LEN
                                    HASH-TABLE-SIZE
                                    HASH
                                    HASH-FACTOR
             Evaluate True
             When HASH = 0
      *      garde-fou (la fonction hash ne devrait jamais rendre 0)
               Add 1 to HASH-ECHEC-CPT HASH-ZERO-CPT
               Perform EXEC-MODULE-CALL
               set END-LOOKUP To True

             When HASH-SLOT(HASH) = High-Value
      *      l'emplacement est libre: Call + memoization paramètres
               Perform EXEC-MODULE-CALL
               Move HASH-KEYS   To HASH-TABLE-KEYS(HASH)
               Move Space       To HASH-SLOT(HASH)
               Add 1 To  KEY-CPT
                   :DATATOTABLE:
```

```
            set END-LOOKUP To True

          When HASH-TABLE-KEYS(HASH) = HASH-KEYS
      *     clé trouvée en table : renvoyer paramètres
            :DATAFROMTABLE:
            set END-LOOKUP To True

          When HASH-FACTOR >= HASH-LOOP-MAX
            If HASH-ECHEC-CPT > 1000 and Not MESSAGE-WARNING
              If (HASH-ECHEC-CPT * 2 > EXEC-MODULE-CPT) Or
                 (KEY-CPT * 6 >  HASH-TABLE-SIZE * 7)
                Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
                Move Taux To Taux-x
                Display "<WARNING> HASH-TABLE ("
                        RESSOURCE-NAME ") "  "is " Taux-x "% Full"
                Compute Taux =
                        100 * (HASH-ECHEC-CPT / EXEC-MODULE-CPT)
                Move Taux To Taux-x
                Display "  hash echec: " Taux-x "% "
                       "(" HASH-ECHEC-CPT "/" EXEC-MODULE-CPT ")"
                Compute MO-SIZE = MSIZE / 1000000
                Move MO-SIZE To MO-SIZE-X
                Display "  HASH-TABLE-SIZE="
                        HASH-TABLE-SIZE " saturée ("
                        MO-SIZE-X " Mo), performances dégradées"
      *         Perform STATISTIQUES-TABLE
                Set MESSAGE-WARNING To True
              End-if
            End-if
            Add 1 to HASH-ECHEC-CPT
            Perform EXEC-MODULE-CALL
            set END-LOOKUP To True

          When Other
      *     HASH-TABLE-KEYS(HASH) <> KEYS ==> on a collision de clés
            Add 1 to COLLISIONS-CPT(HASH-FACTOR + 1)
      *     => boucler tant que hash-factor < HASH-LOOP-MAX, cad:
      *        calculer un nouveau hash avec hash-factor + 1
          End-evaluate
        End-perform
        .

       DESALLOCATION-TABLE.
      *====================*
        Call 'CEEFRST' Using ADDR-STORAGE FC
        If CEE000 Of FC
          Set Address Of HASH-TABLE To Null
          Set NOT-ALLOCATED To True
        Else
          Display "Erreur libération mémoire (Heap):"
          Display " heapid=" HEAPID " heapsize=" MSIZE
          Display " Ressource name :"  RESSOURCE-NAME
          Display " Severity=" SEVERITY " Msg-No=" MSG-NO
          Call "CEE3ABD" Using ABDCODE TIMING
        End-if
         .
```

```
 ALLOCATION-TABLE.
*================*
     Perform SET-PRIME-NUMBER-SIZE
     Compute MSIZE = Length Of HASH-LINE * HASH-TABLE-SIZE
     Call 'CEEGTST' Using HEAPID
                          MSIZE
                          ADDR-STORAGE
                          FC
     If CEE000 Of FC
       Set Address Of HASH-TABLE To ADDR-STORAGE
       Move High-Value To HASH-TABLE
       Set ALLOCATED To True
     Else
       Display "Erreur allocation mémoire (Heap):"
       Display " heapid=" heapid " heapsize=" msize
       Display " Ressource name :"  RESSOURCE-NAME
       Display " Severity=" severity " Msg-No=" msg-no
       Call "CEE3ABD" Using ABDCODE TIMING
     End-if
      .

 SET-PRIME-NUMBER-SIZE.
*======================*
* table size * 1.5  (et doit etre impair)
     Compute HASH-TABLE-SIZE = (HASH-TABLE-SIZE * 3) / 2
     Divide HASH-TABLE-SIZE BY 2 Giving N1 Remainder R1
     If R1 = 0
        Add 1 to HASH-TABLE-SIZE
     End-if
     Set IS-NOT-PRIME To True
     Perform Until IS-PRIME
*      Display "Check primality of:" HASH-TABLE-SIZE
       Set IS-PRIME To True
       Compute NUM =
          Function Integer(Function Sqrt(HASH-TABLE-SIZE)) + 1
       Perform Varying I From 2 BY 1
               Until I >= NUM Or IS-NOT-PRIME
         Divide HASH-TABLE-SIZE BY I Giving N1 Remainder R1
         If R1 = 0
           Set IS-NOT-PRIME To True
         End-if
       End-Perform
       If IS-NOT-PRIME
         Add 2 To HASH-TABLE-SIZE
       Else
         Display "<" RESSOURCE-NAME "> "
                 "HashTable size set to "
                 HASH-TABLE-SIZE
       End-if
     End-Perform
      .

 STATISTIQUES-TABLE.
*===================*
     Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
```

```
           Move    Taux To Taux-x
           Display " Ressource name       : " RESSOURCE-NAME
           Display " Taux de remplissage  : " TAUX-X "%"
           Compute Taux =
              100 * (1 - (HASH-ECHEC-CPT / (MODULE-CALL - KEY-CPT)))
           Move    Taux To Taux-x
           Display " Taux de réussite hash: " TAUX-X "%"
           Compute MO-SIZE = MSIZE / 1000000
           Move MO-SIZE To MO-SIZE-X
           Display " Hash table size      : " HASH-TABLE-SIZE
                                     " (" MO-SIZE-X " Mo)"
           Display " Re-hash factor limite: " HASH-LOOP-MAX
           Display " Key size             : " KEY-LEN
           Display " Module Call          : " MODULE-CALL
           Display " Exec Module Call     : " EXEC-MODULE-CPT
           Display " Clés hashées         : " KEY-CPT
           Display " Clés en échec        : " HASH-ECHEC-CPT
           Display " Hash zero value      : " HASH-ZERO-CPT
           If COLLISIONS-CPT(1) = 0
             Display " Rehashing            : 0 (performance optimale)"
           Else
             Display " Niveaux de rehashing :"
             Perform Varying I from 0 by 1
                     Until I >= HASH-LOOP-MAX
                        Or COLLISIONS-CPT(I + 1) = 0
               Display "   collisions (rehash " I ") :"
                       COLLISIONS-CPT(I + 1)
             End-perform
           End-if
            .
       EXEC-MODULE-CALL.
      *================*
           Add 1 to EXEC-MODULE-CPT
      * Call module:
```

5.6 Module HASMEM (chargement + recherche en table hashcodée)

Copy book HASMEMP0 (procedure division template)

```
           03 HASH-TABLE-RC       Pic S9(4) comp.
           03 HASH-SLOT redefines HASH-TABLE-RC Pic X(2).

      *---------------------------------------------------------------+
      *  Optimisations cpu (EROL Technologies)
      *  Nested modules de gestion de table hashcodée en mémoire
      *---------------------------------------------------------------+
      *  - HASH-TABLE-SIZE détermine la taille (lignes) de la table
      *    hashcodée (elle est allouée dynamiquement dès le 1er accès)
      *    Elle est ajustée automatiquement à la valeur d'un nombre
      *    premier proche de 1,5 X la valeur indiquée (performance du
      *    hashcoding)
      *---------------------------------------------------------------+
      *  - HASH-LOOP-MAX détermine le nombre de tentatives d'insertions
      *    ou de lecture en table en cas de collision de clé:
```

```
      *    999 est une valeur raisonnable qui permet bien souvent de
      *    remplir la table à plus de 90%
      *-----------------------------------------------------------------+
      *  - RESSOURCE-NAME sert uniquement lors de l'émission de
      *    messages d'anomalie (personnalisable)
      *-----------------------------------------------------------------+
      *  - Les valeurs du code retour (RC) sont inspirées de DB2:
      *      Zero : insertion/lecture OK
      *      +100 : non trouvé
      *      -811 : insertion ou lecture d'un clé en double
      *      -904 : la table est saturée, plus aucune insertion possible
      *-----------------------------------------------------------------+
       Procedure Division Using HASH-FUNC
                                :DCLGEN:
                                RC.
           Evaluate HASH-FUNC
           When "READ"
             If NOT-ALLOCATED
               Display "<ERROR> HASH-TABLE Not allocated"
               Display " Ressource name :"  RESSOURCE-NAME
               Display " You must 'STOR' some values first"
               Call "CEE3ABD" Using abdcode timing
             End-if
             Add 1 To MODULE-CALL
             Perform HASH-TABLE-LOOKUP

           When "STOR"
             If NOT-ALLOCATED
               Move Length Of HASH-KEYS
                 To KEY-LEN
               Perform ALLOCATION-TABLE
             End-if
             Add 1 To MODULE-CALL
             Perform HASH-TABLE-STORE

           When "DLET"
             If NOT-ALLOCATED
               Display "<ERROR> HASH-TABLE Not allocated"
               Display " Ressource name :"  RESSOURCE-NAME
               Display " You must first 'STOR' some values"
               Call "CEE3ABD" Using abdcode timing
             End-if
             Add 1 To MODULE-CALL
             Perform HASH-TABLE-DELETE

           When "STAT"
             If ALLOCATED
               Perform STATISTIQUES-TABLE
             Else
               Display "<ERROR> HASH-TABLE Not allocated"
               Display " Ressource name :"  RESSOURCE-NAME
               Call "CEE3ABD" Using abdcode timing
             end-if

           When "CLOS"
             If ALLOCATED
```

```
            Perform DESALLOCATION-TABLE
          end-if

        When "DUMP"
          If ALLOCATED
            Perform DUMP-HASHTABLE
          Else
            Display "<ERROR> HASH-TABLE Not allocated"
            Display " Ressource name :"  RESSOURCE-NAME
            Call "CEE3ABD" Using abdcode timing
          end-if

        When OTHER
          Display "<ERROR> Invalide hash function:" HASH-FUNC
          Display " Ressource name :"  RESSOURCE-NAME
          Display " Valides functions are: "
                  " READ, STOR, DLET, STAT, CLOS, DUMP"
          Call "CEE3ABD" Using abdcode timing
        end-evaluate
        Goback
        .

       HASH-TABLE-LOOKUP.
      *==================*
        :LOADKEYS:
        Set START-LOOKUP To True
        Perform Varying HASH-FACTOR From 0 By 1 Until END-LOOKUP
          Call "HASHSUM2" Using  HASH-KEYS
                                 KEY-LEN
                                 HASH-TABLE-SIZE
                                 HASH
                                 HASH-FACTOR
          Evaluate True
          When HASH = 0
      *     garde-fou (la fonction hash ne devrait jamais rendre 0)
            Display "<ERROR> HASH-TABLE ("
                    RESSOURCE-NAME ") Hash Function Error"
            Display "keys: " KEY-CPT " Hash:" HASH
            Move -904      To RC
            Call "CEE3ABD" Using abdcode timing
            set END-LOOKUP To True

          When HASH-SLOT(HASH) = High-Value
      *     l'emplacement est libre: clé inconnue RC=+100
            Add 1 To  SQLCOD-100
            Move +100 To RC
            set END-LOOKUP To True

          When HASH-TABLE-KEYS(HASH) = HASH-KEYS
           and HASH-SLOT(HASH) Not = 'DD'
      *      l'emplacement n'est pas zombie (deleted) et la
      *      clé trouvée en table : renvoyer données + sqlcode
            :DATAFROMTABLE:
            Move HASH-TABLE-RC(HASH) To RC
            set END-LOOKUP To True
```

```
            When HASH-FACTOR >= HASH-LOOP-MAX
      *       échec hashcode (nb d'essais épuisé): cle inconnue RC=+100
              Add 1 To  SQLCOD-100
              Move +100 To RC
      *       display "RC=" RC
              set END-LOOKUP To True

            When Other
      *     HASH-TABLE-KEYS(HASH) <> KEYS ==> collision de clés
      *     => calculer un nouveau hash en incrémentant hash-factor
              Add 1 to COLLISIONS-CPT(HASH-FACTOR + 1)
            End-evaluate
          End-perform
          .

       HASH-TABLE-STORE.
      *================*
          :LOADKEYS:
          Set START-LOOKUP To True
          Perform Varying HASH-FACTOR From 0 By 1 Until END-LOOKUP
            Call "HASHSUM2" Using  HASH-KEYS
                                   KEY-LEN
                                   HASH-TABLE-SIZE
                                   HASH
                                   HASH-FACTOR
            Evaluate True
            When HASH = 0
      *     garde-fou (la fonction hash ne devrait jamais rendre 0)
              Display "<ERROR> HASH-TABLE ("
                      RESSOURCE-NAME ") Hash Function Error"
              Display "keys: " KEY-CPT " Hash:" HASH
              Move -904      To RC
              Call "CEE3ABD" Using abdcode timing
              set END-LOOKUP To True

            When HASH-SLOT(HASH) = High-Value
      *     l'emplacement est libre:  stockage key + data + RC=0
              Add 1 To  KEY-CPT SQLCOD-000
              Move HASH-KEYS To HASH-TABLE-KEYS(HASH)
              Move Zero      To HASH-TABLE-RC(HASH)
                                RC
              :DATATOTABLE:
              set END-LOOKUP To True

            When HASH-TABLE-KEYS(HASH) = HASH-KEYS
             and HASH-SLOT(HASH) Not = 'DD'
      *       display "duplicate store:" HASH-KEYS
              Move -811      To HASH-TABLE-RC(HASH)
                                RC
      *       clé déjà en table : update RC=-811
              set END-LOOKUP To True

            When HASH-FACTOR >= HASH-LOOP-MAX
      *       échec hashcode (nb d'essais épuisé):
              Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
              Move Taux To Taux-x
```

```
           Display "<ERROR> HASH-TABLE ("
                  RESSOURCE-NAME ") "  "is " Taux-x "% Full"
           Display " -> Nb keys: " KEY-CPT " Size:" HASH-TABLE-SIZE
           Display " -> Echec STOR key=<" HASH-KEYS ">"
           Move -904      To RC
           Call "CEE3ABD" Using abdcode timing
      *    Perform EXEC-DB2-REQUEST
           set END-LOOKUP To True

         When Other
      *  HASH-TABLE-KEYS(HASH) <> KEYS ==> collision de clés
      *  => calculer un nouveau hash en incrémentant hash-factor
           Add 1 to COLLISIONS-CPT(HASH-FACTOR + 1)
         End-evaluate
       End-perform
       .

      HASH-TABLE-DELETE.
      *=================*
           :LOADKEYS:
           Set START-LOOKUP To True
           Perform Varying HASH-FACTOR From 0 By 1 Until END-LOOKUP
             Call "HASHSUM2" Using  HASH-KEYS
                                    KEY-LEN
                                    HASH-TABLE-SIZE
                                    HASH
                                    HASH-FACTOR
             Evaluate True
             When HASH = 0
      *      garde-fou (la fonction hash ne devrait jamais rendre 0)
               Display "<ERROR> HASH-TABLE ("
                      RESSOURCE-NAME ") Hash Function Error"
               Display "keys: " KEY-CPT " Hash:" HASH
               Move -904      To RC
               Call "CEE3ABD" Using abdcode timing
               set END-LOOKUP To True

             When HASH-SLOT(HASH) = High-Value
      *      l'emplacement est libre: clé inconnue RC=+100
               Add 1 To  SQLCOD-100
               Move +100 To RC
               set END-LOOKUP To True

             When HASH-TABLE-KEYS(HASH) = HASH-KEYS
      *        clé trouvée en table : détruire données
               Move High-Value To HASH-LINE(HASH)
      *        Slot rendu inutilisable pour 'STOR', indispensable pour
      *        que le chainage "double hash" ne soit pas corrompu :
               Move 'DD' To HASH-SLOT(HASH)
               Move Zero To RC
               set END-LOOKUP To True

             When HASH-FACTOR >= HASH-LOOP-MAX
      *        échec hashcode (nb d'essais épuisé): cle inconnue RC=+100
               Add 1 To  SQLCOD-100
               Move +100 To RC
```

```
            display "RC=" RC
            set END-LOOKUP To True

          When Other
      *     HASH-TABLE-KEYS(HASH) <> KEYS ==> collision de clés
      *     => calculer un nouveau hash en incrémentant hash-factor
            Add 1 to COLLISIONS-CPT(HASH-FACTOR + 1)
          End-evaluate
        End-perform
        .

       DESALLOCATION-TABLE.
      *====================*
        Call 'CEEFRST' Using ADDR-STORAGE FC
        If CEE000 Of FC
          Set Address Of HASH-TABLE To Null
          Set NOT-ALLOCATED To True
        Else
          Display "Erreur libération mémoire (Heap):"
          Display " heapid=" HEAPID " heapsize=" MSIZE
          Display " Ressource name :"  RESSOURCE-NAME
          Display " Severity=" SEVERITY " Msg-No=" MSG-NO
          Call "CEE3ABD" Using ABDCODE TIMING
        End-if
         .

       ALLOCATION-TABLE.
      *=================*
        Perform SET-PRIME-NUMBER-SIZE
        Compute MSIZE = Length Of HASH-LINE * HASH-TABLE-SIZE
        Call 'CEEGTST' Using HEAPID
                             MSIZE
                             ADDR-STORAGE
                             FC
        If CEE000 Of FC
          Set Address Of HASH-TABLE To ADDR-STORAGE
          Move High-Value To HASH-TABLE
          Set ALLOCATED To True
        Else
          Display "Erreur allocation mémoire (Heap):"
          Display " heapid=" heapid " heapsize=" msize
          Display " Ressource name :"  RESSOURCE-NAME
          Display " Severity=" severity " Msg-No=" msg-no
          Call "CEE3ABD" Using ABDCODE TIMING
        End-if
         .
       SET-PRIME-NUMBER-SIZE.
      *======================*
      * table size * 1.5 impair
        Multiply HASH-TABLE-SIZE By 1.5 Giving HASH-TABLE-SIZE
        Divide HASH-TABLE-SIZE BY 2 Giving N1 Remainder R1
        If R1 = 0
           Add 1 to HASH-TABLE-SIZE
        End-if
        Set IS-NOT-PRIME To True
        Perform Until IS-PRIME
```

```
*       Display "Check primality of:" HASH-TABLE-SIZE
        Set IS-PRIME To True
        Compute NUM =
           Function Integer(Function Sqrt(HASH-TABLE-SIZE)) + 1
        Perform Varying I From 2 BY 1
                Until I >= NUM Or IS-NOT-PRIME
          Divide HASH-TABLE-SIZE BY I Giving N1 Remainder R1
          If R1 = 0
            Set IS-NOT-PRIME To True
          End-if
        End-Perform
        If IS-NOT-PRIME
          Add 2 To HASH-TABLE-SIZE
        Else
          Display "<" RESSOURCE-NAME "> "
                  "HashTable size ajusted to  "
                  HASH-TABLE-SIZE " lines"
        End-if
      End-Perform
       .
 IS-PRIME-NUMBER.
*================*
      Compute NUM =
         Function Integer(Function Sqrt(HASH-TABLE-SIZE)) + 1
      Perform Varying I From 2 BY 1
              Until I >= NUM Or IS-NOT-PRIME
        Divide HASH-TABLE-SIZE BY I Giving N1 Remainder R1
        If R1 = 0
          Set IS-NOT-PRIME To True
        End-if
      End-Perform
       .

 STATISTIQUES-TABLE.
*====================*
      Compute Taux = 100 * (KEY-CPT / HASH-TABLE-SIZE)
      Move    Taux To Taux-x
      Display " Ressource name       :" RESSOURCE-NAME
      Display " Hash table size      :" HASH-TABLE-SIZE " (lines)"
      Display " Hash table size      :" MSIZE " (bytes)"
      Display " Taux de remplissage  :" TAUX-X "%"
      Display " Hash loop max        :" HASH-LOOP-MAX
      Display " Key len              :" KEY-LEN
      Display " Total module call    :" MODULE-CALL
      Display " Exec SQL Call        :" EXEC-SQL-CPT
      Display " Clés stockées        :" KEY-CPT
      Display " Clés rejetées        :" HASH-ECHEC-CPT
      Display " Singletons (rc=0)    :" SQLCOD-000
      Display " Inexistants (rc=+100):" SQLCOD-100
      Display " Doublons (rc=-811)   :" SQLCOD-811
      Display " Autres codes         :" SQLCOD-XXX
      Display " Hash zero value      :" HASH-ZERO-CPT
      If COLLISIONS-CPT(1) = 0
        Display " Hash collisions      :0"
      Else
        Perform Varying I from 0 by 1
```

```
                Until I >= HASH-LOOP-MAX
                   Or COLLISIONS-CPT(I + 1) = 0
           Display " collisions (hash-loop " I ") :"
                   COLLISIONS-CPT(I + 1)
         End-perform
       End-if
        .
 DUMP-HASHTABLE.
*==============*
       Display " Dump Hash table      :" RESSOURCE-NAME
       Perform Varying I From 1 By 1
               Until I > HASH-TABLE-SIZE
         If HASH-SLOT(I) = High-Value
             Display "(SLOT " I ") "
                     " *** empty slot ***"
         Else
           Move HASH-TABLE-RC(I) To RCX
           Display "(SLOT " I ") "
                   "RC :" RCX
                   ";KEYS:" HASH-TABLE-KEYS(I)
                   ";DATA:" HASH-TABLE-DATA(I)
         End-if
       End-perform
       .
*USER-EXIT-500.
*================*
*    Display "hash factor=" HASH-FACTOR
```

Copy book HASMEMW0 (working storage template)

```
       Data Division.
       Working-storage Section.
       01 HEAPID                Pic S9(9) Binary Value Zero.
       01 MSIZE                 Pic S9(9) Binary.
       01 ADDR-STORAGE          Pointer          Value Null.
       01 FC.
         03 CONDITION-TOKEN-VALUE.
           88 CEE000          Value X'0000000000000000'.
           04 CASE-1-CONDITION-ID.
             05 SEVERITY        Pic S9(4) Binary.
             05 MSG-NO          Pic S9(4) Binary.
           04 CASE-2-CONDITION-ID Redefines CASE-1-CONDITION-ID.
             05 CLASS-CODE      Pic S9(4) Binary.
             05 CAUSE-CODE      Pic S9(4) Binary.
           04 CASE-SEV-CTL      Pic X.
           04 FACILITY-ID       Pic XXX.
         03 I-S-INFO            Pic S9(9) Binary.
       01  ABDCODE              Pic S9(9) Binary Value +16.
       01  TIMING               Pic S9(9) Binary Value 1.

       01 TABLE-STATUS          Pic X Value Spaces.
         88  NOT-ALLOCATED  Value " ".
         88  ALLOCATED      Value "A".

       01 LOOKUP-STATUS       Pic X Value Spaces.
         88  START-LOOKUP   Value " ".
         88  END-LOOKUP     Value "E".
```

```
       01 PRIME-TEST          Pic X Value Spaces.
         88  IS-PRIME       Value " ".
         88  IS-NOT-PRIME   Value "N".

       01 MESSAGE-PLAY        Pic X Value Spaces.
         88 MESSAGE-WARNING Value "Y".

       01 HASH-LOOP-MAX       Pic S9(8) binary Value +999.
       01 COLLISIONS-TAB.
         05 COLLISIONS-CPT Occurs 1000
                           PIC S9(8) binary Value 0.

       01 NUM                 PIC S9(8) binary.
       01 N1                  PIC S9(8) binary.
       01 R1                  PIC S9(8) binary.
       01 I                   Pic S9(8) Binary.
       01 I3                  Pic S9(8) Binary.

       01 HASH                Pic S9(8) Binary.
       01 KEY-LEN             Pic S9(8) Binary.
       01 HASH-FACTOR         Pic S9(8) Binary.

       01 KEY-CPT             PIC S9(8) binary Value 0.
       01 KEY-STORE           PIC S9(8) binary Value 0.
       01 MODULE-CALL         Pic S9(8) Binary value 0.
       01 EXEC-SQL-CPT        Pic S9(8) Binary value 0.
       01 HASH-ECHEC-CPT      PIC S9(8) binary Value 0.
       01 HASH-ZERO-CPT       PIC S9(8) binary Value 0.
       01 RCX                 PIC +999.
       01 TAUX                PIC S9(3)V99 Comp-3.
       01 TAUX-X              PIC Z99.
       01 KEYNUMERIC          PIC 9(18).
       01 SQLCOD-000          PIC S9(8) binary Value 0.
       01 SQLCOD-100          PIC S9(8) binary Value 0.
       01 SQLCOD-811          PIC S9(8) binary Value 0.
       01 SQLCOD-XXX          PIC S9(8) binary Value 0.
```

5.7 Module SEARCVS (Chargement VSAM + recherche dichotomique)

Copy book SEARCVSF (file section template)

```
       Environment Division.
       Input-output Section.
       File-control.
           Select :VSAMFILE:  ASSIGN :VSAMFILE:
           Organization Indexed
           Access Mode Sequential
           Record Key :VSAMKEY: Of :VSAMRECORD:
           File Status VSAM-STATUS.
       Data Division.
       File Section.
       FD  :VSAMFILE:
           Data  Record :VSAMRECORD:.
```

Copy book SEARCVSW (working storage template)

```
       Working-storage Section.
       01 VSAM-STATUS           Pic 99.
       01 FC.
         03 CONDITION-TOKEN-VALUE.
           88 CEE000         Value X'0000000000000000'.
           04 CASE-1-CONDITION-ID.
             05 SEVERITY        Pic S9(4) Binary.
             05 MSG-NO          Pic S9(4) Binary.
           04 CASE-2-CONDITION-ID Redefines CASE-1-CONDITION-ID.
             05 CLASS-CODE      Pic S9(4) Binary.
             05 CAUSE-CODE      Pic S9(4) Binary.
           04 CASE-SEV-CTL      Pic X.
           04 FACILITY-ID       Pic XXX.
         03 I-S-INFO            Pic S9(9) Binary.
       01  ABDCODE              Pic S9(9) Binary Value +16.
       01  TIMING               Pic S9(9) Binary Value 1.
       01 TABLE-STATUS          Pic X Value Spaces.
         88  NOT-ALLOCATED    Value " ".
         88  ALLOCATED        Value "A".
       01 SEARCH-STATUS         Pic X Value "N".
         88  FOUND            Value "Y".
         88  NOT-FOUND        Value "N".

       01 DEPASSEMENT           Pic X Value "N".
         88  DEPASSEMENT-SIZE Value "Y".

       01 RESSOURCE-NAME        Pic X(32)  Value :RESSOURCE:.
       01 VSAM-LINES            Pic S9(8)  Binary Value 0.
       01 MAX-LINES             Pic S9(8)  Binary Value :MAX-SIZE:.
       01 NUMBER-LINES          Pic S9(8)  Binary Value 0.
       01 SEARCH-SUCCESS        Pic S9(10) Binary Value 0.
       01 MODULE-CALL           Pic S9(10) Binary Value 0.
       01 HEAPID                Pic S9(9)  Binary Value 0.
       01 MSIZE                 Pic S9(9)  Binary Value 0.
       01 ADDR-STORAGE          Pointer           Value Null.
       01 CTRL-KEY              :PICTURE-KEY:.
       01 SAVE-KEY              :PICTURE-KEY:.
```

Copy book SEARCVSP (procedure division template)

```
      *=================================================================*
      *   ,--------------------,                                        *
      *  /  EROL Technologies /  TEMPLATE SEARCVSP:                     *
      * '--------------------'                                          *
      *     Nested module de chargement en mémoire d'un fichier         *
      *     VSAM KSDS avec recherche dichotomique en table cobol        *
      *=================================================================*

       Procedure Division Using SEARCH-FUNC
                                PARM-RECORD
                                RC
                                .

           Evaluate SEARCH-FUNC

             When "STAT    "
```

```
              Perform STATISTIQUES-TABLE
            When "CLOSE   "
              Perform DESALLOCATION-TABLE
            When Other
              Perform SEARCH-TABLE
          End-evaluate
          Goback
           .

       SEARCH-TABLE.
      *=============*
          Add 1 To MODULE-CALL
          If NOT-ALLOCATED
            Move :VSAMKEY: OF PARM-RECORD
              TO SAVE-KEY
            Perform PRE-CHARGEMENT-TABLE
            Move SAVE-KEY
              TO :VSAMKEY: OF PARM-RECORD
          End-If

          Search ALL SEARCH-LINE
           At End
             Set NOT-FOUND To True
           When :VSAMKEY: Of SEARCH-LINE(INDX) =
                :VSAMKEY: OF PARM-RECORD
             Set FOUND To True
          End-Search

          If FOUND
              :MOVESOUT:
              Move Zero To RC
              Add 1 To SEARCH-SUCCESS
          Else
              Move +100 To RC
          End-if
           .

       DESALLOCATION-TABLE.
      *====================*
          Call 'CEEFRST' Using ADDR-STORAGE FC
          If CEE000 Of FC
            Set Address Of SEARCH-TAB To Null
            Set NOT-ALLOCATED To True
            Move Zero To RC
          Else
            Display "Erreur libération mémoire (Heap):"
            Display " heapid=" HEAPID " heapsize=" MSIZE
            Display " Severity=" SEVERITY " Msg-No=" MSG-NO
            Display " Ressource name (vsam):" RESSOURCE-NAME
            Call "CEE3ABD" Using ABDCODE TIMING
          End-if
           .

       ALLOCATION-TABLE.
      *=================*
          Compute MSIZE = Length Of SEARCH-LINE * MAX-LINES
```

```
           Call 'CEEGTST' Using HEAPID
                                MSIZE
                                ADDR-STORAGE
                                FC
           If CEE000 Of FC
             Set Address Of SEARCH-TAB To ADDR-STORAGE
             Move High-value To SEARCH-TAB
             Set ALLOCATED To True
           Else
             Display "Erreur allocation mémoire (Heap):"
             Display " heapid=" heapid " heapsize=" msize
             Display " Severity=" severity " Msg-No=" msg-no
             Display " Ressource name (vsam):" RESSOURCE-NAME
             Call "CEE3ABD" Using ABDCODE TIMING
           End-if
            .

         STATISTIQUES-TABLE.
        *==================*
        *    Display " Module name          :" MODULE-NAME
           Display " Ressource name (vsam):  " RESSOURCE-NAME
           Display " Binary table size    :  " MSIZE " (bytes)"
           Display " Binary table lines   :  " MAX-LINES
           Display " Total Search calls   :" MODULE-CALL
           Display " Successful Search    :" SEARCH-SUCCESS
           Move Zero To RC
            .

         PRE-CHARGEMENT-TABLE.
        *====================*
        * controles de conformité longueur/picture clé:
            If Length Of :VSAMKEY: OF :VSAMRECORD:
               Not = Length Of CTRL-KEY
             Display "Erreur longueur de clé:"
             Display " Ressource name (vsam):" RESSOURCE-NAME
             Display " longueur clé du fichier vsam:"
                     Length Of :VSAMKEY: OF :VSAMRECORD:
             Display " => il faut modifier la valeur de 'PICTURE-KEY'"
                     " (longueur actuelle: " Length Of CTRL-KEY ")"
             Call "CEE3ABD" Using ABDCODE TIMING
            End-if
            Perform ALLOCATION-TABLE
            Open input :VSAMFILE:
            If VSAM-STATUS Not = 0
        *     Display "Module de recherche :" MODULE-NAME
              Display "Erreur Open vsam RC=" VSAM-STATUS
              Display " Ressource name (vsam):" RESSOURCE-NAME
              Call "CEE3ABD" Using ABDCODE TIMING
            End-if
            Move :STARTKEY: To :VSAMKEY: OF :VSAMRECORD:
            Start :VSAMFILE: Key >=
                  :VSAMKEY: OF :VSAMRECORD:
            If VSAM-STATUS Not = 0
        *     Display "Module de recherche :" MODULE-NAME
              Display "Erreur Start vsam RC=" VSAM-STATUS
              Display " Ressource name (vsam):" RESSOURCE-NAME
```

```
            Call "CEE3ABD" Using ABDCODE TIMING
          End-if
          Read :VSAMFILE: NEXT
          If VSAM-STATUS Not = 0
      *     Display "Module de recherche :" MODULE-NAME
            Display "Erreur Read Next vsam RC=" VSAM-STATUS
            Display " Ressource name (vsam):" RESSOURCE-NAME
            Call "CEE3ABD" Using ABDCODE TIMING
          End-if
          Set INDX To 1
          Perform Until VSAM-STATUS Not = 0
            Set VSAM-LINES To INDX
      * controles du bon séquencement de la clé de recherche
            if :VSAMKEY: OF :VSAMRECORD:
                 < CTRL-KEY
              Display "Erreur cobol table ASCENDING Key sequence: "
                      "Next Vsam Key is not >= " CTRL-KEY
              Display " Ressource name (vsam):" RESSOURCE-NAME
              Call "CEE3ABD" Using ABDCODE TIMING
            End-if
            Move :VSAMKEY: OF :VSAMRECORD: To CTRL-KEY
      * controles de dépassement capacité table
            If not depassement-size And
               VSAM-LINES > MAX-LINES
               set depassement-size To True
            End-if
            If not depassement-size
              :MOVESIN:
            End-if
            Read :VSAMFILE: NEXT
            Set INDX Up By 1
          End-perform
      *   display "prechargement vsam OK " RESSOURCE-NAME
          If VSAM-STATUS Not = 0 and +10
           Display "Erreur Read Next VSAM-STATUS=" VSAM-STATUS
           Display " Ressource name (vsam):" RESSOURCE-NAME
           Call "CEE3ABD" Using ABDCODE TIMING
          End-if
          If depassement-size
      *    Display "Module de recherche :" MODULE-NAME
           Display " Ressource name (vsam):" RESSOURCE-NAME
           Display "Erreur SEARCH-TAB size: "
                   "Le fichier contient " VSAM-LINES " enregs"
           Display " => il faut augmenter la valeur de 'MAX-LINES'"
                   " (actuellement: " MAX-LINES ")"
           Call "CEE3ABD" Using ABDCODE TIMING
          End-if
          Close :VSAMFILE:
          .
```