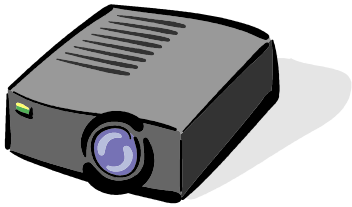
Modernisation et développement d'applications IBM i

Stratégies, technologies et outils

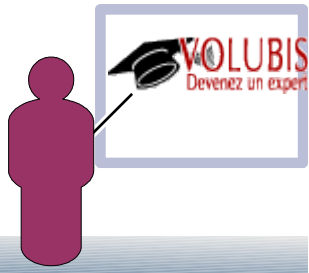
5 et 6 Avril 2012 – IBM Forum de Bois-Colombes



Volubis.fr

Conseil et formation sur OS/400, I5/OS puis IBM ***i*** depuis 1994 !

Christian Massé - cmasse@volubis.fr

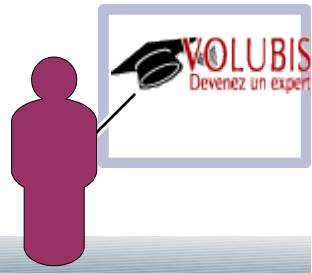


SQL , requêtes imbriquées

- La notion de requêtes imbriquées est ancienne
  - `Select * from stagiaires where note < (select avg(note) from stagaires)`

    donne la liste des stagiaires ayant une note < à la moyenne
  - `Select * from stagiaires S1 where note < (select avg(note) from stagiaires S2 where S2.entree = S1.entree)`

    *donne la liste des stagiaires ayant une note < à la moyenne de **leur session***

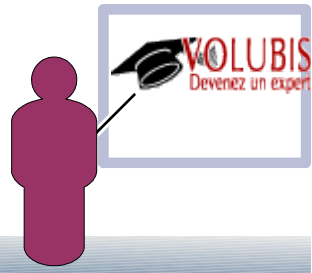


SQL , requêtes imbriquées

- On peut aussi tester l'existence d'une information dans une autre table

  - ```
    Select * from stagiaires where agence not
      in (select agence from agences)
    ```

  - ```
    Select * from stagiaires S1 where not
      exists (select * from agences where
      pays = S1.pays and agence = S1.agence)
    ```

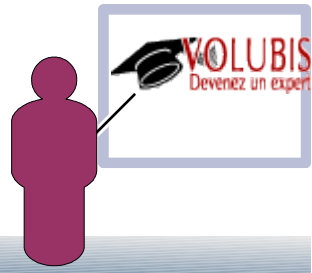


SQL , requêtes imbriquées

- On peut utiliser une requête imbriquée dans un ordre UPDATE ou DELETE

  - ```
    Update commandes C1 set pricde = (select
      pritarif from articles where codart =
      C1.codart)
    where pricde = 0
    ```

  - ```
    Delete from articles A1 where not exists
      (select * from commandes where codart =
      A1.codart)
    ```

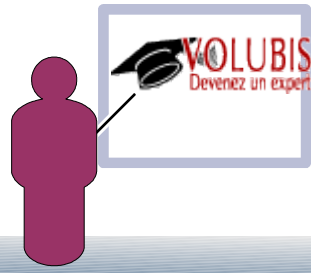


SQL , requêtes imbriquées

- Bref, aujourd'hui on peut mettre une sous-requête partout (liste des colonnes ou clause FROM inclus) :
  - `Select (prix * qte) as montant , (select sum(prix * qte) from commandes where codart = C1.codart) as total_par_article from commandes C1 order by 2 desc`
  - `Select avg(nbr) from (select week_iso(datcde) , count(*) as nbr from commandes group by week_iso(datcde) as par_semaine`

    Donne la moyenne (générale)

    du nombre de commandes par semaine

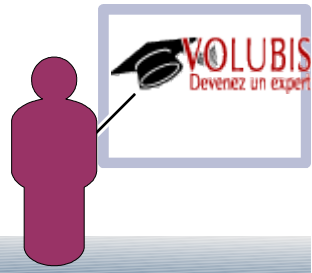


Common Table Expression

- La dernière requête aurai pu s'écrire
  - ```
    With par_semaine as (select
    week_iso(datcde) , count(*) as nbr from
    commandes group by week_iso(datcde)
    select avg(nbr) from par_semaine
    ```

    permettant
  - ```
    With par_semaine as (select
    week_iso(datcde) , count(*) as nbr from
    commandes group by week_iso(datcde)
    select * from par_semaine where nbr >
    (select avg(nbr) from par_semaine)

    -- liste des semaines où il y a
    plus de commandes que d'habitude
    ```



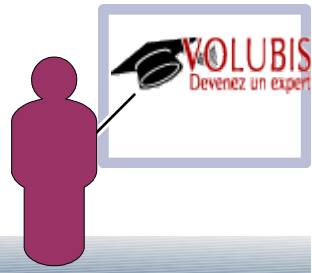
Common Table Expression

- La syntaxe admise par les sous requêtes a évoluée, permettant
  - **ORDER BY et FETCH FIRST n ROWS ONLY**

```
Update vins V set prix = (select priha
from ma_cave where vin_code = V.vin_code
order by dateHA desc fetch first 1 rows
only)
-- si plusieurs prix on prend le dernier
```

  - **UNION** dans les CTE (et les vues) afin de gérer les requêtes récursives

    par exemple le résultat de DSPPGMREF
    ou la hiérarchie de l'entreprise
    ou des liens de nomenclature...



Common Table Expression

- Prenons quelques exemples avec une table des vols d'avion proposés par une agence de voyage

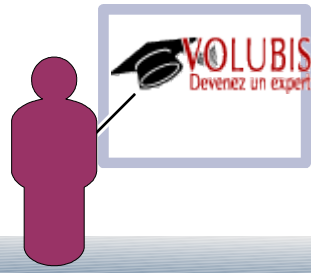
DEPARTURE	ARRIVAL	CARRIER	FLIGHT_NUMBER	PRICE
New York	Paris	Atlantic	234	400
Chicago	Miami	NA Air	2334	300
New York	London	Atlantic	5473	350
London	Athens	Mediterranean	247	340
Athens	Nicosia	Mediterranean	2356	280
Paris	Madrid	Euro Air	3256	380
Paris	Cairo	Euro Air	63	480

. . . / . . .

Ou l'on voit que de New York on peut aller à Madrid en passant par Paris, mais peut-être ensuite aller à Casablanca, etc...

- La syntaxe, depuis la V5R40 consiste à produire un fichier temporaire (temp, par ex.) à l'aide d'une CTE contenant un select qui lit lui-même « temp »

- Regardons...



Common Table Expression

```
with temp (depart, arrivee) as (
    select departure, arrival from flights where
    departure = 'New York'

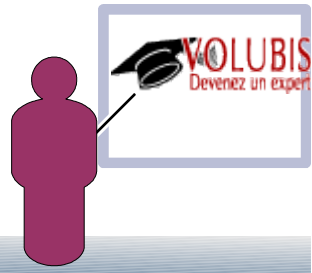
UNION ALL

   select arrivee, arrival from temp
     join flights on arrivee = departure)


select * from temp
```

DEPART	ARRIVEE
New York	Paris
New York	London
New York	Los Angeles
Paris	Madrid
Paris	Cairo
Paris	Rome
London	Athens
Los Angeles	Tokyo
Athens	Nicosia
Tokyo	Hong Kong

- Nous pourrions aussi indiquer le nombre d'escales



Common Table Expression

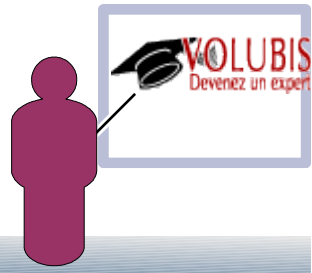
```
with temp (depart, arrivee, escales ) as (
    select departure, arrival, 0 from flights where
    departure = 'New York'

UNION ALL

   select arrivee, arrival, escales + 1 from temp
     join flights on arrivee = departure)


select * from temp
```

DEPART	ARRIVEE	ESCALES
New York	Paris	0
New York	London	0
New York	Los Angeles	0
Paris	Madrid	1
Paris	Cairo	1
Paris	Rome	1
London	Athens	1
Los Angeles	Tokyo	1
Athens	Nicosia	2
Tokyo	Hong Kong	2



Requêtes hiérarchiques

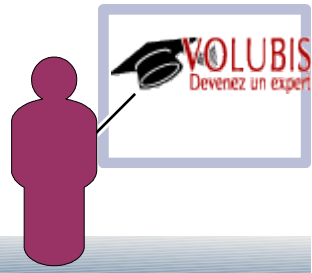
- vous pouvez obtenir désormais la même chose sur une syntaxe plus simple :

```
SELECT departure, arrival
     FROM flights
     START WITH departure = 'Chicago'
     CONNECT BY PRIOR arrival = departure;
```

cette syntaxe est nommée requête hiérarchique et offre quelques différences:

- la requête basée sur une CTE (common Table Expression : clause WITH)
  traite par défaut, les liens par niveau, la requête hiérarchique, par branche

cela est plus clair avec une colonne LEVEL, qui est aussi plus simple à produire avec les requêtes hiérarchiques :



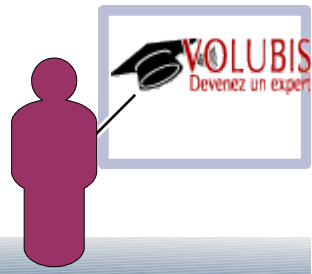
Requêtes hiérarchiques

- résultat

```
SELECT departure, arrival, LEVEL as niveau
      FROM flights
      START WITH departure = 'Chicago'
      CONNECT BY PRIOR arrival = departure
```

DEPARTURE	ARRIVAL	NIVEAU
Chicago	Miami	1
Miami	Lima	2
Chicago	Frankfurt	1
Frankfurt	Vienna	2
Frankfurt	Beijing	2
Frankfurt	Moscow	2
Moscow	Tokyo	3
Tokyo	Hong Kong	4

- La CTE aurait affiché 1/1 puis 2/2/2/2 enfin 3 et 4 (sauf à utiliser SEARCH DEPTH FIRST BY)

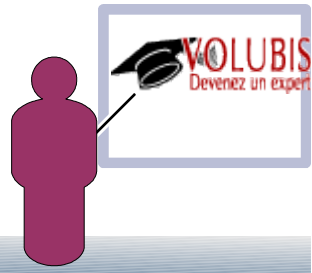


Requêtes hiérarchiques

- les requêtes hiérarchiques proposent en plus :
  - Connect BY ROOT,afin d'afficher le « noeud » racine.
  - un tri, pour les lignes "soeurs" (ayant les mêmes parents).

```
SELECT CONNECT_BY_ROOT departure AS origin, departure, arrival,
      LEVEL niveau,  price prix
  FROM  flights
  START WITH departure = 'New York
  CONNECT BY PRIOR arrival = departure
  ORDER SIBLINGS BY price ASC
```

classe, pour les trajets ayant la même ville d'origine, par prix croissant



Requêtes hiérarchiques

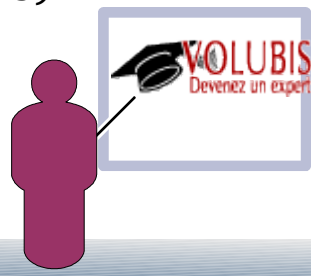
ORIGIN	DEPARTURE	ARRIVAL	NIVEAU	PRIX
New York	New York	Los Angeles	1	330
New York	Los Angeles	Tokyo	2	530
New York	Tokyo	Hong Kong	3	330
New York	New York	London	1	350
New York	London	Athens	2	340
New York	Athens	Nicosia	3	280
New York	New York	Paris	1	400
New York	Paris	Rome	2	340
New York	Paris	Madrid	2	380
New York	Paris	Cairo	2	480

vous remarquerez que le tri principal se fait suivant l'arborescence

(New York->Los Angeles->Tokyo->Hong Kong sur les trois premières)

Ensuite, que les lignes 1 , 4 et 7 (même origine New York) apparaissent

selon leur prix, ainsi que les lignes 8,9 et 10 (origine Paris)



Requêtes hiérarchiques

- La clause **CYCLE** , évitait les boucles infinies avec les CTE

  CYCLE (nom de colonne) SET vartemp = valeur1 DEFAULT valeur0

  quand SQL va se rendre compte qu'il boucle (une ligne déja vue)

  il va attribuer à vartemp (nouvelle variable interne) la valeur "valeur1"

  (dans les autres cas, elle contient "valeur0")

  la ligne va quand même être affichée, mais la boucle se termine

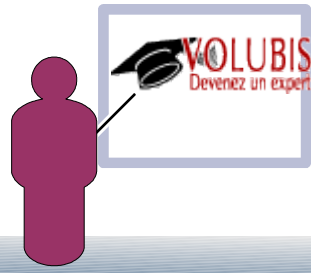
- Avec CONNECT BY, la boucle infinie est automatiquement détectée et signalée

  par SQ20451 : CYCLE DETECTED IN HIERARCHICAL QUERY.

  la requête se termine de manière anticipée (en erreur), sauf à indiquer

  CONNECT BY **NOCYCLE** PRIOR arrival = departure

  la ligne provoquant la boucle sera affichée et la boucle interrompue



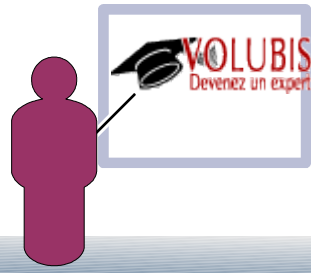
Requêtes hiérarchiques

- la pseudo variable CONNECT_BY_ISCYCLE indique par 1 (oui) ou 0 (non)
  si une ligne provoque une boucle.
  pour éventuellement, filtrer suite à CONNECT BY NOCYCLE

- la pseudo variable CONNECT_BY_ISLEAF indique par 1 (oui) ou 0 (non)
  si une ligne est la dernière (une feuille de l'arborescence)
  une ville n'ayant pas de destination, par exemple.

- enfin la pseudo variable LEVEL que nous avons vu précédemment indique le niveau hiérarchique d'une ligne dans l'arborescence (à partir de 1)

  une astuce peut consister, à l'affichage, à utiliser SPACE(LEVEL)
  ou SPACE(LEVEL * 3), afin d'avoir un affichage incrémenté.



Requêtes hiérarchiques

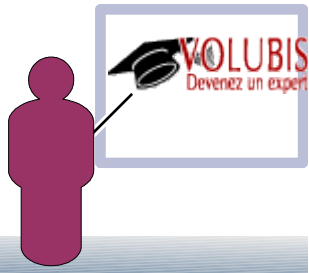
- SYS_CONNECT_BY_PATH(<expression1> , <expression2>)

  retourne le chemin ayant permis d'arriver à cette ligne en concaténant toutes les valeurs de <expression1> , séparées par <expression2>

  par exemple SELECT SYS_CONNECT_BY_PATH(trim(departure), '/') AS chemin

  retourne sous forme de CLOB :

```
/Chicago
/Chicago/Miami
/Chicago/Frankfurt
/Chicago/Frankfurt/Moscow
/Chicago/Frankfurt/Moscow/Tokyo
```

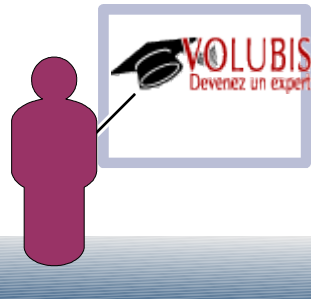


Requêtes hiérarchiques

- Dernier point, on peut faire des requêtes hiérarchiques manipulant plusieurs tables
  - Avec JOIN pour faire des jointures
  - Avec UNION

ex : FLIGHTS contient les vols, TRAINS les trajets en train et vous voulez vous déplacer sans tenir compte du moyen de locomotion !

```
SELECT CONNECT_BY_ROOT departure AS depart, arrival,
    LEVEL-1 AS correspondances
  FROM
      ( SELECT departure, arrival FROM flights
           UNION
        SELECT departure, arrival FROM trains
      ) as trajet
  START WITH departure = 'Chicago'
   CONNECT BY PRIOR arrival = departure
```

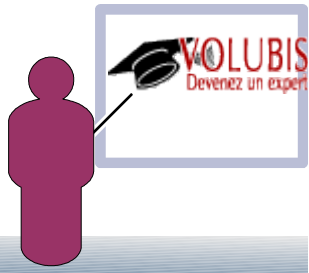


Cryptage de colonnes

- Vous pouvez crypter vos données avec l'un des algorithmes suivants !
  - EnCRYPT_RC2
  - EnCRYPT_TDES
  - EnCRYPT_AES
- En utilisant la syntaxe :

  ENCRYPT_AES(data, pwd, astuce)
  - Data représentant les données à crypter
  - Pwd, la clé avec laquelle crypter et ensuite décrypter
  - Astuce, une astuce mémorisée afin de se rappeler de « pwd »

  - ```
    Insert into clients (numero, nom, CB)
       VALUES(2 , 'Volubis',
       ENCRYPT_AES('1234567890' , 'systemi' , 'Avant IBMi')
       )
    ```

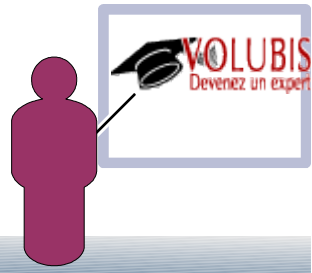

    ```

Cryptage de colonnes

- La zone réceptrice doit être :
  - CHAR ou VARCHAR **FOR BIT DATA**
  - **BINARY ou VARBINARY**
  - **BLOB**

  Pas de CHAR simple avec CCSID

- La longueur insérée sera :
  - Lg de la donnée cryptée
  - + 24 si mot de passe
  - + 56 si astuce
  - Ajusté au multiple de 8 suivant



Cryptage de colonnes

- Le mot de passe peut être fourni à l'avance par :
  - SET ENCRYPTION PASSWORD xxxxxx
    [WITH HINT yyyyy]

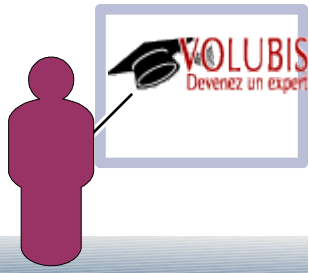
    La fonction est alors utilisée sous la forme

    ENCRYPT_xxx(data)

- L'astuce peut être retrouvée par :
  - GETHINT(variable) → retourne sous forme VARCHAR(32) 'Avant IBMi'

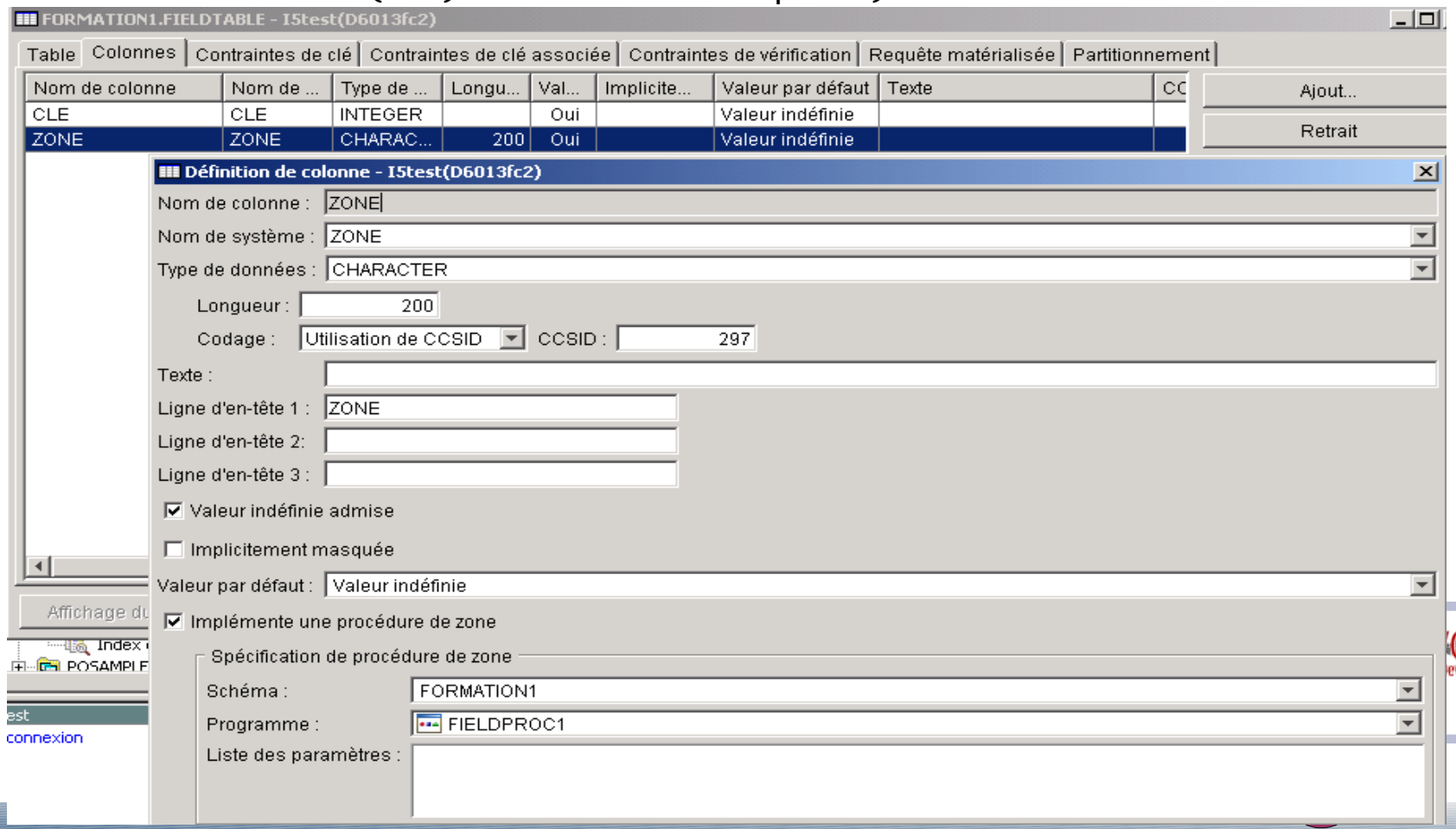
- La donnée peut être retrouvée par :
  - DECRYPT_BIT(variable, [pwd])
  - DECRYPT_BINARY(variable, [pwd])
  - DECRYPT_CHAR(variable, [pwd], [CCSID])
  - DECRYPT_DB(variable, [pwd] , [CCSID])

    si « pwd » n'est pas indiqué, SET ENCRYPTION PASSWORD doit avoir été fixé préalablement.



Cryptage de colonnes

- Depuis la version 7 vous pouvez aussi crypter automatiquement par :
  - Create table fieldtable
    (cle integer ,
    zone char(200) **FIELDPROC** fieldproc1)



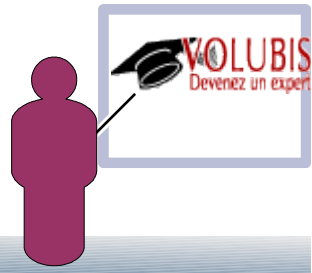
Cryptage de colonnes

- La zone cryptée ne **peut pas** être :
  - une zone de type ROWID
  - une zone numérique avec l'attribut AS IDENTITY
  - une zone de type TIMESTAMP avec AS ROW CHANGE TIMESTAMP
  - un DATALINK
  - une zone avec comme valeur par défaut CURRENT DATE/TIME/TIMESTAMP, USER

- Programmation :
  - la procédure doit être de type PGM/ILE (pas de GAPIII, de java, de programme de service)
  - l'algorithme doit être réversible -;)

  (si la chaîne 'ABCDEF' est transformée en '123456', '123456' doit produire 'ABCDEF')

  - on peut définir des paramètres à envoyer à la procédure, ils sont transmis à chaque appel



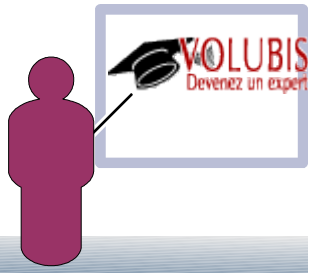
Cryptage de colonnes

- la procédure est appelée lors de la création, pour :
  - valider le type de zone (elle peut refuser de travailler avec des zones numériques, par ex.)
  - indiquer le type de zone à stocker :

    - l'utilisateur saisi du caractère, on stocke du binaire

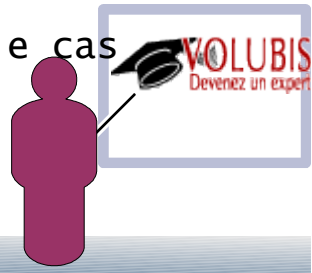
    - lors de la lecture le binaire est transformé à nouveau en caractère
- la procédure est appelée ensuite lors des affectations afin de crypter la donnée, lors des lectures afin de la décrypter

  (Ordres SQL ou Entrées/sorties natives)
- Elle peut décrypter la donnée suivant des conditions, et c'est là que c'est intéressant.(l'utilisateur appartient à la DRH ou pas, par ex.)



Cryptage de colonnes

- Paramètres :
  - une zone fonction indiquant le contexte
    - 8 = appel lors de la création
    - 0 = appel pour crypter
    - 4 = appel pour décrypter
  - une structure décrivant les paramètres
  - une structure décrivant la valeur en clair
    - valeur à utiliser pour le cryptage si fonction=0
    - valeur à produire si fonction=4
  - la valeur en clair
  - une structure décrivant la valeur cryptée
    - valeur à produire si fonction=0
    - valeur à utiliser si fonction=4
  - la valeur cryptée
  - SQLCODE (doit commencer par 38, si erreur, 00000 dans le cas contraire)
  - message complémentaire si SQLCODE <> '00000'

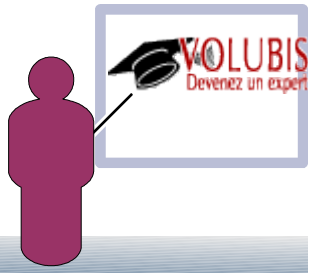


Cryptage de colonnes

Exemple avec un pgm qui inverse les bits (fonction RPG %BITNOT) sur une zone CHAR et ne décrypte que si c'est QSECOFR qui lit.

```
/free
   select;
     when fonction = 8;  // création
      // le type retournée est le même, donc copie de la définition
      decoded_attr= encoded_attr;
     when fonction = 0;  // INSERT => encodage
       lg = decoded_attr.sqlfpLength;
       transforme(decoded_Data : encoded_Data : lg);
     when fonction = 4 ; // SELECT => decodage
       lg = encoded_attr.sqlfpLength;
       if profil = 'QSECOFR';
         transforme(encoded_Data : decoded_Data : lg);
       else;
         %subst(decoded_Data:1:lg) = %subst(encoded_Data:1:lg);
       endif;
     other   ;
       SQLSTATE = '38001';
       message = 'demande inconnue';
   ENDSL;
   *inlr = *on;
 /end-free

 * procédure de codage, inverse tous les bits,x'00 devient x'FF', etc …
 * (algorithme trop simple pour utiliser en production)
Ptransforme       B
D                 PI
D data1                         32767
D data2                         32767
D lg                                5I 0
D i               S                 5I 0
 /free
  for i = 1 to lg;
    // la doc déconseille de crypter les espaces
    if %subst(data1 : i : 1)  = ' ';
       %subst(data2:i:1) = %subst(data1:i:1);
    else;
       %subst(data2:i:1) = %bitnot(%subst(data1:i:1));
    endif;
  ENDFOR;
 /end-free
ptransforme       E
```



Cryptage de colonnes

- La documentation déconseille de crypter les espaces, en effet quand vous comparez à la zone, une constante plus courte, le système complète l'information la plus petite par des espaces.
- Utilisation:

```
Insert into fieldtable values (1 , 'coucou')

Insert into fieldtable values (2 , 'autre test')
```

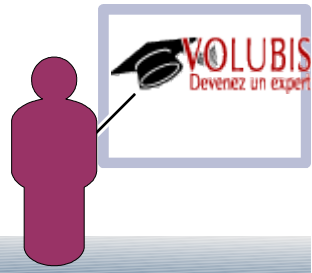
- Sous QSECOFR

```
Première ligne à afficher . . _____
....+....1....+....2....+....3....+....4..
          CLE   ZONE
            1   coucou
            2   autre test
********  Fin de données  ********
```

- Sous un autre profil

Contenu de FORMATION1.FIELDTABLE - I5test(D6013fc2)

	CLE	ZONE
1	1	àÑ$àÑ$
2	2	=$*Ã: *:)*



Cryptage de colonnes

- les tris peuvent être perturbés sur une zone cryptée :

  ex SELECT *FROM FIELDTABEL ORDER BY ZONE

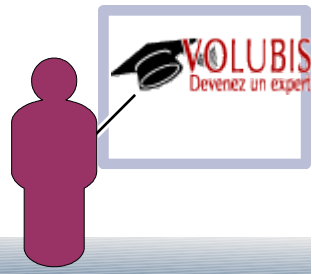
-> sous QSECOFR (la zone est décryptée)

```
Première ligne à afficher . . _____
....+....1....+....2....+....3....+....4....+....5....
          CLE   ZONE
            2   autre test
            1   coucou
********  Fin de données  ********
```

-> sous un autre profil (la zone reste cryptée)

```
Première ligne à afficher . . _____
....+....1....+....2....+....3....+....4....
          CLE   ZONE
            1   àÑ$àÑ$
            2   =$*Ã: *:)*
********  Fin de données  ********
```

- lors d'un CPYF la procédure sera appelée (même sur DSPPFM), ainsi que lors d'un CREATE TABLE AS (SELECT ...)



Cryptage de colonnes

- une option de QAQQINI destinée à l'optimiseur, indique si la zone doit être décryptée systématiquement : FIELDPROC_ENCODED_COMPARISON :
  - *ALLOW_EQUAL (dft)

on crypte les constantes comparées plutôt que de décrypter la valeur du fichier, pour les comparaisons, GROUP BY et DISTINCT.

la fonction doit être déterministe (retourner toujours le même résultat pour la même valeur) et les valeur retournées peuvent ne pas être triées

  - ALLOW_RANGE

on crypte les constantes comparées plutôt que de décrypter la valeur du fichier, comme ALLOW_EQUAL, mais aussi pour MIN, MAX et ORDER BY

la fonction doit être déterministe et les valeur retournées doivent être triées (significatives pour un tri)

  - *ALL

la procédure est appelée le moins souvent possible (on crypte les constantes comme ALLOW_RANGE) pour toutes les opérations y compris LIKE

  - *NONE

la procédure de cryptage est appelée systématiquement et on travaille avec les valeurs en clair.

CQE travaille toujours de cette manière, quelque soit la valeur dans QAQQINI.