Avec mes remerciements à Robert PLAGNARD et à Gregory ADAM

INTEGER, BITS, fonctions binaires, ...[1]

Sommaire

1 Ce document a été créé avec Star Office.

1. Généralités

Depuis VFP6 on peut utiliser des champs de type INTEGER (entier) dans les tables. Ces champs occupent 4 octets (4 caractères) et peuvent contenir une valeur numérique entière comprise entre -2147483647 et 2147483647. Ils sont très efficaces pour les clefs et les index.

Mais ils ont aussi une autre utilité : contenir 32 bits 'indépendants' que l'on va pouvoir utiliser comme autant de 'variables' logiques ou mieux comme un tableau de logiques.

C'est ce que Mike GAGNON appelle la 'double vie des integers'.

Nous allons donc étudier l'utilisation des valeurs entières et celle des bits qui, vous le verrez, peut être très intéressante.

Il y a, dans l'utilisation de ce type de donnée, quelques 'curiosités' que je vous exposerai. Il y a aussi dans VFP6 quelques bugs. Grâce à Robert PLAGNARD, j'ai découvert que la gestion des bits avait été très largement améliorée dans VFP9; Grâce à Gregory ADAM j'ai découvert avec stupéfaction la gestion du bit de signe dans VFP !. cela m'amène à corriger ce document ...

2. Utilisation des entiers en tant que valeur numérique

2.1. Description

Créons une table (le I signifie Integer mot anglais pour 'entier'):

```
CREATE TABLE essai_integer (champ1 I, champ2 I)
```

on va créer deux lignes :

```
INSERT INTO essai_integer VALUES(0,0)
INSERT INTO essai_integer VALUES(2147483647,16384)
BROWSE
```

Vous voyez bien vos deux lignes, chacune contenant deux valeurs numériques. On a stocké de la manière la plus condensée possible ces deux valeurs numériques dans une table (on aurait voulu les enregistrer dans un champ Numérique on aurait du les définir comme N(10,0) ce qui aurait occupé 10 octets (caractères) au lieu de 4).

On peut faire des calculs sur ces valeurs entières : par exemple

```
REPLACE champ1 WITH champ1 – 1, champ2 WITH champ2*2
```

2.2. Curiosités et anomalies

Restez sur la ligne de votre table et faites :

```
REPLACE champ1 WITH champ1 +2
```

dans la fenêtre du browse, le champ s'est rempli d'étoiles et un message d'erreur est apparu : « dépassement de capacité numérique. Données perdues ». On vient effectivement de franchir la limite supérieure de la fourchette de validité. Le même problème existe pour les valeurs négatives. Quand on utilise les entiers, il faut donc bien penser à cette limitation (enfin les limites sont quand même assez élevées et ce dépassement de capacité plutôt rare).

Je vous demande de mémoriser ce point car il va nous enquiquiner un peu plus loin !

On peut imprimer les valeurs d'un champ entier comme on peut le faire avec n'importe quel champ numérique. Avec un petit 'plus' : l'impression en hexadécimal :

? transform(champ1, "@0") && c'est le chiffre 0 derrière le @ et non la lettre O

Mais il y a un 'bug' dans VFP6 (corrigé dans VFP9) : on ne peut pas imprimer une valeur entière négative en hexadécimal. Un autre bug (très curieux) est resté dans la version 9 : on ne peut pas faire un replace avec une valeur négative écrite en hexadécimale : REPLACE champ1 WITH 0xFFFFFFFF provoque une erreur de dépassement de capacité numérique alors que REPLACE champ1 WITH -1 est parfaitement accepté et juste alors que les deux valeurs -1 et 0xFFFFFFFF sont strictement identiques !!!

2.3.Notation hexadécimale

Les vieux de la vieille ont l'habitude de cette notation hexadécimale et elle est très intéressante quand on travaille sur des entiers (et au chapitre suivant sur des mots de 32 bits). 4 bits permettent d'écrire 16 valeurs; les 10 premières valeurs sont représentées comme dans la notation décimale par les chiffres 0 à 9, les 6 valeurs suivantes sont représentées par les lettres A à F.

Lorsque l'on compte, on va donc avoir : 0, 1, ..., 9, A, B, ... F.

Hexadécimal	décimal	binaire
0	0	0000_b
1	1	0001_b
2	2	0010_b
3	3	0011_b
4	4	0100_b
5	5	0101_b
6	6	0110_b
7	7	0111_b
8	8	1000_b
9	9	1001_b
A	10	1010_b
B	11	1011_b
C	12	1100_b
D	13	1101_b
E	14	1110_b
F	15	1111_b

Dans VFP, un entier est sur 4 octets[2] donc on va pouvoir l'écrire avec 8 caractères hexadécimaux (8 caractères * 4bits = 32 bits = 4 octets). Voici quelques exemples :

Hexadécimal	décimal
0x00000000	0
0x00000001	1
0x000000FF	255
0x00000100	256
0x0000FFFF	65535
0x00010000	65536
0x7FFFFFFF	2147483647

La valeur 2147483647 est particulière : c'est la limite supérieure pour un nombre entier !

Le bit 31 (parce que l'on commence à 0 et non à 1 pour la numérotation des bits) est donc le bit de signe : s'il est à 0, le nombre est positif, s'il est à 1 le nombre est négatif. Avec une particularité due à l'algèbre binaire. Ci-dessous quelques valeurs négatives :

Hexadécimal	décimal
0x80000000	**-2147483648**
0x80000001	**-2147483647**
0xFFFFFFFF	-1
0xFFFFFFFE	-2

Je reviens sur notre dépassement de capacité de tout à l'heure, lorsque l'on a ajouté 2 à 2147483646, soit 0x7FFFFFFE, on a obtenu 0x80000000 qui est une valeur négative. Le processeur a détecté cette anomalie (c'est une erreur 'hard') et VFP l'a correctement interprétée. Cette anomalie va nous gêner dans nos manipulations binaires; il faut la garder dans un petit coin de mémoire.

2.4.Utilisation dans les index

Un avantage important des entiers (integer) est leur utilisation dans les clefs. En effet, ils peuvent représenter une valeur très importante (supérieure à celle d'un N(9,0)) et être traités très rapidement : il faut en effet que quelques opérations au microprocesseur pour comparer deux entiers (car, dans les ordinateurs 32 bits, les microprocesseurs travaillent justement sur 32 bits) alors que si il doit comparer deux valeurs numériques, il doit d'abord les convertir du décimal (mode dans lequel elles sont enregistrées dans une table) en binaire puis seulement après les comparer. D'autre part le nombre d'octets qui doit transiter par le réseau est aussi singulièrement diminué ...

2 En VFP9, on peut aller à 8 ...

3. Utilisation des entiers en tant que 31 valeurs logiques

3.1. Description, définitions

Un entier peut être vu comme un ensemble de 32 bits numérotés de 0 à 31. Chaque bit pourra être mis à 1 (.T. par définition) par la commande BITSET(), mis à 0 (.F. par définition) par la commande BITCLEAR() et testé par la commande BITTEST().

La numérotation de 0 à 31 se fait par convention de droite à gauche, le bit 0 étant celui de poids le plus faible. Mais 'de droite à gauche' ne veut rien dire par ce qu'en fait les octets ne sont pas rangés 'dans l'ordre' dans un mot de 4 octets [3]. La seule chose à retenir est que si vous donnez une signification au bit 1, si celui-ci que vous mettrez à 1 ou à 0 et que vous testerez.

Le titre de ce paragraphe ne comporte pas d'erreur : tapez le code suivant :

```
CREATE TABLE essai_bits (champ I) && crée une table avec un champ entier
INSERT INTO essai_bits VALUES (0x00000000) && initialisé à 0
BROWSE  && comme on peut le vérifier
```

on veut mettre le bit 31 à 1 :

```
REPLACE champ WITH BITSET(champ, 31)
```

on reçoit une erreur « dépassement de capacité numérique » qui pourrait être normale (on a effectivement fait passer la valeur numérique correspondante en négatif) si la séquence suivante provoquait l'erreur alors qu'elle ne le fait pas (VFP6 et VFP9) :

```
REPLACE champ WITH BITSET(champ, 0)
REPLACE champ WITH BITSET(champ, 31)
```

Pour résumer, VFP pense bien à neutraliser l'erreur de dépassement de capacité numérique lorsque l'on travaille sur les bits, mais cette neutralisation ne fonctionne pas si la valeur initiale est 0. C'est pourquoi, dans toutes mes applications et aussi dans toute la suite de ce document, le bit 0 de chaque entier sera systématiquement mis à 1.

Il nous reste donc plus que 31 bits librement utilisables, d'où le titre ...

En haut de ce paragraphe, je vous ai introduis les commandes BITSET(), BITCLEAR() et BITTEST(). Deux autres commandes vont nous être utiles : BINTOC() qui ne fait rien d'autre que de dire à VFP de traiter les 4 octets d'un entier comme 4 caractères (cette fonction ressemble à la clause CAST de SQL SERVER) et CTOBIN() pour l'inverse. Remarquez que BINTOC() accepte un paramètre donnant le nombre d'octets à traiter; on verra plus loin que l'on peut travailler sur 16 ou même 8 bits.

3 On reverra ce point avec le paramètre R de BINTOC() plus loin dans ce document

3.2.BITSET(), BITCLEAR(), BITTEST()

Simple rappel :on met à 1 un bit avec la fonction BITSET(), on le met à 0 avec la fonction BITCLEAR() et on le teste avec la fonctions BITTEST() :

```
leaccu = 0
leaccu = BITSET( m.leaccu,4)
leaccu = BITCLEAR(m.leaccu,6)
llresult = BITTEST( m.leaccu,4) OR BITTEST(m.leaccu,6)
```

Plus haut vous avez vu un exemple avec REPLACE.
Avec VFP9, vous disposez d'une deuxième syntaxe pour d'autres types de données (varbinary et blob).

3.3.BITNOT(), BITAND(), BITOR(), BITXOR(), BITLSHIFT(), BITRSHIFT()

Les fonctions précédentes travaillaient sur un seul bit, voyons maintenant des fonctions qui travaillent sur un ou deux mots (4 octets).

BITNOT(lemot) renvoie un mot dont chaque bit est à 1 si le bit correspondant dans lemot est à 0 et vice-versa (c'est la fonction booléenne NON);

BITAND(lemot1, lemot2) renvoie un mot dont chaque bit est à 1 si les bits correspondants dans lemot1 ET dans lemot2 sont à 1 (c'est la fonction booléenne ET); on se servira de cette fonction pour tester plusieurs bits d'un même mot en une seule opération[4]; En VFP6, vous ne pouvez utiliser que deux expressions; en VFP9, vous pouvez en utiliser plusieurs;

BITOR(lemot1, lemot2) renvoie un mot dont chaque bit est à 1 si les bits correspondants dans lemot1 OU dans lemot2 sont à 1 (c'est la fonction booléenne OU); on se servira de cette fonction pour mettre à 1 plusieurs bits d'un même mot en une seule opération; En VFP6, vous ne pouvez utiliser que deux expressions; en VFP9, vous pouvez en utiliser plusieurs;

BITXOR(lemot1, lemot2) renvoie un mot dont chaque bit est à 1 si les bits correspondants dans lemot1 OU dans lemot2 sont à 1 sans que ces deux bits soient simultanément à 1 (c'est la fonction booléenne OU EXCLUSIF); En VFP6, vous ne pouvez utiliser que deux expressions; en VFP9, vous pouvez en utiliser plusieurs;

BITLSHIFT(lemot, ln) renvoie un mot en décalant chaque bit de lemot de ln positions vers la gauche. En binaire, c'est la multiplication par 2 si ln=1, par 4 si ln=2, etc .....;

BITRSHIFT(lemot, ln) renvoie un mot en décalant chaque bit de lemot de ln positions vers la droite. En binaire, c'est la division par 2 si ln=1, par 4 si ln=2, etc .....;

3.4.BINTOC(), CTOBIN()

BINTOC() permet de ranger une valeur binaire dans une variable de type caractère. Cette fonction ne fait en fait presque aucun calcul (contrairement à STR()), elle fait juste une conversion de type (fonction CAST).

CTOBIN() range une chaîne de caractères (1, 2 ou 4 caractères) dans une variable binaire. Comme la précédente, cette fonction ne fait presque aucune conversion.

Les améliorations de VFP9 méritent que l'on s'y attarde un peu et que l'on

4 Voir les tables de vérité en annexe

distingue bien les versions 6 et 9[5].

En VFP6 :

- **BINTOC() admet un 2éme paramètre numérique indiquant le nombre de caractères du résultat. La valeur peut être 1, 2 ou 4. Cette fonction étant surtout utilisée dans les index, les développeurs de Fox y ont introduit une particularité : le bit de signe est systématiquement inversé (par une opération XOR) de façon à ce que les nombres négatifs soient placés avant les nombres positifs (ce qui est naturel).**
- **CTOBIN() inverse aussi le bit de signe avant de ranger la valeur dans une variable binaire; on retrouve ainsi le bon signe.**

En VFP9 :

- **le 2éme paramètre de BINTOC() peut être numérique comme dans VFP6, il peut être aussi de type caractère pour inclure les améliorations suivantes :**
  - **une indication de format des données : la conversion n'a pas forcément pour résultat un entier mais cela peut être une valeur monétaire, un flottant, ...**
  - **'S' indique que le bit de signe ne doit pas être inversé (ce qui correspond donc à une conversion binaire normale) :**
    BINTOC(1, 1) donne un octet de valeur 0x81 (10000001)
    BINTOC(1, '1S') donne un octet de valeur 0x01 (00000001)
    BINTOC(-1, 1) donne un octet de valeur 0x7F (01111111)
    BINTOC(-1, '1S') donne un octet de valeur 0xFF (11111111)
  - **'R' indique les octets sont rangés dans l'ordre utilisé par la processeur. Cette indication est utile lorsque l'on doit interfacer une valeur avec un API.**

  Note : on peut écrire BINTOC(toto,1) ou BINTOC(toto, '1')

- **sans indication contraire, CTOBIN() inverse aussi le bit de signe avant de ranger la valeur dans une variable binaire; on retrouve ainsi le bon signe. Mais comme pour bintoc(), on peut utiliser une indication de format, le 'S' et le 'R'**

Ces deux fonctions sont très utiles. En plus de leur utilisation pour améliorer l'efficacité des index, elles sont utiles pour les cas suivants:

- **travailler sur des champs entier oblige à travailler sur des mots d'un, deux ou quatre octets (jusqu'à 8 sous VFP9). Mais si on range les valeurs binaires dans des chaînes de caractères, celles-ci n'ont pas de limites (même un champ MEMO convient). Ainsi, dans mon application de gestion d'école, je dois enregistrer les présences à la cantine de chaque élève : j'ai un champ de 53 caractères qui représentent les 53 semaines (max) de l'année. Le numéro de la semaine me donne l'octet à traiter ou a tester et dans cet octet, le bit 1 correspond au lundi, le bit 2 au mardi, le bit 3 au mercredi, ... La cantine pourra même ouvrir le samedi ou le dimanche !!![6]**
- **De même si vous voulez enregistrer un tableau de valeurs entières, il sera plus facile de le faire dans une chaîne (en gardant la possibilité de travailler par 1, 2 ou 4 (ou 8) caractères par valeur en fonction de la 'précision' (et du format) demandée !).**

5 Je ne connais pas VFP7 et VFP8 !!

6 Lorsque je passerai cette application sous VFP9, je m'intéresserai au type VARBINARY pour enregistrer ces 53 octets ...

3.5.quelques exemples

a) mettre à 1 plusieurs bits

```
* mettre à 1 les bits 0,1, 4, 6 d'un champ entier
REPLACE ALL champ WITH BITOR(champ, 0x00000053)
```

b) tester plusieurs bits

```
* tester si les bits 8, 14 et 20 sont à 1
IF BITOR(champ, 0x00104100) <> 0
* surtout ne pas mettre IF BITTEST(champ,8) AND BITTEST(..... !!!!
```

c) ranger 4 valeurs entières inférieures à 255 dans un entier

```
REPLACE champ WITH CTOBIN(BINTOC(m.valeur1,1)+ ;
  BINTOC(m.valeur2,1)+ BINTOC(m.valeur3,1)+ BINTOC(m.valeur4,1))
```

pour récupérer ces 4 valeurs on écrira

```
valeur1 = CTOBIN( LEFT( BINTOC(champ,4),1))
valeur2 = CTOBIN( SUBSTR( BINTOC(champ,4),2,1))
valeur3 = CTOBIN( SUBSTR( BINTOC(champ,4),3,1))
valeur4 = CTOBIN( SUBSTR( BINTOC(champ,4),4,1))
```

note : une de ces valeurs ne doit pas dépasser 127 pour ne pas risquer un dépassement de capacité numérique : laquelle ?

d) compter le nombre de bits à 1 dans un mot

```
lnnombre = 0
FOR lni = 0 TO 31
  lnnombre = m.lnnombre + IIF( BITTEST(champ, m.lni), 1, 0)
ENDFOR &&* lni = 0 TO 31
? m.lnnombre
```

e) enregistrer une information 'jour du mois' + matin/après midi en 1 octet

il y a 31 jours dans un mois. On peut codifier la valeur sur 5 bits (32 possibilités) et le matin et/ou après-midi sur deux bits (4 possibilités) : 0 ni le matin ni l'après-midi, 1 le matin, 2 l'après-midi, 3 le matin et l'après-midi. Au total on a donc besoin de 7 bits ce qui tient dans un seul octet (caractère). Comme le bit 7 (le 8éme bit) ne sera jamais mis à 1, on n'aura pas de souci avec le 'bug'.

```
CREATE TABLE uncar FREE (champ C(1))
APPEND BLANK
BROWSE
* le 26 du mois matin + après-midi :
REPLACE champ WITH BINTOC( BITOR( BITLSHIFT(26,2), 3), 1)
* autres manières (elles donnent toutes le même résultat)
APPEND BLANK
REPLACE champ WITH BINTOC( BITLSHIFT(26,2)+ 3, 1)
APPEND BLANK
REPLACE champ WITH BINTOC( BITLSHIFT(26,2)+ BITLSHIFT(3,0), 1)
APPEND BLANK
```

```
REPLACE champ WITH BINTOC( BITOR( BITLSHIFT(26,2), BITLSHIFT(3,0)), 1)
APPEND BLANK
REPLACE champ WITH BINTOC( 26*4 + 3, 1)
```

4. Annexes

4.1. tables de vérité

dans les tableaux ci-dessous, vous trouverez les résultats des fonctions ET (AND), OU (OR) et OU EXCLUSIF (XOR). L'abscisse et l'ordonnée correspondent aux 2 bits utilisés dans le calcul, la cellule contient le résultat.

ET (AND)

	0	1
0	0	0
1	0	1

OU EXCLUSIF (XOR)

	0	1
0	0	1
1	1	0

OU (OR)

	0	1
0	0	1
1	1	1

4.2. Comment enregistrer des entiers dans un fichier texte !

Dans le petit programme ci-dessous on va enregistrer 10 entiers dans un fichier texte; cela représente 10 valeurs entières ou 310 bits dans une chaîne de 40 caractères !

```
mhandle = FCREATE("test_bits.txt")
IF m.mhandle >=0
  DIMENSION tablog(10), tablu(10)
  tablog(1) = 0x00000000
  tablog(2) = 0x00000001
  tablog(3) = 0x00000020
  tablog(4) = 0x00000400
  tablog(5) = 0x00007000
  tablog(6) = 0x000A0000
  tablog(7) = 0x00B00000
  tablog(8) = 0x0C000000
* en VFP6 ET VFP9 si on écrit des valeurs négatives :
  tablog(9) = 0xFFFFFFFF
  tablog(10) = 0xFFFFFF80
* le FWRITE se plante par contre si on écrit
  tablog(9) = -1
  tablog(10) = -128
* ce qui est strictement équivalent, le FWRITE ne se plante pas !!!
* Il y a donc encore des progrès à faire !!
  FOR m.i = 1 TO 10
    = FWRITE(m.mhandle, BINTOC(tablog(i),4), 4)
  ENDFOR &&* m.i = 1 TO 10
  = FCLOSE( m.mhandle)
  MODIFY FILE test_bits.txt NOMODIFY && c'est pour voir le résultat !
* on reouvre le fichier texte pour lire les valeurs enregsitrées ...
  mhandle = FOPEN("test_bits.txt",0)
```

```
  IF m.mhandle >=0
    FOR m.i = 1 TO 10
      tablu(m.i) = CTOBIN( FREAD(m.mhandle,4))  && le FREAD ne pose aucun problème
    ENDFOR &&* m.i = 1 TO 10
  ENDIF && m.mhandle >=0
  = FCLOSE( m.mhandle)
* on compare les deux tableaux qui doivent être identiques
  FOR m.i = 1 TO 10
    * en VFP6, le TRANSFORM d'une valeur négative donne des ******
    ? STR(m.i,3)+ IIF(tablog(m.i) = tablu(m.i), "  ok " + TRANSFORM(tablu(m.i), "@0"), ;
     " nak "+TRANSFORM(tablog(m.i), "@0")+TRANSFORM(tablu(m.i), "@0"))
  ENDFOR &&* m.i = 1 TO 10
ELSE
  WAIT WINDOW "impossible de créer test_bits.txt" NOWAIT
ENDIF && m.mhandle >=0
```

ATTENTION : si vous tentez de relire le fichier texte avec un éditeur hexadécimal, vous n'allez pas retrouver les valeurs enregsitrées car n'oubliez pas que le bit de signe a été inversé !! En VFP9, l'utilisation de l'indicateur S vous permettrait de pouvoir relire ces valeurs correctement.

Fin du document