DE "DO WHILE !EOF()" à SQL

Visual FOXPRO, de part son héritage et son histoire, est très facile d'accès. En quelques minutes, on apprend les quelques commandes simples qui permettent de créer une table et de l'exploiter. Ceci a un petit inconvénient : à partir du moment où 'cela marche', on ne cherche pas à voir si on peut faire mieux. Le but de cette contribution est de vous montrer qu'il y a beaucoup mieux que DO WHILE ! EOF() !!

Dans DBASE3, nous n'avions qu'une seule possibilité pour parcourir les lignes d'une table :

```
GO TOP
DO WHILE !EOF()
      IF condition THEN

      ENDIF
      SKIP
ENDDO
```

et certains d'entre-vous utilisent encore cette ~~méthode~~ horreur (qui fonctionne mais … lentement). Une autre possibilité moyenâgeuse de filtrer les lignes étaient, justement, d'utiliser la commande SET FILTER; comme dans :

```
SET FILTER TO condition
GO TOP
DO WHILE !EOF()

      SKIP
ENDDO
SET FILTER TO
```

FOXPRO a très rapidement apporté une commande spécifique pour le parcourt des tables : SCAN; puis lui ajouté des clauses supplémentaires. Le parcours simple d'une table s'écrit :

```
SCAN

ENDSCAN
```

Quoique VFP le fasse automatiquement, il faut prendre l'habitude de resélectionner la table supportant le parcours avant le ENDSCAN surtout si on change de zone de travail entre le SCAN et le ENDSCAN :

```
SELECT matable
...
SCAN
      ...
      SELECT autrezone
      ...
      SELECT matable
ENDSCAN
```

A l'intérieur d'un tel parcours on a souvent une instruction IF qui sert à détecter les lignes pour lesquelles on veut faire un traitement :

```
SCAN
      IF condition1
            ...
      ENDIF
ENDSCAN
```

Cette séquence est très avantageusement (car optimisée) remplacée par la clause FOR :

```
SCAN FOR condition
      ...
ENDSCAN
```

Et comme je suis un peu tatillon, je vais indiquer que l'on veut réellement parcourir toute la table en ajoutant la clause ALL qui veut dire 'tous les enregistrements' (on verra un peu plus bas, qu'il existe d'autres clauses) :

```
SCAN ALL FOR condition
      ...
ENDSCAN
```

Avec cette écriture, on voit mieux qu'il est inutile de mettre la commande GO TOP avant de commencer le SCAN.

Cette méthode a un inconvénient : on parcourt toutes les lignes de la table alors que, peut-être, on n'en traitera que quelques-unes. Si la table est indexée (ce qui est logique), on peut rechercher la première ligne à traiter en utilisant l'index puis on parcourt les lignes qui suivent comme dans :

```
IF SEEK(maclef)
      SCAN REST FOR condition

      ENDSCAN
ENDIF
```

La clause REST indique que l'on commence le parcourt à partir de la ligne où on est. La fonction SEEK() est équivalente à l'instruction SEEK mais elle rend .T. si la recherche a été fructueuse (on n'a pas à écrire `SEEK clef` suivi de `IF FOUND()`).

On a bien éliminé un grand nombre de lignes avant de commencer le parcours, mais on l'exécute sur toutes les lignes qui suivent jusqu'à la fin de la table Alors que, peut-être, on a une condition qui permet de s'arrêter plus rapidement : on va utiliser la clause WHILE (en français : TANT QUE). Le parcours s'effectue tant que la condition dans le while est vraie :

```
IF SEEK(maclef)
      SCAN REST WHILE condition
            ...
      ENDSCAN
ENDIF
```

Et on peut mettre les 2 clauses :

```
IF SEEK(maclef)
      SCAN REST FOR condition1 WHILE condition2
            ...
      ENDSCAN
ENDIF
```

Les conditions dans les clauses FOR et WHILE peuvent être complexes et faire appel à des

fonctions utilisateur (dans ce dernier cas, il faut que la fonction resélectionne la zone de travail si elle la change). On peut par exemple parcourir toutes les lignes d'une table qui vérifie une condition sur une autre table :

```
SELECT matable
SCAN ALL FOR SEEK(matable.unchamp, "autrezone", "index")
      ...
      SELECT matable
ENDSCAN
```

La fonction SEEK() positionne le pointeur de ligne sur la table ouverte dans la zone "autrezone"; ce qui prend un peu de temps. Si on n'a juste besoin de l'information 'la ligne existe', on peut avantageusement utiliser la fonction INDEXSEEK comme dans

```
SELECT matable
SCAN ALL FOR INDEXSEEK(matable.unchamp, .F., "autrezone", "index")
      ...
      SELECT matable
ENDSCAN
```

Le .F. en deuxième paramètre indique que l'on ne veut pas bouger le pointeur de ligne; la réponse est donc très rapide mais on ne peut pas utiliser "autrezone" dans le parcours (puisqu'on n'est pas sur la ligne 'trouvée'). Attention, la fonction INDEXSEEK a quelques particularités, veuillez consulter l'aide à son sujet.

ASTUCE : Si on n'a besoin que d'une seule information dans la table ouverte dans "autrezone", on peut créer un index temporaire filtré sur l'information comme dans :

```
SELECT autrezone
INDEX ON unchamp FOR unecondition TAG temporaire
SELECT matable
SCAN ALL FOR INDEXSEEK(matable.unchamp, .F., "autrezone", "temporaire")
      ...
      SELECT matable
ENDSCAN
```

On a maintenant fait le tour des possibilités de parcours d'une table en utilisant les instructions standard de VFP. Nous allons utiliser une autre voie : les requêtes SQL.