AtoutFox 2019

Synthèse sur le fonctionnement interne de Visual Foxpro

Référence du document originel complet :

http://foxpert.com/docs/howfoxproworks.en.htm

En vert ce qu'il est essentiel de retenir

Avertissement:

Ce document fait le point sur le fonctionnement interne de Visual FoxPro. Le code source de FoxPro étant la propriété intellectuelle de Microsoft, il ne peut pas être publiquement divulgué. Toute personne qui sait réellement comment Visual FoxPro fonctionne n'a pas le droit de parler de cela en ayant signé un accord de non-divulgation (NDA). Les informations suivantes ont donc été rassemblées auprès de diverses sources publiques. Certaines informations se trouvent dans la bibliothèque MSDN (certains articles n'existent que dans les anciennes versions de MSDN Library), d'autres proviennent d'exemples de codes fournis par Microsoft mais la plupart proviennent de tests et d'observations constatées par de nombreux développeurs sur différents forums. Les différences de comportement des différentes versions permettent notamment de tirer des conclusions sur la structure interne de VFP.

L'index de la « table des noms » (NTI)

Dans Visual FoxPro, nous pouvons nommer divers composants /éléments. Pour ces éléments, Visual FoxPro attribue un « nom ». Ces éléments sont des variables (pas des propriétés), des noms de tableaux, des procédures, des fonctions, des alias de tables, des noms de champs et des objets (pas des classes). Chacun de ces éléments a un nom et une portée. La visibilité (portée) d'une variable, par exemple, dépend de son type, alors que la portée d'un alias est la session de données. Un nom de champ doit être unique dans une table et les procédures sont définies dans un fichier programme.

Chaque fois que vous créez une variable, ouvrez une table, etc.., Visual FoxPro crée une nouvelle entrée dans une liste interne : la «table de noms». La position dans cette liste est l'index de la «table de noms» ou NTI pour faire court. Dans cette liste, chaque «nom» se voit attribuer un numéro unique compris entre 1 et 65 000, car il est conservé en tant que valeur 16 bits. Il n'y a qu'une seule liste globale. C'est pourquoi vous ne pouvez créer qu'au maximum 65 000 variables. Etant donné que les alias et les noms d'objets sont également inclus dans cette liste, le nombre réel de variables est réduit du nombre d'objets instanciés et des zones de travail allouées.

La gestion de cette liste a été optimisée dans les différentes versions de FoxPro. Lorsque vous libérez un « nom » en fermant une table ou parce que vous n'avez plus besoin d'une variable, Visual FoxPro ne supprime pas l'entrée immédiatement. Il y marque l'entrée comme non valide, tout comme pour les enregistrements supprimés d'une table.

Le garbage collector :

Les éléments sont finalement supprimés de la liste « table des noms » par un processus appelé « Garbage Collector ». Ce terme désigne le processus de suppression d'entrées non valides dans la liste, d'entrées de cache obsolètes, de mémoire fragmentée, etc.. Visual FoxPro, pendant son temps d'attente d'activité, exécute ce « garbage collector » . Celui-ci est effectué chaque fois que Visual FoxPro est dans une condition d'attente provoquée par READ, READ EVENTS, INKEY () ou WAIT WINDOW avec NOWAIT.

Vous pouvez utiliser cette astuce WAIT WINDOW "" NOWAIT., pour obliger Visual FoxPro à nettoyer la mémoire ou alors utiliser la commande SYS(1104) qui lance le « garbage collector ».

Cependant, dans son temps d'attente d'activité, Visual FoxPro fait plus qu'exécuter le « garbage collector », il traite aussi traiter les messages Windows. Lors de l'exécution du programme, Visual FoxPro n'est pas dans son temps d'attente d'activité et n'effectue donc pas de nettoyage de la mémoire. Cela a de lourdes conséquences sur les performances, mais également sur la stabilité. Chaque fois qu'une nouvelle entrée est ajoutée à la « table de noms », Visual FoxPro doit rechercher toutes les entrées existantes pour rechercher des « noms » en potentiel conflit. Pour les variables, cela signifie vérifier si une nouvelle variable ayant une portée inférieure existe déjà, car vous ne pouvez pas, par exemple, créer une variable PUBLIC lorsqu'une variable LOCAL de même nom existe. Pour les alias, cela signifie vérifier que le nom d'alias n'est pas déjà utilisé dans la session de données en cours. Ce processus de recherche entraîne une dégradation exponentielle des performances. La création du premier objet se mesure en millisecondes, voire en nanosecondes, mais Visual FoxPro mettra plusieurs minutes pour créer un objet lorsqu'il en détient déjà des dizaines de millier.

Il faut dans le code source de vos applications :

- *limiter au strict nécessaire le nombre d'entrées dans la « table des noms ». Ce qui se traduit souvent par l'abandon de commandes de type SCATTER GATHER MEMVAR au profit de SCATTER GATHER NAME, de n'ouvrir que les tables nécessaires, etc..*
- *permettre à Visual Foxpro de régulièrement déclencher son « garbage collector » en utilisant par exemple un WAIT WINDOWS '' NOWAIT*
- *ou bien déclencher un « garbage collector » avec un SYS(1104) après ou pendant des instructions de codes qui mobilisent en permanence les ressources de Visual Foxpro (routines d'importation, fournisseurs de services, boucles do enddo, etc..)*

Depuis Visual FoxPro 6.0 le comportement de la gestion de la « table de noms » a été considérablement amélioré. Lorsque le nombre d'entrées approche la limite de 95%, Visual FoxPro initie automatiquement un « garbage collector ».

Une autre amélioration majeure est survenue depuis Visual FoxPro 7.0. Toutes les versions précédentes comptabilisaient les objets. Lorsque vous créez deux « label / étiquette » :

```
LOCAL loLabel1, loLabel2
loLabel1 = CreateObject ("Label")
loLabel2 = CreateObject ("Label")
```

Visual FoxPro ajuste automatiquement la propriété « name ». La première étiquette s'appelle "Label1", la deuxième étiquette Label2, etc... Pour ce faire, Visual FoxPro a inscrit chaque objet dans la «table de noms». La conséquence est que au fur et à mesure, chaque création d'objet prend plus de temps que la précédente. Jusqu'à Visual FoxPro 7.0, vous ne pouviez pas créer plus de 65 000 objets (si la variable MVCOUNT = 65000 dans le fichier de configuration CONFIG.FPW, par défaut elle est fixée à 16384), y compris tous les objets figurant dans des formulaires. Une fois cette limite atteinte, vous ne pouvez même pas ouvrir le Concepteur de formulaires, car il tente également de créer des objets.

Depuis la version 7.0 Visual FoxPro vous pouvez créer plus de 65 000 objets car ils ne sont plus enregistrés dans la « table des noms ». L'inconvénient est que les noms d'objets ne sont plus garantis comme étant uniques.

Variables, tableaux et objets

La limite de 65 000 noms d'entrées dans la « table des noms » n'est pas la cause directe qui limite le nombre d'éléments d'un tableau. Un tableau compte comme une entrée unique dans la « table des noms », quel que soit le nombre d'éléments qu'il contient. Par contre, le nombre d'éléments dans un tableau étant également limité à 65 000, vous ne pouvez créer qu'un maximum 65 000 matrices contenant chacune 65 000 éléments. Bien que Visual FoxPro, vous permette de traiter une grande quantité de données de tableaux en mémoire, pour des raisons de performances, aucune application réelle ne devra être proche de ces limites. La raison de toutes ces limitations est la « table des noms », encore une fois. Visual FoxPro ne sait seulement traiter les tableaux que comme de simples variables.

Les variables de FoxPro peuvent stocker des valeurs de tout type de données. Comme Visual FoxPro est écrit en C, les variables FoxPro sont en interne également stockées au format d'une structure C :

```
typedef struct {
  char ev_type;
  char ev_padding;
  ev_width court;
  unsigned ev_length;
  long ev_long;
  double ev_real;
  CCY ev_currency;
  MHandle ev_handle;
  unsigned long ev_object;
} Valeur;
```

Chaque fois qu'un programme crée une nouvelle variable de mémoire, Visual FoxPro alloue un bloc de mémoire avec la structure ci-dessus. Son adresse tout comme son nom doit être stockée quelque part, car il ne font pas partie de la structure C. L'enregistrement de ces informations est effectué dans la « table de noms ». Chaque entrée de la « table de noms » contient, outre le « nom » et la portée, des détails sur le type de l'entrée (variable, champ, alias, etc.) et son emplacement en mémoire. La liste est limitée à 65 000 entrées , d'où le nombre maximal de variables.

Dans la « table de noms », Visual FoxPro crée une entrée unique pour l'ensemble du tableau. L'entrée de tableau dans la « table de noms » actuelle contient un pointeur sur la liste des éléments du tableau. Alors que vous pouvez le faire pour une variable, il n'y a aucun moyen d'accéder directement à un élément de tableau.

Il faut transmettre le tableau par référence (@) à une procédure / méthode / fonction pour pouvoir y manipuler ou y modifier des éléments. Ce passage par référence transmet non pas une copie de la valeur de tous les éléments du tableau, mais simplement le NTI du tableau. Un tableau, ne possédant qu'un seul index de « table de noms », il n'est pas possible de spécifier également dans quel élément le résultat doit être écrit par le programme appelé. Par conséquent, vous ne pouvez transmettre qu'un tableau entier par référence. Dans Visual FoxPro, il est impossible de transmettre un seul élément de tableau par référence. Vous devez copier l'élément dans une variable, passer une référence à cette variable et mettre à jour la valeur dans le tableau.

Idem pour les objets qui sont stockés de la même manière. La valeur dans la structure C est l'adresse d'une autre « table de noms ». Pour chaque objet, Visual FoxPro semble créer une nouvelle « table de noms » complète. Les propriétés dans Visual FoxPro sont en réalité des variables conservées dans un endroit caché. Plus surprenant, cela est aussi vrai pour les méthodes, qui sont aussi des variables. Pour cette raison, vous ne pouvez pas créer une classe comportant plus de 65 000 méthodes, propriétés et événements.

Cette conception présente de nombreux avantages, car sinon, les tableaux et les propriétés satureraient rapidement l’espace disponible de la « table des noms ». L'inconvénient le plus évident, cependant, est le passage de paramètres par référence à l'aide du caractère @.

La même chose s'applique aux propriétés de l'objet. En théorie, vous ne pouvez pas transmettre une propriété par référence, car elle fait partie intégrante de l'objet. Passer une propriété par référence, casserait l’encapsulation. La vraie réponse, cependant, est qu’une propriété n’a pas d’entrée dédiée dans la « table des noms » et qu’il est donc impossible de passer une propriété par référence. Les propriétés des tableaux sont particulièrement impactées par cette conception. Ni les tableaux, ni les propriétés ne peuvent être passés par référence. Vous n'avez donc aucune autre possibilité que de copier le tableau dans une variable locale, de la transmettre à la place et de copier le tableau entier à l'aide de ACOPY () dans les deux sens. Combinée aux méthodes d'affectation et d'accès, cette méthode présente encore plus d'inconvénients que la simple réduction de la vitesse d'exécution. Heureusement, Visual FoxPro 8 a ajouté des collections qui

ressemblent beaucoup à un tableau, mais peuvent facilement être transmises.

L'accès aux variables a été optimisé par Visual FoxPro. Lors de la compilation d'un programme, Visual FoxPro convertit les noms en mots faciles à gérer. Les lignes:

```
LOCAL lcName, lcZIP

? lcName
```

Sont convertis dans le pseudo code suivant lors de la compilation:

```
LOCAL N ° 1, N ° 2

? #1
```

La longueur du nom de la variable n'est pas pertinente. Les noms longs de variables (souvent plus lisibles) n'ont donc aucun effet négatif significatif sur la vitesse d'exécution.

L'accès à un élément de tableau est donc toujours plus lent que l'accès à une variable. Les paramètres d'index sont évalués et transmis à une autre fonction qui copie la structure de valeur pour charger un élément de tableau. Pour compliquer davantage les choses, Visual FoxPro prend en charge les crochets carrés [] et ronds () pour les tableaux et les fonctions. Il n'y a pas de distinction claire entre les deux types de crochets. Étant donné que les tableaux ont la priorité, Visual FoxPro doit rechercher un tableau du même nom à chaque appel de fonction.

Pour les objets c'est encore plus lent car plus difficile d'y accéder. Une fois encore, les noms de propriété sont convertis en nombres. Les objets conservent leur propre « table de noms ». Le NTI de la « table de noms » n'étant pas utilisable pour accéder à un object, Visual FoxPro doit localiser le nom de l'objet dans la « table noms » puis lire la structure C dans la « table privée de l'objet». Cela nécessite de résoudre le nom réel au lieu d'utiliser la valeur d'index et les résultats ne peuvent pas être mis en cache comme pour les valeurs simples.

Par conséquent, accéder à une propriété d'objet est encore plus lent que d'accéder à une variable ou à un élément de tableau.

Cela explique pourquoi WITH .. ENDWITH peut améliorer considérablement les performances. Il conserve simplement la structure des valeurs en mémoire et rend inutile la résolution de la première partie. Pour une utilisation la plus rapide d'une donnée, il est préférable de la conserver dans une variable locale, plutôt que dans une propriété d'objet. Une telle variable peut être résolue beaucoup plus rapidement qu'une propriété d'objet.

Gestion de la mémoire

La gestion de la mémoire dans Visual FoxPro n'est pas moins complexe. Elle combine toutes les gestions mémoires de l'époque des versions de Foxpro fonctionnant sous DOS ou en 16 bits avec une utilisation des accès de la mémoire en 32 bits, raison pour laquelle la plupart des fonctions de mémoire non documentées entre SYS (1001) et SYS (1016) sont toujours opérationnelles.

En plus de la mémoire physique et de divers modèles de mémoire implémentés par le système d'exploitation, Visual FoxPro dispose de sa propre gestion de mémoire.

Il y a beaucoup de données à garder en mémoire. Cela inclut les variables, les menus, les fenêtres, les chaînes, mais aussi les objets, les définitions de classe, les formulaires, etc. Chaque fois que Visual FoxPro requiert de la mémoire, le gestionnaire de mémoire est appelé avec la taille du bloc de mémoire souhaité. Au lieu de renvoyer une adresse, il renvoie un identificateur de mémoire. C'est comme une sorte de clé primaire. Lorsqu'un programme souhaite accéder à cette mémoire, il doit d'abord la verrouiller, un peu comme vous verrouillez un enregistrement. Après cela, il peut déterminer l'adresse à l'aide d'une fonction interne. Une fois l'accès à la mémoire effectué, le code doit déverrouiller le pointeur, comme pour déverrouiller un enregistrement d'une table.

Le but de ces verrous est différent de celui des enregistrements. Ce qui doit être évité, n'est pas d'accéder à la mémoire en parallèle, mais d'éviter de déplacer le bloc mémoire. Lorsque des blocs de mémoire sont constamment alloués et que la mémoire est libérée, elle se fragmente au fil du temps. La même chose s'applique aux disques durs lorsque vous créez et supprimez des

fichiers. Après une longue période d'exécution, il est possible que la mémoire disponible soit suffisante, mais qu'aucun bloc ne soit assez grand pour contenir les données demandées. C'est vital pour un système de base de données qui doit fonctionner sans surveillance pendant des jours ou des semaines, et une telle situation de fragmentation mémoire plante le système.

Pour éviter cela, le « garbage collector » récupère de la place en déplaçant et recombinant des blocs de mémoire. En interne, Visual FoxPro effectue une sorte de défragmentation lorsqu'il est en attente d'activité.

Une partie du pool de ressources est visible pour nous, développeurs, à l'aide de la commande LIST MEMORY, mais pas tous les « handles ». En interne, Visual FoxPro utilise également beaucoup de « handles ». Toutes les fenêtres d'éditeur, par exemple, sont également des « handles ». Par conséquent, il est possible d'utiliser ACTIVATE WINDOW non seulement pour activer nos propres fenêtres, mais également pour toutes les fenêtres fournies par FoxPro, y compris les barres d'outils. Dans Visual FoxPro, le pool de descripteurs n'est limité que par la mémoire physique et virtuelle disponible.

La fonction SYS (1001) renvoie la taille maximale du pool de ressources.

La fonction non documentée SYS (1011) renvoie le nombre de descripteurs attribués.

Cette fonction doit renvoyer la même valeur avant et après un appel de fonction, sinon cela indique une fuite de mémoire.

SYS (1016) renvoie la taille de la partie allouée du pool de ressources.

SYS (1104) lance manuellement un « garbage collector ».

Fondamentalement, tout ce qui n'est pas temporaire, est stocké quelque part dans le pool de ressources. Cela inclut les transactions et les modifications non validées dans une mémoire tampon qui doivent être réécrites avec TABLEUPDATE ().

SYS (3050) gère la zone mémoire dans laquelle Visual FoxPro stocke des bitmaps créés par Rushmore et les utilise pour mettre en cache des tables et des index. Cette partie de la mémoire est la principale responsable de la vitesse d'exécution de Visual FoxPro. Tout ce qui est lu à partir du disque est mis en cache ici pour une réutilisation ultérieure.

Il faut veiller à ce que SYS(3050) soit le plus grand possible mais sans que Windows n'alloue à Visual Foxpro de mémoire virtuelle créée physiquement sur le disque dur.

De plus Visual Foxpro, s'il juge insuffisant la quantité de mémoire allouée par la commande SYS(3050), n'hésite pas à créer des fichiers temporaires pour ses besoins de mémoire.

Le répertoire utilisé pour les fichiers temporaires dépend du paramétrage de TMPFILES = dans le fichier CONFIG.FPW.

SYS (2023) permet de trouver le répertoire temporaire utilisé par Visual Foxpro. Vous devez éviter d'utiliser le répertoire ou est stocké la base de données de l'application car elle se trouve souvent sur une autre machine. Son accès par le réseau Ethernet étant alors source de fort ralentissement.

Pour vraiment savoir où vont les fichiers temporaires, vous pouvez créer un curseur et déterminer sa position avec la fonction DBF (). Même si un curseur est généralement créé dans le même répertoire que celui indiqué par SYS (2023), ce n'est pas toujours le cas.

Rushmore

Rushmore est plus qu'une technologie, c'est presque un mythe. La raison de la performance de Visual FoxPro repose sur deux fondements. L'un des fondements est Rushmore, l'autre est l'utilisation véritablement agressive du cache. La différence de performances par rapport aux autres bases de données utilisant un algorithme de recherche centré sur bitmap (comme Rushmore en est un) n'est généralement pas causée par Rushmore, mais par sa stratégie de mise en cache et son accès réseau optimisé.

Rushmore est un algorithme orienté bitmap. Visual FoxPro crée une liste pour chaque condition pouvant être optimisée avec Rushmore. Cette liste détermine si un enregistrement remplit les

critères de recherche ou non. Visual FoxPro utilise un seul bit pour chaque enregistrement, car il n'y a que deux états possibles pour chaque enregistrement. Huit enregistrements entrent dans un octet. Si vous avez une table de 8 millions d'enregistrements, chaque critère de recherche occupe 1 Mo de mémoire.

Une fois que les bitmaps ont été déterminés pour toutes les conditions, ces bitmaps sont combinés au niveau du bit. Si vous utilisez la condition suivante dans une requête:

```
NAME = "Smith" AND InvoiceDate <DATE () - 60
```

Cette requête entraîne la création de deux bitmaps indépendants contenant tous les enregistrements de "Smith" et tous les enregistrements dont la date de facturation est antérieure à 60 jours. Après cela, les deux bitmaps sont combinées bit à bit avec un « AND ». Le résultat est un bitmap qui n'a qu'un bit défini pour les enregistrements qui remplissent les deux conditions. Il n'est pas difficile d'imaginer que l'utilisation de la mémoire pour créer ces bitmaps peut rapidement augmenter et ralentir une requête. Si vous n'avez qu'une seule condition, vous pouvez utiliser l'ancien style xBase:

```
SET ORDER TO RequiredIndex

Valeur de recherche

SCAN WHILE Champ = Valeurs

  * faire quelque chose

ENDSCAN
```

Dans de nombreux cas, cela est plus rapide qu'une requête optimisée par Rushmore, car Visual FoxPro n'a pas à créer de bitmap.

Comment cette bitmap est-elle réellement créée? Le facteur le plus important est que l'index dans Visual FoxPro est stocké en tant qu'arborescence binaire, une arborescence équilibrée. Chaque nœud est un bloc de 512 octets pouvant contenir jusqu'à 100 pointeurs sur des nœuds subordonnés. Les nœuds proches de la racine font référence aux valeurs de clé, seuls les nœuds terminaux font référence à des enregistrements. Comme chaque nœud ne se réfère pas uniquement à deux autres blocs, comme on aurait pu le penser, la hiérarchie du fichier d'index peut généralement rester très plate. De plus, tous les nœuds sont liés horizontalement.

Lorsque Visual FoxPro recherche une valeur, il parcourt l'arborescence verticalement de haut en bas jusqu'à trouver l'enregistrement. Il continue ensuite à lire horizontalement tant que la valeur de recherche correspond à celle de l'index. Cette stratégie de lecture descendante et latérale réduit le nombre de nœuds d'index à lire, mais augmente le nombre de nœuds à modifier lors de la mise à jour de l'index. Lors de la lecture horizontale, Visual FoxPro définit le bit correspondant à chaque enregistrement trouvé. Cela est possible car l'entrée d'index contient le numéro d'enregistrement ainsi que la valeur de clé. Les données sont cependant stockées compressées.

Avec Rushmore, Visual FoxPro peut déterminer quels enregistrements ne sont pas dans le jeu de résultats, pas quels enregistrements sont dans le jeu de résultats. C'est pourquoi la phase Rushmore est suivie d'une phase post-scan. Chaque enregistrement déterminé par Rushmore est lu intégralement à partir du disque. Sans expression optimisée, ce sont tous les enregistrements de toutes les tables. Chacun de ces enregistrements est ensuite vérifié pour les expressions non optimisées restantes. En outre, toutes les expressions optimisables sont à nouveau évaluées, car une autre personne pourrait avoir modifié l'enregistrement pendant que Visual FoxPro a créé le bitmaps. Par conséquent, un jeu de résultats peut ne pas contenir tous les enregistrements correspondant actuellement aux critères de recherche, mais vous n'obtenez jamais d'enregistrements ne correspondant pas aux critères de recherche.

Il y a une exception à cette règle, difficile. Lors du comptage des enregistrements, Visual FoxPro tente d'éviter la phase post-analyse. Lorsque le bitmap a été créé et qu'il ne reste aucune expression non optimisable, Visual FoxPro commence à compter les bits définis.

Par conséquent, COUNT FOR et SELECT CNT (*) FROM sont extrêmement rapides si une requête est entièrement optimisée. Une optimisation partielle n'est pas suffisante.

Lors de la création de l'image bitmap, Visual FoxPro doit lire tous les blocs d'index correspondant aux critères de recherche. L'accès répété oblige Visual FoxPro à extraire ces blocs du cache.

Toutefois, le cache est supprimé si une station modifie un champ indexé ou ajoute un nouvel enregistrement. Visual FoxPro peut uniquement déterminer qu'un autre poste de travail a modifié l'index, mais pas ce qui a exactement changé. Par conséquent, le cache entier est supprimé.

Vous devez donc garder à l'esprit que le cache a un impact sur Rushmore et que les index CDX y sont stockés compressés.

Si une application modifie de nombreuses données, cela invalide plus fréquemment le cache. Par conséquent, une telle application bénéficie beaucoup moins de Rushmore que, par exemple, une application de catalogue de CD qui a mis toutes les données dans le cache après une courte période d'utilisation.

Si votre application comporte de nombreux enregistrements supprimés, elle bénéficie également de Rushmore. Si votre application accède plusieurs fois à un enregistrement parce que, par exemple, l'algorithme nécessite de naviguer vers les enregistrements suivants et précédents, Rushmore est supérieur car il peut réutiliser les bitmaps alors que le code xBase de style ancien doit relire tous les enregistrements.

Si vous souhaitez déterminer le nombre d'occurrences avant d'exécuter la requête afin de permettre à vos utilisateurs d'éviter des requêtes de longue durée, votre requête doit être entièrement optimisable. Cela vous donne un réel avantage en termes de rapidité, car la création d'un bitmap prend beaucoup moins de temps que la lecture de la table entière. Le bitmap peut même être réutilisé si l'utilisateur décidait d'exécuter la requête.

Un autre facteur important est la distribution des valeurs dans le champ indexé. Il y a une énorme différence si vous recherchez une valeur unique, telle qu'une clé primaire ou dans un champ d'état qui ne peut prendre que dix valeurs différentes. Un exemple extrême est l'index sur DELETED () pour lequel tous les champs ont généralement la même valeur.

En règle générale, évitez d'indexer un champ contenant de nombreuses valeurs identiques.