Indexeur de mots dans des pages statiques

Par
Pascal PRADIER, Responsable Programme 4D S.A.
Note technique 4D-200009-24-FR
Version 1

Date 1 Septembre 2000

Résumé

Cette NT montre comment indexer très facilement et très efficacement les mots d'une ou plusieurs pages statiques en utilisant les blobs.

4D Notes techniques

Introduction

Cette note technique décrit une méthode permettant de stocker un grand nombre de données (des mots) contenus dans des pages statiques HTML et de retrouver toutes les occurrences de ces mots instantanément quelque soit le nombre de pages dans lesquelles ils sont présents.

Il sera ainsi possible d'indexer tous les mots d'un site Web afin d'offrir à l'internaute la possibilité de faire une recherche sur un mot et d'obtenir un résultat où que soit contenu ce mot sur le site.

Note : cette base a été rélisée en version 6.7 de 4ème Dimension.

Cahier des charges

Pouvoir rechercher un mot à l'intérieur d'un site web quelque soit sa taille.
Obtenir des temps de réponses identiques quelque soit la taille du site.
Etre indépendant de la langue.
Pouvoir choisir la taille des mots à indexer.
Permettre l'indexation des nouvelles pages ou des pages modifiées sans avoir à tout réindexer.
Renvoyer le résultat sous forme d'une page HTML contenant autant de liens pointant sur les pages trouvées.
Un code 4D modulaire et préfixé afin d'être facilement intégrable dans une base existante sous forme de Librairie ou de Composant.

Que va-t-on indexer

A priori, il est possible d'indexer tous les mots quels qu'ils soient, mais cela n'est pas souhaitable pour deux raisons :

1) si les temps de recherche sont quasiment identique quel que soit le nombre de mots, les temps d'indexation sont revanche dépendant du nombre de mots à indexer.

2) Le but final de l'indexation étant de retrouver une page contenant tel ou tel mot, est-il souhaitable de retrouver les pages qui contiennent "le", "la", "ce" etc.

Le choix est donc d'indexer par défaut les mots de plus de deux lettres mais ce choix est paramétrable en Préférences.

Le principe

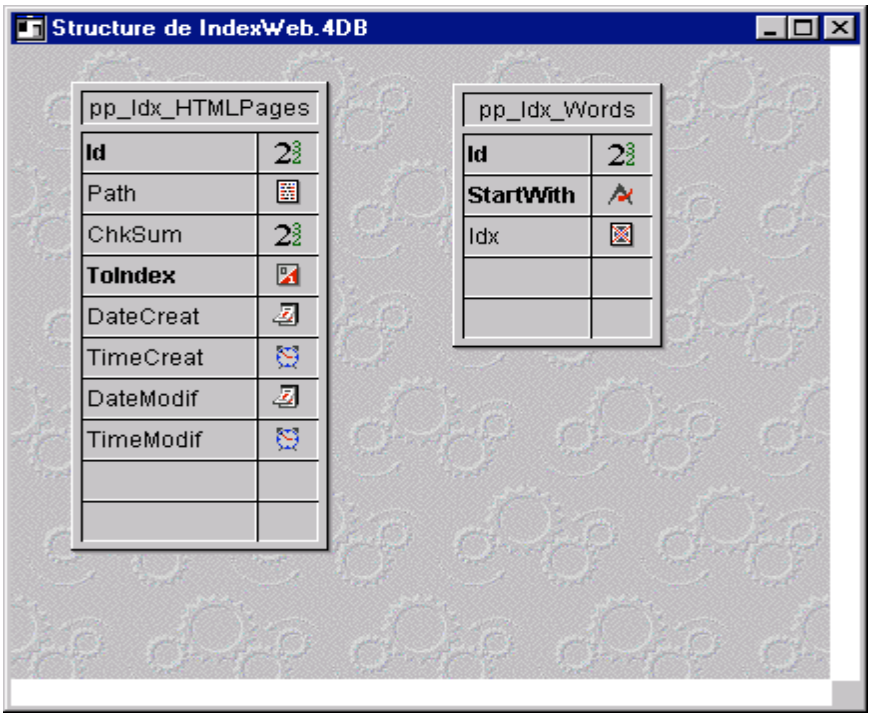
Tout le cœur du système réside dans la façon de stocker un grand nombre de données de petites tailles. Deux possibilités s'offrent à nous : la première qui est la plus évidente mais pas forcément la plus judicieuse consiste à créer autant d'enregistrements 4D qu'il existe de mots à indexer.

L'inconvénient de ce système est que le volume risque d'être énorme et que les temps de réponse seront sans doute médiocres en raison des accès disques répétés. Par ailleurs, ce système deviendrait difficilement intégrable à une base existante dans la mesure ou le fichier de donnée grossirait énormément et qu'une réparation éventuelle deviendrait d'une lourdeur épouvantable.

La deuxième solution consiste à stocker ces informations dans des blobs. L'avantage est que la recherche dans un blob s'effectue exclusivement en mémoire vive ce qui nous garantie des temps de réponses en mode compilé quasiment identiques quelque soit le volume. Cette méthode est très peu gourmande en données puisque tous les mots susceptibles d'être indexés tiennent dans 676 enregistrements.

La structure

Il est difficile de faire plus simple :



Elle se compose de deux tables.
La table [pp_Idx_HTMLPages] contiendra toutes les pages du site Web à indexer. Les différents champs de cette table sont les suivants :

Id	->	L'identificateur de l'enregistrement 4D. C'est un entier long indexé.
Path	->	Contient le chemin de la page Web indexée.
ChkSum	->	Inutilisé. Permet un éventuel stockage du checksum de la page.

ToIndex	->	booléen permettant de savoir quelles sont les pages nécessitant une indexation.
DateCreat	->	Date de création de la page.
TimeCreat	->	Heure de création de la page.
DateModif	->	Date de modification de la page.
HeureModif	->	Heure de modification de la page.

La table [pp_Idx_Words] contiendra les 676 enregistrements référençant tous les mots indexés. Cette table est composée des champs suivants :

Id	->	L'identificateur de l'enregistrement 4D. C'est un entier long indexé.
StartWith	->	Contient les deux premières lettres des mots. Alpha 2 indexé.
Idx	->	Blob contenant les mots.

Le nombre d'enregistrements de cette table est fixe et limité à 676. La raison est que nous créons une fiche pour les mots commençant par AA, une pour les mots commençants par AB … … ZZ. Bien entendu, les blobs de certaines de ces fiches resterons vides car les mots commençants par ZZ ne sont pas légions mais cela ne gene en rien sur les temps de recherche.

Préparation du fichier de données

La première phase de l'indexation est la préparation du fichier de données. Cette préparation consiste à supprimer tous les éventuels enregistrements résiduels contenus dans les deux tables puis à créer les 676 enregistrements de la table [pp-Idx_Words].

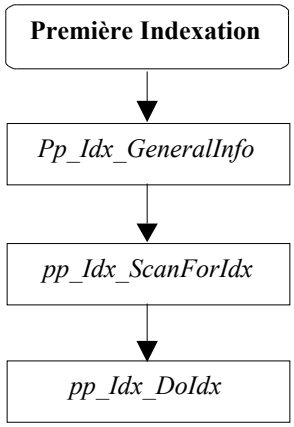
Pas de finesse particulière à ce niveau. Voici le détail du code :

```
TOUT SELECTIONNER([pp_Idx_HTMLPages])

SUPPRIMER SELECTION([pp_Idx_HTMLPages])
TOUT SELECTIONNER([pp_Idx_Words])
SUPPRIMER SELECTION([pp_Idx_Words])

pp_Idx_SetRecWords


Méthode pp_Idx_SetRecWords

C_ENTIER LONG($k;$i;$j)
$k:=0
Boucle ($i;1;26)
 Boucle ($j;1;26)
  $k:=$k+1
  CREER ENREGISTREMENT([pp_Idx_Words])
  [pp_Idx_Words]Id:=$k
  [pp_Idx_Words]StartWith:=Caractere(64+$i)+Caractere(64+$j)
  STOCKER ENREGISTREMENT([pp_Idx_Words])
 Fin de boucle
```

```
Fin de boucle
```

À partir de là, l'indexation proprement dite peut commencer.

L'indexation



pp_Idx_GeneralInfo affiche un dialogue de progression de la tache en cours.
pp_Idx_ScanForIdx recherche quelles sont les pages nécessitant une indexation.
pp_Idx_DoIdx effectue l'indexation proprement dite.

Voici le détail de ces méthodes :

La méthode Pp_Idx_GeneralInfo permet l'affichage dans un nouveau process d'une palette flottante permettant de suivre la progression de l'indexation.
En voici le détail :

```
C_ALPHA(80;$1)
C_ENTIER LONG($Fen)

Si (Nombre de parametres=0)
  <>PcsInfo:=Nouveau process("pp_Idx_GeneralInfo";32000;"Info";"bidon";*)
Sinon
 CHANGER BARRE(1)
  $Fen:=Creer fenetre formulaire([pp_Idx_HTMLPages];"Progress";Fenêtre palette ;A droite ;En haut )
 DIALOGUE([pp_Idx_HTMLPages];"Progress")
Fin de si
```

La méthode pp_Idx_ScanForIdx est juste une méthode appelant une méthode récursive (*pp_Idx_ScanChangedPages*) qui recherche les pages qui devront être ré-indexées. Lors d'une première indexation, toutes ces pages seront ré-indexées.

Nous nous attarderons donc sur la méthode *pp_Idx_ScanChangedPages* et non sur *pp_Idx_ScanForIdx*.

pp_Idx_ScanForIdxest une méthode récursive (qui s'appelle elle même) recherchant les pages HTML dans un dossier donné. La récursivité était préférable dans le cas ou d'autres dossiers seraient présents à l'intérieur.

La recherche d'une page HTML dans un dossier nécessite d'ouvrir chaque document et vérifier la présence des marqueurs <HTML>… …</HTML>. En effet, l'extension .HTM ou .HTML n'est pas suffisante pour déterminer qu'une page est une page HTML ou non. Il suffira donc de tester la position de ces marqueurs :

```
Si ((Position("<HTML";$myText)#0) & (Position("</HTML";$myText)#0))
```

Dans le cas ou ce test est vrai, il nous faudra récupérer les informations relatives aux dates et heures de création et modification de document grâce à la ligne suivante :

```
PROPRIETES DOCUMENT($1+$tDoc{$i}; $Lock; $Visible; $DateCreat; $TimeCreat; $DateModif; $TimeModif)
```

Après quoi nous enregistrerons ces informations dans une fiche de la table [pp_Idx_HTMLPages].
Deux cas peuvent alors se présenter : c'est une première indexation ce qui nous obligera à créer un enregistrement ou bien c'est une ré-indexation et dans ce cas il s'agira d'une mise à jour de la fiche uniquement si la date de modification de la fiche est différent de la date de modification de la page HTML.

Lors de cette opération, le champ *ToIndex* est positionné à **Vrai**.

La méthode pp_Idx_DoIdx permet l'indexation proprement dite.
Nous commençons par chercher tous les enregistrements de la table pp_Idx_ScanForIdx pour lesquels le champ ToIndex est Vrai. Nous bouclons sur chacun de ces enregistrements correspondants à une page HTML à indexer que nous ouvrons afin d'en récupérer le contenu dans une variable de type blob.

Ce contenu sera nettoyé de tous ses marqueurs HTML, ponctuation et chiffres de façon à n'indexer que les mots significatifs. C'est le rôle des trois méthodes pp_Idx_CleanHTML, pp_Idx_ClearPonctu et pp_Idx_ClearNumber.

Ensuite, il nous reste à récupérer dans un tableau tous les mots de la page grâce à la méthode pp_Idx_GetWord. Cette méthode récupère uniquement les mots supérieurs à <>pp_Idx_LongWord caractères (par défaut égal à 2). Bien entendu, seule une occurrence d'un même mot par page sera retenue.

La méthode pp_Idx_IndexPage qui reçoit en paramètre le tableau précédemment créé ainsi que l'Id de l'enregistrement référençant la page demande à la méthode pp_Idx_IndexWord l'indexation de chacun des mots contenus dans le tableau.

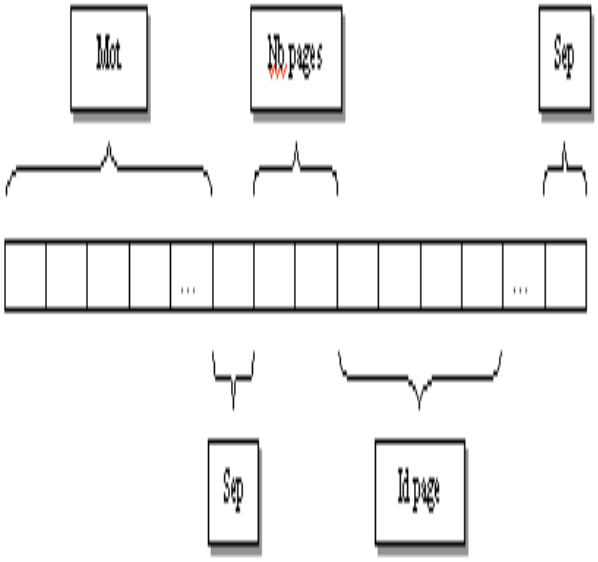
Le codage

Comme nous l'avons vu précédemment, les mots sont stockés dans un blob selon l'organisation suivante :

n octets pour le mot
1 octet pour un caractère séparateur
2 octets pour le nombre de pages dans lequel le mot est présent
4 octets pour l'ID de la page où le mot est présent (répété autant de fois que précisé dans les deux octets précédents)
1 octet séparateur

Le choix du caractère séparateur s'est porté sur le caractère 'Bell' qui est vraiment peu probable dans un mot…
Le nombre de page et les Id de page seront codés en format PC.
Les mots seront systématiquement en majuscule.

Voici à quoi cela ressemble…



Prenons l'exemple du mot "TOTO" qui est contenu dans la page d'Id 17234 uniquement, le blob sera codé de la manière suivante :

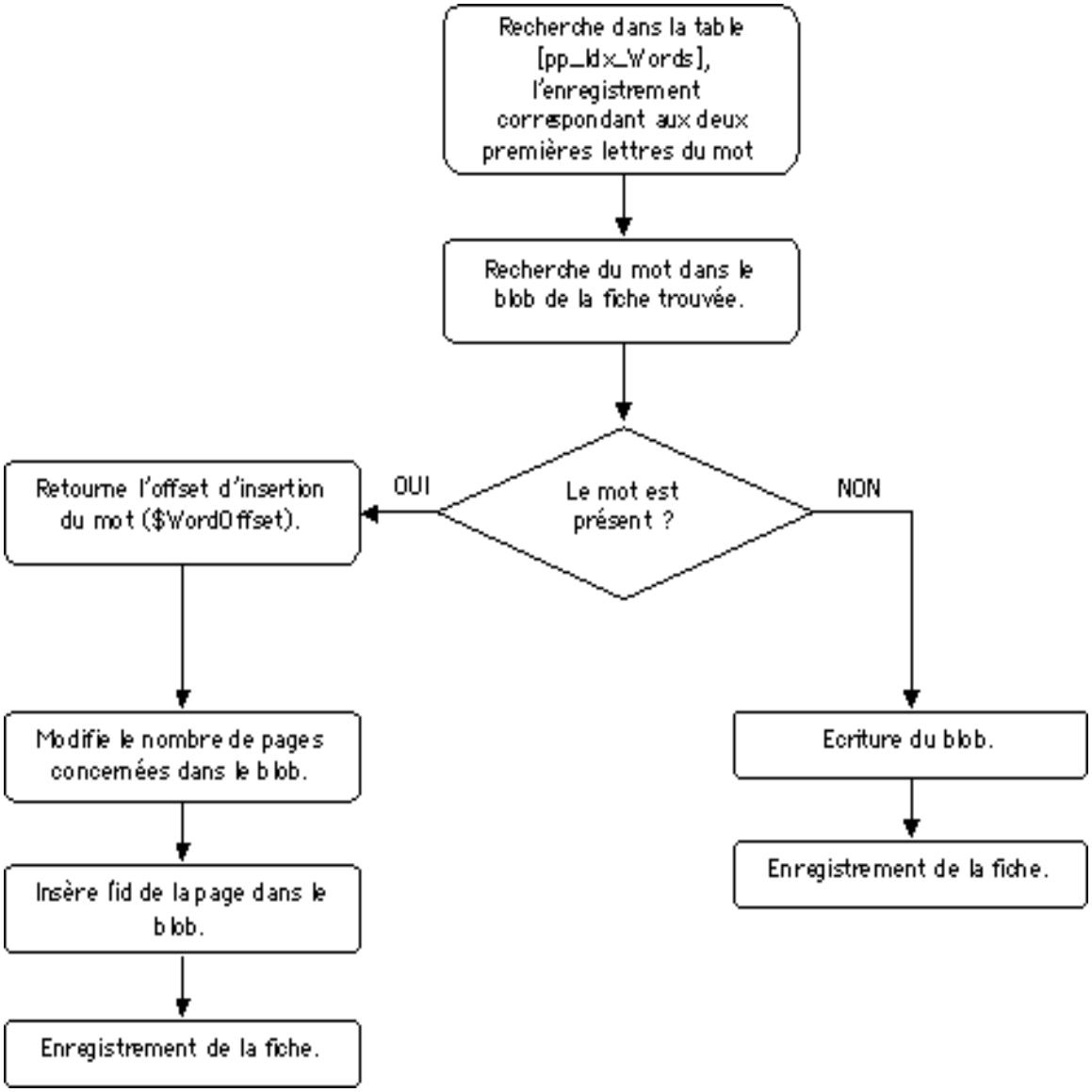
84	79	84	79	7	1	0	174	188	255	255	7

A présent regardons le mot 'CAP' contenu dans les pages 876 et 23

67	65	80	7	2	0	148	252	255	255	233	255	255	255	7

Maintenant que le principe du codage d'un mot est expliqué, nous pouvons décrire le fonctionnement de la méthode pp_Idx_IndexWord.

Cette méthode reçoit en paramètre le mot à indexer ainsi que l'Id de la page Web le contenant. Voici l'organigramme de cette méthode :



Le principe est simple, on cherche l'enregistrement 4D correspondant aux deux premières lettres du mot à indexer puis, grâce à la méthode pp_Idx_SrchWordInBlob, on cherche dans le blob de la fiche trouvée si ce mot est présent.

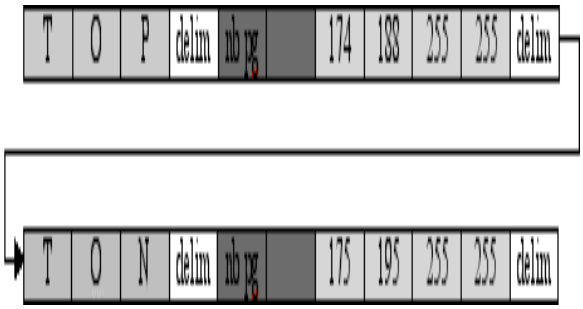
Si le mot n'est pas présent, la méthode pp_Idx_SrchWordInBlob retourne ?1. Dans ce cas, le mot est ajouté dans le blob selon le codage décrit plus haut.

Si le mot est présent, la méthode pp_Idx_SrchWordInBlob retourne l'offset de position du mot. Il reste à modifier le nombre de pages concernées par ce mot et insérer l'Id passé en paramètre dans le blob.

Voyons à présent la méthode pp_Idx_SrchWordInBlob qui est le moteur de recherche d'un mot dans le blob.

Cette méthode reçoit en paramètre le blob dans lequel effectuer la recherche, le mot à rechercher ainsi que le caractère délimiteur.

Considérons le blob suivant :



Et le mot à rechercher : 'TON'

Nous commençons par comparer chacune des lettres du mot à chercher avec les lettres du premier mot du blob (dans notre cas, 'TOP'). C'est le rôle de la ligne :

```
Tant que ((Code ascii($ToSearch[[$j]])=$1{$k+$j-1}) & ($1{$k+$j-1}#$Delim))
```

Dès qu'une différence est trouvée ou bien que nous avons atteint le délimiteur de mot, nous sortons du 'tant que'.
Nous testons alors si le mot trouvé a bien la même taille que le mot recherché :

```
Si ((($j#$Nb) | ($1{$k+$j-1}#$Delim))
```

Si ce n'est pas le cas, nous avançons jusqu'au délimiteur de mot :

```
Tant que ($1{$k}#$Delim) `aller au délimiteur
    $k:=$k+1
Fin tant que
```

Afin de gagner du temps, nous lisons le nombre de pages pour le mot sur lequel nous sommes afin de les sauter et passer directement au mot suivant :

```
$NbPages:=BLOB vers entier($1;Ordre octets PC ;$offset)
```

```
$offset:=$offset+($NbPages*4)+1   `(nbpages * 4 octets) + 1 délimiteur
```

à ce niveau, la variable $offset pointe sur le mot 'TON' c'est à dire sur le douzième octet du blob.

Un test supplémentaire est effectué afin de vérifier le cas ou le mot n'est pas trouvé dans le blob (mot qui n'est pas encore indexé au moment de la recherche) :

```
S i ($offset>=$SizeBlob)   `le mot n'est pas encore indexé
      $i:=$SizeBlob+2   `pour sortir de la boucle
      $offset:=-1
Fin  de  si
```

Nous ne sommes pas dans ce cas et nous pointons sur le mot que nous recherchons. Il nous reste à mettre à jour la variable $offset :

```
$offset:=$k+$j-1
```

et la retourner à la méthode appelante.

Optimisations

Il est possible d'effectuer certaines optimisations dans le codage du blob.

Comme chaque mot contenu dans un même blob commence toujours par les deux mêmes lettres, on pourrait envisager de ne pas écrire ces lettres dans le blob. Ceci ferait gagner deux octets par mot et peut être un peu de vitesse lors des recherches.

Le délimiteur de fin de ligne n'est pas nécessaire. Nous avons deux octets nous donnant le nombre de pages pour mot, il suffit de multiplier ce nombre par 4 octets (un Id de page est codé sur 4 octets) et nous connaîtrons l'offset de fin de ligne. Nous gagnerions encore un octet par mot mais sans doute rien du tout en vitesse.

Recherche d'un mot

La recherche d'un mot passe par la méthode pp_Idx_Search. Cette méthode accepte en paramètre le mot à rechercher (ce qui est la moindre des choses), un pointeur sur un tableau entier long qui contiendra les id des enregistrements où sont stockés les chemins d'accès aux pages HTML et en troisième paramètre le délimiteur. Cette fonction retournera 0 si le mot n'est pas trouvé dans l'index ou bien une valeur positive indiquant le nombre pages dans lesquelles le mot est référencé.

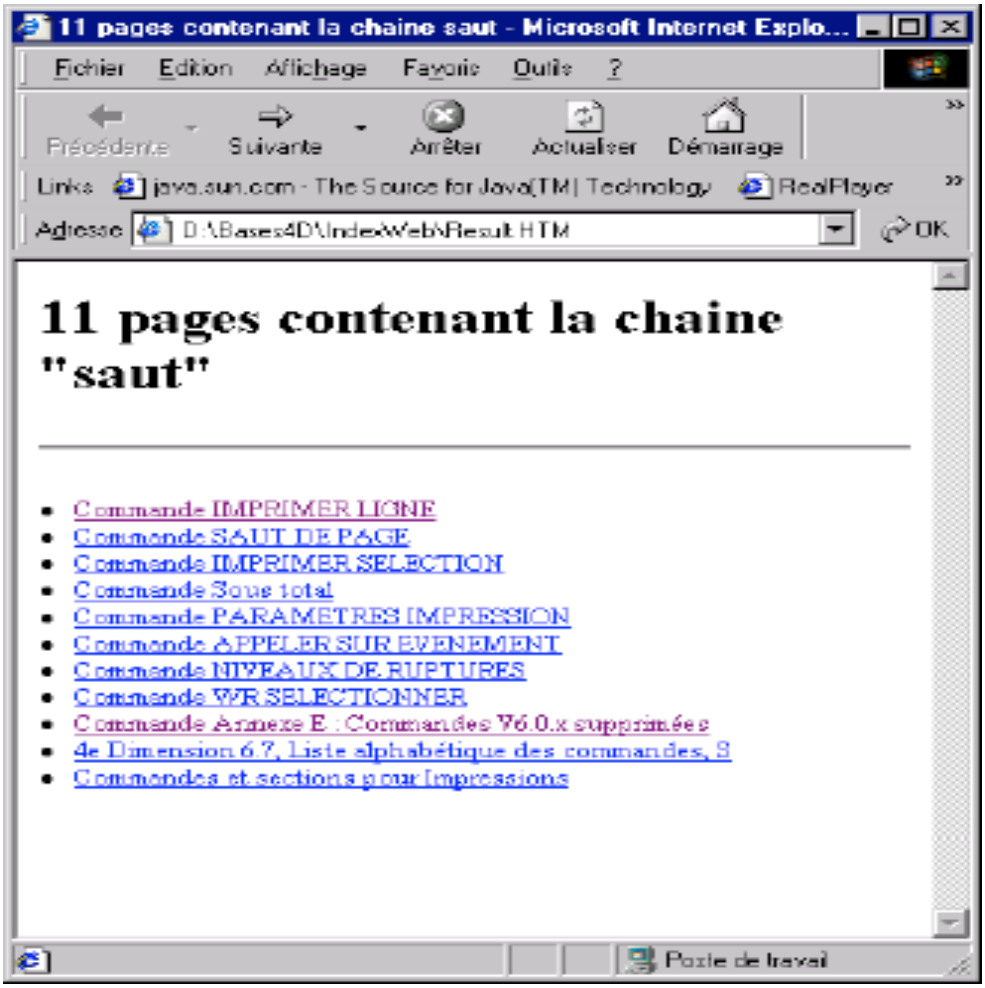
La méthode pp_Idx_Search est très proche de la méthode pp_Idx_SrchWordInBlob mais possède une petite différence :

Une fois le mot trouvé dans le blob, il reste à remplir le tableau avec les id trouvées.

```
INSERER  LIGNES($2->;Taille  tableau($2->)+1;$NbPages)
Boucle ($L;1;$NbPages)
  $2->{$L}:=BLOB  vers  entier  long($vBlob;Ordre octets PC ;$offset)*-1
Fin  de  boucle
```

C'est maintenant au tour de la méthode pp_Idx_CreateHTMLResult de prendre le relais pour l'affichage des résultats.
Cette méthode reçoit en paramètres un pointeur sur le tableau contenant les id des enregistrements 4D contenant les chemins des pages HTML.

Cette méthode utilise la très puissante et très rapide commande Chercher par tableau.
Une fois la sélection effectuée, nous préparons la page Web qui contiendra les résultats des recherches tels que présentés dans la copie d'écran suivante :



Pour obtenir ce résultat, nous sommes obligés d'ouvrir chacune des pages référencées afin de lire l'en-tête pour en extraire le titre. Ce titre deviendra un lien permettant d'appeler la page concernée. C'est le rôle des lignes suivantes :

```
$NewPage:=$NewPage+"<li>"+"<A HREF="

$newPath:=Remplacer chaine([pp_Idx_HTMLPages]Path;"\";"/")
$refDoc:=Ouvrir document([pp_Idx_HTMLPages]Path)
Si (ok=1)
 RECEVOIR PAQUET($refDoc;$txt;32000)
 FERMER DOCUMENT($refDoc)
  $txt:=pp_Idx_GetHTMLTitle ($txt)
Sinon
  $txt:=[pp_Idx_HTMLPages]Path
Fin de si

$NewPage:=$NewPage+Caractere(34)+$newPath+Caractere(34)+">"+$txt+"</A>"+$CR
```

Dans le cas ou un titre n'aurait pas été trouvé, nous donnerons le chemin d'accès comme lien.

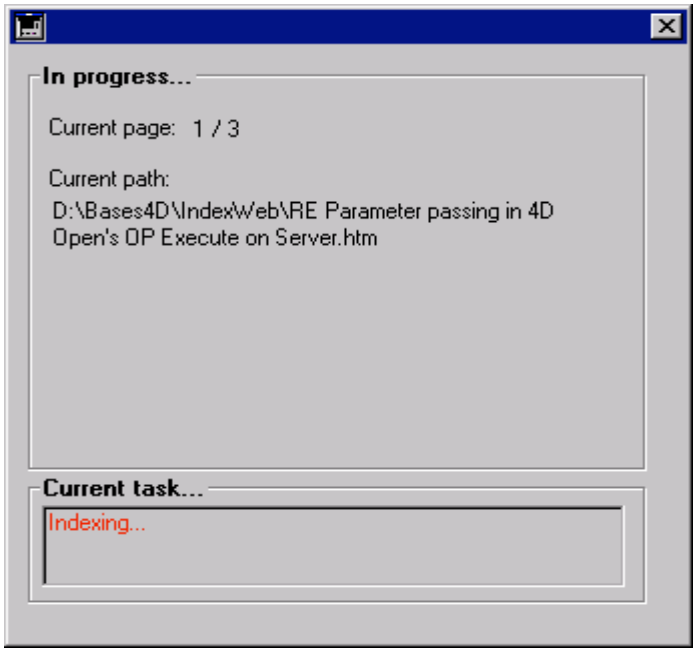
Utilisation

Cette base est très simple à utiliser. Elle dispose d'un menu Index contenant 3 lignes : Search, Index et Clean.

Clean permet de préparer la base de donnée pour l'indexation. C'est-à-dire que nous supprimons toutes les éventuelles données présentes dans les deux tables et que nous créons les 676 enregistrements qui contiendront les futurs index.

Cette ligne de menu est en théorie à utiliser une fois lors de la première utilisation de la base.

Index permet l'indexation ou la ré-indexation du site. Un dialogue de progression s'affiche indiquant quel dossier est en cours d'indexation :



Search permet la recherche d'un mot dans l'index.

Note : cette base a été rélisée en version 6.7 de 4ème Dimension.

Conclusion

Nous avons effectué le test d'indexer la documentation HTML de 4D avec la base compilée (environ 4000 pages).

Les résultats sont les suivants :

- Temps d'indexation inférieur à deux heures
- Temps de recherche quelque soit le nombre d'occurrences : moins d'une seconde.

La machine était un Dell Latitude Cpu cadencée à 333Mhz