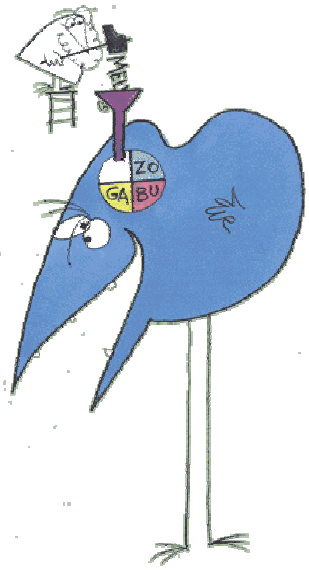
# PROJET
# BASE DE DONNÉES DÉDUCTIVE




# MOTEUR DE RECHERCHE POUR INTERNET

# Table des matières

**INTRODUCTION ..... 1**

**CAHIER DES CHARGES ..... 2**

**I. Définition du cadre du projet ..... 2**
I.1. Qu'est-ce qu'un moteur de recherche sur Internet ..... 2
I.1.1. Point de vue utilisateur ..... 2
I.1.2. Point de vue concepteur ..... 2
I.2. Qu'est-ce qu'Internet ..... 3
I.2.1. Sa structure physique ..... 3
I.2.2. Une modélisation possible ..... 4

**II. Les objectifs du projet ..... 5**
II.1. Les difficultés d'implémenter un moteur de recherche ..... 5
II.1.1. Les contraintes ..... 5
II.1.2. Quelques problèmes et solutions techniques ..... 5
II.1.2.1. Le parcours d'Internet ..... 6
II.1.2.2. Mémoriser les documents trouvés ..... 6
II.1.2.3. Interface utilisateurs ..... 6
II.2. Simplification des objectifs ..... 6
II.2.1. Nouveaux objectifs ..... 6
II.2.2. Nouvelles contraintes ..... 7

**BASE DE DONNEES RELATIONNELLE ..... 8**

**I. Le modèle entités-relations ..... 8**
I.1. Objectifs : Modéliser le fonctionnement d'Internet ..... 8
I.1.1. Avantages ..... 8
I.1.2. Inconvénients ..... 8
I.2. Schéma ..... 9
I.2.1. Conventions ..... 9
I.2.2. Schéma ..... 9
I.2.2.1. Les éléments concernant la structure physique d'Internet ..... 9
I.2.2.2. Les éléments concernant la partie visible d'Internet / Modélisation par graphe ..... 11
I.2.2.3. Résumé ..... 12
I.3. Contraintes ..... 13
I.3.1. Cardinalités ..... 13
I.3.2. Attributs ..... 13

**II. L'implémentation ..... 14**
II.1. Présentation globale ..... 14
II.1.1. La base de données ..... 14
II.1.2. Les données ..... 15
II.2. Les tables en détail ..... 15
II.2.1. Tables concernant les fichiers hébergés par un serveur ..... 15
II.2.1.1. Les serveurs ..... 15
II.2.1.2. Les images ..... 16
II.2.1.3. Les pages web ..... 16
II.2.1.4. Les autres fichiers ..... 17
II.2.1.4.1. Les fichiers divers ..... 17

II.2.1.4.2. Les documents textuels ..... 18
II.2.1.4.3. Les protocoles ..... 19
II.2.1.4.4. L'hébergement ..... 19
II.2.1.5. Le contenu textuel ..... 20
II.2.1.5.1. Les mots ..... 20
II.2.1.5.2. Ce que contiennent les pages web ..... 21
II.2.1.5.3. Ce que contiennent les documents textuels ..... 21
II.2.2. Tables concernant les objets contenus dans une page web ..... 22
II.2.2.1. Les images ..... 22
II.2.2.2. Les adresses électroniques ..... 24
II.2.2.3. Les liens hyper-texte ..... 25

**III. Les requêtes ..... 28**
III.1. Les formules de mots ..... 28
III.2. Les requêtes simples ..... 28
III.2.1. Sans ordonnancement ..... 28
III.2.1.1. Requêtes pseudo atomiques ..... 28
III.2.1.1.1. Concernant les Fichiers divers ..... 28
III.2.1.1.2. Concernant les documents textuels ..... 32
III.2.1.1.3. Concernant les pages web ..... 33
III.2.1.1.4. Concernant les images ..... 34
III.2.1.1.5. Concernant les adresses électroniques ..... 34
III.2.1.2. Requêtes complètes ..... 35
III.2.2. Avec ordonnancement ..... 37
III.3. Les requêtes avancées et limites du modèle relationnel ..... 38
III.3.1. Exploitation du graphe d'Internet ..... 38
III.3.2. Les limites ..... 38

**BASE DE DONNEES DEDUCTIVE ..... 40**

**I. Constitution du programme Prolog ..... 40**
I.1. Les faits ..... 40
I.2. Les règles ..... 41

**II. Les requêtes ..... 43**
II.1. Les requêtes SQL ..... 43
II.1.1. Requêtes pseudo atomiques ..... 43
II.1.2. Requêtes complètes ..... 45
II.2. Les requêtes spécifiques à Prolog ..... 47

**CONCLUSION ..... 48**

# Liste des figures

Figure 1 : Un réseau simple d'ordinateurs ....3
Figure 2 : Internet, un regroupement de réseaux ....3
Figure 3 : Mini site web....4
Figure 4 : Graphe orienté du mini site ....5
Figure 5 : Les conventions de schéma entités relations....9
Figure 6 : Schéma entités relations de la base de données ....9
Figure 7 : Schéma des éléments concernant la structure physique....10
Figure 8 : Schéma des éléments concernant les pages web....11
Figure 9 : Schéma combiné des deux parties....12
Figure 10 : Schéma des tables ....14

# Introduction

Avant de commencer la description du présent projet, rappelons les buts de celui-ci :

- dans un premier temps, il doit montrer la puissance et les limites du langage SQL et plus généralement du modèle relationnel pour les bases de données ;
- ensuite il doit comparer celui-ci au modèle déductif.

C'est donc ce que nous allons nous efforcer de faire.

Pour cela, nous allons implémenter ce que pourrait être la base de données d'un moteur de recherche sur Internet. En effet, comme nous le verrons bientôt, une telle base suppose un certain nombre de contraintes particulièrement difficiles à respecter. Elle exige donc un SGBD efficace (par exemple, au niveau des accès concurrents, de l'optimisation des tables…), même si cela ne rentre pas directement dans le cadre du projet, un schéma de la base bien fait et surtout une utilisation poussée de SQL.

En fait, c'est là son principal intérêt, elle permet d'exploiter toute la puissance du modèle relationnel et même de dépasser ses limites en remarquant que certaines requêtes (pourtant légitimes d'un point de vue utilisateur) ne sont pas réalisables dans ce modèle. Nous pourrons donc montrer en implémentant justement ces requêtes dans le modèle déductif que celui-ci possède un pouvoir expressif strictement supérieur à celui du modèle relationnel.

Ce dossier s'organise assez naturellement en trois parties. En effet, avant de parler des performances de tel ou tel modèle de base de données, il nous faut présenter de manière exhaustive en quoi consiste la base de données sur laquelle nous travaillerons. Une fois cela fait, nous étudierons une implémentation de celle-ci dans le modèle relationnel. Enfin, grâce à Prolog nous allons simuler cette base de données dans le modèle déductif afin de démontrer la supériorité de ce modèle.

# Cahier des charges

Pour définir précisément la base de données sur laquelle nous allons travailler, il est indispensable de donner d'abord quelques définitions annexes pour fixer clairement le cadre du projet. En d'autres termes, il nous faut expliquer ce que l'on entend par « moteur de recherche » et par « Internet ». Ensuite nous pourront énoncer les contraintes que notre base de données doit respecter.

## I. Définition du cadre du projet

### I.1. Qu'est-ce qu'un moteur de recherche sur Internet

#### I.1.1. Point de vue utilisateur

Commençons par une définition très générale. D'un point de vue utilisateur, un moteur de recherche sur Internet est une application permettant de rechercher un document sur Internet.

Précisons maintenant quelques termes :

- Cette application se présente le plus souvent sous forme d'une page web, comme c'est le cas pour les moteurs les plus connus (Google™ [1], Lycos™ [2], Yahoo!® [3], Altavista™ [4]…). Cependant on pourrait très bien imaginer que ce soit un logiciel dédié, bien que cela soit probablement beaucoup moins pratique.
- Les documents recherchés peuvent être à priori de n'importe quel type. Cependant en pratique, la majorité des recherches porte sur des documents textuels (par exemple, un mode d'emploi de Microsoft® Word® en Latex ☺) ou des pages web qui sont en fait elles-même des documents textuels avec en plus la propriété très intéressante de pourvoir pointer vers d'autres documents (d'autres documents textuels, des images, des archives…).

#### I.1.2. Point de vue concepteur

La définition précédente, même avec les précisions apportées, est extrêmement floue et cache beaucoup de choses. En effet, la partie interface utilisateur n'est que la partie émergée de l'iceberg que constitue un moteur de recherche sur Internet. En fait, celui-ci peut se décomposer en trois parties :

- un module qui parcourt Internet à la recherche de tous les documents existants ;
- une base de données pour mémoriser tous les documents trouvés par le précédent module ;
- une interface utilisateur pour permettre à n'importe qui de faire des recherches dans cette base de données.

Dans tout cela, la conception de la dernière partie ne représente qu'une infime partie du travail exigé par les deux premières.

A ces trois parties, il faut également en rajouter une quatrième (mais elle peut aussi être intégrée directement au premier module) pour mettre à jour la base de données (par exemple, supprimer les liens qui ne fonctionnent plus).

[1] http://www.google.fr/
[2] http://www.lycos.fr/
[3] http://fr.yahoo.com/
[4] http://fr.altavista.com/

## I.2. Qu'est-ce qu'Internet

Puisque notre moteur de recherche doit travailler sur Internet, il est important de préciser ce qu'on entend par « Internet ».

### I.2.1. Sa structure physique

Internet est parfois appelé « le réseau des réseaux » (en anglais on précise toujours l'article : « The Internet ») ; cette appellation donne une idée de sa structure physique. En effet, Internet est le réseau mondial regroupant des milliers (au moins) de réseaux locaux (locaux par opposition à mondial, mais ils peuvent néanmoins être très étendus et se constituer eux-même de sous-réseaux) interconnectés entre eux. Donc si on admet qu'un réseau (Figure 1) est au moins constitué d'un serveur partageant des fichiers avec d'autres ordinateurs, alors Internet (Figure 2) n'est rien d'autre qu'un énorme regroupement de serveurs dans lequel on peut trouver une quantité astronomique de documents (au sens défini dans I.1.1).

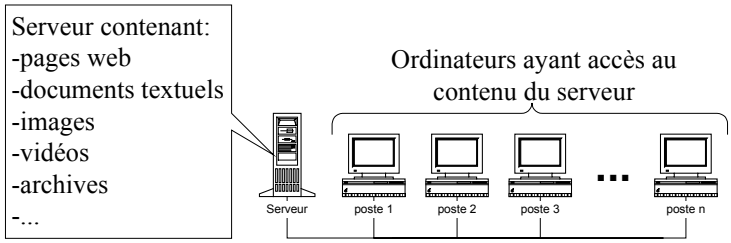



**Figure 1 : Un réseau simple d'ordinateurs**

**Rem :** Dans Figure 1 et les suivantes, on suppose qu'un poste ne fait jamais office de serveur et vice versa. Cela n'est bien entendu pas toujours vrai, cependant cette hypothèse simplificatrice ne porte pas à préjudices.

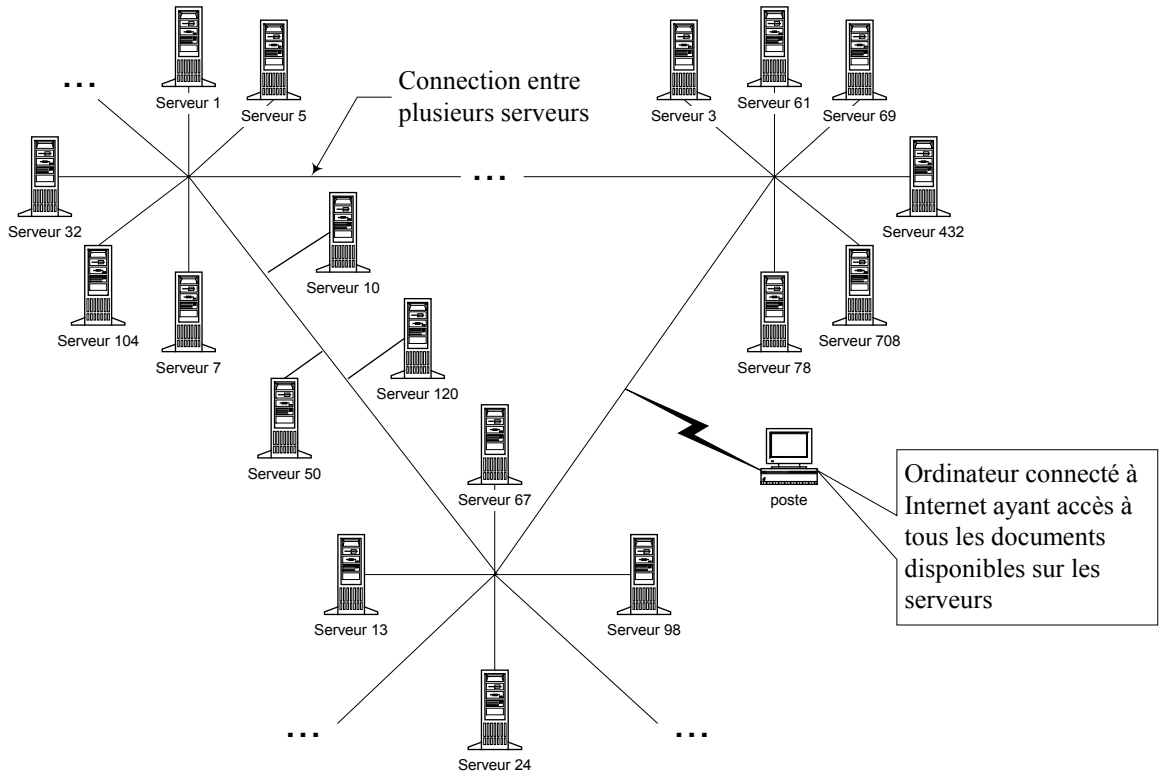



**Figure 2 : Internet, un regroupement de réseaux**

**Rem :** Dans Figure 2, les solutions techniques pour interconnecter les serveurs et pour accéder depuis n'importe quel ordinateur à n'importe quel serveur ne sont volontairement pas indiquées. D'abord parce qu'il y en a un grand nombre ; et surtout parce que cela ne joue aucun rôle dans un moteur de recherche sur Internet.

### I.2.2. Une modélisation possible

Il n'est pas du tout inintéressant de connaître la structure physique d'Internet (nous verrons au chapitre I page 8 de la partie suivante, que notre base de données reflète profondément cette structure). Cependant, ce n'est pas non plus, à priori, ce qu'il y a de plus important à savoir.

En effet, Internet est le plus souvent vu, non pas comme un regroupement de serveurs, mais comme un ensemble de pages web reliées entre elles par des liens dits hypertextes (ou hyper liens). Il peut donc être modélisé par un (gigantesque) graphe orienté dont les sommets sont des pages web et le arcs sont les liens hypertextes.

**Exemple I.1**

Un site est un ensemble de pages web hébergées sur un serveur. C'est donc un « morceau » d'Internet tel que nous venons de le définir.

Observons la correspondance Internet ↔ graphe orienté sur le mini site[5] suivant :

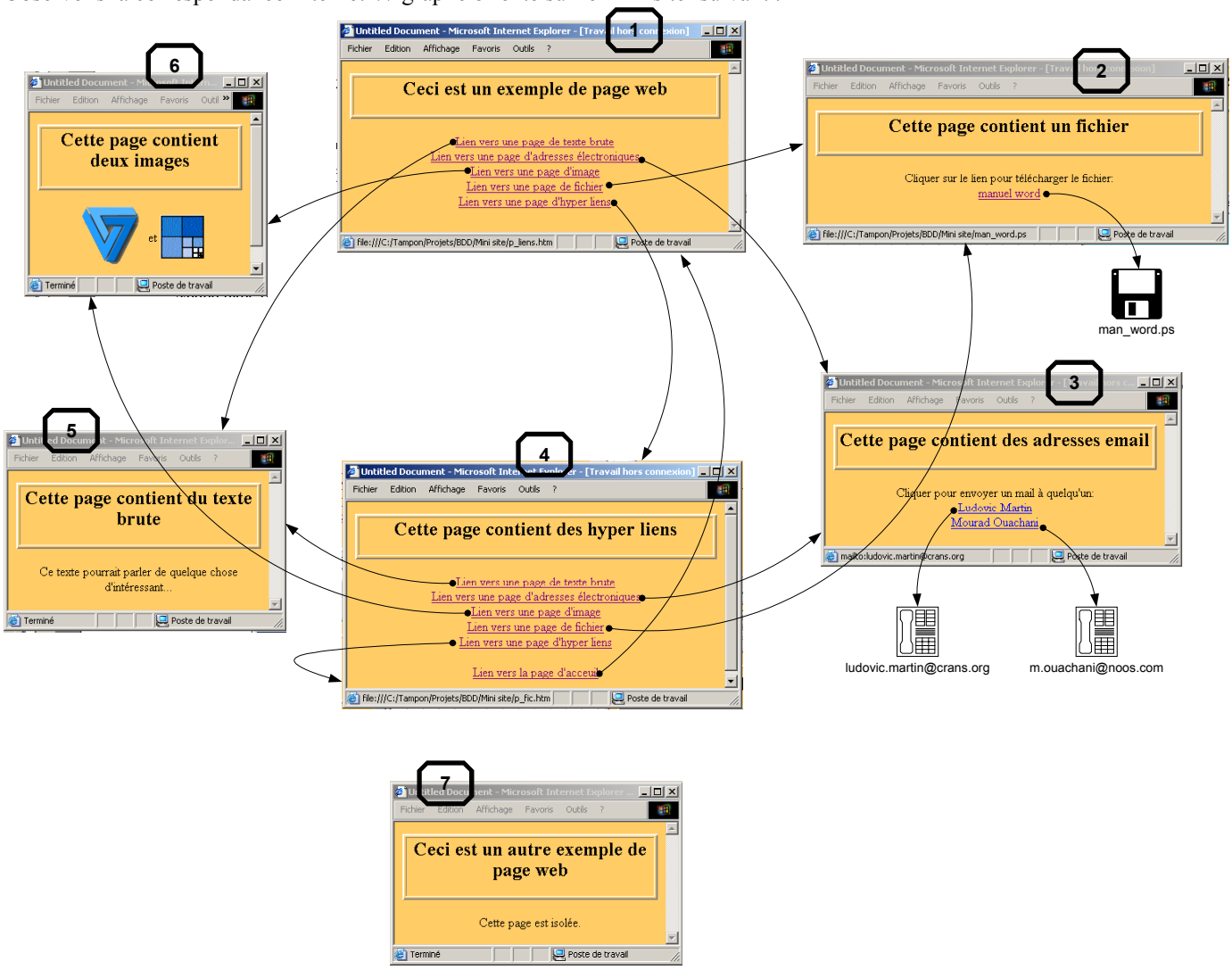



**Figure 3 : Mini site web**

[5] Les pages HTML correspondant à ce mini site sont jointes en annexe sur le CD-ROM dans le répertoire Annexes/Mini site/

A ce mini site, on peut associer intuitivement le graphe orienté suivant :

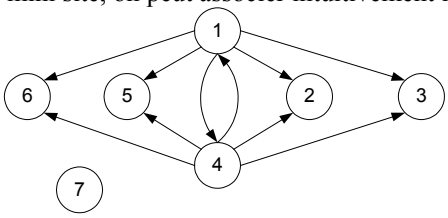



**Figure 4 : Graphe orienté du mini site**

Typiquement une page web est un document au format HTML (ou quelque chose de similaire). Sans rentrer dans une explication de ce format, on peut dire qu'une page web peut contenir du texte, des images, des (liens vers des) fichiers quelconques, des adresses électroniques et des hyper liens. De plus, fichiers, adresses électroniques et hyper liens peuvent être commentés. Ainsi, il est possible d'affiner la modélisation en précisant que les sommets et les arcs possèdent un certain nombre de propriétés :

- un arc a comme propriétés les commentaires associés à l'hyper lien correspondant ;
- un sommet a comme propriétés le texte, les images, les adresses électroniques et les fichiers que la page web correspondante contient.

**Rem :** Dans une modélisation mathématique, ces propriétés peuvent être données par l'intermédiaire de deux fonctions :

$$\pi : S \rightarrow P$$
$$\psi : A \rightarrow P$$

où S et A sont les sommets et les arcs du graphe G (S,A) et P est un ensemble de propriétés à définir.

Cette modélisation est très intéressante car c'est cette propriété d'Internet qui est utilisée par tous les moteurs de recherche pour parcourir Internet à la recherche des documents existants comme nous allons le voir au chapitre II.1.2.1.

# II. Les objectifs du projet

## II.1. Les difficultés d'implémenter un moteur de recherche

### II.1.1. Les contraintes

L'implémentation d'un moteur de recherche sur Internet est très complexe. En effet, les exigences des utilisateurs sont difficiles à réaliser. De fait, le moteur doit :

- être toujours disponible (la maintenance ne doit pas perturber le fonctionnement du moteur) ;
- contenir le plus grand nombre possible de références (l'idéal serait de réussir à référencer tout Internet) ;
- répondre intelligemment aux requêtes qu'on lui soumet (moteur de recherche télépathe ?) ;
- répondre rapidement (quelque soit l'état du réseau et le nombre d'utilisateurs connectés simultanément).

Notons de plus que la topologie d'Internet (du point de vue de sa structure physique, mais aussi et surtout de sa représentation sous forme de graphe) est inconnue, gigantesque et en constante évolution.

### II.1.2. Quelques problèmes et solutions techniques

Dans ce projet, nous n'allons pas implémenter complètement un moteur de recherche puisque nous nous limiterons à sa base de données (ce que nous verrons dans le chapitre suivant). Il est donc possible de sauter ce chapitre puisqu'il n'est pas strictement indispensable à la compréhension du projet. En effet, nous allons donner ici quelques solutions techniques générales pour réaliser certaines tâches (parmi celles vues au chapitre I.1.2 page 2) en respectant les contraintes imposées un peu plus haut.

#### *II.1.2.1. Le parcours d'Internet*

La technique qui est toujours employée pour parcourir Internet s'appuie sur la représentation d'Internet sous forme de graphe. L'idée est de partir d'un (ou plusieurs) sommet(s) et de passer aux sommets adjacents après avoir mémoriser ses propriétés. Concrètement, cela revient à regarder le contenu d'une page web et à suivre les hyper liens qui s'y trouvent.

On voit immédiatement dans **Exemple I.1** page 4 que cette stratégie présente plusieurs inconvénients :

1. Si une page est isolée (comme la page 7),
   1.1. soit elle ne sera jamais découverte, si on commence par une autre page
   1.2. soit, si on commence par elle, on ne découvrira aucune autre page.
2. Si un ensemble de pages forment un circuit, il y a un risque de boucler à l'infini dessus.

Pour résoudre partiellement le problème 1.2, il suffit de commencer par plusieurs pages à la fois (en espérant ne pas tomber que sur des pages isolées). Pour le problème 1.1, souvent on utilise la technique de l'enregistrement : c'est le créateur de la page qui indique explicitement au moteur de recherche que sa page existe. Le dernier problème, lui, n'a pas de solution évidente.

#### *II.1.2.2. Mémoriser les documents trouvés*

La mémorisation des documents trouvés se fait bien sûr à l'aide d'une base de données. Cependant, l'architecture de celle-ci ainsi que le SGBD qui la manipule sont soumis à fortes contraintes :

1. la quantité de données à stocker est énorme (idéalement il faudrait stocker tout Internet) ;
2. les accès concurrents sont très nombreux ;
3. le tout doit être stable (on ne peut pas se permettre de perdre tout ou partie des données).

La seule façon de résoudre le premier problème est d'utiliser un grand nombre d'ordinateurs, par exemple Google utilise plusieurs milliers de PCs. Malheureusement cette solution va à l'encontre du problème 3 car plus on multiplie les ordinateurs, plus on a de chance d'avoir des avaries. Il est donc nécessaire de prévoir des sauvegardes de la base de données, mais cela aggrave le problème 2 puisqu'il ne faut pas oublier que le moteur de recherche doit être toujours disponible. Ici encore, il n'y a pas de solution toute faite.

#### *II.1.2.3. Interface utilisateurs*

L'interface utilisateur est assurée par une page web dynamique capable d'interroger la base de données et d'afficher le résultat sous forme exploitable par l'utilisateur. Même s'il faut veiller à ce que cette page soit facile à utiliser, rapidement accessible (même pour les utilisateurs n'ayant pas une connexion rapide) et néanmoins complète, c'est probablement la seule partie du moteur de recherche qui ne pose pas de véritable problème.

## II.2. Simplification des objectifs

### II.2.1. Nouveaux objectifs

Etant donné la difficulté de réaliser un moteur de recherche sur Internet complet, et puisque cela n'aurait de toute façon aucun intérêt dans un projet de Bases de Données Déductives, nous allons limiter nos objectifs.

Tout d'abord, nous n'allons pas travailler sur Internet mais sur une infime partie du réseau mondial (environ un millier de pages visitées). Cela représente cependant un échantillon assez conséquent et surtout suffisant pour pouvoir tester tout type de requête.

Ensuite, nous ne nous intéresserons qu'à une toute petite partie de la conception d'un moteur de recherche :

- mettre en place une base de données efficace ;
- interroger cette base de données.

Cela signifie que nous ne nous occuperons ni de comment parcourir Internet ni de comment remplir la base de données. En fait, pour remplir notre base de données, nous avons utilisé le programme (légèrement modifié) que nous avions conçu pour le projet[6] de Réseaux au premier semestre. Ce programme nous a permis de remplir une première base de données dont nous avons ensuite extrait[7] les données nécessaires pour remplir partiellement notre base de données. Ensuite, nous avons terminé de la remplir grâce à un programme[8] aléatoire permettant de simuler le contenu des documents textuels.

### II.2.2. Nouvelles contraintes

Vue les objectifs modestes que nous nous sommes fixés, le nombre de contraintes à respecter est assez réduit :

- la base de données doit être robuste (on ne doit pas pouvoir insérer n'importe quoi dedans) ;
- elle doit permettre les accès concurrents (pour les requêtes) ;
- elle doit être le plus possible évolutive (il ne doit pas être nécessaire de changer son schéma en cas de modifications légères d'Internet) ;
- elle doit permettre n'importe quelle requête imaginable (étant donné que nous n'implémentons pas d'interface utilisateur, on peut imaginer que celle-ci autorise n'importe quel type d'interrogation) ;
- les réponses aux requêtes doivent être rapides.

[6] La documentation (tenant compte des modification apportée) et le code source de ce projet sont joints en annexe sur le CD-ROM dans le répertoire Annexes/Crawler/
[7] Le code source du programme de remplissage de la nouvelle base de données est joint en annexe sur le CD-ROM dans le répertoire Annexes/Remplisseur
[8] Ce programme est en fait une sous-partie du programme de remplissage

# Base de données relationnelle

Maintenant que les objectifs du projet ont été définis précisément, voyons une façon de les réaliser dans le modèle relationnel.

## I. Le modèle entités-relations

### I.1. Objectifs : Modéliser le fonctionnement d'Internet

Le schéma de la base de données (Figure 6 page 9) s'appuie à la fois sur la structure physique (chapitre I.2.2.1 page 9) et sa modélisation sous forme de graphe d'Internet (chapitre I.2.2.2 page 11) vues dans la partie précédente respectivement au chapitre I.2.1 page 3 et au chapitre I.2.2 page 4.

#### I.1.1. Avantages

Ce schéma a d'abord l'avantage d'être assez naturel, donc plus facilement compréhensible, du moins si on connaît le fonctionnement d'Internet.

De plus, si on fait l'hypothèse qu'Internet fonctionnera à peu prêt toujours de la même manière (ce qui est raisonnable), alors ce schéma devrait être pérenne. En effet, il permet n'importe quelles modifications dans la mesure où les invariants suivants sont respectés :

- les fichiers (de tout type) sont toujours localisés sur un et un seul serveur ;
- l'accès à un fichier se fait toujours grâce à l'adresse de son serveur et de son chemin sur celui-ci plus éventuellemnt quelques informations supplémentaires (qui elles peuvent être modifiées sans que cela n'entraîne de changement fondamental dans la structure que la base de données) ;
- la navigation sur les pages web se fait toujours par l'intermédiaire d'hyper liens.

**Rem :** Pour les personnes sceptique en ce qui concerne l'hypothèse faite, on peut argumenter que si Internet changeait radicalement, alors c'est peut être l'intérêt même des moteurs de recherche qui serait remis en cause, donc la base de données n'aurait de toute façon plus de raison d'exister.

Enfin et surtout, toutes les requêtes imaginables sont implémentables (dans la limite de la puissance du modèle relationnel).

#### I.1.2. Inconvénients

Du fait du fonctionnement non parfaitement trivial d'Internet, le schéma est relativement dense. Donc il peut être un peu compliqué à comprendre même s'il est naturel.

Enfin, si toutes les requêtes imaginables sont réalisables, cela ne veut pas dire que cela soit facile. Comme nous le verrons au chapitre II.1.2 page 15, certaines requêtes pourtant naturelles sont particulièrement compliquées (du moins si on considère que le SGBD n'a pas d'optimisateur de requête).

## I.2. Schéma

### I.2.1. Conventions

Avant de donner le schéma entités-relations de la base de données, voici les conventions nécessaires à sa compréhension :


**Figure 5 : Les conventions de schéma entités relations**

### I.2.2. Schéma

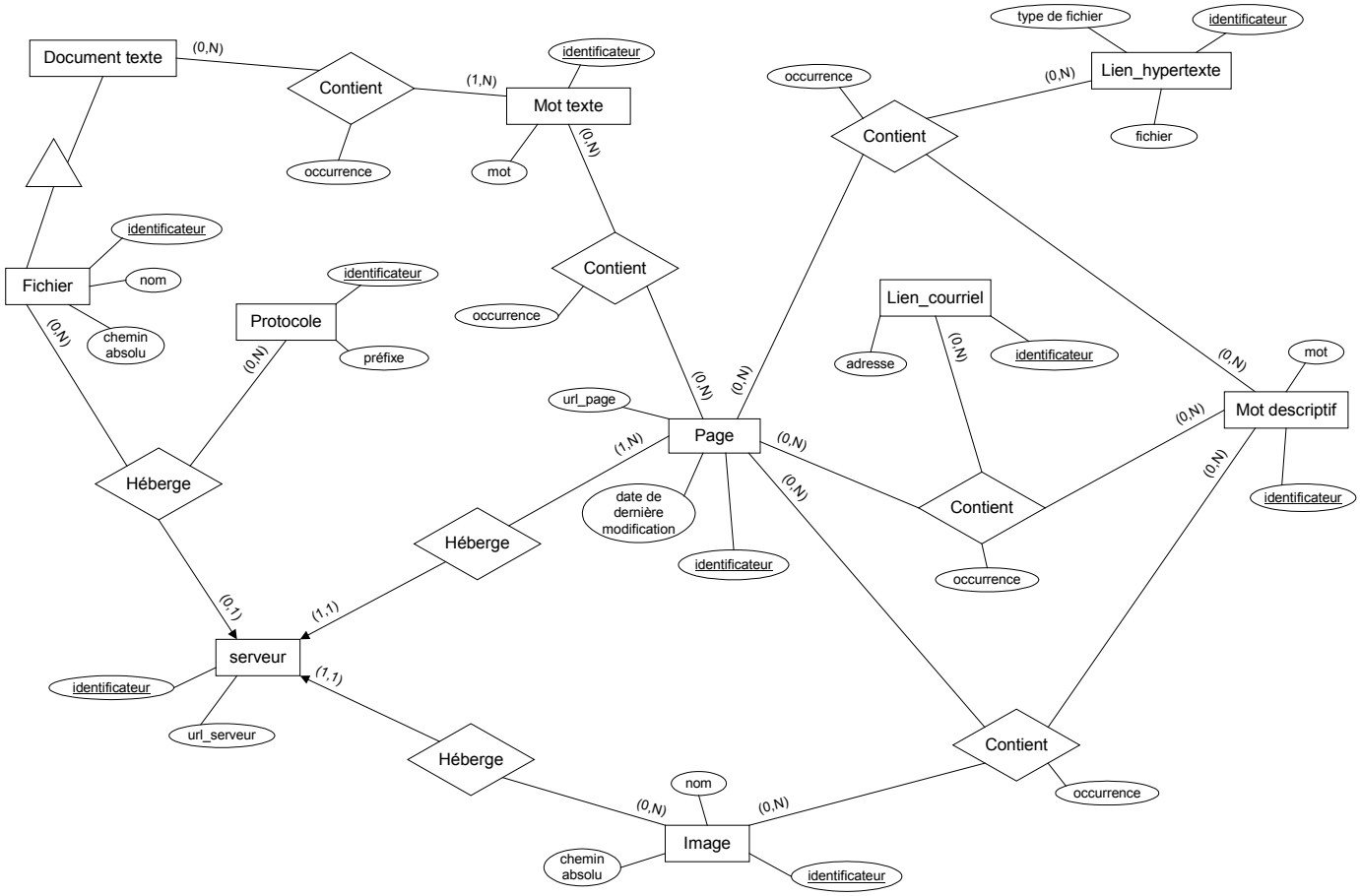



**Figure 6 : Schéma entités relations de la base de données**

Avant de regarder plus en détail à quoi correspondent chaque élément de ce schéma, on peut tout de suite dire que les deux entités principales sont :

- `Serveur` autour de laquelle s'organisent tous les éléments traitant de la structure physique d'Internet ;
- `Page` autour de laquelle on retrouve tous les éléments correspondant à la partie visible / visuelle d'Internet, c'est à dire en gros ce qu'on peut voir sur son écran lorsqu'on navigue sur Internet.

#### *I.2.2.1. Les éléments concernant la structure physique d'Internet*

Nous avons vu dans la partie précédente qu'un serveur peut contenir (dans le langage Internet, on dit héberger) n'importe quel type de fichiers. Cependant, tous ces types ne jouent pas le même rôle et n'ont pas la même importance. Dans notre schéma, nous avons distingué quatre catégories de types de fichier :

- les pages web qui est un type très particulier comme nous le verrons dans la section suivante ;

- les documents textuels (autres que les pages web) qui ont comme propriété de contenir du texte qui est un critère important pour la recherche ;
- les images visibles sur les pages web (cela n'inclut donc pas les images stockées sur le serveur mais non directement visible par un navigateur Internet) qui est un type souvent recherché par les utilisateurs ;
- enfin, tous les autres types.

Ces types correspondent respectivement aux entités `Page`, `Document texte`, `Image` et `Fichier` reliées à l'entité `Serveur` par des relations `Héberge`.

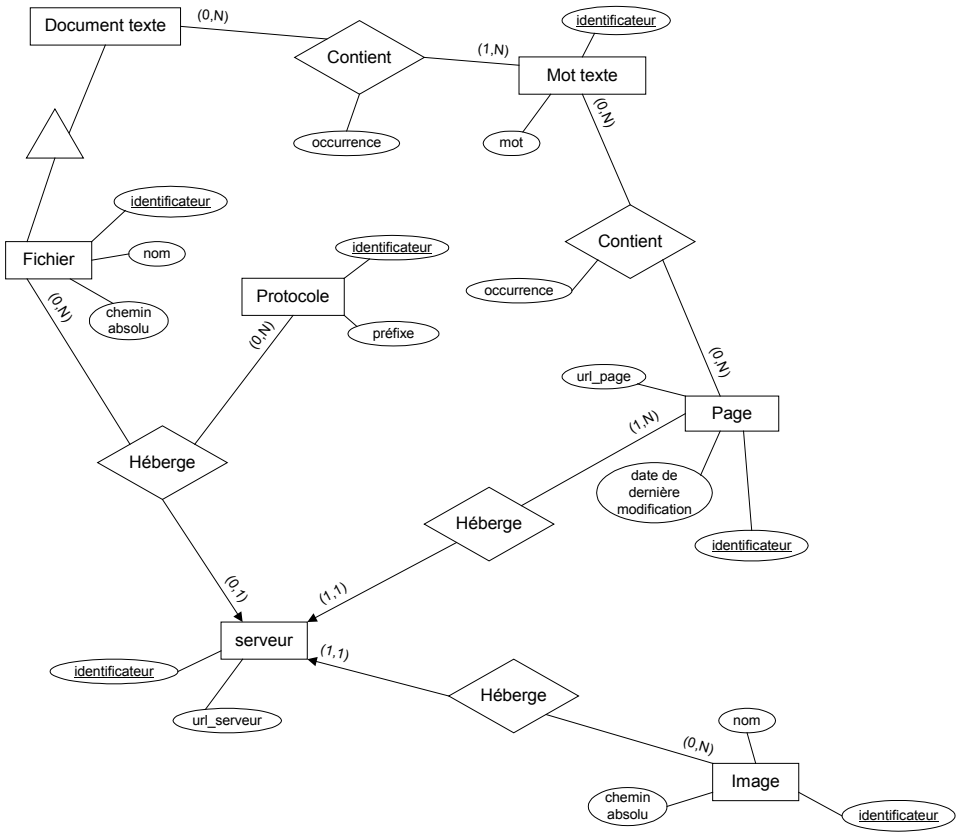



**Figure 7 : Schéma des éléments concernant la structure physique**

Sur l'extrait du schéma ci-dessus, on peut remarquer que nous avons fait le choix de :

- mettre `Document texte` en tant que sous-type de `Fichier` ;
- laisser `Page` et `Image` séparé de `Fichier`.

Ce choix peut sembler étrange à priori. En ce qui concerne `Page` (qui pourrait même être un sous-type de `Document texte`), le choix est à peu prêt naturel car les pages web sont un type vraiment très particulier. De plus, si c'est utile pour tous les fichiers de préciser d'un coté leur nom (attribut `nom`) et de l'autre leur chemin absolu (attribut `chemin absolu`) notamment pour faire des recherche sur leur nom, ça n'a aucun intérêt pour les pages web et l'entité `Page` n'a donc qu'un attribut `url_page` qui concatène ces deux attributs en un. Cela rend donc impossible de déclarer `Page` comme sous-type de `Fichier` ou `Document texte`. Certaines personnes critiques pourraient objecter que « qui peut le plus, peut le moins » ; c'est à dire que, puisqu'à partir de deux attributs on peut toujours en faire un seul, alors il vaut mieux découper `url_page` et mettre `Page` en sous-type. Cependant cela n'est pas tout à fait vrai car parfois le nom de la page web n'est pas spécifier (par exemple, http://www.ufr-info-p7.jussieu.fr/) donc il faudrait laisser l'attribut `nom` vide ce qui devrait être interdit pour tous les autres fichiers, mais alors comment faire la différence entre les différents types ? Bien sûr, il existe des solutions pour faire cela, mais ce ne sont pas celles-ci que nous avons retenues car peu explicites. Enfin, on peut remarquer que pour `Page` et `Image` contrairement à `Fichier` la relation `Héberge` ne prend pas en compte le protocole de connexion (HTTP ou FTP) car on

considère qu'il s'agit toujours de HTTP. Cela est vrai aujourd'hui mais pourrait changer dans l'avenir. Cependant, ce serait tout de même très étonnant et c'est pourquoi nous avons fait ce choix.

**Rem :** Dans notre première base de données, nous n'avions pas déclaré `Document texte` comme sous-type de `Fichier`. Nous avons modifié cela pour deux raisons. D'abord car c'est beaucoup plus naturel (un document textuel est juste un type particulier de fichier et il n' y a pas de problème d'incompatibilité comme pour les images et les pages web) et cela améliore la lisibilité du schéma. Ensuite car cela sous-entend qu'on pourrait rajouter plus tard d'autres sous-types si le besoin s'en faisait sentir.

Le protocole de connexion, celui-ci est indiqué par l'intermédiaire de l'entité `Protocole` et de la relation `Héberge`.

Enfin, il reste une entité (`Mot_texte`) et deux relations (`Contient`) dans le schéma Figure 7 dont nous n'avons pas encore parlés. Ces éléments se comprennent assez facilement. Ils permettent de mémoriser tout le contenu textuel (les mots en d'autre termes) d'une part des pages web et d'autre part des documents textuels. L'attribut `occurrence` des relations `Contient` est peut être un peu moins explicite. Il sert à connaître combien de fois un mot donné apparaît dans le document ce dont nous nous servirons pour classer le résultat des requêtes au chapitre III.3 page 38.

#### *I.2.2.2. Les éléments concernant la partie visible d'Internet / Modélisation par graphe*

Nous avons dit dans la partie précédente qu'une page web peut contenir du texte, des images, des liens vers des fichiers quelconques (archives, documents textuels, images, vidéo…), des adresses électroniques et des hyper liens. En fait, les liens vers fichier sont simplement des hyper liens. Cela fait donc quatre types d'objet pouvant apparaître sur une page web. Chacun correspond à une entité dans le schéma de la base de données.

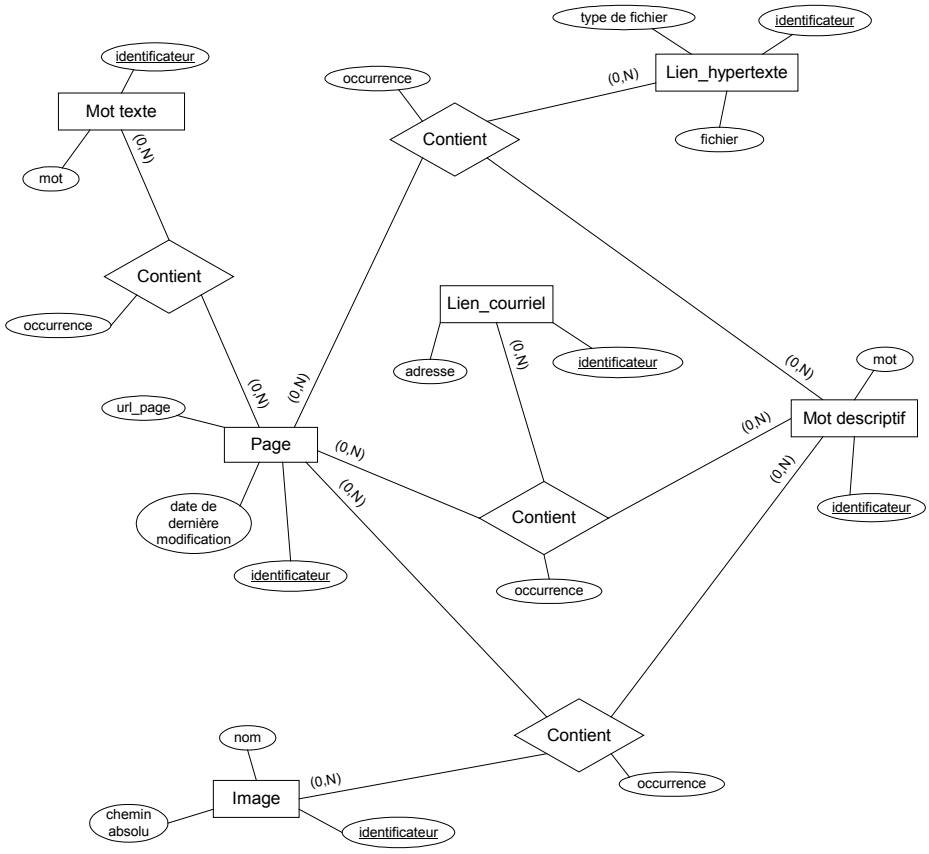



**Figure 8 : Schéma des éléments concernant les pages web**

Nous avons déjà parlé des entités `Mot_texte` et `Image`. Pour l'entité `Page`, rajoutons seulement que l'attribut `date de dernière modification` sert au moteur de recherche lorsqu'il parcourt Internet de ne pas re-regarder une page si elle n'a pas été modifiée depuis sa dernière visite. Les entités `Lien_courier` et `Lien_hypertexte` correspondent respectivement aux adresses électroniques et aux hyper liens rencontrées sur les pages web.

Parmi les attributs de `Lien_hypertexte` deux attributs (`type de fichier` et `fichier`) ne sont pas très intuitifs.

Pour pouvoir les comprendre il faut savoir qu'un hyper lien peut pointer vers tous les fichiers présents sur le serveur, c'est à dire vers tous les objets correspondant aux entités `Page`, `Document texte`, `Image` et `Fichier`. Donc `type de fichier` indique le type de fichier vers lequel l'hyper lien pointe et `fichier` de quel fichier précisément il s'agit.

Les entités `Image`, `Lien_courier` et `Lien_hypertexte` sont reliées à `Page` par des relations `Contient` qui indiquent d'une part, quels objets se trouvent dans quelle page web et d'autre part, quel commentaire est associé sur cette page à cet objet grâce à l'entité `Mot descriptif` qui correspond aux commentaires découpés en mots. En pratique, le commentaire d'une image se trouve dans le champ `ALT` de la balise HTML `<IMG>` et celui d'un hyper lien ou d'une adresse électronique entre les balises `<HREF>` et `</HREF>`. L'attribut `occurrence` de ces relations `Contient` a le même sens que pour la relation `Contient` entre `Page` et `Mot texte`, c'est à dire qu'il indique combien de fois un objet est associé, dans une page web donnée, à un même mot.

Enfin, il faut noter que les éléments de cette partie de schéma permettent bien de modéliser Internet comme nous l'avions défini au chapitre I.2.2 page 4 de la partie précédente :

- les entités `Page` et `Lien_hypertexte` et la relation `Contient` qui les relie permettent de reconstruire le graphe G(S,A) d'Internet ;
- les entités `Mot texte`, `Lien_courier`, `Image` et `Lien_hypertexte` (pour les types de fichiers autres que page web) et les relations `Contient` correspondantes servent à fabriquer les fonctions φ (resp. ψ) qui associent à chaque sommet (resp. arc) un ensemble de propriétés.

#### *I.2.2.3. Résumé*

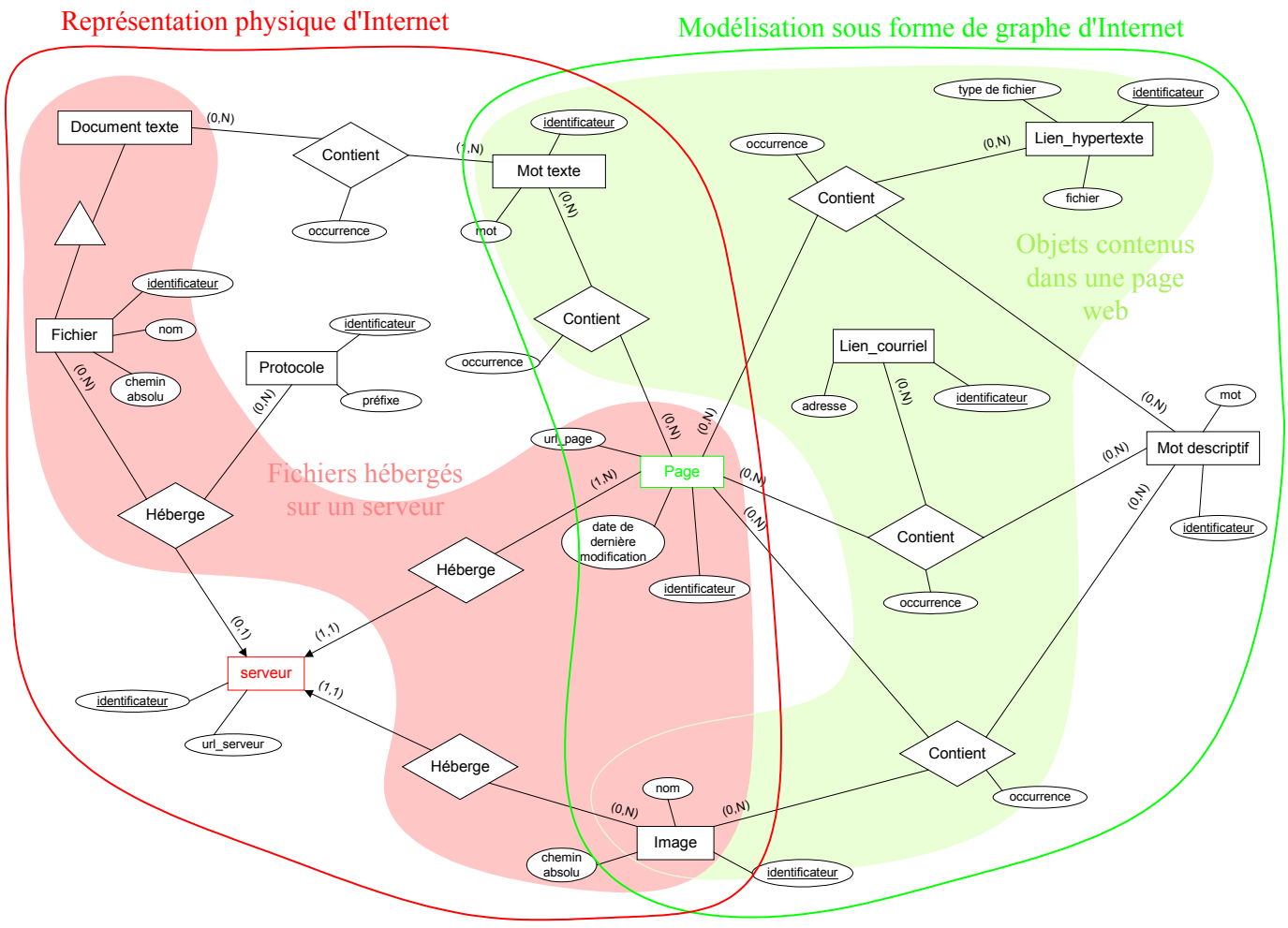


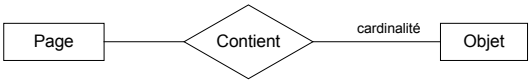

**Figure 9 : Schéma combiné des deux parties**

## I.3. Contraintes

### I.3.1. Cardinalités

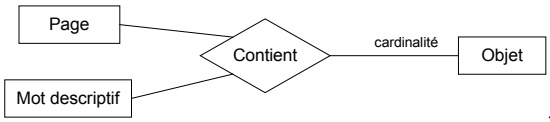
Les cardinalités sont écrites directement sur le schéma de la base (Figure 6 et Figure 9). Elles sont toutes facilement compréhensibles.

Par exemple, une page web peut être vide ou contenir un grand nombre d'objets de tout type ; donc la cardinalité de la relation




est comprise entre 0 et N (cela reste vrai si on ajoute à cette relation des commentaires). En revanche, un objet est contenu dans au moins une page web, donc on serait tenté de mettre la cardinalité inverse entre 1 et N. Mais dans le cas de l'entité `Mot texte`, il ne faut pas oublier qu'elle est partagée avec `Document texte` donc certains mots peuvent n'exister que sur des documents textuels ; alors la cardinalité est comprise entre 0 et N. Et dans le cas de `Lien_hypertexte`, `Lien_courier` et `Image`, il faut prendre en compte `Mot descriptif`. Voici le raisonnement fait : sur une page web, il peut exister zéro ou plusieurs objet(s) de même type associé(s) à un commentaire donné.

Ce qui nous donne une cardinalité de 0 à N pour la relation




.

Remarquons que contrairement aux pages web, les documents textuels ne peuvent pas être vides (auquel cas ils n'auraient de toute façon aucun intérêt) d'où une cardinalité de (1,N) au lieu de (0,N) de `Document texte` vers `Mot texte`.

Les seules cardinalités intéressantes sont entre `Page` et `Serveur`. En effet, dans le sens `Page` vers `Serveur`, la cardinalité est (1,1) ce qui implique que la relation et fonctionnelle (ce qui aura des conséquences lors de l'implémentation de la base de données). Dans le sens inverse, nous avons fait un choix discutable mais néanmoins naturel en disant qu'il ne peut pas y avoir de serveur n'hébergeant aucune page web, d'où une cardinalité de (1,N).

### I.3.2. Attributs

Nous avons fait le choix d'associer à toutes les entités un attribut identificateur pour servir tout naturellement de clef primaire. Ce choix se justifie car il permet :

- à priori un gain de place dans la base de données : si la clef sert à plusieurs endroits (ce qui est souvent les cas) on ne répète qu'un entier qui prend généralement moins de place qu'une chaîne de caractère pas exemple ;
- peut être d'accélérer légèrement les requêtes puisque les algorithmes de recherche d'un élément dans un ensemble sont souvent plus à l'aise en manipulant des entiers que d'autres types d'objet.

Notons aussi que tous les attributs doivent être remplis, éventuellement avec une valeur par défaut pour `date de dernière modification`.

# II. L'implémentation

## II.1. Présentation globale

### II.1.1. La base de données

Afin de faciliter la compréhension du rôle de chaque table et de leurs attributs, voici le schéma déjà vu (Figure 6) de la base de données où les noms des entités / relations ont été remplacés par celui des tables correspondantes :

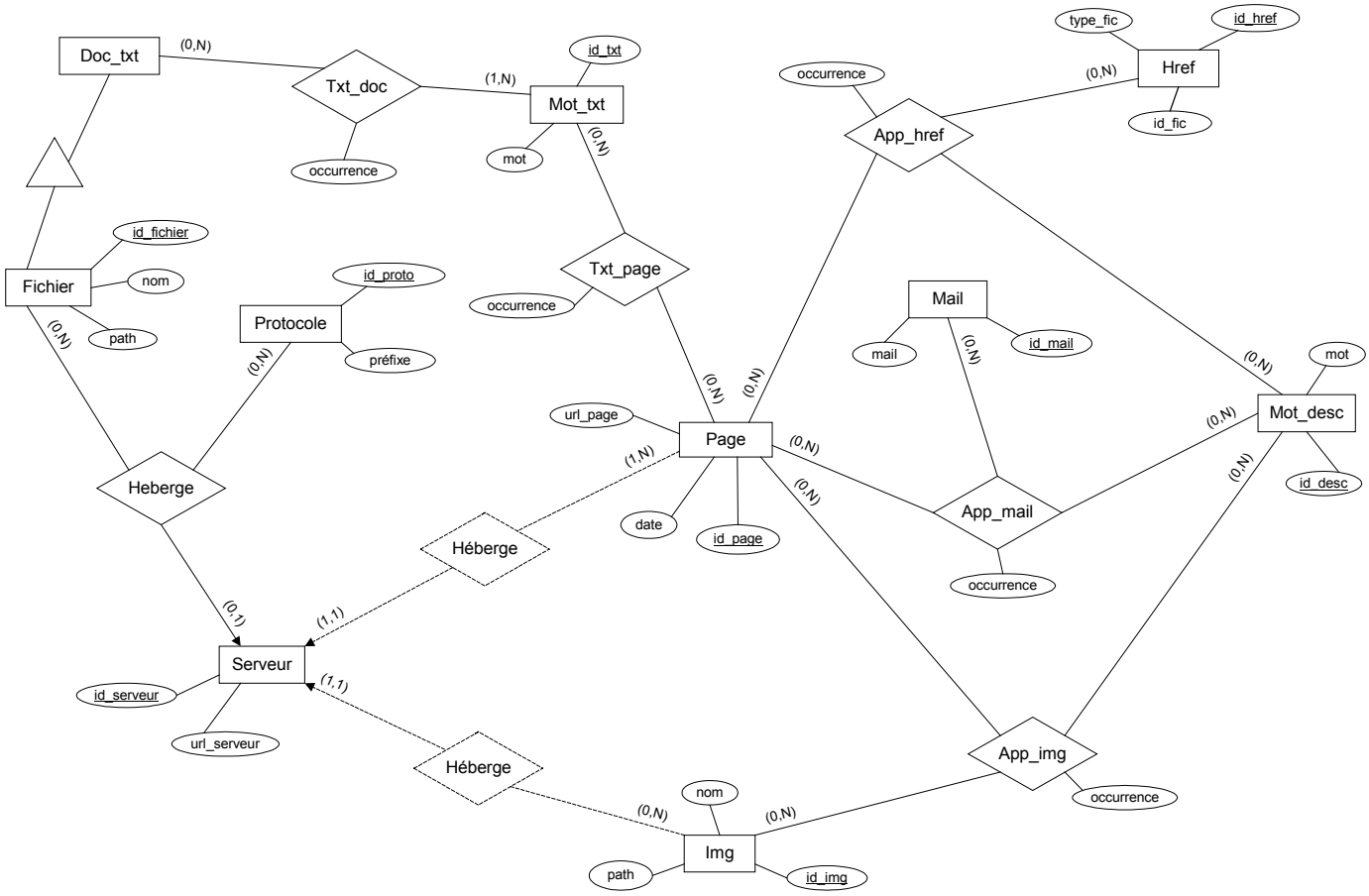



**Figure 10 : Schéma des tables**

**Rem :** Les éléments en pointillés ne correspondent à aucune table.

On peut voir que certaines relations `Héberge` ne sont pas traduites en table. Cela s'explique par le fait qu'il s'agit de relations fonctionnelles comme nous l'avons vu au chapitre précédent.

**Rem :** Dans notre première implémentation, nous n'avions pas traduit l'entité `Document texte` en une table `Doc_txt`. La raison de ce choix est que le sous-type `Document texte` n'ayant pas d'attribut supplémentaire par rapport à `Fichier`, cette table n'apporte pas d'information nouvelle puisqu'il est possible de retrouver son contenu grâce à la table `Txt_doc` (car tous les documents textuels contiennent au moins un mot et possède donc au moins une entrée dans cette table). Cela permettait donc d'économiser de la place (n'oublions pas que la quantité de données à stocker est gigantesque, donc un gain de place est toujours le bienvenu). Cependant ce choix était trop pénalisant :

- la base de données perdait de son pouvoir d'évolution puisqu'il n'était plus possible à priori de rajouter des attributs à cette table (puisqu'elle n'existe pas !) ;
- la simple requête pour savoir, parmi les fichiers, lesquels sont de type textuel était très coûteuse puisque la table `Txt_doc` est beaucoup plus grande (normalement d'un facteur 100 ou 1000) que `Doc_txt`.

### II.1.2. Les données

Nous avons déjà parlé rapidement du remplissage de la base de données au chapitre II.2.1 page 6. Insistons sur le fait que les majorité des données proviennent réellement d'Internet ce qui explique par exemple pourquoi la table `Mot_desc` contient des mots étranges.

La page (http://java.sun.com/j2se/1.4.1/docs/index.html) qui a été choisie pour commencer le parcours d'Internet l'a été pour plusieurs raisons :

- pas ou peu de liens morts ou mal formés (le meilleur site pour trouvé des liens mal formés c'est http://www.ufr-info-p7.jussieu.fr/Paccueil.html) ;
- peu de commentaires incohérents ou illisibles ;
- tous les types de fichiers existent.

Une partie des données de la base ne vient cependant pas d'Internet. Il s'agit de toutes les entrées des tables `Txt_doc`, `Mot_txt` et `Txt_page`. En effet, il est très difficile de récupérer les mots de n'importe quel type de fichier contenant du texte (cela dépasse largement le cadre du simple projet scolaire) et surtout il faut quand même garder à la base de données une taille raisonnable. Donc pour remplir ces pages nous avons fait un programme aléatoire associant quelques mots à chaque document textuel et page web.

## II.2. Les tables en détail

Maintenant que la signification de toutes les tables est à peu prêt claire, nous allons voir chacune d'elles en détail. Etant donné que nous en avons déjà bien parlé auparavant, nous nous contenterons le plus souvent de donner leur code ainsi qu'un extrait des données qu'elles contiennent.

### II.2.1. Tables concernant les fichiers hébergés par un serveur

#### *II.2.1.1. Les serveurs*

Les serveurs sont contenus dans la table `Serveur` :

```
CREATE TABLE Serveur (
  url_serveur TEXT NOT NULL UNIQUE,
  id_serveur SERIAL PRIMARY KEY
);
```

Il y en a 47 au total et voici un extrait de la table :

```
moteur_web=> SELECT * FROM Serveur;
      url_serveur       | id_serveur
------------------------+------------
 java.sun.com           |          1
 www.sun.com            |         15
 developer.java.sun.com |         39
 ftp.java.sun.com       |         97
 docs.sun.com           |        118
 www.w3.org             |        128
 servlet.java.sun.com   |        149
 wwws.sun.com           |        150
 search.java.sun.com    |        170
 www201.ikiosk.com      |        185
 www.jcp.org            |        190
 jcp.org                |        379
 www.ietf.org           |        385
 ftp.omg.org            |        475
 ftp.javasoft.com       |        510
 wireless.java.sun.com  |        513
```

```
 forum.java.sun.com     |        514
 research.sun.com       |        553
 www.confluent.fr       |        555
 www.winzip.com         |        559
 home.mcom.com          |        566
 ...
```

#### *II.2.1.2. Les images*

Les images visibles sur une page web sont stockées dans la table `Img` :

```
CREATE TABLE Img (
  id_serveur INTEGER,
  path TEXT NOT NULL,
  nom TEXT NOT NULL,
  id_img SERIAL PRIMARY KEY,
  UNIQUE(id_serveur,path,nom),
  FOREIGN KEY  (id_serveur) REFERENCES Serveur
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

On peut remarquer l'attribut `id_serveur` qui provient de la relation fonction entre les entités `Serveur` et `Image`. De plus, comme nous l'avions dit plus haut, aucun champ ne peut être `null`.

Il y a 65 entrées dans cette table, et en voici un extrait :

```
moteur_web=> SELECT * FROM Img;
id_serveur|            path                 |              nom                 | id_img
----------+-----------------------------------+----------------------------------+-------
    1 | /j2se/1.4.1/docs/images/             | download.arrow.gif               |      1
    1 | /j2se/1.4.1/docs/images/             | javalogo52x88.gif                |      2
    1 | /j2se/1.4.1/docs/images/             | sunlogo64x30.gif                 |      3
    1 | /images/                             | v4_java_sun_com.gif              |      4
    1 | /images/                             | v3_sun_logo.gif                  |      5
    1 | /j2se/1.4.1/ja/images/               | Japanese12B.gif                  |      6
    1 | /images/                             | v3_print.gif                     |      7
    1 | /images/                             | v4_java_logo.gif                 |      8
   15 | /im/home/                            | masthead_slogan.gif              |      9
   15 | /im/                                 | sun_logo.gif                     |     10
   15 | /pics/promos/                        | b5_nc_03_q2.gif                  |     11
   15 | /2003-0422/images/                   | b3_nascar-java.jpg               |     12
   15 | /pics/promos/                        | b5_sun_fire_blade_platform.gif   |     13
   15 | /pics/promos/                        | b5_get_java.gif                  |     14
   15 | /pics/promos/                        | b5_why_not.gif                   |     15
   15 | /im/                                 | a.gif                            |     16
    1 | /j2se/1.4.1/images/                  | view.option.gif                  |     17
    1 | /j2se/1.4.1/docs/guide/rmi/images/   | javalogo52x88.gif                |     18
    1 | /j2se/1.4.1/images/                  | download.option.gif              |     19
    1 | /j2se/images/                        | j2se_sm.gif                      |     20
    1 | /images/                             | Japanese12B.gif                  |     21
    1 | /images/misc/                        | arrow.gif                        |     22
   39 | /developer/onlineTraining/new2java/  | professionals.gif                |     23
   39 | /developer/onlineTraining/new2java/  | new.gif                          |     24
   39 | /developer/onlineTraining/new2java/  | step3.gif                        |     25
   39 | /developer/onlineTraining/new2java/  | step2.gif                        |     26
   39 | /developer/onlineTraining/new2java/  | step1.gif                        |     27
   39 | /images/                             | v4_java_sun_com.gif              |     28
   39 | /images/                             | v4_java_logo.gif                 |     29
 ...
```

En se référant à la table `Serveur`, on peut dire par exemple que l'URL de la première image est :

http://java.sun.com/j2se/1.4.1/docs/images/download.arrow.gif

#### *II.2.1.3. Les pages web*

Les pages web sont stockées dans la table Page :

```
CREATE TABLE Page (
  id_serveur INTEGER,
  url_page TEXT NOT NULL,
  date INTEGER DEFAULT 0,
  id_page SERIAL PRIMARY KEY,
  UNIQUE(url_page,id_serveur),
  FOREIGN KEY  (id_serveur) REFERENCES Serveur
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Ici aussi on retrouve l'attribut `id_serveur` qui provient de la relation fonction entre les entités `Serveur` et `Page`.

Il y a 1139 entrées dans la table et voici un extrait :

```
moteur_web=> SELECT * FROM Page;
 id_serveur |                        url_page                        | date | id_page
------------+--------------------------------------------------------+------+---------
          1 | /j2se/1.4.1/docs/guide/xml/index.html                  |    0 |       1
          1 | /j2se/1.4.1/docs/guide/serialization/index.html        |    0 |       2
          1 | /j2se/1.4.1/docs/guide/awt/index.html                  |    0 |       3
          1 | /j2se/1.4.1/compatibility.html                         |    0 |       4
          1 | /j2se/1.4.1/docs                                       |    0 |       5
          1 | /j2se/1.4.1/docs/guide/util/logging/index.html         |    0 |       6
          1 | /j2se/1.4.1/docs/guide/beans/index.html                |    0 |       7
          1 | /j2se/1.4.1/docs/relnotes/features.html                |    0 |       8
          1 | /j2se/1.4.1/docs/tooldocs/tools.html                   |    0 |       9
          1 | /j2se/1.4.1/docs/guide/jdbc/index.html                 |    0 |      10
          1 | /docs/books/jls/second_edition/html/j.title.doc.html   |    0 |      11
          1 | /j2se/1.4.1/docs/guide/2d/index.html                   |    0 |      12
          1 | /j2se/1.4.1/docs/guide/security/index.html             |    0 |      13
          1 | /j2se/1.4.1/docs/relnotes/demos.html                   |    0 |      14
         15 | /                                                      |    0 |      15
          1 | /j2se/1.4.1/docs/guide/jni/index.html                  |    0 |      16
          1 | /j2se/1.4.1/docs/guide/swing/index.html                |    0 |      17
          1 | /docs/windows_format.html                              |    0 |      18
          1 | /j2se/1.4.1/fixedbugs/index.html                       |    0 |      19
          1 | /j2se/1.4.1/docs/guide/jws/index.html                  |    0 |      20
          1 | /j2se/1.4.1/changes.html                               |    0 |      21
          1 | /j2se/1.4.1/docs/guide/imf/index.html                  |    0 |      22
          1 | /j2se/1.4.1/docs/guide/collections/index.html          |    0 |      23
          1 | /                                                      |    0 |      24
          1 | /docs/codeconv/index.html                              |    0 |      25
          1 | /j2se/1.4.1/download.html                              |    0 |      26
          1 | /j2se/1.4.1/docs/guide/standards/index.html            |    0 |      27
          1 | /j2se/1.4.1/docs/guide/sound/index.html                |    0 |      28
          1 | /products/jdk/faq.html                                 |    0 |      29
        ...
```

On remarque que l'attribut `date` est toujours à 0. Cela n'est pas normal, il s'agit d'un petit défaut dans le programme qui a permis d'extraire les données de la première base vers celle sur laquelle nous travaillons. Cependant, comme nous ne nous servirons jamais de cet attribut dans les requêtes que nous ferons plus tard, cela n'a pas grande importance.

On peut lire dans cet extrait que la première page (dans l'ordre des identificateurs) a comme URL :

http://java.sun.com/j2se/1.4.1/docs/guide/xml/index.html

#### *II.2.1.4. Les autres fichiers*

##### *II.2.1.4.1. Les fichiers divers*

Tous les fichiers trouvés sur un serveur qui ne sont ni du type page web, ni du type image sont stockés dans la table `Fichier` :

```
CREATE TABLE Fichier (
  path TEXT NOT NULL,
  nom TEXT NOT NULL,
  id_fichier SERIAL PRIMARY KEY,
```

```
  UNIQUE(path,nom)
);
```

Il y a 69 entrées dans cette table. En voici quelques unes :

```
moteur_web=> SELECT * FROM Fichier;
             path              |                nom                 | id_fichier
-------------------------------+------------------------------------+------------
 /docs/j2se1.4/                | serial-spec.pdf                    |          1
 /docs/j2se1.4/                | serial-spec.ps                     |          2
 /products/javabeans/glasgow/  | beancontext.ps                     |          3
 /products/javabeans/glasgow/  | beancontext.pdf                    |          4
 /docs/j2se1.4/                | programmer_guide.pdf               |          5
 /internet-drafts/             | draft-ietf-ldapext-locate-08.txt   |          6
 /docs/j2se/1.4/               | j2d-book.pdf                       |          7
 /docs/j2se/1.4/               | j2d-book.ps                        |          8
 /pub/docs/ptc/                | 99-12-03.pdf                       |          9
 /docs/codeconv/               | codeconv.zip                       |         10
 /docs/codeconv/               | CodeConventions.pdf                |         11
 /docs/codeconv/               | CodeConventions.ps                 |         12
 /docs/specs/                  | langspec-2.0.html.tar.Z            |         13
 /docs/specs/                  | langspec-1.0.html.zip              |         14
 /docs/specs/                  | langspec-1.0.html.tar.Z            |         15
 /docs/specs/                  | langspec-1.0.ps.zip                |         16
 /docs/specs/                  | langspec-2.0.pdf                   |         17
 /docs/specs/                  | langspec-1.0.pdf                   |         18
 /docs/specs/                  | langspec-2.0.html.zip              |         21
 /docs/specs/                  | langspec-1.0.ps.Z                  |         22
 /docs/j2se1.4/                | JPS_PDF.pdf                        |         23
 /docs/specs/                  | vmspec.html.tar.Z                  |         24
 /docs/specs/                  | vmspec.2nded.html.zip              |         25
 /docs/specs/                  | vmspec.2nded.html.tar.Z            |         26
 /docs/specs/                  | vmspec.html.zip                    |         27
 /docs/specs/                  | vmspec.2nded.html.tar.gz           |         28
 /docs/specs/                  | vmspec.html.tar.gz                 |         29
 /docs/j2se1.4/                | versioning.pdf                     |         30
 /docs/j2se1.4/                | versioning.ps                      |         31
 ...
```

Pour l'instant, il n'est pas possible de déterminer où se trouvent ces fichiers.

#### *II.2.1.4.2. Les documents textuels*

Parmi les fichiers ci-dessus certains ont une propriété particulière : ils contiennent du texte. Ces fichiers textuels sont stockés dans la table `Doc_txt` :

```
CREATE TABLE Doc_Txt (
  id_fichier INTEGER PRIMARY KEY,
  FOREIGN KEY  (id_fichier) REFERENCES Fichier
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient 50 entrées, en voici un extrait :

```
moteur_web=> SELECT * FROM Doc_txt;
 id_fichier
------------
          1
          2
          3
          4
          5
          6
          7
          8
          9
         11
```

```
          12
          17
          18
          23
          30
          31
          32
          33
          34
          35
          37
          39
          40
          41
          42
         ...
```

#### *II.2.1.4.3. Les protocoles*

Il existe plusieurs manières possibles pour se connecter à un serveur (et donc pour récupérer les fichiers qui s'y trouvent). Ces différentes façons de procéder s'appellent protocoles. Il est parfois possible (mais pas obligatoirement) de pouvoir accéder à un même fichier (donc sur un même serveur) par l'intermédiaire de plusieurs protocoles. Les protocoles sont stockés dans la table `Protocole` :

```
CREATE TABLE Protocole (
  prefixe TEXT NOT NULL UNIQUE,
  id_proto SERIAL PRIMARY KEY
);
```

Il y 2 entrées dans cette table :

```
moteur_web=> SELECT * FROM Protocole;
 prefixe | id_proto
---------+----------
 ftp://  |        1
 http:// |        2
```

#### *II.2.1.4.4. L'hébergement*

Les trois tables que nous venons de voir ne permettent pas de localiser un fichier sur Internet. Pour cela, il nous faut une table supplémentaire nous indiquant sur quel serveur un fichier donné est hébergé. C'est le rôle de la table `Heberge` :

```
CREATE TABLE Heberge (
  id_fichier INTEGER,
  id_proto INTEGER,
  id_serveur INTEGER,
  UNIQUE (id_fichier,id_proto,id_serveur),
  FOREIGN KEY  (id_fichier) REFERENCES Fichier
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_proto) REFERENCES Protocole
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_serveur) REFERENCES Serveur
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient au moins autant d'entrées que la table Fichier. En l'occurrence, elle en contient le même nombre c'est à dire 69. Voici un extrait :

```
moteur_web=> SELECT * FROM heberge;
 id_fichier | id_proto | id_serveur
------------+----------+------------
          1 |        1 |         97
          2 |        1 |         97
          3 |        2 |          1
```

```
          4 |         2 |           1
          5 |         1 |          97
          6 |         2 |         385
          7 |         1 |          97
          8 |         1 |          97
          9 |         1 |         475
         10 |         1 |         510
         11 |         1 |         510
         12 |         1 |         510
         13 |         1 |         510
         14 |         1 |         510
         15 |         1 |         510
         16 |         1 |         510
         17 |         1 |         510
         18 |         1 |         510
         19 |         2 |           1
         20 |         2 |           1
         21 |         1 |         510
         22 |         1 |         510
         23 |         1 |          97
         24 |         1 |         510
         25 |         1 |         510
         26 |         1 |         510
         27 |         1 |         510
         28 |         1 |         510
         29 |         1 |         510
         30 |         1 |          97
        ...
```

On peut lire sur cet extrait (première ligne) que le fichier /docs/j2se1.4/serial-spec.pdf est hébergé sur le serveur ftp.java.sun.com et que pour y accéder, il faut utiliser le protocole FTP. Autrement dit, son URL est ftp://ftp.java.sun.com/docs/j2se1.4/serial-spec.pdf.

#### *II.2.1.5. Le contenu textuel*

##### *II.2.1.5.1. Les mots*

Les documents textuels et les pages web peuvent contenir du texte. Les mots de ces textes sont stockés dans `Mot_txt` :

```
CREATE TABLE Mot_Txt (
  mot TEXT NOT NULL UNIQUE,
  id_txt SERIAL PRIMARY KEY
);
```

Cette table contient 968 entrées dont voici un extrait :

```
        mot          | id_txt
---------------------+--------
 introduction        |      1
 the                 |      2
 java                |      3
 programming         |      4
 language            |      5
 and                 |      6
 environment         |      7
 is                  |      8
 designed            |      9
 to                  |     10
 solve               |     11
 number              |     12
 of                  |     13
 problems            |     14
 in                  |     15
 modern              |     16
 practice            |     17
 started             |     18
 as                  |     19
```

```
 part                |     20
 larger              |     21
 project             |     22
 develop             |     23
 advanced            |     24
 software            |     25
 for                 |     26
 consumer            |     27
 electronics         |     28
 these               |     29
 ...
```

*II.2.1.5.2. Ce que contiennent les pages web*

Pour savoir quel texte se trouve sur une page web, il faut interroger la table `Txt_page` :

```
CREATE TABLE txt_page (
  id_txt INTEGER,
  id_page INTEGER,
  occurrence INTEGER DEFAULT 1,
  UNIQUE (id_page,id_txt),
  FOREIGN KEY  (id_page) REFERENCES Page
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_txt) REFERENCES Mot_txt
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette page contient 12531 entrées :

```
id_txt | id_page | occurrence
--------+---------+------------
    650 |       1 |          1
    242 |       1 |          1
    578 |       1 |          1
    359 |       1 |          1
     75 |       1 |          1
    277 |       1 |          1
    187 |       1 |          1
    841 |       1 |          1
    362 |       1 |          1
    238 |       1 |          1
    916 |       1 |          1
    827 |       1 |          1
    226 |       1 |          1
    736 |       1 |          1
    866 |       1 |          1
    268 |       1 |          1
    334 |       1 |          1
     74 |       1 |          1
    209 |       1 |          1
    565 |       1 |          1
    926 |       2 |          1
    591 |       2 |          1
     44 |       2 |          1
    863 |       2 |          1
    ...
```

*II.2.1.5.3. Ce que contiennent les documents textuels*

Pour savoir quel texte se trouve dans un document textuel, il faut interroger la table `Txt_doc` :

```
CREATE TABLE txt_doc (
  id_txt INTEGER,
  id_fichier INTEGER,
  occurrence INTEGER DEFAULT 1,
  UNIQUE (id_fichier,id_txt),
  FOREIGN KEY  (id_fichier) REFERENCES Doc_txt
    ON UPDATE CASCADE
    ON DELETE CASCADE,
```

```
  FOREIGN KEY  (id_txt) REFERENCES Mot_txt
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient 22571 entrées dont :

```
moteur_web=> SELECT * FROM Txt_page;
 id_txt | id_page | occurrence
--------+---------+------------
    650 |       1 |          1
    242 |       1 |          1
    578 |       1 |          1
    359 |       1 |          1
     75 |       1 |          1
    277 |       1 |          1
    187 |       1 |          1
    841 |       1 |          1
    362 |       1 |          1
    238 |       1 |          1
    916 |       1 |          1
    827 |       1 |          1
    226 |       1 |          1
    736 |       1 |          1
    866 |       1 |          1
    268 |       1 |          1
    334 |       1 |          1
     74 |       1 |          1
    209 |       1 |          1
    565 |       1 |          1
    926 |       2 |          1
    591 |       2 |          1
    ...
```

### II.2.2. Tables concernant les objets contenus dans une page web

#### *II.2.2.1. Les images*

Nous avons déjà parlé de la table `Img` qui permet de stocker toutes les images visibles sur une page web, c'est à dire en pratique celles qui se trouvent dans une balise `<IMG>`. Nous allons voir maintenant les tables qui permettent d'une part d'associer à une image un commentaire (ou une description) et d'autre part, de savoir dans quelle page elle se trouve.

Tout d'abord pour pouvoir associer un commentaire, il faut une table pour stocker les mots de celui-ci. C'est le rôle de `Mot_desc` :

```
CREATE TABLE Mot_Desc (
  mot TEXT NOT NULL UNIQUE,
  id_desc SERIAL PRIMARY KEY
);
```

Cette table contient 1486 entrées et voici un extrait :

```
moteur_web=> SELECT * FROM Mot_desc;
               mot                | id_desc
----------------------------------+---------
 including                        |     669
 incompatibilities                |     670
 in-depth                         |     671
 index                            |     672
 industry                         |     673
 inetaddress                      |     674
 inetaddresss                     |     675
 inetsocketaddress                |     676
 info                             |     677
 information                      |     678
 informed                         |     679
 inheritance                      |     680
```

```
 initial                          |     681
 initialization                   |     682
 initializers                     |     683
 initializing                     |     684
 innerclasses                     |     685
 input                            |     686
 inputstreamreader                |     687
 ins                              |     688
 install                          |     689
 installation                     |     690
 installshield                    |     691
 instance                         |     692
 instances                        |     693
 instruction                      |     694
 instructions                     |     695
 int                              |     696
 integral                         |     697
 intel                            |     698
 interaction                      |     699
 interceptors                     |     700
 interface                        |     701
 ...
```

**Rem :** Normalement la liste des mots de la table `Mot_desc` n'est pas classée par ordre alphabétique. Dans notre base de données, c'est pourtant le cas. Cela provient du fait que PostgreSQL réalise un classement par ordre croissant lorsqu'il fait une union et que justement cette table est le résultat de l'union des trois autres tables. C'est donc un effet involontaire de notre méthode de remplissage.

Pour savoir maintenant quel commentaire est associé à une image ou dans quelle page cette image se trouve, il faut faire appelle à la même table `App_Img` :

```
CREATE TABLE App_img (
  id_page INTEGER,
  id_img INTEGER,
  id_desc INTEGER,
  occurrence INTEGER DEFAULT 1,
  UNIQUE (id_page,id_img,id_desc),
  FOREIGN KEY  (id_page) REFERENCES Page
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_desc) REFERENCES Mot_desc
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_img) REFERENCES Img
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient 680 entrées dont :

```
moteur_web=> SELECT * FROM App_img;
 id_page | id_img | id_desc | occurrence
---------+--------+---------+------------
      87 |      1 |     452 |          1
      87 |      1 |     444 |          1
      87 |      1 |      41 |          1
      87 |      1 |       9 |          2
      87 |      1 |     728 |          1
      87 |      2 |     736 |          1
      87 |      3 |    1310 |          1
       2 |      2 |     736 |          1
       2 |      3 |    1310 |          1
       3 |      2 |     736 |          1
       3 |      3 |     899 |          1
       3 |      3 |    1310 |          1
       3 |      3 |     668 |          1
       6 |      2 |     736 |          1
       6 |      3 |    1310 |          1
```

```
        7 |      2 |     736 |          1
        7 |      3 |    1310 |          1
        1 |      2 |     736 |          1
        1 |      3 |    1310 |          1
        9 |      2 |     736 |          1
        9 |      3 |    1310 |          1
        4 |      4 |    1337 |          1
        4 |      4 |     736 |          1
        4 |      4 |    1014 |          1
       10 |      3 |    1310 |          1
        4 |      4 |     634 |          1
        4 |      5 |     899 |          1
        4 |      5 |    1310 |          1
        4 |      5 |     668 |          1
      ...
```

Nous verrons plus tard des exemples d'utilisation de cette table.

#### *II.2.2.2. Les adresses électroniques*

La table contenant les adresses électroniques est `Mail` :

```
CREATE TABLE Mail (
  mail TEXT NOT NULL UNIQUE,
  id_mail SERIAL PRIMARY KEY
);
```

Elle contient 34 entrées :

```
moteur_web=> SELECT * FROM Mail;
                    mail                     | id_mail
---------------------------------------------+---------
 mailto:java-io@java.sun.com                 |       1
 mailto:java-awt@java.sun.com                |       2
 mailto:java-beans@java.sun.com              |       3
 mailto:jdbc-odbc@sun.com                    |       4
 mailto:jdbc@sun.com                         |       5
 mailto:jls@java.sun.com                     |       6
 mailto:jdk-comments@java.sun.com            |       7
 mailto:java2d-comments@sun.com              |       8
 mailto:jni@java.sun.com                     |       9
 mailto:java-security@sun.com                |      10
 mailto:swing-feedback@java.sun.com          |      11
 mailto:collections-comments@java.sun.com    |      12
 mailto:java-intl@java.sun.com               |      13
 mailto:f.allimant@confluent.fr              |      14
 mailto:j2se-comments@sun.com                |      15
 mailto:javasound-comments@sun.com           |      16
 mailto:java-debugger@sun.com                |      17
 mailto:rmi-comments@java.sun.com            |      18
 mailto:java-print@sun.com                   |      19
 mailto:j2se-comments@eng.sun.com            |      20
 mailto:jre-comments@java.sun.com            |      21
 mailto:jndi@java.sun.com                    |      22
 mailto:pelegri@eng.sun.com                  |      23
 mailto:reflection-comments@sun.com          |      24
 mailto:java-gc-comments@eng.sun.com         |      25
 mailto:java-dnd@java.sun.com                |      26
 mailto:jsr015-comments@sun.com              |      27
 mailto:jvmpi@eng.sun.com                    |      28
 mailto:rosanna.lee@sun.com                  |      29
 mailto:copyrights@sun.com                   |      30
 mailto:rmi-iiop@sun.com                     |      31
 mailto:sdk-docl10nbugs@java.sun.com         |      32
 mailto:javaidl@sun.com                      |      33
 mailto:access@sun.com                       |      34
```

De même que pour les images, il est possible de savoir dans quelle(s) page(s) web une adresse électronique et à quel(s) mot(s) grâce à la table `App_mail` :

```
CREATE TABLE App_mail (
  id_page INTEGER,
  id_mail INTEGER,
  id_desc INTEGER,
  occurrence INTEGER DEFAULT 1,
  UNIQUE (id_page,id_mail,id_desc),
  FOREIGN KEY  (id_page) REFERENCES Page
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_desc) REFERENCES Mot_desc
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_mail) REFERENCES Mail
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient 144 entrées. En voici un extrait :

```
moteur_web=> SELECT * FROM App_mail;
 id_page | id_mail | id_desc | occurrence
---------+---------+---------+------------
       2 |       1 |     753 |          1
       2 |       1 |     309 |          1
       2 |       1 |    1310 |          1
       2 |       1 |     736 |          1
       3 |       2 |     739 |          1
       3 |       2 |     309 |          1
       3 |       2 |    1310 |          1
       3 |       2 |     736 |          1
       7 |       3 |     309 |          1
       7 |       3 |    1310 |          1
       7 |       3 |     736 |          1
       7 |       3 |     741 |          1
      10 |       4 |     309 |          1
      10 |       4 |    1310 |          1
      10 |       4 |     767 |          1
      10 |       5 |     309 |          1
      10 |       5 |    1310 |          1
      10 |       5 |     766 |          1
      11 |       6 |     309 |          1
      11 |       6 |    1310 |          1
      11 |       6 |     736 |          1
      11 |       6 |     775 |          1
      14 |       7 |     309 |          1
      14 |       7 |    1310 |          1
      14 |       7 |     736 |          1
      14 |       7 |     770 |          1
      12 |       8 |     309 |          1
      12 |       8 |    1310 |          1
      12 |       8 |     738 |          1
     ...
```

#### *II.2.2.3. Les liens hyper-texte*

Les hyper liens sont stockés dans la table `Href` :

```
CREATE TABLE Href (
  type_fic INTEGER NOT NULL, -- 1 pour designer un fichier, 2 pour une page web
  id_fic INTEGER NOT NULL,
  id_href SERIAL PRIMARY KEY,
  UNIQUE (type_fic,id_fic)
);
```

Les champs `type_fic` et `id_fic` correspondent aux attributs `type de fichier` et `fichier` de l'entité `Lien_hypertexte`. Nous avons vu au chapitre I.2.2.2 page 11 que ces attributs agissent comme des pointeurs, soit vers un fichier, soit vers une page web. Lors de l'insertion d'une nouvelle entrée dans cette table, il est donc important de vérifier que ce pointeur est valide, c'est à dire qu'il référence soit un fichier existant si `type_fic` = 1 soit une page web existante si `type_fic` = 2. C'est la raison pour laquelle nous avons mis en place le trigger `check_id_trig` :

```
CREATE FUNCTION check_id_proc() RETURNS opaque
AS '
  DECLARE
    reponse INTEGER;
  BEGIN
    --RAISE NOTICE \'    debut position_perso_proc\';
    IF NEW.type_fic = 1 THEN -- Les fichiers
      SELECT id_fichier INTO reponse FROM Fichier WHERE id_fichier = NEW.id_fic;
      IF reponse IS NULL THEN
        --RAISE NOTICE \'    fin check_id_proc: echec\';
        RETURN NULL;
      END IF;
    ELSE
      IF NEW.type_fic = 2 THEN -- Les pages web
        SELECT id_page INTO reponse FROM Page WHERE id_page = NEW.id_fic;
        IF reponse IS NULL THEN
          --RAISE NOTICE \'    fin check_id_proc: echec\';
          RETURN NULL;
        END IF;
      ELSE
        --RAISE NOTICE \'    fin check_id_proc: echec\';
        RETURN NULL;
      END IF;
    END IF;
    --RAISE NOTICE \'    fin check_id_proc: reussite\';
    RETURN NEW;
  END;
'
LANGUAGE 'plpgsql';

CREATE TRIGGER check_id_trig
  BEFORE INSERT ON Href
  FOR EACH ROW
    EXECUTE PROCEDURE check_id_proc();
```

Dans notre base de données, la table `Href` contient 1208 entrées, dont :

```
moteur_web=> SELECT * FROM Href;
 type_fic | id_fic | id_href
----------+--------+---------
        2 |    361 |     366
        2 |    362 |     367
        2 |    363 |     368
        2 |    364 |     369
        2 |    365 |     370
        2 |    366 |     371
        2 |    367 |     372
        2 |    368 |     373
        2 |    369 |     374
        2 |    370 |     375
        2 |    371 |     376
        2 |    372 |     377
        2 |    373 |     378
        2 |    374 |     379
        2 |    375 |     380
        2 |    376 |     381
        2 |    377 |     382
        2 |    378 |     383
        2 |    379 |     384
        1 |      6 |     385
        2 |    380 |     386
```

```
         2 |     381 |      387
         2 |     382 |      388
         2 |     383 |      389
         2 |     384 |      390
         2 |     385 |      391
         2 |     386 |      392
         2 |     387 |      393
       ...
```

On remarque que la très grande majorité des hyper liens pointent vers des pages web ce qui correspond bien à ce à quoi on pouvait s'attendre.

Enfin, comme pour les images et les adresses électroniques, il nous faut aussi une table `App_href` implémentant la relation `Contient` :

```
CREATE TABLE App_href (
  id_page INTEGER,
  id_href INTEGER,
  id_desc INTEGER,
  occurrence INTEGER DEFAULT 1,
  UNIQUE (id_page,id_href,id_desc),
  FOREIGN KEY  (id_page) REFERENCES Page
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_desc) REFERENCES Mot_desc
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY  (id_href) REFERENCES Href
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Cette table contient 8458 entrées. En voici un extrait :

```
moteur_web=> SELECT * FROM App_href;
 id_page | id_href | id_desc | occurrence
---------+---------+---------+------------
       87 |       1 |    1473 |          1
       87 |       2 |    1207 |          1
       87 |       2 |     967 |          1
       87 |       3 |    1363 |          1
       87 |       3 |    1455 |          1
       87 |       3 |     117 |          1
       87 |       4 |    1139 |          1
       87 |       4 |    1424 |          1
       87 |       4 |     318 |          1
       87 |       4 |    1070 |          1
       87 |       4 |    1460 |          1
       87 |       5 |     408 |          1
       87 |       5 |     540 |          1
       87 |       5 |     595 |          1
       87 |       5 |     616 |          1
       87 |       5 |     677 |          1
       87 |       5 |    1385 |          1
       87 |       5 |     810 |          1
       87 |       5 |     145 |          1
       87 |       5 |     850 |          1
       87 |       5 |    1360 |          1
       87 |       5 |    1192 |          1
       87 |       6 |     854 |          1
       87 |       7 |    1358 |          1
       87 |       7 |     324 |          1
       87 |       7 |     145 |          1
       87 |       7 |     740 |          1
       87 |       8 |     540 |          2
       87 |       8 |    1041 |          1
       87 |       8 |     941 |          2
     ...
```

# III. Les requêtes

Maintenant que la base de données est implémentée et remplie, voyons comment en extraire de informations pertinentes.

## III.1. Les formules de mots

Avant de voir en détail les requêtes possibles, il faut définir ce qu'on appelle des « formules de mots » (FdM). En effet, lorsqu'un utilisateur recherche des documents sur Internet, il veut pouvoir faire cette recherche non pas sur un mot mais sur une série de mots. De plus, ces mots doivent être reliés entre eux par trois types de connecteurs : ET, OU et SANS. Ces connecteurs correspondent en fait au opérations ensemblistes INTERSECTION, UNION et SOUSTRACTION. Donc toutes les requêtes qui sont valables pour un mot seul, le sont aussi pour une FdM puisqu'il suffit d'appliquer ces requêtes à tous les mots de la formule puis de réaliser les opérations ensemblistes correspondant aux connecteurs sur les résultats obtenus.

**Ex :** Pour rechercher tous les documents correspondant à la FdM mot1 OU mot2 SANS (mot3 ET mot4) il faut une requête du type :

```
(SELECT … FROM … WHERE mot = mot1
 UNION SELECT … FROM … WHERE mot = mot2
 EXCEPT (SELECT … FROM … WHERE mot = mot3
      INTERSECT SELECT … FROM … WHERE mot = mot4))
```

Dans la suite nous nous limiterons donc toujours à des exemples basés sur un seul mot.

De plus, une FdM peut contenir des mots non complets dans lesquels « * » remplace un groupe de lettres (éventuellement vide) et « . » remplace au plus une seule lettre. Le langage SQL permet aussi de réaliser ce type de recherche très facilement grâce à `LIKE` et respectivement au caractère `%` et `_`.

**Ex :** Pour rechercher tous les documents relatifs au mot deb*fin. il faut une requête du type :

```
SELECT … FROM … WHERE mot LIKE 'deb%fin_';
```

Dans la suite, nous nous contenterons généralement de recherches sur des mots entiers.

## III.2. Les requêtes simples

### III.2.1. Sans ordonnancement

#### *III.2.1.1. Requêtes pseudo atomiques*

Les requêtes dites pseudo atomiques sont des requêtes de base sur lesquelles les requêtes complètes (c'est à dire celles qui répondent aux questions couramment posées par les utilisateurs) s'appuient. Elles sont généralement assez simples mais elles ne sont pas non plus triviales dans le sens où elles font toutes intervenir plusieurs tables.

##### *III.2.1.1.1. Concernant les Fichiers divers*

###### *III.2.1.1.1.a. Fichiers / noms*

On peut se demander quels sont les fichiers ayant pour nom une FdM. Il existe plusieurs façons de procéder possibles pour répondre à cette question.

La façon la plus simple est d'utiliser une jointure naturelle (une jointure est dite naturelle si elle identifie les champs ayant le même nom dans les deux tables).

**Ex :** Recherchons tous les fichiers de type pdf :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier
                  NATURAL JOIN Heberge
             WHERE nom LIKE '%.pdf';
 prefixe |    url_serveur    |              path           |         nom
```

```
---------+-----------------+-----------------------------+--------------------
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | serial-spec.pdf
 http:// | java.sun.com     | /products/javabeans/glasgow/ | beancontext.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              |programmer_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/             | j2d-book.pdf
 ftp://  | ftp.omg.org      | /pub/docs/ptc/              | 99-12-03.pdf
 ftp://  | ftp.javasoft.com | /docs/codeconv/             | CodeConventions.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/                | langspec-2.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/                | langspec-1.0.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | versioning.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | networking.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | ipv6_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | rmi-spec-1.4.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | jndi.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | jndispi.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | jndiexecsumm.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4.1/            | developer_guide.pdf
 http:// | java.sun.com     | /products/java-media/2D/    | perf_graphics.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | dnd1.pdf
 http:// | java.sun.com     | /j2se/1.4.1/docs/guide/rmi-iiop/ | interop.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | imageio_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | j2d-book.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | rmi-spec-1.3.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/             | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | VolatileImage.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/              | beancontext.pdf
(26 rows)
```

La jointure naturelle permet donc de faire des requêtes très simples. Cependant, si on considère que notre SGBD n'a pas d'optimiseur de requêtes, c'est une opération très coûteuse. En effet, la quantité totale de données manipulées est gigantesque.

**Ex :** Regardons la taille de la jointure de l'exemple précédent sans optimisation de la recherche :

```
moteur_web=> SELECT count(*) FROM Serveur,Protocole,Fichier,Heberge;
 count
--------
 447534
(1 row)
```

Donc pour faire la jointure naturelle entre les tables `Serveur`, `Protocole`, `Fichier` et `Heberge`, le SGBD devra essayer plus de 400 000 combinaisons (pour arriver à une table de seulement 69 entrées) alors que notre base de données est toute petite.

Pour éviter tous ces calculs inutiles, on peut utiliser un ensemble de requêtes.

**Ex :** Table `Fichier` restreinte :

```
moteur_web=> CREATE TABLE A AS (SELECT * FROM Fichier WHERE nom LIKE '%.pdf');
```

Table `Heberge` restreinte :

```
moteur_web=> CREATE TABLE B AS
               (SELECT * FROM heberge WHERE id_fichier IN
                 (SELECT id_fichier FROM A));
```

Table `Serveur` restreinte :

```
moteur_web=> CREATE TABLE C AS
               (SELECT * FROM Serveur WHERE id_serveur IN
                 (SELECT id_serveur FROM B));
```

Table `Potocole` restreinte :

```
moteur_web=> CREATE TABLE D AS
               (SELECT * FROM Protocole WHERE id_proto IN
                 (SELECT id_proto FROM B));
```

Requête finale :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM A,B,C,D
             WHERE (A.id_fichier=B.id_fichier AND C.id_serveur=B.id_serveur
                    AND D.id_proto=B.id_proto);
 prefixe |   url_serveur    |              path           |         nom
---------+-----------------+-----------------------------+--------------------
```

```
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | serial-spec.pdf
 http:// | java.sun.com     | /products/javabeans/glasgow/ | beancontext.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                |programmer_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/               | j2d-book.pdf
 ftp://  | ftp.omg.org      | /pub/docs/ptc/                | 99-12-03.pdf
 ftp://  | ftp.javasoft.com | /docs/codeconv/               | CodeConventions.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/                  | langspec-2.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/                  | langspec-1.0.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | versioning.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | networking.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | ipv6_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | rmi-spec-1.4.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | jndi.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | jndispi.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | jndiexecsumm.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4.1/              | developer_guide.pdf
 http:// | java.sun.com     | /products/java-media/2D/      | perf_graphics.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | dnd1.pdf
 http:// | java.sun.com     | /j2se/1.4.1/docs/guide/rmi-iiop/ | interop.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | imageio_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | j2d-book.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | rmi-spec-1.3.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/               | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | VolatileImage.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/                | beancontext.pdf
(26 rows)
```

On retrouve bien le même résultat qu'avec la jointure naturelle. Regardons quelle est la taille des données manipulées.

```
moteur_web=> SELECT count(*) FROM A;
 count
-------
    26
moteur_web=> SELECT count(*) FROM B;
 count
-------
    26
moteur_web=> SELECT count(*) FROM C;
 count
-------
     4
moteur_web=> SELECT count(*) FROM D;
 count
-------
     2
moteur_web=> SELECT count(*) FROM A,B,C,D;
 count
-------
  5408
```

Donc on peut considérer que la taille des données manipulées est de 26+26+4+2+5408 = 5466 c'est à dire environ 82 fois moins qu'avec la jointure.

Décomposer les jointures en plusieurs requêtes permet donc de réduire considérablement la taille des données manipulées (si le SGBD ne fait pas d'optimisation). Cependant, cela oblige à créer des tables temporaires ce qui est parfaitement impossible dans le cas d'un moteur de recherche qui doit être capable de traiter un grand nombre de requêtes en parallèle. Ainsi, pour implémenter un moteur de recherche efficace, on voit qu'il ne suffit pas d'avoir une excellente base de données, il faut aussi disposer d'un SGBD puissant pour l'exploiter au mieux.

*III.2.1.1.1.b. Fichiers / références*

On peut aussi vouloir connaître tous les fichiers référencés par une FdM.

**Rem :** On appelle fichier référencé par un mot, un fichier tel qu'il existe au moins une page web dans laquelle on trouve un hyper lien associé à ce mot et pointant vers ce fichier.
Cela est également valable pour les adresses électroniques, les pages web et les images.

Cette requête est un peu plus compliquée que la précédente car elle fait intervenir plus de tables.

**Ex :** Quelles sont les fichiers référencés par le mot download ?

```
moteur_web=>
SELECT prefixe,url_serveur,path,nom
FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
     NATURAL JOIN (SELECT DISTINCT id_fic AS id_fichier
                   FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                   WHERE mot='download' AND type_fic=1) AS A;
 prefixe |   url_serveur    |          path          |           nom
---------+------------------+------------------------+-------------------------
 http://  | java.sun.com     | /docs/books/effective/ | effective.zip
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | codeconv.zip
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | CodeConventions.pdf
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | CodeConventions.ps
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-1.0.html.zip
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-1.0.html.tar.Z
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-1.0.ps.zip
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-2.0.pdf
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-1.0.pdf
 ftp://   | ftp.javasoft.com | /docs/specs/           | langspec-1.0.ps.Z
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.html.tar.Z
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.2nded.html.zip
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.2nded.html.tar.Z
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.html.zip
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.2nded.html.tar.gz
 ftp://   | ftp.javasoft.com | /docs/specs/           | vmspec.html.tar.gz
(16 rows)
```

**Ex :** Le même exemple en décomposant les jointures pour simuler l'optimiseur

La liste des identificateurs des mots de la FdM :

```
moteur_web=> CREATE TABLE A AS (SELECT id_desc FROM Mot_desc
                                WHERE mot='download');
```

La table `App_href` restreinte :

```
moteur_web=> CREATE TABLE B AS (SELECT id_href FROM App_href
                                WHERE (id_desc IN (A.id_desc)));
```

La liste des identificateurs des fichiers recherchés :

```
moteur_web=> CREATE TABLE C AS
               (SELECT id_fic FROM Href
                WHERE (type_fic=1 AND id_href IN (B.id_href)));
```

Table `Heberge` restreinte :

```
moteur_web=> CREATE TABLE D AS (SELECT * FROM heberge
                                WHERE id_fichier IN (C.id_fic)) ;
```

Table `Serveur` restreinte :

```
moteur_web=> CREATE TABLE E AS
               (SELECT * FROM Serveur WHERE id_serveur IN
                 (SELECT id_serveur FROM D));
```

Table `Potocole` restreinte :

```
moteur_web=> CREATE TABLE F AS
               (SELECT * FROM Protocole WHERE id_proto IN
                 (SELECT id_proto FROM D));
```

Table `Fichier` restreinte :

```
moteur_web=> CREATE TABLE G AS (SELECT * FROM Fichier
                                WHERE id_fichier IN (C.id_fic)) ;
```

Requête finale :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM D,E,F,G
             WHERE (G.id_fichier=D.id_fichier AND E.id_serveur=D.id_serveur
                    AND F.id_proto=D.id_proto);
prefixe |   url_serveur    |          path          |           nom
---------+------------------+------------------------+-------------------------
 http://  | java.sun.com     | /docs/books/effective/ | effective.zip
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | codeconv.zip
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | CodeConventions.pdf
 ftp://   | ftp.javasoft.com | /docs/codeconv/        | CodeConventions.ps
```

```
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-1.0.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-1.0.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-1.0.ps.zip
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-2.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-1.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/          | langspec-1.0.ps.Z
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.2nded.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.2nded.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.2nded.html.tar.gz
 ftp://  | ftp.javasoft.com | /docs/specs/          | vmspec.html.tar.gz
(16 rows)
```

**Rem :** La décomposition des jointures en ensembles de requêtes étant assez lourde et peu compréhensible, nous ferons l'hypothèse dans le reste de ce dossier que notre SGBD dispose d'un optimiseur de requêtes et nous emploierons des jointures lorsque cela sera nécessaire.

*III.2.1.1.2. Concernant les documents textuels*

Pour les documents textuels, il est possible de faire le même genre de requête que pour les fichiers divers plus une nouvelle.

*III.2.1.1.2.a. Fichiers / noms*

**Ex :** Pour savoir quels sont les documents textuels dont le nom commence par « langspec » :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier
                  NATURAL JOIN Heberge NATURAL JOIN Doc_txt
             WHERE nom LIKE 'langspec%';
 prefixe |   url_serveur    |     path       |       nom
---------+------------------+----------------+------------------
 ftp://  | ftp.javasoft.com | /docs/specs/   | langspec-2.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/   | langspec-1.0.pdf
(2 rows)
```

Une requête peut-être plus intuitive ou compréhensible serait :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier
                  NATURAL JOIN Heberge
             WHERE nom LIKE 'langspec%' AND id_fichier IN (Doc_txt.id_fichier);
```

D'un point de vue performance, il est impossible de dire laquelle de ces requêtes est la plus rapide sans connaître le fonctionnement de l'optimiseur de requêtes.

*III.2.1.1.2.b. Fichiers / références*

**Ex :** Pour savoir quels sont les documents textuels référencés par le mot download :

```
moteur_web=>
SELECT prefixe,url_serveur,path,nom
FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN
     Heberge NATURAL JOIN Doc_txt
WHERE id_fichier IN (SELECT DISTINCT id_fic
                     FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                     WHERE mot='download' AND type_fic=1);
 prefixe |   url_serveur    |       path       |        nom
---------+------------------+------------------+---------------------
 ftp://  | ftp.javasoft.com | /docs/codeconv/  | CodeConventions.pdf
 ftp://  | ftp.javasoft.com | /docs/codeconv/  | CodeConventions.ps
 ftp://  | ftp.javasoft.com | /docs/specs/     | langspec-2.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/     | langspec-1.0.pdf
(4 rows)
```

*III.2.1.1.2.c. Fichiers / texte*

Une requête particulièrement utile pour les utilisateurs est de savoir quels sont les documents qui contiennent un certain texte.

**Ex :** Pour connaître tous les documents contenant le mot java :

```
moteur_web=> SELECT prefixe,url_serveur,path,nom
             FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier
                  NATURAL JOIN Heberge
             WHERE id_fichier IN (SELECT id_fichier
```

```
                                  FROM Mot_txt NATURAL JOIN Txt_doc
                                  WHERE mot='java');
 prefixe |   url_serveur    |         path             |         nom
---------+------------------+--------------------------+-----------------------
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/           | programmer_guide.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/             | langspec-1.0.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/             | langspec-1.0.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/             | langspec-1.0.pdf
 http:// | java.sun.com     | /docs/books/jls/seco...  | langspec-2.0.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/             | vmmspec.2nded.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/             | vmspec.2nded.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/             | vmspec.2nded.html.tar.gz
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/           | networking.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/           | ipv6_guide.pdf
 http:// | java.sun.com     | /docs/books/chanlee/...  | readme.txt
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/           | jndispi.ps
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/          | PS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/           | VolatileImage.pdf
 ftp://  | ftp.javasoft.com | /docs/books/chanlee/ |JavaClassLibVol2Examples.zip
(15 rows)
```

*III.2.1.1.3. Concernant les pages web*

Pour les pages web, on peut faire le même genre de requêtes que pour les documents textuels, sauf la recherche sur le nom du fichier contenant la page car d'une part on ne le connait pas toujours et d'autre part, celui-ci n'est généralement pas intéressant.

*III.2.1.1.3.a. Pages / références*

**Ex :** Pour connaître toutes les pages référencées par le mot « interface » :

```
moteur_web=> SELECT url_serveur,url_page
             FROM Serveur NATURAL JOIN Page
             WHERE id_page IN
               (SELECT id_fic
                FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                WHERE (mot='interface' AND type_fic=2));
 url_serveur  |                          url_page
--------------+-------------------------------------------------------------
 java.sun.com | /j2se/1.4.1/docs/guide/jni/index.html
 java.sun.com | /j2se/1.4.1/docs/guide/jndi/index.html
 java.sun.com | /j2se/1.4.1/docs/guide/jvmpi/index.html
 java.sun.com | /docs/books/tutorial/ui
 java.sun.com | /j2se/1.4.1/guide/jni/index.html
 java.sun.com | /j2se/1.4.1
 java.sun.com | /j2se/1.4.1/guide/jvmpi/index.html
 java.sun.com | /j2se/1.4.1/guide/jndi/index.html
 java.sun.com | /j2se/1.4.1/docs/guide/jpda/jvmdi-spec.html
 java.sun.com | /j2se/1.4.1/docs/guide/jpda/architecture.html
 java.sun.com | /products/jdk/faq/jnifaq.html
 java.sun.com | /docs/books/tutorial/native1.1/index.html
 java.sun.com | /j2se/1.4.1/docs/guide/awt/1.3/AWT_Native_Interface.html
 java.sun.com | /j2se/1.4.1/docs/relnotes
 java.sun.com | /j2se/1.4.1/docs/tooldocs/javadoc/whatsnew-1.4.1.html
 java.sun.com | /docs/books/vmspec/2nd-edition/html/Concepts.doc.html
 java.sun.com | /docs/books/vmspec/2nd-edition/html/ClassFile.doc.html
 java.sun.com | /j2se/1.4.1/nio/index.html
 java.sun.com | /j2se/1.4.1/docs/guide/idl/jidlDSI.html
(19 rows)
```

De la première ligne par exemple, on peut savoir qu'il existe au moins une page dans laquelle un hyper lien associé au mot « interface » pointe sur la page http://java.sun.com/j2se/1.4.1/docs/guide/jni/index.html

*III.2.1.1.3.b. Pages / texte*

**Ex :** Pour connaître toutes les pages contenant le mot java :

```
moteur_web=> SELECT url_serveur,url_page
             FROM Serveur NATURAL JOIN Page
             WHERE id_page IN (SELECT id_page
                               FROM Txt_page NATURAL JOIN Mot_txt
```

```
                                WHERE mot='java');
     url_serveur      |                              url_page
----------------------+-----------------------------------------------------------
 java.sun.com         | /j2se/1.4.1/docs/guide/awt/index.html
 java.sun.com         | /j2se/1.4.1/docs/guide/reflection/index.html
 java.sun.com         | /j2se/1.4.1/docs/guide/awt/AWTChanges.html
 java.sun.com         | /j2se/1.4.1/docs/guide/serialization/relnotes14.html
 servlet.java.sun.com | /help/legal_and_licensing
 java.sun.com         | /blueprints
 java.sun.com         | /j2se/1.4.1/guide/jws/index.html
 java.sun.com         | /j2se/1.4.1/guide/jps/index.html
 java.sun.com         |/j2se/1.4.1/docs/api/javax/security/auth/login/...
 docs.sun.com         | /?q=kadmin&p=/doc/816-0211/6m6nc66th&a=view
 www.sun.com          | /downloads
 servlet.java.sun.com | /developer.java.sun.com/developer/earlyAccess
 java.sun.com         | /j2se/1.4.1/docs/guide/jndi
 java.sun.com      | /j2se/1.4.1/docs/api/javax/imageio/spi/package-summary.html
 java.sun.com         | /j2se/1.4.1/docs/api/org/xml/sax/package-summary.html
 java.sun.com         | /docs/books/chanlee/second_edition/vol1/index.html
 java.sun.com         | /docs/books/tutorial/information/copyright.html
 java.sun.com         | /j2se/1.4/docs/guide/plugin/index.html
 java.sun.com         | /j2se/1.4.1/docs/api/java/net/DatagramSocket.html
 java.sun.com         | /j2se/1.4.1/docs/guide/idl/servantactivator.html
 java.sun.com         | /j2se/1.4/docs/index.html
 java.sun.com         | /support
(22 rows)
```

*III.2.1.1.4. Concernant les images*

Pour les images, on peut faire le même type de requête que pour les fichiers divers.

*III.2.1.1.4.a. Images / noms*

**Ex :** Recherchons toutes les images de type jpg :

```
moteur_web=> SELECT url_serveur,path,nom
             FROM Serveur NATURAL JOIN Img
             WHERE nom LIKE '%.jpg';
 url_serveur  |        path         |           nom
--------------+---------------------+-------------------------
 www.sun.com  | /2003-0422/images/  | b3_nascar-java.jpg
 java.sun.com | /images/            | j2se_homepage_button.jpg
 java.sun.com | /images/            | j2ee_homepage_button.jpg
 java.sun.com | /images/            | j2me_homepage_button.jpg
(4 rows)
```

*III.2.1.1.4.b. Images / références*

**Ex :** Pour connaître toutes les pages référencées par le mot « interface » :

```
moteur_web=> SELECT url_serveur,path,nom
             FROM Serveur NATURAL JOIN Img NATURAL JOIN App_img
                  NATURAL JOIN Mot_desc
             WHERE mot='sun');
      url_serveur       |           path            |              nom
------------------------+---------------------------+---------------------------
 developer.java.sun.com | /images/                  | v3_sun_logo.gif
 java.sun.com           | /images/                  | smi.logo.gif
 java.sun.com           | /images/                  | v3_sun_logo.gif
 java.sun.com           | /images/buttons/          | lgsun.gif
 java.sun.com           | /j2se/1.4.1/docs/images/  | sunlogo64x30.gif
 www.sun.com            | /im/                      | sun_logo.gif
 www.sun.com            | /pics/promos/             | b5_nc_03_q2.gif
 www.sun.com            | /pics/promos/        | b5_sun_fire_blade_platform.gif
(8 rows)
```

*III.2.1.1.5. Concernant les adresses électroniques*

Pour les adresses électroniques, nous ne considérerons qu'un seul type de requête : par rapport aux références.

**Ex :** Toutes les adresses électroniques référencées par « rosanna » :

```
moteur_web=> SELECT DISTINCT mail
             FROM Mail NATURAL JOIN App_mail NATURAL JOIN Mot_desc
             WHERE mot='rosanna';
            mail
----------------------------
 mailto:rosanna.lee@sun.com
(1 row)
```

#### *III.2.1.2. Requêtes complètes*

A partir des précédentes requêtes, il est possible de répondre à n'importe quelles questions couramment posées par les utilisateurs. Le principe est de faire des unions entre le résultat de chaque requête.

**Ex :** Quels sont tous les fichiers (documents textuels inclus) relatif au mot java :

```
moteur_web=>
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
 WHERE id_fichier IN (SELECT id_fichier
                      FROM Mot_txt NATURAL JOIN Txt_doc
                      WHERE mot='java'))
UNION
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN
      Heberge NATURAL JOIN Doc_txt
 WHERE id_fichier IN (SELECT DISTINCT id_fic
                      FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                      WHERE mot='java' AND type_fic=1))
UNION
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN
      Heberge NATURAL JOIN Doc_txt
 WHERE nom='java.%')
UNION
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
 WHERE id_fichier IN (SELECT DISTINCT id_fic
                      FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                      WHERE mot='java' AND type_fic=1))
UNION
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
 WHERE nom='java.%');
 prefixe |   url_serveur    |        path          |          nom
---------+------------------+----------------------+---------------------------
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/      | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | VolatileImage.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | ipv6_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | jndispi.ps
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | networking.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | programmer_guide.pdf
 ftp://  | ftp.javasoft.com | /docs/books/chanlee/ | JavaClassL....zip
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.tar.gz
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.zip
 http:// | java.sun.com     | /docs/books/chanl... | readme.txt
 http:// | java.sun.com     | /docs/books/jls/...  | langspec-2.0.html.tar.Z
 http:// | www.ietf.org     | /internet-drafts/    | draft-ietf-ldapext-....txt
 http:// | www.ietf.org     | /rfc/                | rfc2246.txt
(17 rows)
```

Cependant, on voit clairement sur l'exemple précédent que la recherche n'est pas optimale car certaines opérations sont répétées plusieurs fois. Tout en gardant la même idée de réutiliser les requêtes pseudo atomiques, essayons d'éviter les répétitions.

**Ex :** Quels sont tous les fichiers (documents textuels inclus) relatif au mot java :

```
moteur_web=>
SELECT prefixe,url_serveur,path,nom
FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
WHERE id_fichier IN ((SELECT id_fichier
                      FROM Mot_txt NATURAL JOIN Txt_doc
                      WHERE mot='java')
                     UNION
                     (SELECT DISTINCT id_fic AS id_fichier
                      FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                      WHERE mot='java' AND type_fic=1))
      OR nom='java.%';
prefixe |   url_serveur    |        path          |          nom
---------+------------------+----------------------+---------------------------
 ftp://  | ftp.java.sun.com | /docs/j2se/1.4/      | JPS_PDF.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | VolatileImage.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | ipv6_guide.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | jndispi.ps
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | networking.pdf
 ftp://  | ftp.java.sun.com | /docs/j2se1.4/       | programmer_guide.pdf
 ftp://  | ftp.javasoft.com | /docs/books/chanlee/ | JavaClassL....zip
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.html.zip
 ftp://  | ftp.javasoft.com | /docs/specs/         | langspec-1.0.pdf
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.tar.Z
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.tar.gz
 ftp://  | ftp.javasoft.com | /docs/specs/         | vmspec.2nded.html.zip
 http:// | java.sun.com     | /docs/books/chanl... | readme.txt
 http:// | java.sun.com     | /docs/books/jls/...  | langspec-2.0.html.tar.Z
 http:// | www.ietf.org     | /internet-drafts/    | draft-ietf-ldapext-....txt
 http:// | www.ietf.org     | /rfc/                | rfc2246.txt
(17 rows)
```

Cette fois-ci, toutes les répétitions ont été évitées. Cependant, il ne faut pas perdre de vue que cette requête doit être générée automatiquement par un programme à partir de la question d'un utilisateur. Or celle-ci est assez difficile à obtenir. Sans rentrer dans les détails de l'implémentation d'un programme de transformation des requêtes utilisateur vers les requêtes SQL, voici plutôt la requête pour répondre à la question initiale :

```
moteur_web=>
SELECT prefixe,url_serveur,path,nom
FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
WHERE id_fichier IN ((SELECT id_fichier
                      FROM Mot_txt NATURAL JOIN Txt_doc
                      WHERE mot='java')
                     UNION
                     (SELECT DISTINCT id_fic AS id_fichier
                      FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                      WHERE mot='java' AND type_fic=1)
                     UNION
                     (SELECT id_fichier FROM Fichier WHERE nom='java.%'));
```

L'idée est de ne transcrire l'identificateur d'un fichier en URL (`prefixe,url_serveur,path,nom`) qu'au dernier moment.

Il est également possible de faire des recherches à la fois sur des fichiers et des pages web en comblant les champs manquants de la table `Page` par rapport à `Fichier`.

**Ex :** Tous les fichiers, documents textuels et pages web relatifs au mot java :

```
moteur_web=>
(SELECT prefixe,url_serveur,path,nom
 FROM Serveur NATURAL JOIN Protocole NATURAL JOIN Fichier NATURAL JOIN Heberge
 WHERE id_fichier IN ((SELECT id_fichier
                       FROM Mot_txt NATURAL JOIN Txt_doc
```

```
                        WHERE mot='java')
                       UNION
                       (SELECT DISTINCT id_fic AS id_fichier
                        FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                        WHERE mot='java' AND type_fic=1)
                       UNION
                       (SELECT id_fichier FROM Fichier WHERE nom='java.%')))
UNION
(SELECT 'http' AS prefixe, url_serveur, url_page AS path, '' AS nom
 FROM Serveur NATURAL JOIN Page
 WHERE id_page IN ((SELECT id_page
                    FROM Mot_txt NATURAL JOIN Txt_page
                    WHERE mot='java')
                   UNION
                   (SELECT DISTINCT id_fic AS id_page
                    FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                    WHERE mot='java' AND type_fic=2)));
```

Par soucis d'économie de place, nous ne mettons pas les 338 réponses.

### III.2.2. Avec ordonnancement

Dans tous les exemples donnés jusqu'ici l'ordre du résultat des requêtes était parfaitement quelconque. Cependant, dans la première partie nous avions fixé comme contrainte que les réponses du moteur de recherche soient pertinentes. Essayons de les rendre telles en ordonnant les réponses.

Pour cela, on considère que plus un objet a de référence correspondant à la FdM recherchée, plus il est pertinent. En effet, un objet qui est référencé dans 20 pages différentes est probablement plus pertinent qu'un autre qui n'a été trouvé que dans une seule page. Ensuite, s'il y a des objets ex æquo, on les départage grâce à l'attribut `occurrence` des relations `Contient` : plus celui-ci est grand, plus l'objet est bien placé.

**Ex :** Recherchons toutes les pages web relatives à interface **?** OU java.
Afin de mettre en évidence les différentes étapes pour afficher une réponse ordonnée, décomposons la requête en plusieurs sous-requêtes.
Première étape, la collecte de toutes les informations nécessaires :

```
moteur_web=> CREATE TABLE A AS
               (SELECT occurrence,id_fic AS id_page
                FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                WHERE mot LIKE 'interface_' AND type_fic=2)
               UNION ALL
               (SELECT occurrence,id_fic AS id_page
                FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
                WHERE mot='java' AND type_fic=2)
               UNION ALL
               (SELECT occurrence,id_page
                FROM Txt_page NATURAL JOIN Mot_txt
                WHERE mot LIKE 'interface_')
               UNION ALL
               (SELECT occurrence,id_page
                FROM Txt_page NATURAL JOIN Mot_txt
                WHERE mot='java');
```

Deuxième étape, le pré calcul de l'ordre :

```
moteur_web=> CREATE TABLE B AS SELECT count(*),sum(occurrence),id_page
                               FROM A GROUP BY id_page;
```

Le `GROUP BY` permet de compter combien de fois une page apparaît dans la table `A` (ce qui correspond à peu près au nombre de fois qu'elle est référencée) et également de faire la somme de ses occurrences.
Troisième étape, l'affichage de la réponse ordonnée :

```
moteur_web=> SELECT url_serveur,url_page
             FROM B NATURAL JOIN Page NATURAL JOIN Serveur
             ORDER BY count DESC, sum DESC ;
```

Grâce au pré calcul, l'ordre est obtenu par un simple `ORDER BY`. On obtient :

```
      url_serveur       |                          url_page
------------------------+-------------------------------------------------------
```

```
 developer.java.sun.com | /developer/onlineTraining
 java.sun.com           | /j2se/1.4.1/docs/guide/jndi/index.html
 java.sun.com           | /j2se/1.4.1/docs/guide/awt/index.html
 java.sun.com           | /j2se/1.4.1/docs/api/index.html
 java.sun.com           | /j2se/1.4.1/docs/guide/reflection/index.html
 java.sun.com           | /j2se/1.4.1/docs/guide/awt/AWTChanges.html
 java.sun.com           | /j2se/1.4.1/docs/guide/serialization/relnotes14.html
 java.sun.com         | /j2se/1.4.1/docs/guide/beans/spec/beancontextTOC.fm.html
 java.sun.com           | /j2se/1.4.1/docs/tooldocs/windows/extcheck.html
 java.sun.com           | /xml/tutorial_intro.html
 servlet.java.sun.com   | /help/legal_and_licensing
 ...
(343 rows)
```

## III.3. Les requêtes avancées et limites du modèle relationnel

### III.3.1. Exploitation du graphe d'Internet

Nous avons dit que la structure de la base de données permet de reconstituer la modélisation sous forme de graphe d'Internet. Cependant, nous n'avons pas encore exploité cette propriété. Les requêtes que l'on peut poser en se basant sur le graphe d'Internet ne sont en effet pas très courantes ; elles sont néanmoins intéressantes et peuvent être utiles. Ces requêtes se servent toutes de la relation mère / fille induite par l'orientation du graphe des pages web.

**Ex :** lorsqu'on fait une recherche sur un moteur, il peut être intéressant de connaître les pages mères des pages trouvées. En effet, si on découvre un document particulièrement intéressant, il est probable que les pages pointant sur ce document le soit aussi.

```
moteur_web=> SELECT DISTINCT url_serveur,url_page,id_page
             FROM Serveur NATURAL JOIN Page
             WHERE id_page IN (SELECT id_page
                               FROM Href NATURAL JOIN App_href
                               WHERE id_fic=165 AND type_fic=2);
 url_serveur  |             url_page              | id_page
--------------+-----------------------------------+---------
 java.sun.com | /j2se/1.4.1/README.html           |      79
 java.sun.com | /j2se/1.4.1/changes.html          |      21
 java.sun.com | /j2se/1.4.1/compatibility.html    |       4
 java.sun.com | /j2se/1.4.1/docs                  |       5
 java.sun.com | /j2se/1.4.1/install-linux.html    |      75
 java.sun.com | /j2se/1.4.1/install-solaris.html  |      67
 java.sun.com | /j2se/1.4.1/install-windows.html  |      80
 java.sun.com | /j2se/1.4.1/relnotes.html         |      62
 java.sun.com | /j2se/1.4.1/search.html           |      76
(9 rows)
```

### III.3.2. Les limites

Dès qu'on travaille sur le graphe d'Internet, on se rapproche dangereusement de la limite du modèle relationnel, à savoir sa non complétude.

**Ex :** Imaginons qu'on veuille faire du contrôle parental au niveau du moteur de recherche. L'une des règles de ce contrôle est d'interdire l'accès aux pages relatives à Windows® par exemple (tout le monde sait que ces pages sont fortement déconseillées aux enfants !!!). Pour cela, on peut rechercher toutes les pages relatives au mot windows et les mettre dans la liste des pages interdites, c'est à dire que quelque soit la recherche demandée par l'enfant, ces pages n'apparaîtront jamais dans la réponse. Cette mesure est déjà efficace mais l'enfant peut la contourner en recherchant une page qui contienne un lien vers une page relative à Windows® (par exemple il sait que toutes pages contenant le mot Excel pointent directement sur le site de Windows®). Il nous faut donc aussi interdire toutes les pages pointant vers au moins une page relative au mot windows.
Recherche des pages directement relatives à Windows® :

```
moteur_web=> CREATE TABLE A AS
               (SELECT id_fic AS id_page
                FROM Href NATURAL JOIN App_href NATURAL JOIN Mot_desc
```

```
                 WHERE mot='windows' AND type_fic=1)
                UNION
                (SELECT id_page
                 FROM Txt_page NATURAL JOIN Mot_txt
                 WHERE mot='windows');
```

Recherche des pages mères (et union avec les pages filles) :

```
moteur_web=> CREATE TABLE B AS
                 (SELECT DISTINCT id_page
                  FROM Href NATURAL JOIN App_href NATURAL JOIN Page
                  WHERE (id_fic IN (A.id_page) AND type_fic=2))
                 UNION (SELECT * FROM A);
```

Affichage de toute les pages interdites :

```
moteur_web=> SELECT url_serveur,url_page
             FROM B NATURAL JOIN Page NATURAL JOIN Serveur ;
      url_serveur       |                  url_page
------------------------+-------------------------------------------------------
 java.sun.com           | /j2se/1.4.1/compatibility.html
 java.sun.com           | /j2se/1.4.1/docs
 java.sun.com           | /j2se/1.4.1/docs/relnotes/features.html
 java.sun.com           | /j2se/1.4.1/docs/tooldocs/tools.html
 java.sun.com           | /j2se/1.4.1/docs/relnotes/demos.html
 java.sun.com           | /j2se/1.4.1/docs/guide/swing/index.html
 java.sun.com           | /docs/windows_format.html
 java.sun.com           | /j2se/1.4.1/fixedbugs/index.html
 java.sun.com           | /j2se/1.4.1/changes.html
 java.sun.com           | /
 java.sun.com           | /docs/codeconv/index.html
 java.sun.com           | /j2se/1.4.1/download.html
 java.sun.com           | /products/jdk/faq.html
 java.sun.com           | /j2se/1.4.1/docs/relnotes/contacts.html
 java.sun.com           | /j2se/1.4.1/docs/api/overview-summary.html
 java.sun.com           | /docs/books/effective/index.html
 developer.java.sun.com | /developer/onlineTraining/new2java/index.html
 java.sun.com           | /docs/books/jls/index.html
 java.sun.com           | /docs/books/vmspec/index.html
 java.sun.com           | /j2se/1.4.1/docs/guide/vm/index.html
 java.sun.com           | /j2se/1.4.1/docs/guide/reflection/index.html
 java.sun.com           | /docs/books/tutorial/index.html
 java.sun.com           | /j2se/1.4.1/docchanges.html
 developer.java.sun.com | /developer/onlineTraining
 java.sun.com           | /docs/books/chanlee
 java.sun.com           | /features/2001/09/goslingrev.html
 ...
(51 rows)
```

Nous avons donc pu améliorer le contrôle parental en recherchant les pages mères. Cependant, on voit tout de suite que ce n'est pas une solution parfaite puisqu'on peut faire le même raisonnement sur les pages mères que celui que l'enfant avait fait pour les pages filles. Il faut donc rechercher aussi les pages mères des pages mères etc. pour établir un contrôle parental imparable. Malheureusement, la recherche de toutes les ancêtres d'une page est impossible dans le modèle relationnel.

# Base de données déductive

Maintenant que nous avons vu en détail ce qu'il est possible de faire dans le modèle relationnel avec notre base de données, il est intéressant de regarder d'une part si tout peut être traduit dans le modèle déductif, d'autre part s'il est possible d'en faire encore plus.

Pour répondre à cela, nous allons transformer notre base de données en programme Prolog, grâce auquel nous travaillerons dans le modèle déductif.

## I. Constitution du programme Prolog

### I.1. Les faits

Les faits ont été générés automatiquement par un programme[9]. En effet, ils correspondent exactement aux données de la base que nous venons de voir :

- chaque table a donné son nom à un prédicat ;
- chaque entrée d'une table a été traduite en un fait dont
  - le nom de prédicat est celui de la table à laquelle l'entrée appartient,
  - les arguments sont les valeurs des champs de la table pour cette entrée.

Voici un extrait du programme, où on peut remarquer par exemple que la table `Serveur` qui contient 47 entrées a bien été traduite en 47 faits :

```
/*************************************************************************
 *   COMPOSITION DES PREDICATS:
 * serveur(url_serveur,id_serveur).
 * img(id_serveur,path,nom,id_img).
 * page(id_serveur,url_page,date,id_page).
 * fichier(path,nom,id_fichier).
 * doc_txt(id_fichier).
 * protocole(prefixe,id_proto).
 * heberge(id_fichier,id_proto,id_serveur).
 * mot_txt(mot,id_txt).
 * txt_doc(id_txt,id_fichier,occurrence).
 * txt_page(id_txt,id_page,occurrence).
 * mot_desc(mot,id_desc).
 * href(type_fic,id_fic,id_href).
 * app_href(id_page,id_href,id_desc,occurrence).
 * mail(mail,id_mail).
 * app_mail(id_page,id_mail,id_desc,occurrence).
 * app_img(id_page,id_img,id_desc,occurrence).
 ************************************************************************/

/**
 * Table serveur
 */

serveur('java.sun.com',1).
serveur('www.sun.com',15).
```

[9] Le fichier source sqlTopl2.c est joint en annexe sur le CD-ROM dans le répertoire Annexes/ Convertisseur/.

```
serveur('developer.java.sun.com',39).
serveur('ftp.java.sun.com',97).
serveur('docs.sun.com',118).
serveur('www.w3.org',128).
serveur('servlet.java.sun.com',149).
serveur('wwws.sun.com',150).
serveur('search.java.sun.com',170).
serveur('www201.ikiosk.com',185).
serveur('www.jcp.org',190).
serveur('jcp.org',379).
serveur('www.ietf.org',385).
serveur('ftp.omg.org',475).
serveur('ftp.javasoft.com',510).
serveur('wireless.java.sun.com',513).
serveur('forum.java.sun.com',514).
serveur('research.sun.com',553).
serveur('www.confluent.fr',555).
serveur('www.winzip.com',559).
serveur('home.mcom.com',566).
serveur('developer.netscape.com',578).
serveur('home.netscape.com',579).
serveur('www.amazon.com',599).
serveur('cseng.aw.com',602).
serveur('www.awl.com',605).
serveur('12.31.179.2',615).
serveur('developer.apple.com',664).
serveur('cseng.awl.com',679).
serveur('archives.java.sun.com',712).
serveur('vig.pearsoned.com',791).
serveur('cgi.omg.org',866).
serveur('suned.sun.com',923).
serveur('www.informit.com',951).
serveur('www.digitalguru.com',963).
serveur('sunonedev.sun.com',999).
serveur('www.corba.org',1008).
serveur('www.nvidia.com',1031).
serveur('www.research.avayalabs.com',1037).
serveur('www.atitech.com',1057).
serveur('www.omg.org',1084).
serveur('www.jguru.com',1086).
serveur('sunsolve.sun.com',1116).
serveur('www.adobe.com',1166).
serveur('support.installshield.com',1199).
serveur('www.microsoft.com',1200).
serveur('support.microsoft.com',1201).
/* (47 entrees) */


/**
 * Table img
 */

img(1,'/j2se/1.4.1/docs/images/','download.arrow.gif',1).
img(1,'/j2se/1.4.1/docs/images/','javalogo52x88.gif',2).
img(1,'/j2se/1.4.1/docs/images/','sunlogo64x30.gif',3).
img(1,'/images/','v4_java_sun_com.gif',4).
...
```

A partir de ces faits, il est possible de simuler toutes les requêtes que nous avons vues dans le modèle relationnel.

### I.2. Les règles

Nous avions vu dans la partie précédente (page 38) que le modèle relationnel ne permet pas de répondre à toutes les questions des utilisateurs. Afin de prouver la supériorité du modèle déductif, nous introduisons quelques règles avec lesquelles nous pourrons y répondre.

```
/*************************************************************************
 * Les règles
```

```
 ***************************************************************************/

% mere(page_mere,page_fille)
mere(M,F):-app_href(M,Id_href,_,_),href(2,F,Id_href).

% filles(+page,-ensemble_de_ces_filles)
filles(Page,Set):-
    findall(Y,mere(Page,Y),L),
    list_to_set(L,Set).

% descendant(+page_ancetre,-page_descendante)
descendant(Start,D) :-
    filles(Start,Filles),               % Filles={pages a visiter}
    merge_set([Start], Filles, Closed), % Closed={pages vues (avec 1 iteration d'avance)}
    descendant_iter(Filles,Closed,D).
% descendant_iter(liste_pages_a_visiter,liste_pages_vues)
descendant_iter([X|_],_,X).
descendant_iter([X|Open1],Closed,Y) :-
    filles(X,Filles),
    subtract(Filles,Closed,Jamais_vues), % Jamais_vues={pages non encore visitees}
    append(Jamais_vues,Open1,Open2),     % Open2={pages a visiter ds la prochaine iteration}
    merge_set(Closed,Filles,Closed1),    % Closed1={pages vues (avec 1 iteration d'avance)}
    descendant_iter(Open2,Closed1,Y).

% meres(+page,-ensemble_de_ces_meres)
meres(Page,Set):-
    findall(Y,mere(Y,Page),L),
    list_to_set(L,Set).

% ancetre(+page_ancetre,-page_ancetre)
ancetre(Start,A) :-
    meres(Start,Meres),                % Meres={pages a visiter}
    merge_set([Start], Meres, Closed), % Closed={pages vues (avec 1 iteration d'avance)}
    ancetre_iter(Meres,Closed,A).
% ancetre_iter(liste_pages_a_visiter,liste_pages_vues)
ancetre_iter([X|_],_,X).
ancetre_iter([X|Open1],Closed,Y) :-
    meres(X,Meres),
    subtract(Meres,Closed,Jamais_vues),  % Jamais_vues={pages non encore visitees}
    append(Jamais_vues,Open1,Open2),     % Open2={pages a visiter dans la prochaine
iteration}
    merge_set(Closed,Meres,Closed1),     % Closed1={pages vues (avec 1 iteration d'avance)}
    ancetre_iter(Open2,Closed1,Y).
```

Ces règles traduisent la relation mère / fille que toutes les pages ont entre elles dans la graphe orienté d'Internet.

**Ex :** Quelles sont les pages directement accessibles depuis la page d'identificateur 1 ?

```
?- mere(1,X).
X = 87 ;
X = 87 ;
X = 120 ;
X = 120 ;
X = 121 ;
X = 121 ;
X = 121 ;
X = 121 ;
X = 121 ;
X = 121 ;
X = 121 ;
X = 124 ;
X = 125 ;
X = 125 ;
X = 125 ;
X = 125 ;
X = 125 ;
X = 125 ;
X = 126 ;
X = 126 ;
X = 126 ;
X = 35 ;
```

```
X = 15 ;
X = 15 ;
X = 15 ;
No
```

On remarque qu'il y a des répétitions. Cela est normal car une page peut être accessible depuis une autre par l'intermédiaire de plusieurs hyper liens et que pour un seul hyper lien il y a autant d'entrées dans la table App_href qu'il y a de mots associé à ce lien. Nous avions besoin de cette caractéristique en SQL pour ordonner les réponses ; ici, cela est plutôt gênant. C'est pour cela que la règle `filles` a été introduite.

```
?- filles(1,X).
X = [87, 120, 121, 124, 125, 126, 35, 15]
Yes
```

Pour l'instant, cette requête n'est pas très parlante. On verra plus tard comment traduire les identificateurs de page en URL.

**Ex :** Quelles sont toutes les pages accessibles depuis la page d'identificateur 999 ?

```
?- descendant(999,X).
No
```

Comme nous n'avons pas parcouru tout Internet, un grand nombre de pages dans notre base de données sont considérées comme vide donc elle ne pointent vers aucune autre page. C'est le cas pour la page 999.

**Ex :** Quels sont toutes les pages depuis lesquelles on accède directement à la page d'identificateur 165 ?

```
?- meres(165,X).
X = [4, 5, 21, 67, 62, 75, 76, 79, 80]
Yes
```

Ce qui correspond bien à ce qu'on obtient en SQL (voir exemple page 38) .

**Ex :** Quels sont tous les ancêtres de la page d'identificateur 1 ?

```
?- findall(X,ancetre(1,X),L).
X = _G158
L = [87, 3, 2, 6, 7, 9, 21, 5, 19|...]
Yes
```

Il est difficile de vérifier que Prolog trouve bien toutes les pages ancêtre. Cependant, on peut espérer que c'est le cas. C'est donc un résultat très intéressant, mais qui demande un temps de calcul assez long.

# II. Les requêtes

## II.1. Les requêtes SQL

Afin de montrer que le pouvoir expressif du modèle déductif est strictement supérieur à celui du modèle relatif, il est indispensable dans un premier temps de traduire en requêtes Prolog au moins une partie des requêtes SQL que nous avons vu dans la partie précédente.

### II.1.1. Requêtes pseudo atomiques

La traduction des requêtes SQL en requêtes Prolog est très simple (cela correspond à faire une sorte de jointure naturelle) et peu intéressante. Nous ne donnerons donc que deux exemples.

**Ex :** Quelles sont les fichiers référencés par le mot guide ? (requête équivalente à l'exemple page 30)

```
?- mot_desc('download',Id_mot),app_href(_,Id_href,Id_mot,_),
   href(1,Id_fic,Id_href),heberge(Id_fic,Id_proto,Id_serveur),
   protocole(Prefixe,Id_proto),serveur(Url_serveur,Id_serveur),
   fichier(Path,Nom,Id_fic).
```

Rem : Afin d'économiser un peu de place, seules les variables utiles (`Prefixe`, `Url_serveur`, `Path`, `Nom`) ont été recopiées.

```
Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'codeconv.zip' ;
```

```
Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'CodeConventions.pdf' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'CodeConventions.ps' ;

Prefixe = 'http://'
Url_serveur = 'java.sun.com'
Path = '/docs/books/effective/'
Nom = 'effective.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.html.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.html.tar.Z' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.ps.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-2.0.pdf' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.html.tar.Z' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.2nded.html.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.2nded.html.tar.Z' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.html.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.2nded.html.tar.gz' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.html.tar.gz' ;
```

```
Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.pdf' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.ps.Z' ;

No
```

On retrouve bien les 16 fichiers que nous avions obtenu avec la requête SQL.

**Ex :** Pour connaître toutes les pages contenant le mot java : (exemple SQL vue en page 33)

```
?-mot_txt('java',Id_mot),txt_page(Id_mot,Id_page,_),
  page(Id_serveur,Url_page,_,Id_page),
  serveur(Url_serveur,Id_serveur).
Url_page = '/j2se/1.4.1/docs/guide/awt/index.html'
Url_serveur = 'java.sun.com' ;
Url_page = '/j2se/1.4.1/docs/guide/reflection/index.html'
Url_serveur = 'java.sun.com' ;
Url_page = '/j2se/1.4.1/docs/guide/awt/AWTChanges.html'
Url_serveur = 'java.sun.com' ;
Url_page = '/j2se/1.4.1/docs/guide/serialization/relnotes14.html'
Url_serveur = 'java.sun.com' ;
Url_page = '/help/legal_and_licensing'
Url_serveur = 'servlet.java.sun.com' ;
...
```

On retrouve les mêmes 22 pages que pour la requête SQL.

### II.1.2. Requêtes complètes

Les requêtes SQL complètes ne sont pas beaucoup plus difficiles à traduire en Prolog que les requêtes pseudo atomiques car les unions se traduisent assez naturellement par des disjonctions.

**Ex :** Reprenons l'exemple page 36 : Quels sont tous les fichiers (documents textuels inclus) relatif au mot java ?
Rem : Ne sachant comment traduire l'opérateur `LIKE` de SQL, nous avons mis `*` pour simuler `%`. Cela est probablement faut mais ce n'est pas important puisqu'il n'y a de toute façon pas de fichier dont le nom commence par « java » dans la base.

```
?-((mot_txt('java',Id_txt),txt_doc(Id_fic,Id_txt,_));
   (mot_desc('java',Id_mot),app_href(_,Id_href,Id_mot,_),
       href(1,Id_fic,Id_href));
   (fichier(_,'java.*',Id_fic))),
  heberge(Id_fic,Id_proto,Id_serveur),protocole(Prefixe,Id_proto),
  serveur(Url_serveur,Id_serveur),fichier(Path,Nom,Id_fic).
Prefixe = 'ftp://'
Url_serveur = 'ftp.java.sun.com'
Path = '/docs/j2se1.4/'
Nom = 'VolatileImage.pdf' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.pdf' ;

Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/rfc/'
Nom = 'rfc2222.txt' ;

Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/internet-drafts/'
Nom = 'draft-ietf-ldapext-locate-08.txt' ;
```

```
Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/rfc/'
Nom = 'rfc2830.txt' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.java.sun.com'
Path = '/docs/j2se1.4/'
Nom = 'serial-spec.ps' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'CodeConventions.pdf' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'CodeConventions.ps' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.java.sun.com'
Path = '/docs/j2se/1.4/'
Nom = 'JPS_PDF.pdf'

Prefixe = 'ftp://'
Url_serveur = 'ftp.java.sun.com'
Path = '/docs/j2se1.4/'
Nom = 'versioning.ps' ;

Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/rfc/'
Nom = 'rfc2246.txt' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'langspec-1.0.html.zip' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/specs/'
Nom = 'vmspec.2nded.html.tar.gz' ;

Prefixe = 'ftp://'
Url_serveur = 'ftp.javasoft.com'
Path = '/docs/codeconv/'
Nom = 'codeconv.zip' ;

Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/rfc/'
Nom = 'rfc2246.txt' ;

Prefixe = 'http://'
Url_serveur = 'www.ietf.org'
Path = '/internet-drafts/'
Nom = 'draft-ietf-ldapext-locate-07.txt' ;

No
```

### II.2. Les requêtes spécifiques à Prolog

Toutes les requêtes concernant le parcourt non borné du graphe d'Internet n'est réalisable qu'avec Prolog.

**Ex :** Si on reprend l'exemple page 38 où on voulait faire du contrôle parental au niveau du moteur de recherche, il est possible d'établir un contrôle totalement imparable en interdisant toutes les pages ancêtres de toutes les pages à bloquer. Cela peut se faire grâce à la règle `ancetre`.

Toutes les pages ancêtres de pages relatives à windows :

```
?-assert( page_windows(Id_page) :-
            ((mot_txt('windows',Id_txt),txt_page(Id_page,Id_txt,_));
             (mot_desc('windows',Id_mot),app_href(_,Id_href,Id_mot,_),
                  href(2,Id_page,Id_href))),
             page(_,_,_,Id_page) ).
?-findall(Id_page,page_windows(X),L),
  maplist(ancetre,L,L2),
  flatten(L2,Ancetres).
```

Cette exemple ne marche pas en l'état. Cependant, il existe très certainement une requête correcte dans le même esprit.

# Conclusion

Dans ce projet, nous avons implémenté une base de données pour un moteur de recherche sur Internet alliant plusieurs qualités :

- Elle est très complète d'un point de vue utilisateur puisqu'elle lui permet de rechercher des documents (aussi des fichiers, que des images, des pages web ou des images) selon un nombre important de critères. Elle autorise également certaines recherches rarement proposées par les moteurs usuels comme, par exemple, le contrôle parental ou n'importe quelle requête faisant intervenir le parcours d'Internet sous forme de graphe.
- Elle est robuste d'un point de vue développeur car sa structure garantit une espérance de fonctionnement, sans modification fondamentale, assez longue.

A côté de cela, ce projet nous a surtout permit de constater, exemples à l'appuie, que le modèle déductif avait un pouvoir d'expression strictement supérieur au modèle relatif qui reste malgré tout très utilisé dans le cas particulier d'un moteur de recherche car il est rapide et suffit pour répondre à la majorité des demandes d'utilisateurs.