Bases de données avancées

Recherche d'information

Auteurs : Raphaël Fournier-S'niehotta, Nicolas Travers, Philippe Rigaux
fournier@cnam.fr, nicolas.travers@cnam.fr, philippe.rigaux@cnam.fr

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan

Plan du cours

Information Retrieval, définition

Une définition

La Recherche d'Information (*Information Retrieval*, IR) consiste à trouver des documents peu ou faiblement structurés, dans une grande collection, en fonction d'un besoin d'information.

- Recherche sur le Web. Utilisée quotidiennement par des milliards d'utilisateurs.
- Recherche dans votre boîte mail.
- Recherche sur votre ordinateur (*Spotlight*).
- Recherche dans une base documentaire, publique ou privée.

- Recherche plein texte : on cherche à examiner tous les mots de chaque document enregistré et à essayer de les faire correspondre à ceux fournis par l'utilisateur

- À bien distinguer d'une recherche type "base de données" : requête structurée, données structurées, réponse « exacte ».

Évaluation d'un moteur de recherche

Contrairement à une base de données (SQL), le résultat dépend de l'interprétation d'un besoin. On ne peut jamais dire qu'un résultat est totalement exact (ou totalement faux).

Deux notions importantes :

- Faux positifs : ce sont les documents non pertinents inclus dans le résultat; ils ont été sélectionnés à tort.
- Faux négatifs : ce sont les documents pertinents qui ne sont pas inclus dans le résultat.

En général, chercher à réduire les faux positifs entraîne l'augmentation des faux négatifs, et réciproquement.

Évaluation d'un moteur de recherche (suite)

- La recherche plein texte est susceptible de récupérer beaucoup de faux positifs.
- La récupération de documents non pertinents est souvent provoquée par l'ambiguïté inhérente au langage naturel ;
- par exemple, le mot avocat désigne aussi bien un fruit qu'une profession
- les documents traitant de l'un ne sont pas pertinents pour le chercheur qui s'intéresse à l'autre.

Évaluation de la pertinence : précision et rappel

La précision mesure la fraction des vrais positifs dans le résultat r.

Si on note t_p et t_n le nombre de vrais et de faux positifs dans r, alors

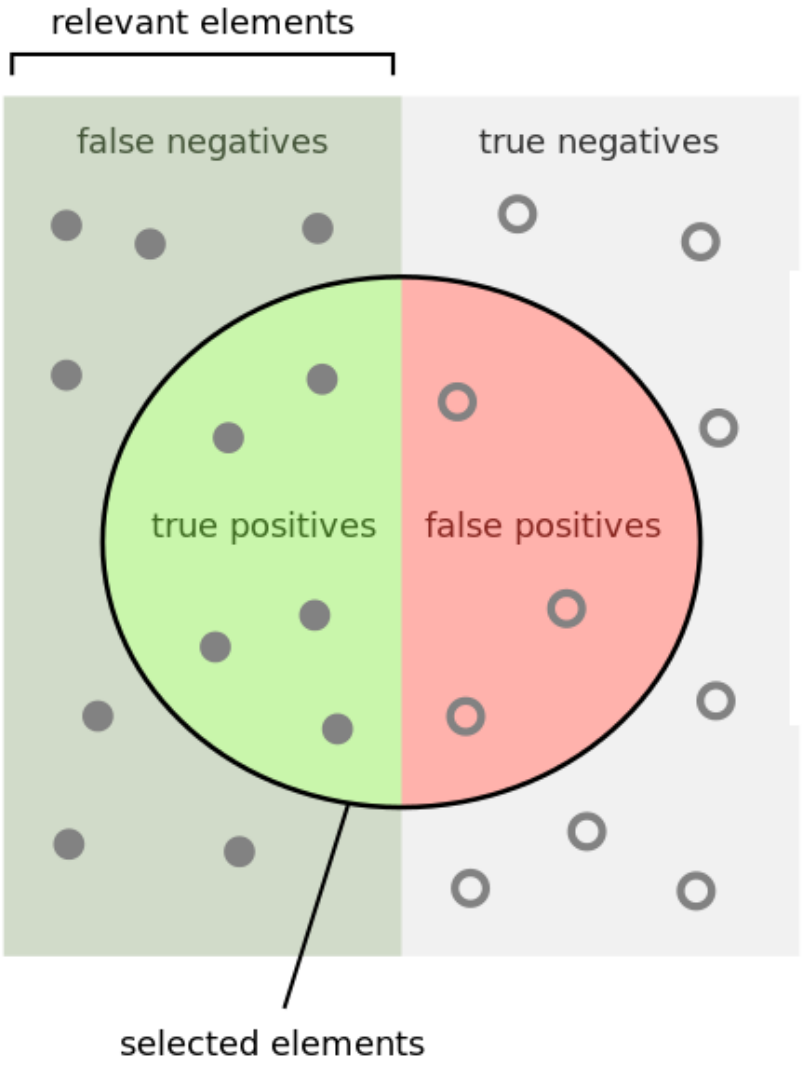
$$\text{précision} = \frac{t_p}{t_p + f_p} = \frac{t_p}{|r|}$$

Le rappel mesure la fraction de faux négatifs.

$$\text{rappel} = \frac{t_p}{t_p + f_n}$$

L'évaluation d'un système de RI est difficile : implique des tests rigoureux avec des utilisateurs sur un échantillon.

Illustration des faux-positifs et négatifs



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

Performances

On peut améliorer les performances des moteurs de recherche à l'aide de plusieurs techniques :

- des requêtes plus structurées
  - requêtes booléennes,
  - expressions rationnelles,
  - proximité,
  - recherche d'expression
- des résultats classés
  - modèle vectoriel
  - PageRank
  - analyse sémantique latente

Moteurs de recherche

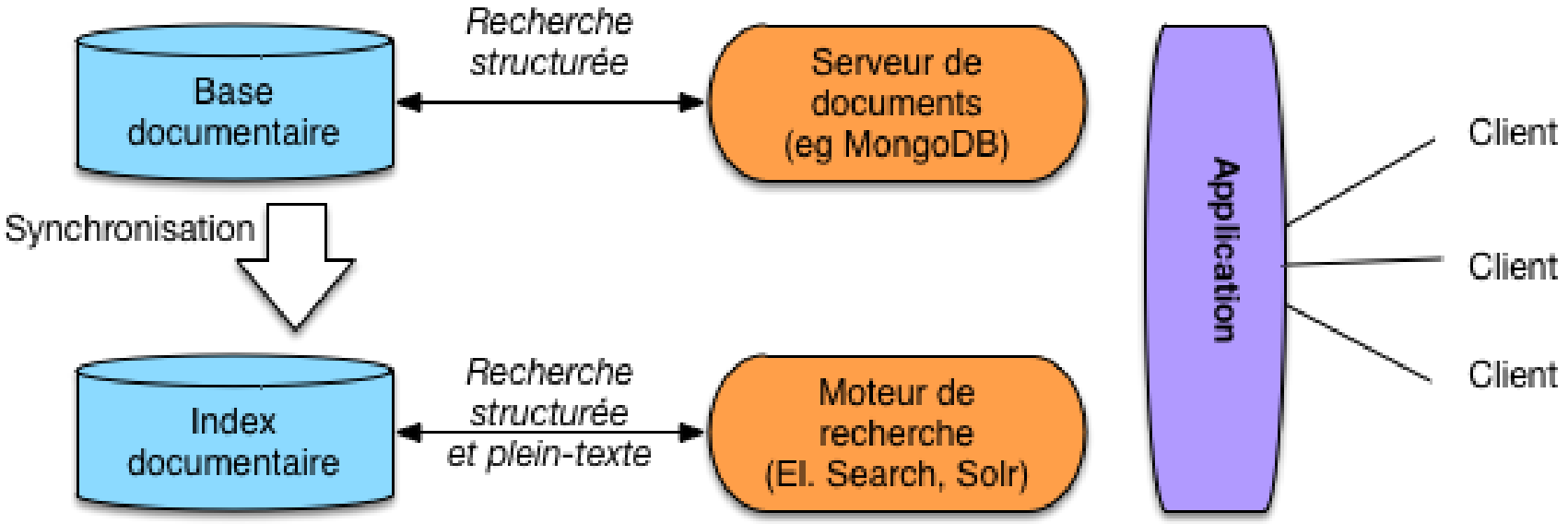
Moteurs libres

- Apache Solr (Lucene)
- Elastic Search (Lucene)
- Sphinx
- Xapian

Moteurs commerciaux

- Google Search Appliance
- Exalead
- Amazon CloudSearch
- Microsoft Azure Search

Bases documentaires et moteur de recherche



Bases documentaires et moteur de recherche

Un scénario typique :

- une recherche par mot-clé dans un site
- Les données du site sont gérées classiquement par une base documentaire (MySQL, Postgres, MongoDB)
- On extrait de la base tous les textes à indexer et on en fait des documents ES/Solr, à qui on les confie
- Ce dernier se charge alors de répondre quand un utilisateur emploie le un champ de recherche.

Bases documentaires et moteur de recherche

- Un moteur de recherche comme ElasticSearch ou Solr s'appuie sur un index
- Pourquoi ne pas utiliser directement le moteur de recherche comme gestionnaire des documents ?
- ElasticSearch permet des recherches puissantes, efficaces, ainsi que le stockage et l'accès aux documents.

Bases documentaires et moteur de recherche

- Mais, comme ElasticSearch est entièrement consacré à la recherche, c'est-à-dire la lecture la plus efficace possible de documents.
- il s'appuie pour cela sur des structures compactes, compressées, optimisées (les index inversés) que l'on va voir plus tard
- ce n'est pas un très bon outil pour les autres fonctionnalités d'une base de données :
  - Le stockage par exemple n'est ni aussi robuste ni aussi stable.
  - Pour des raisons qui tiennent à la structure de ces index, les mises à jour sont difficiles et s'effectuent difficilement en temps réel

Plan du cours

Plan du cours

Exemple de base

Un ensemble (modeste) de documents nous servira de guide.

- d_1 Le loup est dans la bergerie.
- d_2 Le loup et le trois petits cochons.
- d_3 Les moutons sont dans la bergerie.
- d_4 Spider Cochon, Spider Cochon, il peut marcher au plafond.
- d_5 Un loup a mangé un mouton, les autres loups sont restés dans la bergerie.
- d_6 Il y a trois moutons dans le pré, et un mouton dans la gueule du loup.
- d_7 Le cochon est à 12 le Kg, le mouton à 10/Kg.
- d_8 Les trois petits loups et le grand méchant cochon.

Le besoin et la solution

Besoin

On veut chercher tous les documents parlant de loups, de moutons mais pas de bergerie.

Parcourir tous les documents ? (Grep)

- potentiellement long;
- critère "pas de bergerie" n'est pas facile à traiter;
- autres types de recherche ("le mot 'loup' doit être près du mot 'mouton'") sont difficiles;
- comment classer par pertinence les documents trouvés ?

Structure spécialisée : la matrice d'incidence et surtout son inversion.

Matrice avec documents en ligne

On sélectionne un ensemble de mots (ou termes), constituant notre vocabulaire (ou dictionnaire).

Documents en ligne, termes en colonnes. Dans chaque cellule : 1 si le terme est dans le document, 0 sinon.

La matrice d'incidence

	loup	mouton	cochon	bergerie	pré	gueule
d_1	1	0	0	1	0	0
d_2	1	0	1	0	0	0
d_3	0	1	0	1	0	0
d_4	0	0	1	0	0	0
d_5	1	1	0	1	0	0
d_6	1	1	0	0	1	1
d_7	0	1	1	0	0	0
d_8	1	0	1	0	0	0

Pour effectuer la recherche

(loup, mouton, et pas bergerie) On prend les vecteurs binaires des termes (les colonnes).

- Loup : 11001101
- Mouton : 00101110
- Bergerie : 01010011

Puis :

- ET logique sur les vecteurs de Loup et Mouton, on obtient 00001100.
- ET logique avec le complément du vecteur de Bergerie (01010111)

On obtient 00000100, d'où on déduit que la réponse est limitée au document d_6.

Opération binaires très efficace, mais...

Passons à grande échelle

Quelques hypothèses :

- Un million de documents, mille mots chacun en moyenne.
- Disons 6 octets par mot, soit 6 Go (ce n'est pas une si grosse base que cela)
- Disons 500 000 termes distincts

$\Rightarrow$ la matrice d'incidence a :

- 10^6 lignes et 500 000 colonnes soit 500×10^9 bits
- soit 62 Go approximativement

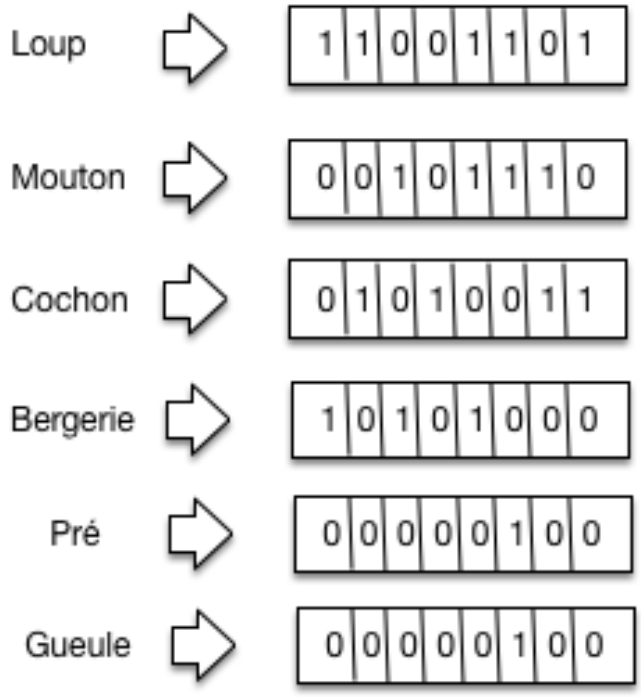
Ne tient pas en mémoire, ce qui va beaucoup compliquer les choses....

Comment faire mieux ?

On peut faire mieux

Il vaut mieux avoir les termes en ligne pour disposer des vecteurs dans une zone mémoire contigue.

On parle de matrice inversée, et de liste inversée. Une liste par terme; dans chaque liste, 1 pour les documents contenant le terme.

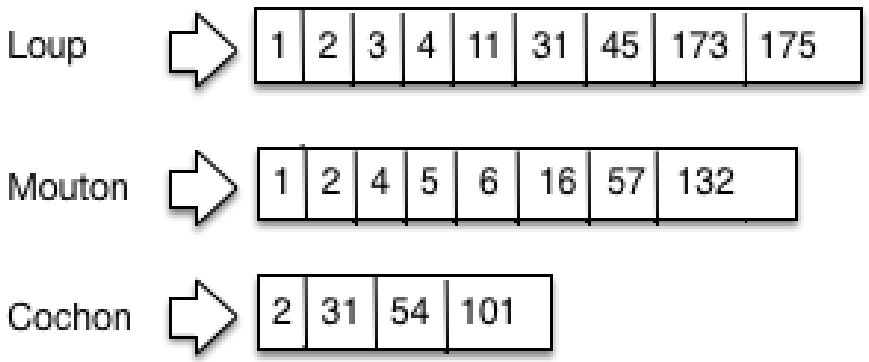


Important

Les mises à jour beaucoup plus difficiles car elles impliquent la réorganisation d'une liste compacte. Prix à payer pour l'efficacité en lecture (recherche).

On peut encore faire mieux

La matrice est creuse : il n'y a que 10^9 positions avec des 1, soit un sur 500.



Essentiel

On place dans les cellules l'identifiant du document. De plus chaque liste est triée sur l'identifiant du document.

Index inversé

La structure utilisée dans tous les moteurs de recherche.

- Un répertoire contient tous les termes.
- Une liste (inversée) est associée à chaque terme, triée par docId.
- Chaque élément de la liste est appelé une entrée.

Concept : la notion de terme (token en anglais) est différente de celle de “mot”.

Vocabulaire : le répertoire est parfois appelé *dictionnaire* ; les listes sont des *posting list* en anglais; les entrées sont des *postings*.

Efficacité : le répertoire devrait toujours être en mémoire ; les listes, autant que possible en mémoire, sinon fichiers contigus sur le disque.

Plan du cours

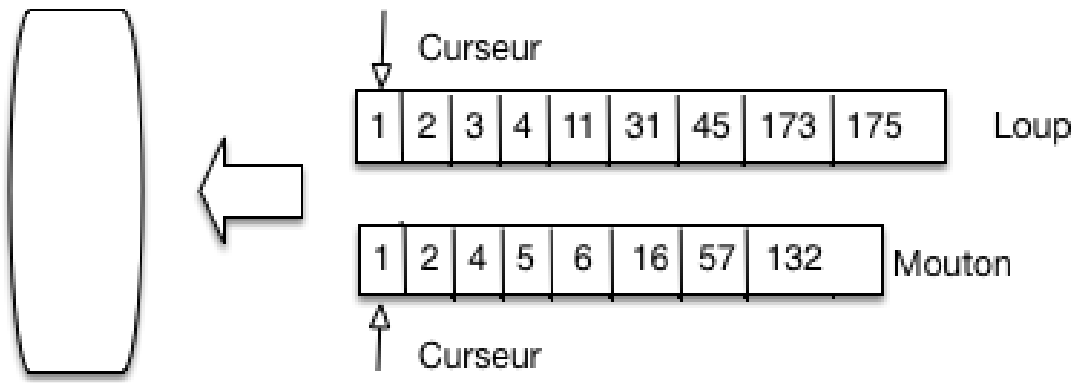
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



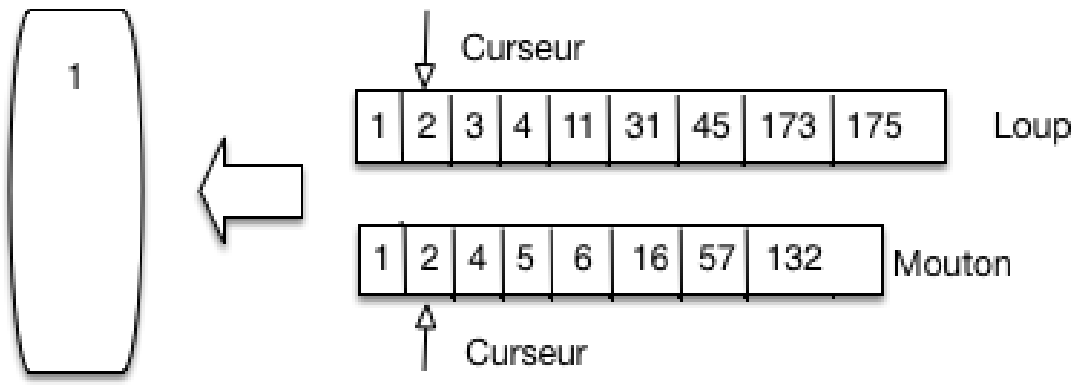
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

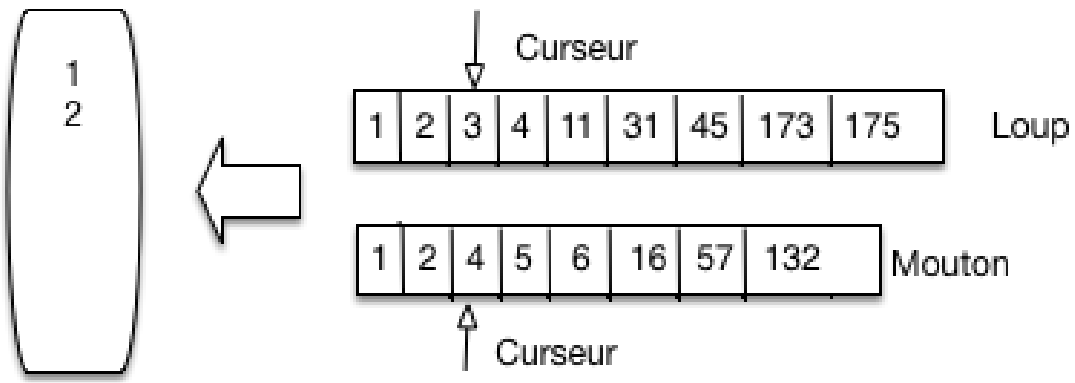
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

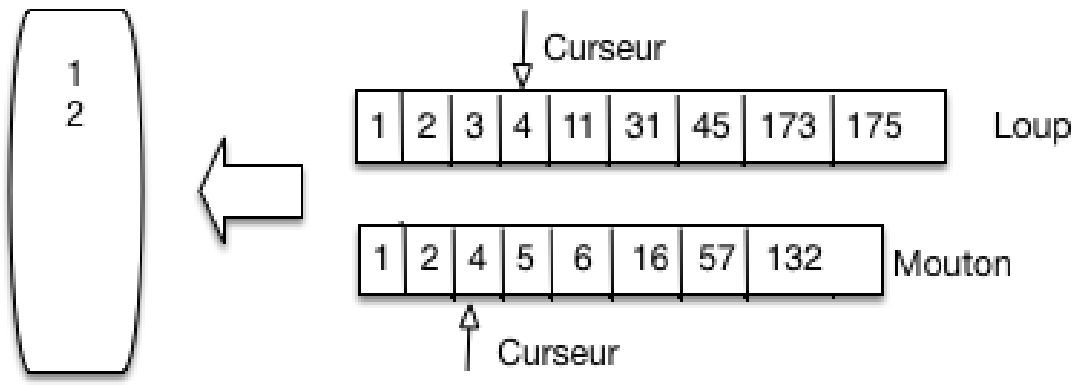
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

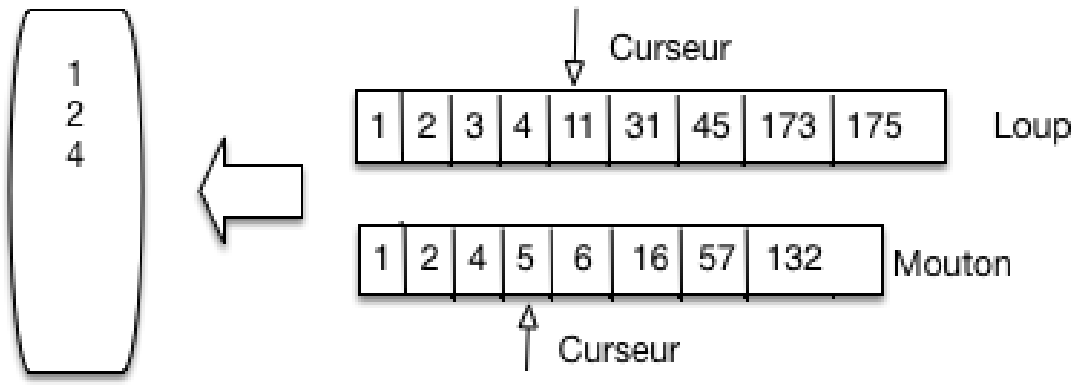
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

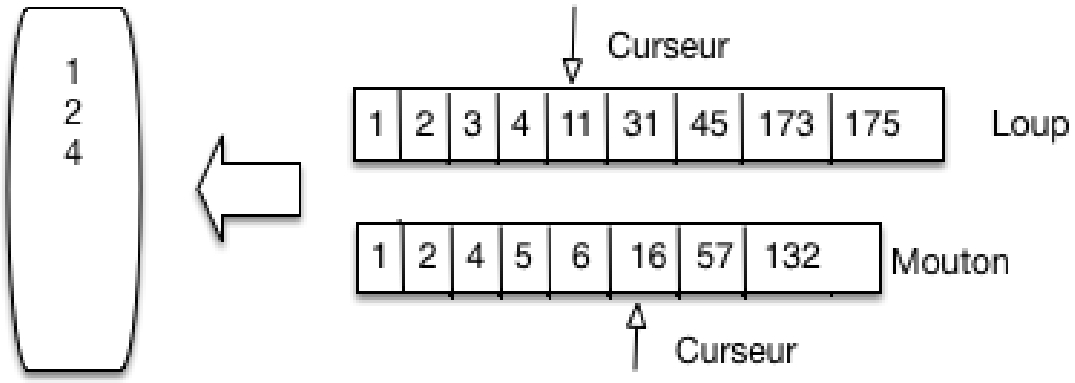
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

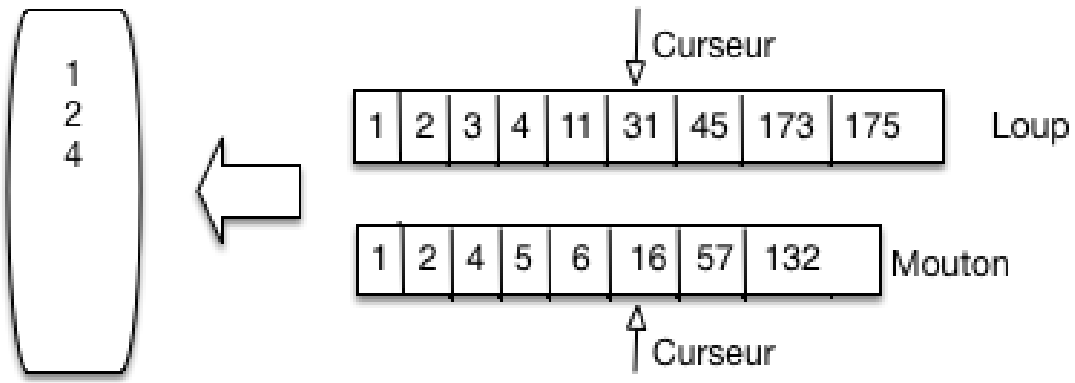
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

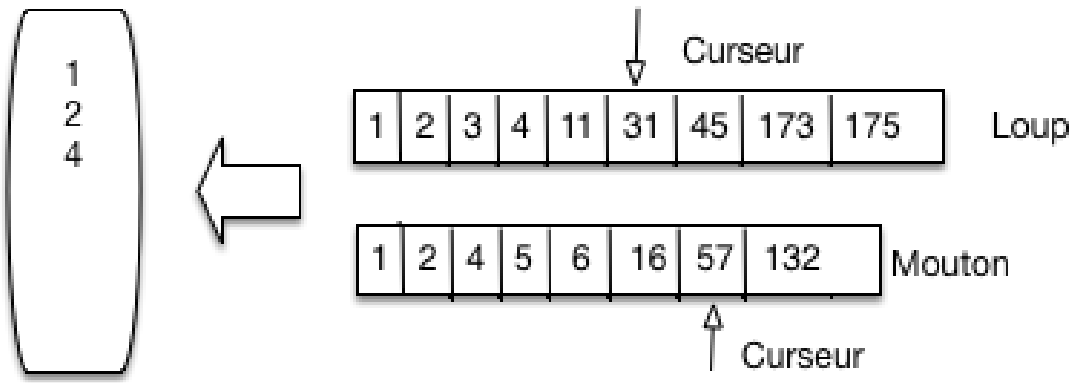
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

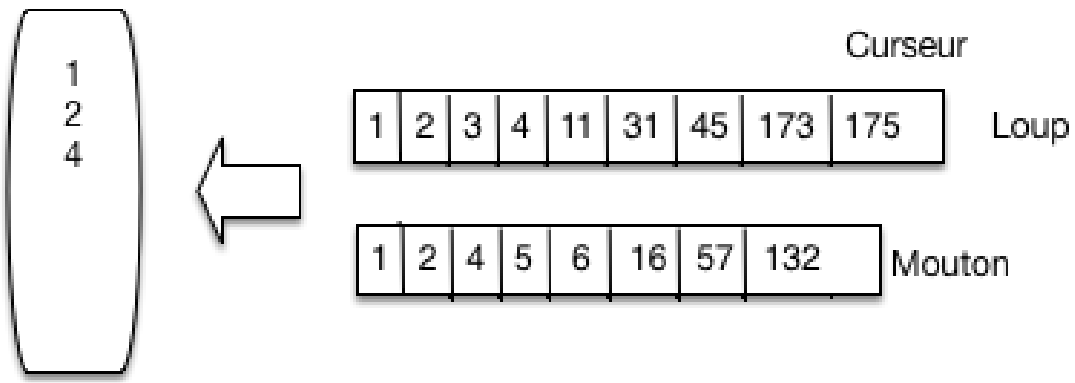
Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

On avance sur la liste du plus petit docId.



Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

Algorithme

```
// Fusion de deux listes l1 et l2
function Intersect($l1, $l2)
{
  $résultat = [];
  // Début de la fusion des listes
  while ($l1 != null and $l2 != null) {
    if ($l1.docId == $l2.docId) {
      // On a trouvé un document contenant les deux termes
      $résultat += $l1.docId;
      // Avançons sur les deux listes
      $l1 = $l1.next; $l2 = $l2.next;
    }
    else if ($l1.docId < $l2.docId) {
      // Avançons sur l1
      $l1 = $l1.next;
    }
    else {
       // Avançons sur l2
      $l2 = $l2.next;
    }
  }
}
```

Optimisation

- Si l'on veut effectuer la recherche :

```
A AND B AND C
```

- Quelle manière de procéder va optimiser le résultat ?

Optimisation

- Si l'on veut effectuer la recherche :

```
A AND B AND C
```

- Quelle manière de procéder va optimiser le résultat ?
- on stocke la taille des listes et l'on commence par faire l'intersection des plus petites
- de nombreuses autres questions d'optimisation existent
- exemple : stocker l'une des listes en mémoire et calculer les intersections à la volée, en lisant depuis le disque

Premier bilan

En résumé : index inversé, tri et compaction, parcours linéaire très rapide sont les fondements techniques de la RI.

- Permet d'effectuer des recherches puissantes (combinaison de critères) et flexibles.
- Garantissent une très grande efficacité.

Que reste-t-il à faire ?

Quels termes ? Quels termes indexe-t-on ? Beaucoup moins facile que ça n'en a l'air...

Quelles requêtes ? De la plus simple ("sac de mots") à plus structurée (index struturé, requêtes Boolénnes, recherche de phrases).

Performance. Construction, compression, distribution, optimisation des accès, mises à jour, etc.

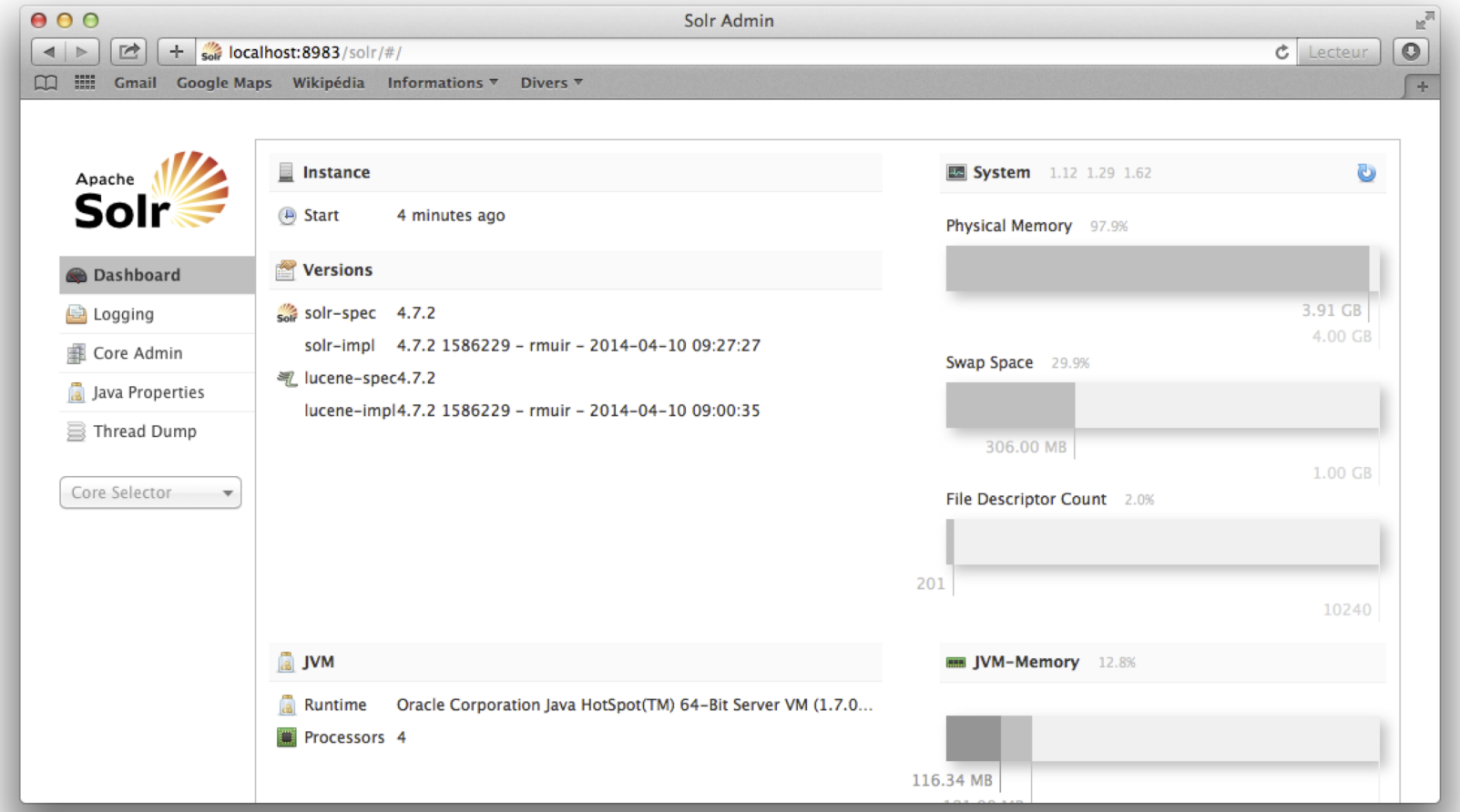
Classement ? Si j'ai des millions de documents dans le résultat, je veux les classer. Comment ?

Exercices

Exercices

Plan du cours

Interface


Solr Admin
localhost:8983/solr/#/
Lecteur
Gmail
Google Maps
Wikipédia
Informations
Divers
Apache Solr
Dashboard
Logging
Core Admin
Java Properties
Thread Dump
Core Selector
Instance
Start 4 minutes ago
Versions
solr-spec 4.7.2
solr-impl 4.7.2 1586229 - rmuir - 2014-04-10 09:27:27
lucene-spec 4.7.2
lucene-impl 4.7.2 1586229 - rmuir - 2014-04-10 09:00:35
JVM
Runtime Oracle Corporation Java HotSpot(TM) 64-Bit Server VM (1.7.0...
Processors 4
System 1.12 1.29 1.62
Physical Memory 97.9%
3.91 GB
4.00 GB
Swap Space 29.9%
306.00 MB
1.00 GB
File Descriptor Count 2.0%
201
10240
JVM-Memory 12.8%
116.34 MB

Interagir avec Solr

- Solr fournit des services REST (HTTP) pour recevoir/envoyer des instructions
- codées en JSON ou XML
- insérer de documents
- effectuer des requêtes

Exemple pour une indexation:

- accessible à l'URL `http://localhost:8983/solr/update`
- attente de document de type "Content-type:application/xml" (binaire)
- on utilise curl :
  curl http://localhost:8983/solr/update –data-binary @item.xml -H 'Content-type:application/xml'

Interroger un index avec Solr

- Solr dispose d'une interface REST pour **rechercher** des documents :

```
http://localhost:8983/solr/select?q=video
```

- dans le navigateur, ou avec curl :

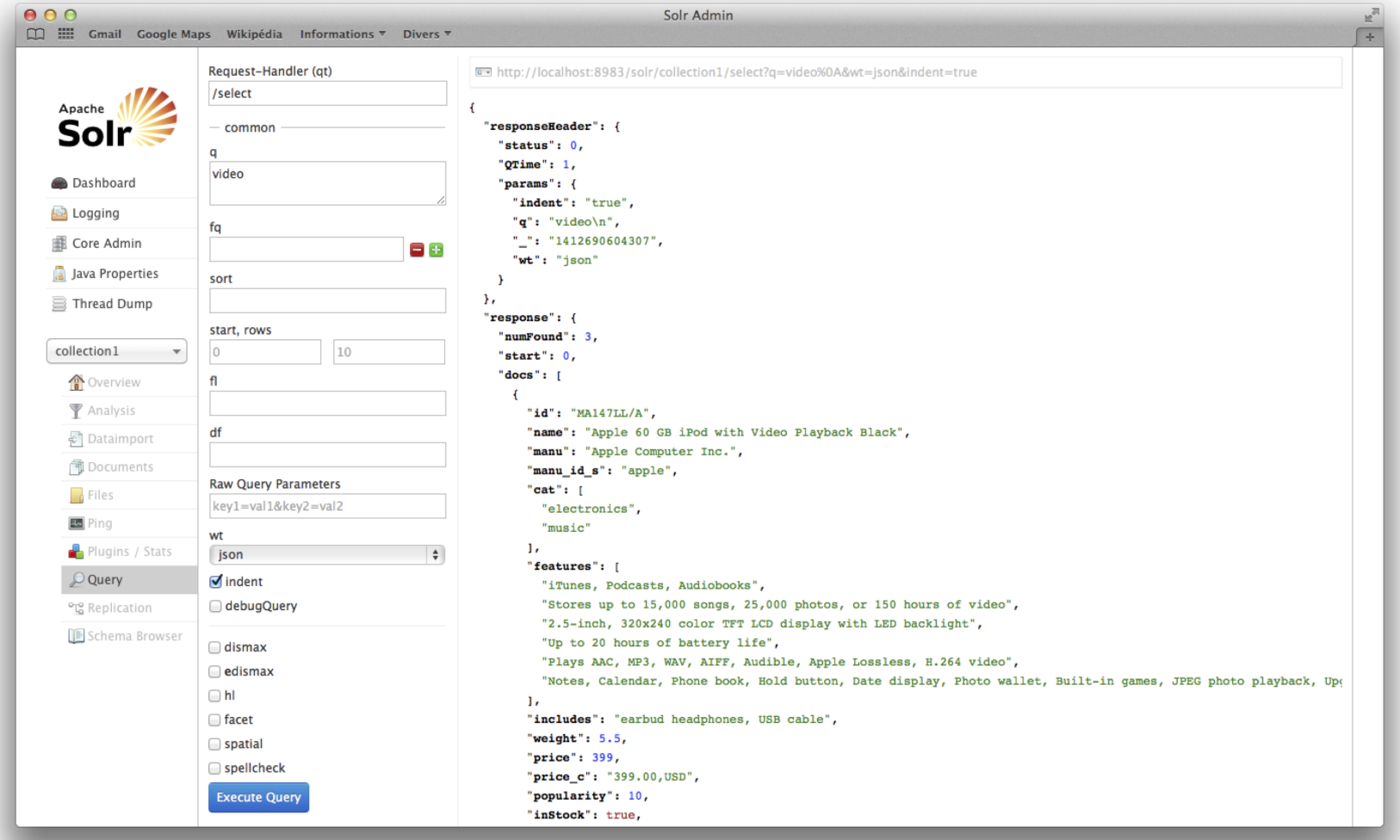
```
curl http://localhost:8983/solr/select?q=video
```

- La réponse est formattée en XML. On peut aussi demander un retour en JSON :

```
http://localhost:8983/solr/select?q=video&wt=json&indent=yes
```

- interface d'administration intégrée

Fenêtre d'interrogation de Solr



Réponse de Solr

- En-tête (propriétés sur l'exécution, dont temps de réponse)
- Liste des **documents Solr** considérés comme satisfaisant les critères
  - constitués des valeurs de champs insérés dans l'index
  - cela ne suffira cependant souvent pas, il faudra accéder au **document applicatif** correspondant
  - importance du champ **id** (clef d'accès base documentaire)

♠ Certains champs (**cat** et **features** par exemple) sont multivalués (ils sont représentés par des tableaux en JSON). L'index en a tenu compte

Illustration de la puissance de Solr

- recherche par mot-clef
- **projection** (SQL) avec le paramètre **fl** (**field list**) :

```
http://localhost:8983/solr/select?q=video&fl=id,name
```

- spécifier dans quel champ on cherche :

```
http://localhost:8983/solr/select?q=name:black
```

- pagination de résultats :

```
http://localhost:8983/solr/select?q=video&start=1&rows=10
```

Démonstration

démonstration

Termes

- Notion de base : le **terme**
- c'est un mot au sens usuel
- ou une séquence de mots entre apostrophes

On peut interroger un index avec :

```
hard drive
```

Puis :

```
"hard drive"
```

- Première recherche : documents avec "hard", "drive" ou les deux
- Deuxième : seulement "hard drive" (côte à côte)

Termes (suite)

- la recherche d'un terme s'effectue toujours sur un champ.
- La syntaxe complète pour associer le champ et le terme est:

```
champ:terme
```

- si non précisé, c'est le champ par défaut qui est utilisé
- pratique courante : concaténer toutes les chaînes de caractères en un champ "text" général, défini par défaut
- Nos requêtes deviennent :

```
text:hard text:drive
```

- et

```
text:"hard drive"
```

Termes (suite)

- Les valeurs des termes (dans la requête) et le texte indexé sont tous deux soumis à des transformations spécifiées dans le schéma.
- Une transformation simple est de tout transcrire en minuscules.

```
text:"Hard Drive"
```

- Les transformations appliquées à la requête ET au texte indexé doivent être cohérentes : si les termes sont transformés en majuscules, et le texte indexé en minuscules, on n'aura jamais de résultat!

Termes (suite)

On peut spécifier des termes (pas des séquences) incomplets

- le '?' indique un caractère inconnu
  - "opti?al" désigne "optimal", "optical", etc.
- le '*' indique n'importe quelle séquence de caractères
  - "opti*" pour toute chaîne commençant par "opti"

Approximations avec "∼" :

- Rechercher "optimal" et "optimal∼"
- 0 et 1 résultat ("optical")
- Proximité des termes par une distance d'édition :
  (nb opérations pour passer de "optimal" à "optical")

Intervalles :

- [] bornes comprises
- { } bornes exclues

```
%price:[100 TO 200]
```

Requêtes Booléennes

- Les critères peuvent être combinés avec des opérateurs Booléens : **AND**, **OR** et **NOT**
- Attention : majuscules

```
%price:[100 TO 300] OR popularity:5
%price:[100 TO 300] AND NOT popularity:5
%popularity:6 AND features:matrix
```

- Par défaut, c'est un **OR** qui est appliqué
- Recherche sur plusieurs critères ramène l'union des résultats sur chaque critère pris individuellement

Plan du cours

4 Construction de l'index
- Tokenisation et normalisation
- Racinisation
- Mots vides, synonymes, cas particuliers...
- Construction d'un moteur de recherche

Pré-traitement

- Dès qu'un document est un peu complexe, on ne peut pas le découper arbitrairement sans appliquer un pré-traitement réfléchi
- sinon :
  - que devient "pomme de terre" ?
  - on stocke des listes très longues pour certains mots...
  - l'utilisateur doit faire très attention ("loup", "Loup", "loups" sont différents)
- il faut procéder à une analyse du contenu

Rôle de l'analyse

L'analyse des textes permet d'effectuer une forme de normalisation / unification pour être moins dépendant de la forme du texte.

- un document parle de loup même si on y trouve les formes "loups", "Loup", "louve", etc.
- un document parle de travail quelle que soit la forme du verbe "travailler" ou de ses variantes.
- Jusqu'où va-t-on? Traductions (loup = wolf = lupus) ? Synonymes (loup = prédateur) ? (sujet de recherches)

Pour moins dépendre de la forme on applique des transformations.

Impact de l'analyse

Bien comprendre :

Plus on normalise, plus on diminue la précision.
Car des mots distincts sont unifiés (cote, côte, côté, etc.)

Plus on normalise, plus on améliore le rappel.

Car on met en correspondance les variantes d'un même mot, d'une même signification (conjugaisons d'un verbe).

Impact de l'analyse

Très important

La même transformation doit être appliquée aux documents et à la requête.

Pourquoi ?

- sinon, il se crée un décalage entre ce qui a été analysé et ce qui peut être réellement recherché

Les phases de l'analyse

Important: identification de quelques méta-données (la langue), prise en compte du contexte (quels documents pour quelle application).

Puis, de manière générale :

- Tokenization : découpage du texte en "mots"
- Normalisation : majuscules ? acronymes ? apostrophes ? accents ?
  Exemple : *Windows* et *window*, U.S.A vs USA, *l'étudiant* vs *les étudiants*.
- Stemming ("racinisation"), lemmatization
  Prendre la racine des mots pour éviter le biais des variations (étudier, étudiant, étude, etc.)
- Stop words, quels mots garder ?
  Mots très courants peu informatifs (le, un à, de).

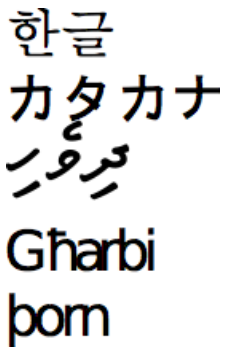
C'est de l'art et du réglage... Dans ce qui suit : introduction / sensibilisation aux problèmes.

Plan du cours

Identification de la Langue

Comment trouver la langue d'un document ?

- Méta information (dans l'entête HTTP p.e.): pas fiable du tout.
- Par le jeu de caractères, pas assez courant!



Identification de la Langue

Comment trouver la langue d'un document ?

- Méta information (dans l'entête HTTP p.e.): pas fiable du tout.
- Par le jeu de caractères, pas assez courant!

  Respectivement: Coréen, Japonais, Maldives, Malte, Islandais.

- Par extension: séquences de caractères fréquents, (*n*-grams)
- Par techniques d'apprentissage (classifiers)

Des librairies font ça très bien (e.g., Tika, `http://tika.apache.org`)

Tokenisation

Principe

Séparation du texte en tokens (“mots”)

Pas du tout aussi facile qu'on le dirait !

- Dans certaines langues (Chinois, Japonais), les mots ne sont pas séparés par des espaces.
- Certaines langues s'écrivent de droite à gauche, de haut en bas.
- Que faire (et de manière cohérente) des acronymes, élisions, nombres, unités, URL, email, etc.

Tokenisation

- Mots composés : les séparer en *tokens* ou les regrouper en un seul ?
  1. Anglais : *hostname*, *host-name* et *host name*, ...
  2. Français : Le Mans, aujourd'hui, pomme de terre, ...
  3. Allemand : *Levensversicherungsgesellschaftsangestellter* (employé dune société dassurance vie)

  Que faire si l'utilisateur cherche *hostname* et qu'on a normalisé en *host-name* ?

Majuscules, ponctuation ? Une solution simple est de normaliser (minuscules, pas de ponctuation).

Exemple pour notre petit jeu de données

On met en minuscules, on retire la ponctuation.

d_1	le loup est dans la bergerie
d_2	le loup et les trois petits cochons
d_3	les moutons sont dans la bergerie
d_4	spider cochon spider cochon il peut marcher au plafond
d_5	un loup a mangé un mouton les autres loups sont restés dans la bergerie
d_6	il y a trois moutons dans le pré et un mouton dans la gueule du loup
d_7	le cochon est à 12 euros le kilo le mouton à 10 euros le kilo
d_8	les trois petits loups et le grand méchant cochon

On considère que l'espace est le séparateur de tokens.

Plan du cours

Stemming (racine), lemmatization

Principe

Confondre toutes les formes d'un même mot, ou de mots apparentés, en une seule racine.

Stemming Morphologique. Retire les pluriels, marque de genre, conjugaisons, modes, etc.

- Très dépendant de la langue : *geese* pluriel de *goose*, *mice* de *mouse*
- Difficile à séparer d'une analyse linguistique ("Les poules du couvent couvent", "la petite brise la glace" : où est le verbe ?)

Stemming lexical Fondre les termes proches lexicalement : "politique, politicien, police (?)" ou "université, universel, univers (?)"

Stemming phonétique. Correction fautes de frappes, fautes orthographes

Exemple de stemming

On retire les pluriels, on met le verbe à l'infinitif.

d_1	le loup etre dans la bergerie
d_2	le loup et les trois petit cochon
d_3	les moutons etre dans la bergerie
d_4	spider cochon spider cochon il pouvoir marcher au plafond
d_5	un loup avoir manger un mouton les autres loups etre rester dans la bergerie
d_6	il y avoir trois mouton dans le pre et un mouton dans la gueule du loup
d_7	le cochon etre a 12 euro le kilo le mouton a 10 euro le kilo
d_8	les trois petit loup et le grand mechant cochon

Plan du cours

Suppression des *Stop Words*

Principe

On retire les mots porteurs d'une information faible afin de limiter le stockage.

articles: *le*, *le*, *ce*, etc.

verbes "fonctionnels" *être*, *avoir*, *faire*, etc.

conjunctions: *that*, *and*, etc.

etc.

♣ Maintenant moins utilisé car (i) espace de stockage peu coûteux et (ii) pose d'autres problèmes ("pomme de terre", "Let it be", "Stade de France")

Autres problèmes, en vrac

Majuscules / minuscules

Lyonnaise des Eaux, Société Générale, etc.

Acronymes

CAT = *cat* ou *Caterpillar Inc.* ? M.A.A.F ou MAAF ou Mutuelle ... ?

Dates, chiffres

Monday 24, August, 1572 – 24/08/1572 – 24 août 1572 10000 ou 10,000.00 ou 10,000.00

Accents, ponctuation

résumé ou résume ou resume...

- ♣ Dans tous les cas, les même règles de transformation s'appliquent aux documents ET à la requête.

Exemple avec suppression des *stop words*

Voici une solution possible.

d_1	loup etre bergerie
d_2	loup trois petit cochon
d_3	mouton etre bergerie
d_4	spider cochon spider cochon pouvoir marcher plafond
d_5	loup avoir manger mouton autres loups etre rester bergerie
d_6	avoir trois mouton pre mouton gueule loup
d_7	cochon etre 12 euro kilo mouton 10 euro kilo
d_8	trois petit loup grand mechant cochon

On a gardé les verbes fonctionnels (être, avoir).

Plan du cours

Un document (exemple)

```
{
  "_id": "movie:57",
  "title": "Jackie Brown",
  "year": "1997",
  "genre": "crime",
  "summary": "Jacky Brown, hotesse de l'air, ...",
  "country": "USA",
  "director": "Quentin Tarantino",
  "actors": ["Robert De Niro", "Pam Grier", "Bridget Fonda",
             "Michael Keaton","Samuel Jackson"]
}
```

Schéma de l'index

- Le schéma donne la liste de tous les champs d'un doc Solr
- Nombreuses options :
  - type (numérique, entier)
  - possibilité de calcul de la valeur du champ à partir d'un autre
  - traitements divers sur les valeurs du champ

Squelette

```
 <?xml version="1.0" encoding="UTF-8" ?>

 <schema name="example" version="1.5">
  <!--  Liste des champs de l'index -->
<fields>
  <field name="_id" type="string" indexed="true" stored="true" required="true" />
  <field name="title" type="string" indexed="true" stored="true" required="true" />
  <field name="summary" type="text" indexed="true" stored="false" required="false" />
  <!-- A completer -->

  <!-- Un champ dans lequel on concatene les autres pour une recherche "plein-texte" -->
  <field name="text" type="text" indexed="true" stored="false"
    multiValued="true" />
  <copyField source="summary" dest="text" />
  <copyField source="title" dest="text" />

      <!-- Un champ "technique" requis par Solr/Lucene -->
  <field name="_version_" type="long" indexed="true" stored="true" />
</fields>

<!-- La cle d'acces a un document dans l'index -->
<uniqueKey>_id</uniqueKey>

  <!-- Configuration des types de champ -->
<types>
  <fieldType name="string" class="solr.StrField" />
  <fieldType name="int" class="solr.IntField" />
  <fieldType name="long" class="solr.LongField" />
  <fieldType name="text" class="solr.TextField">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory" />
              <filter class="solr.LowerCaseFilterFactory" />
    </analyzer>
  </fieldType>
</types>
 </schema>
```

Squelette de l'index

- la liste des champs, dans l'élement "fields", complétée par l'indication du champ de recherche par défaut;
- le champ qui identifie le document Solr, dans l'élément "uniqueKey";
- la liste des **types de champ**, dans l'élément "types".

Note : Pour des besoins internes, tout schéma doit contenir un champ "_version_" défini comme ci-dessus.

Définition des types et de la clé

- Chaque type utilisé dans le schéma d'un index doit apparaître dans un des élements **fieldType** du fichier **schema.xml**
- Solr fournit tout un ensemble de types pré-définis qui suffisent pour les besoins courants; on peut associer des options à un type
- Les options indiquent d'éventuels traitements à appliquer à chaque valeur du type avant son insertion dans l'index
- ex: type **text**
  - on lui définit un "analyseur" "StandardTokenizerFactory"
  - se charge de découper le texte en **tokens** pour une recherche plein-texte (détails plus tard)
  - retenir : cela permet d'indexer chacun des mots, et donc de faire des recherches sur toutes les combinaisons de mots
- L'élément **uniqueKey** permet de rechercher un document dans l'index par sa clé. Indispensable, ne serait-ce que pour savoir qu'un document est indexé

Définition des champs

```
<field name="_id" type="string" indexed="true" stored="true"
      required="true" multiValued="false" />
```

- Les attributs de l'élement XML caractérisent le champ
- Le **nom** et le **type** sont les informations de base
- Ensuite, divers attributs (souvent optionnels) :
  - **indexed** indique simplement que le champ pet être utilisé dans une recherche;
  - **stored** indique que la **valeur** du champ est **stockée** dans l'index, et qu'il est donc possible de récupérer cette valeur comme résultat d'une recherche, **sans avoir besoin de retourner à la base principale**; en d'autres termes, "stored" permet de traiter l'index **aussi** comme une base de données;
  - **required** indique que le champ est obligatoire;
  - enfin, **multivalued** vaut **true** pour les champs ayant plusieurs valeurs, soit, concrètement, un **tableau** en JSON; c'est le cas par exemple pour le nom des acteurs.

Définition des champs

Les champs **indexed** et **stored** sont très importants

Toutes les combinaisons de valeur sont possibles :

- **indexed=true**, **stored=false**: on pourra interroger le champ, mais il faudra accéder au document principal dans la base documentaire si on veut sa valeur;
- **indexed=true**, **stored=true**: on pourra interroger le champ, **et** accéder à sa valeur dans l'index;
- **indexed=false**, **stored=true**: on ne peut pas interroger le champ, mais on peut récupérer sa valeur dans l'index;
- **indexed=false**, **stored=false**: n'a pas de sens à priori; le seul intérêt est d'ignorer le champ s'il est fourni dans le document Solr.

Définition des champs (suite)

Comment peut-on indexer un champ sans le stocker ?

- c'est notamment le cas pour les textes qui sont décomposés en **termes**: chaque terme est indexé indépendamment
- très difficile pour l'index de reconstituer le texte
- d'où l'intérêt de conserver ce dernier dans son intégralité, à part

C'est une question de compromis:

- **stocker** une valeur prend plus d'espace que **l'indexer**
- Dans la situation la plus extrême, on dupliquerait la base documentaire en stockant chaque document **aussi** dans l'index
- un stockage plus important dégrade les performances

Champ calculé

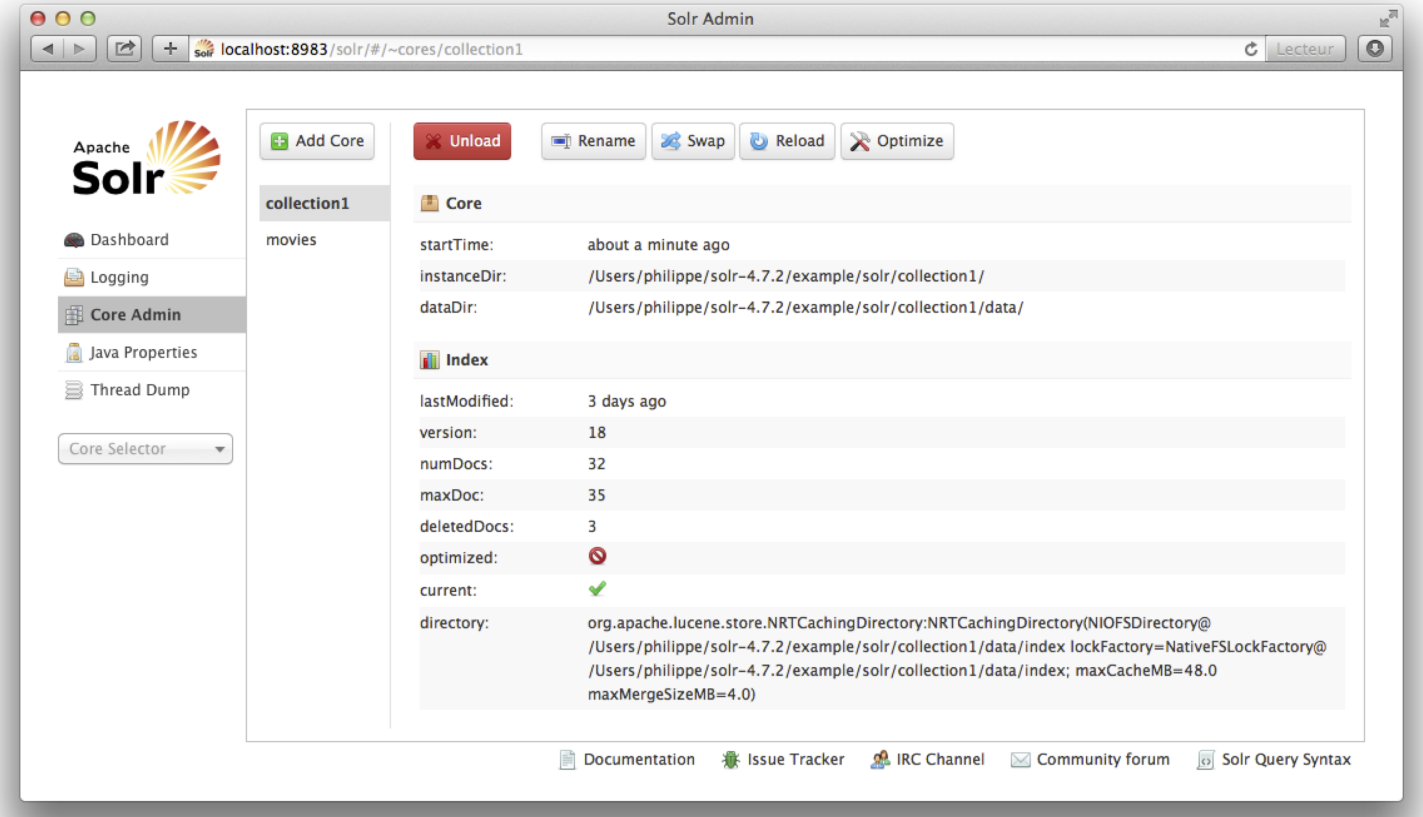
Le squelette de schéma comprend également un champ *calculé*, le champ **text**.

Les instructions **copyField** indiquent qu'au moment de l'insertion d'un document, on va "copier" certains champs dans celui-ci.

- le type du champ **destination** correspond à un mode particulier d'indexation, éventuellement différent et complémentaire de celui du champ **origine**;
  => par exemple le contenu d'un titre est indexé comme une chaîne de caractères dans le champ **title**, et comme un texte "tokenisé" quand on le copie dans le champ **text**;
- si **toutes** les occurrences de chaînes de caractères sont concaténées dans un même champ, on obtient, en prenant ce champ pour cible, une recherche plein-texte globale.

Recharger le schéma

- Après tout changement de schéma, il faut **recharger** l'index.
- Pour recharger un index, à partir de l'interface d'administration, utilisez l'option **Reload** après avoir sélectionné le **core**



Rechargement

```
curl "http://localhost:8983/solr/movies/update/json?commit=true" \\
--data-binary @solr_doc.json -H "Content-type:application/json"
```

- Attention, si l'index existant ne correspond pas au nouveau schéma, le rechargement échouera.
- Avec Solr, il est (plus) difficile de faire évoluer un schéma (qu'avec BDD classique)

Reconstruction (destruction puis validation) :

```
curl http://localhost:8983/solr/movies/update  \\
    --data "<delete><query>*:*</query></delete>" \\
    -H "Content-type:text/xml; charset=utf-8"
curl http://localhost:8983/solr/movies/update \\
  --data "<commit/>' -H 'Content-type:text/xml; charset=utf-8"
```

La semaine prochaine

- Le classement des résultats : TF-IDF, PageRank