Interface intelligente d’une BD

Interface utilisateur intelligente

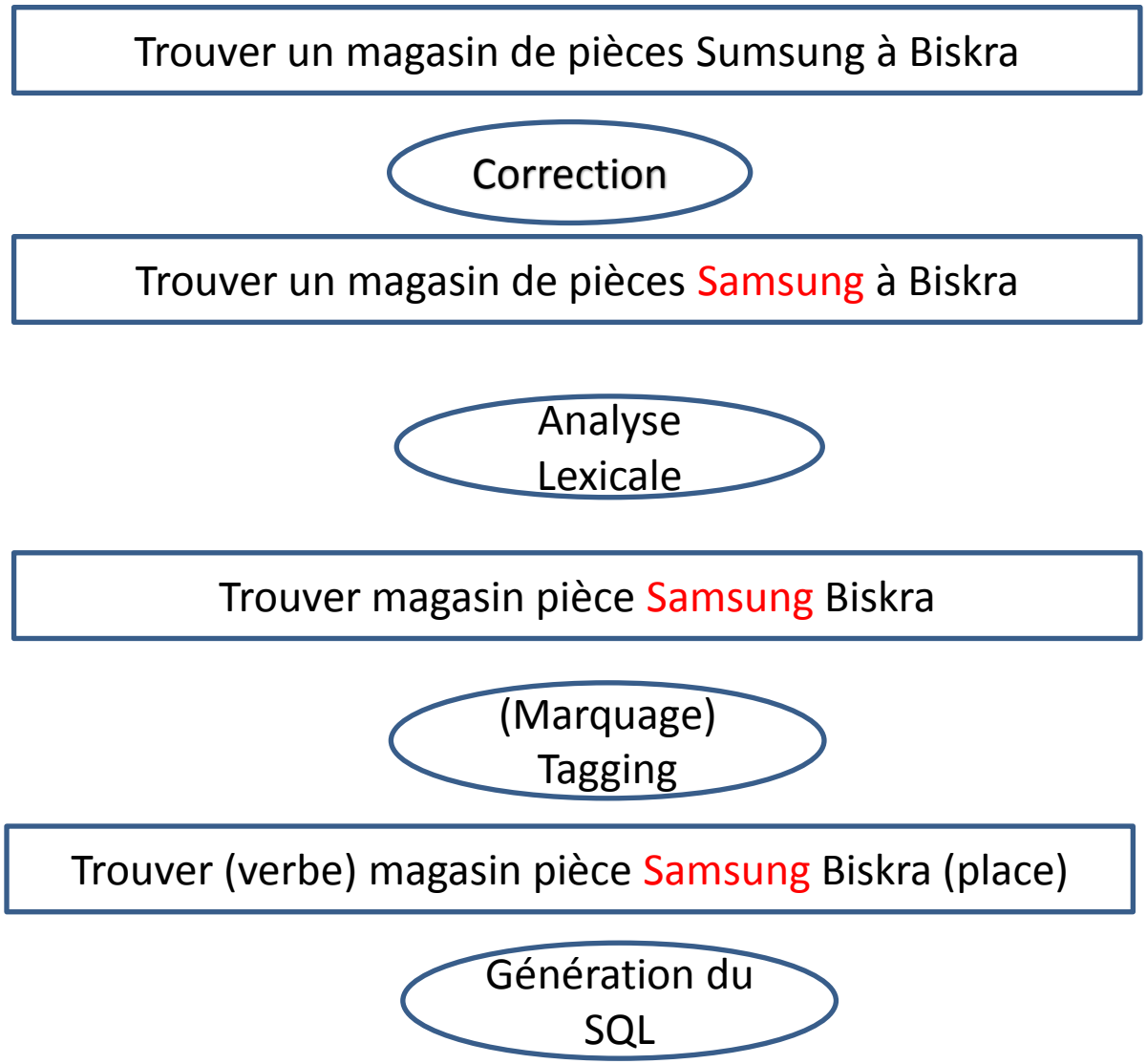
- Objectif

Transformer la demande de l'utilisateur formuler en langue naturelle vers une requête comprise par la BD

Ex dans le BD relationnel :

« Trouver un magasin de pièces Samsung à Biskra »➔SQL

Interface utilisateur intelligente

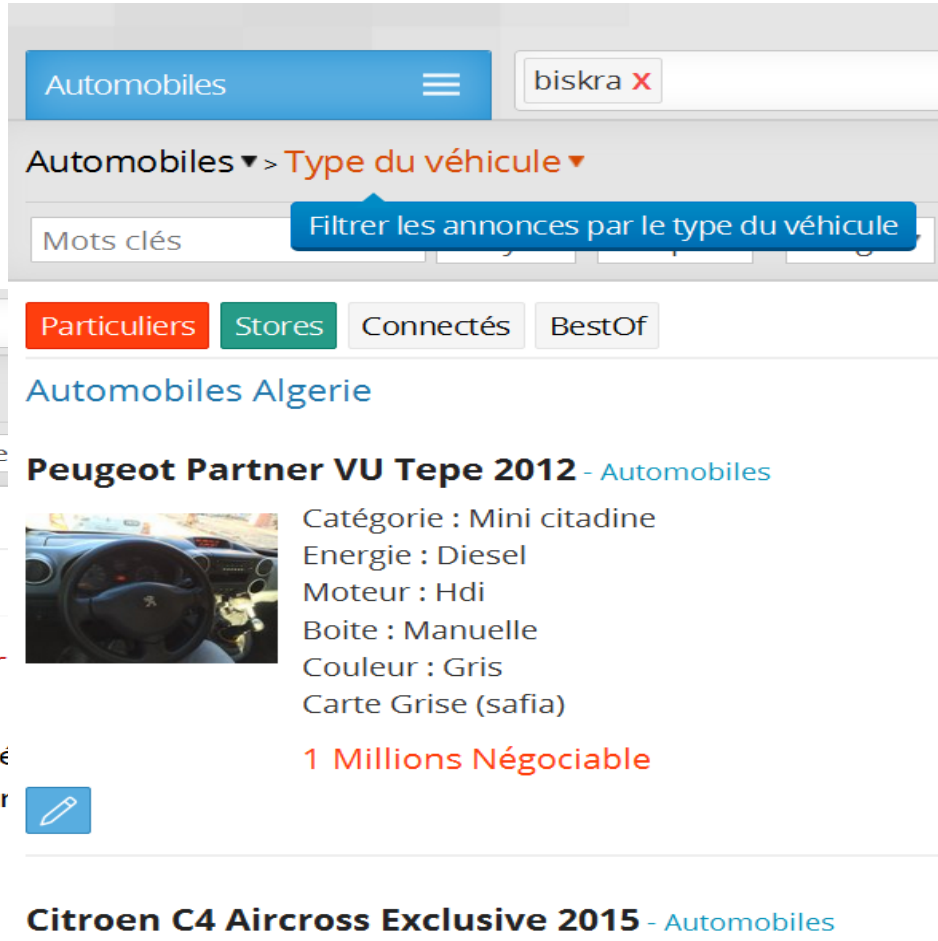


Interface utilisateur intelligente

- Correction des erreurs

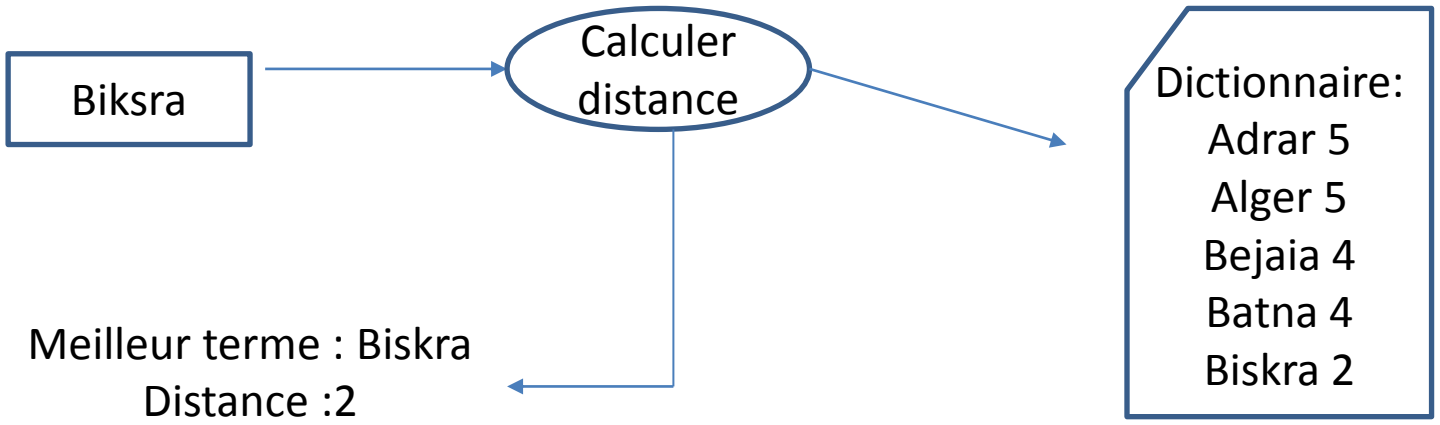
Rechercher Biksra

Voulez vous dire : Biskra ?



Interface utilisateur intelligente

- Correction des erreurs (**Spell Check)**



Interface utilisateur intelligente

- Correction des erreurs

  ➔ **Distance de Levenshtein**

donne par un calcul assez simple des indications sur le degré de ressemblance entre les chaînes

Distance(Biskra, Biskra)= 0 identique

Distance(Biksra, Batna)=4 loin

Distance(Biksra, Biskra)= 2 plus proche

Interface utilisateur intelligente

Distance de Levenshtein

		B	I	S	K	R	A
	0	1	2	3	4	5	6
B	1	0	1	2	3	4	5
I	2	1	0	1	2	3	4
S	3	2	1	0	1	2	3
K	4	3	2	1	0	1	2
R	5	4	3	2	1	0	1
A	6	5	4	3	2	1	0

		B	I	S	K	R	A
	0	1	2	3	4	5	6
B	1	0	1	2	3	4	5
I	2	1	0	1	2	3	4
K	3	2	1	1	1	2	3
S	4	3	2	1	2	2	3
R	5	4	3	2	2	2	3
A	6	5	4	3	3	3	2

Interface utilisateur intelligente

Distance de Levenshtein

Le minimum entre :

- le nombre à sa gauche +1
- le nombre au-dessus de lui +1
- le nombre en haut à gauche (en diagonale) +
  - 0 si les lettres à gauche et en haut sont identiques
  - 1 si ces lettres sont différentes

		B	I	S	K	R	A
	0	1	2	3	4	5	6
B	1						
I	2						
S	3						
K	4						
R	5						
A	6						?

0= Min((1+1),(1+1),(0+0(B=B)))

Interface utilisateur intelligente
Optimisation de la requête

- SELECT * FROM Personne WHERE Prénom = ‘Ali';

- Big Table ? > 1 000 000 Enregistrements !??

- Recherche séquentielle → full table scan

Interface utilisateur intelligente
Optimisation de la requête

- La table contient 1 000 000 Enregistrements
- Taille 1 enregistrement = 1200 octets
- Taille de bloc de disque dur est 4 KO
- → 3 enregistrement par bloc
- → 333 334 blocs (>1,4 G)
- Temps de Recherche !!!!!! (parcourir >300000 blocs) → 5ms /bloc =60 s

Interface utilisateur intelligente
Optimisation de la requête

- Optimiser une requête
  Si une requête a un temps de calcul trop long, on peut voir si il ne serait pas utile d'ajouter un index sur un des champs de la requête

Interface utilisateur intelligente
Optimisation de la requête

- Solution : Création d'index

- Un index est une structure de données utilisée par le SGBD pour lui permettre de retrouver rapidement les données

Interface utilisateur intelligente
Optimisation de la requête

- Index dans les livres
- Un index est construit sur une clé

- ballon, page 56
- ballon, page 80
- ballon, page 95
- bille, page 57, page 77, page 83
- bulle, page 65, page 66, page 88,
- câble, page 72

Interface utilisateur intelligente
Optimisation de la requête

- Un index est construit sur une clé
- Avantages

utiliser l'index pour des recherches :

1. par valeur de clé : ballon , bille
2. par *intervalle* des valeurs de clé : entre bille et câble
3. par *préfixe* de la valeur de la clé: les clés commençant par b

Interface utilisateur intelligente
Optimisation de la requête

- **Index Dans les BD**

- Un index est une table à deux champs.
- Le premier est le champ sur lequel porte l'index (clé).
- Le second est le numéro d'enregistrement correspondant dans la table(adresse).
- La table d'index est triée sur les valeurs du premier champ

Interface utilisateur intelligente
Optimisation de la requête

- Exemple
- Soit une table CLIENTS(numclient, nom, prénom, ...). Un index sur le champ CLIENTS.nom est une table :
-

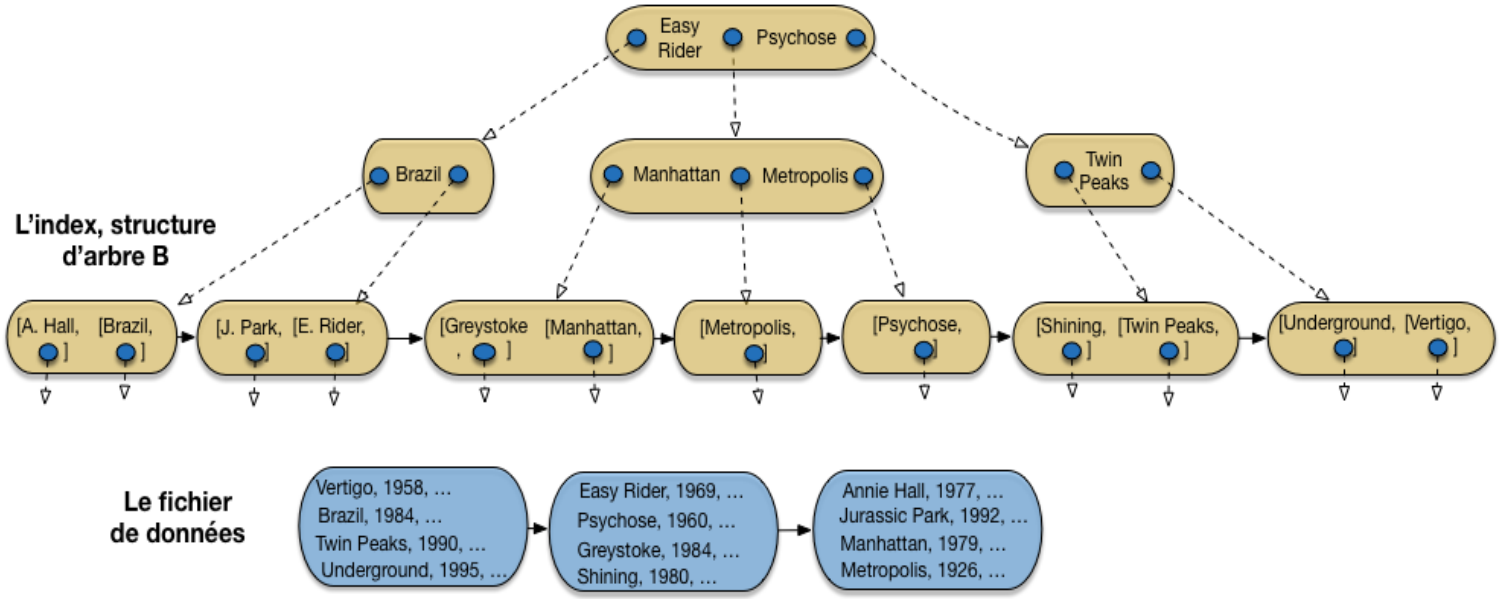
…….	…….
Ahmed	55 330
Ahmed	133 487
Ali	194 451
…….	

Interface utilisateur intelligente
Optimisation de la requête

- Quelques définitions sur l'index
  1. Une **clé** (d'indexation)
  2. Une **adresse** est un emplacement physique dans la base de données
  3. une **entrée** (d'index) est un enregistrement constitué d'une paire de valeurs

Interface utilisateur intelligente
Optimisation de la requête

- Utilisation de arbre-B



Interface utilisateur intelligente
Optimisation de la requête

- **Intérêts des index**

- Un index sur un champ permet un accès rapide sur les valeurs de ce champ
- le temps moyen d'accès à un élément est de N sans index
- Recherche par dichotomie avec (log2 N) avec un index → log2 (1000 000)=20
- 20 enregistrement → 7 bloc
- → 5ms *7 =35 ms

Interface utilisateur intelligente
Optimisation de la requête

- **Syntaxe**

CREATE INDEX ‘index_nom’ ON ‘table’;

CREATE INDEX `index_nom` ON `table` (`colonne1`);

- On peut créer un index sur un ou plusieurs attributs

Interface utilisateur intelligente
Optimisation de la requête

- Définir un index → sur quels champs ?

1. La table doit avoir un grand nombre de lignes
2. L'attribut doit avoir beaucoup de valeurs différentes
3. Pas tous les champs à cause du compromis temps-espace

Hyundai Accent 2014 - Automobiles

Catégorie : Berline
Energie : GPL
Moteur : 1.5 ess 91ch
Boite : Manuelle
Couleur : Gris Argent
Carte Grise (safia)

112000 km

accent gls 2014 , 00 peinture, sauf retouche sur capot voir photos , avec glp 60 l mémoire, offert 135 m

138 Millions Fixe

Interface utilisateur intelligente
Optimisation de la requête

- Définir un index → sur quels champs ?

4. Indexer les attributs servant aux jointures
5. Indexer les attributs qui interviennent dans les clauses WHERE

Interface utilisateur intelligente
Optimisation de la requête

- Problème

SELECT * FROM PRODUCTS
WHERE LOWER(title) like LOWER('%$phrase%')
OR LOWER(description) like LOWER('%$phrase%');

Title			
Apple iPhone 4G black 16GB			
…..			

Requête : “iPhone 16GB” → résultat ?

Interface utilisateur intelligente
Optimisation de la requête

Title
Apple iPhone 4G black 16GB

- → Solution “%”
  - « %iPhone %16GB% »
- Requête:

  “iPhone 16GB 4G”? → résultat ?

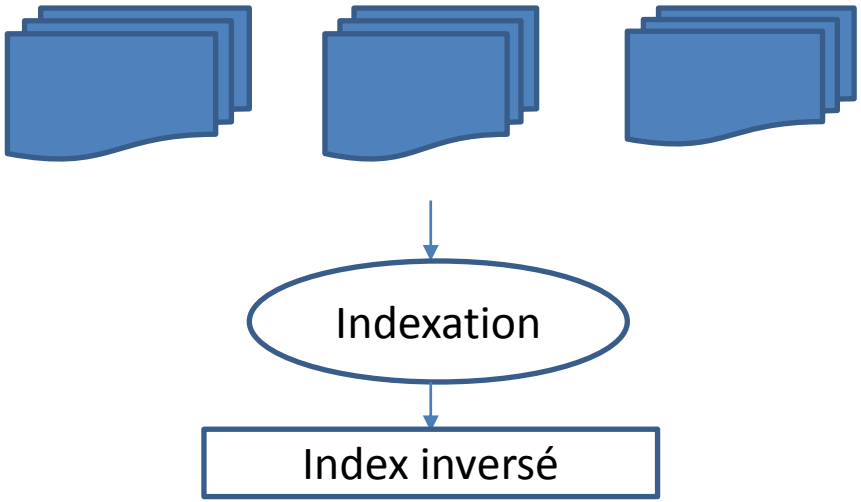
- Pb de recherche de texte
- Moteur de recherche

Optimisation de la requête

- Intégration de Moteur de recherche dans les BDs
  - Exemple Apache Solr
- Solr Basé sur Apache lucene
- Lucene permet d'indexer et de recherche dans les documents Textuels

Optimisation de la requête

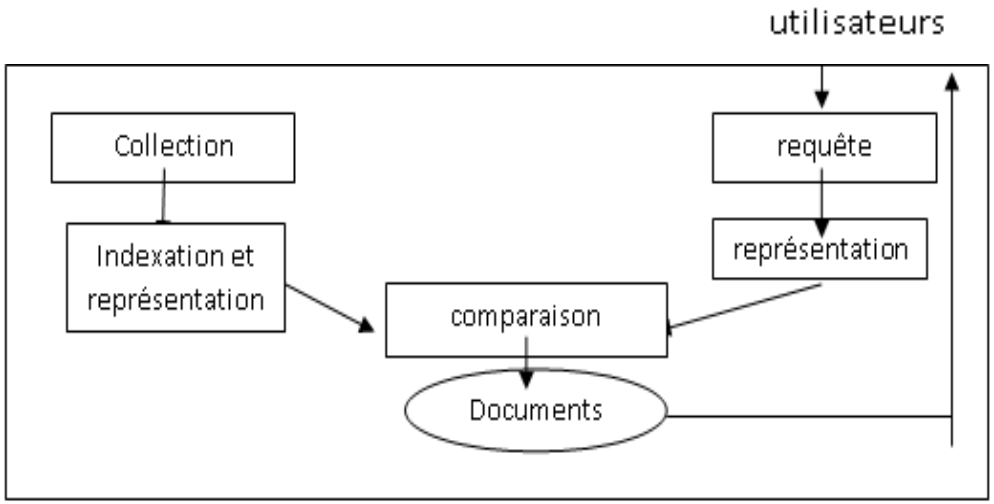
- Indexation de collection de Documents



La	Doc1, Doc2, Doc3, Doc4, Doc5,…..
De	Doc1, Doc2, Doc3, Doc4, Doc5,…..
Recherche	Doc2, Doc5
Information	Doc2, Doc5
Base	Doc3, Doc5

Optimisation de la requête

- Indexation et représentation



Optimisation de la requête

- Indexation et représentation

1. Analyse lexicale
2. L'élimination des mots vides (La, le, de, …….)
   - Une liste de Mots vides
   - Seuil d'occurrence dans la collection
3. Lemmatisation
   - «informé», «informer», «informe», « information » → informe
   - Algorithme de lemmatisation de Porter [POR 80].

Optimisation de la requête

- Indexation et représentation

4. Pondération des termes
   - Associer à chaque terme un poids qui représente son importance dans la collection de documents
   - Deux type de pondération: locale et globale
   - pondération locale : l'importance d'un terme dans un seul document → la fréquence TF
   - pondération globale: l'importance d'un terme dans toute la collection de documents
     - Elle permet de réduire l'importance des termes qui se trouvent souvent dans un document
     - IDF (Inverse of Document Frequency)

Optimisation de la requête

- Indexation et représentation
- Pondération des termes (Exemple)
- comment est calculé le TF*IDF de trois termes dans chaque document de la collection (10 documents dans la collection).
- IDF est calculé par la formule : (Log (N/n)) où
  - N représente le nombre total de documents (10 documents)
  - n est le nombre de documents qui contient le terme.

N° Document	Beaucoup	xml	recherche	tf*idf Beaucoup	tf*idf xml	tf*idf recherche	SOMME	classement
1	20	1	1	0,91515	0,2218487	0,301029996	1,438029	10
2	5	4	15	0,228787	0,887395	4,515449935	5,631632	2
3	26	12	2	1,189695	2,662185	0,602059991	4,45394	5
4	5	10	10	0,228787	2,2184875	3,010299957	5,457575	3
5	32	0	0	1,46424	0	0	1,46424	9
6	35	0	0	1,601512	0	0	1,601512	8
7	80	0	0	3,660599	0	0	3,660599	6
8	15	22	0	0,686362	4,2151262	0	4,901489	4
9	20	20	2	0,91515	4,436975	0,602059991	5,954185	1
10	0	9	0	0	1,9966387	0	1,996639	7
idf (Log (N/n))	0,045757	0,2	0,30103					

Optimisation de la requête

- On remarque que le terme "Beaucoup", malgré son TF élevé dans le document 3, mais son TF*IDF sera réduit par rapport au TF*IDF du terme "XML" dans le document 8.
-
- Les documents qui contiennent fréquemment les termes "XML" et "recherche" seront mieux classé que les documents qui contient seulement le terme "Beaucoup"

Optimisation de la requête

- Outils d’indexation
- Lucene
- Terrier ir
-

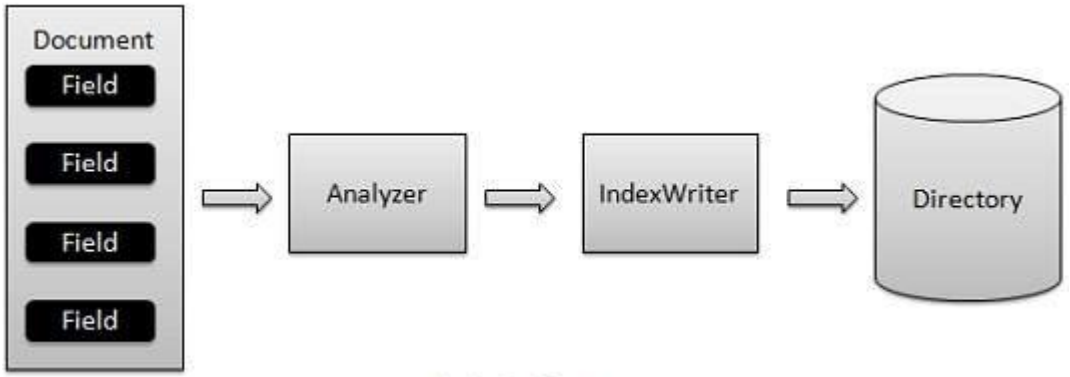
Indexation des Documents avec Lucene

- Lucene est une bibliothèque Java qui permet aux applications d'indexer et de rechercher un texte dans les documents.
- Lucene n'est pas une application, c.-à-d. elle ne peut pas fonctionner toute seule. Mais c'est un ensemble de classes et de méthodes qui sont utilisé dans des applications java.
- Plusieurs projets ont utilisés Lucene comme un outil puissant de recherche d'information. Parmi ces projets : LinkedIn, ifinder, blogdigger,

Indexation des Documents avec Lucene

- **Les classes de Lucene**

- **IndexWriter** est utilisé pour créer et maintenir des index
- **IndexSearcher** est utilisée pour rechercher dans un index
- La classe **Analyzer** est une classe abstraite qui est utilisé pour prendre un document et le transformer en termes qui peuvent être indexés.
- La classe **Document** représente un document dans Lucene. Les documents sont l'unité de l'indexation et de la recherche. Un document est un ensemble de champs (**Field**)



Indexing Process

Indexation des Documents avec Lucene

- **Les classes de Lucene**
- La classe **Field** est un champ qui représente une section d'un document. chaque champ a un nom et contient un texte comme données
- La classe **QueryParser** est utilisée pour construire un parseur qui peut analyser la requête pour chercher ensuite dans un index