



Le moteur de recherche SolR

david.thibau@gmail.com
2020

Agenda

- Introduction
- Installation
 - Exemple cloud
 - Création de coeur
- L'indexation de données
 - API Rest
 - SolRCell
 - Data Import Handler
- Recherche
 - Les différents parseurs
 - Syntaxe Lucene
 - Formule de boost
- Fonctionnalités de recherches
 - Vérification orthographique
 - Surbrillance
 - Recherche à facettes
 - Suggestion
 - Elévation
- Annexes

Problématique

« trouver de l'information
(*en général des documents*)
non-structurée

(*en général du texte*)
qui répond à une intention de
recherche, dans une large
collection
(*en général numérisée*) »



Search...

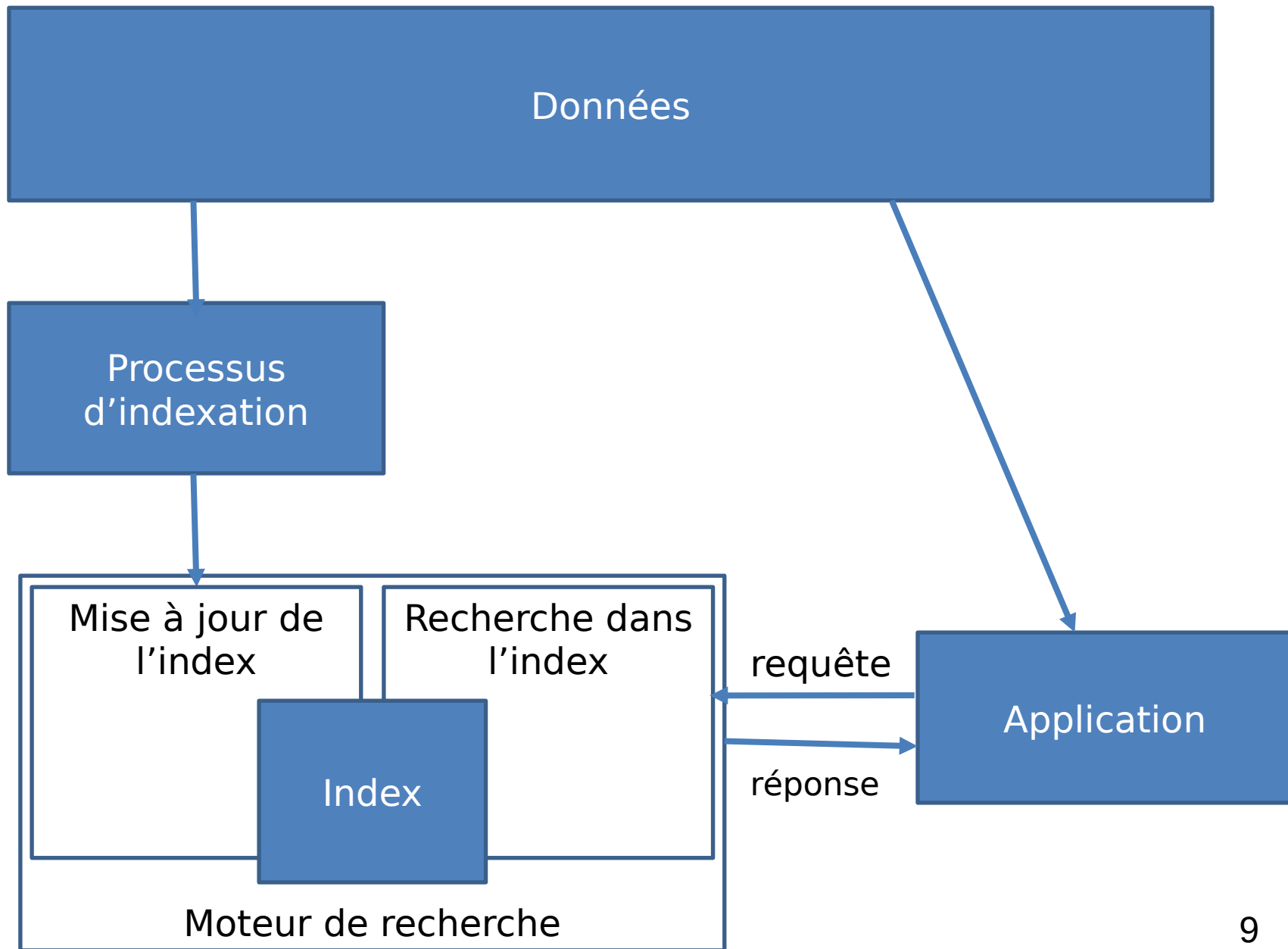
A horizontal search bar with a light blue border and rounded corners. On the left side, there is a magnifying glass icon. The central part is a white text input field containing the placeholder text "Search...". On the right side, there is a dark blue button with a white right-pointing arrow.

Implémentation SGBD ?

```
SELECT * FROM post  
WHERE  
    topic LIKE '%keyword%'  
    OR author LIKE '%keyword%'  
ORDER BY id DESC
```

Les SGBD ne sont pas adaptés pour faire de la recherche plein-texte

- Comment fait-on pour les recherches avec plusieurs termes ? sur plusieurs tables ?
- Full table scan = mauvaise performances
- Pas de scoring/pertinence des résultats
- Gestion des langues, des recherches approchantes (pluriels, féminins, etc.)

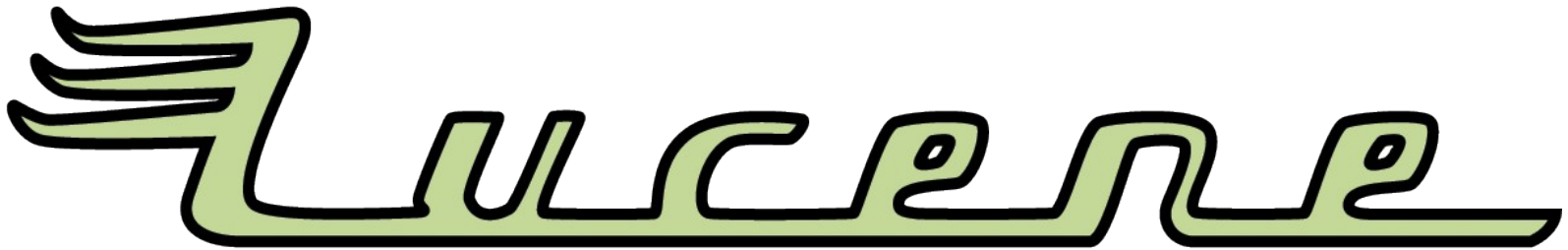


Critères pour une solution d'entreprise

- Rapide :
 - Temps d'attente
 - Suggestion au fur et à mesure
- Pertinent
 - Les meilleurs documents présentés en premier
- Flexible
 - S'adapter à la langue, au métier, ...
- Scalable
 - Charge utilisateur
 - Volumes de données



- Open-source, écrit en Java;
- Serveur de recherche autonome proposant une API Rest
- Basé sur Lucene, une librairie java de recherche plein-texte;



- La « couche basse » de SolR : une librairie Java pour écrire et rechercher dans les fichiers d'index;
- Un index contient des documents
 - Un document contient des champs (« fields »)
 - Un field contient des termes (« terms »)
- SolR expose les fonctionnalités de Lucene dans un serveur, au travers de HTTP;

SolR : historique

- Développé au sein de CNET par Yonik Seeley
- Transféré à la fondation Apache en 2006;
- Incubé jusqu'en 2007 : v1.2;
- Aujourd'hui, Lucene et SolR sont dans le même projet Apache, et relasés ensemble;
 - On est donc passé directement de SolR 1.4 à 3.0 pour que les n° de version correspondent
- Juin 2013 : v4.3.1
- Février 2015 : v5.x (mode standalone)
- Grosse communauté d'utilisateurs
- LucidWorks : support commercial

ELS vs SolR

- La société elastic propose **Elastic Search** qui est très proche de SolR en termes de fonctionnalités

	SOLR	ELS
Installation & Configuration	Documentation très détaillée	Simple et intuitif
Indexation/ Recherche	Orienté Texte	Texte et autres types de données pour les agrégations
Scalability	Cluster via ZooKeeper et SolRCloud	Nativement en cluster
Communauté	Importante mais stagnante	A explosé ces dernières années
Documentation	Très complète et très technique	Très complète, facile d'accès, bcp de tutoriaux

Et l'utilisateur alors ?

- SolR s'arrête à fournir un flux de réponse en JSON, XML
- SolR ne propose pas d'interface utilisateur de recherche
- Pour cela, on peut utiliser des librairies d'intégration en fonction du langage
- Voir <https://cwiki.apache.org/confluence/display/solr/IntegratingSolr>
 - Java : SolRJ
 - Javascript : ajax-SolR
 - Ruby :
 - PHP, .Net, etc...

Qu'est-ce qu'un index ?

	A	B
1	term	docs
2	pizza	3, 5
3	solr	2
4	lucene	2, 3
5	sourcesense	2, 4
6	paris	1, 10
7	tomorrow	1, 2, 4, 10
8	caffè	3, 5
9	big	6
10	brown	6
11	fox	6
12	jump	6
13	the	1, 2, 4, 5, 6, 8, 9

- Chaque document a un *id* et est associé à une liste de termes
- Pour chaque terme, on garde la liste des *id* de documents qui contiennent ce terme

Recherche plein-texte



solr

Submit Query

Reset

6 results found in 8 ms Page 1 of 1 <<>>

<http://thetechietutorials.blogspot.com/2011/06/how-to-build-and-start-apache-solr.html> [More Like This](#)

[Techie Tutorials: How to build and start Apache Solr admin app from source with Maven](#)

<http://thetechietutorials.blogspot.com/2011/06/how-to-build-and-start-apache-solr.html>

<http://thetechietutorials.blogspot.com/2011/07/updated-pom-for-building-and-starting.html> [More Like This](#)

[Techie Tutorials: Updated POM for building and starting Solr Admin App from Solr 3.3 source](#)

<http://thetechietutorials.blogspot.com/2011/07/updated-pom-for-building-and-starting.html>

<http://thetechietutorials.blogspot.com/2011/06/solr-and-nutch-integration.html> [More Like This](#)

[Techie Tutorials: Solr and Nutch Integration](#)

<http://thetechietutorials.blogspot.com/2011/06/solr-and-nutch-integration.html>

« SolR browse », interface de recherche d'exemple fournie par SolR

Surbrillance des résultats

Enter your keywords:

education

Search

☐ Search all languages

Abusive, Inaccurate 'Virginity Tests' Won't Help, Educating Children Will

News - Sep 14 2013

for high school girls emerge regularly in Indonesia, with **education** officials, politicians and ...

<http://www.hrw.org/news/2013/09/14/abusive-inaccurate-virginity-tests-won-t...>

Letter to Minister of Education Mr. Yuan Guiren

News - Jul 15 2013

Convention on the Rights of Persons with Disabilities to ensure an inclusive **education** system at all levels. ...

<http://www.hrw.org/news/2013/07/15/letter-minister-education-mr-yuan-guiren...>

China: Revise Disability Regulations for Education

News - May 18 2013

schools to develop individualized **educational** plans for students with disabilities. However, the revisions continue to reinforce a parallel system of segregated special **education** schools and do not remove the ...

<http://www.hrw.org/news/2013/05/18/china-revise-disability-regulations-educ...>

Human Right Watch : <http://www.hrw.org/>

Recherche à facettes

Document type

Publications & Research (2600)
 Economic & Sector Work (1303)
 Publications, Publications & Research (700)
 Publication (51)
 Journal Article (34)
 Working Paper (21)
 Economic and Sector Work (6)
 UNDP-Water & Sanitation Program (5)
 Accounting and Auditing Assessment (4)
 World Development Report (3)
 All Document types »

Keyword

ECONOMIC GROWTH (1682)
 INCOME (1410)
 HUMAN DEVELOPMENT (1312)
 RURAL AREAS (1278)
 HUMAN CAPITAL (1235)
 PRIVATE SECTOR (1202)



administration is improving, and large numbers ...

Title: Promoting Social Cohesion through Education : Case Studies and Tools for Using Textbooks and Curricula

Author: Roberts-Schweitzer, Eluned

Date: 2006

Abstract: Since 2003, the Civic Engagement, Empowerment, and Respect for Diversity (CEERD) program of the World Bank Institute has included a program on Education and Respect for Diversity. The ...



Title: Developing the Workforce, Shaping the Future : Transformation of Madagascar's Post-basic Education

Author: Bashir, Sajitha

Date: 2009

Abstract: Sub-Saharan African countries are increasingly recognizing the contribution of post basic education to economic growth and social development. However, policy makers in many poor ...



Title: Brazil : Higher Education Sector Study, Volume 1

Author: World Bank

Date: 2000-06-30

Abstract: Brazil has put significant resources into developing its higher education system over the past three decades. As a result, a system has evolved in which some institutions have achieved ...



Title: Survey of ICT and Education in Africa : Tunisia Country Report


Author: Hamdy, Amr

Suggestions et corrections

The screenshot displays the UNESCO IIEP search interface. On the left, a search bar contains the text "teach". Below the search bar, a dropdown menu lists ten suggestions: "teacher education", "teacher conditions of employment", "teaching profession", "teacher qualifications", "teachers", "teacher recruitment", "teacher effectiveness", "teacher supply", "teacher wages", and "teaching methods". Below the suggestions, there are three categories with checkmarks and counts: "INTERNATIONAL COOPERATION (429)", "EDUCATIONAL SCIENCES (454)", and "REFERENCE WORKS (18)". On the right, the search results are displayed. The title "Results (4078)" is shown in green. Below it, a "Results/page" dropdown menu is set to "10". The first result is titled "Professionalism in teaching" and "Le Professionnalisme d'enseignement", by Hargreaves, Andy; Lo, Leslie N., published in 2000, with page ranges p. 183-308 and p. 167-273. The second result is titled "The Secondary education".

UNESCO IIEP (International Institute for Educational Planning) : <http://plan4learning.iiep.unesco.org>

Recherche sur synonymes



Le site emploi de la logistique

Chauffeur poids lourd emploi chauffeur poids lourd

Emploi Logistique > Chauffeur Poids Lourd

▼ Métier

- ☒ Chauffeur PL (148)
- ☐ Acheteur industriel (1)
- ☐ Acheteur (31)
- ☐ Adjoint de direction (1)
- ☐ Affréteur (30)
- ☐ Agent administratif (30)
- ☐ Agent de distribution (1)
- ☐ Agent de planning (8)
- ☐ Agent de quai (28)
- ☐ Agent de réception quai (4)

▼ Type de Contrat

- ☐ CDI (13)
- ☐ CDD (5)
- ☐ INTERIM (128)
- ☐ STAGE (2)

▼ Expérience

- ☐ Débutant (0 à 1 an) (11)
- ☐ Junior (2 à 4 ans) (20)
- ☐ Confirmé (5 à 9 ans) (6)

▼ Niveau d'étude

- ☐ Autodidacte (1)
- ☐ CAP (2)

► Cargaison

Votre recherche - Tout effacer

Métier


X Chauffeur PL


Localité

<input type="checkbox"/> Alsace (5)	<input type="checkbox"/> Aisne - 02 - (1)	<input type="checkbox"/> Abbeville (1)
<input type="checkbox"/> Aquitaine (8)	<input type="checkbox"/> Allier - 03 - (1)	<input type="checkbox"/> Andrézieux-Bouthéon (1)
<input type="checkbox"/> Auvergne (4)	<input type="checkbox"/> Alpes-Maritimes - 06 - (2)	<input type="checkbox"/> Anse (1)
<input type="checkbox"/> Basse-Normandie (5)	<input type="checkbox"/> Aveyron - 12 - (1)	<input type="checkbox"/> Aulnay-sous-Bois (1)
<input type="checkbox"/> Bourgogne (7)	<input type="checkbox"/> Bas-Rhin - 67 - (2)	<input type="checkbox"/> Avignon (1)


Trier par : ☐ Date ☒ Pertinence

148 annonces actives de - de 30 jours


 **Ne ratez plus d'opportunités !**
Recevez les nouvelles annonces de cette recherche par e-mail

 **Conducteur PL/SPL grutier (H/F) Aulnay-sous-Bois** CDI

Île-de-France - Seine-Saint-Denis (93) - 93600 - Emploi GROUPE GT - 03/01/2014
: 182H/mois du lundi au samedi (prise de poste à 6h) GT « pour réussir ensemble » GT Ile de France Nord recrute pour soutenir son développement : **Conducteur...**

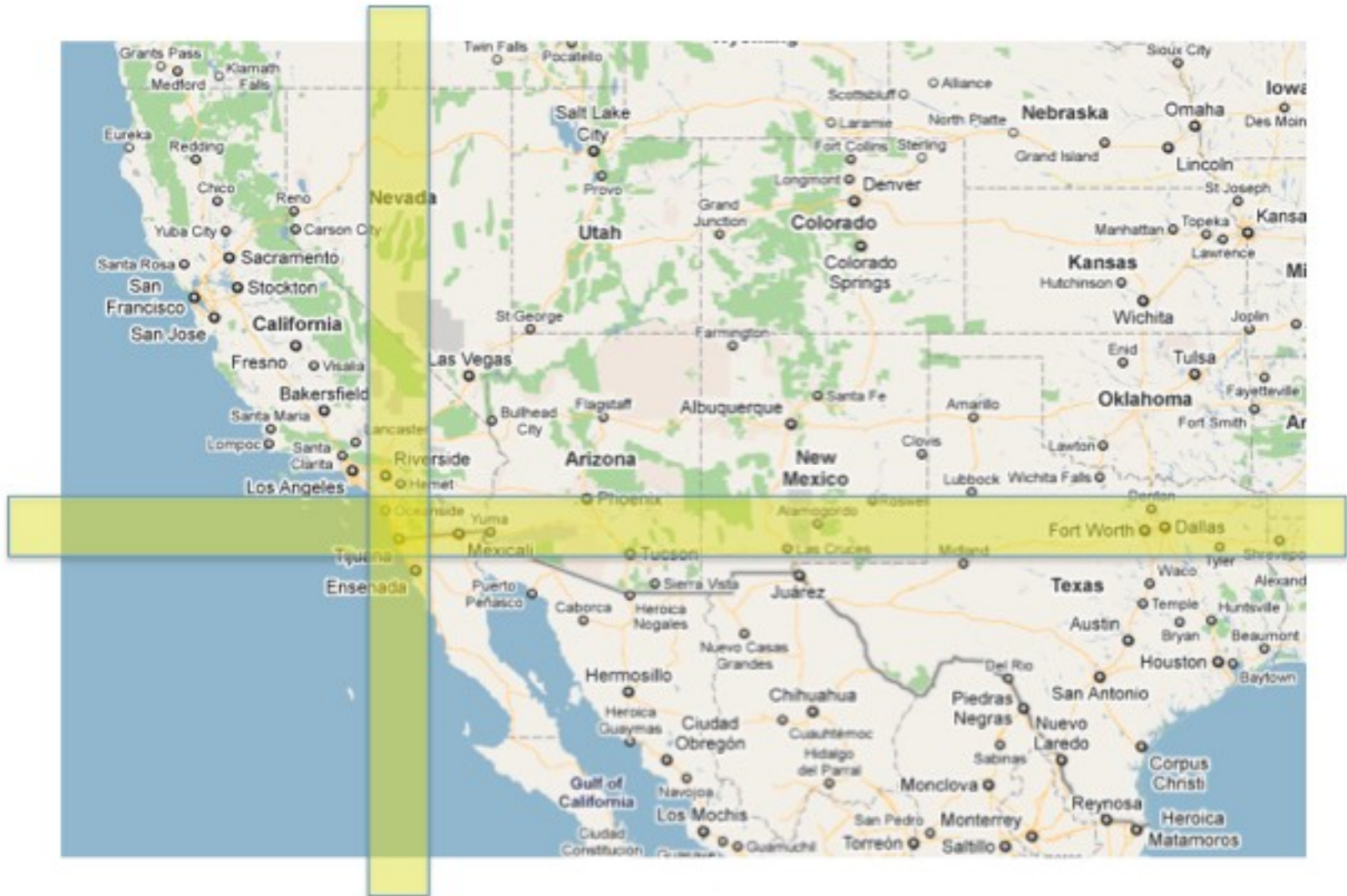
 **Conducteur PL grutier (H/F) Reims** CDI

Champagne-Ardenne - Marne (51) - 51100 - Emploi GROUPE GT - 03/01/2014
/ mois GT GT Nord recrute pour soutenir son développement : **Conducteur PL** grutier (H/F) Vous êtes titulaire du **Permis C**. Vous souhaitez travailler

 **Conducteur PL (H/F) Distribution frigorifique Vire** CDD

Basse-Normandie - Calvados (14) - 14500 - Emploi GROUPE GT - 03/01/2014
Travail du lundi au vendredi prise de poste à 6h GT GT Nord recrute pour soutenir son développement : **Conducteur PL** (H/F) Distribution frigorifique Vous...

Recherche spatiale



Interface d'admin

- SolR ne propose pas d'interface utilisateur mais offre différents moyen d'intégration
- Il offre une interface web d'administration qui permet :
 - Gérer et parcourir les index (simple noeud ou disribué) :
 - Tester les analyseurs
 - Effectuer des recherches
 - Indexer les données
 - Visualiser les fichiers de config
 - Modifier le Schema
 - Surveiller un serveur standalone
 - Surveiller un cloud et les noeuds qui le constitue

Solr admin



- Dashboard
- Logging
- Cloud
- Collections
- Java Properties
- Thread Dump

gettingstarted

- Overview
- Analysis
- Dataimport
- Documents
- Files
- Query
- Stream
- Schema

Core Selector

Use [original UI](#)

Collection: gettingstarted

Config name: gettingstarted
Max shards per 2
node:
Replication 2
factor:
Auto-add
replicas:
Router name: compositeld

Shards

shard1

Range: 80000000-ffffff
Active:
Replicas: gettingstarted_shard1_replica2
gettingstarted_shard1_replica1

shard2

Range: 0-7ffffff
Active:
Replicas: gettingstarted_shard2_replica2
gettingstarted_shard2_replica1

Documentation

Issue Tracker

IRC Channel

Community forum

Solr Query Syntax

Exemple d'interface SolR Browse

Find:

Envoyer

Effacer

☐ Boost by Price

Field Facets

cat

[electronics](#) (2)

[hard drive](#) (2)

manu_exact

[Maxtor Corp.](#) (1)

[Samsung Electronics Co. Ltd.](#) (1)

Query Facets

[GB](#) (2)

Range Facets

price

[50.0 - 100.0](#) (1)

[350.0 - 400.0](#) (1)

2 results found in 28 ms Page 1 of 1

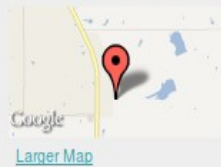
Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133 [More Like This](#)

Id: SP2514N

Price: 92,USD

Features: 7200RPM, 8MB cache, IDE Ultra ATA-133 NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor

In Stock: true



[Larger Map](#)

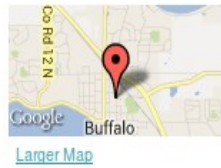
Maxtor DiamondMax 11 - hard drive - 500 GB - SATA-300 [More Like This](#)

Id: 6H500F0

Price: 350,USD

Features: SATA 3.0Gb/s, NCQ 8.5ms seek 16MB cache

In Stock: true



[Larger Map](#)

2 results found. Page 1 of 1

Options: [enable debug](#) [enable annotation](#) [XML](#)

TP1 : Installation + démo

- Télécharger, dézipper
- Lancer une configuration exemple:

```
cd <solr>
```

```
bin/solr start -e cloud -noprompt
```

- Indexer les documents de test

```
cd <solr>
```

```
bin/post -c gettingstarted docs/
```

- Rechercher

<http://localhost:8983/solr> (interface d'admin)

Autres exemples

- La distribution propose 4 exemples :
 - **cloud** : Permet de démarrer 1 à 4 nœuds en utilisant des *réplica* et des *shards*
 - **techproducts** : Mode standalone avec un schéma correspondant aux documents du répertoire *exampledocs*
 - **dih** : Mode standalone avec activation du *DataImportHandler* (contenu stocké en RDBMS ou autre)
 - **schemaless** : Mode standalone avec possibilité de créer des champs à la volée

Création de coeur

Script de démarrage

- Pour créer un cœur/collection, SolR doit être démarré

- Le script solr et la commande **start**

```
bin/solr start [options]
```

```
bin/solr start -help
```

```
bin/solr restart [options]
```

```
bin/solr stop [options]
```

```
bin/solr status
```

Options de démarrage

- **-c** : Mode cloud
- **-h** et **-p** : Host et port d'écoute
- **-d <dir>** : Répertoire du serveur par défaut : *server*
- **-z <zkHost>** : Mode cloud : Adresse de ZooKeeper
- **-m <memory>** : -Xms et -Xmx
- **-s <dir>** : *solr.solr.home* Répertoire home de la configuration
- **-t <dir>** : *solr.data.home*, Répertoire de dstockage des index
- **-e <example>** : cloud,techproducts,dih,schemaless
- **-a** : paramètres additionnels de la JVM
- **-v** et **-q** : Niveau de verbosité

Création de coeur

- La création d'un cœur s'effectue alors avec le script ***solr*** et la commande ***create***

```
bin/solr create options  
bin/solr create -help
```

La commande *create* détecte le mode d'exécution de SolrR et construit soit un cœur soit une collection (*SolrCloud*)

Options de *create*

- **-c <name>** (requis) : Le nom du cœur ou de la collection à créer
- **-d <confdir>** : Le répertoire de configuration.
 - Soit une valeur prédéfinie :
 - **_default**: Configuration minimale avec détection automatique de champs
 - **sample_techproducts_configs**: Configuration avec les fonctionnalités optionnelles de l'exemple *techproduct*
 - Soit un chemin vers une configuration spécifique contenant un schéma

Autres commandes liées aux coeurs

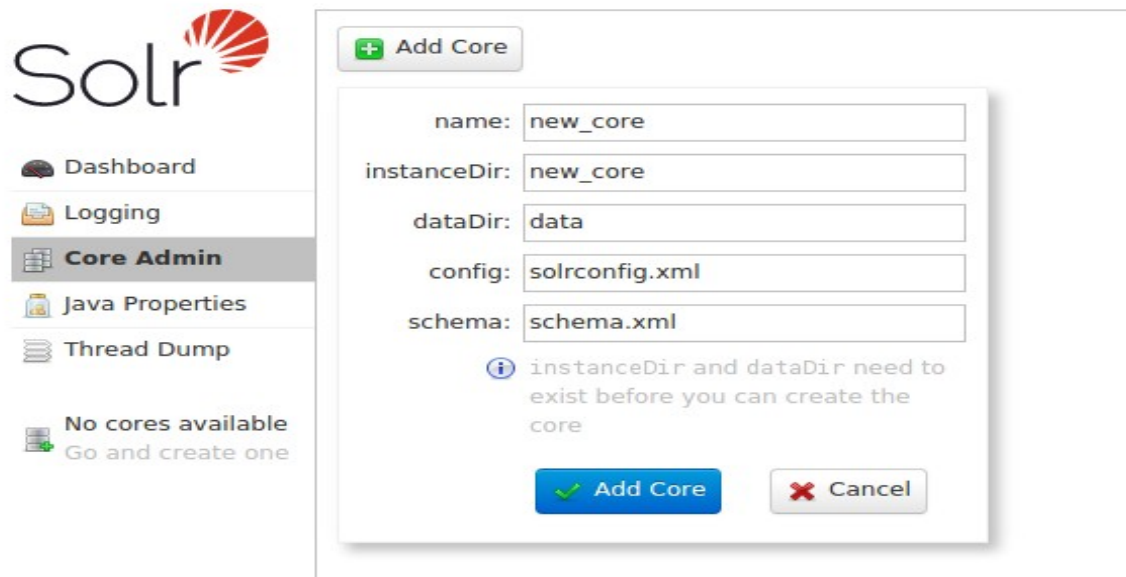
- *Solr* propose 2 autres commandes pour la gestion des coeurs

```
bin/solr healthcheck -c <name>
```

```
bin/solr delete -c <name>
```


Interface d'administration

- L'interface d'administration permet également de créer un cœur
- Il faut cependant avoir préalablement créer les répertoires de configuration et de données avec les fichiers *solrconfig.xml* et *schema.xml*
-



The screenshot displays the Solr Admin interface. On the left is a sidebar with the Solr logo and navigation links: Dashboard, Logging, Core Admin (highlighted), Java Properties, Thread Dump, and a message 'No cores available Go and create one'. The main area shows a modal dialog titled 'Add Core'. The dialog contains five input fields: 'name' (new_core), 'instanceDir' (new_core), 'dataDir' (data), 'config' (solrconfig.xml), and 'schema' (schema.xml). Below these fields is an information icon and a message: 'instanceDir and dataDir need to exist before you can create the core'. At the bottom of the dialog are two buttons: 'Add Core' (with a green checkmark) and 'Cancel' (with a red X).

API Rest

- Les 2 méthodes utilisent l'API Rest :
- `http://localhost:8983/solr/admin/cores?action=CREATE&name=formation&instanceDir=formation`

Atelier 2.1

Création de coeurs

- Créer 1 cœur :
 - “*formation*” en utilisant la configuration prédéfinie ***_default***
- Accéder à l'interface d'administration et vérifier le schema avec le schéma browser
- Regarder les fichiers de configuration créés

Configuration d'un coeur

Fichiers de configuration

Pour un *solr.home* donné (par défaut : *server*) :

Configuration du serveur :

`<solr.home>/solr.xml`

Données d'un coeur :

`<solr.home>/<core>/data`

Configuration d'un coeur:

`<solr.home>/<core>/conf`

Configuration : gestionnaires HTTP et schéma

- L'API REST de Solr est fournie par des « **gestionnaires de requêtes** » (**request Handler**) de différents types :
 - Parseur de requête, Indexeurs, ...
 - Ils sont configurés dans
<solr.home>/<core>/conf/solrconfig.xml
- L'autre aspect de la configuration consiste à définir la structure de l'index et en particulier à spécifier le comportement des différents champs des documents.
- Par défaut, ceci est décrit dans :
<solr.home>/<core>/conf/managed-schema
- Cela peut être modifié

Fichiers de configuration du schéma

- Les informations de schéma peuvent être dans différents fichiers en fonction de comment on a créé le core :
 - ***managed-schema*** nom par défaut lorsque le schéma n'est mis à jour que via l'API ou par les documents indexés (schemaless)
Dans ce cas, seul Solr met à jour le fichier
 - ***schema.xml*** : L'utilisateur édite et met à jour le fichier. Pour activer ce mode :

```
<schemaFactory class="ClassicIndexSchemaFactory"/>
+
<updateRequestProcessorChain name="add-unknown-fields-to-the-schema" default="{update.autoCreateFields:false}"
```
 - Dans la config cloud, le schéma n'est éditable que par l'API

Balises schema

- Balises
 - ***field***, ***dynamicField***, ***uniqueKey*** :
définissent un champ d'un document
 - ***copyField*** : duplique automatiquement les
valeurs d'un champ dans un autre
(concaténation ou type différent)
 - ***fieldType*** : déclare un type de champs
possible pour un « *field* »
Si champ full-text est associé à un ou
plusieurs « analyzer »
 - ***analyzer*** : définit les traitements qui seront
appliqués aux valeurs du *field*, à l'indexation
et à la recherche

Attributs de *<field>*

name : son nom;

type : son type, une référence à *fieldType*;

multivalued : si un document peut avoir plusieurs valeurs pour ce champ;

required : si la valeur est obligatoire

indexed : si on veut pouvoir chercher ou trier sur les valeurs de ce champs;

stored : si on veut ramener les valeurs de ce champs dans un résultat de recherche;

docValues="true" : Si on veut trier ou grouper sur un champ. (Compatible avec certains types de champs)

fieldType

name : son nom (référéncé dans un field);

class: sa classe Java (1 classe = 1 type de données)

(dans ce fichier, « solr... » est un raccourci pour « *org.apache.solr.analysis* »)

- Quelques types de données :
 - ***solr.BoolField*** : booléen
 - ***solr.IntPointField***/***solr.LongPointField*** : entiers
 - ***solr.FloatField*** décimaux
 - ***solr.DatePointField***, ***solr.DateRangeField***: date
 - ***solr.LatLonPointSpatialField*** : latitude/longitude
 - ***solr.CurrencyFieldType*** : Monnaie
 - ***solr.TextField*** : texte

Attributs *required* et *multiValued*

- En dehors de son type, les 2 premiers attributs d'un champ sont :
 - ***required*** : Un document doit obligatoirement avoir ce champ renseigné lors de l'indexation
 - ***multiValued*** : Le champ peut contenir plusieurs valeurs (tableau)

Attributs : *indexed* et *stored*

On peut avoir des champs qui ne servent qu'à rechercher sans jamais être ramenés dans un résultat de recherche : ***indexed***

Inversement, on peut avoir des champs qui ne servent qu'à être ramenés dans un résultat de recherche et sur lesquels on ne cherchera jamais : ***stored***

Balise *<uniqueKey>*

- La balise *<uniqueKey>* permet de préciser le champ qui sert de clé pour ce schéma

`<uniqueKey>id</uniqueKey>`

- Cette balise n'est pas requise mais recommandée

Balises *dynamicField*

- La balise ***dynamicField*** permet d'affecter un type à un champ en fonction de son nom

– Ex :

```
<dynamicField name="*_num" type="pdouble" indexed="true"
stored="true" multiValued="false" />
```

=> Tous les champs qui seront terminés par le suffixe *_num* seront de type *pdouble*

Balises *copyField*

- La balise ***copyField*** permet de dupliquer certaines valeurs dans certains champs afin de pouvoir appliquer différents analyseurs et donc proposer plusieurs types de recherche
 - Ex : copier une chaîne vers un *field*
« phonétique » sur lequel portera la recherche phonétique

Champs spéciaux

- Un index comporte généralement les champs suivants :
 - ***id*** : Identifiant du document
 - ***__version__*** : Identification de la version du document.
(Un document est immuable ce champ est incrémenté à chaque mis à jour)
 - ***__root__*** : Nécessaire si l'on veut utiliser les nested documents
 - ***__text__*** : Réceptacle pour tous les champs “searchable”

fieldType

name : son nom (référéncé dans un field);

class: sa classe Java (1 classe = 1 type de données)

(dans ce fichier, « solr... » est un raccourci pour « *org.apache.solr.analysis* »)

- Quelques types de données :
 - ***solr.BoolField*** : booléen
 - ***solr.TrieIntField***/***solr.TrieLongField*** : entiers
 - ***solr.TrieFloatField*** : décimaux
 - ***solr.TrieDateField*** : date
 - ***solr.LatLonType*** : latitude/longitude
 - ***solr.CurrencyField*** : monnaie
 - ***solr.TextField*** : texte

Indexation

Principe des analyseurs

API d'indexation

Recommandations pour la configuration

Analyseurs


Introduction

- SolR utilise 3 concepts pour traiter le texte des documents :
 - Les **analyseurs** de champs sont utilisés à l'indexation et lors de la recherche. Ils transforment un texte en un flux de “token”. Ils sont en général constitué de :
 - Les **filtres de caractères** effectuant du remplacement de caractères (& devient et) ou en supprimant (suppression des balises HTML)
 - Un **tokenizer** : Responsable de splitter un texte en token ou terme
 - Les **filtres** prennent en entrée un flux de token et le transforme en un autre flux de token

Analyseurs définis

- Lors d'une création de coeur, SolR créé par défaut de nombreux types de données associés à des analyseurs dédiés à un usage.
- Les plus utiles sont :
 - ***text_general*** : Le meilleur choix lorsque le champ est dans des langues diverses. Il consiste à :
 - Séparer le texte en mots
 - Supprime la ponctuation
 - Supprimer certains mots (*stopword*)
 - Passe tous les mots en minuscule
 - ***text_gen_sort*** : Idema avec des capacités de tri
 - **Analyseurs de langues** : *text_en*, *text_fr*, Ce sont des analyseurs spécifiques à la langue. Ils incluent les « stop words » (enlève les mots les plus courant) et extrait la racine d'un mot. C'est le meilleur choix le champ est en une seule langue

L'interface d'analyse



- Dashboard
- Logging
- Core Admin
- Java Properties
- Thread Dump
- collection1
- Overview
- Ping
- Query
- Schema
- Config
- Replication
- Analysis**
- Schema Browser
- Plugins / Stats
- Dataimport

Field Value (Index)

Une formation débutant pour SolR.

Field Value (Query)

Analyse Fieldname / FieldType:

text_fr

?

☒ Verbose Output

Analyse Values

ST	text	Une	formation	débutant	pour	SolR
	raw_bytes	[55 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[53 6f 6c 52]
	start	0	4	14	23	28
	end	3	13	22	27	32
	type	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>
	position	1	2	3	4	5
EF	text	Une	formation	débutant	pour	SolR
	raw_bytes	[55 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[53 6f 6c 52]
	position	1	2	3	4	5
	start	0	4	14	23	28
	end	3	13	22	27	32
	type	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>
LCF	text	une	formation	débutant	pour	solr
	raw_bytes	[75 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[73 6f 6c 72]
	position	1	2	3	4	5
	start	0	4	14	23	28
	end	3	13	22	27	32

Atelier :

Comprendre l'analyse

- Testez l'analyse sur la chaîne « Une formation débutant sur SolR »
 - Avec le type de champ *text_ws*
 - Avec le type de champ *text_general*
 - Avec le type de champ *text_fr*
 - Que constatez-vous ?
 - Lisez les commentaires associés à ces types de champs dans *schema.xml*
- Essayez avec :
 - « Overview of Documents, Fields, and Schema Design » et le champ phonétique *english*

Un analyzer

Mission : analyser les valeurs texte, soit au moment de l'insertion d'une valeur, soit au moment de la recherche d'une valeur.

Un *fieldType* de classe *solr.Text* a :

- Un analyzer pour le texte indexé :
type="index"
- Un analyzer pour le texte de la query
type="query"

Si un seul analyzer est paramétré, il est utilisé pour l'indexation **et** la recherche

Exemple analyzer simple classe pour l'indexation et la recherche

```
<fieldType name="nametext" class="solr.TextField">  
  <analyzer class="org.apache.lucene.analysis.WhitespaceAnalyzer"/>  
</fieldType>
```

Un analyzer

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
```

```
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
```

```
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" />
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
```

```
</fieldType>
```

Phases d'analyse

L'analyseur est utilisé lors de l'indexation et la recherche

- Indexation : Le flux de tokens résultants est ajouté à l'index et définit l'ensemble des termes (incluant la position, la taille, etc.) pour le champ

- Recherche : Les valeurs de la recherche sont analysées et transformées en un flux de token qui sont comparés à ceux de l'index

En général, on utilise le même analyseur pour l'indexation et la recherche pour un champ donné

Un tokenizer

Mission : découper la chaîne de caractères en tokens ou termes

Certains caractères d'entrée peuvent être supprimés (ex. Espace, tabulation), d'autres peuvent être remplacés ou ajoutés (ex. abréviation)

Des méta-données sont ajoutées à chaque token (ex. La position)

Tokenizer : Quelques possibilités

- ***WhitespaceTokenizer***

- Découpe sur les espaces, tabulations, sauts de ligne

```
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
```

- ***StandardTokenizer***

- Espaces et ponctuation. Marche pour toutes langues européennes. A utiliser par défaut.

```
<tokenizer class="solr.StandardTokenizerFactory"/>
```

- ***KeywordTokenizer***

- Aucune tokenization ! Utile pour les valeurs à stocker telles quelles

```
<tokenizer class="solr.KeywordTokenizerFactory"/>
```

- ***PatternTokenizerFactory***

- Découpe en fonction d'une expression régulière

Un filter

- Mission : prendre un flux de *token* en entrée, retourner un autre flux de token
- Les filtres sont en général chaînés et l'ordre a une importance

Filtres communs

- ***LowerCaseFilterFactory***
 - Met tout en minuscule. A utiliser quasi-systématiquement, à l'index ET à la query
- ***LengthFilterFactory***
 - Pour ne garder que les tokens d'une certaine taille
- ***PatternReplaceFilterFactory***
 - Pour faire du rechercher-remplacer dans les tokens

Filter : Elision

- Elision : effacement d'une voyelle.
Le filtre supprime l'article et l'apostrophe
- Utile pour le français, le catalan, l'italien et l'irlandais

```
<filter class="solr.ElisionFilterFactory"  
ignoreCase="true" articles="lang/contractions_fr.txt"/>
```

– Exemple :

L'histoire de l'art => histoire de art

Filter : Stopwords

```
<filter class="solr.StopFilterFactory"
ignoreCase="true" words="stopwords.txt"
enablePositionIncrements="true" />
```

- Format du fichier : un terme par ligne

a
à
et
un

- ***solr.KeepWordFilterFactory*** : inverse de *stopWords* (ne garde que les termes spécifiés)

Filter : Stemming

- Stemming : ramener les formes fléchies à un radical
 - Pluriels, féminins, conjugaisons
 - « cheval », « chevaux » => « cheval »
 - « portera », « porterait » => « porte »
- Plusieurs algorithmes possibles :
 - ***FrenchLightStemFilterFactory*** : Défaut
 - ***FrenchMinimalStemFilterFactory*** : Moins de contraction
 - ***SnowballPorterFilterFactory*** : Plus de contraction

Filter : Synonyms

```
<filter class="solr.SynonymFilterFactory"  
synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
```

- Le remplacement de synonyme peut se faire de 3 façons :
 - Expansion simple : Si un des termes est rencontré, il est remplacé par tous les synonymes listés
"jump,leap,hop"
 - Contraction simple : un des termes rencontré est remplacé par un synonyme
"leap,hop => jump"
 - Expansion générique : un terme est remplacé par plusieurs synonymes
"puppy => puppy,dog,pet"

Synonymes :

index-time ou query-time ?

- Les synonymes peuvent être utilisés au moment de l'indexation ou au moment de la query
- Il est conseillé de les utiliser au moment de l'indexation
 - question de performance
 - problèmes liés aux synonymes comportant plusieurs mots à la query

Recherche phonétique

```
<filter class="solr.DoubleMetaphoneFilterFactory"  
inject="false"/>
```

- A mettre à l'index ET à la query
- Plusieurs algorithmes possibles
 - *Caverphone*, *Metaphone*, *DoubleMetaphone*, etc.
 - *DoubleMetaphone* donnerait de meilleurs résultats même en dehors de l'anglais
- Attention, peut donner des résultats hasardeux
- A utiliser dans un champ dédié

Filtres de caractères

- Les filtres de caractères traitent des caractères en entrée, ils peuvent être chaînés comme les filtres de token
- Ils peuvent ajouter, supprimer ou changer des caractères tout en préservant l'offset original des caractères pour supporter la surbrillance
 - ***solr.MappingCharFilterFactory*** : Basé sur un fichier de correspondance
 - ***solr.HTMLStripCharFilterFactory*** : Supprime les balises HTML
 - ***solr.ICUNormalizer2CharFilterFactory*** : Normalisation Unicode avec icu4j
 - ***solr.PatternReplaceCharFilterFactory*** : Utilisation d'expression régulières

Indexation

Introduction

- L'indexation consiste à ajouter du contenu à l'index SolR et éventuellement en modifier ou en supprimer
- L'indexation s'effectue via **l'API REST**, cela consiste à poster des données via des requêtes HTTP utilisant les formats XML, JSON ou CSV
- Pour indexer, Apache SolR fournit également
 - l'outil en ligne de commande *post*
 - L'interface d'administration
 - Les API clientes propres à chaque langage
 - Pour Java : *Solrj* et sa classe *SolrClient*

Plugins

- SolR propose 2 plugins :
 - ***Solr Cell*** basé sur Apache Tika pour ingérer des fichiers binaires ou structurés comme des documents Offices, des PDF ou autres
 - ***DataImportHandler*** pour aspirer du contenu à partir d'un support persistant (BD ou autre)

Index Handler

- Par défaut, SolR configure un `updateRequestHandler` capable de supporter les formats XML, CSV et JSON

```
<requestHandler name="/update"  
class="solr.UpdateRequestHandler" />
```

Le format XML : add

```
<add overwrite="true">
  <doc>
    <field name="id">5432a</field>
    <field name="type">Album</field>
    <field name="name">Murder Ballads</field>
    <field name="artist">Nick Cave</field>
    <field name="release_date">2012-07-31T09:40:00Z</field>
  </doc>
  <doc boost="2.0">
    <field name="id">myid</field>
    <field name="type">Album</field>
    <field name="name">Ilo veyou</field>
    <!-- etc. -->
  </doc>
</add>
```

Le format XML

- ***overwrite***

- Basé sur le champ *uniqueKey*
- Mettre à *false* si on est sûr de n'envoyer que des nouveaux record

- ***boost***

- Le document sortira plus haut dans les résultats que les autres

Le format XML : delete

```
<delete>  
  <id>5432a</id>  
  <id>monId</id>  
</delete>
```

```
<delete>  
  <query>Artist:"Nick Cave"</query>  
</delete>
```

```
<delete>  
  <query>*:*</query>  
</delete>
```

- Pour supprimer tout l'index, il faut mieux supprimer le répertoire *data* et relancer *So/R*

Le format XML : commit

```
# Ecriture des nouveaux documents (en mémoire) sur le disque
<commit />
# Annulation des derniers ordres d'indexation
<rollback />
# Commit + fusion segments Lucene
<optimize />
```

- Commits :
 - Lents : donc faire un seul gros commit à la fin
 - Pas de transactions par clients (commit global)
 - Mode auto-commit dans *solrconfig.xml*
 - Tant que les documents ne sont pas commités, ils ne sont pas searchable
- Optimize : committe et optimise l'index
- Aucune de ces opérations ne bloque la recherche
- Un *commit* ou un *optimize* peuvent se faire via une recherche via la query string : *?commit=true*

Transformation XSL

- En utilisant le paramètre **tr**, il est possible d'appliquer une transformation XSL au document d'origine

Le feuille de style XSLT doit être dans le dossier *conf/xslt*

Exemple :

```
curl  
"http://localhost:8983/solr/my_collection/update  
?commit=true&tr=updateXml.xsl"  
-H "Content-Type: text/xml" --data-binary  
@myexporteddata.xml
```

Opérations de groupe

Il est possible en un seul requête de faire plusieurs opérations :

```
<update>
  <add>
    <doc><!-- doc 1 content --></doc>
  </add>
  <add>
    <doc><!-- doc 2 content --></doc>
  </add>
  <delete>
    <id>0002166313</id>
  </delete>
</update>
```


Exemple curl

- L'utilitaire curl et son option *--data-binary* peut être utilisé pour les commandes d'indexation

```
curl http://localhost:8983/solr/my_collection/update -H "Content-Type: text/xml"
--data-binary '
<add>
<doc>
<field name="authors">Patrick Eagar</field>
<field name="subject">Sports</field>
<field name="dd">796.35</field>
<field name="isbn">0002166313</field>
<field name="yearpub">1982</field>
<field name="publisher">Collins</field>
</doc>
</add>'
```

Exemples curl

- En passant un fichier

```
curl http://localhost:8983/solr/my_collection/update -H "Content-Type: text/xml" --data-binary @myfile.xml
```

- En passant un gros fichier

```
curl http://localhost:8983/solr/my_collection/update -H "Content-Type: text/xml" -T "myfile.xml" -X POST
```

- Si configuration adéquate (*stream.body=true*), une requête GET est autorisée

```
curl http://localhost:8983/solr/my_collection/update?stream.body=%3Ccommit/%3E
```

Format de réponse

La réponse fournit 2 informations :

- Status : ==0 OK, !=0 NOK
- Le temps de traitement en ms

```
<response>  
<lst name="responseHeader">  
<int name="status">0</int>  
<int name="QTime">127</int>  
</lst>  
</response>
```

Utilitaire post

- Dans un environnement Linux, Apache SolR propose le shell **post** pour poster différents types de contenus vers un serveur *ApacheSolR*
- `post -c <collection> [OPTIONS] <files|directories|urls|-d ["...",...]>`
- Exemple :
 - `bin/post -c gettingstarted *.xml`
 - `bin/post -c signals -params "separator=%09" -type text/csv data.tsv`
 - `bin/post -c gettingstarted afolder/`
 - `bin/post -c gettingstarted -filetypes ppt,html afolder/`

Environnement Windows

- En environnement Windows, il est possible d'utiliser le programme Java sous-jacent :

SimplePostTool

```
java -jar example/exampledocs/post.jar -h
SimplePostTool version 5.0.0
Usage: java [SystemProperties] -jar post.jar [-
h|-] [<file|folder|url|arg>
[<file|folder|url|arg>...]]
```

Atelier 3.1

Indexation XML

- Utiliser les coeurs précédemment créés :
- Modifier *schema.xml* pour pouvoir indexer les données de test
- Indexer le XML, tester une recherche

Le format JSON

- Des requêtes au format JSON peuvent également être envoyées au gestionnaire de requête */update*
- Il faut alors préciser le mime-type
Content-Type:application/json ou *Content-Type:text/json*
- Les mises à jour via JSON peuvent prendre 3 formes de base :
 - Un unique document à ajouter
le paramètre *json.command=false* est alors nécessaire.
 - Une liste de documents à ajouter (JSON Array)
 - Une séquence de commandes de mises à jour

URLs JSON

- En plus du gestionnaire de requête */update*, il existe des gestionnaires spécifiques JSON qui surcharge le comportement des paramètres de requêtes

`/update/json => stream.contentType=application/json`

`/update/json/docs => stream.contentType=application/json
json.command=false`

- Il est également possible d'appliquer des transformations sur les documents JSON d'origine. Cela s'effectue alors avec des paramètres spécifiques.

Curl JSON : Exemples

Indexation listes de document

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8983/solr/my_collection/update' --data-binary '[
{
  "id": "1",
  "title": "Doc 1"
},
{
  "id": "2",
  "title": "Doc 2"
}
]'
```

Indexation commande

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8983/solr/my_collection/update' --data-binary '{
  "delete": { "id":"ID" },
  "delete": { "query":"QUERY" }
/* delete by ID */
/* delete by query */
}'
```

Curl JSON : Transformation

```
curl 'http://localhost:8983/solr/my_collection/update/json/docs'\
'?split=/exams'\
'&f=first:/first'\
'&f=last:/last'\
'&f=grade:/grade'\
'&f=subject:/exams/subject'\
'&f=test:/exams/test'\
'&f=marks:/exams/marks'\
-H 'Content-type:application/json' -d '{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
    {
      "subject": "Maths",
      "test"
      : "term1",
      "marks" : 90},
    {
      "subject": "Biology",
      "test"
      : "term1",
      "marks" : 86}
  ]
}'
```

Atelier 3.2 :

Indexation JSON

- Utiliser les cœurs précédemment créés pour ajouter un document au format JSON
- Tester la recherche

Le format CSV

- Des requêtes au format CSV peuvent également être envoyées au gestionnaire de requête */update*
- Il faut alors préciser le mime-type :
Content-Type: application/csv ou
Content-Type: text/csv
- On peut également utiliser l'URL :
/update/csv => stream.contentType=application/csv
- Les noms des colonnes du CSV doivent correspondre au noms des *fields*
- Ce format est le seul qui soit le même en input et en output de query
- On pourrait donc faire une query dans un *SoIR* en demandant un résultat CSV et réinjecter ce résultat dans un autre *SoIR*

Paramètres d'une requête CSV

Paramètres possibles de l'URL :

separator : séparateur à utiliser (',' par défaut)

header=true : indique que la première ligne est une ligne d'entête

fieldnames=field1,field2 : indique le nom des fields à utiliser si le CSV ne contient pas d'entête

overwrite=false : si on n'est sûr que l'on n'overwrite rien

skipLines=1000 : si on veut sauter des lignes

skip=field1,field2 : si certaines colonnes ne doivent pas être importées

escape= caractère d'échappement pour échapper le séparateur dans les valeurs

encapsulator=" : indique le caractère qui entoure les valeurs de champs qui contiennent le séparateur

Exemple curl

- curl
'http://localhost:8983/solr/techproducts/
update?commit=true' --data-binary
@example/exampledocs/books.csv -H
'Content-type:application/csv'

Atelier 3.3 :

Pratique : Requête CSV

- Utiliser les coeurs précédemment créés pour ajouter un document au format CSV
- Tester la recherche

Near Real Time

- Lorsque la cadence d'indexation est élevée et que l'on veut que les documents soient rapidement disponibles à la recherche, on configure un soft commit
- Le recherchabilité d'un document est contrôlée par les commits
 - **soft** : Le document est visible mais pas stocké sur disque. Si le serveur crash, il faudra rejouer la transaction (*tlog*)
 - **hard** : Le document est stocké sur le disque. La transaction correspondante ne fait plus partie du journal des transactions
- La cadence des commits soft et hard est configurable dans *solrconfig.xml*

Exemple de configuration

<!-- Configuration des hardCommit tous les 60 secondes -->

`<autoCommit>`

`<maxTime>${solr.autoCommit.maxTime:60000}</maxTime>`

`<openSearcher>>false</openSearcher>`

`</autoCommit>`

<!-- Configuration des softCommit tous les 30 secondes -->

`<autoSoftCommit>`

`<maxTime>${solr.autoSoftCommit.maxTime:30000}</maxTime>`

`</autoSoftCommit>`

Nested documents

- *Solr* supporte les “**nested documents**” permettant de modéliser des relations 1-N entre documents
- L’indexation se fait par bloc : il faut fournir le document parent et ses documents enfants en même temps (même si un seul enfant a été modifié !)

Règles sur les nested

- Le schéma doit inclure le champ ***_root_*** ('indexé et non stocké). Le valeur du champ est identique pour tous les documents du bloc (indépendamment de sa profondeur dans le graphe)
- Certaines contraintes sur les documents imbriqués :
 - Le schéma doit spécifier leurs champs
 - Impossible d'utiliser *required* sur le champ des enfants
 - Nécessite un *id* unique
- Un champ doit permettre d'identifier un document parent.
- Si le document enfant est associé à un champ, celui ne doit pas être défini dans le schéma. Il n'y a pas de type "child"

Exemple XML

```
<add>
  <doc> <!-- Les documents d'id 1 et 2 ont la même valeur dans le champ _root_ -->
    <field name="id">1</field>
    <field name="title">Solr adds block join support</field>
    <field name="content_type">parentDocument</field> <!-- Champ identifiant un document parent -->
    <field name="content"> <!-- Le champ n'est pas défini dans le schéma -->
      <doc>
        <field name="id">2</field>
        <field name="comments_txt_en">SolrCloud supports it too!</field>
      </doc>
    </field>
  </doc>
  <doc> <!-- Les documents d'id 3 et 4 ont la même valeur dans le champ _root_ -->
    <field name="id">3</field>
    <field name="title">New Lucene and Solr release is out</field>
    <field name="content_type">parentDocument</field>
    <doc>
      <field name="id">4</field>
      <field name="comments">Lots of new features</field>
    </doc>
  </doc>
</add>
```

Exemple JSON

```
[
  {
    "id": "1",
    "title": "Solr adds block join support",
    "content_type": "parentDocument",
    "comment": {
      "id": "2",
      "comments": "SolrCloud supports it too!"
    }
  },
  {
    "id": "3",
    "title": "New Lucene and Solr release is out",
    "content_type": "parentDocument",
    "_childDocuments_": [
      {
        "id": "4",
        "comments": "Lots of new features"
      }
    ]
  }
]
```

Update Request Processor

Introduction

- Toutes les requêtes de mise à jour sont traitées par une chaîne de *plugins* : les ***Update Request Processor***
- Ils peuvent être utilisés pour ajouter un champ au document, changer sa valeur ou pour éviter une mise à jour si certaines conditions ne sont pas respectées

Chaîne

- Un *Update Request Processor* fait partie d'une **chaîne**
- Une chaîne par défaut est fournie par SolR et il est possible de configurer sa propre chaîne
- Un *UpdateRequestProcessor*
 - dans ses conditions normales, appelle l'*UpdateRequestProcessor* suivant après son traitement
 - Dans des conditions d'exception, il peut interrompre le traitement et éviter la mise à jour de l'index

Chaîne par défaut

- La chaîne par défaut comprend :
 - ***LogUpdateProcessorFactory*** : Trace les commandes d'update lors de la requête
 - ***DistributedUpdateProcessorFactory*** : Responsable de distribuer les requêtes de mise à jour au bon noeud dans le cas d'un cloud
 - ***RunUpdateProcessorFactory*** : Exécute les mises à jour en utilisant l'API interne de SolR

Config par défaut

```
<updateRequestProcessorChain
  name="add-unknown-fields-to-the-schema"
  default="${update.autoCreateFields:false}"
  processor="uuid,remove-blank,field-name-mutating,parse-boolean,parse-
long,parse-double,parse-date,add-schema-fields">
  <processor class="solr.LogUpdateProcessorFactory"/>
  <processor class="solr.DistributedUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

Configuration d'une chaîne spécifique

```
<!-- Une chaîne incluant un filtre évitant les duplications -->
<updateRequestProcessorChain name="dedupe">
  <processor class="solr.processor.SignatureUpdateProcessorFactory">
    <bool name="enabled">true</bool>
    <str name="signatureField">id</str>
    <bool name="overwriteDupes">false</bool>
    <str name="fields">name,features,cat</str>
    <str name="signatureClass">solr.processor.Lookup3Signature
  </str>
</processor>
<processor class="solr.LogUpdateProcessorFactory" />
<processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

Configuration d'une chaîne spécifique (2)

```
<updateProcessor
class="solr.processor.SignatureUpdateProcessorFactory"
name="signature">
  <bool name="enabled">true</bool>
  <str name="signatureField">id</str>
  <bool name="overwriteDups">false</bool>
  <str name="fields">name,features,cat</str>
  <str name="signatureClass">solr.processor.Lookup3Signature</str>
</updateProcessor>
<updateProcessor
class="solr.RemoveBlankFieldUpdateProcessorFactory"
name="remove_blanks"/>
...
<updateProcessorChain name="custom"
processor="remove_blanks,signature">
  <processor class="solr.RunUpdateProcessorFactory" />
</updateProcessorChain>
```

Utilisation des chaînes

- Le paramètre ***update.chain*** permet d'indiquer la chaîne que l'on veut utiliser lors d'une requête
- On peut également créer une chaîne à la volée en indiquant le paramètre ***processor*** est la liste des processeurs à utiliser
`?processor=remove_blanks,signature&commit=true`

Quelques Processeurs disponibles

- Ajout automatique de champ
- Valeur par défaut pour un champ
- Ajout automatique d'un champ *timestamp*
- Traitement de date d'expiration
- Attribution d'une valeur de *boost* en fonction de la valeur d'un champ ... et d'une regexp
- Calcul d'une signature pour éviter la duplication
- Concaténation automatique de champs
- Suppression de balises HTML
- Détection de langues
-

Exemple détection de langue

- SolR peut tenter de détecter la langue d'un champ et de l'associer ensuite à un champ dédié à une langue. Le processeur s'appelle *langid* et SolR fournit 2 implémentations :
 - **Tika**
 - **LangDetect** (qui apparemment donne de meilleurs résultats actuellement)
 -

Configuration

- La configuration minimale consiste à spécifier les champs pour l'identification et un champ pour stocker le résultat (un code langue)

```
<processor
class="org.apache.solr.update.processor.LangDetectLanguage
IdentifierUpdateProcessorFactory">
<lst name="defaults">
<str name="langid.fl">title,subject,text,keywords</str>
<str name="langid.langField">language_s</str>
</lst>
</processor>
.
```


Fichiers binaires

Solr Cell

Solr utilise Apache Tika pour intégrer différents formats de fichiers.

Le handler ***ExtractingRequestHandler*** utilise Tika et les projets associés (Apache PDFBox, Apache POI) pour supporter l'upload de fichiers binaires et l'extraction de données à indexer.

Au départ ce framework était appelé « *Solr Content Extraction Library* » ou « *CEL* »; il est devenu ensuite : ***Solr Cell***.

Pour le mettre en place, il faut placer des librairies additionnelles dans le classpath

Concepts clés

- Tika essaie de déterminer **automatiquement le type** du document.

Il est possible de positionner le mime-type explicitement avec le paramètre *stream.type*

- Tika produit un **flux XHTML** qui alimente un parseur SAX : *ContentHandler*. SolR répond aux événements SAX et crée les champs à indexer. Il est possible d'implémenter son propre *ContentHandler*

- Il est possible de fournir une expression **XPath** afin de restreindre le contenu

Concepts clés (2)

- Tika produit des méta-données comme le **Titre**, le **sujet** et **l'auteur** selon les spécifications des formats. Il est possible de les associer aux champs SolR et de leur affecter un facteur de boost
- Tika ajoute tout le texte du fichier au champ **content**
- Il est possible de surcharger les valeurs parsées par Tika via ses **propres valeurs**

Examples

Indexation

curl

```
'http://localhost:8983/solr/techproducts/  
update/extract?literal.id=doc1&commit=true'  
-F "myfile=@example/exampledocs/solr-  
word.pdf"
```

```
bin/post -c techproducts  
example/exampledocs/solr-word.pdf -params  
"literal.id=a"
```

Recherche

```
http://localhost:8983/solr/techproducts/select?q=pdf119
```

Exemples avancés

- Capture les tags `<div>`, les fait correspondre au champ `foo_t` et le booste par 3

```
bin/post -c techproducts
example/exampledocs/sample.html -params
"literal.id=doc3&captureAttr=true&defaultField=_text
_&capture=div&fmap.div=foo_t&boost.foo_t=3"
```

- Utilisation de littéral

```
bin/post -c techproducts -params
"literal.id=doc4&captureAttr=true&defaultField=text&captur
e=div&fmap.div=foo_t&boost.foo_t=3&literal.blah_s=Bah"
example/exampledocs/sample.html
```

- XPath

```
bin/post -c techproducts -params
"literal.id=doc5&captureAttr=true&defaultField=text&captur
e=div&fmap.div=foo_t&boost.foo_t=3&xpath=/xhtml:html/
xhtml:body/xhtml:div//node()"
example/exampledocs/sample.html
```

Extraire sans indexer

Il est possible d'extraire les données du document sans les indexer. Cela permet de tester l'extraction et d'adapter le mapping

```
curl "http://localhost:8983/solr/techproducts/update/extract?  
&extractOnly=true"  
--data-binary @example/exampledocs/sample.html -H  
'Content-type:text/html'
```

```
bin/post -c techproducts -params  
"extractOnly=true&wt=ruby&indent=true" -out yes  
example/exampledocs/sample.html
```

Mise en place

- La mise en place de Tika consiste à :
 - Ajouter les librairies dans le classpath :

```
<lib dir="${solr.install.dir:../../..}/contrib/extraction/lib" regex=".*\.jar" />  
<lib dir="${solr.install.dir:../../..}/dist/" regex="solr-cell-\.d.*\.jar" />
```

- Configurer le request handler pointant vers
solr.extraction.ExtractingRequestHandler

```
<requestHandler name="/update/extract" startup="lazy"  
class="solr.extraction.ExtractingRequestHandler" >  
  <lst name="defaults">  
    <str name="lowernames">true</str>  
    <str name="fmap.content">_text_</str>  
  </lst>  
</requestHandler>
```


Atelier 4 :

Importation de documents

- Créer un nouveau coeur
- Configurer le gestionnaire d'importation
- Utiliser l'utilitaire *post* pour ajouter les fichiers fournis

Source de données

Data Import Handler

Le ***Data Import Handler (DIH)*** fournit un mécanisme pour importer du contenu à partir d'un support persistant
Il est capable d'indexer du contenu à partir d'une base de données relationnelles des flux RSS ou ATOM, des e-mails ou du contenu XML extrait via *XPath*

Un exemple est disponible :
bin/solr -e dih

Concepts

DIH utilise plusieurs concepts :

- ***Datasource*** : Emplacement du data store
- ***Entity*** : Une entité génère un ensemble de documents avec plusieurs champs. Pour une base de données : une vue ou une table
- ***Processor*** : Responsable d'extraire le contenu et de l'indexer. Il est possible de fournir ses propres *processors* qui surchargent ceux fournis par défaut
- ***Transformer*** : Chaque champs de l'entité peuvent être transformés. *ApacheSolR* fournit des *transformers* et il est possible d'écrire ses propres *transformers*

Configuration

Le DIH doit être configuré dans solrconfig.xml

Le seul paramètre nécessaire est **config** qui donne le chemin vers le fichier de configuration spécifique du DIH

```
<requestHandler name="/dataimport"  
  class="org.apache.solr.handler.dataimport.DataImportHandler">  
  <lst name="defaults">  
    <str name="config">/path/to/my/DIHconfigfile.xml</str>  
  </lst>  
</requestHandler>
```

Configuration base et requêtes

- La configuration de la datasource consiste à :
 - Ajouter les **drivers** jdbc dans le classpath
 - Fournir **l'URL jdbc**
 - Les attributs ***autocommit*** et ***batch-size***
 - Un élément ***document*** contenant plusieurs balises ***entity*** potentiellement imbriquées permettant de suivre les relations de la base
 - Les éléments *entity* contiennent des éléments ***field*** sur lesquels peuvent être appliqués des transformers

Example

```
<dataConfig>
  <dataSource driver="org.hsqldb.jdbcDriver"
url="jdbc:hsqldb:./example-DIH/hsqldb/ex" user="sa" />
  <document>
    <entity name="item" query="select * from item"
      deltaQuery="select id from item where last_modified >
        '${dataimporter.last_index_time}'">
      <field column="NAME" name="name" />
      <!-- One to Many -->
      <entity name="feature"
        query="select DESCRIPTION from FEATURE where
          ITEM_ID='{item.ID}'"
        deltaQuery="select ITEM_ID from FEATURE
          where last_modified > '${dataimporter.last_index_time}'"
        parentDeltaQuery="select ID from item where
          ID=${feature.ITEM_ID}">
        <field name="features" column="DESCRIPTION" />
      </entity>
    </entity>
  </document>
</dataConfig>
```

API Rest : Commandes DIH

Les commandes *DIH* sont lancées via des requêtes HTTP :

- ***full-import*** : Importe toutes les entités
- ***delta-import*** : Importation incrémentale et détection de changement
- ***reload-config*** : Rechargement de la configuration
- ***status*** : Retourne des statistiques (documents créés, ...)

Examples

Import global

curl

```
'http://localhost:8983/solr/dih/dataimport?  
command=full-import&entity=item'
```

Statistiques

curl

```
'http://localhost:8983/solr/dih/dataimport?  
command=status'
```

Atelier 5

Base de données

Optimisation index

- Minimiser l'index en fonction des cas d'utilisation de recherche
 - *index=false*, pour les champs non utilisés pour la recherche
 - *copyField* pour rassembler dans un seul champ les champs texte utilisés pour la recherche

Optimisation Indexation

- L'optimisation du temps d'indexation se fait généralement
 - En augmentant la taille du batch (nombre de documents en 1 requête d'update)
 - En multipliant le nombre de threads effectuant le travail d'indexation
 - En utilisant *SolrCloud*

Recherches

Query parser

Rappels sur la syntaxe Lucene

Spell check

Surbrillance

Auto-complétion

Recherche par facette

Recherche par groupe

Recherche spatiale

Élévation

Introduction

- Solr propose un mécanisme de recherche très flexible
- Une requête de recherche est traitée par un ***RequestHandler*** (défini dans la configuration SolR)
- Le *RequestHandler* fait appel à un ***Query parser*** qui prend en général des paramètres d'entrée :
 - La chaîne à chercher
 - Des paramètres de tuning de la requête
 - Des paramètres contrôlant la présentation de la réponse
- Les parseurs ont en commun certains paramètres d'entrée, d'autres leur sont spécifiques

Query Parsers

- Les parseurs les plus courants sont :
 - ***StandardQueryParser*** : Extension du parseur Lucene. Très puissant mais syntaxe compliquée et peu tolérante
 - ***DisMaxParser*** : Fait pour traiter de simples phrases directement saisies par l'utilisateur. Effectue la recherche sur différents champs qui ont différents poids (boosts)
 - ***ExtendedDisMax*** : Ajoute des fonctionnalités avancées à *DisMax*

Cache, Réponse

- Les paramètres de recherche peuvent également spécifier un **query filter** qui met en cache les résultats et permet d'améliorer les performances
- Une requête de recherche peut demander la **surbrillance** de certains termes
- Les réponses peuvent également inclure des document **snippets** (extrait du document)

Regroupement des résultats

- SolR permet également de regrouper les résultats de recherche de 2 façons, facilitant l'exploration de données :
 - Les **facettes** groupent les résultats de la recherche dans des catégories (qui sont basées sur les termes indexés).
 - Le **clustering** groupent les résultats selon des similarités découvertes algorithmiquement lors de la recherche
- SolR supporte des requêtes de type **MoreLikeThis** qui se basent sur des termes retournés par une requête précédente

Présentation des résultats

- Les composants Solr de type ***Response Writer*** gèrent la présentation finale de la réponse.
- Solr fournit différents *Response Writers* comme :
 - *XML Response Writer*
 - *JSON Response Writer*
 - *Velocity Response Writer*

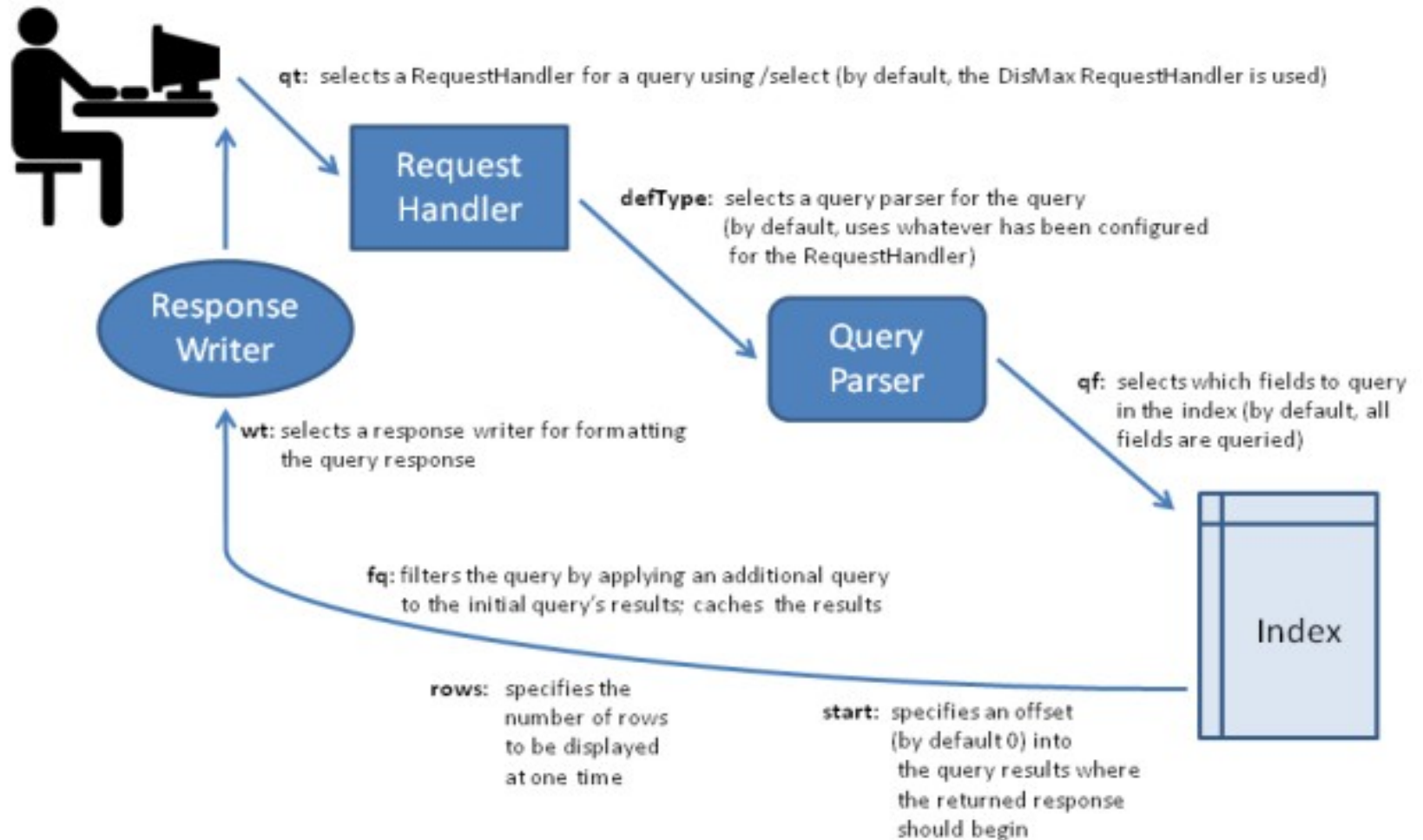
Pertinence

- La pertinence mesure l'adéquation de la réponse à la requête
- Elle est fortement dépendante du contexte de la requête
- Il est souvent utile dans les étapes de préparation du déploiement SolR de spécifier les types de réponses que l'application doit retourner pour des exemples de requêtes.
- Il est alors possible de jouer sur la configuration pour obtenir les résultats voulus

Réponses

```
{
  "responseHeader":{
    "status":0, // Code retour
    "Qtime":0,  // Temps d'exécution
    "params":{
      "q":"_root_:1",
      "_":"1542533170700"}}},
  "response":{"numFound":2,"start":0,"docs":[ // Total et position de départ
    {
      "id":"2",
      "comments_txt":["SolrCloud supports it too!"],
      "_version_":1617463248575004672},
    {
      "id":"1",
      "title":["Solr adds block join support"],
      "content_type":["parentDocument"],
      "_version_":1617463248575004672}]
  }}
```

Big Picture



Interface utilisateur exemple

- *SolR* fournit une interface exemple basée sur *VelocityResponseWriter* qui démontre certaines fonctionnalités de SolR (recherche, facette, surbrillance, autocomplete et recherche spatiale)

<http://localhost:8983/solr/techproducts/browse>

RequestHandler

- Configure un type de recherche (ou d'update)
 - Indique des paramètres par défaut
 - Indique quels composants de recherche utiliser (*<searchComponent/>*)

=> Recommandation : Configurer un RequestHandler par type de recherche dans votre application

- Chercher le *RequestHandler* « */browse* » de l'exemple *techproducts* pour voir sa longue liste de paramètres

Appeler un *RequestHandler*

- Il y a 2 façons d'appeler un *RequestHandler* spécifique :
 1. Soit à l'URL */select* avec le paramètre « **qt** » (pour « *queryType* ») contenant le nom du requestHandler

<http://localhost:8389/core01/select/?q=primaire&qt=myRequestHandler>

2. Soit en appelant l'URL correspondant au nom du *RequestHandler*

<http://localhost:8389/core01/myRequestHandler?q=primaire>

Handler par défaut

- */select* : Handler générique permettant de dispatcher via le paramètre *qType*
- */query* : Format JSON identé
- */browse* : Interface par défaut pour parcourir la collection exemple techproducts. (Velocity)

Paramètres de recherche communs

- La query :

<i>q</i> (query)	query de recherche
<i>fq</i> (filterQuery)	queries de filtrage (~ WHERE SQL)
<i>defType</i>	le parser de query (lucene ou edismax)

- Pagination

<i>rows</i>	Nombre de résultats
<i>start</i>	Offset de départ de la liste

- Output :

<i>fl</i> (fieldList)	Champs à remonter
<i>sort</i>	Critère de tri (la pertinence en général)
<i>wt</i> (writer type)	Format de la réponse

- Diagnostic :

<i>indent</i>	Indentation du résultat
<i>explainOther</i>	ce que SolR a compris de la recherche
<i>debug</i>	votre ami pour le tuning

RequestHandler : liste « defaults »

- La liste « **defaults** » donne les valeurs des paramètres qui seront utilisés si aucune valeur n'est précisée explicitement dans la query

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="defaults">  
    <str name="echoParams">explicit</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```

RequestHandler : liste « *appends* »

- La liste « ***appends*** » donne les valeurs des paramètres qui seront ajoutées aux paramètres multivalués de la query (comme *fq*)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="appends">  
    <str name="fq">a_type:group</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```

RequestHandler : liste « invariants »

- La liste « ***invariants*** » donne les valeurs des paramètres qui seront toujours utilisés, quelque soit les valeurs indiquées dans la requête (utile pour sécuriser les *requestHandler*)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="invariants">  
    <str name="facets">false</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```

Les « composants » de recherche

- Un « ***search component*** » exécute une fonctionnalité de recherche : *highlight, facettes, MLT, etc.*
- Chaque « *component* » est déclaré dans une section de *solrconfig.xml*

```
<searchComponent class="solr.HighlightComponent" name="highlight">  
  <highlighting>  
    <!-- etc... -->  
  </highlighting>  
</searchComponent>
```

SearchComponents par défaut

- Les composants de recherche s'appliquant par défaut sont dans l'ordre :
- ***query*** : Parser.
- ***facet*** : Facette
- ***Mlt*** : MoreLikeThis.
- ***highlight*** : Surbrillance
- ***stats*** : Génère des statistiques
- ***debug*** : Debug
- ***expand*** : Gestion des groupes

RequestHandler : array

« components »

- Un *RequestHandler* peut redéfinir la liste des composants de recherche par défaut via une balise **array nommée « components »**

```
<requestHandler name="/myHandler" class="solr.SearchHandler">
  <arr name="components">
    <!-- This is default components ! -->
    <str>query</str>
    <str>facet</str>
    <str>mlt</str>
    <str>highlight</str>
    <str>stats</str>
    <str>debug</str>
  </arr>
</requestHandler>
```


« first-components » et « last-components »

- On peut ajouter un composant de recherche au début ou à la fin de la liste par défaut avec « **first-components** » et « **last-components** »

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <arr name="last-components">  
    <str>elevator</str>  
  </arr>  
</requestHandler>
```

Atelier 6.1

Configuration request handler

- Reparamétrer le handler */browse*
- Commentez toutes les parties non nécessaires :
 - Mlt
 - Facets
 - highlight
 - Spellcheck
 - Conserver seulement les parties « *VelocityResponseWriter* » et « *Query settings* »
 - Ajuster le paramètre *qf*
- Copier les nouveaux templates Velocity dans le sous-répertoire « *velocity* » pour correspondre à notre schéma

Le score et le tri

- Champ spécial « *score* »
 - on peut le ramener avec le paramètre « ***fl=score*** »
- « *sort* » se fait sur ce pseudo-champ « *score* » par défaut
- Tester avec *sort=titre_en* pour trier par ordre alphabétique des titres

Calcul du score

- Les facteurs influant le score sont :
 - **tf** (*term frequency*): La fréquence du terme dans le document
 - **idf** (*inverse document frequency*): La fréquence du terme dans l'index
 - **coord** : Le nombre de termes de la requête trouvés dans le document
 - **lengthNorm** : L'importance du terme par rapport au nombre total de terme pour ce champ
 - **queryNorm** : Facteur de normalisation permettant de comparer les requêtes
 - **boost** (index) : Le boost du champ au moment de l'indexation
 - **boost** (query) : Le boost du champ au moment de la requête

Implémentation et implication

- *tf* : **$\text{sqrt}(\text{freq})$** \Rightarrow Plus le terme apparaît dans le document plus le score est élevé
- *idf* : **$\log(\text{numDocs}/(\text{docFreq}+1)) + 1$** \Rightarrow Plus le terme est présent dans l'index moins le score est élevé
- *coord* : **$\text{overlap} / \text{maxOverlap}$** \Rightarrow Plus il y a de termes plus le score est élevé
- *lengthNorm* : **$1/\text{sqrt}(\text{numTerms})$** \Rightarrow Moins il y a de termes pour ce champ plus le score est élevé

Fonctions

- Les fonctions permettent de générer un score de pertinence à partir de la valeur de champs numériques :
- Les fonctions peuvent être :
 - Une constante (string ou nombre)
 - Un champ
 - Une autre fonction
- Elles permettent de changer le rang d'un résultat à partir d'un calcul et influe donc l'ordre de présentation des résultats

Usage

- Il y a plusieurs façons d'utiliser les fonctions dans une requête SolR:
 - En utilisant un parseur spécifique
`q={!func}div(popularity,price)&fq={!frange l=1000}customer_ratings`
 - Dans une expression de tri
`sort=div(popularity,price) desc, score desc`
 - Dans une expression d'un pseudo champs
`&fl=sum(x, y),id,a,b,c,score`
 - Dans un paramètre qui le supporte (ex: boost de EDisMax ou bf DisMax)
`q=dismax&bf="ord(popularity)^0.5
recip(rord(price),1,1000,1000)^0.3"`

Syntaxe de recherche

- La syntaxe dépend du parser de query (paramètre *defType*)
 - **Standard** (Lucene)
 - Ou **dismax** ou **edismax**
- Le parser « lucene » est le plus précis et permet de combiner des critères sur tous les champs
 - mais une « search box » dans une interface est généralement branchée sur un *defType=edismax* qui est plus simple pour du plein-texte

Syntaxe Lucene : indication des champs

- ***:*** matche tous les documents
- Indiquer un champ spécifique :
pays:France

Syntaxe Lucene : clauses

- Une recherche est un ensemble de clauses :
 - education*** : clause optionnelle
 - +education*** : clause obligatoire
 - education*** : clause interdite
- Combiner les clauses :
 - AND / && : ***+education AND pays:france***
 - OR / || : ***+education OR pays:france***
- Parenthèses
 - (education AND teacher) OR (quality AND plan)***
 - (+education +teacher) (+quality +plan)***
 - différent de* ***+(education teacher) +(quality plan)***

Syntaxe Lucene : phrase et wildcard

- « Phrase query » :
quality education vs. ***“quality education”***
- Proximité des mots pour une phrase
“ quality education ”~3
- Wildcard queries
educat*
In*tion -innovation
Innova????
****tion***

Syntaxe Lucene : fuzzy, range, boost

- Recherches floues (fuzzy, un peu équivalent au stemming)

auteur:ewing vs. ***auteur:ewing~*** vs.
auteur:ewing~0.8

- Range queries

education AND annee:[1999 TO 2001]
education AND annee:[2001 TO *]

- Score boosting (joue sur la pertinence)

child primary^10

Mr « Eddy Smax » (edismax)

- edismax vs. lucene :
 - Fait pour traiter des phrases simples
 - Supporte un sous-ensemble simplifié de la syntaxe Lucene
 - Recherche dans plusieurs champs en même temps avec différents poids
 - Query par défaut
 - + Nombre minimum de mots à matcher
 - + Smart boosting (proximity)
- Activer edismax avec le paramètre
 - *defType*=« *edismax* »

Edismax : paramètres

- **qf** : query fields

children education

*titre_en^10 resume^2 vs. titre_en^2
resume^10*

- **q.alt** : alternative query si *q* n'est pas précisé

– Souvent **:**

- **mm** : minimum de clauses devant matchée

- Voir

http://wiki.apache.org/solr/DisMaxQParserPlugin#mm_.28Minimum_.27Should.27_Match.29

Atelier 6.2

1ères requêtes

- Effectuer les recherches suivantes en utilisant la syntaxe lucene :
 - Documents répondant à « Java »
 - Documents ne répondant pas à « Java »
 - Limiter les documents retournés de la première requête
 - Documents dont le contenu répond à « Java »
 - Documents PDF dont le contenu répond à « Java »
 - Documents dont le contenu répond à « SolR»
 - Documents dont le champ titre contient administration
 - Document créés après une date particulière
 - Document créés après une date particulière et dont le contenu répondant « Java Elastic Search » mais pas « Administration »

Bloc Join Query Parsers

- Il y a 2 parseurs qui supportent les jointures sur les blocs de documents (i.e. nested documents, parent/children)
 - ***Block Join Children Query Parser*** : Critère sur le parent et retourne les documents enfants
 - ***Block Join Parent Query Parser*** : Critère sur les enfants et retourne des documents parent

Children

- La syntaxe est :
q={!child of=<allParents>}<someParents>
 - ***allParents*** est un critère qui retourne tous les parents. (On utilise en général le champ qui identifie un document parent)
 - ***someParents*** : critère sur les parents
- Exemple :
`q={!child of="content_type:parentDocument"}title:lucene&wt=xml`

Parent

- La syntaxe est :

`q={!parent which=<allParents>}<someChildren>`

- ***allParents*** est un critère qui retourne tous les parents. (On utilise en général le champ qui identifie un document parent)
- ***someChildren*** : critères sur les enfants

- Exemple :

```
q={!parent  
  which="content_type:parentDocument"}comments:SolrCloud&wt=xml
```

Boolean Query

- **BqueryParser** permet de combiner plusieurs requêtes via des opérateurs qui influe sur le type de requête effectué et comment le score de pertinence est calculé :
 - **must** : Une liste de requêtes que les documents doivent matcher, les requêtes contribuent au score.
 - **must_not** : Une liste de requêtes que les documents ne doivent pas matcher, ne contribuent pas au score
 - **should** :
 - Si pas de requêtes must : Les documents retournés doivent matcher au moins une requête should
 - Sinon, ne fait qu'augmenter le score.
 - **filter** : Les documents doivent matcher mais les requêtes ne contribuent pas au score
- Exemples :

```
{!bool must=foo must=bar}  
{!bool filter=foo should=bar}
```

Join Query

- Le **JoinQParser** permet d'exprimer des jointures entre documents.
- Il prend les paramètres :
 - **from** : La “clé étrangère”
 - **to** : La “clé primaire”
- Exemple :
`q={!join from=manu_id_s to=id}ipod`
- La jointure peut également s'effectuer entre deux collections, si on a fait attention aux shards:
`fq={!join from=region_s to=region_s
fromIndex=people}mgr_s:yes`

SpellCheck

Spellcheck : configuration du schema

- « Voulez-vous dire... ? »
- Les suggestions orthographiques doivent se baser sur un dictionnaire :
 - Un fichier texte
 - Ou bien les valeurs d'un champ de l'index (recommandé)
 - En général, un champ spécifiquement dédié à cela et rempli via des *copyField*
 - stored=false
 - Pas de stemming
 - lowercasing

Spellcheck : configuration du component

- Le component « *spellcheck* » existe déjà dans *solrconfig.xml*. Ses options :
 - *name* : un nom de configuration
 - *classname* :
 - ***solr.DirectSolrSpellchecker*** : Basé directement sur un index Solr
 - ***solr.IndexBasedSpellChecker*** : Basé sur un index *Solr* et crée un autre index dédié pour le spellchecking
 - ***solr.FileBasedSpellChecker***: Basé sur un fichier externe

Configuration DirectSolr

- **field** : nom du champ dans lequel lire les valeurs
- **distanceMeasure** : Le mode de calcul de la distance entre 2 mots (internal = Levenshtein)
- **accuracy** : Le seuil de distance pour la suggestion
- **thresholdTokenFrequency** : entre 0 et 1; pour exclure les termes qui n'apparaissent pas souvent (défaut : 0, tous les termes. A priori 0.01 élimine les mots bizarres)
- **maxEdits** : nombre de lettres de différence autorisé (1 ou 2) pour effectuer la requête
- **minPrefix** : nombre de lettres communes au début du terme
- **minQueryLength** : nombre mini de lettres dans la recherche pour déclencher un suggest

Example :DirectSolr

```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="field">name</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="distanceMeasure">internal</str>
    <float name="accuracy">0.5</float>
    <int name="maxEdits">2</int>
    <int name="minPrefix">1</int>
    <int name="maxInspections">5</int>
    <int name="minQueryLength">4</int>
    <float name="maxQueryFrequency">0.01</float>
    <float name="thresholdTokenFrequency">.01</float>
  </lst>
</searchComponent>
```

Exemple : *IndexBase*

- ```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str
name="classname">solr.IndexBasedSpellChecker</str>
 <!-- Emplacement de l'index dédié spellecheck -->
 <str name="spellcheckIndexDir">./spellchecker</str>
 <!-- Champ utilisé pour créer l'index
(Ne pas prendre un champ avec trop de traitement de
termes / synonym, stem -->
 <str name="field">content</str>
 <!-- Recronstruction de l'index lors de l'ajout de
doc. -->
 <str name="buildOnCommit">true</str>
 </lst>
</searchComponent>
```

# Exemple : *FileBase*

```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str name="classname">solr.FileBasedSpellChecker</str>
 <str name="name">file</str>
 <!-- Source du fichier -->
 <str name="sourceLocation">spellings.txt</str>
 <str name="characterEncoding">UTF-8</str>
 <str name="spellcheckIndexDir">./spellcheckerFile</str>
 </lst>
</searchComponent>
```

# Correcteur d'espaces

- Un autre correcteur orthographique nommé ***WordBreakSolrSpellChecker*** permet de corriger les mauvais espaces (supprimer ou ajouter)

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
<lst name="spellchecker">
<str name="name">wordbreak</str>
<str name="classname">solr.WordBreakSolrSpellChecker</str>
<str name="field">lowerfilt</str>
<str name="combineWords">true</str>
<str name="breakWords">true</str>
<int name="maxChanges">10</int>
</lst>
</searchComponent>
```

# Spellcheck : configuration du handler

- Le spellcheck component est normalement toujours ajouté à un *requestHandler* existant (jamais appelé séparément)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">
 <lst name="defaults">
 <!-- Active le spellchecking -->
 <str name="spellcheck">true</str>
 <!-- Nom du spellchecker (attribut "name") -->
 <str name="spellcheck.dictionary">default</str>
 <!-- Nombre de propositions par mot -->
 <str name="spellcheck.count">1</str>
 </lst>
 <arr name="last-components">
 <str>spellcheck</str>
 </arr>
</requestHandler>
```

# Spellcheck : paramètres de query

- **spellcheck=true** : active le spellcheck
- **spellcheck.dictionary** : doit correspondre à « name » du component
- **spellcheck.count** : nombre de suggestion à retourner
- **spellcheck.onlyMorePopular** : Retourne seulement des termes de l'index + fréquents que le terme d'origine
- **spellcheck.extendedResults** : Retourne les informations de fréquence des termes (pour du debug)
- **spellcheck.collate=true** : dans le cas de query multi-termes, retourne en plus une suggestion de query complète agrégeant les suggestions pour chaque terme
  - **spellcheck.maxCollations=1** nombre max de suggestions de collations
  - **spellcheck.maxCollationsTries=0** si >0, nombre d'essais pour voir si la suggestion de collation donne des résultats (mettre à 5)
  - **spellcheck.collateExtendedResults** : ajoute des détails, pour du debug

# Spellcheck : résultats

```
<response>...
<lst name="spellcheck">
 <lst name="suggestions">
 <lst name="dell">
 <int name="numFound">1</int>
 <int name="startOffset">18</int><int name="endOffset">23</int>
 <int name="origFreq">0</int>
 <arr name="suggestion"><lst>
 <str name="word">dell</str>
 <int name="freq">2</int>
 </lst></arr>
 </lst>
 <lst name="ultrashar">
 <int name="numFound">1</int>
 <int name="startOffset">24</int><int name="endOffset">33</int>
 <int name="origFreq">0</int>
 <arr name="suggestion"><lst>
 <str name="word">ultrasharp</str>
 <int name="freq">2</int>
 </lst></arr>
 </lst>
 <bool name="correctlySpelled">>false</bool>
 <str name="collation">price:[80 TO 100] dell ultrasharp</str>
 </lst>
</lst>
</response>
```

# Surbrilliance



# Highlight : paramètres

- **hl=true**
  - Active le highlight
- **hl.fl=<field1>,<field2>,<fieldN>**
  - Liste des champs à highlighter
  - Par défaut, liste des champs dans le paramètre « qf » de edismax
- **hl.usePhraseHighlighter=true**
  - Fait en sorte qu'une « phrase query » comme « a b c » highlighte « b » seulement dans un morceau de texte « a b c »
- Beaucoup d'autres options avancées existent, voir <http://wiki.apache.org/solr/HighlightingParameters>

# Highlight : résultats

- Le highlight est un component à part entière, et ses résultats sont donc séparés de la liste de résultat des documents en tant que telle !

# Recherches par facettes

# Les facettes

- Les facettes enrichissent un résultat de recherche avec des informations **agrégées** sur la liste de résultats : *Similaire à un « GROUP BY » en SQL, fait automatiquement lors d'une recherche;*

# Atelier

## Les facettes

- Dans l'interface d'admin, cocher « *facet* »  
Spécifier « année » dans « *facet.field* » et lancer la recherche
  - Regarder la partie « *facet\_counts* » à la fin du résultat de recherche;
- Combiner avec une recherche texte;
- spécifier « motcle » à la place et relancer la recherche
  - Regarder la partie « *facet\_counts* » à la fin du résultat de recherche;
  - Que constatez-vous ?

# Plusieurs types de facettes

- `facet_fields` : Facette sur valeur de champs (les plus communes)
  - Pour chaque valeur d'un champ, compte le nombre de résultats qui ont cette valeur;
  - « combien ai-je d'items par pays ? Par mot-clé ? »
- `facet_ranges` : Facette sur plage de valeur numérique ou date
  - Pour une liste de plages, compte le nombre de résultats dans chaque plage;
  - « combien d'items font moins de 100 euros, entre 100 et 200 euros, 200 et 300 euros, plus de 300 euros ? »
- `facet_queries` : Facette sur des query dynamiques
  - Pour une liste de queries, compte le nombre de résultats pour chaque requêtes;

# Les facettes : contraintes

- Un champ facetté doit être indexé  
« `indexed=true` »
- Un champ facetté ne doit pas être tokenisé (en général) => Type string sans analyse, entier, date, booléen

# Paramètres de la requête

- `facet=true`
  - Active les facettes
- `facet.field=<nom_du_champ>`
  - Donne le nom d'un champ sur lequel on veut facetter
  - On peut **répéter** ce paramètre plusieurs fois pour facetter sur plusieurs champs (ce que ne permet pas l'interface d'admin)



# Les facettes : paramètres

- Tous les paramètres suivants peuvent être mis :
  - Soit tels quels pour s'appliquer à toutes les facettes;
  - Soit préfixés par « `f.<nom_du_champ>.<parametre>` » pour s'appliquer à une seule facette; c'est préférable.
- `facet.sort=count` ou `facet.sort=index`
  - Tri la liste de valeurs par nombre de documents associés (défaut) ou par ordre alphabétique
  - Exemple : `f.motcle.facet.sort=index`
- `facet.limit=100`
  - Nombre maxi de valeurs pour une facette (défaut : 100)
  - Exemple : `f.motcle.facet.limit=10`
- `facet.mincount=1`
  - Nombre mini de résultats associés à une valeur pour que celle-ci s'affiche
  - Par défaut, `mincount=0`, ce qui veut dire que même les valeurs sans résultat s'affichent
  - Exemple : `f.annee.facet.mincount=1`

# Les facettes : paramètres

- `facet.missing=on`
  - Inclut une valeur vide supplémentaire pour compter les résultats qui n'ont pas de valeur pour ce champ
  - Exemple : `f.motcle.facet.missing=on`
- `facet.offset=10`
  - Offset de départ dans la liste des valeurs (à combiner avec `facet.limit` pour paginer dans les valeurs de facettes)
  - Exemple : `f.motcle.facet.offset=10`
- `facet.prefix=<chaine>` (advanced)
  - Ne garde que les valeurs de facettes qui commencent par la chaine indiquée
  - Utilisée pour des facettes hiérarchiques ou de l'autocomplétion

# Atelier

## Les facettes

- Modifier *schema.xml* pour permettre une recherche à facette sur les pays.
  - Que faut-il ajouter ?
  - Réindexer les données après modification
  - Tester une query avec facette sur les pays pour valider
- Modifier *solrconfig.xml* pour ajouter les facettes « annee » et « motcle » au handler « /browse »
  - Tester en naviguant à <http://localhost:8983/solr/formation/browse>
  - Ajuster les paramètres de chaque facette : limiter le nombre de mot-clés, ne pas afficher les facettes avec 0 résultats, trier la facette année par ordre alphabétique;

# Les facettes ranges : paramètres

- `facet . range=<nom_du_champ>`
  - Fait une facette range sur un champ
- Tous les paramètres suivants peuvent être mis :
  - Soit tels quels pour s'appliquer à toutes les facettes;
  - Soit préfixés par  
« `f . <nom_du_champ> . <parametre>` » pour s'appliquer à une seule facette; c'est préférable.

# Les facettes ranges : paramètres

- `facet.range.start=<valeur>`
  - Valeur de début de la facette
- `facet.range.end=<valeur>`
  - Valeur de fin de la facette
- `facet.range.gap=<valeur>`
  - Le pas de l'itération pour chaque range (10 en 10, 5 en 5, etc.)
- `facet.range.hardend=true`
  - Tronque le dernier range si `facet.range.gap` va au-delà de `facet.range.end` (défaut : `false`)
- `facet.range.other=all|none|before|after|between`
  - Inclut des valeurs supplémentaires pour compte le nombre de résultats avant/après/dans la place spécifiée; « none » (par défaut) ne compte rien de plus, « all » compte avant/après/dans la place en même temps.

# Les facettes : pratique

- Modifier solrconfig.xml pour remplacer la facette « annee » par une facette range de 5 ans en 5 ans
  - Tester en naviguant à <http://localhost:8983/solr/formation/browse>

```
<str name="facet.range">annee</str>
<int name="f.annee.facet.range.start">2000</int>
<int name="f.annee.facet.range.end">2014</int>
<int name="f.annee.facet.range.gap">5</int>
<str name="f.annee.facet.range.hardend">true</str>
<str name="f.annee.facet.range.other">all</str>
```

# Regroupement

# Introduction

- Le **groupement de résultat** regroupe les documents ayant en commun la valeur d'un champ et retourne les meilleurs documents pour chaque groupe
- La fonctionnalité ***Collapse et Expand*** de SolR est plus récente que le grouping et offre de meilleure performance. Elle est donc préférée au regroupement.



# Composants

- La fonctionnalité de regroupement est basée sur 2 composants :
  - Le parseur de requête ***Collapsing*** qui groupe le document en fonction des paramètres fournis
  - Le composant ***Expand*** qui fournit un accès aux documents d'un groupe
- Attention : Avec SolrCloud, les documents doivent être sur le même shard

# Paramètres

- **field** : Le champ de regroupement. Type String, Int ou Float.
- **min** ou **max** ou **sort** : Permet de sélectionner les documents d'entête via la valeur *min* ou *max* du champ, d'une fonction ou d'un critère de tri. Si aucun de ces paramètres n'est spécifié, les documents d'entêtes sont sélectionnés en fonction du score
- **nullPolicy** :
  - **ignore** (défaut): ignore les documents ayant le champ à null
  - **expand** : crée un groupe pour chaque document ayant le champ à null,
  - **collapse** : crée un seul groupe pour tous les documents ayant le champ à null .

# Exemples

- Sélection du document avec le meilleur score dans chaque groupe

```
fq={!collapse field=group_field}
```

- Sélection du document ayant la valeur minimale dans chaque groupe :

```
fq={!collapse field=group_field
min=numeric_field}
```

# Expand

- Le paramètre **expand** utilisé avec une query collapse permet de visualiser les documents des groupes
- Des paramètres additionnels peuvent être précisés :
  - **expand.sort** : Le tri utilisé à l'intérieur d'un groupe (par défaut le score).
  - **expand.rows** : Le nombre de documents à l'intérieur d'un groupe à retourner (Par défaut 5)

# Auto-complétion

# Plusieurs types d'autocomplétion

## 1. Recherche instantanée

Chaque proposition correspond à un résultat de recherche

## 2. Suggestion de recherche basée sur les logs

Chaque proposition correspond à une recherche fréquemment effectuée (à la Google)

## 3. Suggestion de recherche basée sur l'index

Chaque proposition correspond à un terme présent dans l'index

## 4. Suggestion de valeur de facette

Chaque proposition correspond à une valeur de facette possible (avec le nom de la facette indiquée)

# Suggestions basées sur l'index :

## solution 1 : *facet.prefix*

- Les facettes peuvent être utilisées pour implémenter l'autocomplete
- Attention, prends beaucoup de mémoire !
- Recherche « michael ja »
  - Les premiers mots de la requête sont pris comme une query normale
  - q=michael
  - On facette sur le champ texte :
  - facet=on
  - facet.field=text
  - facet.limit=5 (on veut 5 propositions)
  - facet.mincount=1 (on ne veut que des termes pour lesquels il y a effectivement un résultat)
  - facet.sort=count (le défaut) pour avoir les termes les plus utilisés au début de la liste
  - Le dernier mot incomplet est pris comme préfixe de facette, on ne veut que les valeurs de facette qui commence par ces lettres
  - facet.prefix=ja
  - On ne veut retourner que les résultats des facettes, pas les documents résultats !
- rows=0

# Suggestions basées sur l'index : solution 1 : facet.prefix

.../solr/core01/select/?

q=michael&facet=on&rows=0&facet.limit=5&facet.mincount=1&facet.field=text  
&facet.sort=count&facet.prefix=ja&qt=myHandler&wt=json&indent=on&

```
{ "responseHeader" : {
 "status":0,
 "QTime":5},
 "response":{"numFound":2498,"start":0,"docs":[]},
 "facet_counts":{"
 "facet_queries":{},
 "facet_fields": {
 "text":[
 "jackson",18,
 "james",16,
 "jason",4,
 "jay",4,
 "jane",3]
 }
 }
 }
}
```



# Suggestions basées sur l'index :

## solution 2 : Suggester

- Suggester : basé sur le component « Spellcheck »
  - Désavantage par rapport à *facet.prefix* : propose tous les termes, pas uniquement ceux valides par rapport au début de la recherche
- Nécessite de définir un nouveau « component », qu'il faut ajouter dans un handler
- Peut lire ses valeurs depuis un fichier (mais attention aux problèmes de caractères accentués)
- Voir <http://wiki.apache.org/solr/Suggester>

# Suggestions basées sur l'index : solution 2 : Suggester

```
<searchComponent name="suggest" class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str name="name">suggest</str>
 <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
 <str name="lookupImpl">org.apache.solr.spelling.suggest.tst.FSTLookupFactory</str>
 <str name="field">name</str> <!-- nom du champ d'index -->
 <float name="threshold">0.005</float>
 <str name="buildOnCommit">true</str>
 <!-- <str name="sourceLocation">chemin-vers-un-fichier.txt</str> -->
 </lst>
</searchComponent>

<requestHandler name="/suggest" class="org.apache.solr.handler.component.SearchHandler">
 <lst name="defaults">
 <str name="spellcheck">true</str>
 <str name="spellcheck.dictionary">suggest</str>
 <str name="spellcheck.onlyMorePopular">true</str>
 <str name="spellcheck.count">5</str>
 <str name="spellcheck.collate">true</str>
 </lst>
 <arr name="components">
 <str>suggest</str>
 </arr>
</requestHandler>
```

# Recherche géographique

# Introduction

- Solr a du support pour des recherches géographiques.
- Il permet de :
  - Indexer des points ou des formes
  - Filtrer des résultats via des distance ou des formes
  - Trier ou influencer le score via la distance ou la surface d'intersection entre 2 formes
  - Générer des données pour le tracing de points ou des agrégations thermiques sur une carte.

# Types de données

- Il y a 3 principaux types de données liés à la géoloc. :
  - ***LatLonPointSpatialField*** : Le plus commun, représente un couple latitude, longitude
  - ***SpatialRecursivePrefixTreeFieldType*** (RPT), incluant *RptWithGeometrySpatialField* . Formation GeoJSON et autres
  - ***BBoxField*** : Permettant de stocker des formes

# Indexation

- Pour indexer des points géographiques
  - Schéma :

```
<fieldType name="location" class="solr.LatLonPointSpatialField" docValues="true"/>
```
  - Indexation :
    - fournir la latitude, longitude séparées des virgules
    - Utiliser le paramètre *format* et fournir les coordonnées en :
      - WKT
      - GeoJSON

# Filtres

- Lucene/SolR propose 2 filtres spatiaux permettant de filtrer des documents en fonction de leurs coordonnées géographiques :
  - **geofilt** retourne les documents à partir de leur distance à un point d'origine  
I.e les documents dont les coordonnées géographiques sont incluses dans un cercle autour d'un point d'origine
  - **bbox** retourne les documents dont les coordonnées géographiques sont incluses dans un carré autour d'un point d'origine

# Recherche spatiale

- Les paramètres pour ces 2 filtres sont :
  - **d** : la distance (unité par défaut : km, ou précisée par la configuration *distanceUnits*).
  - **pt** : Le point central avec le format "lat,lon"
  - **sfield** : Le champ spatial.
  - **score** : Indique le mode de calcul du score. Par exemple : *kilometers* ou *overlapRatio*
  - **filter** : Si false, la requête ne filtre pas mais est utilisée pour le calcul du score



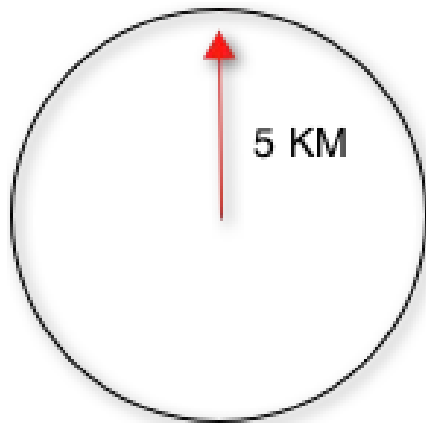
# Exemples

# Distance de 5km par rapport à un point central

&q=\*:\*&fq={!geofilt sfield=store}&pt=45.15, -93.85&d=5

# Carré autour d'un point d'origine (+rapide)

&q=\*:\*&fq={!geofilt sfield=store}&pt=45.15, -93.85&d=5



# Rectangle arbitraire

- Il est également possible de filtrer les documents par un rectangle arbitraire en utilisant une *range query*.
  - La première valeur correspond aux coordonnées du coin haut-gauche
  - La seconde au coin bas-droit

`&q=*:*&fq=store:[45, -94 TO 46, -93]`

# Tri et fonctions

- Il y a 4 fonctions se basant sur les distances, la plus appropriée est ***geodist***

## # Utilisation comme clé de tri

```
&q=*:*&fq={!geofilt}&sfield=store&pt=45.15, -93.85&d=50&sort=geodist() asc
```

## # Utilisation dans le score

```
&q={!func}geodist(&sfield=store&pt=45.15, -93.85&sort=score+asc&fl=*, score
```

## # Facettes par distance

```
&q=*:*&sfield=store&pt=45.15, -93.85&
facet.query={!frange l=0 u=5}geodist(&
facet.query={!frange l=5.001 u=3000}geodist()
```

# *RPT vs LatLonPoint*

- RPT apporte des améliorations fonctionelles vis à vis du type *LatLonPointSpatialField* :
  - Prise en charge de coordonnées non-géographiques (*geo=false*). Juste des x & y
  - Requêtes via des polygones et autres formes, en plus des cercles et des rectangles
  - Possibilité d'indexer des formes (polygones)
  - Agrégation de type Heatmap

# Options de Configuration

- **geo** : *true* ou *false*. Les calculs de distance sont alors différentes.
- **format** : Définit le format de description des formes. Par défaut WKT, mais possible de configurer pour GeoJSON
- **distanceUnits** : Unité de distance. Par défaut *kilometers* si *geo=true*, *degrees* sinon
- **distErrPct** : Définit la précision (influe sur la taille de l'index et la performance de la recherche), une fraction 0.0 (complètement précis) jusqu'à 0.5 .
- **maxDistErr** : Le plus haut niveau de détail pour les données indexées.  
Par défaut : 1m
- **distCalculator** : Algorithme de calcul de distance . Par défaut : si *geo=true* , *haversine* sinon *cartesian*
- **prefixTree** : Définit l'implémentation de grille. Par défaut : si *geo=true* *geohash* , sinon *quad*.
- **maxLevels** : Profondeur maximale de grille pour les données indexés.

# Formes prédéfinies

- Le champ RPT supporte des formes standard : points, cercles, rectangles, lignes, polygones, ...
- En sous-main, la librairie Spatial4j est utilisée
- Exemples d'indexation :

```
<field name="rptField">CIRCLE(28.57,77.32 d=0.051)</field>
```

```
<field name="rptField">POLYGON(20 50, 18 60, 24 68, 26 22, 30
55, 20 50)</field>
```

# Agrégations de type *heatmap*

- Un champ RPT permet d'agréger des documents en utilisant une grille associée à la carte ou l'espace. Tous les documents de la même cellule sont agrégés.
  - Les cellules de grilles sont déterminées à l'indexation.
  - La précision d'agrégation est fournie à la requête
- *Solr* retourne les données sous forme de tableau d'entiers à 2 dimensions ou au format PNG
- Cette fonctionnalité étend la fonctionnalité de facettes

# Paramètres

- ***facet*** : *true* pour activer l'agrégation.
- ***facet.heatmap*** : le nom du champ RPT.
- ***facet.heatmap.geom*** : La région à prendre en compte (top-left, bottom-right). Exemple : `["-180 -90" TO "180 90"]` .
- ***facet.heatmap.gridLevel*** : Le niveau d'agrégation. Une valeur par défaut est calculée
- ***facet.heatmap.format*** : le format .



# Réponse

- La réponse rappelle la dimension de la grille et fournit le décompte pour chaque cellule

```
{gridLevel=6,columns=64,rows=64,minX=-180.0,
maxX=180.0,minY=-90.0,maxY=90.0,counts_ints2D=[[0, 0, 2,
1,],[1, 1, 3, 2, ...],...]}
```

# BBox

- Le type Bbox permet d'associer un rectangle à un document
- Il supporte les recherches spatiales et permet d'affiner la pertinence par la zone d'intersection entre 2 formes

```
<fieldType name="bbox" class="solr.BBoxField"
geo="true" distanceUnits="kilometers"
numberType="pdouble" />
```

```
<fieldType name="pdouble" class="solr.DoublePointField"
docValues="true"/>
```

# Indexation et recherche

- Pour indexer un document, la syntaxe WKT est nécessaire :

ENVELOPE(-10, 20, 15, 10)

- Pour rechercher :

```
&q={!field f=bbox
score=overlapRatio}Intersects(ENVELOPE(-10, 20, 15,
10))
```

# Elévation

# Component

## « QueryElevation »

- Consiste à tuner query par query les résultats en remontant certains documents au début de la liste
- Ne remplace pas une bonne configuration d'index ! Scénarios :
  - Corrections d'erreurs évidentes sur des recherches populaires
  - Mots-clés payants pour un site de pages jaunes
  - Choix éditoriaux pour un site de news

# QueryElevation : component

```
<searchComponent name="elevator"
class="org.apache.solr.handler.component.QueryElevationComponent">
 <!-- fieldType pour comparer 'q' au fichier de config -->
 <str name="queryFieldType">text</str>
 <!-- Référence au fichier de config dans conf -->
 <str name="config-file">elevate.xml</str>
 <!-- Force l'élévation indépendamment du param 'sort' -->
 <str name="forceElevation">true</str>
</searchComponent>

<requestHandler name="/elevate" class="solr.SearchHandler">
 <lst name="defaults">
 <str name="echoParams">explicit</str>
 </lst>
 <arr name="last-components">
 <str>elevator</str>
 </arr>
</requestHandler>
```

# QueryElevation : fichier de config

```
<elevate>
 <query text="education">
 <doc id="A" />
 <doc id="B" />
 <doc id="C" exclude="true" />
 </query>

 <query text="africa">
 <doc id="D" />
 </query>

 <!-- etc... -->
</elevate>
```

# Production

Configuration pour la production  
SolRCloud



# Mise en service

- SolR fournit pour CentOS, Debian, Red Hat, SUSE and Ubuntu Linux :  
`bin/install_solr_service.sh`
- Choix : SolR HOME et le user associé
- Recommandation : Séparer les logs et les index de la distribution

# Dimensionnement mémoire

- Maximum Heap size à 10 ou 20 Go est courant sur les environnements de production

Le script *bin/oom\_solr.sh* est exécuté lors d'un OutOfMemory  
Il prévient ZooKeeper dans une configuration Cloud

- Dans le fichier include :  
`SOLR_JAVA_MEM="-Xms10g -Xmx10g"`

# Approches liés à la sécurité

- Différentes approches sont possibles pour sécuriser Solr
  - Utiliser un plugin pour l'authentification et l'autorisation
    - Basic authentication: SolrCloud seulement.
    - Kerberos authentication: SolrCloud ou mode standalone
    - Rule-based authorization: SolrCloud seulement.
    - PKI authentication: SolrCloud seulement – pour sécuriser le trafic inter-noeud
  - Utiliser SSL
  - Contrôler les accès via ZooKeeper (Solr Cloud)

# Mise en place des plugins

- La mise en place de plugins pour l'authentification et les autorisations est différente selon le mode de SolR :
  - SolRCloud : un fichier *security.json* doit être créé et chargé dans ZooKeeper avant utilisation
  - Mode standalone : La propriété système `-DauthenticationPlugin=<pluginClassName`

# Démarrer Solr sur HDFS

- Il est possible que Solr stocke ses index et ses logs de transactions sur un système de fichier HDFS (Apache Hadoop)
- Exemple standalone :  
`bin/solr start -`  
`Dsolr.directoryFactory=HdfsDirectoryFactor`  
`y`  
`-Dsolr.lock.type=hdfs`  
`-Dsolr.data.dir=hdfs://host:port/path`  
`-Dsolr.updatelog=hdfs://host:port/path`

# Distribution et réplication

Distribution et réplication  
SolrCloud : Concepts  
Configuration pour la production

# Introduction à la distribution

- Si les recherche commencent à être longues ou si la taille de l'index approche les limites de l'infrastructure, SolR propose de distribuer un index sur plusieurs serveurs
- L'index est divisé en plusieurs parties : les shards.
- La recherche s'effectue en parallèle sur chaque shard et les résultats sont ensuite agrégés

# Introduction à la réplication

- Un index ou shard peut être répliqué. La réplication d'un index est intéressante :
  - La charge d'un serveur est trop importante pour une seule machine. La charge peut alors être équilibrée sur plusieurs répliques "read-only".
  - Le débit des requêtes d'indexation consomme trop de ressources et pénalise les recherches. On sépare alors l'indexation de la recherche.
  - Pour tout simplement faire un backup



# Legacy vs SolrCloud

- SolR seul permet la distribution et la réplication
- Cependant, pour avoir un réel cluster de noeuds Solr avec équilibrage de charge et tolérance aux pannes, il vaut mieux se tourner vers SolrCloud

# SolR Cloud

# Introduction

- ***SolrCloud*** permet de mettre en place un cluster de serveurs Solr permettant la tolérance aux pannes et la scalabilité.
- Les caractéristiques de la solution :
  - Configuration centralisée de l'intégralité du cluster
  - Equilibrage de charge et fail-over pour les requêtes
  - Intégration de *ZooKeeper* pour la coordination et la configuration des noeuds.

# ZooKeeper

- Solr utilise *ZooKeeper* pour coordonner les nœuds du cluster
- Les recherches et les demandes d'indexations peuvent être effectuées sur n'importe quel nœud du cluster
  - Solr utilise les informations de la BD ZooKeeper pour déterminer quel serveur doit traiter la requête.

# Collections et Shards

- Un cluster peut héberger plusieurs **Collections** de Documents.
- Une collection peut être partitionnée en plusieurs **shards**, contenant un sous-ensemble des Documents de la Collection.
- Le nombre de Shards d'une Collection détermine :
  - La limite théorique du nombre de documents qu'une Collection peut contenir.
  - Le degré de parallélisation possible pour chaque requête.

# Noeuds et répliques

- Un Cluster est constitué de un ou plusieurs noeuds exécutant le serveur Solr.
  - Chaque noeud peut héberger une réplique physique d'un shard
  - Chaque réplique utilise la même configuration spécifiée pour sa Collection
- Le nombre de répliques d'un shard détermine :
  - Le niveau de redondance et la tolérance aux pannes d'un noeud
  - La limite théorique du nombre de requête concurrente

# Réplique leader et indexation

- Chaque shard a au moins une réplique : la réplique leader automatiquement élue.
- Lors de l'indexation,
  - La requête parvient à un des noeuds du cluster qui détermine le Shard auquel appartient le document.
  - Le document est ensuite transmis au noeud hébergeant la réplique leader
  - Le leader fait ensuite suivre la mise à jour aux autres répliques

# Routage de documents

- La stratégie de routing des documents est indiquée par le paramètre ***router.name*** lors de la création de la collection. Il peut prendre les valeurs :
  - ***compositeld*** ( défaut) : calcule une clé de hash à partir de l'ID du document. Il est possible de fournir un préfixe à l'ID pour contrôler le routage.  
Ex : IBM!12345
  - ***implicit*** : Permet d'indiquer le champ (*router.field*) qui servira à router. Si le champ n'est pas présent, le document est rejeté



# Recherche distribuée

- Lorsqu'un noeud Solr reçoit une requête de recherche, elle est routée vers une réplique appartenant à la collection.
- Le noeud agit alors comme un aggregateur :
  - Il crée des requêtes internes qu'il route vers un réplique de chaque shard de la collection
  - Il coordonne les réponses et peut effectuer d'autres requêtes pour compléter la réponse
  - Finalement, il construit la réponse finale pour le client
- Si le paramètre ***debug=track***, la requête est tracée et des informations de temps sont disponibles pour chaque phase de la requête distribuée.

# Limitation à des shards

- Dans certains cas, la recherche peut être limitée à certains shards. Les performances sont alors accrues

```
http://localhost:8983/solr/gettingstarted/select?
q=*:*&shards=shard1, shard2
```

```
http://localhost:8983/solr/gettingstarted/select?
q=*:*&shards=localhost:7574/solr/
gettingstarted, localhost:8983/solr/gettingstarted
```

- Ou utiliser le paramètre *\_route\_*

# Répartition de charge

- Pour répartir la charge sur les noeuds du cluster, il faut un répartiteur externe sachant interroger et lire les méta-données stockées dans *ZooKeeper*
- Solr fournit un client Java : ***CloudSolrClient*** qui utilise les données de ZooKeeper pour répartir la charge sur les noeuds up du cluster

# Tolérance aux pannes en écriture

- Un journal de transaction est créé pour chaque noeud permettant de rejouer les mises à jour en cas de crash. Le journal est remis à zéro lors d'un hard commit
- Lors de la création d'une réplique, le journal du Leader est utilisé pour synchroniser la réplique.
- Lors du crash d'un leader, il se peut que certaines répliques n'est pas les dernières mises à jour, après la réélection du leader un processus de synchronisation s'assure que toutes les répliques sont cohérentes

# Tolérance aux pannes en lecture

- Tant qu'au moins une réplique de chaque shard est accessible
- Possibilité d'accepter des résultats incomplets ou incertain. Paramètre ***shards.tolerant*** :
  - ***zkConnected*** : Tous les shards sont accessibles et le noeud servant la requête doit pouvoir obtenir des informations correctes auprès de ZooKeeper
  - ***false*** : Erreur si un des shards n'est pas disponible, même si il est impossible de connecter zooKeeper
  - ***true*** : Réponse même si un shard n'est pas accessible

# Changer le nombre de shards

- Le nombre de shards ne peut être changer sans de lourdes conséquences :  
Création de nouveau coeurs et réindexation
- Solr propose quand même de diviser un shard existant en 2. (API Collections)
  - Dans ce cas, 2 nouveaux coeurs sont créés et le shard de départ n'est pas touché.
  - Un fois l'opération terminée le shard original peut être supprimé

Mise en place

# Etapes

- Installation ZooKeeper
  - Décompresser
  - Créer un répertoire de stockage de données (*/var/lib/zookeeper*)
  - Créer un fichier de configuration *zoo.cfg*
- Démarrer ZooKeeper
- Lancer les nœuds Solr avec l'option *-k*



# Cluster de *ZooKeeper*

- SolR propose un un serveur ZooKeeper embarqué. Par défaut, le premier noeud du cluster le démarre. Problème : Si le noeud crash il n'y a plus de serveur *ZooKeeper*
- En production, il est nécessaire d'installer un cluster de ZooKeeper externe qui apporte le fail-over.
- Pour que le cluster fonctionne, il lui faut un quorum de serveurs up  
=> En général 3 noeuds *ZooKeeper* est un bon chiffre, cela permet la tolérance d'une panne d'un des noeuds

# Configuration : zoo.cfg

```
Vérification si les serveurs sont up toutes les 2s
tickTime=2000
Répertoire de stockage
dataDir=/var/lib/zookeeper
Port d'écoute
ClientPort=2181
Ensemble ZooKeeper
En nombre de ticks, le temps autorisé pour le démarrage et pour les synchros
initLimit=5
syncLimit=2
Adresses de tous les serveurs
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
#
autopurge.snapRetainCount=3
autopurge.purgeInterval=1
```

# *chroot*

- L'ensemble ZooKeeper peut être utilisé par d'autres applications que SolR
- Dans ce cas, il faut créer un ***znode***  
`bin/solr zk mkroot /solr -z zk1:2181,zk2:2181,zk3:2181`
- Ensuite le *znode* sera ajouté à la string de connection à ZooKeeper

# Mise en service, serveurs SolR

- SolR fournit pour CentOS, Debian, Red Hat, SUSE and Ubuntu Linux :  
*bin/install\_solr\_service.sh*
- Choix : SolR HOME et le user associé
- Recommandation : Séparer les logs et les index de la distribution

# Dimensionnement mémoire

- Maximum Heap size à 10 ou 20 Go est courant sur les environnements de production

Le script *bin/oom\_solr.sh* est exécuté lors d'un *OutOfMemory*  
Il prévient ZooKeeper dans une configuration Cloud

- Dans le fichier `include` :  
`SOLR_JAVA_MEM="-Xms10g -Xmx10g"`

# *Configuration ZooKeeper pour SolR*

- Il faut indiquer à SolR où se trouvent les serveurs ZooKeeper
- Soit en ligne de commande :

```
bin/solr start -e cloud -z zk1:2181,zk2:2181,zk3:2181/solr
```

- Soit dans le fichier de configuration *`solr.in.sh`* (.cmd) de SolR

```
ZK_HOST="zk1:2181, zk2:2181, zk3:2181/
solr"
```

# Rôle de ZooKeeper et fichiers de configuration

- Les fichiers de configuration SolrCloud sont conserés par ZooKeeper. Ils sont chargés lorsque :
  - l'on démarre SolrCloud via le script bin/solr
  - l'on crée une collection via le script bin/solr.
  - Lors d'un chargement explicite

# Création de collections

- Usage:  
`solr create_collection [-c collection]  
[-d confdir] [-n configName] [-shards  
#] [-replicationFactor #] [-p port]`



# Collections API

- Création

`/admin/collections?action=CREATE&name=name`

- Principaux paramètres :

- *router.name*
- *numShards*
- *replicationFactor*
- *maxShardsPerNode*
- ...

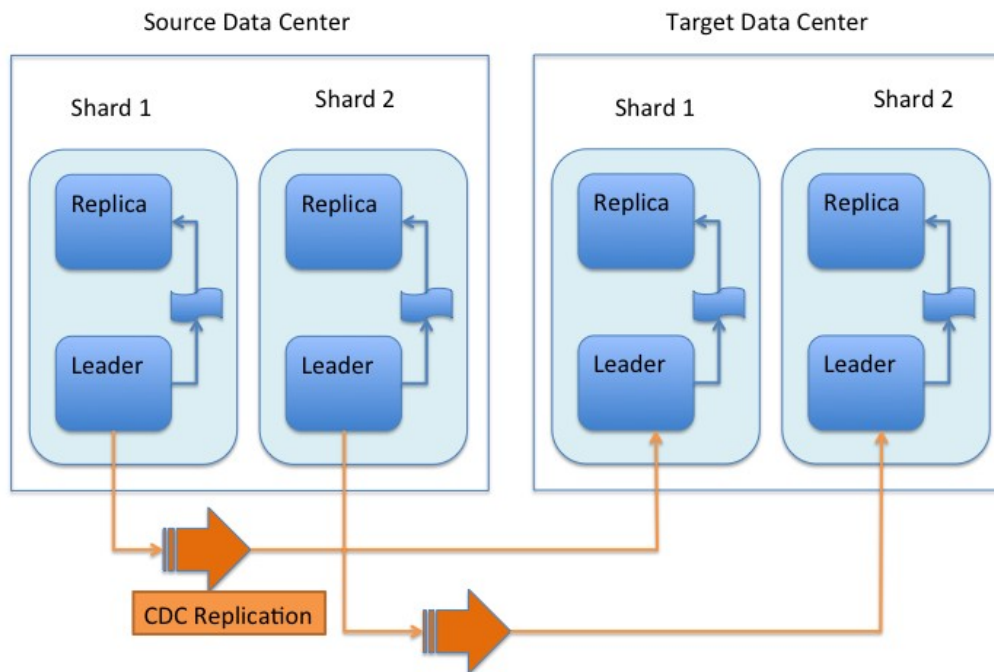
# Collections API (2)

- Mise à jour  
`/admin/collections?`  
`action=MODIFYCOLLECTION&collection=name`
- Attributs pouvant être modifiés :
  - *replicationFactor*
  - *maxShardsPerNode*
  - ...
- Rechargement  
`/admin/collections?action=RELOAD&name=name`

# Cross Data Center Replication

- CDCR permet plusieurs scénarios :
  - Répliquer une collection vers une autre collection
    - À l'intérieur d'un même cluster
    - Entre 2 clusters distinct
  - Synchroniser 2 cluster distincts. (Les écritures peuvent se faire indifféremment sur les clusters)

# Réplication



# Configuration

- L'adresse de ZooKeeper est la seule information requise pour la communication avec le cluster cible
- La configuration peut être affinée par :
  - Le dimensionnement du nombre de threads de réplication
  - La configuration du batch
  - ...

# Configuration source

```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
 <lst name="replica">
 <str name="zkHost">10.240.18.211:2181,10.240.18.212:2181/solr</str>
 <str name="source">collection1</str>
 <str name="target">collection1</str>
 </lst>
 <lst name="replicator">
 <str name="threadPoolSize">8</str>
 <str name="schedule">1000</str>
 <str name="batchSize">128</str>
 </lst>
 <lst name="updateLogSynchronizer">
 <str name="schedule">1000</str>
 </lst>
</requestHandler>
<!-- Modify the <updateLog> section of your existing <updateHandler> in your config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
 <updateLog class="solr.CdcrUpdateLog">
 <str name="dir">${solr.ulog.dir}</str>
 <!--Any parameters from the original <updateLog> section -->
</updateLog>
<!-- Other configuration options such as autoCommit should still be present -->
</updateHandler>
```

# Configuration cible

```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
 <!-- recommended for Target clusters -->
 <lst name="buffer">
 <str name="defaultState">disabled</str>
 </lst>
</requestHandler>
<requestHandler name="/update" class="solr.UpdateRequestHandler">
 <lst name="defaults">
 <str name="update.chain">cdcr-processor-chain</str>
 </lst>
</requestHandler>
<updateRequestProcessorChain name="cdcr-processor-chain">
 <processor class="solr.CdcrUpdateProcessorFactory"/>
 <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
<!-- Modify the <updateLog> section of your existing <updateHandler> in your
config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
 <updateLog class="solr.CdcrUpdateLog">
 <str name="dir">${solr.ulog.dir}</str>
 <!--Any parameters from the original <updateLog> section -->
</updateLog>
<!-- Other configuration options such as autoCommit should still be present -->
```

# Colocation

- Un collection peut être liée à une autre via la propriété ***withCollection***
- Cela garantit que la collection sera allouée sur le même noeud que l'autre, permettant des jointures
- Attention, la collection n'a alors qu'un shard (nommé *shard1*)



# Sécurité, HDFS, Backup, Réplication

# Approches liés à la sécurité

- Différentes approches sont possibles pour sécuriser Solr
  - Utiliser un plugin pour l'authentification et l'autorisation
    - Basic authentication: SolrCloud seulement.
    - Kerberos authentication: SolrCloud ou mode standalone
    - Rule-based authorization: SolrCloud seulement.
    - PKI authentication: SolrCloud seulement – pour sécuriser le trafic inter-noeud
  - Utiliser SSL
  - Contrôler les accès via ZooKeeper (Solr Cloud)

# Mise en place des plugins

- La mise en place de plugins pour l'authentification et les autorisations est différente selon le mode de SolR :
  - SolRCloud : un fichier *security.json* doit être créé et chargé dans ZooKeeper avant utilisation
  - Mode standalone : La propriété système `-DauthenticationPlugin=<pluginClassName`

# Démarrer Solr sur HDFS

- Il est possible que Solr stocke ses index et ses logs de transactions sur un système de fichier HDFS (Apache Hadoop)
- Exemple standalone :  
`bin/solr start -`  
`Dsolr.directoryFactory=HdfsDirectoryFactor`  
`y`  
`-Dsolr.lock.type=hdfs`  
`-Dsolr.data.dir=hdfs://host:port/path`  
`-Dsolr.updatelog=hdfs://host:port/path`

# Backup / Restore

- 2 approches pour la sauvegarde et la restauration d'index !

- Collections API pour SolrCloud

/admin/collections?

**action=BACKUP**&name=myBackupName&collection=myCollectionName&location=/path/to/my/shared/drive

/admin/collections?**action=RESTORE**&name=myBackupName&location=/path/to/my/shared/drive&collection=myRestoredCollectionName

- Gestionnaire de réplication pour le mode standalone

# Gestionnaire de réplication

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
 <lst name="master">
 <str name="replicateAfter">optimize</str>
 <str name="backupAfter">optimize</str>
 <str name="confFiles">schema.xml,stopwords.txt,elevate.xml</str>
 <str name="commitReserveDuration">00:00:10</str>
 </lst>
 <int name="maxNumberOfBackups">2</int>
 <lst name="invariants">
 <str name="maxWriteMBPerSec">16</str>
 </lst>
</requestHandler>
```

<http://localhost:8983/solr/gettingstarted/replication?command=backup>

# Configuration des traces

- Interface d'administration
- Logging API :  
# Set the root logger to level WARN  
`curl -s http://localhost:8983/solr/admin/info/logging --data-binary "set=root:WARN&wt=json"`
- Au démarrage du serveur :
  - Variable d'environnement : *SOLR\_LOG\_LEVEL*
  - Options *-v* ou *-q*
- Changement total de la configuration :  
`server/resources/log4j.properties`
- Logger sous forme de WARN les requêtes lentes :  
`<slowQueryThresholdMillis>1000</slowQueryThresholdMillis>`