



Master in Computer Science

Advanced Databases



Project Search Engine Solr

Students

- **Mulham ARYAN**
- **Samia AZZOUZI**
- **Thomas Borel Kamdem Tagne**

Table Of Contents

1. Introduction

2. Relational Database

2.1 Benefits of Relational Databases

2.2 Challenges of Relational Databases

3. Non-Relational Databases

3.1 Non-Relational Database Features & Capabilities

4. Solr

4.1 Solr Features

4.2 Installation of Solr

4.3 Concepts & Terminology

4.4 Solr search applications

4.5 Solr Architecture

4.6 The principle of functioning

4.7 Configuration file

5. Indexing

5.1 Starting Solr

5.2 Indexing in Apache Solr

- Adding document using command line**
- Updating document**
- Deleting Document**
- Commit**

6. Searching in Solr

6.1 Querying data

6.2 Retrieving data

6.3 The Standard Query Parser Response

6.4 Filter Query

6.5 Field List

7. Conclusion

1- Introduction:

In nowadays applications, the amount of data in the database grows exponentially. So, the DBMS must process these huge amounts of data as fast as possible. So, at first, we will see the difference between Relational database and NoSQL then we will study one of the most popular NoSQL database Apache Solr

2- Relational Database:

A relational database management system (RDBMS or just RDB) is a common type of database whose data is stored in tables. You'll find that most databases used in businesses these days are relational databases, as opposed to a at le or hierarchical database. Relational databases have the clout to handle multitudes of data and complex queries, whereas a at le takes up more space and memory and is less efficient. So modern databases use multiple tables as standard. The data is stored in lots and lots of tables, or `relations. These tables are divided into rows (records) and columns (fields).

• 2.1 - Benefits of Relational Databases:

If you want to design a data storage system that makes it easy to manage lots of information, and is scalable and flexible, the relational database is a good bet.

Manageability: for starters, an RDB is easy to manipulate. Each table of data can be updated without disrupting the others. You can also share certain sets of data with one group but limit their access to others - such as confidential information about employees.

Flexibility: if you need to update your data, you only have to do it once - so no more having to change multiple les one at a time. And it's pretty simple to extend your database. If your records are growing, a relational database is easily scalable to grow with your data.

Avoid Errors: there's no room for mistakes in a relational database because it's easy to check for mistakes against the data in other parts of the records. And since each piece of information is stored at a single point, you don't have the problem of old versions of data clouding the picture

• 2.2 - Challenges of Relational Databases:

Scalability: Because relational databases are built on a single server. This means, in order to scale, you'll need to purchase more expensive hardware with more power, storage, and memory

Performance: Rapid growth in volume, velocity, variety, and complexity of data creates even more complicated relationships. Relational databases tend to have a hard time keeping up, which can slow down performance.

Relationships: databases don't actually store relationships between elements, which makes understanding connections between your data reliant on other joins

3- Non-Relational Databases:

Non-relational databases do not use the rows/columns table format of relational databases. They have different and varying frameworks of storing and modeling data. By relaxing certain rules these databases provide increased scalability and availability. The term "non-relational database" is sometimes used synonymously with NoSQL databases. Document-oriented databases, key-value databases, object databases and graph databases are non-relational databases.

Non-relational databases grew in popularity due to their ability to meet the aggressive scaling needs of web applications appearing on popular websites (e.g. social media). They also are suited to support Big Data applications with their high throughput of unstructured data. Non-relational databases can also store data in memory for persistence, to more easily read this fast-moving data. Finally, popular non-relational databases are open source and present little or no upfront cost, and no licensing fees

• 3.1 - Non-Relational Database Features & Capabilities:

Notable capabilities and advantages of non-relational database are:

- Can be purpose-built to specific data models
- "Table less" and opaque data storage
- Can manage unstructured or multi-structured data
- No need for a predefined schema
- Better manage abstract data
- Support graph data modeling
- Support document-oriented data store
- Less strict consistency (e.g. eventual consistency) models
- Less strict consistency (e.g. eventual consistency) models
- Better operational performance
- Require fewer computing resources
- More horizontal and vertical scalability

4- Solr

Apache Solr is an enterprise-capable, open source search platform based on the Apache Lucene search library. The Solr search engine is one of the most widely deployed search platforms worldwide.

Solr is written in Java, and provides both a RESTful XML interface, and a JSON API with which search applications can be built.

The Solr search engine enjoys a reputation for being extremely stable, scalable, and reliable. It is under constant development by a large community of open source committers, under the direction of the Apache Software Foundation.

It provides a rich set of core search functions. In experienced hands, it is an extremely powerful and flexible platform on which search-based and big data analytics applications can be built

- **4.1 Solr features:**

- Advanced Full-Text Search Capabilities
- Optimized for High Volume Traffic
- Standards Based Open Interfaces – XML, JSON and HTTP
- Comprehensive Administration Interfaces
- Easy Monitoring
- Highly Scalable and Fault Tolerant
- Flexible and Adaptable with easy configuration
- Near Real-Time Indexing
- Extensible Plugin Architecture
- Performance Optimizations
- Advanced Storage Options
- Multiple search indices
- Geospatial Search

- **4.2 Installation of Solr**

You can install Solr in any system where a suitable Java Runtime Environment (JRE) is available, as detailed below. Currently this includes Linux, OS X, and Microsoft Windows. The instructions in this section should work for any platform, with a few exceptions for Windows as noted.

- **Debian/Ubuntu**

```
# sudo apt-get -y install openjdk-7-jdk
# mkdir /usr/java
# ln -s /usr/lib/jvm/java-7-openjdk-amd64 /usr/java/default
# sudo apt-get -y install solr-tomcat
```

- **Centos**

```
# wget http://www-eu.apache.org/dist/lucene/solr/7.5.0/solr-7.5.0.tgz
# tar xzf solr-7.5.0.tgz solr-7.5.0/bin/install_solr_service.sh --strip-components=2
# sudo bash ./install_solr_service.sh solr-7.5.0.tgz
```

- **OS X**

```
#brew install java
#brew install solr
```

- **Windows**

Go to cmd prompt and check for JRE with correct version. If JRE is available in your system, it will show you the version.

```
C:/> java -version
```

Then download Solr from : <https://archive.apache.org/dist/lucene/solr/>

- Managing Solr

```
# sudo "service/systemctl" solr start // to start new solr default port 8983
# sudo "service/systemctl" solr status // list all working solr
# sudo "service/systemctl" solr stop // to stop solr service
```

• 4.3 Concepts & Terminology

Apache Lucene is a full text search engine library written entirely in Java. Lucene is embedded with Solr.

Apache Solr is an enterprise search platform written in Java. It exposes web services that can manage the lifecycle of documents in the index.

Document is Lucene/Solr's primary unit of storage - representing a at collection of fields (no nesting).

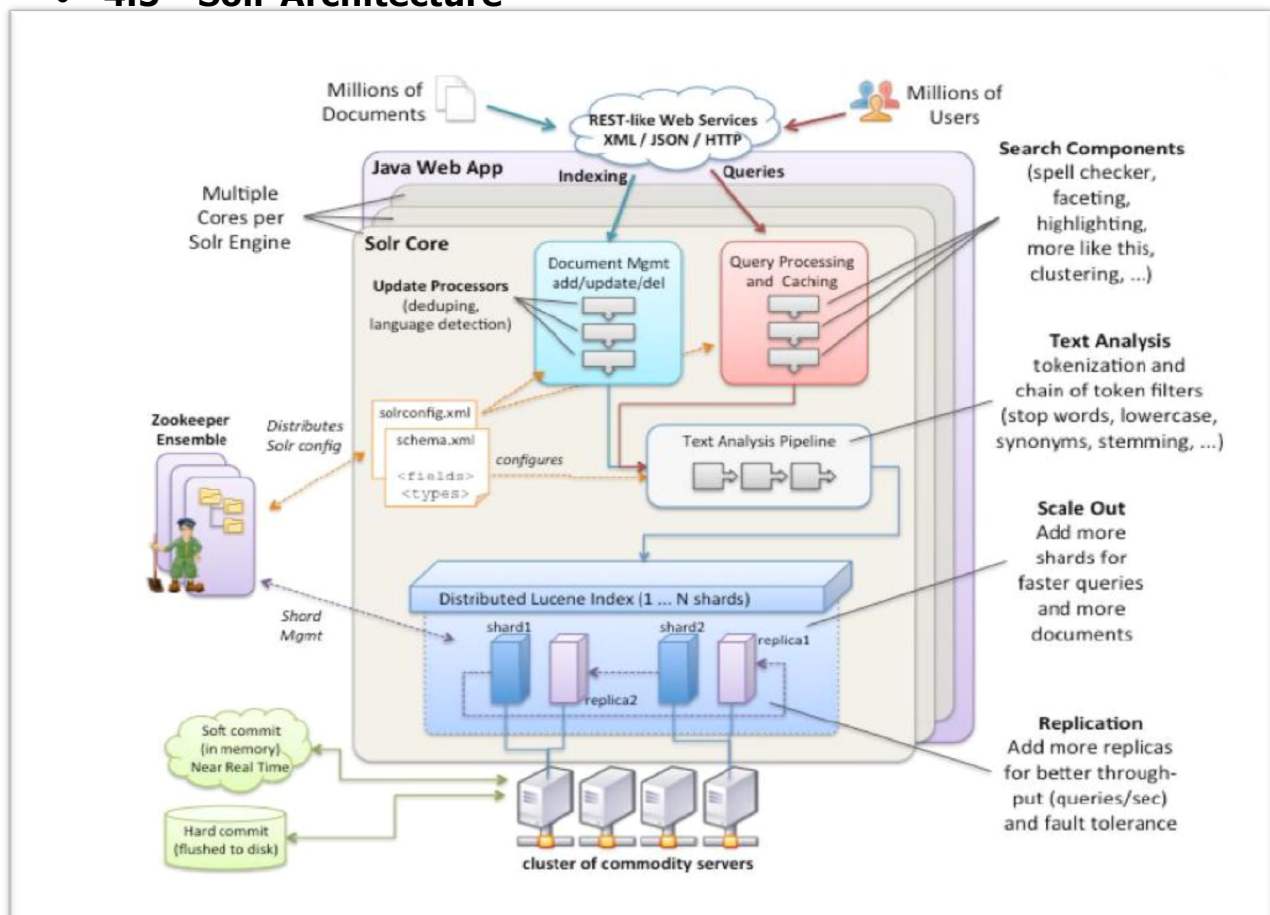
Field definition consists of a name and configurable type (text, integer, double, date).

Core separate index and configuration. A single server can support multiple cores and it is used for data partitioning. Supports multitenant applications.

• 4.4 Solr search applications:

Most Solr search applications are web-facing. However, it is also used as the basis for intranet and enterprise search solutions. In addition, Solr is often used in Hadoop-based "big data" projects, and it is the basis for products such as Cloudera Search.

• 4.5 - Solr Architecture



- **Request Handler** – The requests we send to Apache Solr are processed by these request handlers. The requests might be query requests or index update requests. Based on our requirement, we need to select the request handler. To pass a request to Solr, we will generally map the handler to a certain URI end-point and the specified request will be served by it.

- **Search Component** – A search component is a type (feature) of search provided in Apache Solr. It might be spell checking, query, faceting, hit highlighting, etc. These search components are registered as **search handlers**. Multiple components can be registered to a search handler.

- **Query Parser** – The Apache Solr query parser parses the queries that we pass to Solr and verifies the queries for syntactical errors. After parsing the queries, it translates them to a format which Lucene understands.

- **Response Writer** – A response writer in Apache Solr is the component which generates the formatted output for the user queries. Solr supports response formats such as XML, JSON, CSV, etc. We have different response writers for each type of response.

- **Analyzer/tokenizer** – Lucene recognizes data in the form of tokens. Apache Solr analyzes the content, divides it into tokens, and passes these tokens to Lucene. An analyzer in Apache Solr examines the text of fields and generates a token stream. A tokenizer breaks the token stream prepared by the analyzer into tokens.

- **Update Request Processor** – Whenever we send an update request to Apache Solr, the request is run through a set of plugins (signature, logging, indexing), collectively known as **update request processor**. This processor is responsible for modifications such as dropping a field, adding a field, etc.

- **Shard** – In distributed environments, the data is partitioned between multiple Solr instances, where each chunk of data can be called as a Shard. It contains a subset of the whole index.

• 4.6 - the principle of functioning

The functional principle is based on XML/ JSON / HTTP APIs. the dialogue with the server is via the HTTP protocol. So, to add a document to the index, we use the POST method. To search for documents, we use the GET method

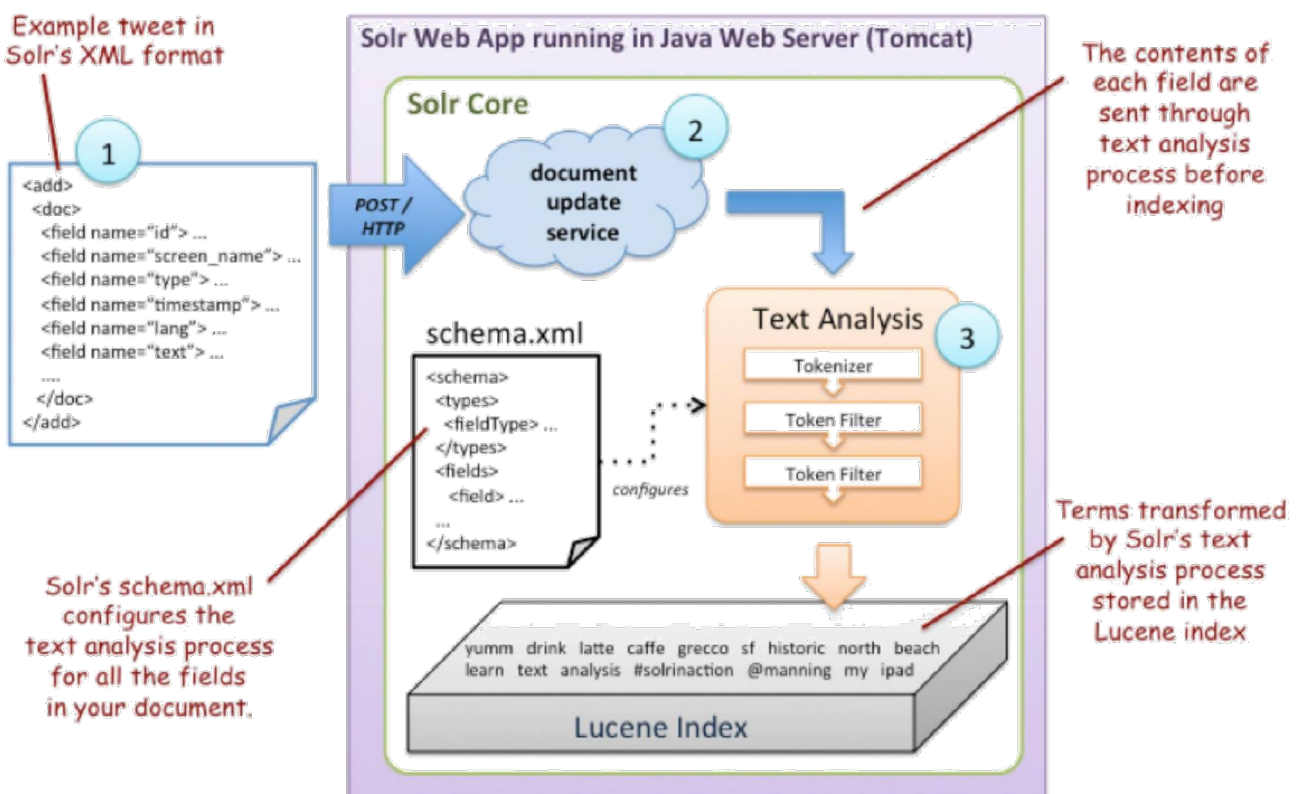
There are then two main entry points on the server, corresponding to two URLs, since we use HTTP:

URL add/update document:

`http://localhost:6464/solr/update`

URL select:

`http://localhost:6464/solr/select`



• 4.7 - Configuration file:

The main configuration files in Apache Solr are as follows

- **Solr.xml** – It is the file in the \$SOLR_HOME directory that contains Solr Cloud related information. To load the cores, Solr refers to this file, which helps in identifying them.

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

  <solrcloud>

    <str name="host">${host:}</str>
    <int name="hostPort">${jetty.port:8983}</int>
    <str name="hostContext">${hostContext:solr}</str>

    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>

    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <int name="distribUpdateSoTimeout">${distribUpdateSoTimeout:600000}</int>
    <int name="distribUpdateConnTimeout">${distribUpdateConnTimeout:60000}</int>
    <str name="zkCredentialsProvider">${zkCredentialsProvider:org.apache.solr.common.cloud.DefaultZkCredentialsProvider}</str>
    <str name="zkACLProvider">${zkACLProvider:org.apache.solr.common.cloud.DefaultZkACLProvider}</str>

  </solrcloud>

  <shardHandlerFactory name="shardHandlerFactory"
    class="HttpShardHandlerFactory">
    <int name="socketTimeout">${socketTimeout:600000}</int>
    <int name="connTimeout">${connTimeout:60000}</int>
  </shardHandlerFactory>

</solr>
```

- **Solrconfig.xml**: This file contains the definitions and core-specific configurations related to request handling and response formatting, along with indexing, configuring, managing memory and making commits.

- **Schema.xml**: This file contains the whole schema along with the fields and field types.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema name="example" version="1.6">

  <uniqueKey>id</uniqueKey>

  <fields>
    <field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
    <field name="cat" type="string" multiValued="true" indexed="true" stored="true"/>
    <field name="description" type="text_general" indexed="true" stored="true"/>
    <field name="links" type="string" multiValued="true" indexed="true" stored="true"/>
    <field name="name" type="text_general" indexed="true" stored="true"/>
    <field name="price" type="pfloat" indexed="true" stored="true"/>
  </fields>

  <dynamicField name="ignored_*" type="ignored" multiValued="true"/>
  <dynamicField name="random_*" type="random"/>
  <dynamicField name="*_txt_ko" type="text_ko" indexed="true" stored="true"/>
  <dynamicField name="*_s_ns" type="string" indexed="true" stored="false"/>
  <dynamicField name="*_l_ns" type="plong" indexed="true" stored="false"/>

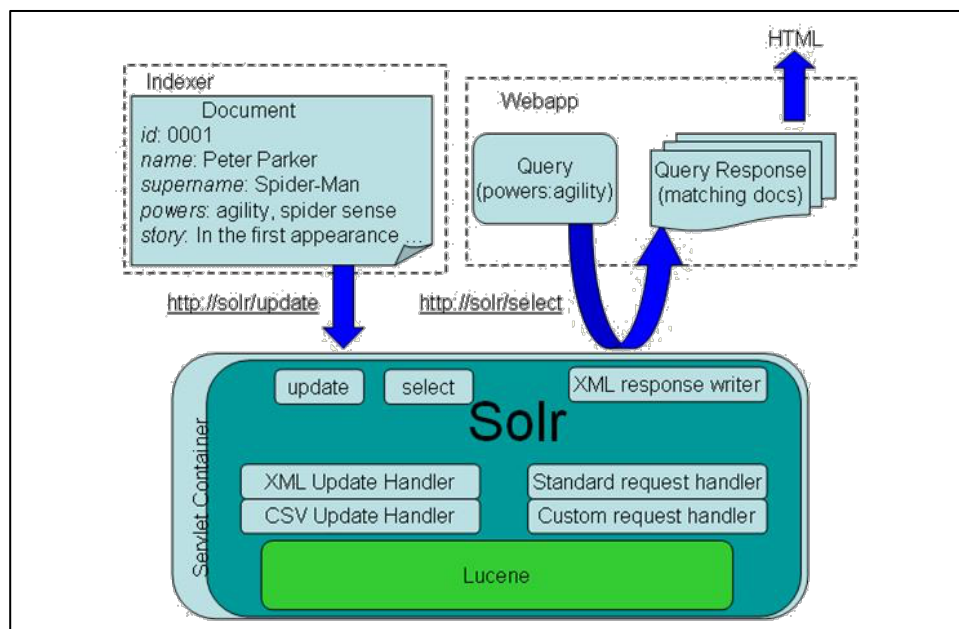
  <copyField source="author" dest="text"/>
  <copyField source="cat" dest="text"/>
  <copyField source="content" dest="text"/>
  <copyField source="content_type" dest="text"/>
  <copyField source="description" dest="text"/>
  <copyField source="features" dest="text"/>

  <fieldType name="string" class="solr.StrField" sortMissingLast="true"/>
  <fieldType name="text_ar" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.ArabicStemFilterFactory"/>
    </analyzer>
  </fieldType>

</schema>
```

- **a – Unique key (<uniqueKey>):** element specifies which field is a unique identifier for documents. Although uniqueKey is not required, it is nearly always warranted by your application design.
- **b – Fields (<fields>):** contains the set of fields.
- **c – Field (<field>):** Fields are defined in the field's element of schema.xml. Once you have the field types set up, defining the fields themselves is simple.
- **d – Dynamic field (<dynamicField>):** allow Solr to index fields that you did not explicitly define in your schema.
- **e – Copy field (<copyField>):** Solr has a mechanism for making copies of fields so that you can apply several distinct field types to a single piece of incoming information.
- **f – Field Type (<fieldType>):** A field type defines the analysis that will occur on a field when documents are indexed, or queries are sent to the index.

- **Core.properties** – This file contains the configurations specific to the core. It is referred for **core discovery**, as it contains the name of the core and path of the data directory. It can be used in any directory, which will then be treated as the **core directory**.



In the diagram above, in the left part, we want to index a document where it is identified by an id, a name and some other fields.

Below, is represented Solr, in its servlet container, with Lucene as a base. So, we go through the URL update, since it is an addition of a new document. Then, depending on whether the document is in XML or CSV format, a handler performs the analysis of the query. Most often, it is XML that is used.

The right-hand side is the document search request. So, we go through the Select URL, asking all the superheroes who hold the power of agility.

The documents that match the request are returned to the client application, which can for example transcribe the whole in HTML format to present the data.

5 - indexing:

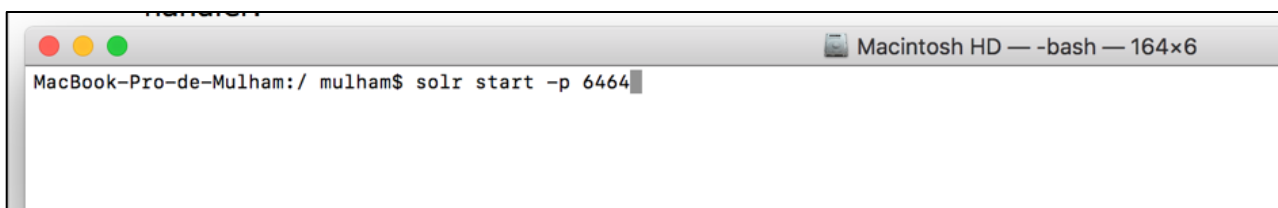
Solr maintains in his index a collection of documents. A document is a set of fields that are associated with values.

Solr is used to index XML and CSV files by default, but it is possible to index data from "Rich Documents" such as PDF, PPT, DOC or XLS, via the specific handler.

Solr offers the possibility of importing data from a database, or even an RSS feed. For the database, the columns of a table will become the fields of Document manipulates

- **5.1 – Starting Solr**

Using the terminal, we should start our Solr by specifying the port of our Solr



In our project we are using the port 6464 to communicate with our application

NB: We can use multiple Solr in one machine and we can use this command line to get all the Solr on our machine "**solr status**"

- **5.2 – Indexing in Apache Solr:**

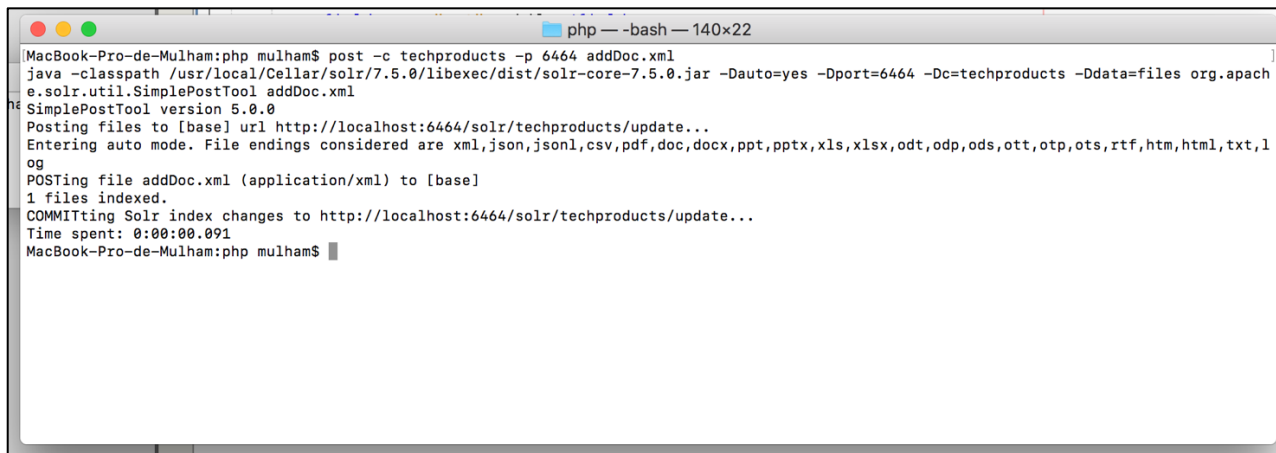
In Apache Solr, we can index (add, delete, modify) various document formats such as xml, csv, pdf, etc. We can add data to Solr index in several ways.

In this report we will discuss how to index data in Apache Solr using command line and our PHP API (there is a web interface integrated in Apache Solr also).

- **A – Adding document using command line:**

To add document, we should use **post** command we can index document in various formats as **XML, Json, CSV** etc ...

A.1 – Using command line



```
MacBook-Pro-de-Mulham:php mulham$ post -c techproducts -p 6464 addDoc.xml
java -classpath /usr/local/Cellar/solr/7.5.0/libexec/dist/solr-core-7.5.0.jar -Dauto=yes -Dport=6464 -Dc=techproducts -Ddata=files org.apache.solr.util.SimplePostTool addDoc.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:6464/solr/techproducts/update...
Entering auto mode. File endings considered are xml,json,jsonl,csv,pdf,doc,docx,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log
POSTing file addDoc.xml (application/xml) to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:6464/solr/techproducts/update...
Time spent: 0:00:00.091
MacBook-Pro-de-Mulham:php mulham$
```

```
#post -c "name_of_our_core" -p "our_port" our_doc.xml
```

Content of our xml document



```
<add>
  <doc>
    <field name="id">001</field>
    <field name="name">Samsung A6</field>
    <field name="description">Mobile samsung a6</field>
    <field name="cat">mobile</field>
    <field name="price">200</field>
    <field name="links">http://amazon.fr/samsunga6</field>
  </doc>

  <doc>
    <field name="id">002</field>
    <field name="name">iPhone 8</field>
    <field name="description">Mobile iPhone 8</field>
    <field name="cat">mobile</field>
    <field name="price">400</field>
    <field name="links">http://amazon.fr/iPhone8</field>
  </doc>

  <doc>
    <field name="id">003</field>
    <field name="name">MAC AIR</field>
    <field name="description">Macbook air 15pouce</field>
    <field name="cat">pc,macbook</field>
    <field name="price">800</field>
    <field name="links">http://amazon.fr/mac</field>
  </doc>
</add>
```

The add tag specifies that documents are added. Each document is then described between <doc> tags. Here we add the product, identified 001, whose name is Samsung A6 and which includes the category, the price and link of the web site.

As you can observe, the XML file written to add data to index contains three important tags namely, <add> </add>, <doc></doc>, and <field></field>.

- **add** – This is the root tag for adding documents to the index. It contains one or more documents that are to be added.
- **doc** – The documents we add should be wrapped within the <doc></doc> tags. This document contains the data in the form of fields.
- **field** – The field tag holds the name and value of the fields of the document.

After preparing the document, you can add this document to the index.

A.2 - Using Solarium PHP API

```
<?php
2  include "vendor/autoload.php";
3  include "config.php";
4  $client = new Solarium\Client($config);
5
6  $update = $client->createUpdate();
7
8  // create a new document for the data
9  $doc1 = $update->createDocument();
10 $doc1->id      = 001;
11 $doc1->name    = 'Samsung A6';
12 $doc1->description = "Mobile samsung a6";
13 $doc1->cat     = "mobile";
14 $doc1->price   = "200";
15 $doc1->link    = "links";
16
17 $doc2 = $update->createDocument();
18 $doc2->id      = 002;
19 $doc2->name    = 'iPhone 8';
20 $doc2->description = "Mobile iPhone 8";
21 $doc2->cat     = "mobile";
22 $doc2->price   = "400";
23 $doc2->link    = "http://amazon.fr/iPhone8";
24
25 $doc3 = $update->createDocument();
26 $doc3->id      = 003;
27 $doc3->name    = 'MAC AIR';
28 $doc3->description = "Macbook air 15pouce";
29 $doc3->cat     = "pc,macbook";
30 $doc3->price   = "800";
31 $doc3->link    = "http://amazon.fr/mac";
32
33 $update->addDocuments(array($doc1, $doc2, $doc3));
34 $update->addCommit();
35
36
```

The same logic in previous example first we create the doc tag using createUpdate function then we put the value of each field as shown in the previous code.

- **B – Updating document**

B.1 – Using command line

Following is the XML file used to update a field in the existing document. Save this in a file with the name **addDoc.xml**.

```

1  <add>
2    <doc>
3      <field name="id">004</field>
4      <field name="name">Samsung A6</field>
5      <field name="description">Mobile samsung a6</field>
6      <field name="cat">mobile</field>
7      <field name="price">200</field>
8      <field name="links">http://amazon.fr/samsunga6</field>
9    </doc>
10 </add>
11
12

```

As you can observe, the XML file written to update data is just like the one which we use to add documents. But the only difference is we use the **update** attribute of the field.

In our example, we will use the above document and try to update the fields of the document with the id **004**.

Since we are updating the index which exists in the core named **techproducts**, you can update using the **post** tool as follows

```
#post -c techproducts -p 6464 addDoc.xml
```

B.2 – Updating using Solarium PHP API

```

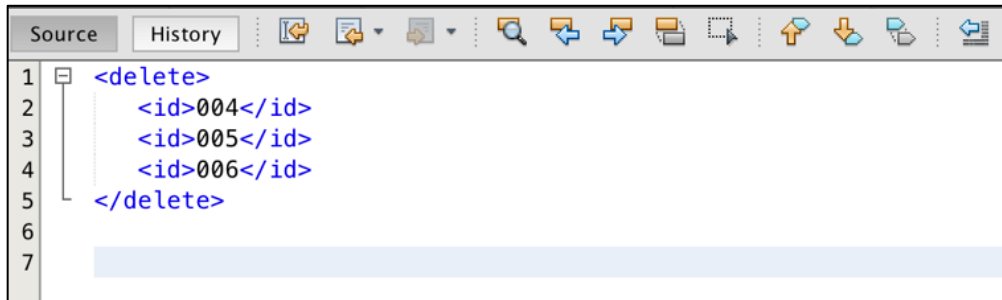
1  <?php
2      include "vendor/autoload.php";
3      include "config.php";
4      $client = new Solarium\Client($config);
5
6      $update = $client->createUpdate();
7
8      // create a new document for the data
9      $doc1 = $update->createDocument();
10     $doc1->id = 001;
11     $doc1->name = 'Samsung A6';
12     $doc1->description = "Mobile samsung a6";
13     $doc1->cat = "mobile";
14     $doc1->price = "200";
15     $doc1->link = "links";
16
17     $update->addDocuments(array($doc1));
18     $update->addCommit();
19

```

• C – Deleting Document

C.1 – Deleting document using command line

To delete documents from the index of Apache Solr, we need to specify the ID's of the documents to be deleted between the `<delete>` `</delete>` tags.



Here, this XML code is used to delete the documents with ID's **004** and **006**. Save this code in a file with the name **delDoc.xml**.

If you want to delete the documents from the index which belongs to the core named **techproducts**, then you can post the **delDoc.xml** file using the **post** tool, as shown below.

```
#post -c techproducts -p 6464 delDoc.xml
```

```

MacBook-Pro-de-Mulham:php mulham$ post -c techproducts -p 6464 delDoc.xml
java -classpath /usr/local/Cellar/solr/7.5.0/libexec/dist/solr-core-7.5.0.jar -Dauto=yes -Dport=6464 -Dc=techproducts -Ddata=files org.apache.solr.util.SimplePostTool delDoc.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:6464/solr/techproducts/update...
Entering auto mode. File endings considered are xml,json,jsonl,csv,pdf,doc,docx,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,txt,log
POSTing file delDoc.xml (application/xml) to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:6464/solr/techproducts/update...
Time spent: 0:00:00.083
MacBook-Pro-de-Mulham:php mulham$

```

C.2 – Deleting document using Solarium PHP API

Following is the Java program to add documents to Apache Solr index. Delete the document using **addDeleteQuery** function.

```

1  <?php
2      include "vendor/autoload.php";
3      include "config.php";
4      $client = new Solarium\Client($config);
5
6      $update = $client->createUpdate();
7
8      $update->addDeleteQuery('id:004');
9
10     $update->addCommit();
11

```

- **D – Commit**

Adding and removing is not enough to completely add or delete a document. It is obligatory to proceed **commit** to make visible any index update.

Using command line: **<commit />**
 Using Solarium API: **\$this->addCommit();**

6 – Searching in Solr

Solr offers a rich, flexible set of features for search. To understand the extent of this flexibility, it's helpful to begin with an overview of the steps and components involved in a Solr search.

When a user runs a search in Solr, the search query is processed by a **request handler**. A request handler is a Solr plug-in that defines the logic to be used when Solr processes a request. Solr supports a variety of request handlers. Some are designed for processing search queries, while others manage tasks such as index replication.

Search applications select a particular request handler by default. In addition, applications can be configured to allow users to override the default selection in preference of a different request handler.

To process a search query, a request handler calls a **query parser**, which interprets the terms and parameters of a query. Different query parsers support different syntax. Solr's default query parser is known as the Standard Query Parser, or more commonly just the "lucene" query parser. Solr also includes the DisMax query parser, and the Extended DisMax (eDisMax) query parser. The standard query parser's syntax allows for greater precision in searches, but the DisMax query parser is much more tolerant of errors. The DisMax query parser is designed to provide an experience similar to that of popular search engines such as Google, which rarely display syntax errors to users. The Extended DisMax query parser is an improved version of DisMax that handles the full Lucene query syntax while still tolerating syntax errors. It also includes several additional features.

In addition, there are common query parameters that are accepted by all query parsers.

Input to a query parser can include:

- search strings---that is, *terms* to search for in the index
- *parameters for fine-tuning the query* by increasing the importance of particular strings or fields, by applying Boolean logic among the search terms, or by excluding content from the search results
- *parameters for controlling the presentation of the query response*, such as specifying the order in which results are to be presented or limiting the response to particular fields of the search application's schema.

Search parameters may also specify a **filter query**. As part of a search response, a filter query runs a query against the entire index and caches the results. Because Solr allocates a separate cache for filter queries, the strategic use of filter queries can improve search performance. (Despite their similar names, query filters are not related to analysis filters. Filter queries perform queries at search time against data already in the index, while analysis filters, such as Tokenizers, parse content for indexing, following specified rules).

A search query can request that certain terms be highlighted in the search response; that is, the selected terms will be displayed in colored boxes so that they "jump out" on the screen of search results. **Highlighting** can make it easier to find relevant passages in long documents returned in a search. Solr supports multi-term highlighting. Solr includes a rich set of search parameters for controlling how terms are highlighted.

Search responses can also be configured to include **snippets** (document excerpts) featuring highlighted text. Popular search engines such as Google and Yahoo! return snippets in their search results: 3-4 lines of text offering a description of a search result.

To help users zero in on the content they're looking for, Solr supports two special ways of grouping search results to aid further exploration: faceting and clustering.

Faceting is the arrangement of search results into categories (which are based on indexed terms). Within each category, Solr reports on the number of hits for relevant term, which is called a facet constraint. Faceting makes it easy for users to explore search results on sites such as movie sites and product review sites, where there are many categories and many items within a category.

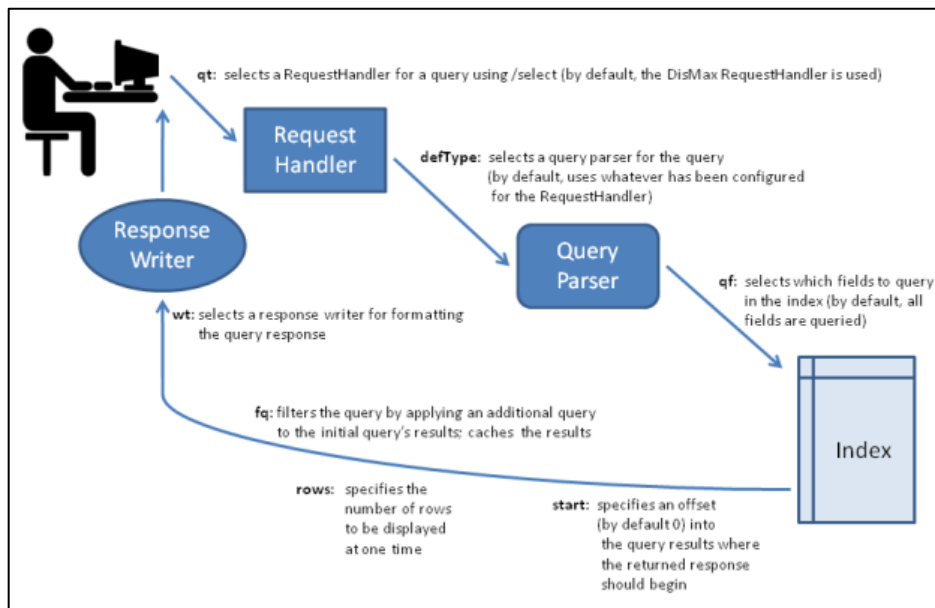
Faceting makes use of fields defined when the search applications were indexed. In the example above, these fields include categories of information that are useful for describing digital cameras: manufacturer, resolution, and zoom range.

Clustering groups search results by similarities discovered when a search is executed, rather than when content is indexed. The results of clustering often lack the neat hierarchical organization found in faceted search results, but clustering can be useful nonetheless. It can reveal unexpected commonalities among search results, and it can help users rule out content that isn't pertinent to what they're really searching for.

Solr also supports a feature called MoreLikeThis, which enables users to submit new queries that focus on particular terms returned in an earlier query. MoreLikeThis queries can make use of faceting or clustering to provide additional aid to users.

A Solr component called a **response writer** manages the final presentation of the query response. Solr includes a variety of response writers, including an XML Response Writer and a JSON Response Writer.

The diagram below summarizes some key elements of the search process.



• 6.1 – Querying data:

In addition to storing data, Apache Solr also provides the facility of querying it back as and when required. Solr provides certain parameters using which we can query the data stored in it.

In the following table, we have listed down the various query parameters available in Apache Solr.

Parameter	Description
q	This is the main query parameter of Apache Solr, documents are scored by their similarity to terms in this parameter.
fq	This parameter represents the filter query of Apache Solr the restricts the result set to documents matching this filter.
fl	This parameter specifies the list of the fields to return for each document in the result set.
wt	This parameter represents the type of the response writer we wanted to view the result.

You can see all these parameters as options to query Apache Solr. Visit the homepage of Apache Solr. On the left-hand side of the page, click on the option Query. Here, you can see the fields for the parameters of a query.

<http://localhost:6464/solr/#/techproducts/query>

The screenshot displays the Apache Solr Query interface for the 'techproducts' core. The browser address bar shows the URL `localhost:6464/solr/#/techproducts/query`. The interface is divided into a left sidebar and a main content area.

Sidebar (Left):

- Dashboard
- Logging
- Core Admin
- Java Properties
- Thread Dump
- techproducts (selected core)
- Overview
- Analysis
- Dataimport
- Documents
- Files
- Ping
- Plugins / Stats
- Query** (selected)
- Replication
- Schema
- Segments info

Main Content Area:

- Request-Handler (qt):** `/select`
- common**
- q:** `*:*`
- fq:** (empty text box with red minus and green plus icons)
- sort:** (empty text box)
- start, rows:** `0` (start) and `10` (rows)
- fl:** (empty text box)
- df:** (empty text box)
- Raw Query Parameters:** `key1=val1&key2=val2`
- wt:** `-----` (dropdown menu)
- ☐ indent off
- ☐ debugQuery
- ☐ dismax
- ☐ edismax
- ☐ hl
- ☐ facet
- ☐ spatial
- ☐ spellcheck
- Execute Query** (blue button)

• 6.2 Retrieving data

To retrieve data from the selected core **"techproducts"**, you need to pass what you are searching for in query field. For example, if you want to retrieve the record with the value **"Samsung"**, you need to pass the value in the parameter **q** and execute the query.

```

http://localhost:6464/solr/techproducts/select?q=samsung

{
  "responseHeader": {
    "status": 0,
    "QTime": 34,
    "params": {
      "q": "samsung",
      "": "1544977470855"
    }
  },
  "response": {
    "numFound": 1, "start": 0, "docs": [
      {
        "id": "SP2514N",
        "name": "Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
        "manu": "Samsung Electronics Co. Ltd.",
        "manu_id_s": "samsung",
        "cat": [
          "electronics",
          "hard drive"
        ],
        "features": [
          "7200RPM, 8MB cache, IDE Ultra ATA-133",
          "NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor"
        ],
        "price": 92.0,
        "price_c": "92.0,USD",
        "popularity": 6,
        "inStock": true,
        "manufacturedate_dt": "2006-02-13T15:26:37Z",
        "store": "35.0752,-97.032",
        "_version_": 1616763502195638272,
        "price_c___l_ns": 9200
      }
    ]
  }
}

```

Direct link to retrieve the same data =>

<http://localhost:6464/solr/techproducts/select?q=samsung>

In this case the output will be given in Json

```

localhost:6464/solr/techproducts/select?q=samsung

{
  "responseHeader": {
    "status": 0,
    "QTime": 4,
    "params": {
      "q": "samsung"
    }
  },
  "response": {
    "numFound": 1, "start": 0, "docs": [
      {
        "id": "SP2514N",
        "name": "Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
        "manu": "Samsung Electronics Co. Ltd.",
        "manu_id_s": "samsung",
        "cat": [
          "electronics",
          "hard drive"
        ],
        "features": [
          "7200RPM, 8MB cache, IDE Ultra ATA-133",
          "NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor"
        ],
        "price": 92.0,
        "price_c": "92.0,USD",
        "popularity": 6,
        "inStock": true,
        "manufacturedate_dt": "2006-02-13T15:26:37Z",
        "store": "35.0752,-97.032",
        "_version_": 1616763502195638272,
        "price_c___l_ns": 9200
      }
    ]
  }
}

```


Using Solarium PHP API

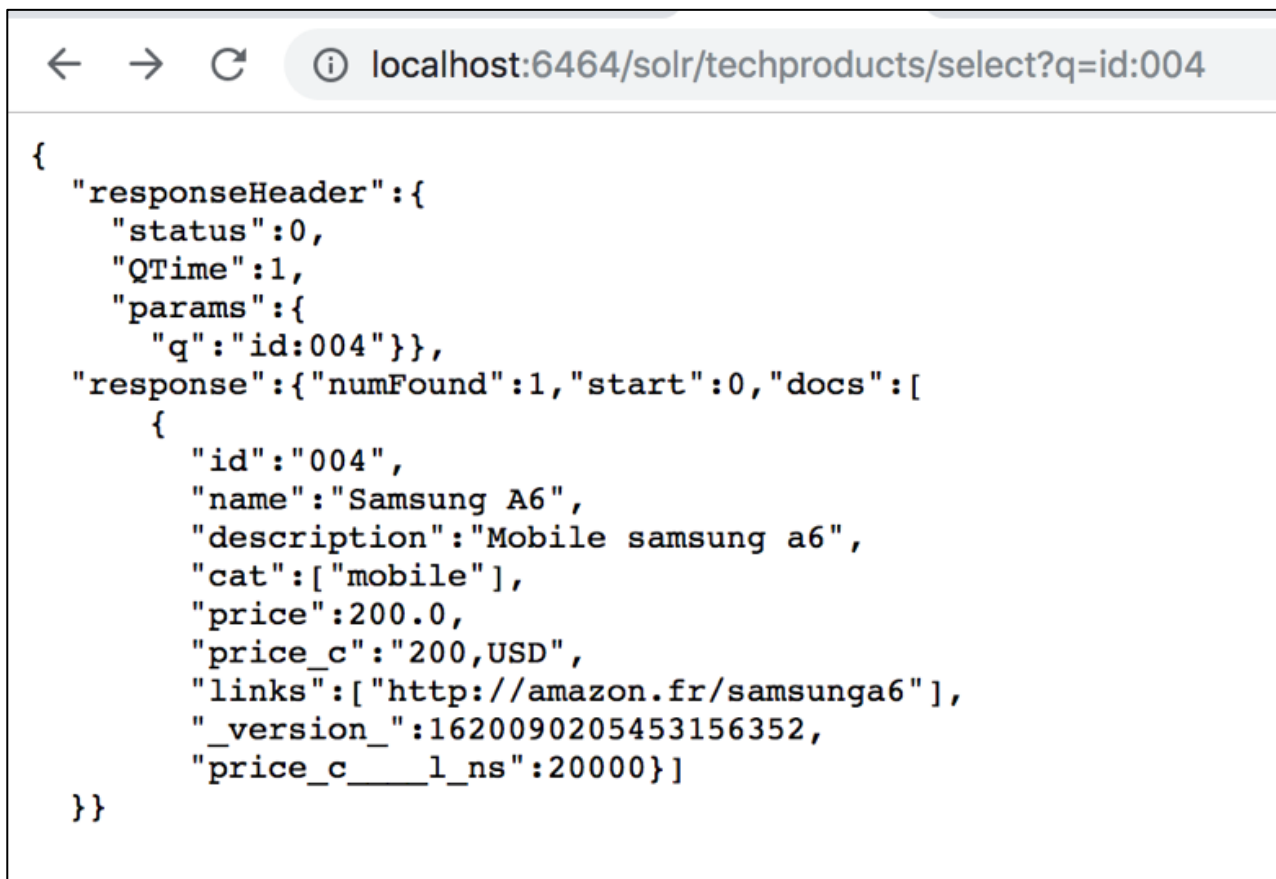
```

1 <?php
2     include "vendor/autoload.php";
3     include "config.php";
4     $client = new Solarium\Client($config);
5
6     $input = $_GET["query"];
7
8     $query = $client->createSelect();
9
10    $query->setQuery($input);
11
12    $result = $client->select($query);
13
14    foreach ($result as $doc) {
15        echo $doc->id."\n";
16        echo $doc->name."\n";
17        echo $doc->price."\n";
18        echo $doc->inStock."\n";
19    }
20
21
22

```

Let suppose that we are searching for data in some defined field, then we have to pass the field in the parameters and the value of what we are searching for **"?q=field:value"**

<http://localhost:6464/solr/techproducts/select?q=id:004>



The screenshot shows a web browser window with the address bar displaying `localhost:6464/solr/techproducts/select?q=id:004`. The main content area shows a JSON response from the Solr API. The response indicates that one document was found, with the following details:

- id:** "004"
- name:** "Samsung A6"
- description:** "Mobile samsung a6"
- cat:** ["mobile"]
- price:** 200.0
- price_c:** "200,USD"
- links:** ["http://amazon.fr/samsunga6"]
- _version_:** 1620090205453156352
- price_c____l_ns:** 20000

• 6.3 The Standard Query Parser Response

By default, the output of query is in JSON, but what if we need to have this result in another method like xml, csv or python that's why we have to use the **response writer**.

A Response Writer generates the formatted response of a search. Solr supports a variety of Response Writers to ensure that query responses can be parsed by the appropriate language or application.

This is all the possible parameters of the response writer

Csv, geojson, javabin, json, php, phps, python, ruby, smile, velocity, xlsx, xml, xslt

How we can do it ?

The same structure of search link but we have to add "wt" as parameter also

<http://localhost:6464/solr/techproducts/select?q=samsung&wt=xml>

Wt=xml and wt=php

```

<!-- This XML file does not appear to have any style information associated with it. The document tree is below -->
<?xml version="1.0"?>
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
    <lst name="params">
      <str name="q">samsung</str>
      <str name="wt">xml</str>
    </lst>
  </lst>
  <result name="response" numFound="3" start="0">
    <doc>
      <str name="id">004</str>
      <str name="name">Samsung A6</str>
      <str name="description">Mobile samsung a6</str>
      <arr name="cat">
        <str>mobile</str>
      </arr>
      <float name="price">200.0</float>
      <str name="price_c">200,USD</str>
      <arr name="links">
        <str>http://amazon.fr/samsunga6</str>
      </arr>
      <long name="version">1620090205453156352</long>
      <long name="price_c___l_ns">20000</long>
    </doc>
    <doc>
      <str name="id">1</str>
      <str name="name">Samsung A6</str>
      <float name="price">200.0</float>
      <str name="price_c">200,USD</str>
      <long name="version">1620090042054606848</long>
      <long name="price_c___l_ns">20000</long>
    </doc>
    <doc>
      <str name="id">SP2514N</str>
      <str name="name">
        Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133
      </str>
      <str name="manu">Samsung Electronics Co. Ltd.</str>
      <str name="manu_id_s">samsung</str>
      <arr name="cat">
        <str>electronics</str>
        <str>hard drive</str>
      </arr>
      <arr name="features">
        <str>7200RPM, 8MB cache, IDE Ultra ATA-133</str>
        <str>
          NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor
        </str>
      </arr>
      <float name="price">92.0</float>
      <str name="price_c">92.0,USD</str>
      <int name="popularity">6</int>
      <bool name="inStock">true</bool>
      <date name="manufacturedate_dt">2006-02-13T15:26:37Z</date>
      <str name="store">35.0752,-97.032</str>
      <long name="version">1616763502195638272</long>
      <long name="price_c___l_ns">9200</long>
    </doc>
  </result>
</response>

```

```

array(
  'responseHeader'=>array(
    'status'=>0,
    'QTime'=>0,
    'params'=>array(
      'q'=>'samsung',
      'wt'=>'php'),
    'response'=>array('numFound'=>3,'start'=>0,'docs'=>array(
      array(
        'id'=>'004',
        'name'=>'Samsung A6',
        'description'=>'Mobile samsung a6',
        'cat'=>array('mobile'),
        'price'=>200.0,
        'price_c'=>'200,USD',
        'links'=>array('http://amazon.fr/samsunga6'),
        'version'=>1620090205453156352,
        'price_c___l_ns'=>20000),
      array(
        'id'=>'1',
        'name'=>'Samsung A6',
        'price'=>200.0,
        'price_c'=>'200,USD',
        'version'=>1620090042054606848,
        'price_c___l_ns'=>20000),
      array(
        'id'=>'SP2514N',
        'name'=>'Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133',
        'manu'=>'Samsung Electronics Co. Ltd.',
        'manu_id_s'=>'samsung',
        'cat'=>array('electronics',
          'hard drive'),
        'features'=>array('7200RPM, 8MB cache, IDE Ultra ATA-133',
          'NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor'),
        'price'=>92.0,
        'price_c'=>'92.0,USD',
        'popularity'=>6,
        'inStock'=>true,
        'manufacturedate_dt'=>'2006-02-13T15:26:37Z',
        'store'=>'35.0752,-97.032',
        'version'=>1616763502195638272,
        'price_c___l_ns'=>9200)
      )
    )
  )
)

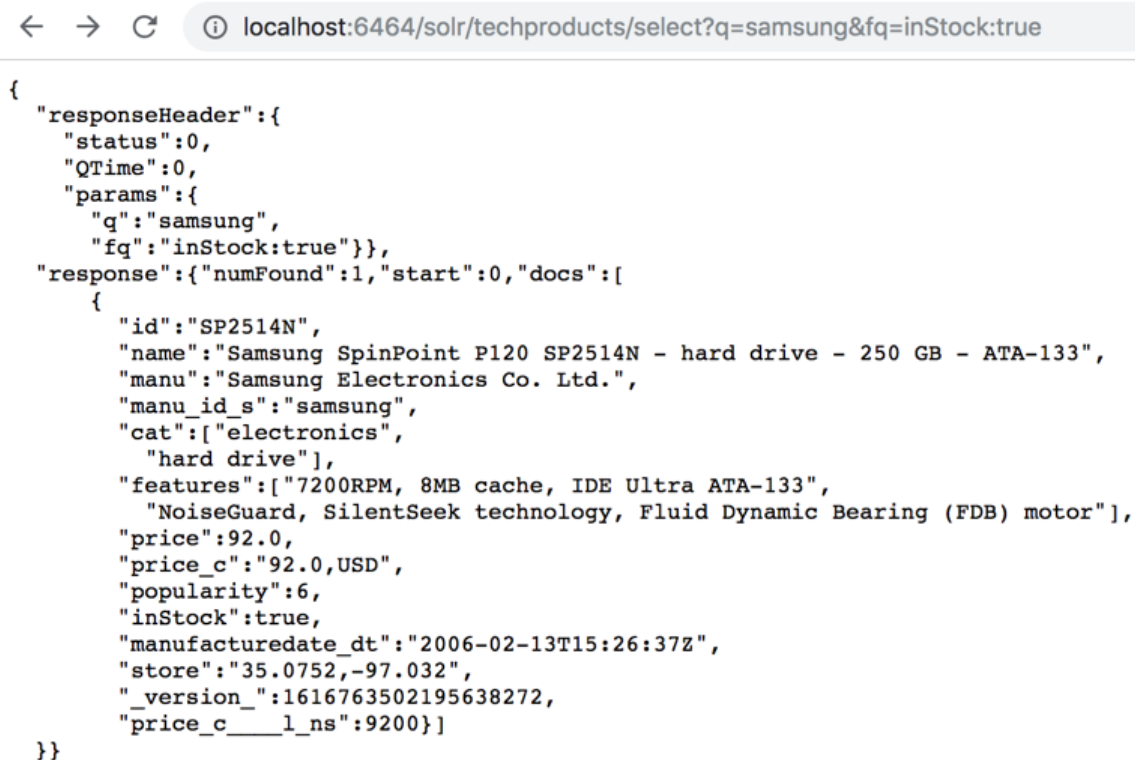
```

• 6.4 Filter Query

The **fq** parameter defines a query that can be used to restrict the superset of documents that can be returned, without influencing score. It can be very useful for speeding up complex queries, since the queries specified with **fq** are cached independently of the main query. When a later query uses the same filter, there's a cache hit, and filter results are returned quickly from the cache.

The **fq** parameter can be specified multiple times in a query. Documents will only be included in the result if they are in the intersection of the document sets resulting from each instance of the parameter. In the example below, only documents which have a popularity greater than 10 and have a section of 0 will match.

<http://localhost:6464/solr/techproducts/select?q=samsung&fq=inStock:true>




```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "q":"samsung",
      "fq":"inStock:true"}},
  "response":{"numFound":1,"start":0,"docs":[
    {
      "id":"SP2514N",
      "name":"Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
      "manu":"Samsung Electronics Co. Ltd.",
      "manu_id_s":"samsung",
      "cat":["electronics",
        "hard drive"],
      "features":["7200RPM, 8MB cache, IDE Ultra ATA-133",
        "NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor"],
      "price":92.0,
      "price_c":"92.0,USD",
      "popularity":6,
      "inStock":true,
      "manufacturedate_dt":"2006-02-13T15:26:37Z",
      "store":"35.0752,-97.032",
      "_version_":1616763502195638272,
      "price_c___l_ns":9200}]
  ]}
}
```

- **6.5 Field List**

The **fl** parameter limits the information included in a query response to a specified list of fields.

```
http://localhost:6464/solr/techproducts/select?q=samsung&fl=id,name,price
```



```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "samsung",
      "fl": "id,name,price"
    }
  },
  "response": {
    "numFound": 3,
    "start": 0,
    "docs": [
      {
        "id": "004",
        "name": "Samsung A6",
        "price": 200.0
      },
      {
        "id": "1",
        "name": "Samsung A6",
        "price": 200.0
      },
      {
        "id": "SP2514N",
        "name": "Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
        "price": 92.0
      }
    ]
  }
}
```

In this example, we can see that only the selected fields are shown

- **7 – Conclusion**

Apache Solr is proven in terms of an agile search engine. It has all the core features requisite to build a great E-commerce search platform. Solr's scalability, its cloud features and flexibility enable the online retailers to easily adapt to meet the higher demands of traffic, and Lucene's search libraries ability can be leveraged to meet any retailer's requirements. A significant increase in the adoption of Solr and Lucene in search platforms is increasing interest in deploying the technology for ECommerce Search. Also, Open source has many pragmatic effects, and Solr is no exception. It's free/very low cost and source plugins can be created as per business requirements. Apache Solr also has larger community of developers and committers. It is continually evolving in real time as developers add to it and modify it to make it a superior quality search platform.