

# Implementing Search Using Open-Source Tools

## TABLE OF CONTENTS

Background . . . . .	1
Architecture . . . . .	1
First Steps . . . . .	1
Configuration . . . . .	2
Schema Definition . . . . .	2
Defining Data Sources . . . . .	2
Importing Data . . . . .	3
Administration Interface . . . . .	3
Search Examples . . . . .	3
Integration Examples . . . . .	4
Application Express . . . . .	4
PL/SQL Example . . . . .	5
Java Example . . . . .	5
Conclusion . . . . .	6
References . . . . .	6

## BACKGROUND

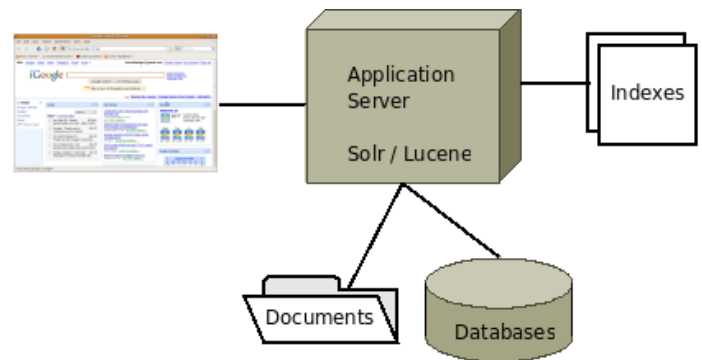
A common requirement for many Web applications is the ability to search for information that may come from a variety of sources including documents, databases, and web content. Search results are expected to be quickly displayed with sophisticated analysis including relevance ranking, prioritization based on business rules, spelling suggestions, and more like this functionality. Application users have grown accustomed to this kind of functionality and often expect this from internally developed applications.

This document describes how to implement search functionality using the Apache Lucene<sup>1</sup> and Apache Solr<sup>2</sup> open-source toolkits and gives examples of retrieving data from an Oracle 10g database and displaying it in Oracle Application Express.

Lucene and Solr are used by large web sites including: Netflix, CNET Reviews, Shopper.Com, Yankee Group, Homeland Security Digital Library, and many others<sup>3</sup>.

## ARCHITECTURE

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. Apache Solr is a layer on top of the Lucene platform and provides a higher-level method for working with Lucene. Solr extends the Lucene platform with XML, HTTP, and JSON APIs. Solr also provides caching, replication, a web administration interface and many more features. A simplified architecture diagram is shown below.



The ability to interact with the search server using XML or HTTP makes it easy to integrate this capability into a service-oriented architecture (SOA) or Oracle development environments like Oracle Application Express, Oracle WebCenter or PL/SQL.

Solr includes client wrappers for a number of program languages including: Java, C#, PHP, Ruby, and Python.

## FIRST STEPS

Before you begin your implementation you should consider the following questions.

- ‡ How will you associate documents with a particular product, product line, or SKU number?
- ‡ How will you join the information together? For example, say you want to join employee resumes that reside in separate PDF documents with information about employees from your database. If the PDF document names match the employee name you can use the document title to link to the employee information in the database.

display purposes?

you can re-index only those changes?

world or layer it behind a firewall and control access to it by applications that reside outside the firewall? Solr has a variety of deployment options that can fit most enterprise security requirements.

technology?

Once you are ready to begin, you can download the software from <http://lucene.apache.org/solr/> and uncompress it.

Solr uses the Apache Ant build process to create example code and scripts, deployment files, and unit tests. Follow the installation instructions at <http://wiki.apache.org/solr/SolrInstall>.

Solr uses the Apache Ant build process to create example code and scripts, deployment files, and unit tests. Follow the installation instructions at <http://wiki.apache.org/solr/SolrInstall>.

## CONFIGURATION

### Schema Definition

The first configuration step is to define what fields your documents or data will contain. This is directly analogous to schema definition in a traditional database environment. Schemas can be very simple (for example, document name and title) or have more complex hierarchical relationships. Schemas are defined in the schema.xml file and list the field names, data types, index, compression and storage options. For our example, we'll build a schema that provides information about employees.

```
<field name="id"
  type="integer"
  required="true" />
<field name="name"
  type="textSpell"
  required="true" />
<field name="email"
  type="string"
  required="false" />
<field name="city"
  type="textSpell"
  required="false" />
<field name="state"
  type="textSpell"
  required="false" />
<field name="resume"
  type="text"
  required="false" />
```

In this schema, we're going to provide information about employee's names, email, city, state, and their resume.

### Defining Data Sources

Data can come from a number of sources including: databases, documents in Office or PDF formats, XML files, CSV files, RSS feeds, and web sites.

In our example, we'll pull employee information from the HR example schema that is included in an Oracle database. The data source is defined in the data-config.xml file and relevant portions are shown below.

```
<dataConfig>
<dataSource type="JdbcDataSource" driver="oracle.
jdbc.OracleDriver" url="jdbc:oracle:thin:@
host:port:xe"
user="hr" password="hr"/>

<document name="employee">
  <entity name="item" query="select e.employee_
e.email, l.city, l.state_province
from employees e ...">

<field column="EMPLOYEE_ID"
  name="id" />

<field column="NAME"
  name="name" />

<field column="EMAIL"
  name="email" />

<field column="CITY"
  name="city" />

<field column="STATE_PROVINCE" name="state" />

  </entity>
</document>
</dataConfig>
```

The <datasource> tag defines the database driver and login information. Queries used to retrieve information are defined in the <entity> tag. Finally, the mapping from the SQL column name to the search schema name are listed in the <field> tag.

Employee resumes are stored in PDF files and the data source definitions are defined when the content is loaded.

## Importing Data

You can add data using command line programs or by sending an HTTP request to Solr. For example, a full import can be started by executing

```
http://<host>:<port>/solr/
dataimport?command=full-import
```

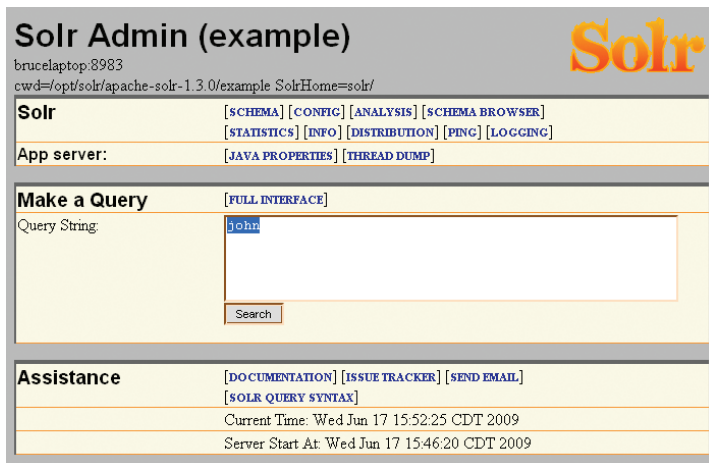
in your browser. Similarly, the URL

```
http://<host>:<port>/solr/
dataimport?command=status
```

provides detailed information about the number of documents loaded.

## ADMINISTRATION INTERFACE

Most administration tasks are performed using Solr's web interface.



It provides links to:

- ‡ Dg` fWef cgWdW
- ‡ 6 [eb`Sk eUZW\_ S S` V Ua` X`YgdcSf[a` VVW` [f[a` e-
- ‡ EZai UgddW f efSf[ef[Ue-
- ‡ 3` S`ki WWSfS V[efdTgf[a` - S` V
- ‡ B[ Y fZWewhWdadUZWU] a` Sbb [USf[a` eWhWdZWS`fZz

## SEARCH EXAMPLES

Searches can be performed using HTTP requests and passing the desired parameters. The following request will search for any employees with the name of John .

```
http://<host>:<port>/solr/select/?q=john
```

Results can be displayed in a variety of formats. XML results for this query are:

```
<result name="response" numFound="1" start="0">
<doc>
<str name="city">Seattle</str>
<str name="email">JCHEN</str>
<int name="id">110</int>
<str name="name">John Chen</str>
<str name="state">Washington</str>
</doc>
</result>
```

You can also specify certain fields to search. This example requests all employees in the State of Washington.

```
http://<host>:<port>/solr/
select/?q=state:washington
```

The results for this query are:

```
<result name="response" numFound="15" start="0">
<doc>
<str name="city">Seattle</str>
<str name="email">SKING</str>
<int name="id">100</int>
<str name="name">Steven King</str>
<str name="state">Washington</str>
</doc>
<doc>
<str name="city">Seattle</str>
<str name="email">NKOCHHAR</str>
<int name="id">101</int>
<str name="name">Neena Kochhar</str>
<str name="state">Washington</str>
</doc>
...
```

Common parameters that you can control include:

- ‡ EV\$cdZ fWd\_e
- ‡ Ead[ Y
- ‡ EfSdf[ Y dai S` V fZW` g\_ TWdaXdai efa [ b`W\_ W f pagination
- ‡ >[ef aXXWWe fa dWfgd
- ‡ EV\$cdZ XSUMfe
- ‡ EbW^UZWU] a` !aX
- ‡ : [YZ [YZf[ Y bScS\_ WVVb
- ‡ ? adW>[] WFZ[e bdaWef[ Y

You can also specify the style and formatting of the results display and apply custom-developed style sheets.

## INTEGRATION EXAMPLES

### Application Express

Once you become familiar with running queries using the Administration interface or using HTTP requests, let's explore adding a search capability into an Application Express page. Here's what the finished page should look like:

[Logout](#)

The screenshot shows a search page with a 'Search Term' input field and a 'Go' button. Below the search bar is a table with columns: City, State, Email, Employee Id, and Name. The table contains six rows of employee data. At the bottom of the page, there is a breadcrumb trail: Home | Application 104 | Edit Page 2 | Create | Session | Activity | Debug | Show Edit Links.

City	State	Email	Employee Id	Name
Seattle	Washington	JCHEN	110	John Chen
Seattle	Washington	JMURMAN	112	Jose Manuel Urman
South San Francisco	California	JFLEAUR	181	Jean Fleaur
South San Francisco	California	JDELLING	186	Julia Dellinger
South San Francisco	California	JDILLY	189	Jennifer Dilly
South San Francisco	California	VJONES	195	Vance Jones

This query uses the built-in Oracle XML processing functions to

1. Extract the SEARCH\_TERMS fields.

2. Convert the XML fragment into a row in a table.

3. Display the results as a table.

4. Format the output as a table.

Here's an example of what the report definition should look like.

The screenshot shows the 'Source' tab of the report definition editor. The 'Region Source' is defined as a SQL query that uses the EXTRACT function to parse XML data from the REST\_RESPONSE field. The query selects city, state, email, employee\_id, and name from the XML data.

```

SELECT
  extractValue(value(rest_response), '/doc/str[@name="city"]') city,
  extractValue(value(rest_response), '/doc/str[@name="state"]') state,
  extractValue(value(rest_response), '/doc/str[@name="email"]') email,
  extractValue(value(rest_response), '/doc/int[@name="id"]') employee_id,
  extractValue(value(rest_response), '/doc/str[@name="name"]') name
FROM table ( XMLSEQUENCE
  ( EXTRACT
    ( HTTPURITYPE('http://localhost:8963/solr/select/?q=' || :P2_SEARCH_TERM
  ).getXML(), '/response/result/doc') )
  ) REST_RESPONSE
    
```

To begin, create a new blank Application Express page. Then create a blank HTML region on the page. Then, create some page items. We will need:

1. A page item field will be named SEARCH\_TERMS

2. A page item that will redirect back to this same page when it is pushed.

Here's what the general structure of the page looks like.

Finally, create a Report that will appear on the HTML region. This report will be based on a SQL Query. In the report's Region Source area, enter this query:

SELECT

```

SELECT
  extractValue(value(rest_response), '/doc/str[@name="city"]') city,
  extractValue(value(rest_response), '/doc/str[@name="state"]') state,
  extractValue(value(rest_response), '/doc/str[@name="email"]') email,
  extractValue(value(rest_response), '/doc/int[@name="id"]') employee_id,
  extractValue(value(rest_response), '/doc/str[@name="name"]') name
FROM table ( XMLSEQUENCE
  ( EXTRACT
    ( HTTPURITYPE('http://localhost:8963/solr/select/?q=' || :P2_SEARCH_TERM
  ).getXML(), '/response/result/doc') )
  ) REST_RESPONSE
    
```

Due to the query's complexity, you may want to experiment running it in a SQL development environment like Oracle SQL Developer as you modify it to fit your requirements.

When you are all done, run the Application Express page. When it first appears the Search Term field will be blank and the Results report will show no data found. Then enter a search word and press the Search button. The screen should display your results.

[Logout](#)

The screenshot shows the same APEX page as before, but with the search term 'John Chen' entered in the search bar. The table now displays only the first two rows of data, which correspond to John Chen and Jose Manuel Urman. The breadcrumb trail remains the same.

## PL/SQL Example

Let's show another example of using Solr's JSON interface to execute a search in PL/SQL and parse the results.

To run this example, you will need to download and install an open-source PL/SQL JSON parsing package from <http://reseau.erasme.org/Librairie-JSON>.

Here's what the finished code looks like:

```
declare
  v_output  clob;
  v_url     varchar2(2000);
  v_obj     json.JSONStructObj;
  v_array   json.JSONArray;
  i         integer;
begin
  -- Construct the URL
  {Ž|ā→ĀĪKĀā\*ĪĐĐ→~`á→ā~b\ĪĪĪĪĪĪĐĐb~→āĐ
  bā→æ`ĪĐŁ@KĪ~ā^ōĀ•ĀĐBōĀ•ĀĐ}ĪKĪb~^ōĪ
  -- Send the request
  -- and get the contents
  v_output := HTTPURITYPE(v_url).getClob();
  -- Convert the output to a JSON structure
  {Ž~āĪĪKĪāĪb~^ĒU\ā↔^&GŌUŠSÇ{Ž~|\*|\ĒĀĐ°ōDĪ
  -- Get an array with all the response name/value
  pairs
  {ŽāāāāĪĪKĪāĪb~^Ē&æ\N\āNāāā]ÇĀ{Ž~āĪĒĀāāāb*~^bāōDĪ
  -- Print out each name/value pair
  i := v_array.first;
  while (i is not null) loop
    ĀĀāāāābŽ~|\*|\Ē*|\ÇøSāāāā↔bĪĀĐĀ•Ā{Žāāāāā]Ç↔DDĪ
    i := v_array.next(i);
    ĀĀāāāābŽ~|\*|\Ē*|\Ž~↔^æÇøĒĀ{ā→|āā↔bĪĀĐĀ•Ā{Ž
    array(i);
    i := v_array.next(i);
  end loop;
end;
```

Just like in the Application Express example, we will first create a string that has the URL for executing the search.

Note that the parameter &wt=json is used to tell the Solr engine to return the results in a JSON formatted string.

Next we'll use the built-in HTTPURITYPE object to execute the URL and return the results as a CLOB. The results are passed into the JSON package which returns a structured object.

The line `v_array := json.getAttrArray(v_obj, 'name/value')`

asks the JSON package to parse the response object and to return the results in an array of name/value pairs.

Finally, we loop through all the rows in the array and print out the information.

Additional details about using the JSON format can be found on the Solr Wiki at

<http://wiki.apache.org/solr/SolrJSON>.

## Java Example

A code fragment to search for the text "John" with spell check enabled is shown below.

```
SolrServer solr = new
  CommonsHttpSolrServer(
    "http://localhost:8983/solr"
  );
ModifiableSolrParams params =
  new ModifiableSolrParams();

params.set("qt", "/spellCheckCompRH");

params.set("q", "john");

params.set("spellcheck", "on");

QueryResponse response =
  solr.query(params);

System.out.println("response = "
  + response);
```

The first line creates a new instance of a Solr server. Next a Solr parameters object is created and various parameters are set. The line `solr.query(params)` executes the query and returns the response. Finally the response is displayed.

A complete Java example program can be found on the Solr web site. Note that the Solr Java client includes an option to access an embedded Solr instance without the need to run an HTTP or Application server.

