

Introduction to Apache Solr & example usage.

Rapport done by Student: ***Oussema Hidri.***

Table of contents:

Introduction

- I. Configuring and Working with Solr
 - 1. Simple Configuration
 - 2. Working with CSV data format
 - 3. Working with XML data format
- II. Indexing text
- III. Usage Example with CSV dataset

Introduction:



Apache Solr is an open source search platform built upon a Java library called Lucene. **Solr** is a popular search platform for Web sites because it can index and search multiple sites and return recommendations for related content based on the search queries' taxonomy.

This rapport was drawn from my blog: <https://dmidma.wordpress.com/>.
So if this rapport seems a little bit off, you can check the blog for more details.

I. Configuring and Working with Solr:

I will go through some details of how to index your data from scratch using Apache Solr. Although this is not my own example, it is inspired by Alexandre Rafalovitch example, this is how you would do things in Solr 6.0 (not Solr 4.x). You can find the previous example in this [link](#).

1. Simple Configuration:

Start by downloading a suitable version for your OS from the solr official (website)[<http://lucene.apache.org/solr/>].

I will be running this example on Ubuntu.

To create your own collection you will need two main files:

- managed-schema
- solrconfig.xml

To do that, you can run this simple command:

```
$ mkdir -p collection1/conf && cd collection1/conf && touch  
managed-schema solrconfig.xml
```

The **managed-schema** file should have:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<schema version="1.5">  
  <fields>  
    <field name="id" type="string" indexed="true" stored="true"  
      required="true"/>  
    <field name="addr_from" type="string" indexed="true" stored="true"  
      required="true"/>  
    <field name="addr_to" type="string" indexed="true" stored="true"  
      required="true"/>  
    <field name="subject" type="string" indexed="true" stored="true"  
      required="true"/>  
  </fields>  
</schema>
```

```
</fields>
<uniqueKey>id</uniqueKey>
<types>
<fieldType name="string" class="solr.StrField" />
</types>
</schema>
```

This file defines the schema of your data that you will store and index.

Each field defines its type, whether it is stored, indexed, required, and other properties.

If you noticed, the type is also defined in the same file:

```
<fieldType name="string" class="solr.StrField" />
```

The solrconfig.xml file should have:

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
<luceMatchVersion>6.4.2</luceMatchVersion>
<requestDispatcher handleSelect="false">
<httpCaching never304="true" />
</requestDispatcher>
<requestHandler name="/select" class="solr.SearchHandler" />
<requestHandler name="/update" class="solr.UpdateRequestHandler" />
<requestHandler name="/analysis/field"
class="solr.FieldAnalysisRequestHandler" startup="lazy" />
</config>
```

This file defines and tunes the components that make up Solr's runtime environment and which URLs can be called to:

- *add record -> /update*
- *query a collection -> /select*
- *specify multiple field types/names -> /analysis/field*

At this point you only have two files which will be used to configure your collection/core. Go ahead and start solr (you should be at the root of the downloaded directory, you need to unzip it of course):

```
$ bin/solr start
```

You should see something like this:

```
Archiving 1 old GC log files to /home/dmidma/Downloads/solr-6.4.2/server/logs/archived
Archiving 1 console log files to /home/dmidma/Downloads/solr-6.4.2/server/logs/archived
Rotating solr logs, keeping a max of 9 generations
Waiting up to 180 seconds to see Solr running on port 8983 [\]
Started Solr server on port 8983 (pid=9700). Happy searching!
```

Since this is the first time your run solr, you will have no collections/cores. Therefore, you need to create them manually.

Manually create the collection/core with the configuration files that you just created in the earlier step:

```
$ bin/solr create_core -c collection1 -d collection1/conf/
```

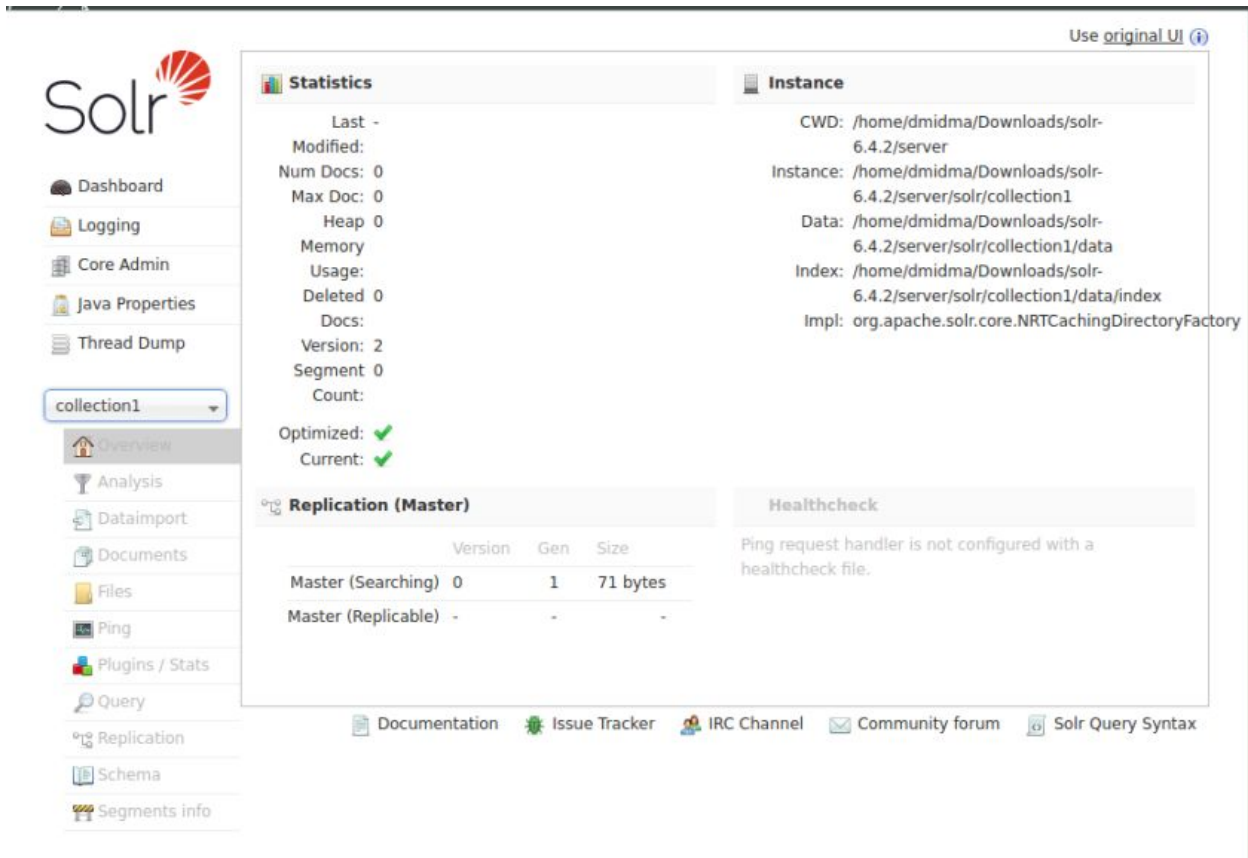
You should see something like this:

```
Copying configuration to new core instance directory:
/home/dmidma/Downloads/solr-6.4.2/server/solr/collection1

Creating new core 'collection1' using command:
http://localhost:8983/solr/admin/cores?action=CREATE&name=collection1&instanceDir=collection1

{
  "responseHeader":{
    "status":0,
    "QTime":611},
  "core":"collection1"}
```

You can check if every thing until now is working correctly by going to the Admin WebUI : <http://localhost:8983/solr/>
From the WebUI, select the core you have just created and you will get this:



The screenshot shows the Solr Admin WebUI interface for a core named 'collection1'. The left sidebar contains a navigation menu with options: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, Overview (selected), Analysis, Dataimport, Documents, Files, Ping, Plugins / Stats, Query, Replication, Schema, and Segments info. The main content area is divided into four panels: Statistics, Instance, Replication (Master), and Healthcheck. The Statistics panel shows various metrics like Num Docs, Max Doc, Heap, Memory, Usage, Deleted, Docs, Version, Segment, and Count, along with status indicators for Optimized and Current. The Instance panel displays configuration details such as CWD, Instance, Data, Index, and Impl. The Replication (Master) panel shows a table with columns for Version, Gen, and Size, listing Master (Searching) and Master (Replicable) nodes. The Healthcheck panel indicates that the ping request handler is not configured with a healthcheck file. At the bottom, there are links to Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

Solr

Use [original UI](#) ⓘ

Statistics

Last -
Modified:
Num Docs: 0
Max Doc: 0
Heap 0
Memory
Usage:
Deleted 0
Docs:
Version: 2
Segment 0
Count:
Optimized: ✓
Current: ✓

Instance

CWD: /home/dmidma/Downloads/solr-6.4.2/server
Instance: /home/dmidma/Downloads/solr-6.4.2/server/solr/collection1
Data: /home/dmidma/Downloads/solr-6.4.2/server/solr/collection1/data
Index: /home/dmidma/Downloads/solr-6.4.2/server/solr/collection1/data/index
Impl: org.apache.solr.core.NRTCachingDirectoryFactory

Replication (Master)

	Version	Gen	Size
Master (Searching)	0	1	71 bytes
Master (Replicable)	-	-	-

Healthcheck

Ping request handler is not configured with a healthcheck file.

[Documentation](#) [Issue Tracker](#) [IRC Channel](#) [Community forum](#) [Solr Query Syntax](#)

2. Working with CSV data format:

Congrats, you have created your first collection in solr!

But it does not have any data yet, go to **collection1** directory and edit **input1.csv** file and put this sample data in it:

```
id,addr_from,addr_to,subject
email1,fulan@acme.example.com,kari@acme.example.com,"Kari, we need more
Junior Java engineers"
email2,kari@acme.example.com,maiya@acme.example.com,"Updating vacancy
description"
```

Now run this **post** tool to add your data to your created core:

```
$ bin/post -c collection1 collection1/input1.csv
```

You will see something like this:

```
dmdma@dmdma-Lenovo:~/Downloads/solr-6.4.2$ bin/post -c collection1 collection1/input1.csv
/usr/lib/jvm/jdk1.8.0_71/bin/java -classpath /home/dmdma/Downloads/solr-6.4.2/dist/solr-core-6.4.2.jar -
Dauto=yes -Dc=collection1 -Ddata=files org.apache.solr.util.SimplePostTool collection1/input1.csv
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/collection1/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 input1.csv (text/csv) to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/collection1/update...
Time spent: 0:00:00.259
```

Now you can visit this link to check your indexed data:

<http://localhost:8983/solr/collection1/select?q=%3A&wt=ruby&indent=true>

Or you can query your data through CLI:

I am using HTTPie instead of CURL.

```
$ http -b :8983/solr/collection1/select q==*: indent=true wt==ruby
```

```
{
  'responseHeader'=>{
    'status'=>0,
    'QTime'=>21},
  'response'=>{'numFound'=>2,'start'=>0,'docs'=>[
    {
      'id'=>'email1',
      'addr_from'=>'fulan@acme.example.com',
      'addr_to'=>'kari@acme.example.com',
      'subject'=>'Kari, we need more Junior Java engineers'},
    {
      'id'=>'email2',
      'addr_from'=>'kari@acme.example.com',
      'addr_to'=>'maija@acme.example.com',
      'subject'=>'Updating vacancy description'}}
  ]}
}
```

Or

```
$ http -b :8983/solr/collection1/select q==*:~ indent==true wt==json
```

```
{
  "response": {
    "docs": [
      {
        "addr_from": "fulan@acme.example.com",
        "addr_to": "kari@acme.example.com",
        "id": "email1",
        "subject": "Kari, we need more Junior Java engineers"
      },
      {
        "addr_from": "kari@acme.example.com",
        "addr_to": "maiya@acme.example.com",
        "id": "email2",
        "subject": "Updating vacancy description"
      }
    ],
    "numFound": 2,
    "start": 0
  },
  "responseHeader": {
    "QTime": 0,
    "status": 0
  }
}
```

- `q=*` -> returns all the documents.
- `wt=format` -> the format of the returned document (csv, json, ruby, xml, ...)
- If the format is not recognized by solr, it will output with the default format (xml).

You can run several collection at once which gives us an easier way to compare the content of different examples without having to start and stop different Solr instances.

Let's create another core with the same conf of the previous one:

```
$ bin/solr create_core -c multivalued -d collection1/conf/
```

Each of the created core will have a directory with its name under /server/solr directory and will contain core.properties files that takes care of the configuration of that core. For now it will only have:

- `name=core_name`

You can go to the Admin WebUi cores section:

<http://localhost:8983/solr/#/~cores/> to add core, Unload, and Reload.

Now we will make few changes related to the second core.

First, we are going to change the **schema** of the data that will be indexed in the second core. Therefore, edit the file **managed-schema** under:

–**/server/solr/multivalued/conf** and add the attribute `multivalued="true"` to the `add_to` field:

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

Now go back to the old **input1.csv** file and create a copy of it:

```
$ cp input1.csv input2.csv
```

Edit **input2.csv** and change the `addr_to` column value for the second data line (third file line) so it can have two email addresses (separate by a ;):

```
email2,kari@acme.example.com,maijs@acme.example.com;ivan@acme.example.com,"
Updating vacancy description"
```

Now we need to post the data as we did before. But before that we are going to restart our server, so the second core realize that we made a change to its **managed-schema** file:

```
$ bin/solr stop -all && bin/solr start

$ bin/post -c multivalued collection1/input2.csv
```

Now if you query the core of all docs, you will see that the `addr_to` is actually multi valued (`[]`) but we only have one value:

```
maijs@acme.example.com;ivan@acme.example.com
```

So, solr didn't separate the two addresses, and now all we need to do is re-post the data but with an additional flag:

```
$ bin/post -c multivalued collection1/input2.csv -params  
"f.addr_to.split=true&f.addr_to.separator=;"
```

What we have added is the `-params` flag and:

*`f.addr_to.split=true`
`f.addr_to.separator=;`*

The general syntax is this: `f.<fieldname>.<option>` and what we actually did, is instructed solr to separate the `addr_to` data value through the `;` separator.

So now if you query the indexed data:

```
$ http -b :8983/solr/multivalued/select q==*:~ indent==true wt==json
```

You should see something like this:

```
{
  "response": {
    "docs": [
      {
        "addr_from": "fulan@acme.example.com",
        "addr_to": [
          "kari@acme.example.com"
        ],
        "id": "email1",
        "subject": "Kari, we need more Junior Java engineers"
      },
      {
        "addr_from": "kari@acme.example.com",
        "addr_to": [
          "maiya@acme.example.com",
          "ivan@acme.example.com"
        ],
        "id": "email2",
        "subject": "Updating vacancy description"
      }
    ],
    "numFound": 2,
    "start": 0
  },
  "responseHeader": {
    "QTime": 0,
    "status": 0
  }
}
```

But didn't we post the data twice, shouldn't we have more than this?

In fact, the:

`<uniqueKey>id</uniqueKey>`

field allows to run the `bin/post` tool as many time as we want, and our data will not be **appended** but **updated** instead.

3. Working with XML Data format:

Since we have talked about the multi value field, let's do an example with **XML** format.

XML is more verbose than CSV, and it deals better with multi value fields, optional fields, complex text string, and many other requirements.

Start by closing the server:

```
$ bin/solr stop -all
```

What we need to do now, is to copy the *server/solr/multivalued* directory and delete the **server/solr/xml/data** directory:

```
$ cp -r server/solr/multivalued server/solr/xml  
$ rm -rf server/solr/xml/data
```

Now we have a new core named xml with the same config files as the multivalued core.

This is another way of creating a collection/core without the command.

Be sure to change `name=multivalued` to `name=xml` in the `core.properties` file!

Before starting the server, add this new three fields inside **fields** (**managed-schema** file) to the xml core:

```

<field name="date" type="string" indexed="true" stored="true"
required="true" />
<field name="message" type="string" indexed="true" stored="true"
required="true" />
<field name="priority" type="int" indexed="true" stored="true" />

```

If you noticed, we have a new type `int` for the `priority` field, therefore inside the `types` section add:

```

<fieldType name="int" class="solr.TrieIntField" precisionStep="0"
positionIncrementGap="0"/>

```

Solr does not have any hard-coded type names; you have to define which ones you want and what you want to name them.

Now create an xml file with this data in it; I will name my file `input3.xml`, you can name it whatever you want as long as stick to that name when posting the data:

```

<add>
  <doc>
    <field name="id">email1</field>
    <field name="addr_from">Fulan AlFulani
    &lt;fulan@acme.example.com&gt;</field>
    <field name="addr_to">Kari Nordmann &lt;kari@acme.example.com&gt;</field>
    <field name="subject">Kari, we need more Junior Java engineers</field>
    <field name="date">5 Jan 2012</field>
    <field name="message">Kari, some of our Java engineers left at the end of
last year. Can you please advertise some open positions at Junior level.
Thanks, Fulan</field>
  </doc>
  <doc>
    <field name="id">email2</field>
    <field name="addr_from"><![CDATA[Kari Nordmann
<kari@acme.example.com>]]></field>
    <field name="addr_to"><![CDATA[Maija Meikäläinen
<maija@acme.example.com>]]></field>
    <field name="addr_to"><![CDATA[Ivan Ivanovich Ivanov
<ivan@acme.example.com>]]></field>
    <field name="subject">Updating vacancy description</field>
    <field name="date">6 Jan 2012</field>
  </doc>
</add>

```



```
<field name="message">Maija, Vania I need you to work together to update  
the description for Junior Java programmers. Fulan wants to hire some more  
for both New York and San Francisco location. See if we can wrap this up in  
a week. Thank you, Kari</field>  
<field name="priority">2</field>  
</doc>  
</add>
```

This is a custom XML format designed for Solr needs.

Since we need to escape characters such as <, >, and &; the example dictates two way of dealing with it:

- *Escaping the individual characters.*
- *Enclosing the whole content in a CDATA section.*

We have indexed the date as a string, since solr expects the real dates to come in precise ISO 8601 format. You can read more about it in this [link](#).

Start the server and post the data in the new core:

```
$ bin/solr start  
$ bin/post -c xml collection1/input3.xml
```

Now check if your data have been added correctly to your `xml` core:

```
$ http -b :8983/solr/xml/select q==*:* indent==true wt==json
```

```
{
  "response": {
    "docs": [
      {
        "addr_from": "Fulan AlFulani <fulan@acme.example.com>",
        "addr_to": [
          "Kari Nordmann <kari@acme.example.com>"
        ],
        "date": "5 Jan 2012",
        "id": "email1",
        "message": "Kari, some of our Java engineers left at the end of last year. Can you please
advertise some open positions at Junior level. Thanks, Fulan",
        "subject": "Kari, we need more Junior Java engineers"
      },
      {
        "addr_from": "Kari Nordmann <kari@acme.example.com>",
        "addr_to": [
          "Maija Meikäläinen <maija@acme.example.com>",
          "Ivan Ivanovich Ivanov <ivan@acme.example.com>"
        ],
        "date": "6 Jan 2012",
        "id": "email2",
        "message": "Maija, Vania I need you to work together to update the description for Junior
Java programmers. Fulan wants to hire some more for both New York and San Francisco location. See if we
can wrap this up in a week. Thank you, Kari",
        "priority": 2,
        "subject": "Updating vacancy description"
      }
    ],
    "numFound": 2,
    "start": 0
  },
  "responseHeader": {
    "QTime": 2,
    "status": 0
  }
}
```

We can delete a document by executing an update query that takes an XML data:

```
$ bin/post -c xml -d "$"<delete><query>priority:2</query></delete>"
```

We can delete a document by:

- its ID: `<delete><id>552</id></delete>`

- a query: `<delete><query>priority:2</query></delete>`

It is also possible to have multiple id and query element within one delete statement.

The id element does not refer to the field named “id” but to the uniqueKey element defined in the managed-schema file.

II. Indexing Text:

The real strength of Lucene/Solr is in indexing the text.

For this section we are going to find documents based on the following scenarios:

- A: Find a message for Kari.
- B: Find all messages with the word Kari in any text field.
- C: Find all messages with the phrase “Java engineers” in any text field.

Let’s start by copying the xml collection directory, and this time without deleting its data sub-directory:

```
$ cp -r server/solr/xml server/solr/text1
```

As we did before, change the name in *core.properties* of the text1 collection to:

```
name=text1
```

Before restarting solr, modify the request handler in `server/solr/text1/conf/solrconfig.xml` to include:

```
<requestHandler name="/select" class="solr.SearchHandler" >
  <lst name="defaults">
    <str name="wt">json</str>
    <str name="indent">true</str>
  </lst>
</requestHandler>
```

The results will be automatically indented and in JSON format.

Let's start with scenario A:

The obvious way to do this is to run a query with `addr_to:kari`:

```
$ http -b :8983/solr/text1/select q==addr_to:kari
```

Yeah, no result. Try with an uppercase 'K' (`addr_to:Kari`):

```
$ http -b :8983/solr/text1/select q==addr_to:Kari
```

Again, no result. What about a wildcard (`addr_to:Kari*`):

```
$ http -b :8983/solr/text1/select q==addr_to:Kari*
```

Okey, one record. This means we can search with wildcards (`addr_to:*ari*`):

```
$ http -b :8983/solr/text1/select q==addr_to:*ari*
```

Again one record.

Now we need to extend our current configuration to realize the described scenarios. Edit `server/solr/text1/conf/managed-schema` and add this `fieldType` into the `types` section:

```
<fieldType name="email" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.UAX29URLEmailTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Then change the type for both `addr_from` and `addr_to` fields to `email`:

```
<field name="addr_from" type="email" indexed="true" stored="true"
required="true" />
<field name="addr_to" type="email" multiValued="true" indexed="true"
stored="true" required="true" />
```

Now, go into <http://localhost:8983/solr/#/~cores/> and reload the text1 core.

After that you will need to rerun the indexing process:

```
$ bin/post -c text1 collection1/input3.xml
```

Go ahead and test the query for the first scenario, no need to specify neither a wildcard or to check your capitalization (`addr_to:kari`):

```
$ http -b :8983/solr/text1/select q==addr_to:kari
```

You should get this:

```
{
  "response": {
    "docs": [
      {
        "addr_from": "Fulan AlFulani <fulan@acme.example.com>",
        "addr_to": [
          "Kari Nordmann <kari@acme.example.com>"
        ],
        "date": "5 Jan 2012",
        "id": "email1",
        "message": "Kari, some of our Java engineers left at the end of last year. Can you please
advertise some open positions at Junior level. Thanks, Fulan",
        "subject": "Kari, we need more Junior Java engineers"
      }
    ],
    "numFound": 1,
    "start": 0
  },
  "responseHeader": {
    "QTime": 1,
    "status": 0
  }
}
```

Let's move to the other scenarios. For that, we need to change or `solrconfig.xml` once again (be sure you change the config files under `/server/solr/text1/conf` and not other place):

```
<requestHandler name="/select" class="solr.SearchHandler" >
<lst name="defaults">
<str name="df">text</str>
...

```

What we added here is a default field name that will be used upon /select query.

The parameters are set in the defaults section, which mean that we can override them during a query by explicitly specifying the parameter name.

It is possible to define multiple requestHandlers with different configurations for different capabilities.

But hey, we don't have a `text` field. That's right, go ahead and add the this new instructions in the `managed-schema` file:

```
<field name="text" type="text_general" multiValued="true" indexed="true"
stored="false" />
<copyField source="message" dest="text" />
<copyField source="subject" dest="text" />
<copyField source="addr_*" dest="text" />
```

What we have done here, is instructed solr to copy message, subject, and addr_ (addr_to, addr_from) to text field.*

You can read more about copyField in this [link](#).

and add the new type `text_general` into the `types` section:

```
<fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100">
<analyzer>
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
</fieldType>
```

As we did before, we need to restart the text1 core and rerun the indexing process:

```
$ http -b :8983/solr/admin/cores action==RELOAD core==text1
$ bin/post -c text1 collection1/input3.xml
```

The moment of truth, for the scenario:

- B:

```
$ http -b :8983/solr/text1/select q==kari
```


This should get you too results:

```
{
  "response": {
    "docs": [
      {
        "addr_from": "Fulan AlFulani <fulan@acme.example.com>",
        "addr_to": [
          "Kari Nordmann <kari@acme.example.com>"
        ],
        "date": "5 Jan 2012",
        "id": "email1",
        "message": "Kari, some of our Java engineers left at the end of last year. Can you please
advertise some open positions at Junior level. Thanks, Fulan",
        "subject": "Kari, we need more Junior Java engineers"
      },
      {
        "addr_from": "Kari Nordmann <kari@acme.example.com>",
        "addr_to": [
          "Maija Meikäläinen <maija@acme.example.com>",
          "Ivan Ivanovich Ivanov <ivan@acme.example.com>"
        ],
        "date": "6 Jan 2012",
        "id": "email2",
        "message": "Maija, Vania I need you to work together to update the description for Junior
Java programmers. Fulan wants to hire some more for both New York and San Francisco location. See if we
can wrap this up in a week. Thank you, Kari",
        "priority": 2,
        "subject": "Updating vacancy description"
      }
    ],
    "numFound": 2,
    "start": 0
  },
  "responseHeader": {
    "QTime": 1,
    "status": 0
  }
}
```

- C:

```
$ http -b :8983/solr/text1/select q=="Java+engineers"
```

Which will get you one record:

```
{
  "response": {
    "docs": [
      {
        "addr_from": "Fulan AlFulani <fulan@acme.example.com>",
        "addr_to": [
          "Kari Nordmann <kari@acme.example.com>"
        ],
        "date": "5 Jan 2012",
        "id": "email1",
        "message": "Kari, some of our Java engineers left at the end of last year. Can you please
advertise some open positions at Junior level. Thanks, Fulan",
        "subject": "Kari, we need more Junior Java engineers"
      }
    ],
    "numFound": 1,
    "start": 0
  },
  "responseHeader": {
    "QTime": 0,
    "status": 0
  }
}
```

It is important to understand that Lucene breaks text into tokens and then stores those tokens in a specialized structure that allows us to very quickly look them up during query and find all the places where those tokens come from.

Faceting is counting values in a particular field based on the current search. It allows uses to do basic search first and then start narrowing it down based on categories and counts returned.

Solr has a very strong support for faceting.

Keep in mind that the facet values come from the indexed and not the stored values of the field.

What we want to do now is to:

- See which people participate in the email and how often.
- count either by name, or email alone.
- search for people's names both with and without accents.

- documents where people are senders to rank higher than those that are receivers.

We are going to create another core and name it **text2**, this will have the same configuration as the **text1** core, therefore:

```
$ bin/solr create_core -c text2 -d server/solr/text1/conf
```

Now for its data, copy the **input3.xml** into **input4.xml** and add another **doc** within the **add** section:

```
<doc>
  <field name="id">email3</field>
  <field name="addr_from"><![CDATA[Maija Meikäläinen
<maija@acme.example.com>]]></field>
  <field name="addr_to"><![CDATA[Kari Nordmann
<kari@acme.example.com>]]></field>
  <field name="addr_cc"><![CDATA[Ivan Ivanovich Ivanov
<vania@home.example.com>]]></field>
  <field name="subject">Re: Updating vacancy description</field>
  <field name="date">10 Jan 2012</field>
  <field name="message">Kari, please find the proposed description attached.
Vania is not feeling too well today. He will check his home email, if
something is urgent. Salut, Maija</field>
</doc>
```

Before posting this data, go into **managed-schema** and change the definition of **addr_to** field to make it dynamic:

```
<dynamicField name="addr_*" type="email" multiValued="true" indexed="true"
stored="true" />
```

Since we have replaced the field definition from static to a dynamic, the `addr_c` field will be identical in every way to `addr_to` but not to `addr_from` because we kept its definition separate (it is not multivalued).

and add this into the `fields` section:

```
<field name="facet_contacts" type="address_noemail" multiValued="true"
indexed="true" stored="false" />
<field name="facet_emails" type="address_email" multiValued="true"
indexed="true" stored="false" />
<copyField source="addr_*" dest="facet_contacts" />
<copyField source="addr_*" dest="facet_emails" />
```

And since you used new types, go ahead and declare them in the `types` section:

```
<fieldType name="address_noemail" class="solr.TextField">
  <analyzer type="index">
    <charFilter class="solr.PatternReplaceCharFilterFactory" pattern=" &lt;.*"
replacement="" />
    <tokenizer class="solr.KeywordTokenizerFactory" />
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.KeywordTokenizerFactory" />
  </analyzer>
</fieldType>
<fieldType name="address_email" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.UAX29URLEmailTokenizerFactory"/>
    <filter class="solr.TypeTokenFilterFactory" types="filter_email.txt"
useWhitelist="true"/>
  </analyzer>
</fieldType>
```

The `solr.UAX29URLEmailTokenizerFactory` splits the names and drops the punctuation but keeps the email address as a single token.

One last thing, create a new file filter_email.txt under the `server/solr/text2/conf` directory and populate it with:

```
<HEAD>
```

The tokenizer assigns different token types and we can use `solr.TypeTokenFilterFactory` to keep or get rid of particular types.

In our case, we whitelisted `<EMAIL>` as the only type we want to let through.

The same process as before, restart the core and index the input4.xml data into it:

```
$ http -b :8983/solr/admin/cores action==RELOAD core==text2
$ bin/post -c text2 collection1/input4.xml
```

Okey, let's run a query to get facet counts for `facet_contacts` and `facet_emails`:

```
$ http -b :8983/solr/text2/select q==*:~* facet==true
facet.field==facet_contacts facet.field==facet_emails
```

You should get the list of the 3 emails, but more importantly you should get this:

```
[
  "facet_counts": {
    "facet_fields": {
      "facet_contacts": [
        "Kari Nordmann",
        3,
        "Ivan Ivanovich Ivanov",
        2,
        "Maija Meikäläinen",
        2,
        "Fulan AlFulani",
        1
      ],
      "facet_emails": [
        "kari@acme.example.com",
        3,
        "maija@acme.example.com",
        2,
        "fulan@acme.example.com",
        1,
        "ivan@acme.example.com",
        1,
        "vania@home.example.com",
        1
      ]
    },
    "facet_heatmaps": {},
    "facet_intervals": {},
    "facet_queries": {},
    "facet_ranges": {}
  },
  "response": {
    "docs": [
```

Notice that the name counts, do not match the email counts, as *Ivan* has used two different e-mails.

Let's run another query, but this time for *Fulan*:

```
$ http -b :8983/solr/text2/select q==fulan facet==true
facet.field==facet_contacts facet.field==facet_emails
```

You should only get two emails, and the facet counts should now be lower so that it matches only the messages found.

Let's improve the ability to search for fields with a special boost for names found in the `addr_from` field. Therefore, add more default parameters to the select handler in `solrconfig.xml`:

```
<lst name="defaults">
<str name="defType">edismax</str>
<str name="qf">addr_from^10 addr_to addr_cc subject message</str>
```

edismax stands for Extended Disjunction max. One of the many features it supports is giving a boost to a document when the query matches the content of its boosted field.

The Query Fields parameter (qf) allow us to assign a boost factor to the provided list. So in our case we have the `addr_from` is boosted by 10 and all the other field have a default boost.

and in `managed-schema`, replace the type for `subject` and `message` fields from `string` to `text_general`.

Done from the changes right now. Restart the core and rerun the indexing process:

```
$ http -b :8983/solr/admin/cores action==RELOAD core==text2
$ bin/post -c text2 collection1/input4.xml
```

Let's compare searching for **Kari** and **Maija**:

```
$ http -b :8983/solr/text2/select q==kari
$ http -b :8983/solr/text2/select q==maija
```

You should see that the email ids show up in a different order, bringing Kari's and Maija's email ids to the top respectively.

But if we want to query **Meikäläinen**, how would we do that?

As long as this is done during both indexing and query time, the content will be found whether it is searched with or without the accents.

In fact let's go into **managed-schema** and replace `LowerCaseFilterFactory` with the `ICUFoldingFilterFactory` filter for :

- `email`
- `text_general`

types :

```
<filter class="solr.ICUFoldingFilterFactory"/>
```

We need to add the library references needed for `ICUFoldingFilterFactory` filter in the **`solrconfig.xml`** file (it should be at the top of the file):

```
<config>
  <lib dir="${user.dir}/../contrib/analysis-extras/lucene-libs/" />
  <lib dir="${user.dir}/../contrib/analysis-extras/lib/" />
```

solr.ICUFoldingFilterFactor is a token filter that does several steps at once (the unicode NFKC normalization, unicode aware case folding, and accent mapping).

It is not a part of the core Solr libraries. It part of the contrib/analysis-extras/ library in the example directory (substitution value of user.dir) directory.

Once more time, restart and rerun:

```
$ http -b :8983/solr/admin/cores action==RELOAD core==text2  
$ bin/post -c text2 collection1/input4.xml
```

If we query “meikalainen” from the text1 collection:

```
$ http -b :8983/solr/text1/select q==meikalainen
```

We will get no result.

But if we query it from the text2 collection:

```
$ http -b :8983/solr/text2/select q==meikalainen
```

We will get the documents where “Meikäläinen” was indeed mentioned.

```
{
  "response": {
    "docs": [
      {
        "addr_cc": [
          "Ivan Ivanovich Ivanov <vania@home.example.com>"
        ],
        "addr_from": "Maija Meikäläinen <maija@acme.example.com>",
        "addr_to": [
          "Kari Nordmann <kari@acme.example.com>"
        ],
        "date": "10 Jan 2012",
        "id": "email3",
        "message": "Kari, please find the proposed description attached. Vania is not feeling too well today. He will check his home email, if something is urgent. Salut, Maija",
        "subject": "Re: Updating vacancy description"
      },
      {
        "addr_from": "Kari Nordmann <kari@acme.example.com>",
        "addr_to": [
          "Maija Meikäläinen <maija@acme.example.com>",
          "Ivan Ivanovich Ivanov <ivan@acme.example.com>"
        ],
        "date": "6 Jan 2012",
        "id": "email2",
        "message": "Maija, Vania I need you to work together to update the description for Junior Java programmers. Fulan wants to hire some more for both New York and San Francisco location. See if we can wrap this up in a week. Thank you, Kari",
        "priority": 2,
        "subject": "Updating vacancy description"
      }
    ],
    "numFound": 2,
    "start": 0
  },
  "responseHeader": {
    "QTime": 2,
    "status": 0
  }
}
```

III. Usage Example with CSV dataset:

was looking around for a data set to work on with [Apache Solr](#) and shortly I found a [Video Games dataset](#).

So I downloaded it, and renamed the file to vgd.csv (Just a short name for Video Game Dataset).

As you can see it is a .csv file containing 1770 lines (not counting the one line of the columns name).

You can easily check this (if you are a Linux User) with:

```
$ wc -l vgd.csv
```

But also contains 166 columns, again you can easily check this with (If you know another way to count them please tell me.):

```
$ head -n 1 vgd.csv | tr ',' '\n' | wc -l
```

The problem now is that we have too much columns, and I don't know which will be more useful to us. For that we will omit the majority of the columns and we will work only with this list:

- Console
- Title
- US Sales (millions)
- YearReleased
- Publisher
- Genre

- Usedprice
- lnUsedPrice
- Review Score
- MaxPlayers
- Online
- Multiplatform

Let's start by launching Solr, creating a core, and indexing our data:

```
$ bin/solr start  
$ bin/solr create -c games  
$ bin/post -c games vgd.csv
```

You can check your new collection through the Admin WebUi:

<http://localhost:8983/solr/#/games>

The screenshot displays the Solr Admin UI for the 'games' index. The left sidebar contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu with 'games' selected. Below the dropdown are links for Overview, Analysis, Dataimport, Documents, Files, Ping, Plugins / Stats, Query, Replication, Schema, and Segments info.

The main content area is divided into several sections:

- Statistics:** Shows the last update 13 minutes ago. Metrics include Num Docs: 1770, Max Doc: 1770, Heap: -1, Memory Usage, Deleted: 0, Docs, Version: 6, Segment: 1, and Count. Status indicators show 'Optimized' and 'Current' as green checkmarks.
- Instance:** Lists configuration paths: CWD: /home/dmidma/Downloads/solr-6.4.2/server, Instance: /home/dmidma/Downloads/solr-6.4.2/server/solr/games, Data: /home/dmidma/Downloads/solr-6.4.2/server/solr/games/data, Index: /home/dmidma/Downloads/solr-6.4.2/server/solr/games/data/index, and Impl: org.apache.solr.core.NRTCachingDirectoryFactory.
- Replication (Master):** A table showing replication status:

	Version	Gen	Size
Master (Searching)	1491158575081	2	1.59 MB
Master (Replicable)	1491158575081	2	-
- Healthcheck:** A message stating 'Ping request handler is not configured with a healthcheck file.'

At the bottom, there are links to Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

In the statistics you can see that we have Num Docs: 1770.

But more importantly, if you go to: <http://localhost:8983/solr/#/games/query> and “Execute Query” with q=: (which returns all the documents), you will see all these fields:

The screenshot shows the Solr Admin UI at `localhost:8983/solr/#/games/query`. The left sidebar contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu for 'games' with options: Overview, Analysis, Dataimport, Documents, Files, Ping, Plugins / Stats, Query (selected), Replication, Schema, and Segments info.

The main panel displays the 'Request-Handler (qt)' section with the following fields:

- `/select`
- `q`: `.*`
- `fq`
- `sort`
- `start, rows`: `0` to `10`
- `fl`
- `df`
- `Raw Query Parameters`: `key1=val1&key2=val2`
- `wt`: `json`
- ☒ `indent`
- ☐ `debugQuery`
- ☐ `dismax`
- ☐ `edismax`
- ☐ `hl`
- ☐ `facet`
- ☐ `spatial`
- ☐ `spellcheck`

The `Execute Query` button is visible at the bottom of the request handler section.

The response is displayed in the right panel as a JSON object:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": ".*",
      "indent": "on",
      "wt": "json",
      "_": "1491158589311"
    }
  },
  "response": {
    "numFound": 1770,
    "start": 0,
    "docs": [
      {
        "Console": ["Nintendo DS"],
        "Title": ["Super Mario 64 DS"],
        "Block4": [1],
        "Block2": [1],
        "Block1": [1],
        "Block0.5": [1],
        "YearReleased": [2004],
        "2004": [1],
        "2005": [0],
        "2006": [0],
        "2007": [0],
        "2008": [0],
        "2009": [0],
        "2010": [0],
        "YearReleasedSq": [4016016],
        "Publisher": ["Nintendo Australia Pty., Ltd., Nintendo Co., Ltd., Ni"],
        "Genre": ["Action"],
        "Sequel": [1],
        "Re-release": [1],
        "UsedPrice": [24.95],
        "lnUsedPrice": [3.216873822],
        "ReviewSq": [7225],
        "RatingE": [1],
        "RatingT": [0],
        "RatingM": [0],
        "Lifecycle": [0],
        "LifecycleSq": [0],
        "MaxPlayers": [1],
        "MaxPlayersSq": [1],
        "Online": [0],
        "Licensed": [0],
        "Handheld": [1],
        "Accessory": [0],
        "LtdEdition": [0],
        "Multiplatform": [0],
        "GBA": [0]
      }
    ]
  }
}
```

If you noticed that all the fields are multivalued (they in brackets []) which mean they can take more than one value.

When you create a core in solr without an explicit configuration, it will be configured to the schemaless mode.

You can read more about it [here](#).

One thing before running the request and getting only the fields that we will play with. Some of the column names in the CSV have spaces and parentheses that will be automatically replaced with underscores.

The output of our query will be redirected to a new file fvgd.csv (filtered video game data) [Don't judge the name =P]:

```
$ http -b :8983/solr/games/select q==*:* wt==csv rows==1770 fl=="Console, Title, US_Sales__millions_, YearReleased, Publisher, Genre, Usedprice, lnUsedPrice, Review_Score, MaxPlayers, Online, Multiplatform" > fvgd.csv
```

Quick overview of the parameters used in the query:

- *q=*:**
Returns all documents without any specific constraint.
- *wt=csv*
Output format will be csv.
- *rows=1770*
Returns all 1770 lines.
- *fl="list_of_fields"*
This is the list of field that our query will output.

Before jumping to the configuration, I am going to change the names of the columns to the following list:

- Console -> console
- Title -> title
- US_Sales__millions_ -> us_sales_millions
- YearReleased -> year_released
- Publisher -> publisher
- Genre -> genre
- Usedprice -> used_price

- lnUsedPrice -> ln_used_price
- Review_Score -> review_score
- MaxPlayers -> max_players
- Online -> online
- Multiplatform -> multi_platform

You can change them manually or run this command:

```
$ sed -i.old  
'1s;.*;console,title,us_sales_millions,year_released,publisher,genre,used_p  
rice,ln_used_price,review_score,max_players,online,multi_platform;'  
fvgd.csv
```

Go ahead and create a directory (I will name mine game_config) with the two files in it (managed-schema, solrconfig.xml).

- **managed-schema:**


```

<?xml version="1.0" encoding="UTF-8" ?>
<schema version="1.5">
<fields>
<field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false"/>
<field name="console" type="string" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="title" type="string" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="us_sales_millions" type="float" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="year_released" type="string" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="publisher" type="space_comma_delimited" indexed="true"
stored="true" required="true" multiValued="true"/>

<field name="genre" type="string" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="used_price" type="float" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="ln_used_price" type="float" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="review_score" type="int" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="max_players" type="int" indexed="true" stored="true"
required="true" multiValued="false"/>

<field name="online" type="boolean" indexed="true" stored="true"
required="true" multiValued="false"/>
<field name="multi_platform" type="boolean" indexed="true" stored="true"
required="true" multiValued="false"/>

</fields>
<uniqueKey>id</uniqueKey>
<types>
<fieldType name="space_comma_delimited" class="solr.TextField"
multiValued="true">
  <analyzer>
    <tokenizer class="solr.PatternTokenizerFactory" pattern="\s*,\s*" />
    <filter class="solr.LowerCaseFilterFactory" />
  </analyzer>
</fieldType>
<fieldType name="int" class="solr.TrieIntField" precisionStep="0"
positionIncrementGap="0"/>
<fieldType name="boolean" class="solr.BoolField" sortMissingLast="true" />
<fieldType name="string" class="solr.StrField" sortMissingLast="true"
docValues="true" />

```

```
<fieldType name="float" class="solr.TrieFloatField" precisionStep="0"
docValues="true" />

</types>
</schema>
```

- **solrconfig.xml:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
<uceneMatchVersion>6.4.2</uceneMatchVersion>

<updateRequestProcessorChain name="dedupe">
  <processor class="solr.UUIDUpdateProcessorFactory"> <!-- adds a newly
generated UUID value to any document that does not have an id -->
    <str name="fieldName">id</str>
  </processor>
  <processor class="solr.LogUpdateProcessorFactory" /> <!-- Tracks the
commands processed during this request and logs them -->
  <processor class="solr.RunUpdateProcessorFactory" /> <!-- Executes the
update using internal solr APIs -->
</updateRequestProcessorChain>

<requestDispatcher handleSelect="false">
<httpCaching never304="true" />
</requestDispatcher>
<requestHandler name="/select" class="solr.SearchHandler" />
<requestHandler name="/update" class="solr.UpdateRequestHandler">
  <lst name="defaults">
    <str name="update.chain">dedupe</str>
  </lst>
</requestHandler>
<requestHandler name="/analysis/field"
class="solr.FieldAnalysisRequestHandler" startup="lazy" />
</config>
```

Now we need to stop Solr, delete the old games core and recreate it with our config and our new data:

```
$ bin/solr stop -all
```

```
$ rm -rf server/solr/games  
$ bin/solr start  
$ bin/solr create -c games -d game_config  
$ bin/post -c games fvgd.csv -pramas  
"f.publisher.split=true&g.commit=true"
```

If you check it in the Admin WebUI, you should find this :

The screenshot shows the Solr Admin UI at `localhost:8983/solr/#/games/query`. The left sidebar contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu for 'games'. The main area displays the 'Request-Handler (qt)' as '/select', the 'q' field with a query, and the 'wt' set to 'json'. The 'Execute Query' button is visible. The response is a JSON array of game records, including details like console, title, sales, year released, publisher, genre, price, review score, and online status.

If you want to change the configuration of the files and re-post the data, you need to do first:

```
bin/post -c games -d '<delete><query>*:*/</query></delete>'
```

Because the ids are generated automatically, every post command will append another 1770 records.

Add these fields into the `fields` section:

```
<field name="consoles" type="facets_type" multiValued="true" indexed="true"
stored="false" />
<field name="titles" type="facets_type" multiValued="true" indexed="true"
stored="false" />
<field name="genres" type="facets_type" multiValued="true" indexed="true"
stored="false" />
<copyField source="console" dest="consoles" />
<copyField source="title" dest="titles" />
<copyField source="genre" dest="genres" />
```

And this to the `types` section:

```
<fieldType name="facets_type" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.KeywordTokenizerFactory" />
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.KeywordTokenizerFactory" />
  </analyzer>
</fieldType>
```

Before querying, we need to reload our core, delete the old records and reindex:

```
$ http -b :8983/solr/admin/cores action==RELOAD core==games
$ bin/post -c games -d '<delete><query>*:*/query</delete>'
$ bin/post -c games fvgd.csv -params "f.publisher.split=true&g.commit=true"
```

So let's see how many games we do have for each console and how many times the title of the same game does repeat (which mean if the game is multi platform or not):

```
$ http -b :8983/solr/games/select q==*:~ facet==true facet.field==consoles
facet.field==titles facet.field==genres wt==json
```

```

{
  "facet_counts": {
    "facet_fields": {
      "consoles": [
        "x360",
        437,
        "nintendo ds",
        418,
        "playstation 3",
        312,
        "sony psp",
        307,
        "nintendo wii",
        296
      ],
      "genres": [
        "action",
        660,
        "sports",
        228,
        "strategy",
        91,
        "action\\, role-playing (rpg)",
        88,
        "racing / driving",
        73,
        "role-playing (rpg)",
        71,
        "action\\, adventure",
        62,
        "action\\, strategy",
        62,
        "action\\, racing / driving",
        60,
        "action\\, simulation",
        57,
        "simulation",
        50,
        "adventure",
        35,
        "action\\, sports",
        34,
        "role-playing (rpg)\\, strategy",
        29,
        "racing / driving\\, sports",
        25,
        "simulation\\, strategy",
        15,
        "racing / driving\\, simulation\\, sports",
        11,
        "simulation\\, sports",
        11,
        "racing / driving\\, simulation",
        10,
        "action\\, racing / driving\\, sports",
        9,
        "adventure\\, simulation",
        8,
        "action\\, adventure\\, role-playing (rpg)",

```

```

1
],
"titles": [
    "harry potter and the order of the phoenix",
    5,
    "iron man",
    5,
    "lego batman: the videogame",
    5,
    "mx vs. atv untamed",
    5,
    "spider-man 3",
    5,
    "the simpsons game",
    5,
    "wwe smackdown vs. raw 2008",
    5,
    "fantastic four: rise of the silver surfer",
    4,
    "fifa soccer 08",
    4,
    "ghostbusters: the video game",
    4,
    "harry potter and the half-blood prince",
    4,
    "juiced 2: hot import nights",
    4,
    "kung fu panda",
    4,
    "lego indiana jones: the original adventures",
    4,
    "lego star wars: the complete saga",
    4,
    "madden nfl 07",
    4,
    "madden nfl 08",
    4,
    "marvel ultimate alliance",
    4,
    "marvel ultimate alliance 2",
    4,
    "monsters vs. aliens",
    4,
    "sega superstars tennis",
    4,
    "shrek the third",
    4,
    "surf's up",
    4,
    "the bigs",
    4,
    "the golden compass",
    4,
    "tiger woods pga tour 07",
    4,
    "tony hawk's proving ground",
    4,
    "wwe smackdown! vs raw 2009",
    4,
    "band hero",

```

Yess, that neat =D

So what's the top most popular genres that were popular and in which year?

You can do this with this query:

```
$ curl http://localhost:8983/solr/games/select -d
'q=*&wt=json&indent=true&json.facet={"year": {"terms": {"field":
"year_released", "limit":10, "facet":{"genre": {"terms": {"field":"genre",
"limit":3, "facet": {"top_title": {"terms":{"field":"title",
"limit":2}}}}}}}}}'
```

This is done by using [Faceted Search](#).

You will get something like this:

```
"facets":{
  "count":1770,
  "year":{
    "buckets":[{
      "val":"2007",
      "count":465,
      "genre":{
        "buckets":[{
          "val":"Action",
          "count":187,
          "top_title":{
            "buckets":[{
              "val":"Harry Potter and the Order of the Phoenix",
              "count":5},
              {
                "val":"The Simpsons Game",
                "count":5}]]},
          {
            "val":"Sports",
            "count":66,
            "top_title":{
              "buckets":[{
                "val":"FIFA Soccer 08",
                "count":4},
                {
                  "val":"Madden NFL 08",
                  "count":4}]]},
          {
            "val":"Strategy",
            "count":30,
            "top_title":{
              "buckets":[{
                "val":"7 Wonders of the Ancient World",
                "count":2},
                {
                  "val":"Jenga World Tour",
                  "count":2}]]}}],
            {
              "val":"2008",
              "count":444,
              "genre":{
                "buckets":[{
                  "val":"Action",
                  "count":150,
                  "top_title":{
                    "buckets":[{
                      "val":"Iron Man",
                      "count":5},
                      {
                        "val":"Kung Fu Panda",
                        "count":4}]]},
                  {
                    "val":"Sports",
                    "count":58,
                    "top_title":{
                      "buckets":[{
                        "val":"Sega Superstars Tennis",
```

The query is long and kind of ugly, But you can luckily make it easier to understand [Parameter Substitution](#).

```
$ curl http://localhost:8983/solr/games/select -d
'q=*&wt=json&rows=1&indent=true&year=year_released&limit=3&json.facet={"y
ear": {"terms": {"field": ${year}, "limit":10, "facet":{"genre": {"terms":
{"field":"genre", "limit":${limit}, "facet": {"top_title":
{"terms":{"field":"title", "limit":${limit}}}}}}}}}'
```

But more interestingly, you can make this shorter and much more cleaner using [RequestHandler](#).