Solr Unleashed 5.x

Self-paced Labs

Table of Contents

Introduction

This activity guide accompanies the Solr Unleashed course slides. It contains labs designed to provide effective opportunities for you to practice concepts and techniques explored in the class.

Throughout this document, $installdir refers to the directory where your course materials are installed, $version is the version of Solr bundled within those materials, and $release is the release number of this distribution.

Answers to selected questions

Most labs contain questions that help you reflect upon and explore specific aspects of the course. You are encouraged to interact with other students and the instructor while exploring these questions.

1. Lab 1 – Getting Solr Up and Running

30 minutes

The Solr server is a HTTP server that uses a Java Virtual Machine to operate. The server runs stand-alone using the Jetty Server library and is operated using "shell" scripts.

Goals

At the end of this lab you will be able to:

1. Install Solr and the course materials.
2. Start, stop and test Solr.

1.1. Task 1 – Before You Run Solr

In this exercise, you will learn how to operate the Solr server. You will be able install, start, stop, and reconfigure the basic Solr server environment.

Before we get started, a little bit about the Solr server. Solr requires a Java Virtual Machine to operate. Ensure you have the at-latest Java 7 installed before you start. You can check by simple opening a "terminal" window and executing the following command:

```
java -version
```

This should produce an output to the terminal declaring the version of Java, similar to something like this:

```
java version "1.8.0_31"
Java(TM) SE Run-time Environment (build 1.8.0_31-b13)
Java HotSpot(TM) Client VM (build 25.31-b07, mixed mode, sharing)
```

If for some reason Java is not found, please locate and install the latest version of Java. You can find all the available downloads here: http://www.oracle.com/technetwork/java/javase/downloads/index.html

1.2. Task 2 – Installing and Running Solr

1. Open a command/terminal/shell window.
2. Change to $installdir (the training material folder) and unpack the file solr-$version.zip into the same directory.
3. Change to $installdir/solr-$version/bin.

Optional: Create an environment variable on your Unix/Linux/Mac/Windows environment that points to your installation. This will save you time later on in the labs when you have to refer to $installdir from the command line.

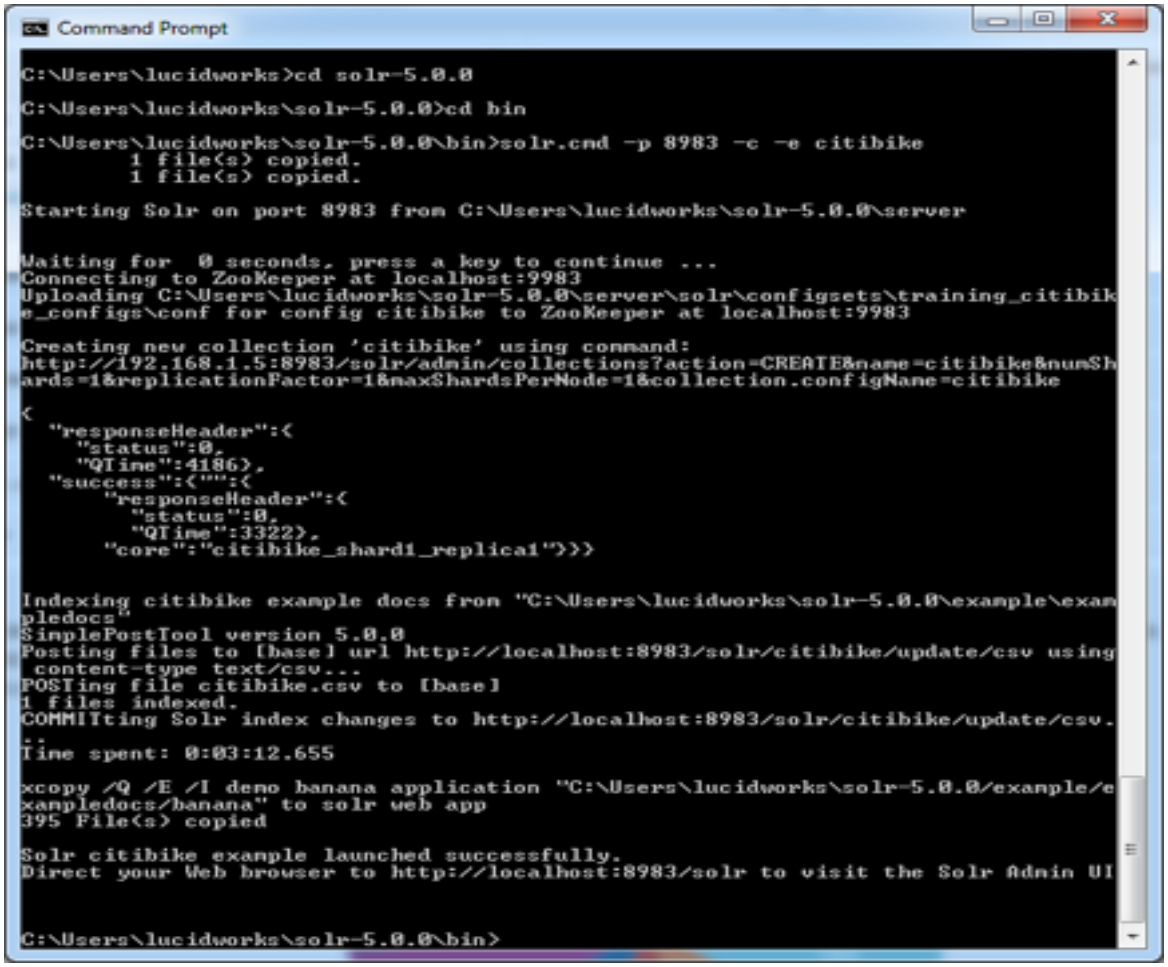
The first time we start Solr we include a command line "switch" that will load a predefined seed data set into Solr. The "-e citibike" will be used once to setup an example data set that we shall use for the first four labs of the course.

On Unix/Linux/Mac environments, start Solr with this command:

```
sh ./solr -c
```

On Windows use this command:

```
solr.cmd -c
```



Open the Solr admin panel at http://localhost:8983/solr/admin . If you can see the admin interface to Solr, then you have successfully installed Solr!

5. We will create an collection called 'citibike' for our labs.

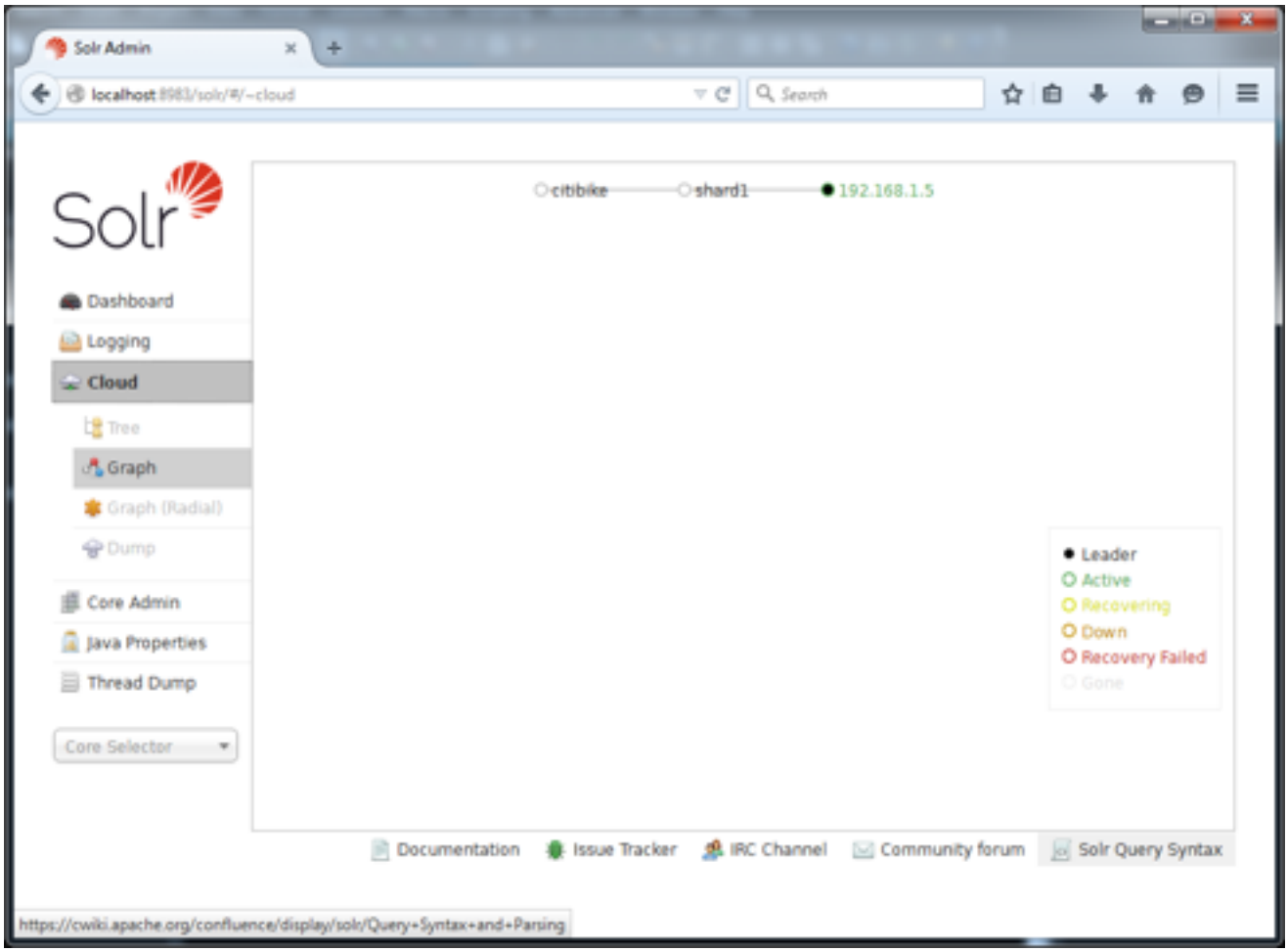
On Unix/Linux/Mac environments, run this command:

```
sh ./solr create_collection -c citibike -n data_driven
```

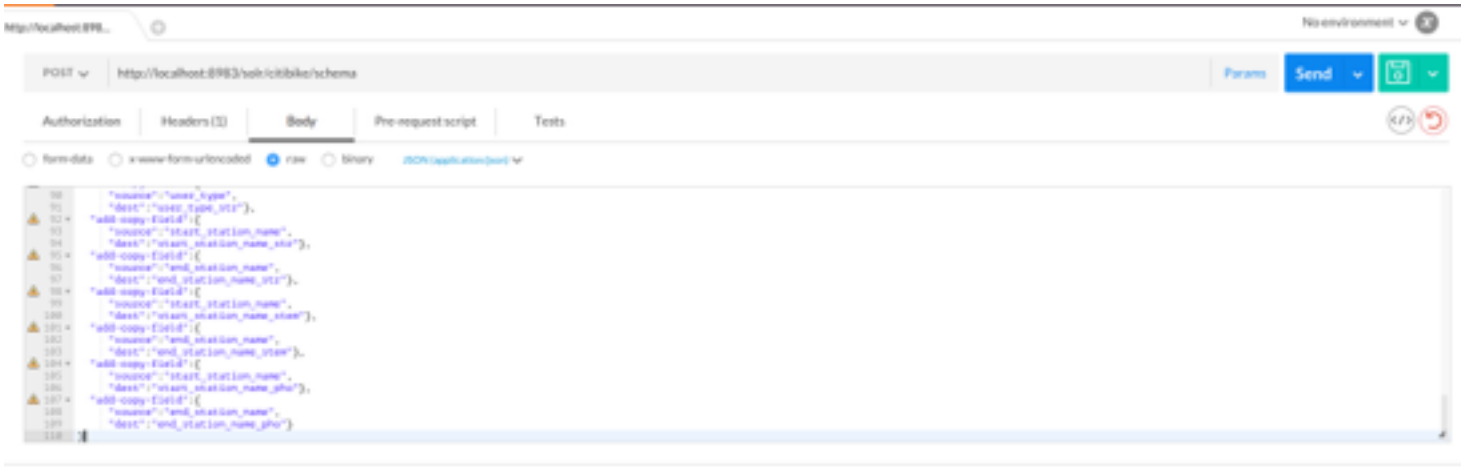
On Windows use this command:

```
solr.cmd create_collection -c citibike -n data_driven
```

Refresh the admin panel and click on 'Cloud' in the left pane. You should see the citibike collection with a single shard.



6. Let's define the schema for the citibike data that we will load in the next section. Run the command posted in citibike-schema-load.txt to define the necessary fields . For Windows users please take the JSON from that file and run it from Postman like shown in the screenshot below. Non windows users can also use this method if they prefer to instead of curl

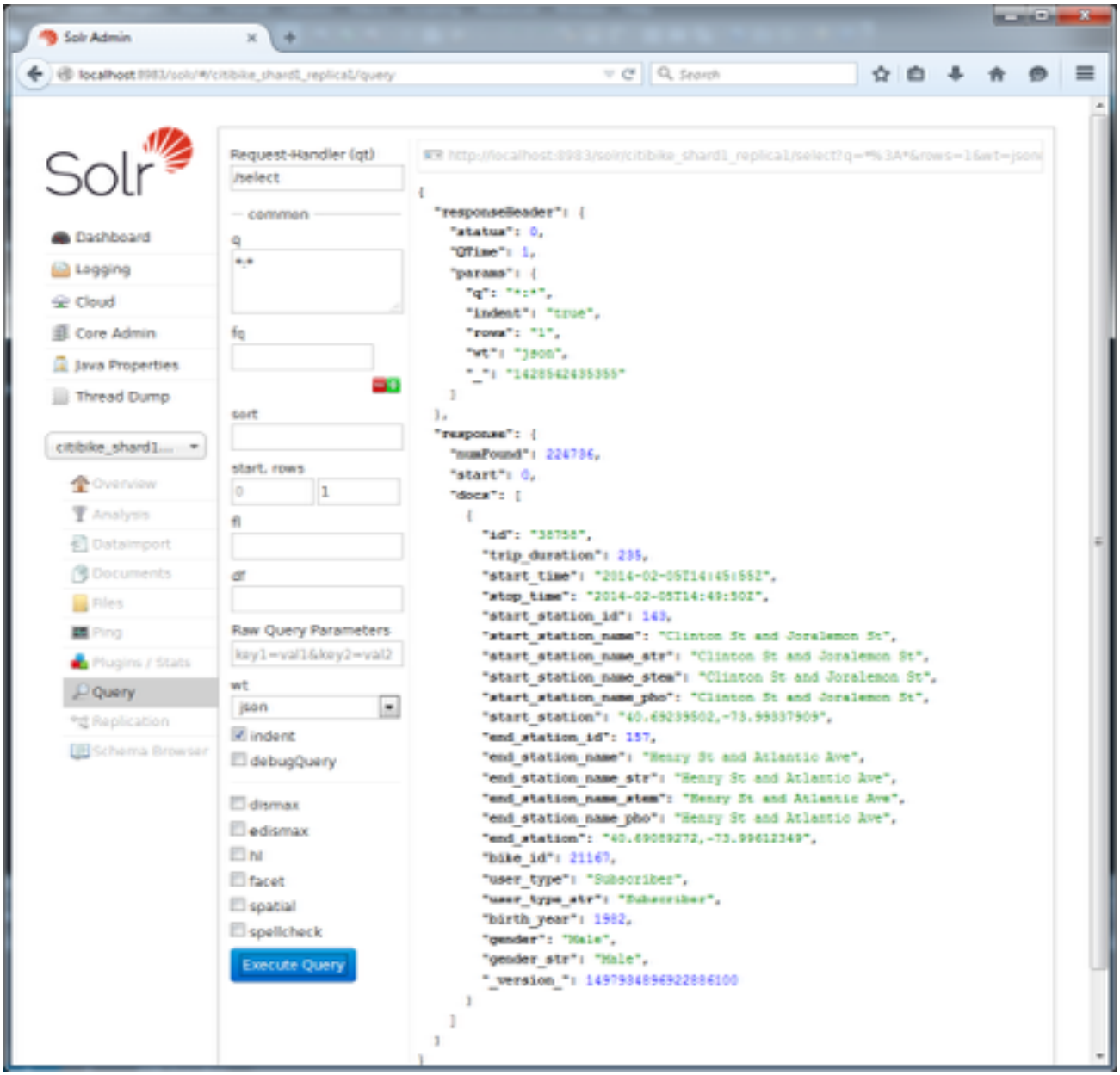


7. Let's load the data into the collection now.

Change to $installdir/solr-$version/example/exampledocs

Run the following command : java -Dc=citibike -Dtype=text/csv -jar post.jar $installdir/biketrips.csv

8. Click on Core Selector at the bottom of the left pane, select citibike_shard1_replica1 and then click on Query. Click Execute Query (the blue button at the bottom of the screen) without changing any of the outputs. JSON output will appear on the right, allowing you to inspect sample individual documents/records in the citibike data.



You have now successfully setup the citibike Solr collection!

1.3. Task 3 – Stopping and Restarting Solr

1. Change to $installdir/bin

   ./solr stop -p 8983

   For Windows use,

   solr.cmd stop -p 8983

2. Start it back up with the steps described above.

2. Lab 2 – Using Solr

75 minutes

The Solr query interface provides all the necessary features to build a responsive “Search Interface” using almost any technology. Using the common HTTP protocol to interact with specialized Solr REST services allows development of user interfaces using such technologies as:

- Java
- c#
- PHP
- JavaScript

Goals

At the end of this lab you will be able to:

1. Identify the parameters in the URL.
2. Explain the effects of setting specific parameters in the URL.
3. Adjust Solr's output by changing a few parameters in the URL

Preparation for Querying

Solr is much more than a mechanism for storing text for search. In addition to use cases that are based on full-text search, Solr is also used as a NoSQL store for key-value lookups. Solr provides powerful analysis capabilities and can be used to build fast, scalable business intelligence-type applications, in addition to core search applications.

Many students come to Solr from a database background. To facilitate the transition, we begin with an example the shows how Solr can be used a powerful NoSQL store, with the ability to do fuzzy and wildcard searches.

Search application development contains the following three elements; extracting, transforming and loading the data; persisting the data in a format that supports the desired user experience; and querying the search engine & designing the user's search experience.

During Day One of the class, we are going to use the "citibike" collection and focus on the querying aspect — the search engine and user experiences.

For this first set of labs, we will be using a dataset that contains information about bike rides taken in New York City using a service called CitiBike. CitiBike provides a convenient solution for quick trips around New York City, with a number of bike stations dispersed throughout the city. All a rider needs to do is unlock a bike from any station, ride wherever they want and return to a destination station. All rides are logged and stored in csv files that contain the following information:

- Trip Duration (seconds)
- Start Time and Date
- Stop Time and Date
- Start Station Name
- End Station Name
- Station ID
- Start/End Station Lat/Long
- Bike ID
- User Type (Customer = 24 hour pass or 7-day pass user. Subscriber = Annual Member)
- Gender (Zero = unknown, 1 = male, 2 = female)
- Year of Birth

This data enables us to ask interesting questions such as:

- "Which stations are most popular as a starting point?"
- "What is the distribution of trip durations? Do men take longer trips than women?"
- "Which stations are frequented most by each gender?"
- "For a given start station, which is the most popular end station?" etc.

The company operating these bikes could use these insights to improve operations (for example making sure there are enough bikes available at each station) and/or provide/sell more targeted advertising for the riders (based on age and gender distributions of the riders frequenting a station).

For our exercises, we have pre-loaded data for the month of February in the year 2014.

Solr Fields

Just like a traditional database, Solr stores information in "fields" that together make "document". So Solr documents are to database tables as fields are to columns in a table. There are different "types" of fields that have different purposes:

- Simple Strings
- Tokenized or Analyzed Terms
- Geo-spatial or Latitude and Longitude
- Dates
- Numbers
- etc…

You will need to use multiple field types while building search applications. You will also sometimes "index" the same information in more than one field. We will cover field definitions and "copyField" directives later in the class; for now just be aware of this concept.

Solr Query Fundamentals

The Solr server at its core is a HTTP server that processes different types of requests. The server supports operations such as:

- Representational State Transfer (REST) requests
- XML formatted requests
- Comma Separated Values (CSV) requests

During the following exercises we will be using the "/solr/select" REST service to query solr and retrieve content based on "parameters" included with the request (in a moment we will show you how to build such a request). The solr "/solr/select" service supports a broad range of "parameters" that provide a robust query API to the solr data. The most fundamental parameters you will use are:

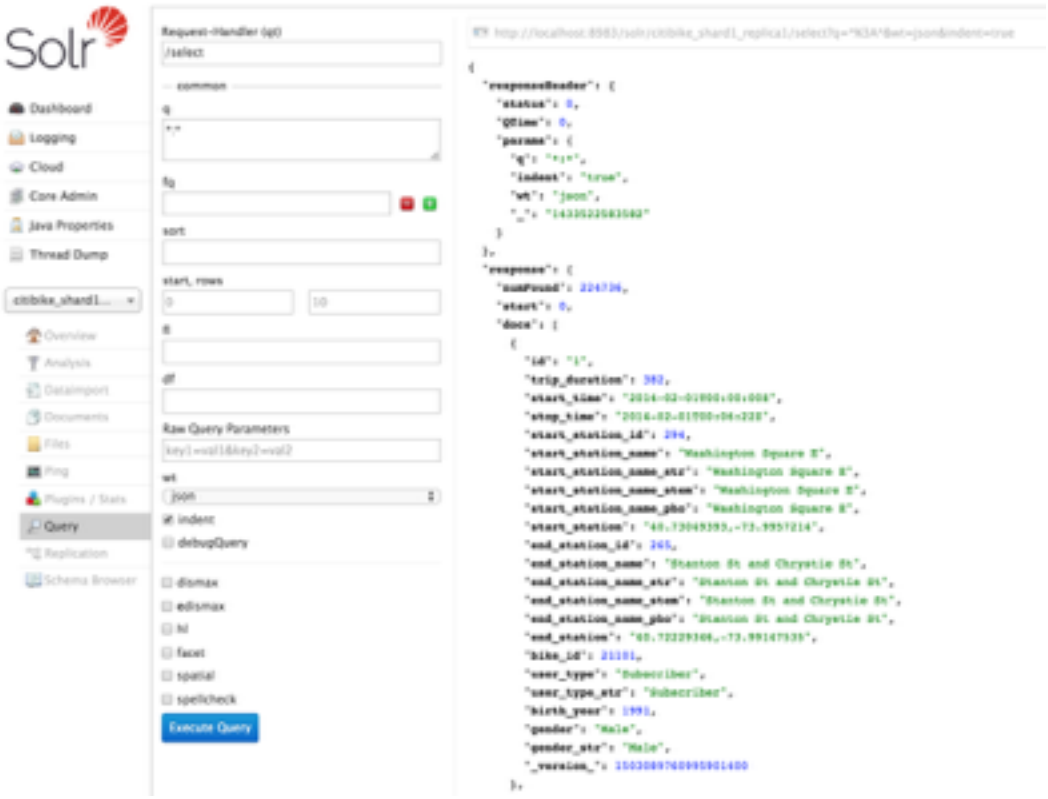
- q → solr query string
- sort → result set ordering
- fl → field list for the result set
- start → starting document result index to return
- rows → number of rows to return
- wt → format of the result set, we will use "json" for most examples
- fq → solr query string that will not affect scoring or relevance

Throughout the training, we will be exploring many other parameters that are available for searching.

2.1. Task 1 – Simple Queries on Specific Fields

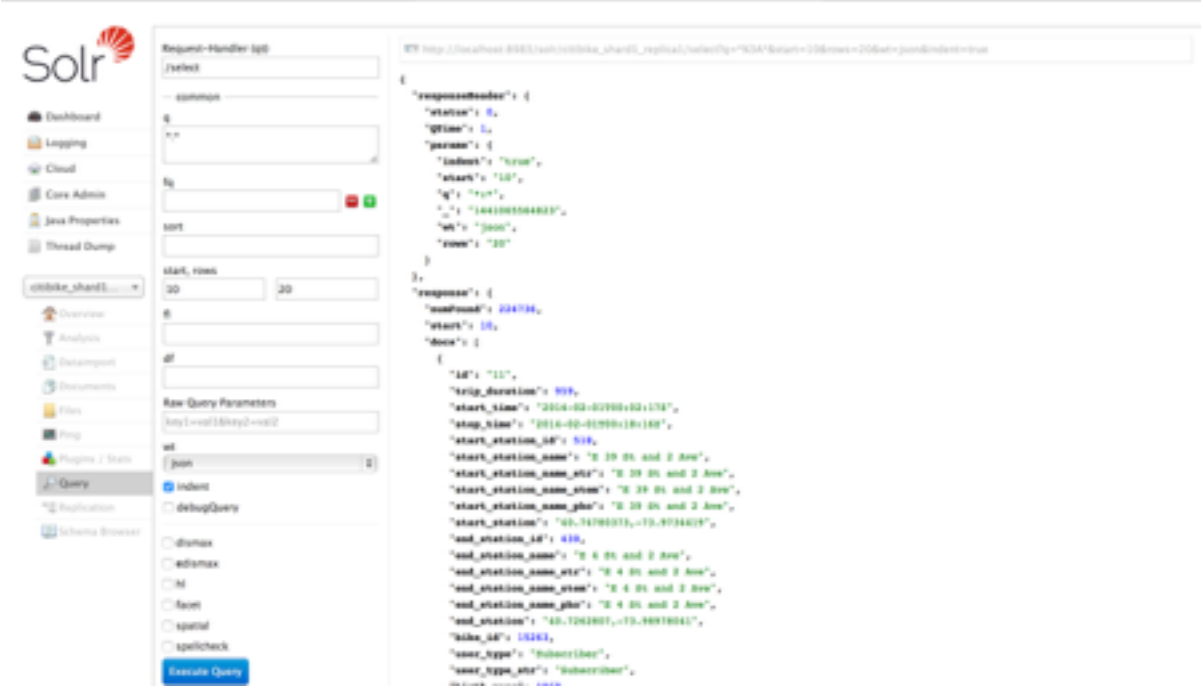
Setup

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the “citibike” core from the “core selector”.
3. Select the “Query” option below the citibike in the core selector.
4. Complete the labs task list in order.

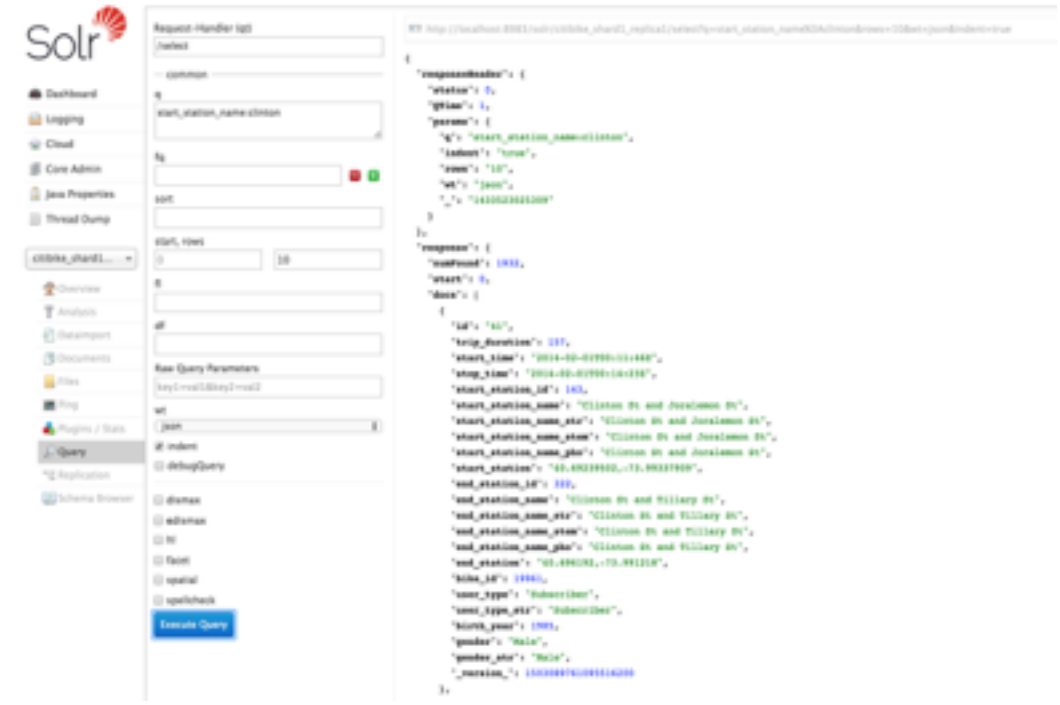


1. Select the “Execute Query” button at the bottom of the page. Study the URL for the query that is displayed at the top of the admin UI page. Clicking on it will take you to a browser window that displays the Solr response. Hit the back button and go back to the Solr Admin UI. You will see a number of documents with various fields. Many fields seem to be copies of each other; we will cover field types and their uses later in the class.

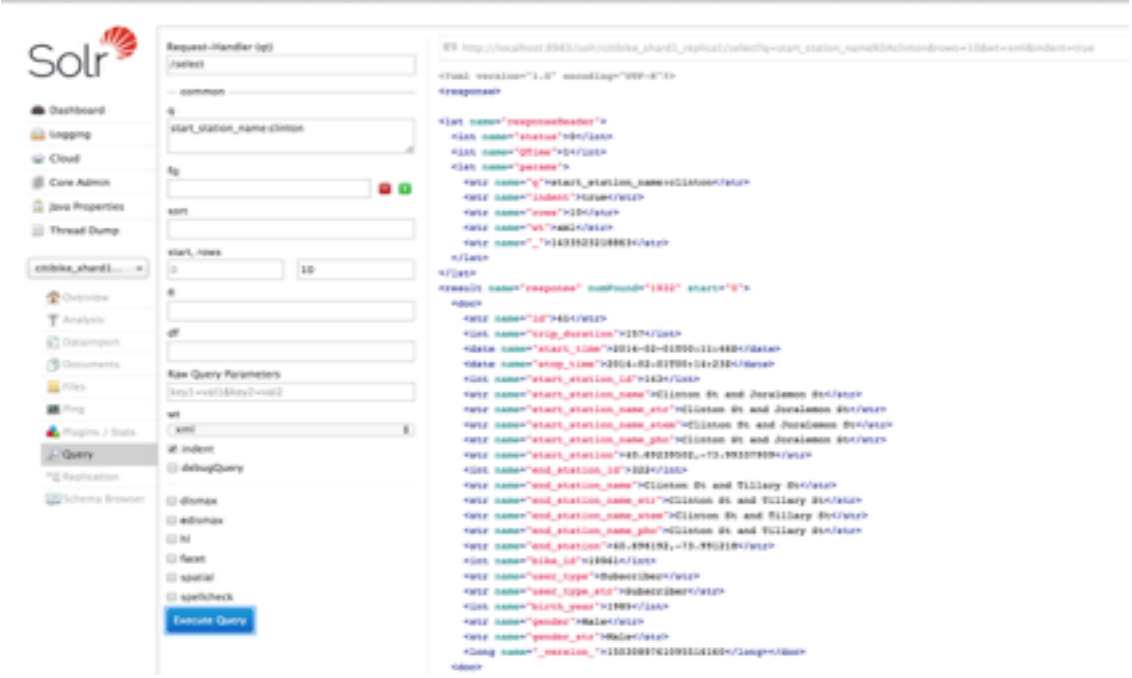
2. Instead of displaying the top 10 results, display results the next 20 results by changing the start and rows parameter. Review the URL and see how the parameters are modified.



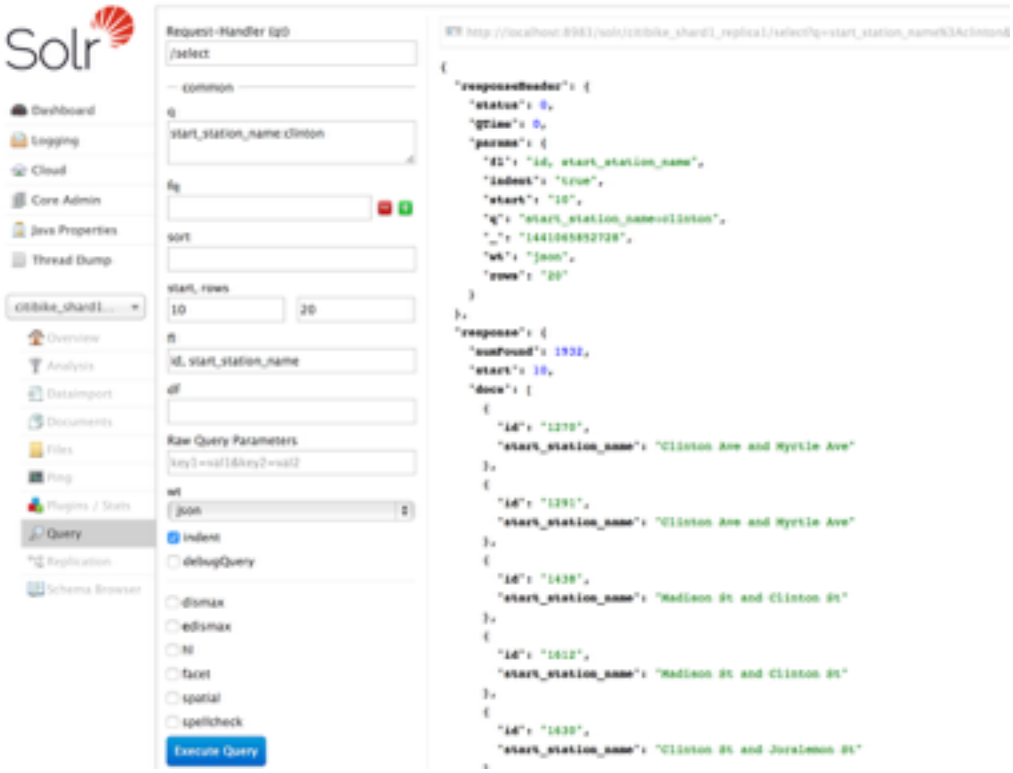
3. Change the query to start_station_name:clinton.



4. Change the wt option from json to xml. How does the response change?



5. Add id, start_station_name to the “fl” text field and set the wt back to json. How does the response change?



6. Try changing the query to start_station_name_str:clinton. What is the result? Notice that the results are different from start_station_name:clinton. The reason is that the two fields are of different types (“string” vs. “text_general”). We will explain this later in the class.

7. Change the query back to start_station_name:clinton. Click on the URL. Add &omitHeader=true to the end of the URL. How does the response change?

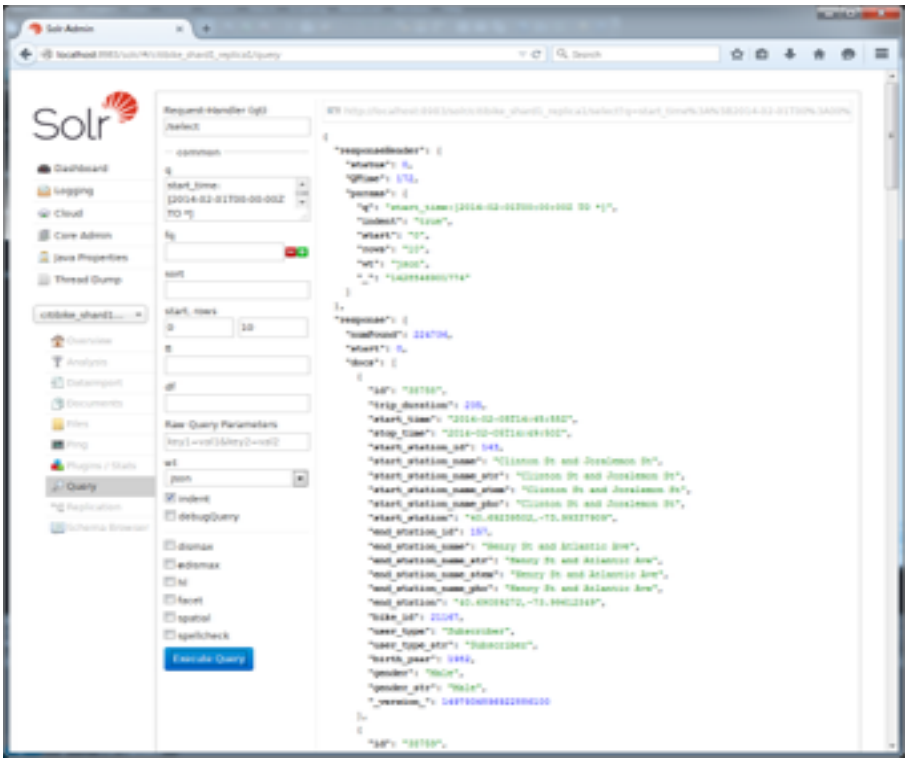

```
{
  "response":{"numFound":1932,"start":0,"docs":[
      {
        "id":"41",
        "trip_duration":153,
        "start_time":"2014-02-01T00:11:46Z",
        "stop_time":"2014-02-01T00:14:23Z",
        "start_station_id":163,
        "start_station_name":"Clinton St and Joralemon St",
        "start_station_name_str":"Clinton St and Joralemon St",
        "start_station_name_stem":"Clinton St and Joralemon St",
        "start_station_name_pho":"Clinton St and Joralemon St",
        "start_station":"40.69239582,-73.99337909",
        "end_station_id":322,
        "end_station_name":"Clinton St and Tillary St",
        "end_station_name_str":"Clinton St and Tillary St",
        "end_station_name_stem":"Clinton St and Tillary St",
        "end_station_name_pho":"Clinton St and Tillary St",
        "end_station":"40.696192,-73.991218",
        "bike_id":19961,
        "user_type":"Subscriber",
        "user_type_str":"Subscriber",
        "birth_year":1985,
        "gender":"Male",
        "gender_str":"Male",
        "_version_":1503089761095516160},
      {
        "id":"123",
        "trip_duration":283,
        "start_time":"2014-02-01T00:38:23Z",
        "stop_time":"2014-02-01T00:43:06Z",
        "start_station_id":366,
        "start_station_name":"Clinton Ave and Myrtle Ave",
        "start_station_name_str":"Clinton Ave and Myrtle Ave",
        "start_station_name_stem":"Clinton Ave and Myrtle Ave",
        "start_station_name_pho":"Clinton Ave and Myrtle Ave",
        "start_station":"40.693261,-73.968896",
        "end_station_id":416,
        "end_station_name":"Cumberland St and Lafayette Ave",
        "end_station_name_str":"Cumberland St and Lafayette Ave",
        "end_station_name_stem":"Cumberland St and Lafayette Ave",
        "end_station_name_pho":"Cumberland St and Lafayette Ave",
        "end_station":"40.68753606,-73.97265183",
        "bike_id":20051,
        "user_type":"Subscriber",
        "user_type_str":"Subscriber",
        "birth_year":1978,
        "gender":"Female",
        "gender_str":"Female",
        "_version_":1503089761117626513}]
  }}
```

2.2. Task 2 – Querying over Ranges of Values

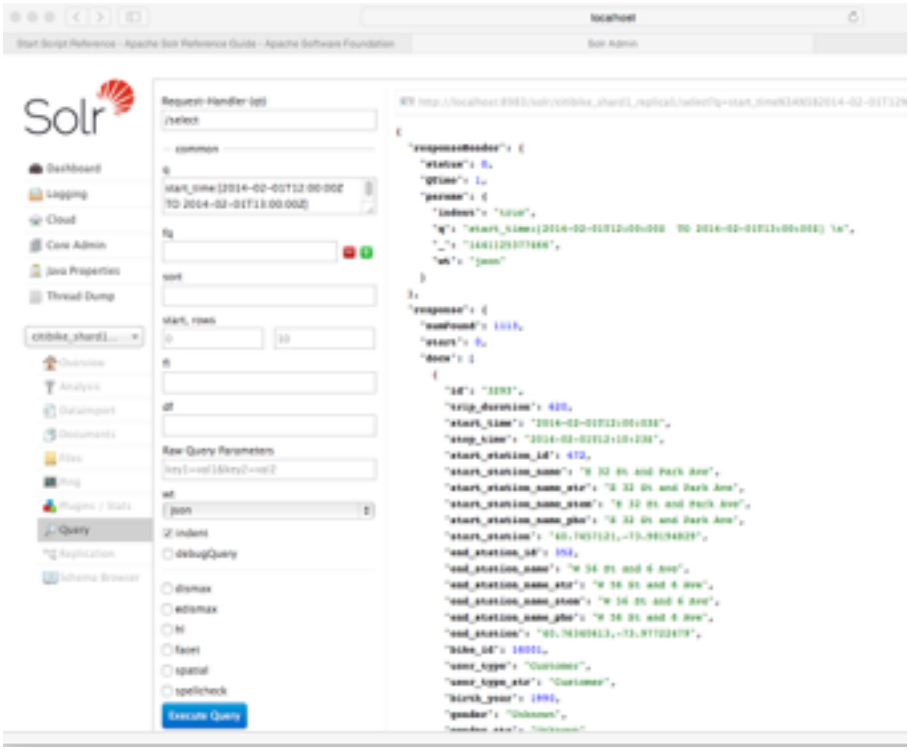
A range search specifies a range of values for a field (a range with an upper bound and a lower bound). The query matches documents whose values for the specified field or fields fall within the range. Range queries can be inclusive or exclusive of the upper and lower bounds.

Steps

1. Clear settings from previous lab before continuing.
2. Set the "q" field to start_time:[2014-02-01T00:00:00Z TO *] and click on the "Execute Query" button at the bottom of the page to get all data for this example. You should get 224,736 results. This query gives us all documents (records) where the field start_time is 2014-02-01T00:00:00Z or later.



3. Now find the rides that started between 12PM and 1PM UTC on 01 February 2014, both end points included.



4. Now we will look for ranges within the integer field trip_duration. Run a query to trip_duration:[0 TO 100} to find all rides of duration shorter than 100s.

Challenge Questions

1. How many female riders started a trip between 12pm and 1pm UTC on 03 February 2015? Answer: 6
2. How many riders took a trip less than 10 minutes (for the entire month of February)? Ans. 126105

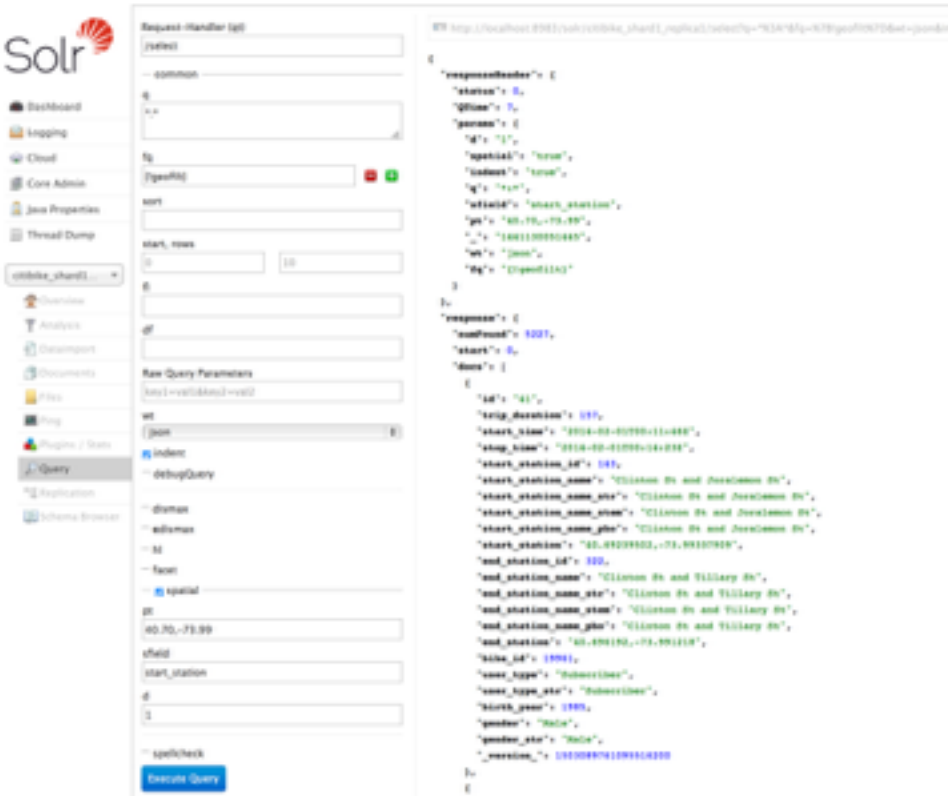
2.3. Task 3 – Geo-Spatial Searches

Solr supports location data for use in spatial/geospatial searches. Using spatial search, you can:

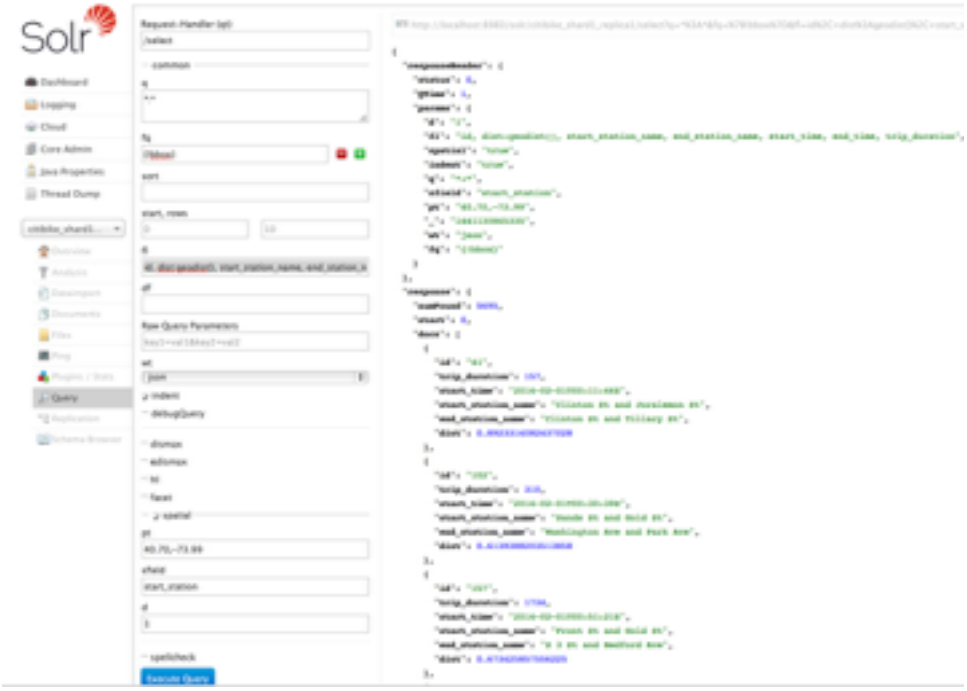
- Index points or other shapes
- Filter search results by a bounding box or circle or by other shapes
- Sort or boost scoring by distance between points, or relative area between rectangles

Steps

1. **Clear settings from previous lab before continuing.
2. Set the “q” field to *:* to get all data for this query.
3. Check the “spatial” box.
4. Set the “pt” to 40.70,-73.99
5. Set the “sfield” to start_station (this is the field used for geo-spatial queries)
6. Set the “d” field to 1 (1 km distance from start_station)
7. Set the “fq” field to {!geofilt} (this is the filter used to find stations within the specified distance)
8. Click on the “Execute Query” button at the bottom of the page. You should get 5,227 results.



9. Click on the URL and inspect the various elements of the URL.
10. Go back to the Solr Admin screen and change “{!geofilt}” to “{!bbox}”. The number of hits changes. The bounding box used by bbox is an approximation and not as accurate as geofilt, but is guaranteed to encompass all the point encompassed by geofilt. In many use cases, we use bbox for performance.
11. You can get distances of various stations from a specified point using the geodist() function. We have not covered function queries and pseudo fields as yet, but basically Solr calculates the geodist() function which can be treated like a field in the document. Keeping the same query, enter the following into the fl box: “id,dist:geodist(),start_station_name,end_station_name,trip_duration” In addition to the fields in the document, you will get a field called dist, with the distance (in kilometers).



12. You can sort by ascending order of geodist() by adding “geodist() asc” in the “sort” textbox.
13. Try sorting by descending order (“geodist() desc”). You will see geo distances greater than 1 due to the approximations used by bbox.

Challenge

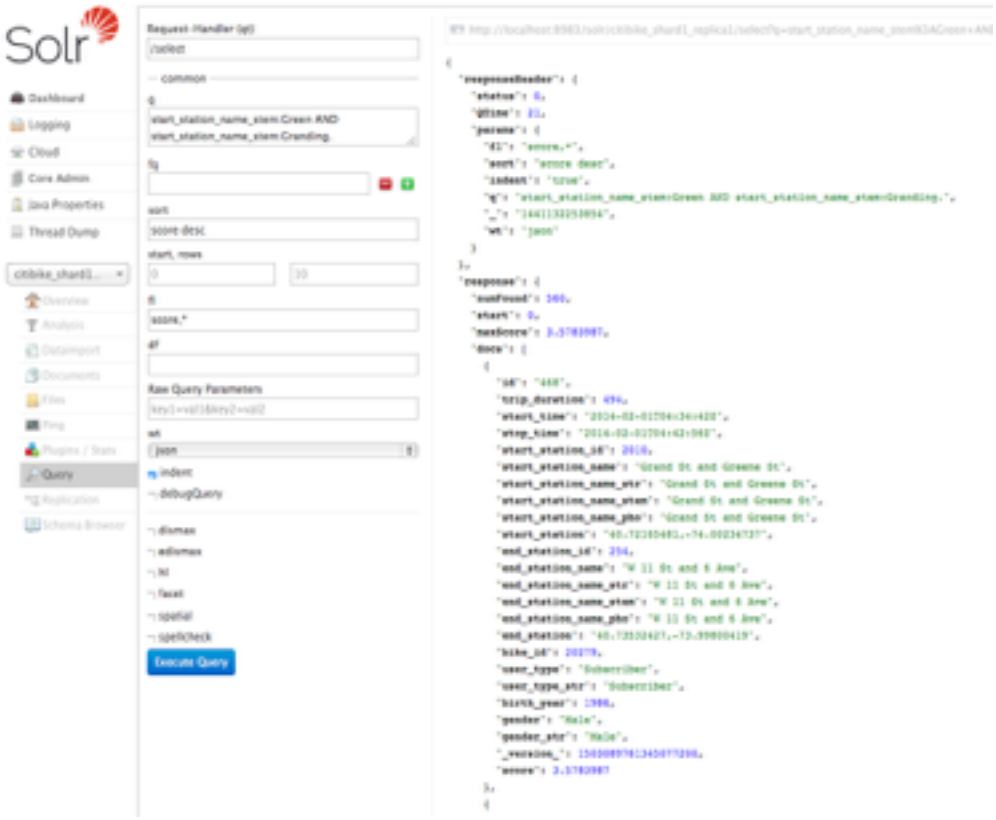
1. How can you compute the distance from [40.70, -73.99] to the station with start_station_id of 422?

2.4. Task 4 – Using Stemming in your search

The concept of “stemming” is a transforming algorithm for English language that reduces any of the forms of a word such as "walks, walking, walked", to its elemental root e.g., "walk". There are different types of stemming strategies. The ones supported by Lucene/Solr all use stemming by reduction, and must thus be applied on both index and query time.

Steps

1. **Clear settings from previous lab before continuing.
2. Set the “q” field to “start_station_name_stem:(Green AND Granging)”
3. Select the “Execute Query” button at the bottom of the page. Why do we find data for Green when there are no records with the term “Green” and the term “Granding”



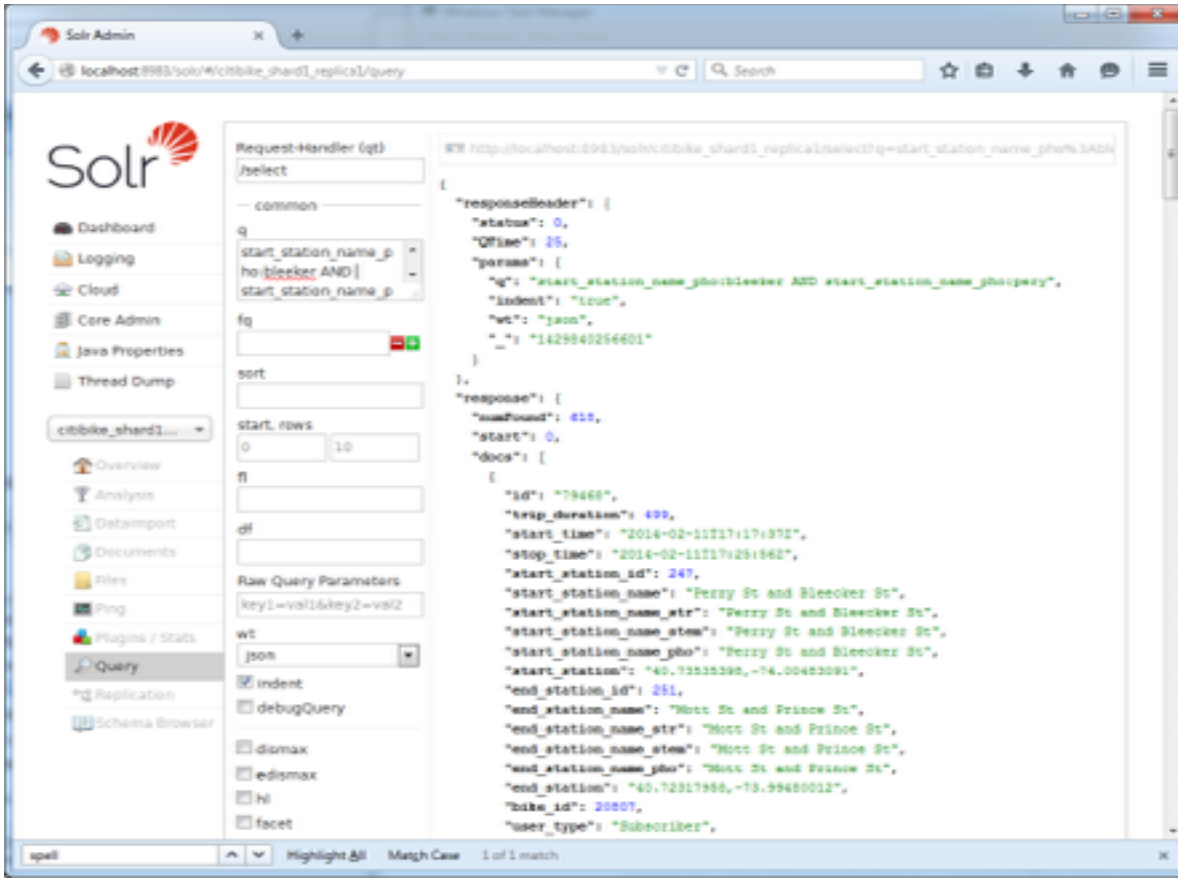
Of course, in this case, the field names contain proper nouns and we do not want to use stemming (Greene matches green and Grand matches Granding). We are using this example to show how this feature can be accomplished in Solr. In the immediately following section of the course, we will discuss field types in more detail and explain how you can configure field analysis chains to do stemming.

2.5. Task 5 – Using Phonetics in your search

The technique of using “phonetics” in your search uses a common codec to generate phonetically similar tokens. So words that sound similar will match during a search. Once again we will simply illustrate how the search works right here and explain more details in the next section of the course.

Steps

1. **Clear settings from previous lab before continuing.
2. Set the “q” field to start_station_name_pho:Bleeker AND start_station_name_pho:Pery.
3. Select the “Execute Query” button at the bottom of the page.



Why do we find data for Bleeker when there are no records with the term “Bleeker” and the term “Pery”?

3. Lab 3 – Solr Configuration Basics

30 minutes

Solr is configured through XML or plain text files. In this exercise, we are going to focus on modifying the schema of a collection within Solr. In particular, we are going to review the following files:

- A schema file that provides definitions to how data will be indexed and searched.
- A Solr configuration file that configures a collection and provides definitions to the REST services that Solr provides.

Goals

At the end of this lab you will be able to:

1. Identify the fundamental elements within the managed-schema and solrconfig.xml file.

3.1. Task 1 Review the Solr Schema File

The Solr schema file provides definition to the fields in which data will indexed and/or stored. Each field has a type, and field types are also defined in this file. Any type of field can be created, but Solr has a set of default types you can start with. Some of the default types that are readily available in a typical Solr installation are:

a. String
b. Boolean
c. Date objects
d. Numbers such as integer, float, long and double.
e. Binary objects
f. Locations: geo-spatial lat/long coordinates.
g. Trie versions of fields that can be used for efficient range searches
h. etc…

When a field is defined there are attributes other than its type that you can specify. Changing these specifications will affect whether you can search or retrieve the data. Some of the more important attributes that are definable are:

a. Stored → will force Solr to store the content in the index so it can be retrieved later.
b. Required → will ensure that a value is included with the document. Solr will throw an error if an indexed document does not contain this field.
c. MultiValued → allows for multiple values to be stored with each field.
d. Indexed → ensures that an inverted index is build from the data within the field. This ensures that the field can be searched.

When a field is defined a "chain" of transformations can be defined for both the "index" and "query" states of the data. For example we might want to remove certain words from the data before it is stored. There are many transformations that you may want to apply to your field such as:

a. Tokenizer → Extracts the "terms" from the input.
b. Stop → Remove words in a "list" file
c. Synonym → Add words based on a "list" file
d. Lower Case → Normalize text to lower case

Finally, because fields with the Solr server are specialized to a specific purpose, using additional fields with alternate types is a normal part of Solr. The schema file provides a "copy field" directive that will

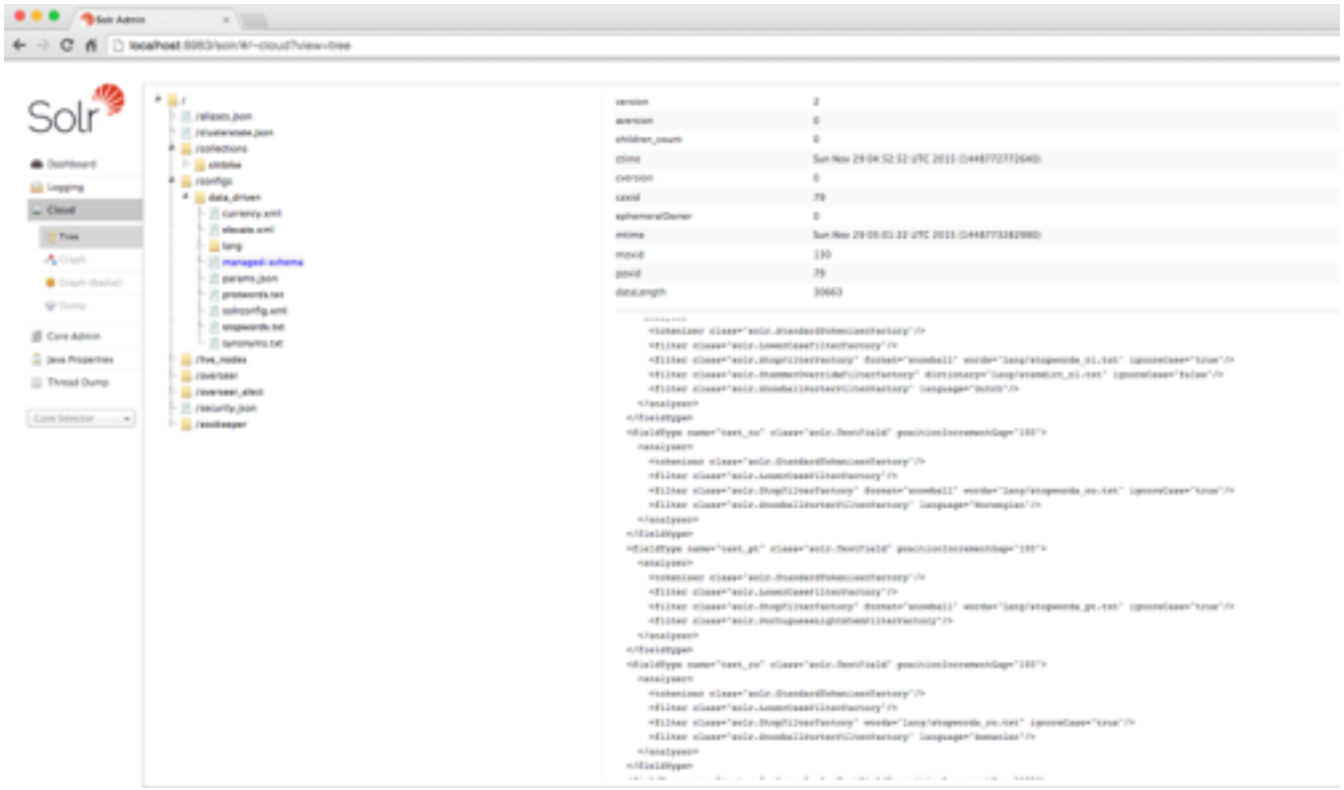
automatically copy data from one field to one or more other fields when the data is inserted into Solr. Typically, you would use “copy field” for:

a. Copying all the text from the document into a single field for searching.

b. Storing data in different fields with different field type definitions so that the same data can be used for different purposes within the application—faceting vs. searching for example.

Steps

1. Open the admin user interface at http://localhost:8983/solr/
2. Go to “Cloud” -> “Tree” -> “/configs/data_driven”
3. Select the managed-schema file.



4. Locate and review the “text_general” field type definition.

```
<!-- A general text field that has reasonable, generic
     cross-language defaults: it tokenizes with StandardTokenizer,
     removes stop words from case-insensitive "stopwords.txt"
     (empty by default), and down cases.  At query time only, it
     also applies synonyms. -->
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Notice the two different “analyzer” sections within the field type definition, one for indexing and the other for querying. These sections define the list of “transformations” that will be applied to the data as it is inserted into Solr (indexed) or selected from Solr (query).

In this case, the default out-of-the-box text_general field goes through a tokenizer, a stop word filter and a lower case filter during indexing.

5. Similarly, locate and review the definition for the field type “string”

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

In this case, there is no change made to field before it is indexed and/or stored.

6. Locate and review the field definitions for “start_station_name” and “start_station_name_str”

```
<field indexed="true" multiValued="false" name="start_station_name_str" required="false" stored="true" type="string"/>
<field name="start_station_name" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true" stored="true" termVectors="false" indexed="true" multiValued="false"/>
```

7. Locate and review the “start_station_name” copyField directive.

```
<copyField source="start_station_name" dest="start_station_name_str"/>
```

It is a common practice to store the same field into different field types in this manner. This difference in analysis chains explains why the query "start_station_name:clinton" and the "start_station_name_str:clinton" produce different results. In the next lab, we shall use these different fields within an application.

8. Review the configuration of the stemming services by looking at the field definition for start_station_name_stem which is defined as field type text_en. Notice that the Porter stemming factory analyzes the field.

9. Similarly, review the configuration of the phonetic services by by looking at the field definition for start_station_name_pho which is defined as field type phonetic_en. Notice the Phonetic Filter Factory used.

Since these fields were analyzed differently we were able to make stemmed and phonetic queries against them respectively.

Challenge

1. Review other phonetic and stemming factories that are available for Solr.

3.2. Task 2 – Review the Solr Configuration File

The Solr configuration file for each collection specifies the available endpoints or request handlers. It also provides pre-defined parameters for each endpoint.

Each request handler defined in the configuration file has a context on the Solr server. Some of the default handlers that are available are:

1. /admin/ping → Health Check Service
2. /select → Search interface to support querying
3. /update → Interface to support insert, editing and deleting data
4. /admin/collections → Tools to configure and manage collections.

By default the Solr configuration file provides all the necessary handlers and components that a basic Solr installation might need. At this point in the course, we want you to be aware of this file and the concepts of endpoints and request handlers.

Search components are granular features within a search request handler. Some of the default search components that are available are:

1. spellcheck → Exposes advanced support for spell checking
2. suggest → More like this support
3. term → Retrieve term vector information inside the query
4. highlight → Highlighting of searched text
5. debug → Detailed query information.

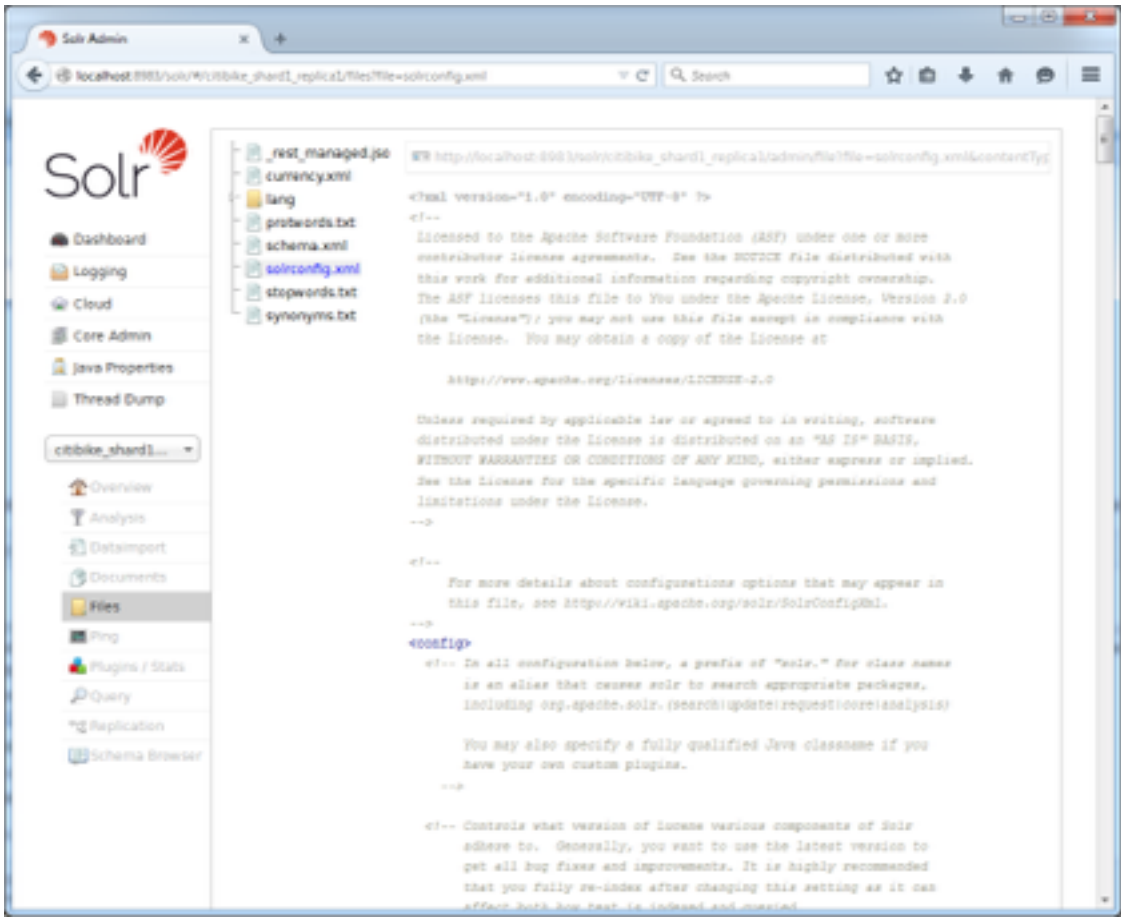
Goals

At the end of this lab you will be able to:

1. Identify the fundamental elements within the solrconfig.xml file.

Steps

1. Open the admin user interface at http://localhost:8983/solr/
2. Select the “citibike” core from the “core selector”.
3. Select the “Files” option below the citibike in the core selector.
4. Select the solrconfig.xml file.



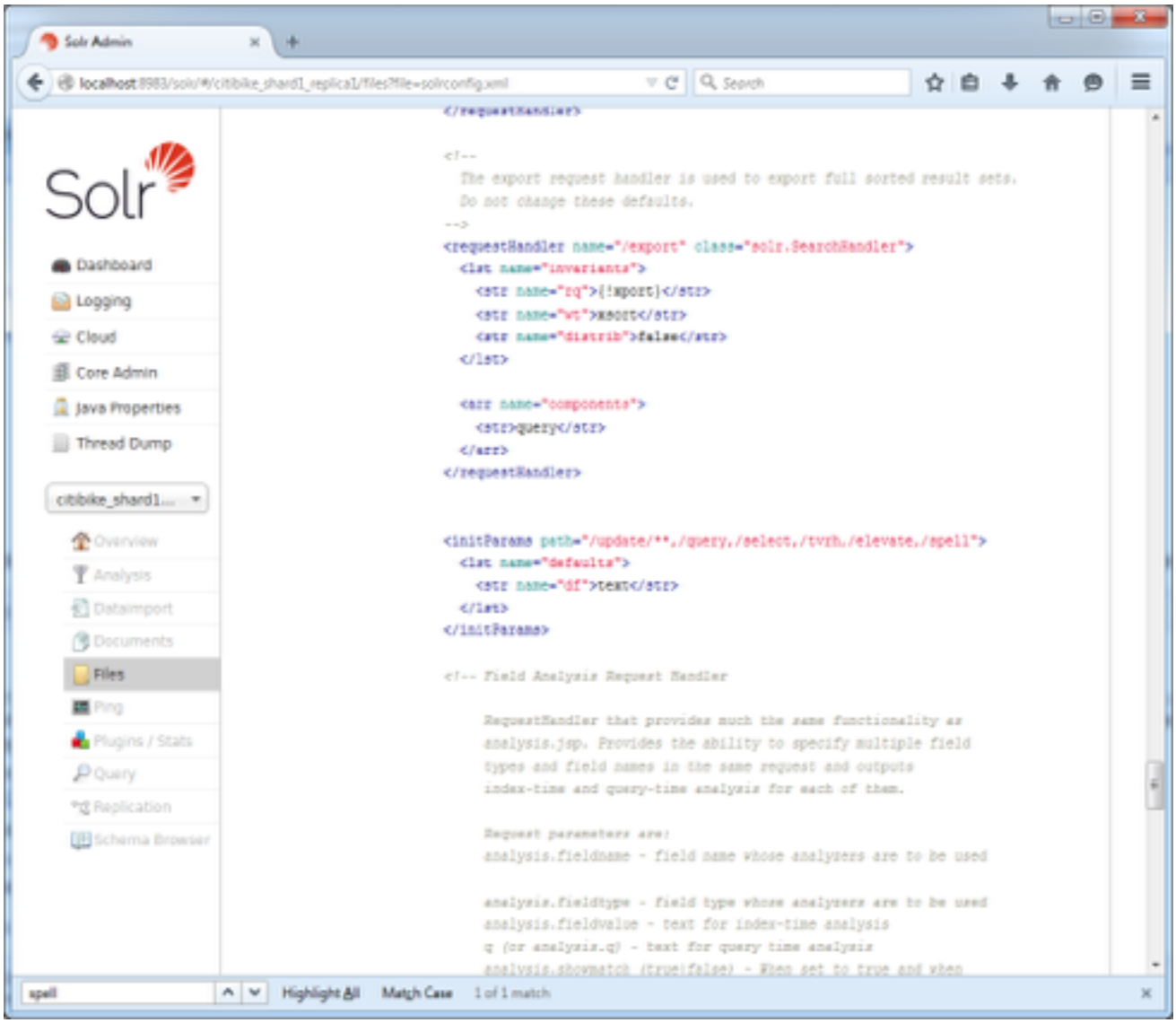
5. Locate and review the “select” request handler.

```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
       will be overridden by parameters in the request
    -->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
    <!-- <str name="df">text</str> -->
  </lst>
```

6. Locate and review the “terms” request handler.

```
<!-- A request handler for demonstrating the terms component -->
<requestHandler name="/terms" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <bool name="terms">true</bool>
    <bool name="distrib">false</bool>
  </lst>
  <arr name="components">
    <str>terms</str>
  </arr>
</requestHandler>
```

7. Locate and review the “initParams” directive.



4. Lab 4 – Visualizing Solr Data

90 minutes

We shall use the open source package banana as the UI for the search application. Banana provides a configurable dashboard framework, enabling users to build complex visualizations. It is written in Javascript and HTML and uses the Angular framework

Goals

- Build complex visualizations using data stored in Solr.
- Interact with the Solr data in interesting ways.
- Understand the Solr queries used to bring the data.

Steps

1. Unzip banana.zip from the training material folder and copy the directory to $installDir/solr-$version/server/solr-webapp/webapp. Now you can access the dashboard at: http://localhost:8983/solr/banana/src/index.html. We suggest using Firefox or Chrome to access the Banana UI.

The Banana Dashboard is configurable. A dashboard can consist of multiple rows and each row can contain multiple panels. Each dashboard may be represented as a JSON object that defines the rows, panels and panel settings within that particular dashboard.

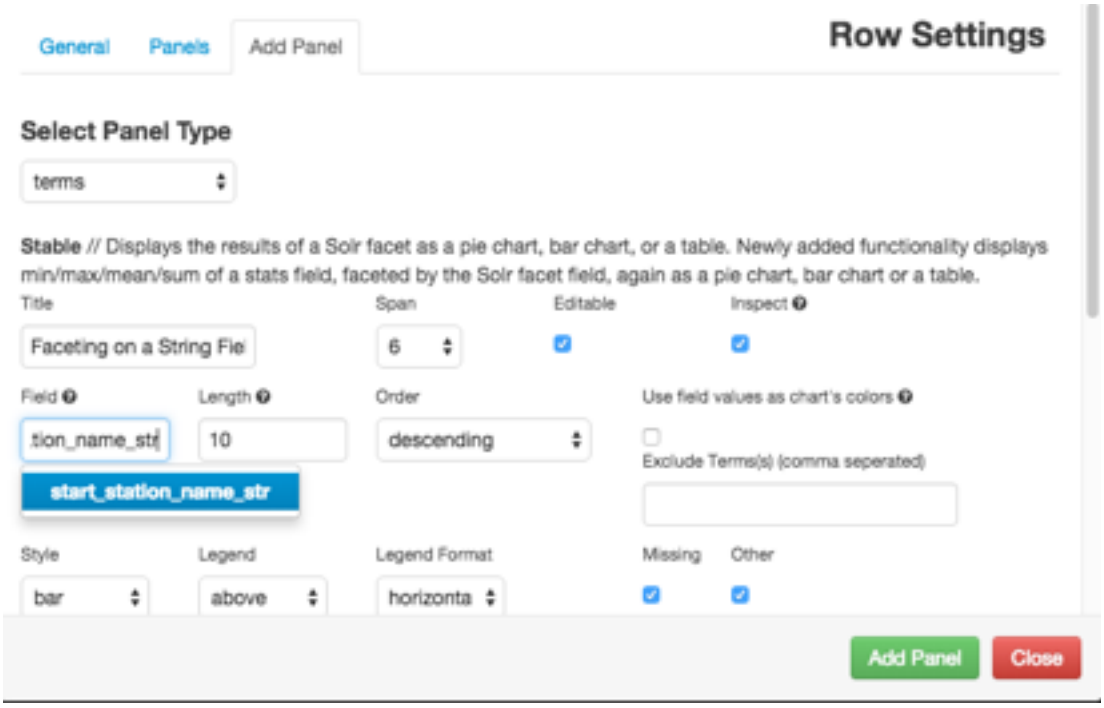
4.1. Task 1 – Basic Faceting

Solr faceting is the arrangement of search results into categories based on indexed terms. Searchers are presented with the indexed terms, along with numerical counts of how many matching documents were found in each term. Faceting makes it easy for users to explore search results, narrowing in on exactly the results they are looking for.

In this exercise, we shall facet by Start Station Names and Gender and illustrate some key principles when faceting with Solr. Banana provides a panel called "Terms" which enables us to build visualizations (like pie and bar charts) using facet counts.

Steps

1. Open the dashboard and make sure that the search panel has the query "*:*" and that the time picker is set to 01/31/2014 00:00:00 to 03/01/2014 00:00:00.
2. Expand the row that is labeled "Facet".
3. Add new panel by clicking on the gear button and select "Add Panel" tab
4. Set Select Panel Type to "terms"
5. Set the Title to "Faceting on a String Field" Span to 6, and the Field to "start_station_name_str" Leave the other settings at their default for now.



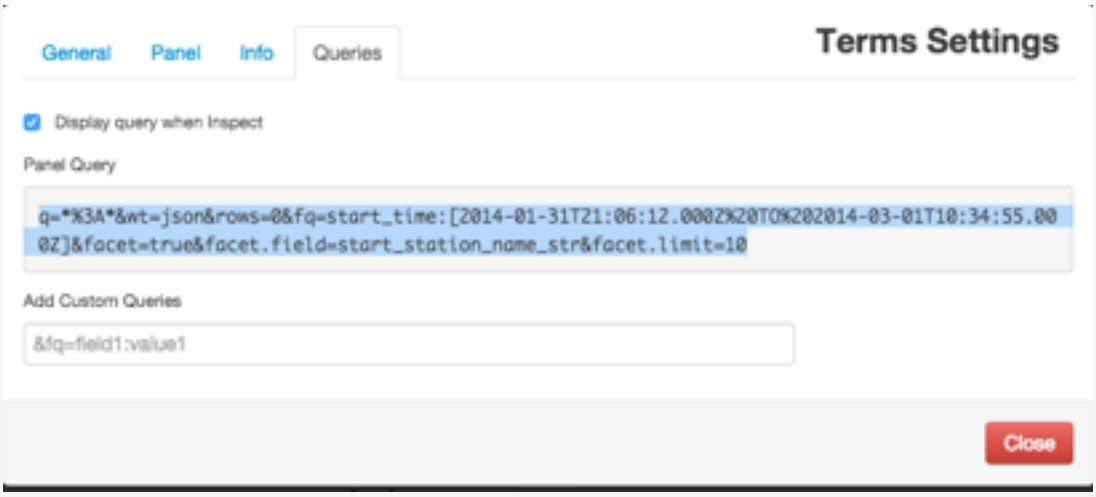
6. Add new panel again, and once select panel type "terms." This time Set the Title to

"Faceting on an Analyzed Text Field" Span to 6, and the Field to "start_station_name"

7. Review the differences between the terms depicted in the two charts. In the analyzed field, the most frequent terms are "and"; "st" (lower cased and stemmed version of St); and "av". This is not very useful for understanding which stations are most popular. On the other hand, when we facet on the string field, we get the most popular start stations, which in this case is "Lafayette St and E 8 St".

   Remember while searching the situation was reversed, the analyzed field provided results when searching for terms like "clinton" while the string field did not. This is a common reason to index/store the same data into two different field types, one for searching and one for faceting and display.

8. Click on the *Inspect* ("i") button on the panel. If the query does not appear, click on the *Configure* (gear) icon, and click on the *Queries* tab. You will see the query that has been constructed in order to get the facets.



   Banana is a single page application (like Google Maps), and changes made in one panel are typically reflected across all panels in the dashboard. In this case, the specific query required to build the panel uses parameters from the timepicker (which is put into the filter query) and the search bar.

   Similarly, look at the query used to build the other terms panel. It is identical except for the different field used for faceting.

```
q=*%3A*&wt=json&rows=0&fq=start_time:[2014-01-31T21:06:12.000Z
%20TO
%202014-03-01T10:34:55.000Z]&facet=true&facet.field=start_sta-
tion_name&facet.limit=10
```

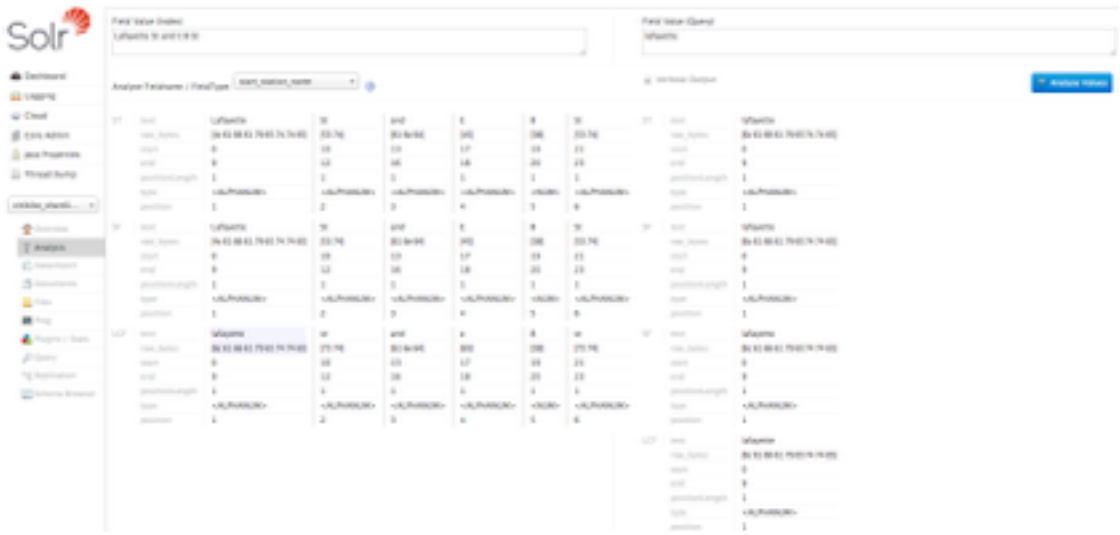
9. Now click on the bar representing “Lafayette St and E 8 St.” in the panel that is faceting on the string field. The dashboard changes because we are now filtering on just this one start_station. This is a common way to narrow down search results. Right now there is not a lot on the dashboard so this is not very interesting, but that will change very soon. However, for now, look at the queries again.

```
q=*%3A*&wt=json&rows=0&fq=start_time:[2014-01-31T21:06:12.000Z%20TO
%202014-03-01T10:34:55.000Z]&fq=start_station_name_str:"Lafayette%20St
%20and%20E
%208%20St"&facet=true&facet.field=start_station_name_str&facet.limit=10
```

Notice the new filter query (fq) that has been added to the query in order to filter the results. We shall use this more later in this lab.

Advanced

10. Finally, to wrap up our discussions of the various field types, we will show you an interesting tool that is part of the Solr Admin UI. Once again select the “citibike_shard1_replica1” Core and now click on Analysis. Select the FieldName/FieldType to “start_station_name.” Enter the term “Lafayette St and E 8 St” in the “Field Value (Index)” and “lafayette” in the “Field Value (Query)”



We see how both the terms are modified as they go through the analysis chain. On the indexing side, the Station Name is lowercased and tokenized as it is on the query side. Thus a search for “lafayette” on this field will match “Lafayette St and E 8 St” among other stations containing the token “Lafayette.”

Now change the Analyse Fieldname/FieldType value to “start_station_name_str.” What is different?

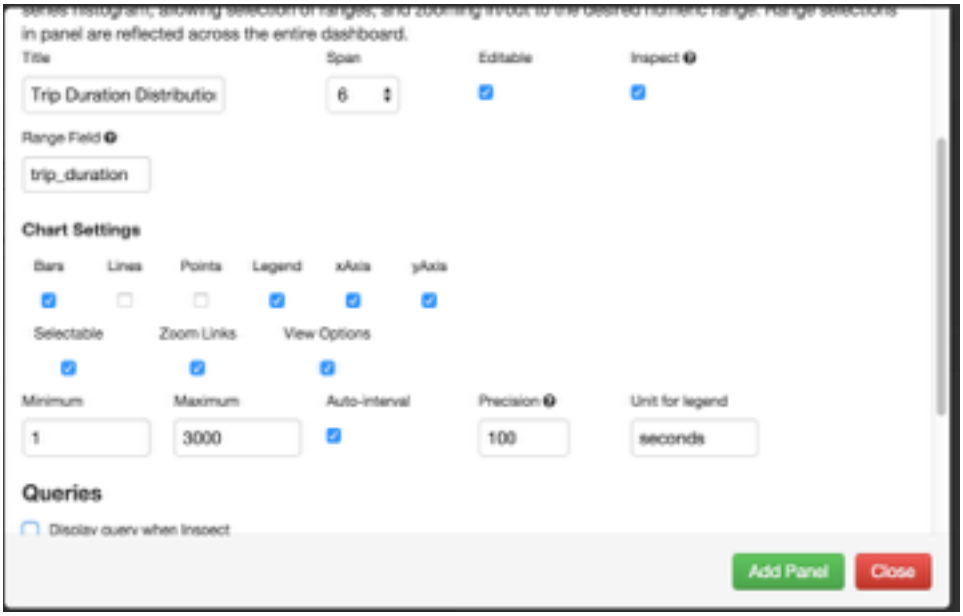
In this case, as a string field, the field is not analyzed but indexed directly as one term. Therefore “lafayette” will not match “Lafayette St and E 8 St.”

4.2. Task 3 – Range Facets

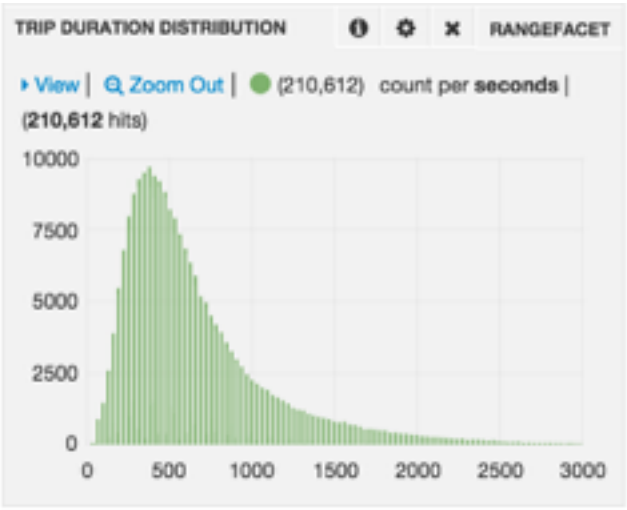
We will next explore Solr's range facets, by looking at an integer field (duration) and a date field (start_-time).

Steps

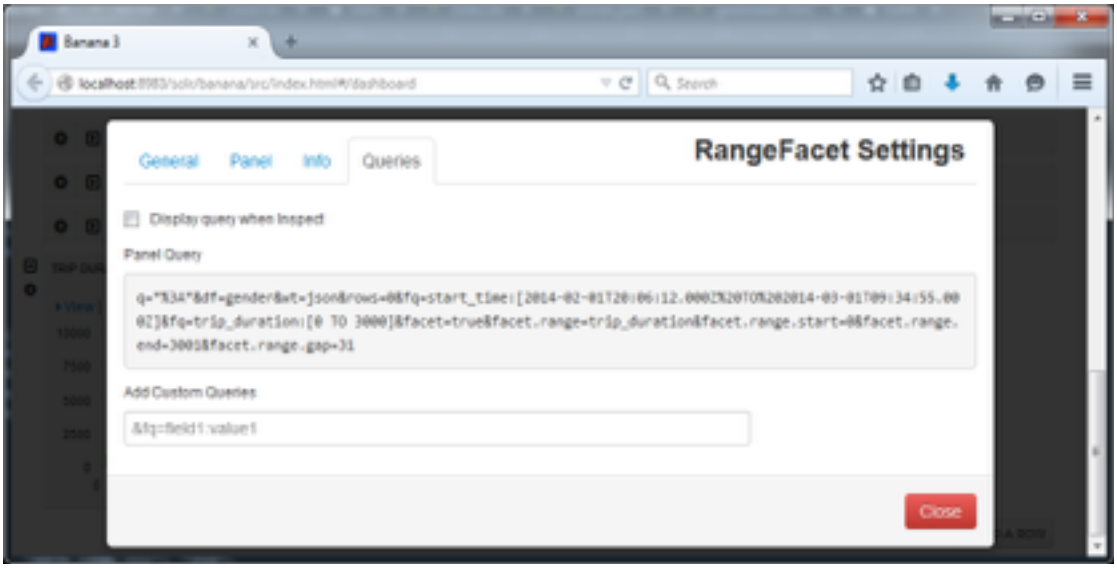
1. Open the empty row entitled "Range Facet"
2. Add a panel to empty row, in this case a panel of type "rangeFacet"
3. Title the panel as "Trip Duration Distribution", set the Range Field to trip_duration, and set Minimum and Maximum to 1 and 3,000 respectively, the Span to 6 and check the box to "Display Query when Inspect." Set the Unit for Legend to seconds.



4. Click on the Add Panel button and close the dialogue window.

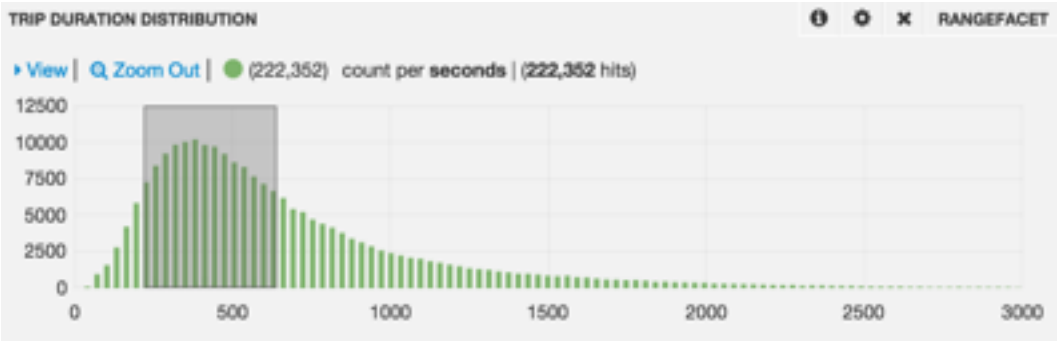


5. For those of you familiar with probability and statistics, you may recognize a classic Poisson distribution. Now inspect the query that produces the data for this visualization. Click on the "gear" icon and click on the "Queries" tab.



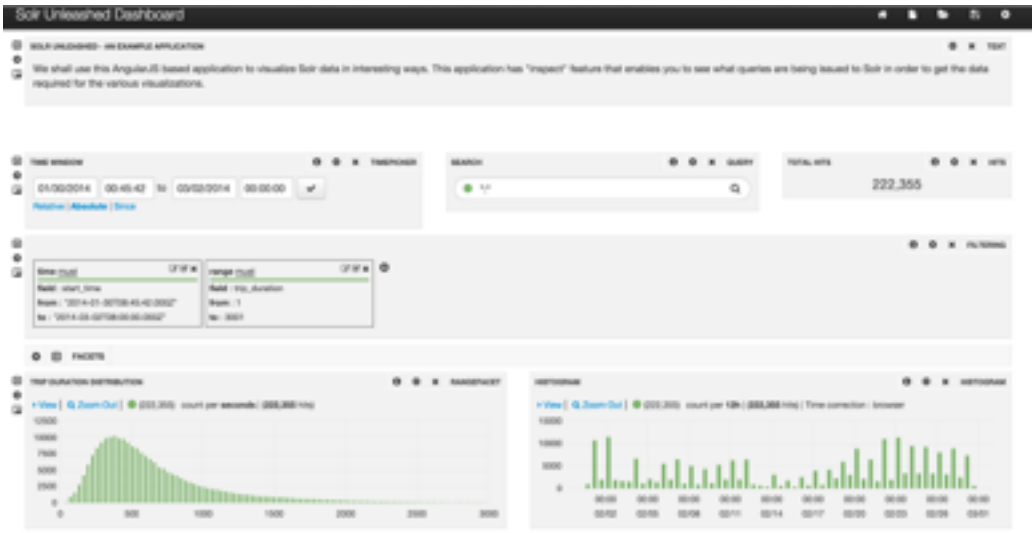
6. Review how the query is constructed.

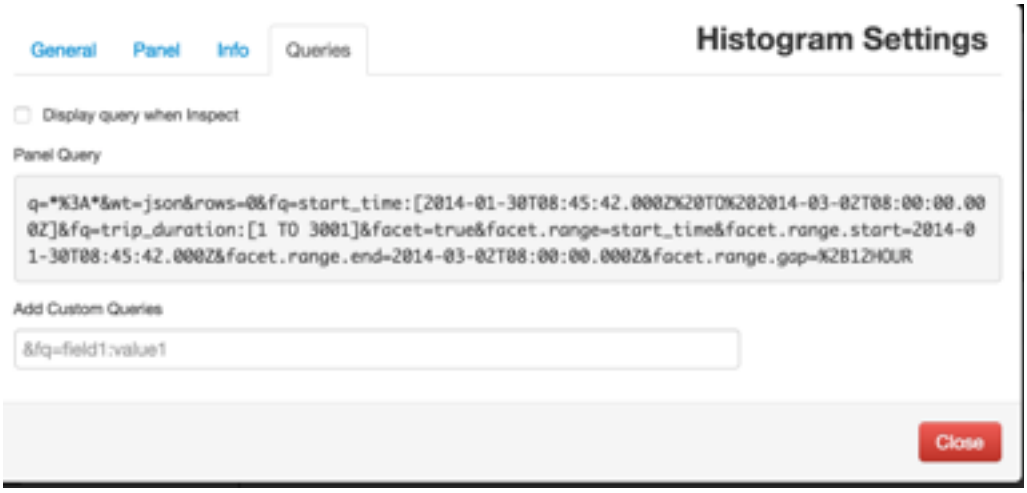
7. Finally, you can zoom in and out of the graph by dragging your mouse over an arrow (to narrow the range) and by clicking on "Zoom Out" to expand the range.



8. Date range Facets work in a similar manner and are visualized using a panel of type "histogram"

9. Add another panel to the row (by clicking on the "+" button) and select panel type "histogram." Give it a title, "Bike Trips over Time," leave the defaults and add the panel. You will see a count of bike trips over time, and once again you can zoom in and zoom out. Note that when you modify this panel the time picker panel is changed (time is treated specially by the Banana UI, which was originally built to handle time series data). Once again Inspect the Query and see how the query is constructed.





Optional

1. Add another panel to this row to visualize the distribution of birth years (Hint: the total span of a row is 12, so you may need to reduce the spans of the existing panels to 4 each before adding another panel to this row.

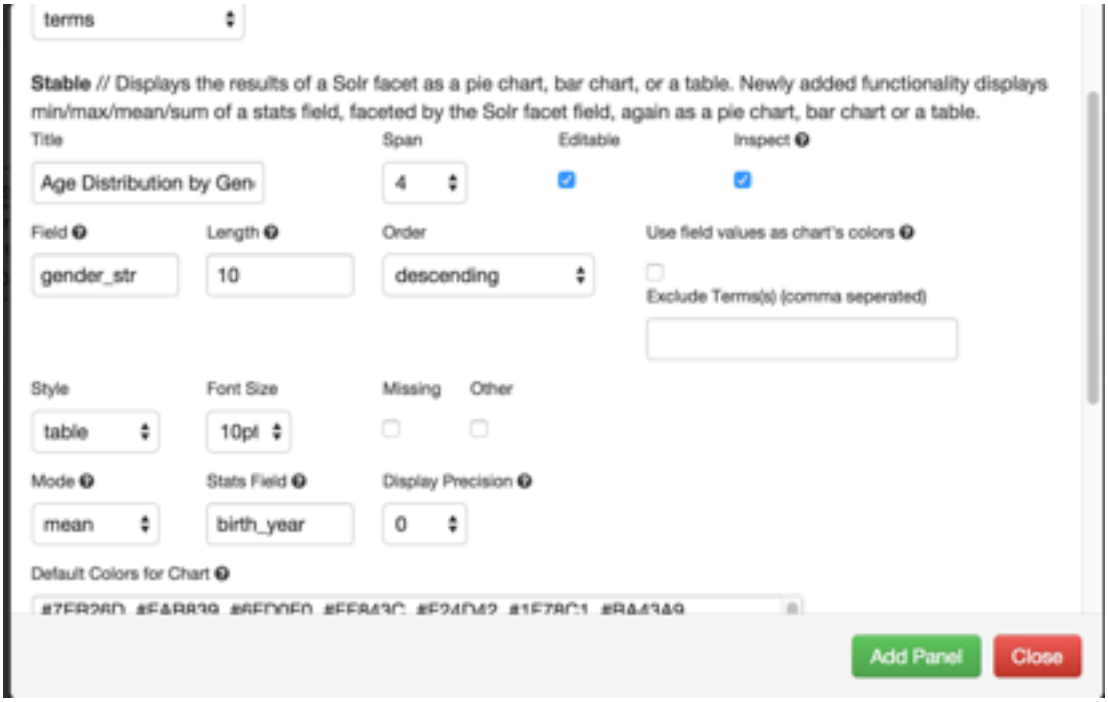
4.3. Task 3 – **Statistical Facets**

Solr's stats component returns simple statistics for indexed numeric fields within the Document Set. We shall use this feature to explore this data in interesting ways. Our goal is to understand things like:

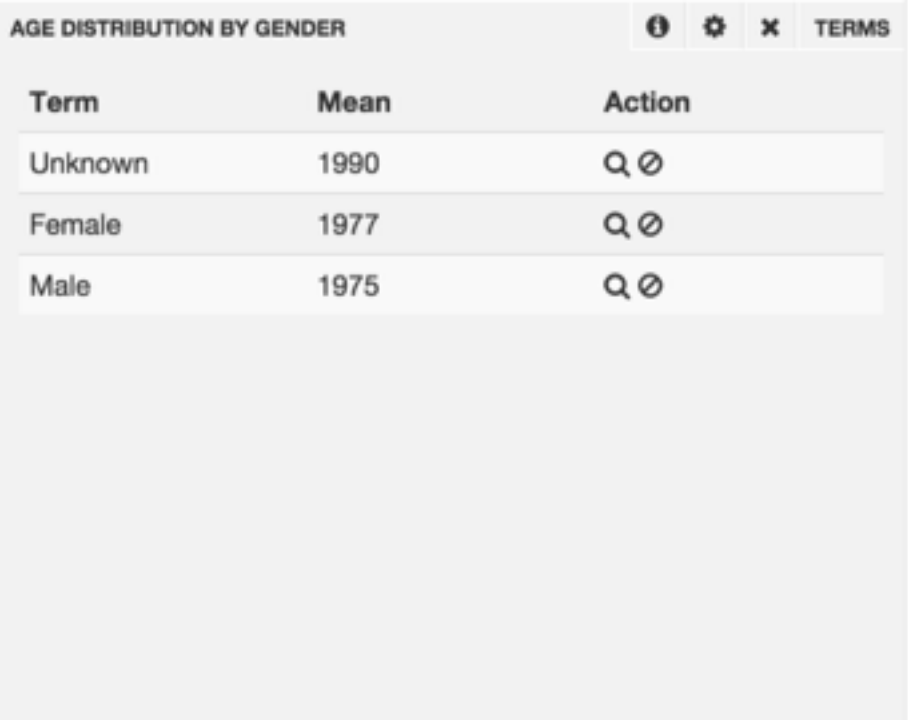
1. What is average age of riders, calculated by gender?
2. Are subscribers more likely to take longer trips than one-time customers?

Steps

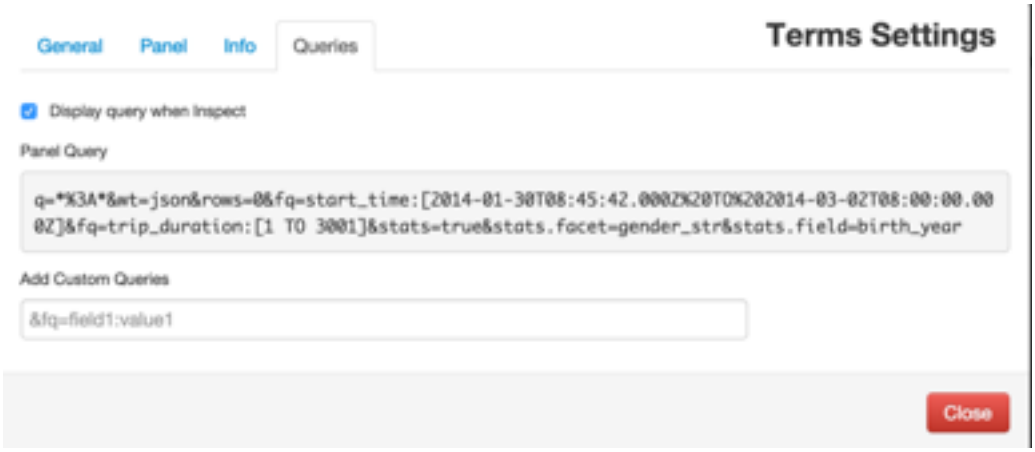
1. Open the row called "Statistical Facets." Click on Add panel to empty row and select add a new "terms" with the following settings.



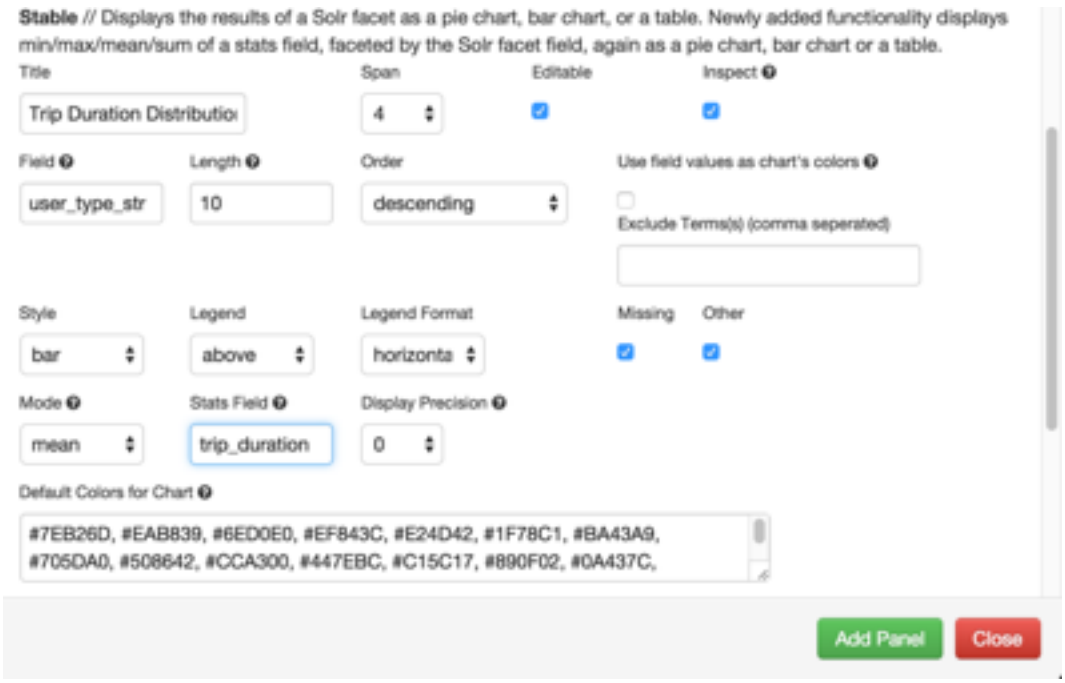
Note that we set the title to "Age Distribution by Gender," the span to 4, the Style as Table and the Mode as mean. When the Mode is selected as "mean" a text box appears to enter the stats field in which we enter the birth year. When you click on Add Panel, you should see:

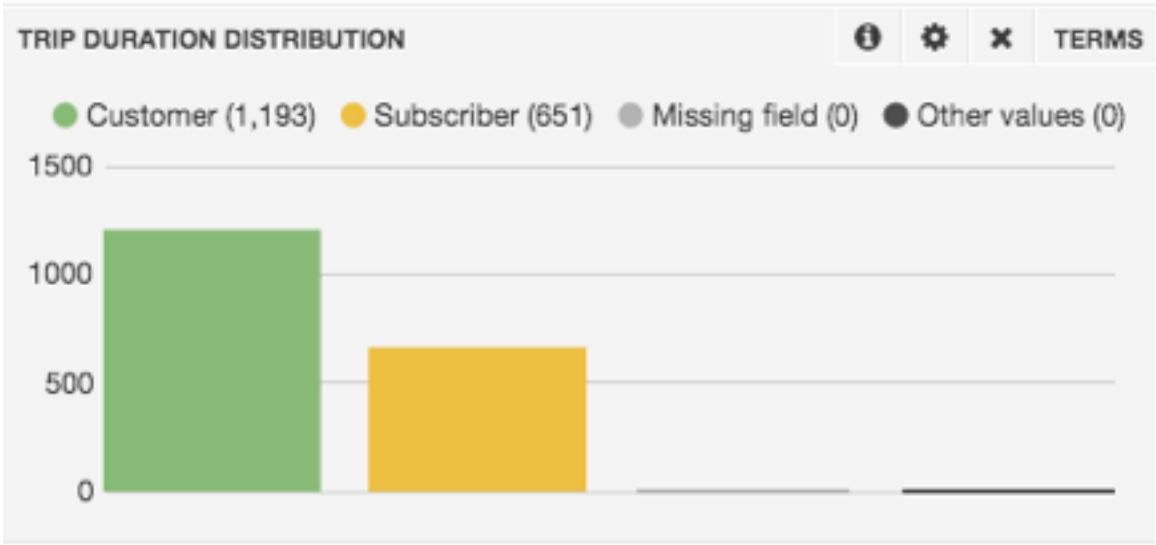


2. Once again inspect the query to see how the stats facet is retrieved from Solr (click on the gear icon and select the queries tab). Note how the filter queries from the other panels still remain. You can remove some or all of these filter queries from the "Filter" panel. Go ahead and delete the filter on trip_duration.



3. Now we will add another panel to look at average trip duration distributions, comparing one-time users to subscribers. Add another terms panel with the following parameters:

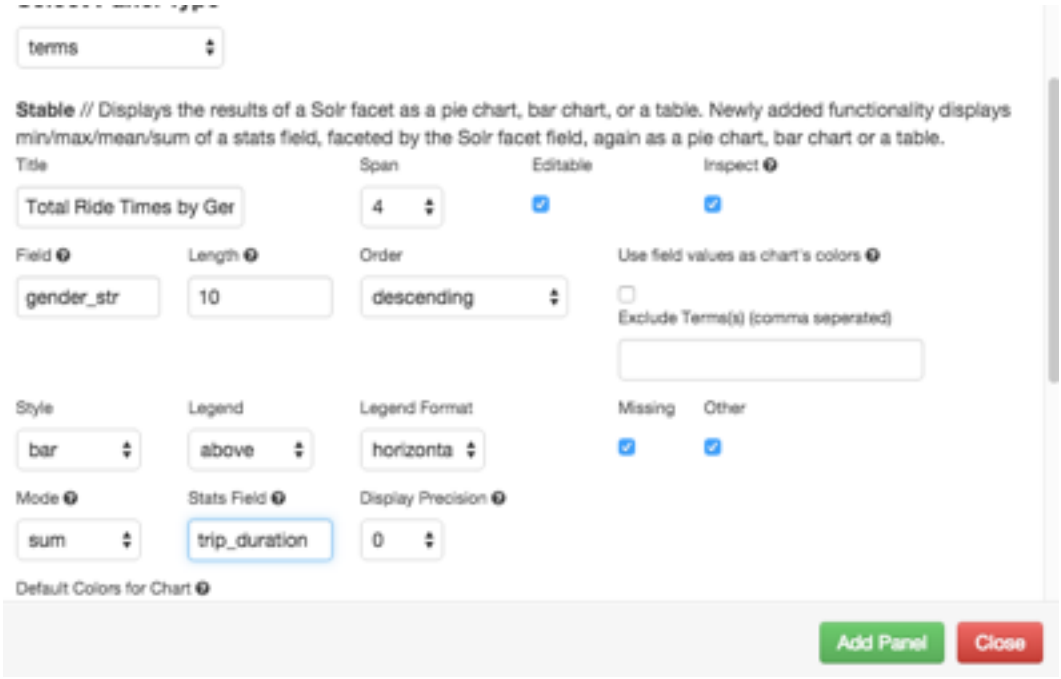


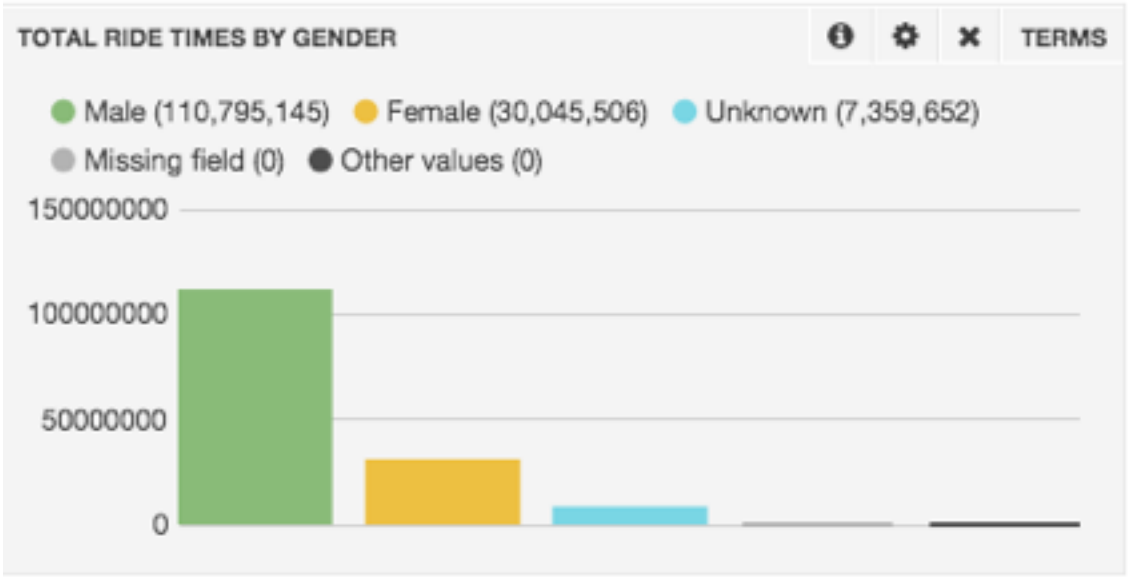


We see here that one-time customers tend to take longer rides, perhaps because they are tourists and take more leisurely rides.

*Note also that the actual numbers you get may vary based on the time windows and trip duration range that you may have selected in the previous exercise.

4. Lastly, we want to provide one of our advertisers a report on how long different genders are exposed to their advertisement, an advertisement that appears on the handlebar of every bike. To setup this report, we add a new *Terms* panel and configure it as follows:



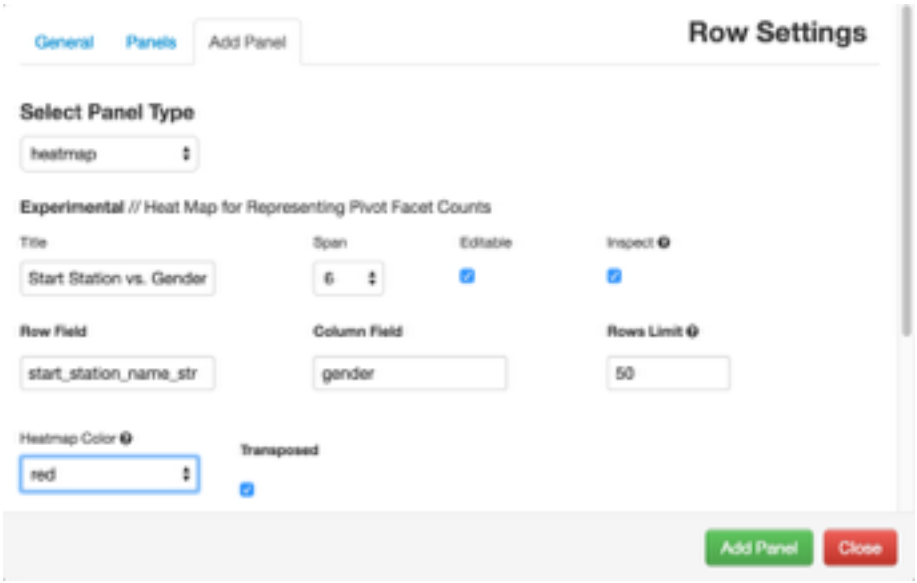


Reviewing this report, it is clear that Males spend more time on the bikes are will be exposed more to an ad on the handlebar. By using the time picker or the histogram panel, we can modify the time window on this report.

4.4. Task 4 – Pivot Facets

Continuing on the targeted advertising theme, we now turn to billboard at the bike docking stations. We want to get a report of which stations are most popular, arranged by gender. We would like to represent this visually once again in a dashboard that can be used by the marketing and advertising manager.

1. Banana provides a powerful visualization of Pivot Facets as heatmaps. To use this, let us open the row entitled “Pivot Facets” and add a heatmap panel with the following parameters (make sure you use start_station_name_str for the Row Field):



Notice how this heatmap provides you with the most popular stations sorted by gender. Which station is most popular among men? Go ahead and inspect the query to see how the Solr query is constructed.

2. Suppose you are only interested in advertising to women. How would you get a heatmap that only showed the most popular stations for women? (HINT: you can set queries and filter queries in other parts of the UI. One way to do this is to issue a search gender:female

Optional

Create another heatmap correlating start stations and end stations (make sure to use the string fields). Keep the rows limit to 50 (do not do this if your machine does not have sufficient memory)

5. Lab 5 – Preparing Solr For New Data

30 minutes

To demonstrate the advanced Solr Query features, we will create a new collection. Using the built in scripts we will create a new Solr configuration set in zookeeper and a new collection that will be loaded with data. The new collection will have text data that is conducive to effectively demonstrating the advanced features of Solr query API.

Goals

At the end of this lab you will be able to:

1. Create a new configuration set in zookeeper.
2. Create a new collection that uses the new configuration set.
3. Query the new Solr collection using advanced relevancy or scoring techniques.

5.1. Task 1 - Create New Solr Collection

Create a new collection in the Solr server to provide a new collection for the "books" data using the built in Solr scripts and a web browser. The Solr server and the ZooKeeper are running from the citibike exercises. Before we can create the new collection we need a "configset" to use as the configuration for the new "books" solr collection. Use the ZooKeeper command line interface we will add a new configuration to the ZooKeeper with the name "books".

Steps

Change to the $installdir/server/scripts/cloud-scripts directory and use the zkcli script upload the "books" config set.

On **Unix/Linux/Mac** environments, execute the following command:

```
./zkcli.sh -zkhost localhost:9983 -cmd upconfig -confname books -confdir $/path/to/training-files/books_configs
```

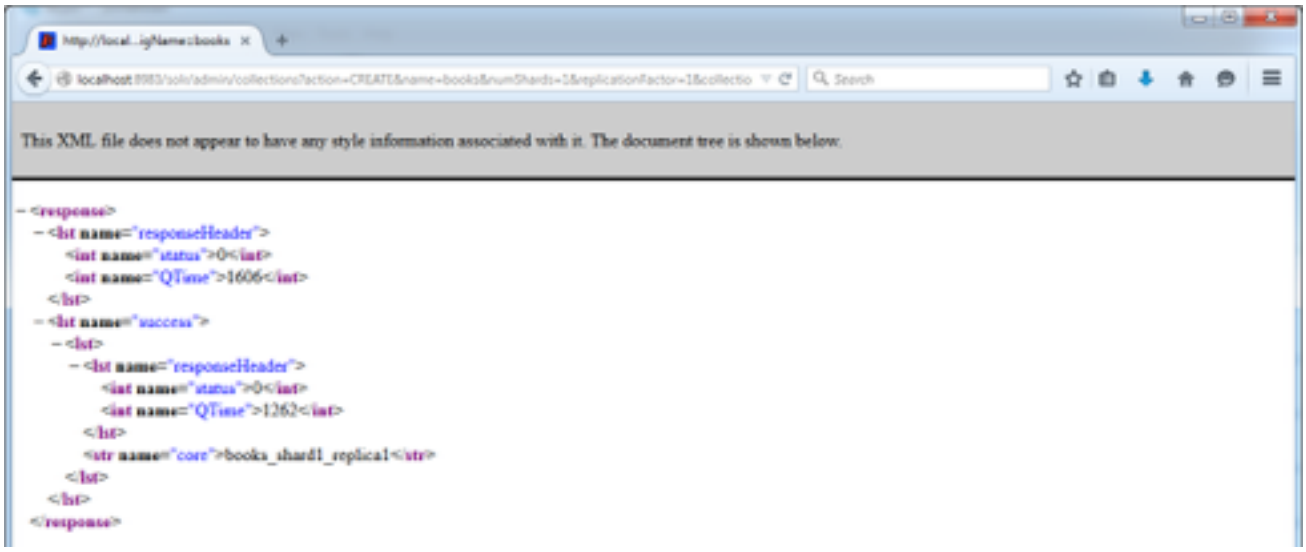
On **Windows** use the command:

```
zkcli.bat -cmd upconfig -confname books -z localhost:9983 -confdir $/path/to/training-
```

files/books_configs

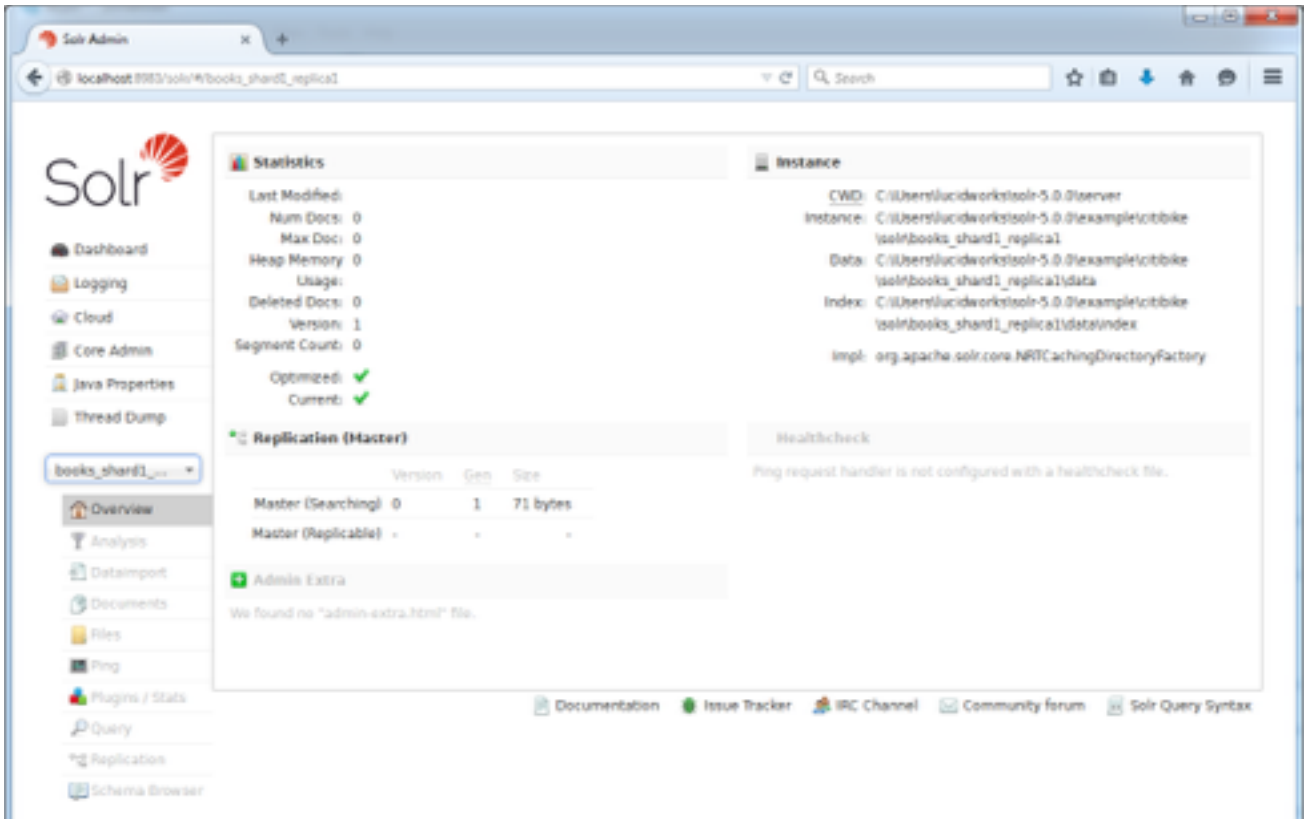
Next we will use the “Collections API” to create a collection called books using the configset “books” that was just added.

http://localhost:8983/solr/admin/collections?action=CREATE&name=books&numShards=1&replicationFactor=1&collection.configName=books



Open the Solr admin console http://localhost:8983/solr/admin

Success, the new collection was created using the collections API.



5.2. Task 2 – Updating Solr Configuration (Optional)

30 minutes

The Solr server that we are using is in "cloud" mode. This means that all "collection" specific configuration is maintained in Zookeeper external to Solr. When we start Solr we used the "-cloud" switch that forced Solr to start up and use the configuration that it extracted from the zookeeper server. The "embedded" zookeeper server was started up automatically when solr started. The "embedded" zookeeper server is on a port number that is the solr server port number plus 1000.

To edit or change the configuration of your collection, you will need to use a tool that will work with the zookeeper server. There are multiple tools that can be used to update a Solr configuration stored in zookeeper, but the simplest is the Command Line Interface (zkcli). In this exercise we will use the zkcli to download a configuration, make minor changes, and then upload the configuration as a new configuration set into the zookeeper server for use with the remaining labs.

Goals

At the end of this lab you will be able to:

1. Pull the collection configuration from zookeeper into a directory.
2. Modify the configuration locally and upload it back to ZK.

Steps

1. Open command window to the location of the pre-installed scripts for the solr installation.
2. Change to $installdir/solr-$version/server/scripts/cloud-scripts.
3. On **Unix/Linux/Mac environments** use the command:

   ```
   ./zkcli.sh -zkhost localhost:9983 -cmd downconfig -confname books -confdir $installdir/solr-$version/current_configs
   ```

4. On **Windows** use the command:

   ```
   zkcli.bat -cmd downconfig -confname books -z localhost:9983 -confdir c:$installdir\solr-$version\current_configs
   ```

5. Inspect the "current_configs" directory that we used as the "confdir" with the cmd "downconfig".
6. Open the solrconfig.xml with a text editor. Add this entry and save the file.

   ```
   <requestHandler name="/admin/ping" class="solr.PingRequestHandler">
   <lst name="invariants">
   <str name="q">solrpingquery</str>
   </lst>
   <lst name="defaults">
   <str name="echoParams">all</str>
   </lst>
   <!-- An optional feature of the PingRequestHandler is to configure the handler with a "healthcheckFile" which can be used to enable/disable the PingRequestHandler. relative paths are resolved against the data dir -->
   <!-- <str name="healthcheckFile">server-enabled.txt</str> -->
   </requestHandler>
   ```

7. Execute the script upload the configuration back into zookeeper.

   On **Unix/Linux/Mac environments** use the command:

   ```
   ./zkcli.sh -zkhost localhost:9983 -cmd upconfig -confname books -confdir $in-
   ```

stalldir/solr-$version/current_configs

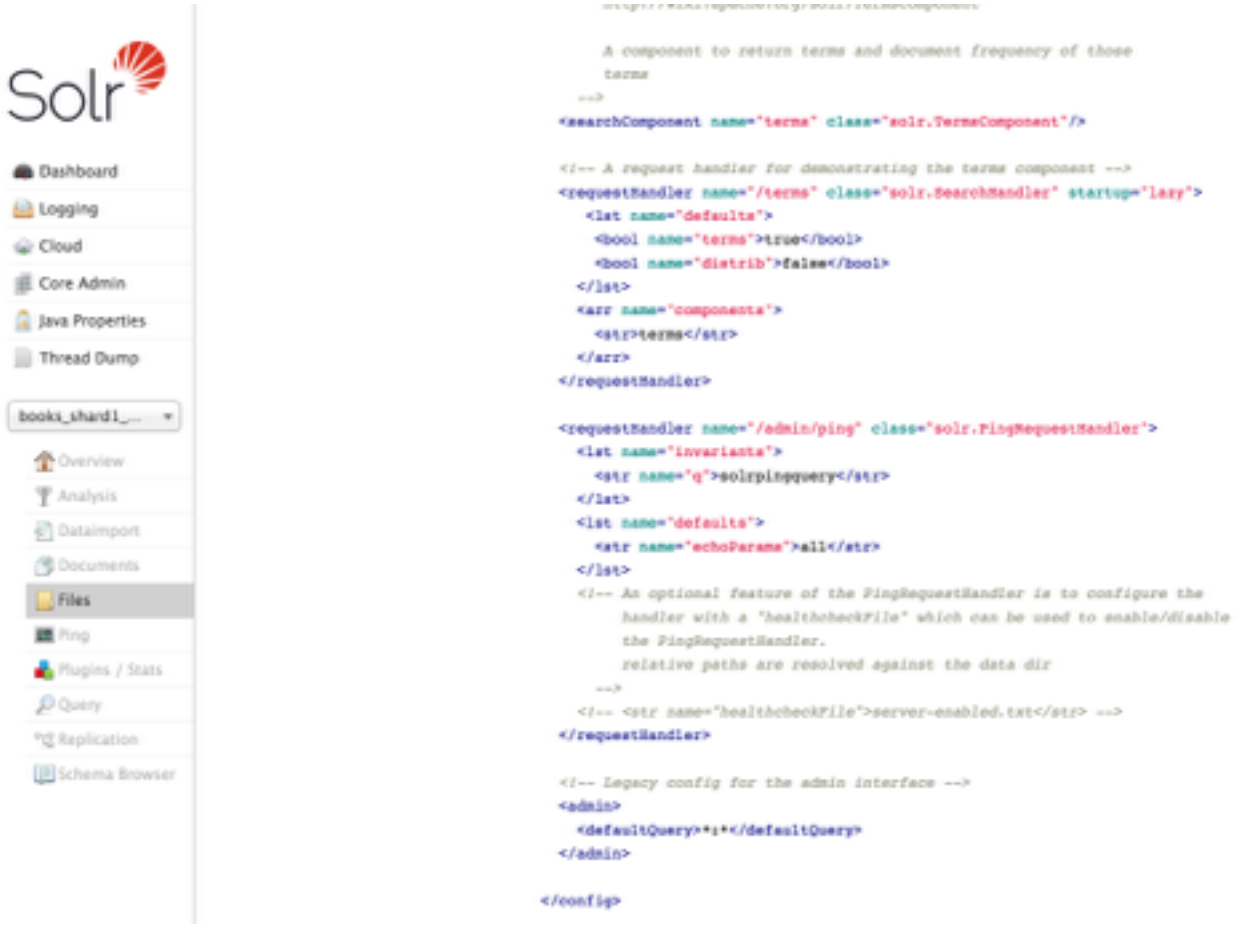
On **Windows** use the command:

zkcli.bat -cmd upconfig -confname books -z localhost:9983 -confdir c:$installdir \solr-$version\current_configs

*Note: Once the changes are made and zookeeper is updated, there is no need to restart our solr instance. The solr instance automatically receives the updates.

Finally, check the changes have been made by opening the solr admin console for that server. Search for the changes we made. http://localhost:8983/solr/admin

8. On successful update, you should see the following in the Solr admin.



9. You can use the Config APIs to directly modify the solrconfig.xml file in ZooKeeper. More documentation here - https://cwiki.apache.org/confluence/display/solr/Config+API

6. Lab 6 – Loading Data Into Solr

15 minutes

Solr provides multiple services for loading data into a collection. In this exercise we will explore the "default" import handler.

Goals

At the end of this lab you will be able to:

- Import CSV data into solr using the default CSV import handler.
- Review how dynamic fields work in Solr

6.1. Task 1 – Using the default import handler

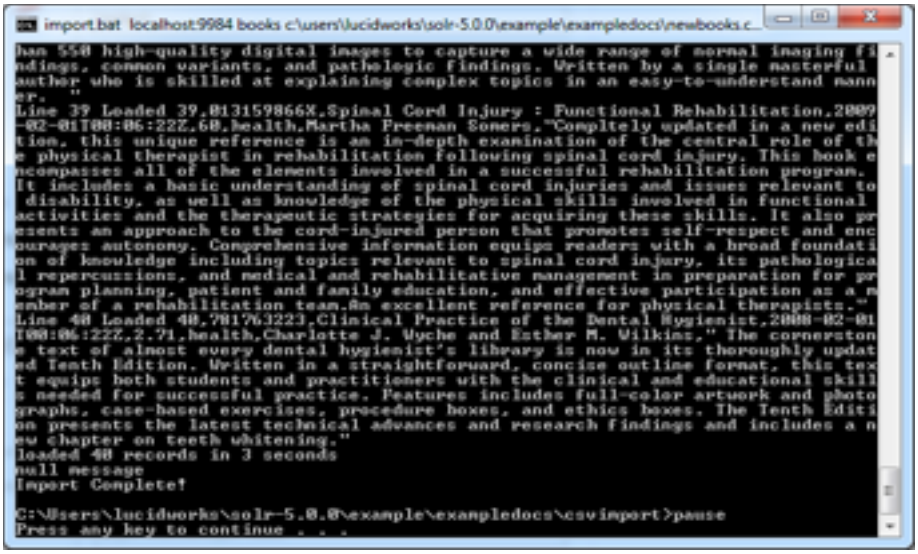
Steps

1. Open a command window and execute the following command:

   ```
   java -Dtype=text/csv -Durl=http://localhost:8983/solr/books/update -jar "$installdir/example/exampledocs/post.jar" "/training-material/newbooks.csv"
   ```

2. The command will report statistics on the result of the import operation.

```
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8984/solr/books/update/csv using content-type text/csv..
.
POSTing file newbooks.csv to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8984/solr/books/update/csv...
Time spent: 0:00:00.446
```



6.2. Task 3 – Review Dynamic Fields

Solr provides the ability to "automatically" create fields in a document. The "dynamic fields" feature allows Solr to index fields that are not explicitly define in the schema. A dynamic field is just like a regular field except it has a name with a wildcard in it. When you are indexing documents, a field that does not match any explicitly defined fields can be matched with a dynamic field.

1. Open the CSV file that contains the book data. Use any text editor or spreadsheet tool.

   The source csv file may be found in $installdir/solr-$version/example/exampledocs/newbooks.csv

2. Locate the "title_t" and "title_s" columns. By inspecting the "schema.xml" configuration file, you can clearly see that there are no "definitions" for these columns. Instead, there are "dynamic field" definitions that use a wild card to match a field name that is being inserted into if the field does not exist in the schema.

	A	B	C	D	E	F	G	H	I	J
1	id	isbn	title	publishdate	price	genre	author	title_t	title_s	summary
2	1	521831431	A First Cours	2004-02-01T	21.4	science	Barton Zwiet	A First Cours	A First Cours	An accessible
3	2	316155799	The Cosmic L	2005-02-01T	4.79	science	Leonard Suss	The Cosmic L	The Cosmic L	Anthropic Pri
4	3	047046724X	String Theor	2009-02-01T	3.42	science	Andrew Zimr	String Theor	String Theor	A clear, plain
5	4	691142890	The Little Bo	2010-02-01T	3.63	science	Steven S. Gu	The Little Bo	The Little Bo	The Little Boo

```
<field name="title" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true"
<field name="isbn" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true" s
<field name="publishdate" type="date" omitNorms="false" omitPositions="true" omitTermFreqAndPositions="true" s
<field name="price" type="float" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true" stored=
<field name="genre" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true"
<field name="author" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true"
<field name="summary" type="text_general" omitNorms="true" omitPositions="true" omitTermFreqAndPositions="true
```

3. Open the admin user interface at http://localhost:8983/solr/admin
4. Select the "books" core in the core selector the left side of the page.
5. Select the "Query" option below the books in the core selector.
6. Enter into the "q" parameter add "*:*".
7. Select the "Execute Query" button.

8. Notice the “dynamic fields” for title_s and title_t. If you explore the Schema Browser tab, you will see that those fields are now a “string” and “text” field respectively.

7. Lab 7 – Advanced Solr Querying

60 minutes

To demonstrate the advanced Solr Query features the “books” collection was created. The new collection will have text data that is conducive to effectively demonstrating the advanced features of Solr query API.

Goals

At the end of this lab you will be able to:

1. Query a Solr collection using advanced relevancy or scoring techniques.

We will be using the Solr Admin UI to construct the queries. Please review the URLs to understand what parameters are being used, and how Solr queries are being constructed.

7.1. Task 1 – Solr Scoring

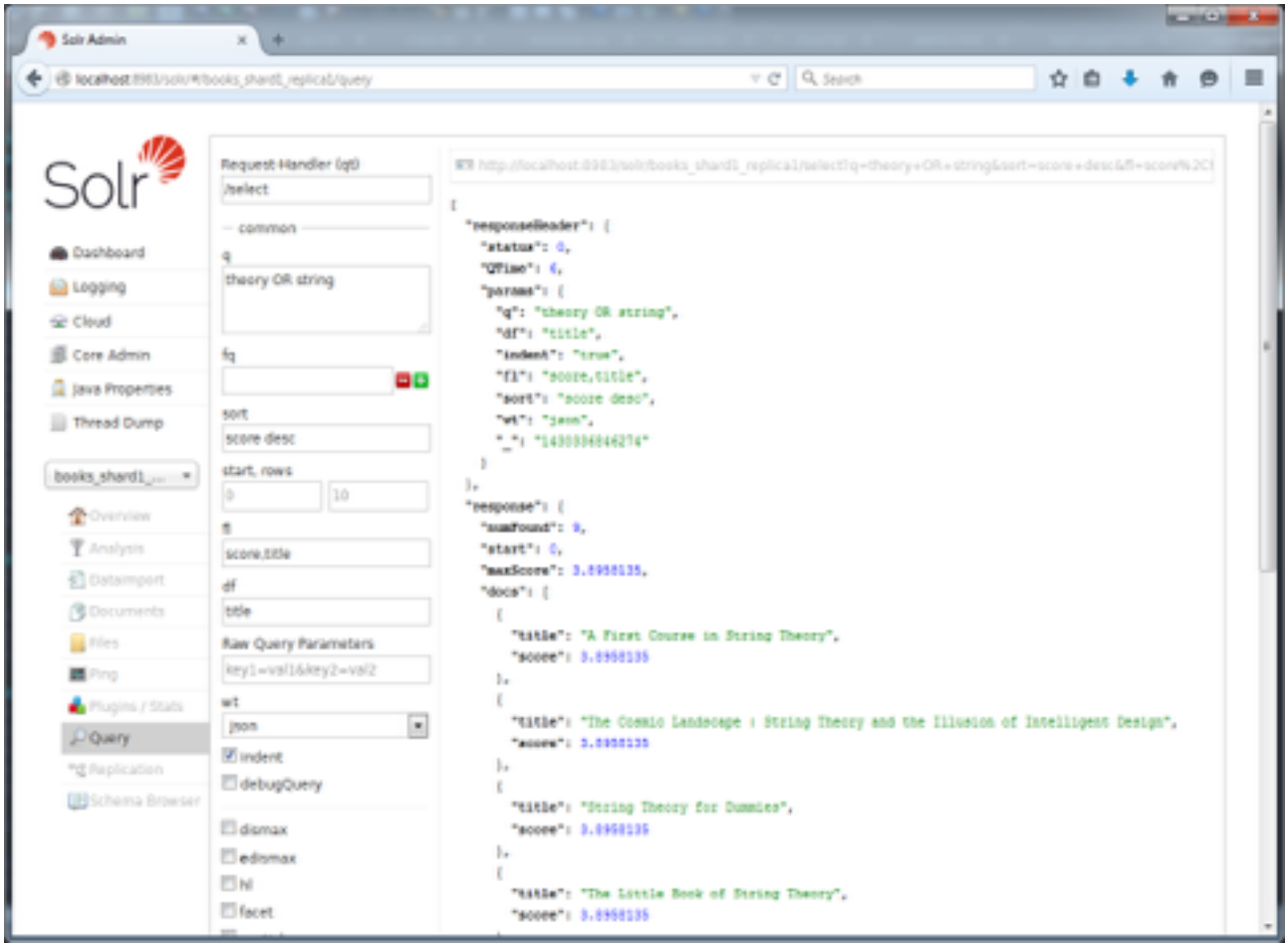
One of the major differences between a search engine and a database is the concept of *relevance*. Solr can "score" documents to provide results that are ordered based on relevance. By default, Solr uses the TF-IDF (Term Frequency Inverse Document Frequency) algorithm to score documents; however it is very flexible and allows you to do many interesting things such as:

1. Make documents more relevant if the search term appears in the title as opposed to the body.
2. Weight more recent documents more than older documents
3. Favor exact matches vs. matches on individual terms, and
4. many variants thereof.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the "books" core in the core selector the left side of the page.
3. Select the "Query" option below the books in the core selector.
4. Enter into the "q" parameter add "theory OR string".
5. Enter into the "sort" parameter add "score desc".
6. Enter into the "fl" parameter add "score, title".
7. Enter into the "df" parameter add title.
8. Select the "Execute Query" button.

1. Notice the list of documents. The documents listed first in the list have a higher score because they have both query terms in the document. The remaining documents have a much lower score because they have only one of the search terms.
2. To better understand how the TFIDF algorithm ordered the results use the "debugQuery" feature.
3. The detailed scoring information shows how the score was derived for each document in the result set.



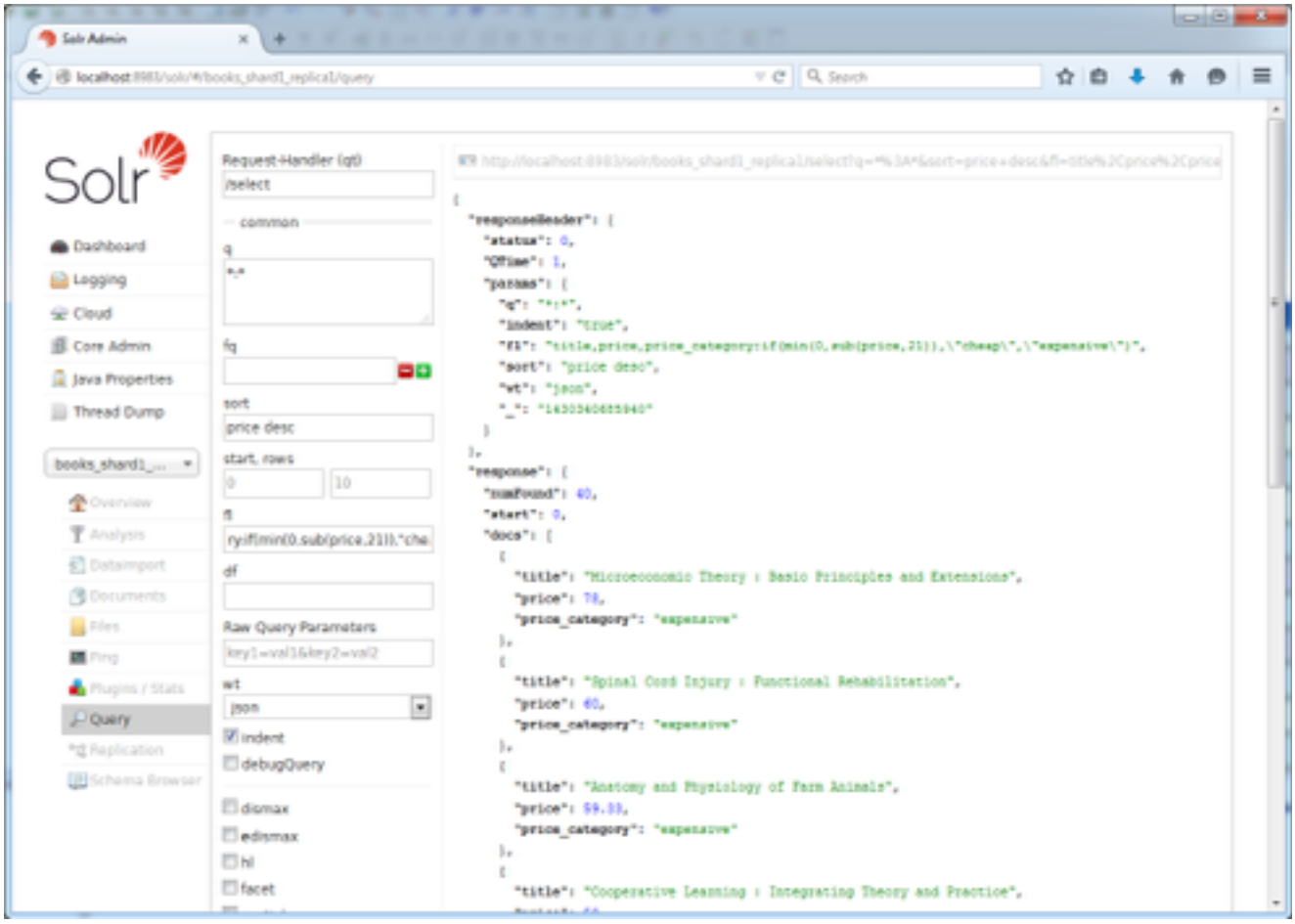

```
"debug": {
  "rawquerystring": "theory OR string",
  "querystring": "theory OR string",
  "parsedquery": "title:theory title:string",
  "parsedquery_toString": "title:theory title:string",
  "explain": {
    "1": "\n3.8958135 = (MATCH) sum of:\n  1.461672 = (MATCH) weight(title:theory in 0) [DefaultSimilarity], result of:\n    1.
    "2": "\n3.8958135 = (MATCH) sum of:\n  1.461672 = (MATCH) weight(title:theory in 1) [DefaultSimilarity], result of:\n    1.
    "3": "\n3.8958135 = (MATCH) sum of:\n  1.461672 = (MATCH) weight(title:theory in 2) [DefaultSimilarity], result of:\n    1.
    "4": "\n3.8958135 = (MATCH) sum of:\n  1.461672 = (MATCH) weight(title:theory in 3) [DefaultSimilarity], result of:\n    1.
    "5": "\n0.730836 = (MATCH) product of:\n  1.461672 = (MATCH) sum of:\n    1.461672 = (MATCH) weight(title:theory in 4) [Def
    "6": "\n0.730836 = (MATCH) product of:\n  1.461672 = (MATCH) sum of:\n    1.461672 = (MATCH) weight(title:theory in 5) [Def
    "7": "\n0.730836 = (MATCH) product of:\n  1.461672 = (MATCH) sum of:\n    1.461672 = (MATCH) weight(title:theory in 6) [Def
    "9": "\n0.730836 = (MATCH) product of:\n  1.461672 = (MATCH) sum of:\n    1.461672 = (MATCH) weight(title:theory in 8) [Def
    "16": "\n0.730836 = (MATCH) product of:\n  1.461672 = (MATCH) sum of:\n    1.461672 = (MATCH) weight(title:theory in 15) [D
  },
```


7.2. Task 2 – Psuedo Fields

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the “books” core in the core selector the left side of the page.
3. Select the “Query” option below the books in the core selector.

4. In the “q” parameter enter “*:*”.
5. In the “sort” parameter enter “price desc”.
6. Enter the following into the “fl” parameter:
    a. title,price,price_category:if(min(0,sub(price,21)),"inexpensive","expensive")
7. Select the “Execute Query” button.



In this example we create a “pseudo field” called “price_category” that is either “inexpensive” or “expensive” based on the price of the book being $21 or above. The “price_category” psuedo field is created using “functions” that help create a logical expression that is evaluted by Solr query parser.

7.3. Task 3 – Term Boosting

The score that solr assigns to each document can be modified by the use of “boosting” on each term in a query. For this example we will use “term boosting” to ensure that documents with the term “string” score higher than documents that have the term “theory”.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin

2. Select the “books” core in the core selector the left side of the page.
3. Select the “Query” option below the books in the core selector.
4. For the “q” parameter enter “string OR space.” Review the 5 results
5. For the “sort” parameter enter “score desc”.
6. For “fl” parameter enter “score,title”.
7. For the “df” parameter enter title.
8. Select the “Execute Query” button.
9. Now, change the q parameter to “string^5 OR space.” How does the ranking change?

The results indicate that documents with the term “string” will score higher than the documents that just have the term “space,” when we boost that term.

7.4. Task 4 – eDismax Query

The Solr scoring algorithm has been customized to provide scoring based on the most "significant" portions of the query. Using the "edismax" parser the query results are scored based on the highest scoring boolean "sub query" within the query expression. This custom scoring algorithm is activated by using a the "edismax" query parser.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the "books" core in the core selector the left side of the page.
3. Select the "Query" option below the books in the core selector.
4. For the "q" parameter, enter "theory OR string OR practice".
5. For the "sort" parameter, enter "score desc".
6. For the "fl" parameter, enter "score,title,summary".
7. Check the "edismax" check box.
8. For the qf parameter, enter "title summary"
9. For the "mm" parameter "2".
10. Select the "Execute Query" button.

The query will force solr to order the documents based on the "maximum" score from each of the "disjunction" (or) queries across multiple fields specified in the qf parameter.

7.5. Task 5 – Field Boosting

The “edismax” query provides easy to API that provides “boost” specific to fields. By using the query fields parameter, a boost can be applied to each query field individually.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the “books” core in the core selector the left side of the page.
3. Select the “Query” option below the books in the core selector.
4. In the “q” parameter enter “practice”.
5. In the “sort” parameter enter “score desc”.
6. In the “fl” parameter enter “score,title,summary”.
7. Check the “edismax” check box.
8. For the “qf” parameter, enter “title summary”.
9. For the “mm” parameter, enter “2”.
10. Select the “Execute Query” button and review the results.
11. Now change the “qf” parameter to “title^10 summary”. How does the order of results change? More importance is given to the search term appearing in the title.
12. Now flip the boosts (“title summary^10). How does the order change?

7.6. Task 6 – Query Boosting

The “edismax” query provides easy to use API that provides a “boost” that is applied based on the result of a sub query. Use standard query syntax to provide a boost to the score based on matching criteria.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the “books” core in the core selector the left side of the page.
3. Select the “Query” option below the books in the core selector.
4. Enter into the “q” parameter add “theory OR string OR practice”.
5. Enter into the “sort” parameter add “score desc”.
6. Enter into the “fl” parameter add “score,title,summary,genre.”
7. Check the “edismax” check box.
8. Enter into the “bq” parameter “genre:science”.
9. For the qf parameter, enter “title”
10. Enter into the “mm” parameter “2”.
11. Select the “Execute Query” button.

Documents that match the query “genre:science” are boosted and score higher.

7.7. Task 7 – Function boosting

The “edismax” query also allows a boost to be applied based on the result of a function. By using the boost query parameter, a boost can be multiplied to the resulting score, using the result of a function call.

Steps

1. Open the admin user interface at http://localhost:8983/solr/admin
2. Select the “books” core in the core selector the left side of the page.
3. Select the “Query” option below the books in the core selector.
4. For the “q” parameter enter “theory OR string OR practice”.
5. Enter into the “sort” parameter add “score desc”.
6. For the “fl” parameter add “score,title,summary”.
7. Check the “edismax” check box.
8. For the “qf” parameter, enter “title summary”
9. For the “mm” parameter, enter “2”.
10. For the “boost” parameter, enter “recip(ms(NOW,publishdate),3.16e-11,1,1)” This ensures that newer documents are weighted higher than older ones.
11. Select the “Execute Query” button.

Note that **recip** a reciprocal function with **recip(x,m,a,b)** implementing a/(m*x+b). m,a,b are constants, x is any numeric field or arbitrarily complex function. The **ms** function returns milliseconds of difference between its arguments.