Getting Started with SolrCloud

Install Steps:

1. Run an ifconfig to get the local private IP address
2. Edit /etc/hosts and create a line to link the private IP to the public hostname example below (do not forget to sudo when editing the hosts file)
    a. localipaddress hostnameyouusedtoconnect
    b. 13.0.0.4 ec2-54-209-117-118.compute-1.amazonaws.com
3. Extract the solr-6.2.1 zip file
4. Edit the solr-6.2.1/bin/solr.in.sh within the solr directory

    a. Find the line where the solr host is configured and change the value to your public hostname that was provided at the beginning of this course
        ■ Example: SOLR_HOST="ec2-54-209-117-118.compute-1.amazonaws.com"
5. Run the following command : **./bin/solr start -e cloud**
6. Don't change any of the defaults when prompted to

Observations:

- You will have a 2 node SolrCloud cluster running
- The first node is running the embedded ZooKeeper
- You have a collection named **gettingstarted**
- It is a 2 shard collection with 2 replicas for every shard
- It is using the **data_driven_schema_configs** meaning schemaless mode i.e (we have the ability to not define a field name and Solr will type guess)
- We will always define the field beforehand as that's recommended.

Indexing Content:

- We will index data some movie data present under the **example/films** directory
- This data consists of the following fields:
    - **id** - unique identifier for the movie
    - **name** - Name of the movie
    - **directed_by** - The person(s) who directed the making of the film
    - **initial_release_date** - The earliest official initial film screening date in any country
    - **genre** - The genre(s) that the movie belongs to
- Use the schema API to define these fields:

```
curl http://localhost:8983/solr/gettingstarted/schema -X POST -H
'Content-type:application/json' --data-binary '{
  "add-field" : {
     "name":"name",
     "type":"text_general",
     "stored":true
  },
  "add-field" : {
     "name":"initial_release_date",
     "type":"tdate",
     "stored":true
  },
  "add-field" : {
     "name":"genre",
     "type":"text_general",
     "stored":true,
     "multiValued":true
  },
  "add-field" : {
```

```
        "name":"directed_by",
        "type":"text_general",
        "stored":true,
        "multiValued":true
      }
    }'
```

- The same command without whitespaces (easier to run on windows machine) :
    - curl http://localhost:8983/solr/gettingstarted/schema -X POST -H "Content-type:application/json" --data-binary "{ \"add-field\" : {\"name\":\"name\",\"type\":\"text_general\",\"stored\":true },\"add-field\" : {\"name\":\"initial_release_date\", \"type\":\"tdate\", \"stored\":true},\"add-field\" : {\"name\":\"genre\", \"type\":\"text_general\",\"stored\":true,\"multiValued\":true},\"add-field\" : {\"name\":\"directed_by\",\"type\":\"text_general\",\"stored\":true,\"multiValued\":true}}"
- To index on a Linux/Mac: **bin/post -c gettingstarted example/films/films.json**
- To index on Windows run this command from the example/exampledocs directory: **java -Dtype="application/json" -Durl="http://localhost:8983/solr/gettingstarted/update" -jar post.jar ../films/films.json**
    - NOTE: This command works for Linux/Mac as well , but since we have a 'post' script we make use of that.

Querying:

- Verify if 1100 docs have been loaded: http://localhost:8983/solr/gettingstarted/query?q=*
- Observe how many documents belong to shard 1: http://localhost:8983/solr/gettingstarted_shard1_replica1/query?q=*&distrib=false
- Observe how many documents belong to shard 2: http://localhost:8983/solr/gettingstarted_shard2_replica1/query?q=*&distrib=false
- Search for 'Batman': http://localhost:8983/solr/gettingstarted/query?q=name:batman
- Show me all 'Super hero' movies: http://localhost:8983/solr/gettingstarted/query?q=*:*&fq=genre:%22Superhero%20movie%22
- Let's see the distribution of genres across all the movies. See the facet section of the response for the counts: http://localhost:8983/solr/gettingstarted/query?q=*:*&facet=true&facet.field=genre

Understanding Analyzers

- Go to the Analysis Page present under a core on the Solr Admin Panel . http://localhost:8983/solr/#/gettingstarted_shard1_replica1/analysis
- Open the schema file stored in ZK on another window. It's present under **Cloud-> /*tree-> /configs -> /gettingstarted -> managed-schema***

- Let’s look at the "text_en" definition in the schema file and understanding what are the analyzers and the ordering of those
- On the analysis page, let’s type ***“the RED apples are on the table”*** into the "Field Value (Index)" text box and see how they get analyzed before finally being stored in the index
- Play around with a few sentences to get a hang of it
    - Example “apple” and “appl” both are equivalent. Some use cases this behaviour is useful while other applications might not want this.
- NOTE: When you try index to query matching to understand why a document should match a query , keep in mind that this is not a query parser. It simply applies the analyzers provided in the fieldType. To understand how a query gets parsed use &debug=query for a search request.
- Let’s add a new field type which doesn’t have the porter stemmer but converts singular to plural.

```
curl -X POST -H 'Content-type:application/json' --data-binary '{
  "add-field-type" : {
    "name": "english_field",
    "class": "solr.TextField",
    "positionIncrementGap": "100",
    "indexAnalyzer": {
       "tokenizer": {
          "class": "solr.StandardTokenizerFactory"
       },
       "filters": [
          {
             "class": "solr.StopFilterFactory",
             "words": "lang/stopwords_en.txt",
             "ignoreCase": "true"
          },
          {
             "class": "solr.LowerCaseFilterFactory"
          },
          {
             "class": "solr.EnglishMinimalStemFilterFactory"
          }
       ]
    },
    "queryAnalyzer": {
       "tokenizer": {
          "class": "solr.StandardTokenizerFactory"
       },
       "filters": [
          {
```

```
            "class": "solr.StopFilterFactory",
            "words": "lang/stopwords_en.txt",
            "ignoreCase": "true"
          },
          {
            "class": "solr.LowerCaseFilterFactory"
          },
          {
            "class": "solr.SynonymFilterFactory",
            "synonyms": "synonyms.txt",
            "ignoreCase": "true",
            "expand": "true"
          },
          {
            "class": "solr.EnglishMinimalStemFilterFactory"
          }
        ]
      }
}}' http://localhost:8983/solr/gettingstarted/schema
```

- Same command without whitespace: curl http://localhost:8983/solr/gettingstarted/schema -X POST -H "Content-type:application/json" --data-binary "{\"add-field-type\" : { \"name\": \"english_field\",\"class\": \"solr.TextField\",\"positionIncrementGap\": \"100\",\"indexAnalyzer\": {\"tokenizer\": {\"class\": \"solr.StandardTokenizerFactory\"},\"filters\": [{\"class\": \"solr.StopFilterFactory\",\"words\": \"lang/stopwords_en.txt\",\"ignoreCase\": \"true\"},{\"class\": \"solr.LowerCaseFilterFactory\"},{\"class\": \"solr.EnglishMinimalStemFilterFactory\"}]},\"queryAnalyzer\": {\"tokenizer\": {\"class\": \"solr.StandardTokenizerFactory\"},\"filters\": [{\"class\": \"solr.StopFilterFactory\",\"words\": \"lang/stopwords_en.txt\",\"ignoreCase\": \"true\"},{\"class\": \"solr.LowerCaseFilterFactory\"},{\"class\": \"solr.SynonymFilterFactory\",\"synonyms\": \"synonyms.txt\",\"ignoreCase\": \"true\",\"expand\": \"true\"},{\"class\": \"solr.EnglishMinimalStemFilterFactory\"}]}}}"
- When using the Schema APIs no collection RELOAD is required. You must re-index your data always if you change a schema field type
- Now return back to the analysis screen and run the same tests from above

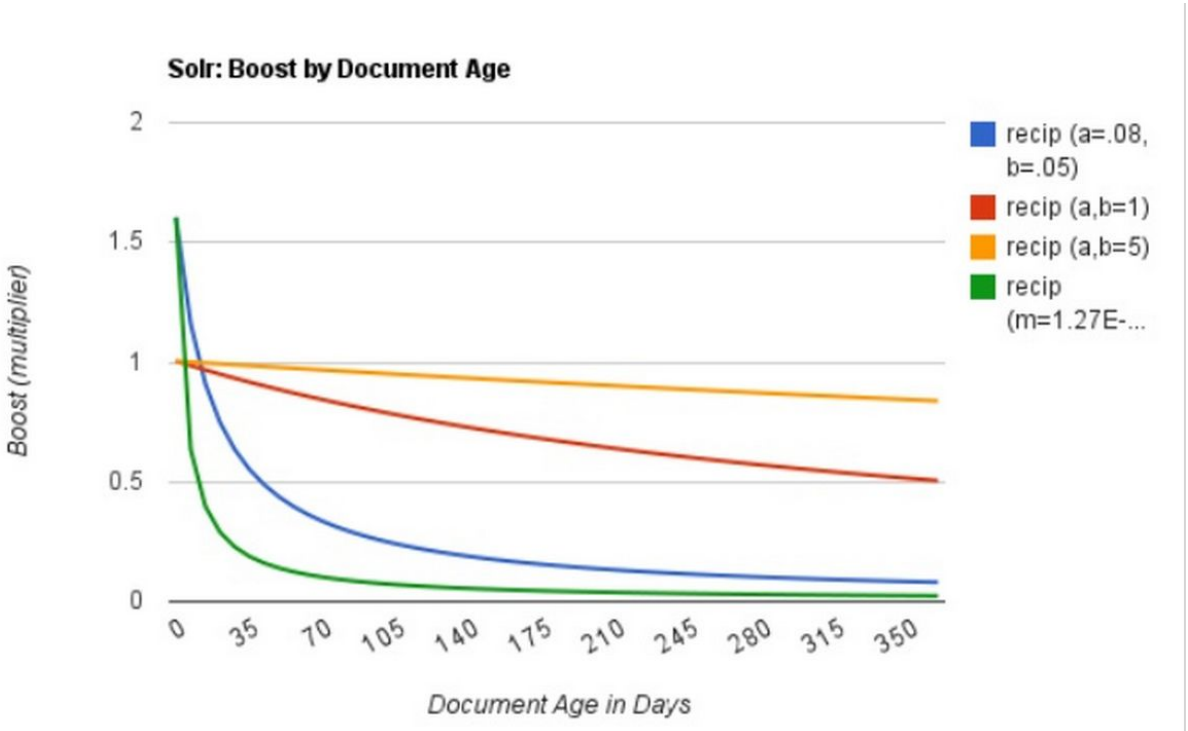
Indexing Content

- The aim of this lab is extract URLs and email addresses from text data and put it in it's own field before sending it to Solr.
- We will make use of update processors and use Javascript to achieve this.
- Let's start by creating a new collection for this data by running the following command: **./bin/solr create_collection -c files -d example/files/conf -s 1 -p 8983** or ***bin\solr create_collection -c files -d example/files/conf -s 1 -p 8983*** on Windows systems.
- To index on a Linux/Mac: ***./bin/post -c files ~/uth/apache-solr-ref-guide-6.2.pdf***
- To index on Windows run this command from the example/exampledocs directory: **java -Durl="http://127.0.0.1:8983/solr/files/update/extract" -jar post.jar ~/uth/apache-solr-ref-guide-5.3.pdf**
    - NOTE: This command works for Linux/Mac as well , but since we have a 'post' script we make use of that.
- We use an update chain called ***files-update-processor*** in this example. It's configured in on Line 860 in the solrconfig.xml file.
- If you look at the update chain we use a ***StatelessScriptUpdateProcessorFactory*** which is responsible for extracting the email and urls out of the text data.
- We also use a ***LangDetectLanguageIdentifierUpdateProcessorFactory*** to detect language of the incoming documents and put the value in a ***language*** field.
- Look at the ***update-script.js*** file to understand what the script is doing
    - How we extract the document type
    - How did we extract the email addresses and urls:
        - The ***type_att*** marks a token as ***<email>*** or ***<url>*** . This is done in the ***text_email_url*** fieldType. Explore this on the Analysis Page!
        - ***term_att*** gives us the actual value which we use to populate the field.

Relevance and Scoring

- For this lab we'll use the films data hence we'll be using the ***gettingstarted*** collection again.
- Search for all ***godzilla*** movies http://localhost:8983/solr/gettingstarted/query?q=name:godzilla&fl=name,initial_release_date,score&defType=edismax
- Let's prefer ***godzilla*** movies which are recently released
    - We will use the ***recip*** function query ***recip(x, m, a, b)*** implemented as ***a / (m*x + b)***
    - There are approximately 3.16e10 milliseconds in a year.
    - m = 3.16E-11

- a = 0.08
- b =0.05
- x = Document Age
- http://localhost:8983/solr/gettingstarted/query?q=name:godzilla&fl=name,initial_release_date,score&defType=edismax&boost=recip(ms(NOW/YEAR,initial_release_date),3.16e-11,1,1)
-



- Add &debugQuery=true&wt=ruby to understand the scoring.

Shard Splitting

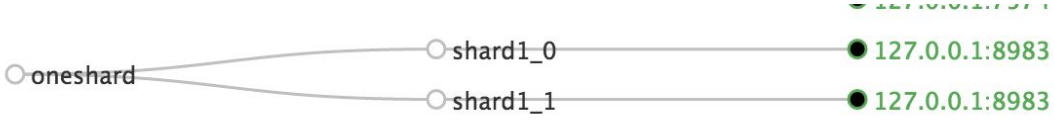
- The aim of this lab is to split a shard.
- **If using a remote machine please replace "localhost" with your AWS instance url.**
- Let's start with creating a new collection. We'll use the API this time and piggyback on the gettingstarted configs which are already present in ZK.
  - Command: http://localhost:8983/solr/admin/collections?action=CREATE&name=oneshard&numShards=1&replicationFactor=1&collection.configName=gettingstarted
- To index on a Linux/Mac: ***bin/post -c oneshard example/films/films.json***
- To index on Windows run this command from the example/exampledocs directory: ***java -Dtype="application/json" -Durl="http://localhost:8983/solr/oneshard/update" -jar post.jar ../films/films.json***

    - NOTE: This command works for Linux/Mac as well , but since we have a ‘post’ script we make use of that.
- Let’s split shard 1 http://localhost:8983/solr/admin/collections?action=SPLITSHARD&collection=oneshard&shard=shard1
- ***shard1*** will become ***shard1_0*** and ***shard1_1***
- Open the state.json file : ***solr admin -> cloud -> tree -> /collections -> /oneshard -> state.json***
    - If you look closely ***shard1*** has been marked as inactive
    - See the two new shards ***shard1_0*** and ***shard1_1***
- Let’s cleanup the old ***shard1*** lying around. Solr doesn’t automatically delete. The user should verify if everything is okay and then issue the delete command.
    - http://localhost:8983/solr/admin/collections?action=DELETESHARD&collection=oneshard&shard=shard1
- Verify if the number of documents in the system is still the same. http://localhost:8983/solr/oneshard/query?q=*
- To see how the documents get split into multiple shards we can query the shards individually.
    - shard1_0 documents : http://localhost:8983/solr/oneshard_shard1_0_replica1/query?q=*&distrib=false
    - shard1_1 documents http://localhost:8983/solr/oneshard_shard1_1_replica1/query?q=*&distrib=false

Collection Management

- The aim of this lab is to understand how to create replicas, move them around nodes and understand what Solr provides us with.
- Before we start there are a couple of important points to remember
    - A shard is a logical concept, a replica is the actual implementation of a shard hosting the data and is implemented in the form of a solr ‘core’ .
    - When we create a collection with two replicas Solr doesn’t guarantee that the replication factor will be maintained. For example we have 3 nodes and a 1 shard 2 replica collection. Both replicas sit on node one and two. Now if node two crashes or is de-commissioned Solr won’t automatically create a replica on node three since the replication factor isn’t satisfied. This needs to be instrumented in your application.
    - **If using a remote machine please replace “localhost” with your AWS instance url.**
- Let’s reuse the collection created in the previous lab ***oneshard*** . This should currently be a two shard collection with one replica per shard. Both shards are currently sitting on the same node.

- Move the shard_1_1 replica1 from one the solr node running on port 8983 to the solr node running on 7574 . There is no move command in Solr so we'll be doing an add replica on node 7574 effectively having two replicas for that shard temporarily before deleting the replica on node 8983
- Step 1 to add a new replica on the other solr node. The API action is called ADDREPLICA
    - Before running this command you should go the ***solr admin -> cloud -> graph*** and verify which solr node is hosting the replica of shard1_1 . Change the "node" parameter appropriately. For example. In the screenshot the current replica is on solr node 8983 , hence the API call explicitly specifies solr node 7574 as the second replica



    - API Call : http://localhost:8983/solr/admin/collections?action=ADDREPLICA&collection=oneshard&shard=shard1_1&node=127.0.0.1:7574_solr

- Verify this on the graph view on the solr admin panel. At this point query requests for that shard will get distributed to both replicas. The new replica already has the data and Solr will continue to keep it in sync for any further updates
- Let's remove the ***shard1_1*** replica sitting on node 8983
    - To find out the replica name, you need to look up the key name for the replica you want to delete. Use the following CLUSTERSTAUS API call and find out the exact name of the replica that we want to delete.r
        - http:/localhost:8983/solr/admin/collections?action=CLUSTERSTATUS&collection=oneshard&wt=json&indent=on&shard=shard1_1
        - Key names are in the format of "core_nodeX"
    - Once you have the exact core_X name , use the following DELETEREPLICA API call to delete the replica
    - http://localhost:8983/solr/admin/collections?action=DELETEREPLICA&collection=oneshard&shard=shard1_1&replica=core_nodeX
- Verify this on the graph view on the solr admin panel.
- Collection Aliases can help your application query one endpoint while having the power to switch solr collections under the covers. This could be useful for re-indexing etc.
    - The API call is in the format of **/admin/collections?action=CREATEALIAS&name=name&collections=collectionlist**
    - So let's create an alias for the "gettingstarted" collection and use that alias for querying.

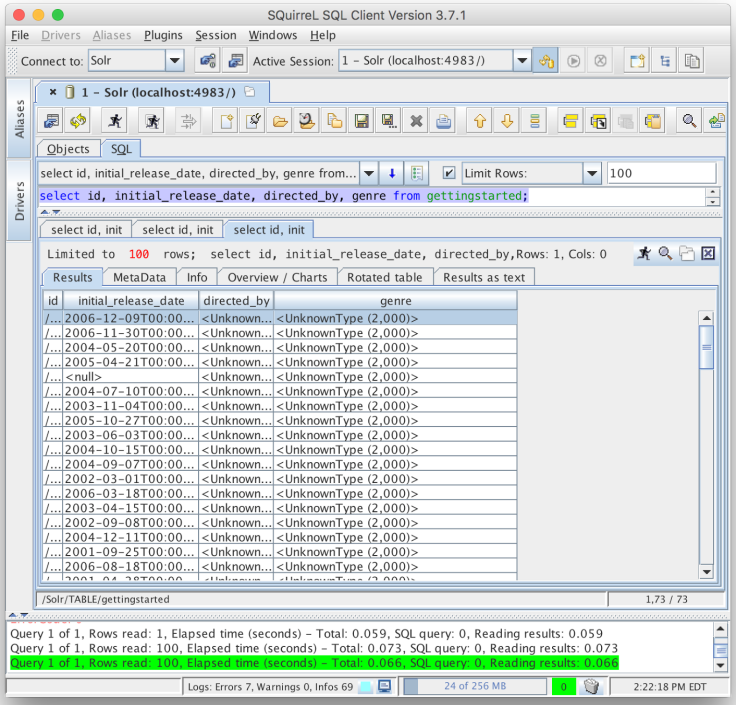
    - http://localhost:8983/solr/admin/collections?action=CREATEALIAS&name=gettingstarted_alias&collections=gettingstarted
    - Now you can run queries against the query_alias collection , for example
    - http://localhost:8983/solr/gettingstarted_alias/query?q=*:*

GC Log Analysis

- The aim of this lab is to be able to look at the GC log files and get an understanding of how is GC performing
- We are going to use a tool called gcviewer which is present in the training material. Run it with this command ***java -jar gcviewer-1.35.jar***
- As we discussed , the start script has GC logging turned on by default. So we're going to use GC logs from the system and go over some important metrics to analyze. Let's open example/cloud/node1/logs/solr_gc.log (current session)
    - scp -i solr-under-the-hood-kp.pem ubuntu@[Insert Hostname Here]:/home/ubuntu/SolrUnderTheHoodTrainingMaterial/solr-6.2.1/example/cloud/node1/logs/solr_gc.log [Insert Directory of Under the Hood Materials]
- In the ***Summary*** tab look at the ***throughput*** value. You should have a throughput of over 99% . This is a good indication that things are going smoothly.
- In the ***Memory*** tab we can look at the total heap usage and see how much of it is tenured vs young heap. If the total heap usage is above 90% one should monitor the application closely to make sure that there aren't any OutOfMemory errors and is a good indication that increasing heap or rethinking some indexing/query strategy is warranted.
- On the ***Pause*** tab the average pause time, max pause time are important statistics . An average pause more than a 5 second is never a good indication. The same would go with max pause time. Any max pause time more than 15 seconds will cause lots of churn in the system
    - 15s is the ZK client timeout and a pause longer than that leads ZK to think the node is ***down***.
- The final thing is the actual graph. A typical gc log is a sawtooth graph and the drop indicating that a GC activity has occurred. You can play around by going to the ***View*** panel and selecting only a few metrics to make more sense of what's going on.

Parallel SQL

- Find the squirrel directory in the lab materials, copy it locally (it should be included in the material you have downloaded locally)
- Enter that directory and run the “java -jar squirrel-sql.jar”
- Follow the instructions in the documentation to add your datasource. Remember that the ZK port is 1000 higher than the one you use to view the admin console. You need to use your VM’s ip address (not localhost as depicted in the picture below). Use the “gettingstarted” collection.
- The connection string should be following these guidelines
    - jdbc:solr://[ip_address]:9983/?collection=gettingstarted
- One connected run “select id, initial_release_date, directed_by, genre from gettingstarted;” from the SQL tab. What are your results.



- Why do you think directed_by and genre UnknownType?

Lab Graph Queries

45 minutes

Understand Solr Graph functions.

https://cwiki.apache.org/confluence/display/solr/Graph+Traversal [Solr wiki] says:

Graph traversal with streaming expressions uses the gatherNodes function to perform a breadth-first graph traversal.

The exercises in the lab provide a step-by-step introduction to this feature, by showing how to use graph expressions to follow a chain of page links between wikipedia entries.

The data is culled from DBpedia data dumps for persondata, as well as page abstracts

and links between wikipedia pages. The dataset contains listings approximately 40,000 actors and actresses and a set of approximately 134,000 links between the these pages.

Goals

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

1. Query using graph search

Task 1 – Compile and Run

CS Algorithms 101 Review: Breadth-First Search (BFS) and Graph Traversal

https://en.wikipedia.org/wiki/Graph_traversal#Pseudocode_2 [Wikipeida psuedocode] for breadth-first search:

```
1 procedure BFS(G, v):
2     create a queue Q
3     enqueue v onto Q
4     mark v
5     while Q is not empty:
6         t ← Q.dequeue()
7         if t is what we are looking for:
8             return t
9         for all edges e in G.adjacentEdges(t) do
12            o ← G.adjacentVertex(t, e)
13            if o is not marked:
14                mark o
15                enqueue o onto Q
16    return null
```

To modify this algorithm for breadth-first traversal, modify lines 7-8, instead of check and return, emit item `t` and continue.

In which case, this function becomes an iterator function which visits all nodes in a graph starting at a designated vertex

and visiting nodes in order of shortest path to longest path.

(Google programming quiz prep: graph traversal always uses a queue, depth-first traversal add nodes to the front of the queue -

FIFO, queue as stack - breadth-first traversal add nodes to the end of the queue - LIFO, queue as queue).

Solr Streaming Expression `gatherNodes`

The work done by the `gatherNodes` function is equivalent to lines 9-12 of the procedure BFS above:

* Given a starting document and a field, it returns the set of documents related to the starting document via the field value (line 9).

* It keeps track of document ids and performs a similar "mark" operation (line 14) so that if the graph contains a cycle, it only visits

the nodes in that cycle once.

The difference between the BFS algorithm and the Solr stream graph expression function `gatherNodes` is that

the former uses a while loop which allows for unlimited exploration of a graph, while the `gatherNodes` function

only takes a step of length one. To take longer walks through the graph, it is necessary to nest this function,

and nesting is limited to length 4. For many applications, this is a very acceptiable limit, especially

for social network analysis, given the popular meme which says that there are only

https://en.wikipedia.org/wiki/Six_degrees_of_separation [six degrees of separation]

between everyone on the planet.

Simple example of syntax of `gatherNodes` expression:

```
gatherNodes(emailsCollection,
        walk="POTUS->from",
        gather="to")
```

```

Given a collection of email documents named "emailsCollection",
for documents which have fields "from" and "to", it will find all documents
which have value "POTUS" in the from field and will retrieve the documents
which correspond to the values in the "to" field.

## Dataset `wikicinema`

Dataset culled from dbpedia.org downloads

* http://wiki.dbpedia.org/services-resources/datasets/data-set-39/downloads-39
* http://downloads.dbpedia.org/3.9/en/

DBpedia download files are like database table dumps.
To create a Solr document corresponding to a wikipedia page we need to
compose information from three data files:

* `persondata_en_cinema.json` list of JSON objects where each object
contains dbpedia person annotations - names and description.
All entries contains one of the following terms:  actor, actress, film, movie

* `page_links_en_cinema_pairwise.json`: list of JSON objects where each object
models link from one actor's wikipedia page to that of another actor.
Attribute "pageID_s"  is the from-page and attribute "linkID_s" is the to-page.

* `short_abstract_en_cinema.json` list of JSON objects where each
object has attributes "id" and "short_abstract_txt".
```

Create collection "wikicinema"

(assumes Solr is installed and running)

> (path/to/solr)/bin/solr create_collection -c wikicinema

Load data:

Use "post.jar" from Solr distro "example/exampledocs" directory.

```

> cd example/exampledocs

> cp $installdir/graph_lab_dataset/*.json .

```

Load file: `persondata_en_cinema.json`

This file contains JSON entries for actors and actresses coded as JSON objects:

```

head persondata_en_cinema.json

[

{"id": "Allan_Dwan", "name_s": "Allan Dwan", "surname_s": "Dwan", "description_txt": "Film director, film producer, screenwriter", "givenName_s": "Allan"},

{"id": "Andrei_Tarkovsky", "name_s": "Andrei Tarkovsky", "surname_s": "Tarkovsky", "description_txt": "Film director", "givenName_s": "Andrei"},

{"id": "Alfred_Hitchcock", "name_s": "Alfred Joseph Hitchcock", "surname_s": "Hitchcock", "description_txt": "British film director and film producer", "givenName_s": "Alfred Joseph"},

{"id": "Akira_Kurosawa", "name_s": "Akira Kurosawa", "surname_s": "Kurosawa", "description_txt": "a Japanese film director, screenwriter, producer, and editor", "givenName_s": "Akira"},

{"id": "Alfonso_Ar%C3%A1u", "name_s": "Alfonso Arau", "surname_s": "Arau", "description_txt": "Film director, actor", "givenName_s": "Alfonso"},
```

{"id": "Arnold_Schwarzenegger", "name_s": "Arnold Schwarzenegger", "surname_s": "Schwarzenegger", "description_txt": "Austrian-American bodybuilder, actor, politician", "givenName_s": "Arnold"},

{"id": "Amitabh_Bachchan", "name_s": "Amitabh Bachchan", "surname_s": "Bachchan", "description_txt": "Film actor", "givenName_s": "Amitabh"},

{"id": "Aaliyah", "name_s": "Aaliyah", "description_txt": "Singer, dancer, actress, model"},

{"id": "Anthony_Hopkins", "name_s": "Anthony Hopkins", "surname_s": "Hopkins", "description_txt": "Actor", "givenName_s": "Anthony"},

```

Load directive:

```

> java -Dc=wikicinema -Dtype=application/json -jar post.jar persondata_en_cinema.json
```

## Load file: `short_abstacts_en_cinema.json`

This file contains JSON entries for wikipedia short abstact text, in field name "short_abstact_txt".

Because Solr already contains documents with these ids, the JSON contains Solr directive to add this field to existing documents.

```

head -4 short_abstracts_en_cinema.json

[

{"short_abstract_txt": {"set": "Alfonso Ar\\u00E1u Inch\\u00E1ustegui (born January 11, 1932) is a Mexican actor and director."}, "id": "Alfonso_Ar%C3%A1u"},

{"short_abstract_txt": {"set": "Arnold Alois Schwarzenegger is an Austrian and American former professional bodybuilder, actor, producer, director, businessman, investor, and politician. Schwarzenegger served two terms as the 38th Governor of California from 2003 until 2011. Schwarzenegger began weight training at the age of 15. He won the Mr. Universe title at age 20 and went on to win the Mr. Olympia contest seven times."}, "id": "Arnold_Schwarzenegger"},

{"short_abstract_txt": {"set": "Amitabh Harivansh Bachchan (11 October 1942) is an Indian film actor. He first gained popularity in the early 1970s as the \\\"angry young man\\\" of Hindi cinema, and has since appeared in over 180 Indian films in a career spanning more

than four decades. Bachchan is widely regarded as one of the greatest and most influential actors in the history of Indian cinema."}, "id": "Amitabh_Bachchan"},
```

Load directive:

```
> java -Dc=wikicinema -Dtype=application/json -jar post.jar short_abstracts_en_cinema.json
```

Confirm that we have good documents - from browser, request URL:

```
http://localhost:8983/solr/wikicinema/select?indent=on&q=id:"Arnold_Schwarzenegger"&wt=json
```

Response should be:

```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":0,
    "params":{
      "q":"id:\"Arnold_Schwarzenegger\"",
      "indent":"on",
      "wt":"json"}},
  "response":{"numFound":1,"start":0,"docs":[
      {
```
```

"id":"Arnold_Schwarzenegger",

"name_s":"Arnold Schwarzenegger",

"surname_s":"Schwarzenegger",

"description_txt":["Austrian-American bodybuilder, actor, politician"],

"givenName_s":"Arnold",

"short_abstract_txt":["Arnold Alois Schwarzenegger is an Austrian and American former professional bodybuilder, actor, producer, director, businessman, investor, and politician. Schwarzenegger served two terms as the 38th Governor of California from 2003 until 2011. Schwarzenegger began weight training at the age of 15. He won the Mr. Universe title at age 20 and went on to win the Mr. Olympia contest seven times."],

"_version_":1546001877694414848}]

}}

```

## Load file: `page_links_en_cinema_pairwise.json`

This file contains all pairwise wikipedia links from wikipedia pagest for one page in the persondata pageset

to another page in this same set.  Taken together, this is a set of pairwise directed links between actors

and actresses who have wikipedia pages.

```

> head page_links_en_cinema_pairwise.json

[

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Keanu_Reeves"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Anthony_Quinn"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Sam_Peckinpah"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Martin_Short"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Chevy_Chase"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Michael_Douglas"},

{"pageID_s": "Alfonso_Ar%C3%A1u", "linkID_s": "Kathleen_Turner"},

{"pageID_s": "Arnold_Schwarzenegger", "linkID_s": "Steve_Reeves"},

{"pageID_s": "Arnold_Schwarzenegger", "linkID_s": "Johnny_Weissmuller"},
```

Load directive:

```
> java -Dc=wikicinema -Dtype=application/json -jar post.jar page_links_en_cinema_pairwise.json
```

Find entries linked to Kevin Bacon's page via request URL:
http://localhost:8983/solr/wikicinema/select?indent=on&q=linkID_s:Kevin_Bacon&wt=json
Response should be:

```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":0,
    "params":{
      "q":"linkID_s:Kevin_Bacon",
      "indent":"on",
      "wt":"json"}},
  "response":{"numFound":61,"start":0,"docs":[
      {
        "pageID_s":"Jennifer_Aniston",
        "linkID_s":"Kevin_Bacon",
        "id":"b7614f99-8938-4777-94c9-06fda14cfa7e",
        "_version_":1546001898791764002},
      {
        "pageID_s":"Dustin_Hoffman",
```
```

```
    "linkID_s":"Kevin_Bacon",
    "id":"d99c1ea2-0c1a-40b4-a9ed-c2b3c8e68d0a",
    "_version_":1546001898803298307},
  {
    "pageID_s":"Clint_Eastwood",
    "linkID_s":"Kevin_Bacon",
    "id":"75643044-53b0-49f5-968c-fb659c6acdd4",
    "_version_":1546001898817978391},
  {
    "pageID_s":"Tom_Hanks",
    "linkID_s":"Kevin_Bacon",
    "id":"fa8f87eb-476d-485a-aee1-19d9ac3a42f6",
    "_version_":1546001898819026980},
  {
    "pageID_s":"Val_Kilmer",
    "linkID_s":"Kevin_Bacon",
    "id":"17108466-9abe-4f65-ad00-be19f5ccc0c6",
    "_version_":1546001898872504356},
 ....
```


Load file: `page_links_en_cinema_multivalue.json`

Like the page_links_en_cinema_pairwise.json, this models the information from the dbpedia "pagelinks_en" dumpfile.

However, this time we code up the page link information as a multi-valued field named "linkID_ss" which will be added

to existing documents for the actor entry.

As in the file `short_abstacts_en_cinema.json`, the Solr "set" directive is used to add this field to an existing document:

```

> head page_links_en_cinema_multivalue.json

[

{"id": "Alfonso_Ar%C3%A1u", "linkID_ss": {"set": ["Michael_Douglas", "Martin_Short", "Sam_Peckinpah", "Chevy_Chase", "Anthony_Quinn", "Keanu_Reeves", "Kathleen_Turner"]}},

{"id": "Arnold_Schwarzenegger", "linkID_ss": {"set": ["Reese_Witherspoon", "Kirk_Douglas", "John_Wayne", "Bruce_Willis", "Lou_Ferrigno", "Sylvester_Stallone", "Demi_Moore", "Jamie_Lee_Curtis", "Jayne_Mansfield", "Dwayne_Johnson", "Johnny_Weissmuller", "Jennifer_Aniston", "Steve_Reeves", "Ann-Margret", "Mickey_Hargitay", "Jeff_Bridges", "Jackie_Chan"]}},

{"id": "Amitabh_Bachchan", "linkID_ss": {"set": ["Mammootty", "Arshad_Warsi", "Madhu_(actor)", "Raakhee", "Jalal_Agha", "Aishwarya_Rai", "Utpal_Dutt", "Raj_Kapoor", "Johnny_Depp", "Shahrukh_Khan", "Dilip_Kumar", "Neetu_Singh", "Rishi_Kapoor", "Nirupa_Roy", "Puneet_Issar", "Manoj_Kumar", "Zeenat_Aman", "Major_Ravi", "Simran_(actress)", "Fran%C3%A7ois_Truffaut", "Abhishek_Bachchan", "Anil_Kapoor", "Mohanlal", "Slumdog_Millionaire", "Shashi_Kapoor", "Rekha"]}},

```

Solr review: `DocValues`

DocValues are critical - (Solr Unleashed slide 87 - needs example?)

https://cwiki.apache.org/confluence/display/solr/DocValues

> DocValue fields are column-oriented fields with a document-to-value mapping built at index time.

The essential thing to understand is that a DocValues field is a field-specific data store that maps the doc-ID to field value.

For a given field, all the values are stored in one place.

This facilitates sorting a large set of documents based on the field value.

If a field is stored as a standard Lucene/Solr inverted index data structure, to fetch the stored field values required

retrieveing the doc IDs from the term index for that field and then fetching the stored

field values on a per-document basis.

Depending on the size of the results set being sorted and the overall number of documents in the index,

using DocValues will result in considerable performance speedup.

Solr review: streaming expression function `search`

* q searches over multi-valued field "linkID_ss:Kevin_Bacon" selects all docs which link to

* fl not same as regular Solr query - only "DocValue" fields can be retrieved

* must specify "sort" field, *note*: can't sort on multi-value fields.

* qt = "/export" used to pull back entire docset

Solr stream syntax example - find all wikipedia entries with links to Kevin Bacon, file `c1.txt`:

```
expr=search(wikicinema,
            q="linkID_ss:Kevin_Bacon",
            fl="id",
            sort="id asc",
            qt="/export")
EOF
{"result-set":{"docs":[
{"id":"Alison_Eastwood"},
{"id":"Ann_Dowd"},
{"id":"Brad_Renfro"},
{"id":"Brian_Benben"},
{"id":"Bruce_Payne"},
{"id":"Cathryn_Damon"},
{"id":"Charles_Siebert"},
{"id":"Chris_Penn"},
```

{"id":"Christina_Applegate"},
{"id":"Clint_Eastwood"},
{"id":"Conor_O'Farrell"},
{"id":"David_Andrews_(actor)"},
{"id":"Dustin_Hoffman"},
{"id":"Elisabeth_Shue"},
{"id":"Emma_Stone"},
{"id":"Evan_Rachel_Wood"},
{"id":"Eve_(entertainer)"},
{"id":"Fred_Ward"},
{"id":"Garrett_Hedlund"},
{"id":"James_McAvoy"},
{"id":"Jennifer_Aniston"},
{"id":"Jennifer_Morrison"},
{"id":"John_Bair"},
{"id":"John_Patrick_Amedori"},
{"id":"Julie_White"},
{"id":"Kyra_Sedgwick"},
{"id":"Laurence_Fishburne"},
{"id":"Liza_Weil"},
{"id":"Mickey_Rourke"},
{"id":"Mohan_Kapoor"},
{"id":"Neve_Campbell"},
{"id":"Paul_Reiser"},
{"id":"Rachel_Weisz"},
{"id":"Reba_McEntire"},
{"id":"Ren%C3%A9e_Zellweger"},
{"id":"Rhona_Mitra"},
{"id":"Rob_Lowe"},
{"id":"Robert_Sedgwick_(actor)"},
{"id":"Ron_Eldard"},
{"id":"Sam_Rockwell"},

{"id":"Sean_Astin"},

{"id":"Sean_McCann_(actor)"},

{"id":"Sosie_Bacon"},

{"id":"Teresa_Palmer"},

{"id":"Tippi_Hedren"},

{"id":"Tom_Hanks"},

{"id":"Tom_Wopat"},

{"id":"Tony_Goldwyn"},

{"id":"Val_Kilmer"},

{"id":"William_Devane"},

{"id":"Zac_Efron"},

{"EOF":true,"RESPONSE_TIME":6}]}}

```

*NOTE* : Can return multi-value fields, file `c2.txt`:

```

curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \

<<EOF

expr=search(wikicinema,

q="linkID_ss:Kevin_Bacon",

fl="id,linkID_ss",

sort="id asc",

qt="/export")

EOF

{"result-set":{"docs":[

{"linkID_ss":["Clint_Eastwood","Kevin_Bacon","Marcia_Gay_Harden","Scott_Eastwood"],"id":"Alison_Eastwood"},

{"linkID_ss":["Brad_Renfro","Carey_Mulligan","Carrie_Snodgress","Channing_Tatum","Clint_Eastwood","Elizabeth_Marvel","Elizabeth_Perkins","Ethan_Hawke","Gretchen_Mol","Jennifer_Aniston","John_Goodman","Julianne_Moore","Kevin_Bacon","Kristin_Scott_Thomas","Mary_Steenburgen","Megan_Mullally","Meryl_Streep","Natalie_Portman","Rooney_Mara","Tom_Hanks","Vera_Farmiga"],"id":"Ann_Dowd"},

{"linkID_ss":["Billy_Bob_Thornton","Brad_Pitt","Dustin_Hoffman","Ian_McKellen","Jonathan_Taylor_Thomas","Joseph_Mazzello","Kevin_Bacon","Mickey_Rourke","Robert_De_Niro","Ron_Eldard","Susan_Sarandon","Tommy_Lee_Jones","Winona_Ryder"],"id":"Brad_Renfro"},

...
```

*NOTE* : Can't pull back any fields that aren't Doc Values, file `c3.txt`:

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \
<<EOF
expr=search(wikicinema,
            q="linkID_ss:Kevin_Bacon",
            fl="id,linkID_ss,short_abstract_txt",
            sort="id asc",
            qt="/export")
EOF
{"result-set":{"docs":[
{"EXCEPTION":"java.util.concurrent.ExecutionException: java.io.IOException: --> http://192.168.1.228:8983/solr/wikicinema_shard1_replica1/:short_abstract_txt must have DocValues to use this feature.","EOF":true,"RESPONSE_TIME":5}]}}
```

Solr graph expression function `gatherNodes` syntax

Required function arguments:

* collection name - 1st argument to this function
* "walk" - specify field that contains start value, must be a DocValues field
* "gather" - specify field that contains values collected by gather function
```

Additional function arguments:

* "fq" - applies a query as a filter on the nodes gathered.
* "scatter" - controls which nodes in the graph to emit. `scatter=leaves, branches` returns all nodes visited in the course of the gatherNodes operation, including the start node.

* aggregation functions: `count(*)`, `sum(field)`, `min(field)`, `max(field)`, `avg(field)`
* "trackTraversal": creates field "ancestor node" - used for graph visualization

Example 1: Actor pages that link to page for "Kevin_Bacon"

Find documents that contain pairwise link fields `linkID_s`, to link `pageID_s`,
gather values in field `pageID_s`, starting from linkID_s "Kevin_Bacon",
file `c4.txt`:

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \
<<EOF
expr=gatherNodes(wikicinema,
                 walk="Kevin_Bacon->linkID_s",
                 gather="pageID_s")
EOF
```


{"result-set":{"docs":[
{"node":"David_Andrews_(actor)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Elisabeth_Shue","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Tom_Wopat","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"James_McAvoy","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Millette_Alexander","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Sosie_Bacon","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Bruce_Payne","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Chris_Penn","collection":"wikicinema","field":"pageID_s","level":1},

{"node":"Eve_(entertainer)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Teresa_Palmer","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Zachary_Dylan_Smith","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Ann_Dowd","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Charles_Siebert","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Christina_Applegate","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Laurence_Fishburne","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Jennifer_Aniston","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Rachel_Weisz","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Kyra_Sedgwick","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Brian_Benben","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Rob_Lowe","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Tony_Goldwyn","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"John_Patrick_Amedori","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Liza_Weil","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Julie_White","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Sean_Astin","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Lee_Hyun-woo","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Mickey_Rourke","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Sean_McCann_(actor)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Neve_Campbell","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Reba_McEntire","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Ron_Eldard","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Jennifer_Morrison","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Fred_Ward","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Cathryn_Damon","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Dustin_Hoffman","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Evan_Rachel_Wood","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Tom_Hanks","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Zac_Efron","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Ren%C3%A9e_Zellweger","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Conor_O'Farrell","collection":"wikicinema","field":"pageID_s","level":1},

{"node":"Val_Kilmer","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Brad_Renfro","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Tippi_Hedren","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Jonathan_Ward_(actor)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Mohan_Kapoor","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Paul_Reiser","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Sam_Rockwell","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Alison_Eastwood","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"John_Bair","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Robert_Sedgwick_(actor)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Rhona_Mitra","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Garrett_Hedlund","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Clint_Eastwood","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"William_Devane","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Emma_Stone","collection":"wikicinema","field":"pageID_s","level":1},
{"EOF":true,"RESPONSE_TIME":161}]}}
```

**Example 2: actor pages that Kevin Bacon links to**

Find documents that contain pairwise link fields `linkID_s`, to link `pageID_s`,
gather values in field `linkID_s`, starting from pageID_s "Kevin_Bacon",
file `c5.txt`:

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \
<<EOF
expr=gatherNodes(wikicinema,
                 walk="Kevin_Bacon->pageID_s",
                 gather="linkID_s")
EOF
```
```

{"result-set":{"docs":[
{"node":"Meryl_Streep","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Kyra_Sedgwick","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Ellen_Barkin","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"James_Dean","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Atom_Egoyan","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Daniel_Stern_(actor)","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Sosie_Bacon","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Mickey_Rourke","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Val_Kilmer","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Mickey_Rooney","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Sean_Penn","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Steve_Guttenberg","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Colin_Firth","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Judy_Garland","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Elizabeth_Perkins","collection":"wikicinema","field":"linkID_s","level":1},
{"EOF":true,"RESPONSE_TIME":9}]}}
```

**Example 3: fail - gatherNode fields must be DocValue fields**

File `c6.txt` - attempt to plug in multivalue field "linkID_ss"

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \
<<EOF
expr=gatherNodes(wikicinema,
                 walk="Kevin_Bacon->id",
                 gather="linkID_ss")
EOF
{"result-set":{"docs":[
{"EXCEPTION":"java.util.concurrent.ExecutionException: java.lang.RuntimeException:
```
```

java.io.IOException: java.util.concurrent.ExecutionException: java.io.IOException: --> http://192.168.1.228:8983/solr/wikicinema_shard1_replica1/:can not sort on multivalued field: linkID_ss","EOF":true,"RESPONSE_TIME":7}]}}

```

**Example 4 - use results of `search` expression as starting point**

File `c7.txt` - this starts from set of pages that are linked-to from Kevin Bacon's wiki page:

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream <<EOF
> expr=gatherNodes(wikicinema,
>   search(wikicinema,q="linkID_ss:Kevin_Bacon",fl="id",sort="id asc",qt="/export"),
>   walk="id->linkID_s",
>   gather="pageID_s")
> EOF
{"result-set":{"docs":[
{"node":"Billy_Boyd_(actor)","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Dylan_McDermott","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Anthony_Steffen","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Teri_Garr","collection":"wikicinema","field":"pageID_s","level":1},
{"node":"Kevin_Costner","collection":"wikicinema","field":"pageID_s","level":1},
...
```

This search pulls back 1028 nodes that are one link beyond pages directly linked-to by Kevin Bacon's wikipedia entry page.

**Example 5 - parameter "scatter"**

To see the distance between the starting node and all gathered nodes, use parameter `scatter="branches,leaves"`, file `c8.txt`
```

Example 6 - nested `gatherNodes` expressions

This example uses a nested gatherNode operation, plus parameter `scatter="branches, leaves"`

to see distance, file `c9.txt`:

```
expr=gatherNodes(wikicinema,
>                gatherNodes(wikicinema,
>                            walk="Kevin_Bacon->pageID_s",
>                            gather="linkID_s"),
>                walk="node->pageID_s",
>                gather="linkID_s",
>                scatter="branches, leaves",
>                count(*))
> EOF
{"result-set":{"docs":[
{"node":"Kevin_Bacon","collection":"wikicinema","field":"node","level":0},
{"node":"Meryl_Streep","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Kyra_Sedgwick","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Ellen_Barkin","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"James_Dean","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Atom_Egoyan","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Daniel_Stern_(actor)","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Sosie_Bacon","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Mickey_Rourke","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Val_Kilmer","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Mickey_Rooney","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Sean_Penn","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Steve_Guttenberg","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Colin_Firth","collection":"wikicinema","field":"linkID_s","level":1},
```

{"node":"Judy_Garland","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Elizabeth_Perkins","collection":"wikicinema","field":"linkID_s","level":1},
{"node":"Caroline_O'Connor_(actress)","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Gig_Young","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Roddy_Piper","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Fred_Savage","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Ronald_Reagan","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Helen_Walker","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Kris_Kristofferson","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2}
,
{"node":"Mary-Kate_Olsen","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
{"node":"Amy_Adams","count(*)":1,"collection":"wikicinema","field":"linkID_s","level":2},
```

**Example 7 - track traversal**

Nesting the expressions 3 deep gives us Kevin Bacon's entire LinkedIn network!
File `c10.txt`:

```
curl -X POST -d @- http://localhost:8983/solr/wikicinema/stream \
<<EOF
expr=gatherNodes(wikicinema,
                 gatherNodes(wikicinema,
                             gatherNodes(wikicinema,
                                         walk="Kevin_Bacon->pageID_s",
                                         gather="linkID_s",
                                         trackTraversal="true",
                                         count(*)),
                             walk="node->pageID_s",
                             gather="linkID_s",
```
```

```
                    trackTraversal="true",
                    count(*)),
            walk="node->pageID_s",
            gather="linkID_s",
            scatter="branches, leaves",
            trackTraversal="true",
            count(*))
EOF
{"result-set":{"docs":[
{"node":"Kevin_Bacon","field":"node","level":0,"count(*)":11,"collection":"wikicinema","ancestors":["Kyra_Sedgwick","Sosie_Bacon","Paul_Reiser","Robert_Sedgwick_(actor)","Tom_Hanks","Clint_Eastwood","Val_Kilmer","Mickey_Rourke","Bruce_Payne","Chris_Penn","Dustin_Hoffman"]},
{"node":"Meryl_Streep","field":"linkID_s","level":1,"count(*)":34,"collection":"wikicinema","ancestors":["Diane_Keaton","Ra%C3%BAl_Juli%C3%A1","Julianne_Moore","George_Cloney","Kurt_Russell","Natalie_Portman","Bryan_Greenberg","Jeremy_Irons","Jack_Nicholson","Sandra_Bullock","Viola_Davis","Leonardo_DiCaprio","Lily_Tomlin","Goldie_Hawn","Robert_Redford","Amy_Adams","Anne_Hathaway","Kevin_Kline","Nicole_Kidman","Clint_Eastwood","Grace_Gummer","Tom_Cruise","Amanda_Seyfried","Carrie_Fisher","John_Cazale","Cher","Kevin_Bacon","Uma_Thurman","Reese_Witherspoon","Pierce_Brosnan","Shirley_MacLaine","Alec_Baldwin","Dustin_Hoffman","Jean_Arthur"]},
{"node":"Kyra_Sedgwick","field":"linkID_s","level":1,"count(*)":5,"collection":"wikicinema","ancestors":["Kevin_Bacon","Sosie_Bacon","Robert_Sedgwick_(actor)","Bruce_Payne","Edie_Sedgwick"]},
{"node":"Ellen_Barkin","field":"linkID_s","level":1,"count(*)":4,"collection":"wikicinema","ancestors":["Kevin_Bacon","Leonardo_DiCaprio","Val_Kilmer","Bruce_Payne"]},
{"node":"James_Dean","field":"linkID_s","level":1,"count(*)":16,"collection":"wikicinema","ancestors":["Liz_Sheridan","Martin_Landau","Rock_Hudson","Gig_Young","Marlon_Brando","Cher","Dennis_Hopper","Elizabeth_Taylor","Kevin_Bacon","Jo_Van_Fleet","Nick_Adams_(actor)","Julie_Harris","Lee_Strasberg","Natalie_Wood","Mickey_Rourke","Pier_Angeli"]},
```