From content to search: Speed-dating Apache Solr

Alexandre Rafalovitch

Apache Solr Committer
Apache Solr Popularizer

@arafalov

APACHECON North America

Sept. 24-27, 2018

Apache Solr search engine features

- Advanced full-text search – very very advanced!
- Facets and aggregations
- SolrCloud for scale
- Extremely configurable
  - Config files
  - Extension points
  - Plugins
- Near Real-Time indexing
- Multiple input formats: XML, JSON, CSV, GeoJSON
- Multiple output formats: XML, JSON, CSV, GeoJSON, PHP, Ruby, Python, XLSX...
- Multiple client options: SolrJ, JavaScript, Python, PHP, R, Ruby.....
- Integrates Apache Lucene, Tika, Log4J, Xerces, ZooKeeper, Velocity, OpenNLP
- Integrates IBM ICU4J (Unicode), Carrot (document clustering)
- Has web-based Admin UI based on public APIs
- Can also do Graphs, Machine Learning, SQL, Linear Algebra...
- The ~~little~~ Search engine that could!

Learning Solr

- Solr Reference Guide
  - 1250! pages of goodness
  - *Important!* Solr WIKI is no longer primary source
- Ships with many examples (with explanations in comments):
  - techproducts
  - cloud
  - schemaless
  - dih (DataImportHandler – 5 examples)
  - films
  - files
- *Solr Users* mailing list, IRC channel, StackOverflow,
- Videos on YouTube from Lucene/Solr Revolution (Activate, as of 2018)
- http://www.solr-start.com/ - My reference website

Can be a bit too much

- Solr Reference Guide is feature-organized
  - Some powerful features are hard to locate
- Examples are a bit of a kitchen-sink
- Printed books cannot keep up
- Many examples use artificial, tuned datasets
- Several ways to do similar things

Today – we start to solve that

- Use the smallest **learning** configuration
- Latest cool features
- End-to-end
- Evolve from the queries backwards
- Real life example(s)
  - Use “Data is plural” - Newsletter of useful/curious real-life datasets
  - https://tinyletter.com/data-is-plural
  - Sent by Jeremy Singer-Vine (https://www.jsvine.com/)
- Just a taste – but a good one
  - Enough for you to understand how to keep learning by yourself
  - Repo: https://github.com/arafalov/solr-apachecon2018-presentation

Learning setup

- Minimal setup
  - 1 Server
  - 1 Collection/Core (per example)
  - Minimal schema (no extra types)
  - Minimal configuration (all defaults, no caches, etc.)
- Larger production setup
  - Multiple servers and Zookeepers (SolrCloud)
  - Multiple collections with shards, replicas
  - Full configuration (see Reference Guide)

Rapid (Search) Application Development

10 Get the Solr server running

20 Create new core with custom config

30 Index some data

40 Run some queries/facets/aggregation/graph analysis/...

50 *IF NOT HAPPY*

60 Identify improvement

70 Evolve schema/configuration

80 Re/Index data

90 GOTO **40**

100 *ENDIF*

Setting up our own server

- Shipped examples are preconfigured
  - Magic location
  - Magic config files (in *server/solr/configsets/<templatedir>/conf)*
  - Hard to do multiple examples
  - Confusing to grow to production, later
- For clarity, let's setup our own server
  - Create server root directory (e.g. *sroot*)
  - Copy *solr.xml* to *sroot* (from *<solr install>/server/solr*)
  - **`bin/solr -s  <path to sroot>`**

    All bin commands are in *<solr install>/bin/*

    Default Solr port is 8983, other ports offset from that

Minimal learning config

- Absolute minimum – 2 files
  - *managed-schema* (an XML file)
    - Definition of fields, field types, unique keys
    - Can be managed by API directly (removes comments on use)
  - *solrconfig.xml (*also an XML file*)*
    - Endpoints, Default parameters, Caches, Pre-processing pipelines, …
    - A lot of system-admin and production config
    - Cannot be modified by API directly, but there are overlays and request parameters APIs
- We will also use:
  - *params.json*
    - supplements *solrconfig.xml*
    - helps to manage request parameter defaults with API
- Get them:
  https://github.com/arafalov/solr-apachecon2018-presentation/tree/master/configsets/minimal

Create Collection/Core

- `bin/solr create -c` **`dip`**
  `-d .../minimal`
  - Collection/core name: **dip**
  - Our 3 config files are in **.../minimal**
  - Uses API to talk to the server
- Creates full structure under ***sroot*/dip**
  - Exception: logs (by default) *<solr install>/server/logs*
- For SolrCloud, configs are in ZooKeeper!

Dataset - “Data is Plural” itself

- Google sheet linked from https://tinyletter.com/data-is-plural
- Save as .tsv (tab-separated values)
  - Mine goes until 12 Sept 2018
- Solr can index .csv, also .tsv with help
  - https://lucene.apache.org/solr/guide/7_4/post-tool.html#indexing-csv
  - https://lucene.apache.org/solr/guide/7_4/uploading-data-with-index-handlers.html#csv-formatted-index-updates

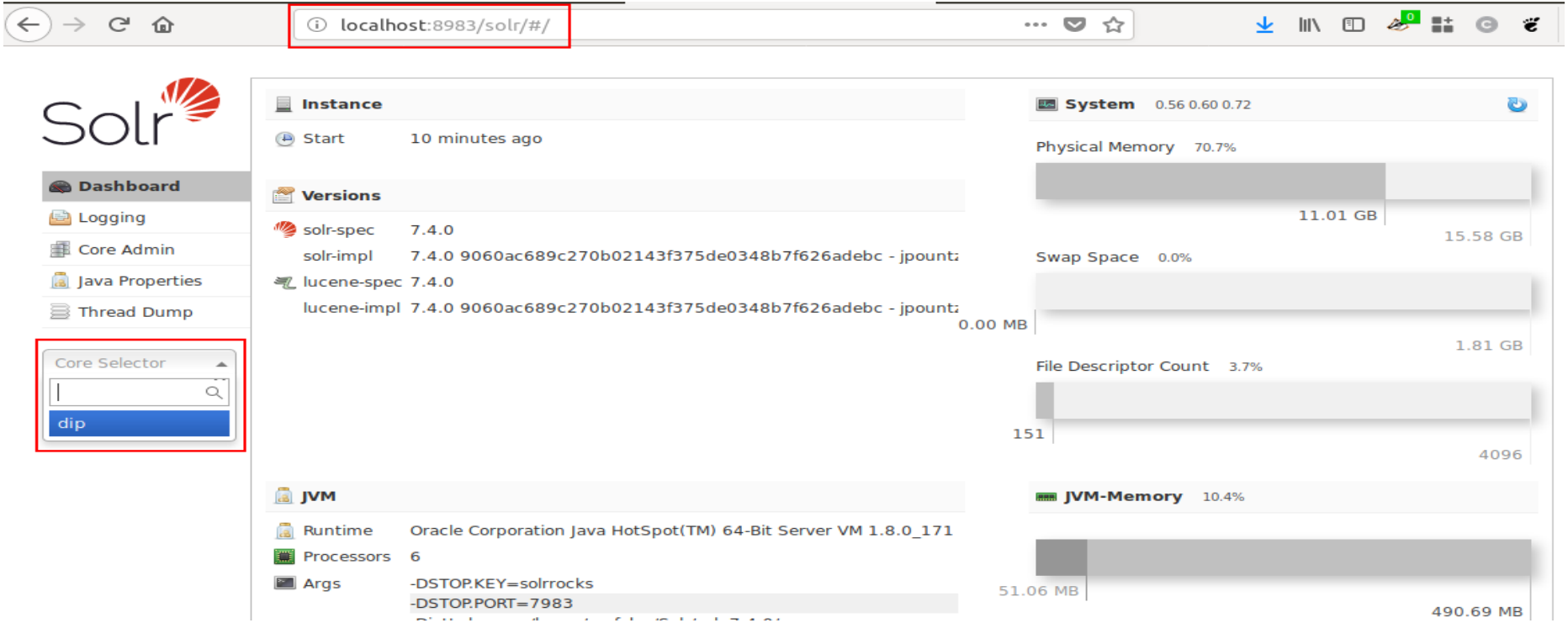
Index the content

- ```
  bin/post -c dip
    -params "separator=%09"
    -type "text/csv"
    .../dip-data.tsv
  ```
- Fails as mandatory uniqueKey id field is missing
- Let's use rowid as unique id
- ```
  bin/post -c dip
    -params "separator=%09&rowid=id"
    -type "text/csv"
    .../dip-data.tsv
  ```
- AND BOOM – We are ready for SEARCH

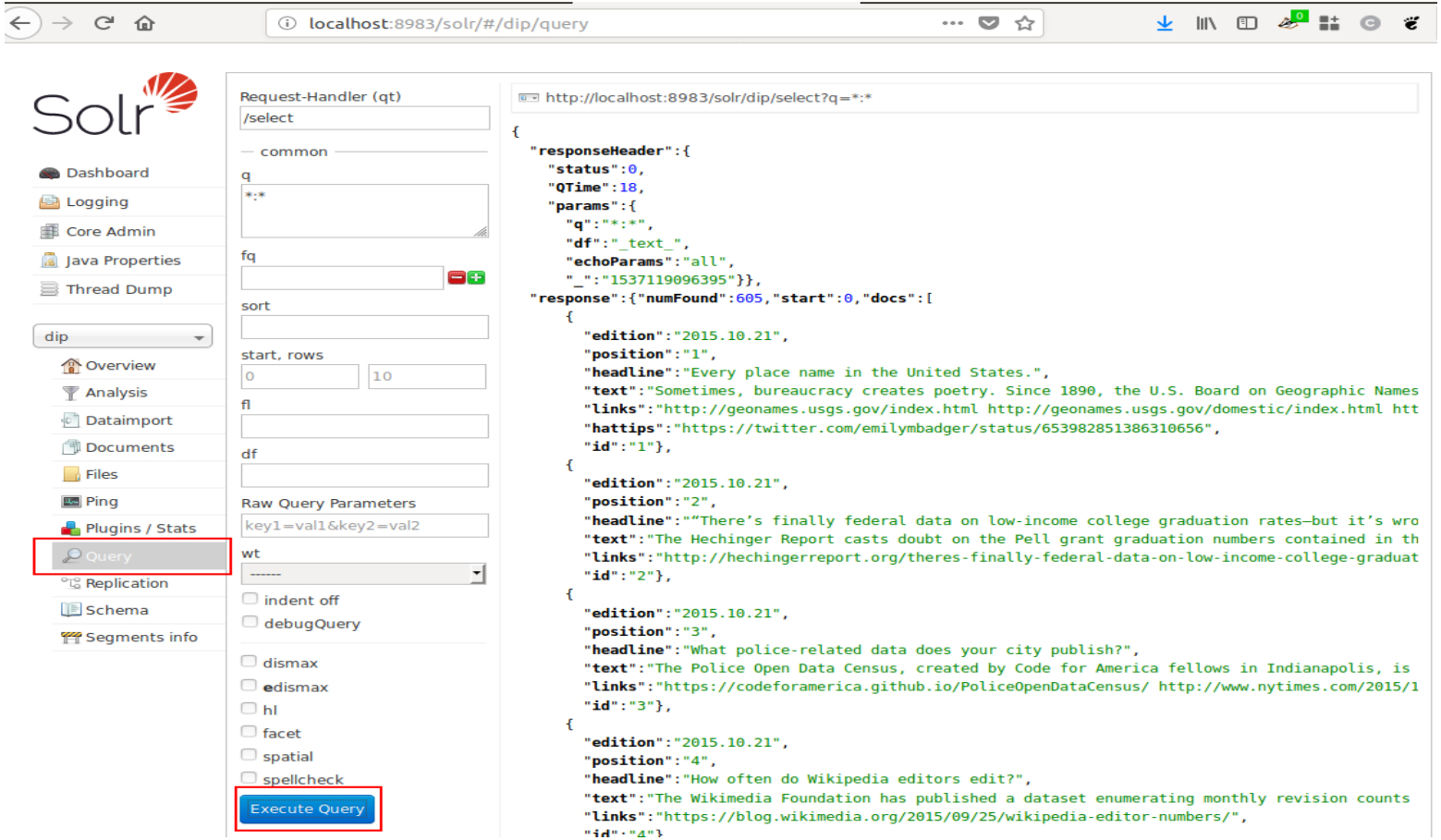
Admin UI

Start searching with Admin UI

http://localhost:8983/ - redirects to UI



Default search



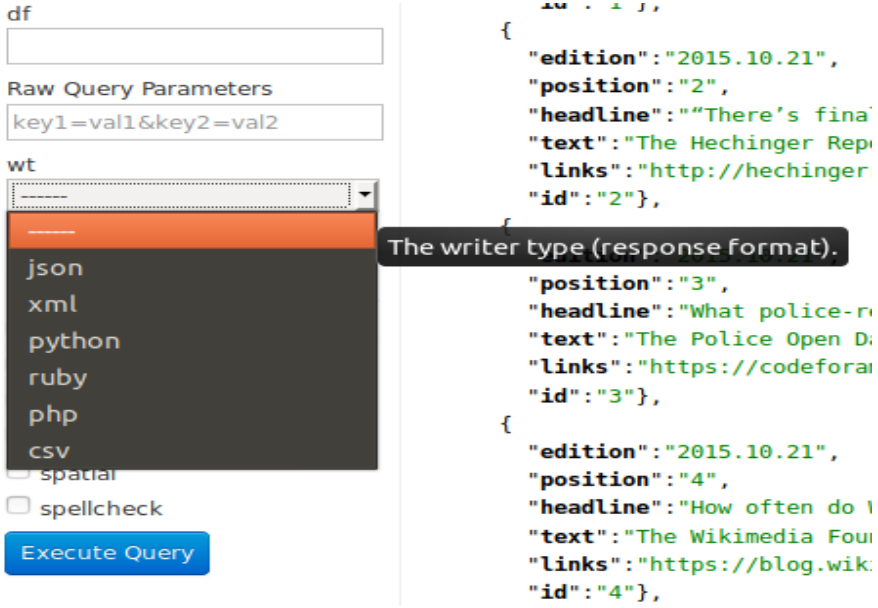
Default search – result details

http://localhost:8983/solr/dip/select?q=*:*


```
{
  "responseHeader":{
    "status":0,
    "QTime":18,
    "params":{
      "q":"*:*",
      "df":"_text_",
      "echoParams":"all",
      "_":"1537119096395"}},
  "response":{"numFound":605,"start":0,"docs":[
      {
        "edition":"2015.10.21",
        "position":"1",
        "headline":"Every place name in the United States.",
        "text":"Sometimes, bureaucracy creates poetry. Since 1890, the U.S. Board on Geographic Names
        "links":"http://geonames.usgs.gov/index.html http://geonames.usgs.gov/domestic/index.html htt
        "hattips":"https://twitter.com/emilymbadger/status/653982851386310656",
        "id":"1"},
      {
        "edition":"2015.10.21",
```

Output formats (wt)

- JSON by default:
  http://localhost:8983/solr/dip/select?q=*:*
- http://localhost:8983/solr/dip/select?q=*:*&wt=ruby



Basic searches

- Case-insensitive search (why?)
  - Search for '*data*' in the *q* input box
  - Search for 'DATA' instead
  - Same 461 results
  - What defines case sensitivity?
- Inclusion and exclusion
  - 'data news' - 469 results
  - '+data +news' - 46 results
  - '+data -news' - 415 results
- What defines search syntax?

Beyond search - Facets

http://localhost:8983/solr/dip/select?facet.field=edition&facet=on&q=*:*&rows=1

q
:

fq

sort

start, rows
0 1

facet

facet.query

facet.field
edition

facet.prefix

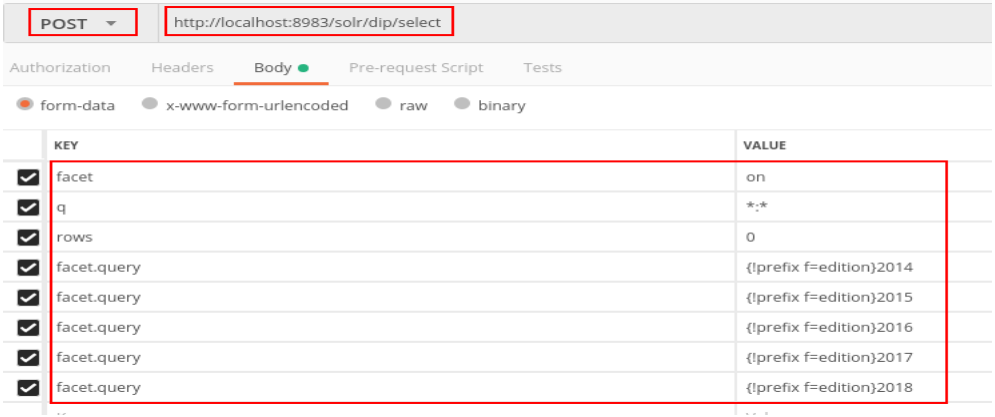

```
  "params":{
    "q":"*:*",
    "facet.field":"edition",
    "df":"_text_",
    "echoParams":"all",
    "rows":"1",
    "facet":"on",
    "_":"1537120852619"}},
 "response":{"numFound":605,"start":0,"docs":[
     {
       "edition":"2015.10.21",
       "position":"1",
       "headline":"Every place name in the United States.",
       "text":"Sometimes, bureaucracy creates poetry. Since 1890, the U.S. Board on Geographic Names
       "links":"http://geonames.usgs.gov/index.html http://geonames.usgs.gov/domestic/index.html htt
       "hattips":"https://twitter.com/emilymbadger/status/653982851386310656",
       "id":"1"}]
 },
 "facet_counts":{
   "facet_queries":{},
   "facet_fields":{
     "edition":[
       "2015.10.21",5,
       "2015.10.28",5,
       "2015.11.04",5,
       "2015.11.11",5,
       "2015.11.18",5,
       "2015.11.25",5,
```

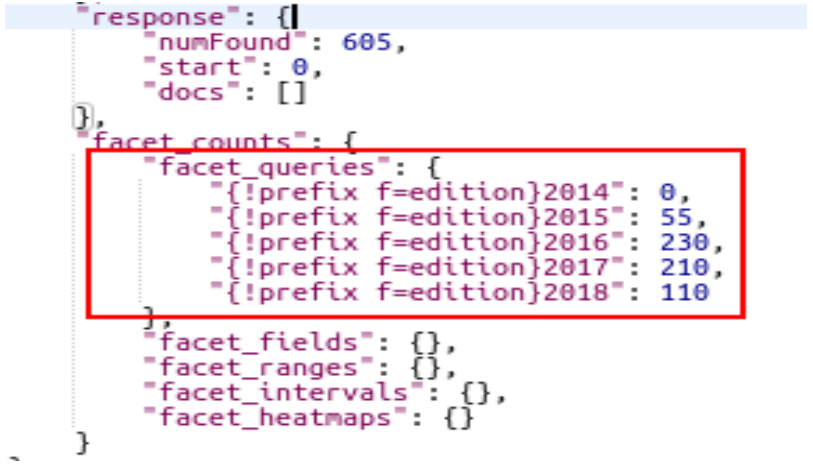
Facets – Group by year - GET

- What about grouping by year
- ```
  http://localhost:8983/solr/dip/select
  ?facet=on&q=*:*&rows=0
  &facet.query={!prefix f=edition}2014
  &facet.query={!prefix f=edition}2015
  &facet.query={!prefix f=edition}2016
  &facet.query={!prefix f=edition}2017
  &facet.query={!prefix f=edition}2018
  ```
- A bit beyond what Admin UI can allow you enter right now
- A bit painful as a GET
- Can be also be done as a POST
  ```

Facet – Group by year - POST

- Easier with Postman (https://www.getpostman.com/)
- Can save requests, create collections, publish





Quick review

- We did basic index of a real dataset
- We did some simple searches
  - Case-insensitive
  - Basic facets
  - Less-basic facets
- Already useful – and just the tip of Solr
- Next: let's understand WHY this works

Minimal learning config

- Absolute minimum – 2 files
  - *managed-schema* (an XML file)
    - Definition of fields, field types, unique keys
    - Can be managed by API directly (removes comments on use)
  - *solrconfig.xml (*also an XML file*)*
    - Endpoints, Default parameters, Caches, Pre-processing pipelines, …
    - A lot of system-admin and production config
    - Cannot be modified by API directly, but there are overlays and request parameters APIs
- We will also use:
  - *params.json*
    - supplements *solrconfig.xml*
    - helps to manage request parameter defaults with API
- Get them:
  https://github.com/arafalov/solr-apachecon2018-presentation/tree/master/configsets/minimal

Minimal config and params

- Minimal *solrconfig.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
  <luceneMatchVersion>7.4.0</luceneMatchVersion>

  <requestHandler name="/select"
     class="solr.SearchHandler" useParams="SELECT" />
</config>
```

- Minimal matching *params.json*

```
{"params":{
  "SELECT":{
    "df":"_text_",
    "echoParams":"all"
}}}
```

- Relies on a lot of defaults, check them with config API:
  http://localhost:8983/solr/***<corename>*/config?expandParams=true**
- **https://lucene.apache.org/solr/guide/7_4/config-api.html**

Minimal managed-schema

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

    <field name="id" type="string" required="true" indexed="true" stored="true" />
    <field name="_text_" type="text_basic"
           multiValued="true" indexed="true" stored="false" docValues="false"/>

    <dynamicField name="*" type="text_basic" indexed="true" stored="true"/>

    <copyField source="*" dest="_text_"/>

    <uniqueKey>id</uniqueKey>

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

    <fieldType name="text_basic" class="solr.SortableTextField" positionIncrementGap="100">
        <analyzer>
            <tokenizer class="solr.StandardTokenizerFactory"/>
            <filter class="solr.LowerCaseFilterFactory"/>
        </analyzer>
    </fieldType>

</schema>
```

Schema definition hierarchy

- Field type CLASS
  - Underlying representation
- Field Type
  - No built-in types
  - Configuration
  - Defaults
  - Analyzers (if supported)
- Field definition
  - Concrete configuration

- Popular field type classes
  - StrField
  - TextField
  - SortableTextField
  - IntPointField
  - DatePointField

- Other classes
  - BinaryField
  - BoolField
  - CollationField
  - CurrencyFieldType
  - DateRangeField
  - DoublePointField
  - ExternalFileField
  - EnumFieldType
  - FloatPointField
  - ICUCollationField
  - LongPointField
  - PointType
  - PreAnalyzedField
  - RandomSortField
  - SpatialRecursivePrefixTreeFieldType
  - UUIDField

managed-schema – field types

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

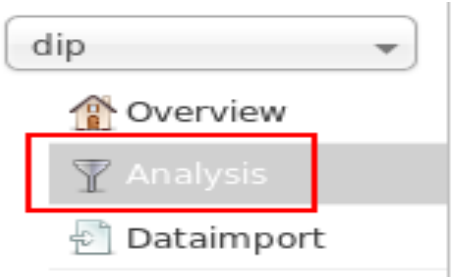
- Name: **string** – convention, not complusory
- Class: **solr.StrField -** keep string as is, no processing
- **sortMissingLast** – what to do with missing values
- **docValues** – enable column store, useful for faceting, sorting, grouping
- Additional configuration will be done on field using this type, can also override values here (not name/class)
- No analyzers magic for this (**solr.StrField**) type

managed-schema – field types

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

- Name: **text_basic** – whatever you want
- Class: **solr.SortableTextField** – new type useful for search and faceting both. Implicitly uses **docValues**
- **positionIncrementGap** – just keep this, a bit of an internal thing
- Basic combined index/query analyzer chain
  - Tokenizer: **`StandardTokenizerFactory`** – generic, split on whitespace and punctuation
  - (Token) Filter: **`LowerCaseFilterFactory`** - lowercase every token during index and search => case-insensitive match
- Many more interesting analyzer chains are in examples
- You can (and should) craft your own for specialty use-cases

Understanding analyzer chains




```
...
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
...
```

Field Value (Index)

This is an amazing, and - very - exciting NEWS!

Field Value (Query)

AMAZING news.

Analyse Fieldname / FieldType: text_basic Schema Browser Verbose Output Analyse Values

ST	This	is	an	amazing	and	very	exciting	NEWS
LCF	this	is	an	amazing	and	very	exciting	news

ST	AMAZING	news
LCF	amazing	news

managed-schema - fields

- `<field name="id" type="string" required="true" indexed="true" stored="true" />`
  - Field name is **id** – we refer to this later, in uniqueKey
  - Its type is **string** – as we defined
  - It is **required** – that's quite rare actually
  - It is **indexed** – means we can **search** on it
  - It is **stored**

    Means we will return it to the user

    Important for ID, a decision point for the rest
- `<uniqueKey>id</uniqueKey>`

managed-schema - fields

```
<field name="_text_" type="text_basic"
       multiValued="true" indexed="true"
       stored="false" docValues="false"/>
```

- Name: **_text_** - underscores to show special usage *in our config*
- Type: **text_basic** – defined later
- **multiValued** – can take multiple values
- **indexed** – we can search against it, using **text_basic** rules
- **(Not) stored** – will not show up in the result
- **(Disabled) docValues** – overrides type definition

Dynamic fields

- **`<dynamicField name="*" type="text_basic" indexed="true" stored="true"/>`**
  - Dynamic Field – match incoming field name that is not matched explicitly by other definitions – a magic fallback
  - Name: "*****" - pattern, means any left-over field name
    - could also be ***_str**, **random_***
  - **stored** – means the user will see the fields
  - **indexed** – means we can search against it
  - Type: **text_basic** – means we do the tokenization and analysis

copyField

```
<copyField source="*" dest="_text_"/>
```

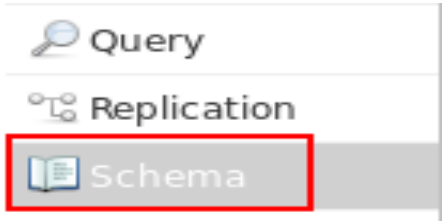
- Duplicates fields for different treatment
- Here:
  - copy ALL (*****) fields (all dynamic fields AND id field)
  - To field **_text_**
- Remember: **_text_**
  - is searchable (we even disabled docValues)
  - not stored (so user does not see this duplicated content)
  - multiValued (as it gets each field's content as separate value)
- Important! Copied content is searched by rules of the destination field…. (e.g. dates copied to text field)
- *Why do we need this duplication?*

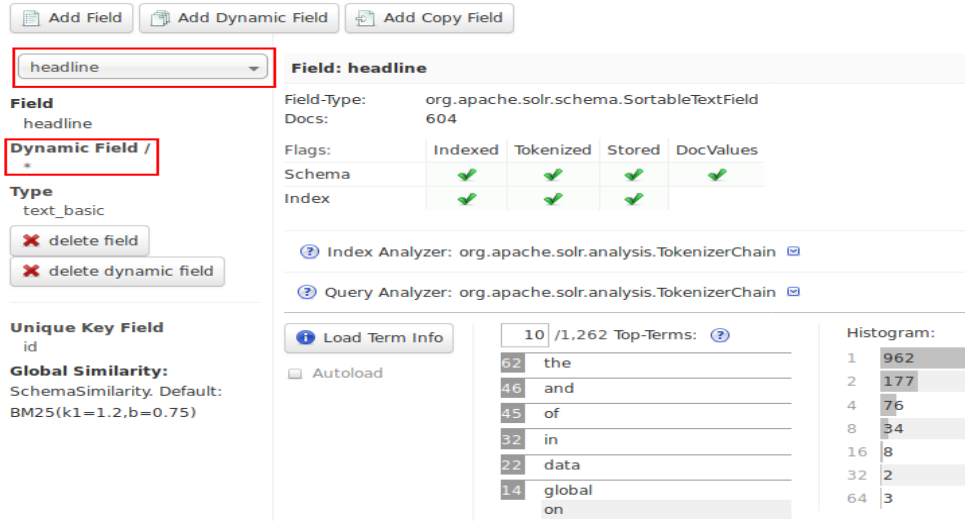
Query path

- What happens when we search “+data -news”
- Enable **debugQuery** flag to find out
  - http://localhost:8983/solr/dip/**select**?**debugQuery**=on&q=**+data -news**
  - "parsedquery":"+**_text_**:data -**_text_**:news"
- Solr has a query parser, which has parameters
  - In *solrconfig.xml*, we defined **/select** endpoint
  - It declared **useParams=SELECT**
  - In *params.json*, we set a **df** parameter to **_text_**
  - For default (**lucene**) Query Parser, **df** is *Default Field* parameter
- Other Query Parsers have very different expectations
  - eDisMax is popular and has much greater searched-fields control
  - Also: https://lucene.apache.org/solr/guide/7_4/other-parsers.html

Record details

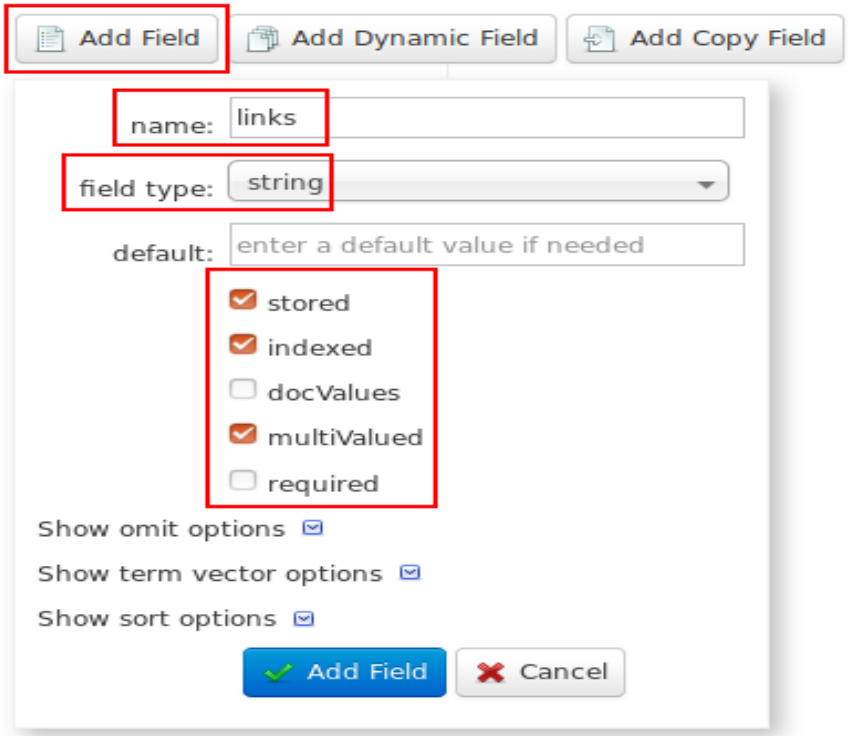

```
{
  "edition":"2015.10.21",
  "position":"1",
  "headline":"Every place name in the United States.",
  "text":"Sometimes, bureaucracy creates poetry. Since 1890, the U.S. Board on Geographic Names
  "links":"http://geonames.usgs.gov/index.html http://geonames.usgs.gov/domestic/index.html htt
  "hattips":"https://twitter.com/emilymbadger/status/653982851386310656",
  "id":"1"},
```



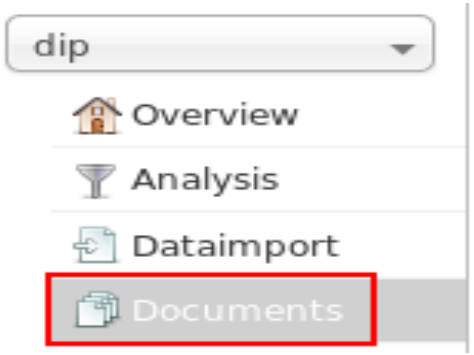
Evolving the schema

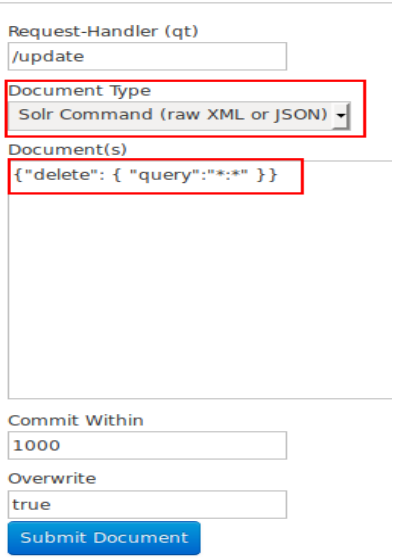
- Let's make **links** multi-valued strings, split while injesting on white-space
- In *managed-schema*, we still just have **id, _text_** as explicit fields
- Can do it in Admin UI
  - Delete **links** (not strictly needed for dynamic field)
  - Create explicit definition: **stored**, **indexed**, **multiValued**, using **string** field type
  - This will rewrite *managed-schema*
- Need to reindex, sometimes delete/reindex
  - Usually docValues require delete/reindex
  - *Remember*! Have a primary source



Deleting content

- `bin/post -c dip -d $"<delete><query>*:*</query></delete>"`
- Can also do it in Admin UI/Documents
- Can use (Solr-specific) XML or JSON, delete by ID or specific query
- https://lucene.apache.org/solr/guide/7_4/uploading-data-with-index-handlers.html





Reindexing

- `bin/post -c dip -params "separator=%09&rowid=id&`**`f.links.split=true`**`&`**`f.links.separator=%20`**`" -type "text/csv" .../dip-data.tsv`
- New parameters
  - f.links.split=true
  - f.links.separator=20


```
{
  "edition":"2015.10.21",
  "position":"1",
  "headline":"Every place name in the United States.",
  "text":"Sometimes, bureaucracy creates poetry. Since 1890, the U.S. Board on Geographic Names
  "links":["http://geonames.usgs.gov/index.html",
    "http://geonames.usgs.gov/domestic/index.html",
    "https://www.google.com/maps/place/Confusion+Creek,+Alaska/@68.4510925,-152.0233116,15.94z/
  "hattips":"https://twitter.com/emilymbadger/status/653982851386310656",
  "id":"1"},
```

Next challenge – links count

- Let’s find records with 3+ links
- Quite hard because we search from index (dictionary) of tokens
- Correct Solr reasoning:
  - Shape the data for search
  - Solr is not a (storage-oriented) database
  - Duplicate and index in multiple ways
  - Pre-calculate what you will search
  - De-normalize if needed

Update Request Processors

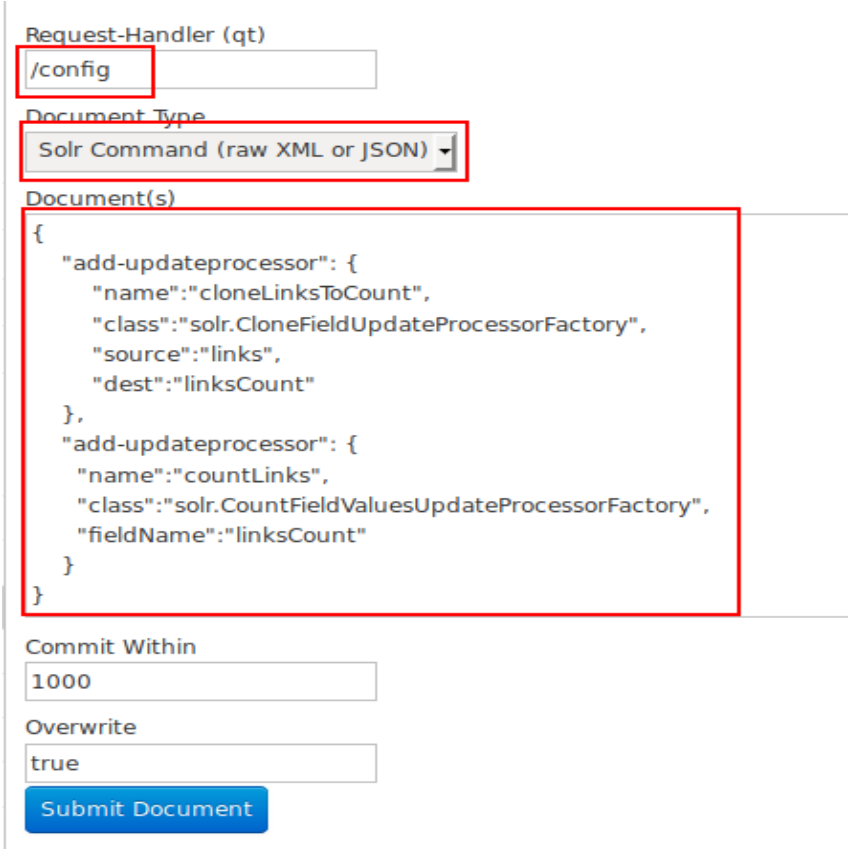
- Update Request Processor (URP)
  - Runs after format (XML, JSON, etc) parser
  - Runs before schema is involved
  - Can run multiple in a chain
- Many (legacy and new) ways to configure
  - Documentation (rather hard to discover): https://lucene.apache.org/solr/guide/7_4/update-request-processors.html
  - Also: http://www.solr-start.com/info/update-request-processors/
- Example complex chain: Solr “schemaless” type-guessing mode

Counting links

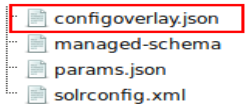
- To count links
  - Copy to another field (not yet involving schema) - *CloneFieldUpdateProcessorFactory*
  - Replace links themselves with their count – *CountFieldValuesUpdateProcessorFactory*
- Defining URPs
  - Some are implicit, but not ours
  - Can define in *solrconfig.xml* (edit, reload core)
  - Can use Config API to define an overlay (in configoverlay.json)

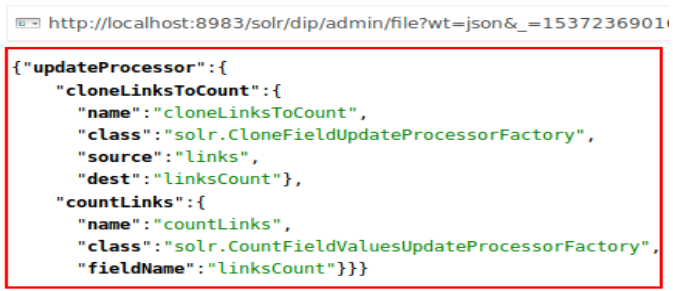
    https://lucene.apache.org/solr/guide/7_4/config-api.html
- Can use curl, Postman, or hack an Admin UI/Documents

Hacking Admin UI for Config API



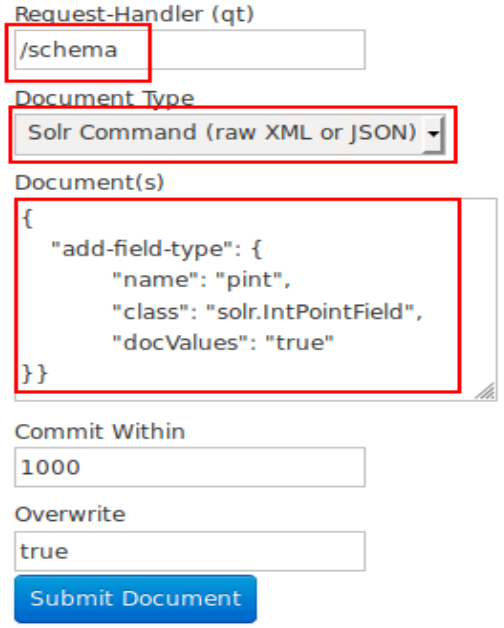
- Define two URPs
- Cannot define a chain via Config API
- Notice: We changed request handler
- Notice: JSON has duplicate keys-names (Ugh!)
- Once submitted, we have new *configoverlay.json* file in core's config directory
- Can check on Admin UI *Files* screen





Evolving schema for linksCount

- URP will generate **linksCount** field
- By default, it would be a **text_basic**
- Let's make it an Integer – better search
- Need to add a field type, no UI for that
- Can edit file manually and reload core
- Or can use Schema API
  https://lucene.apache.org/solr/guide/7_4/schema-api.html
- Again, we can use an Admin UI hack (/schema)
- Once field type (pint) is defined, use Admin UI to define **linksCount** as **pint** with default of 0
- As it is a new field, no need to delete index, can just reindex



Reindex with URPs

- `bin/post -c dip -params "separator=%09&rowid=id&f.links.split=true&f.links.separator=%20&`**`processor=cloneLinksToCount,countLinks`**`" -type "text/csv" .../dip-data.tsv`
- New parameter
  - processor=cloneLinksToCount,countLinks
- But that's getting too long
- *Can we do something about it?*
  - Hint: *params.json*

Dealing with solrconfig.xml

- Can modify solrconfig.xml by hand and reload core (legacy method)
- Can use Config API to create new entries – useful, but overkill for changing parameters
- Can use Request Parameters API with predefined or explicit *useParams*
  - *https://lucene.apache.org/solr/guide/7_4/request-parameters-api.html*
  - */config/params* endpoint

    Takes JSON with set/unset/delete keys

    Notice different escape for tab, space
- Once done, we can refer to it:
  - **`bin/post -c dip -params "useParams=DIP_INDEX" -type "text/csv" .../dip-data.tsv`**

Search for many links

http://localhost:8983/solr/dip/select?q=linksCount:[10 TO *]&sort=linksCount desc

common

q: linksCount:[10 TO *]

fq:

sort: linksCount desc

start, rows: 0 | 10

fl:

df:

Raw Query Parameters: key1=val1&key2=val2

wt: ------

indent off

debugQuery

dismax

edismax

hl

facet


```
"responseHeader":{
  "status":0,
  "QTime":1,
  "params":{
    "q":"linksCount:[10 TO *]",
    "df":"_text_",
    "sort":"linksCount desc",
    "echoParams":"all",
    "_":"1537240073797"}},
"response":{"numFound":6,"start":0,"docs":[
    {
      "edition":"2018.08.08",
      "position":"5",
      "headline":"More street trees.",
      "text":"London, Belfast, Vancouver, Washington (D.C.), Philadelphia, Boston, Cambridge (Mass.
      "links":["https://data.london.gov.uk/dataset/local-authority-maintained-trees",
        "https://www.opendatani.gov.uk/dataset/belfast-trees",
        "https://data.vancouver.ca/datacatalogue/streetTrees.htm",
        "http://dcgis.maps.arcgis.com/home/item.html?id=fea6079cf9bc4310a8b6c94f8c2bf1da",
        "https://www.opendataphilly.org/dataset/philadelphia-street-tree-inventory",
        "https://data.boston.gov/dataset/trees",
        "https://www.cambridgema.gov/GIS/gisdatadictionary/Environmental/ENVIRONMENTAL_StreetTrees"
        "https://data-cityofmadison.opendata.arcgis.com/datasets/street-trees",
        "https://data.providenceri.gov/Neighborhoods/Providence-Tree-Inventory/uv9w-h8i4/data",
        "https://data.sfgov.org/City-Infrastructure/Street-Tree-List/tkzw-k3nq",
        "https://data.oaklandnet.com/Environmental/Oakland-Street-Trees/4jcx-enxf",
        "https://data.cityofberkeley.info/Natural-Resources/City-Trees/9t35-jmin/data",
        "https://www.nycgovparks.org/trees/treescount",
        "https://tinyletter.com/data-is-plural/letters/data-is-plural-2016-11-16-edition"],
      "hattips":"https://twitter.com/vb_jens/status/909600422213455872 https://twitter.com/Sunlight
      "id":"590",
      "linksCount":14},
```

Everybody likes dogs and cats

- Now that we have the basics
- Let's choose another dataset
- How about: *+dogs +australia*
- Export the Sunshine Coast dataset as XML


```
"response":{"numFound":1,"start":0,"docs":[
    {
      "edition":"2018.04.04",
      "position":"5",
      "headline":"Even more dog (and cat) names.",
      "text":"Last year, Data Is Plural pointed readers to dog registration data for NYC, Tacoma, an
      "links":["https://tinyletter.com/data-is-plural/letters/data-is-plural-2017-05-31-edition",
        "https://www.europeandataportal.eu/data/en/dataset/https-data-stadt-zuerich-ch-dataset-pd_st
        "https://data.sunshinecoast.qld.gov.au/Administration/Registered-Animals/7f87-i6kx/data"],
      "hattips":"https://theodi.org/article/the-open-data-olympics-seven-weird-and-wonderful-open-da
      "id":"545",
      "linksCount":3}]
}}
```

Reviewing the dataset format

- Original XML is hard to read
- Best to reformat (I use XMLStarlet)
- Definitely not Solr format
- Can preprocess with XSLT (outside or inside Solr)
- Can import with Data Import Handler (DIH)
  - Custom field mapping
  - Can do data merge, nested requests
  - Has Admin UI screen
  - Good for quick analysis
  - Not production quality

```
<response>
  <row>
    <row _id="1673199" _uuid="02C4D22A-3FB6-4A44-AB2E-519EC093D64C"
      <animaltype>D          </animaltype>
      <name>UNKNOWN DOG NAME</name>
      <specificbreed>POODLETOY </specificbreed>
      <primarybreed>POODLE</primarybreed>
      <primarycolour>Grey          </primarycolour>
      <de_sexed>Y</de_sexed>
      <gender>F</gender>
      <age>31</age>
      <locality>NINDERRY</locality>
    </row>
    <row _id="1673200" _uuid="37E0BA1D-D1EA-425E-87EA-1F378FBFF9BB"
      <animaltype>D          </animaltype>
      <name>Cheer</name>
      <specificbreed>GERMNSHEP </specificbreed>
      <primarybreed>GERMNS</primarybreed>
      <primarycolour>BlackTan  </primarycolour>
      <de_sexed>Y</de_sexed>
      <gender>F</gender>
      <age>22</age>
      <locality>BRIDGES</locality>
    </row>
```

Analyzing the data needs

- Records are at **response/row/row**
- _id is an attribute, the rest are elements
- animaltype can be “d” and “Cat” - normalize to dog/cat
- Some elements have extra space – trim
- age should be a number
- primarycolour can have mixed colours – make it possible to search on sub-colour?
- Lowercase everything

```
<response>
  <row>
    <row _id="1673199" _uuid="02C4D22A-3FB6-4A44-AB2E-519EC093D64C"
      <animaltype>D          </animaltype>
      <name>UNKNOWN DOG NAME</name>
      <specificbreed>POODLETOY </specificbreed>
      <primarybreed>POODLE</primarybreed>
      <primarycolour>Grey        </primarycolour>
      <de_sexed>Y</de_sexed>
      <gender>F</gender>
      <age>31</age>
      <locality>NINDERRY</locality>
    </row>
    <row _id="1673200" _uuid="37E0BA1D-D1EA-425E-87EA-1F378FBFF9BB"
      <animaltype>D          </animaltype>
      <name>Cheer</name>
      <specificbreed>GERMNSHEP </specificbreed>
      <primarybreed>GERMNS</primarybreed>
      <primarycolour>BlackTan  </primarycolour>
      <de_sexed>Y</de_sexed>
      <gender>F</gender>
      <age>22</age>
      <locality>BRIDGES</locality>
    </row>
```

Defining the pipeline

- Data Import Handler (DIH)
  - Parse XML and map to field name
  - Rename **_id** to **id** and **de_sexed** to **desexed**
- Update Request Processors
  - Trim extra space on everything
  - Normalize D to Dog
- Schema definition
  - Default dynamic field – as before (lowercase!)
  - Explicit int field for **age**
  - Copy **primarycolour** into secondary field and split
- Can evolve one step at a time, starting from DIH
  - We are totally out of time for that today!
  - Home work (hint – enable DIH in solrconfig.xml first, then iterate)
- Copy **minimal** directory to **pets-final** to start new configset

Pets managed-schema (age)

- Modify *managed-schema*
- Keep all definitions there from *minimal* configset
- Add new parts anywhere in the file
- field type **int** and field definition for **age**

```
<fieldType name="pint" class="solr.IntPointField"
           docValues="true"/>

<field name="age" type="pint" default="0"
       indexed="true" stored="true"/>
```

Pets managed-schema (splitcolour type)

Define new field type to extract partial colour names (e.g. BlackWhite => Black, White)

```
<fieldType name="splitcolour_type" class="solr.TextField"
           positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.WordDelimiterFilterFactory"
            splitOnCaseChange="1" preserveOriginal="1"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Pets managed-schema (splitcolour field)

- Define new field **splitcolour** and copy from **primarycolour** to it

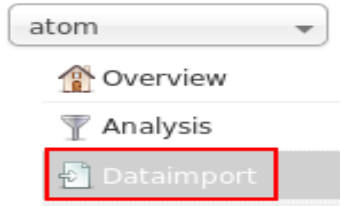
```
<field name="splitcolour" type="splitcolour_type"
       indexed="true" stored="false"/>

<copyField source="primarycolour"
              dest="splitcolour"/>
```

- Notice:
  - we have to search against **splitcolour** explicitly
  - it is not returned to user (stored=false)

Reference DIH example

- Use DIH/atom example for reference (one of 5)
- **`bin/solr start -e dih -p 9999`** (avoid port conflict)
  - http://localhost:9999/solr/#/atom/dataimport//dataimport
- **`bin/solr stop -p 9999`** (when done)
- <solr_install>/example/example-DIH/solr/atom/conf/*solrconfig.xml*
  - Order of elements is important for *solrconfig.xml*
  - Notice library requirement
  - Request Handler definition for **/dataimport**
  - **config** => *atom-data-config.xml* (DIH configuration)
  - Both params and Trim URP are right in solrconfig.xml (instead of params.json and configoverlay.json) – less flexible, though can still be overriden
- *atom-data-config.xml*
  - https://lucene.apache.org/solr/guide/7_4/uploading-structured-data-store-data-with-the-data-import-handler.html
  - Loads XML from URL Feed, not File (both are valid for URLDataSource)
  - Shows Transformers (we will stick to URPs)



Pets solrconfig.xml

```
<lib dir="${solr.install.dir:../../../..}/dist/"
     regex="solr-dataimporthandler-.*\.jar"/>
…
<requestHandler name="/dataimport" class="solr.DataImportHandler">
  <lst name="defaults">
    <str name="config">pets-data-config.xml</str>
    <str name="processor">trim_text,fix_dog</str>
  </lst>
</requestHandler>

<updateProcessor class="solr.processor.TrimFieldUpdateProcessorFactory"
                 name="trim_text" />

<updateProcessor class="solr.processor.RegexReplaceProcessorFactory"
                 name="fix_dog">
  <str name="fieldName">animaltype</str>
  <str name="pattern">D</str>
  <str name="replacement">Dog</str>
  <bool name="literalReplacement">true</bool>
</updateProcessor>
```

pets-data-config.xml

```
<dataConfig>
    <dataSource type="URLDataSource"/>
    <document>

        <entity name="pets"
            url="file://${solr.core.instanceDir}/../../pets/pets.xml"
            processor="XPathEntityProcessor"
            forEach="/response/row/row">

            <field column="id"             xpath="/response/row/row/@_id" />
            <field column="animaltype"     xpath="/response/row/row/animaltype" />
            <field column="name"           xpath="/response/row/row/name" />
            <field column="specificbreed"  xpath="/response/row/row/specificbreed" />
            <field column="primarybreed"   xpath="/response/row/row/primarybreed" />
            <field column="primarycolour"  xpath="/response/row/row/primarycolour" />
            <field column="desexed"        xpath="/response/row/row/de_sexed" />
            <field column="gender"         xpath="/response/row/row/gender" />
            <field column="age"            xpath="/response/row/row/age" />
            <field column="locality"       xpath="/response/row/row/locality" />

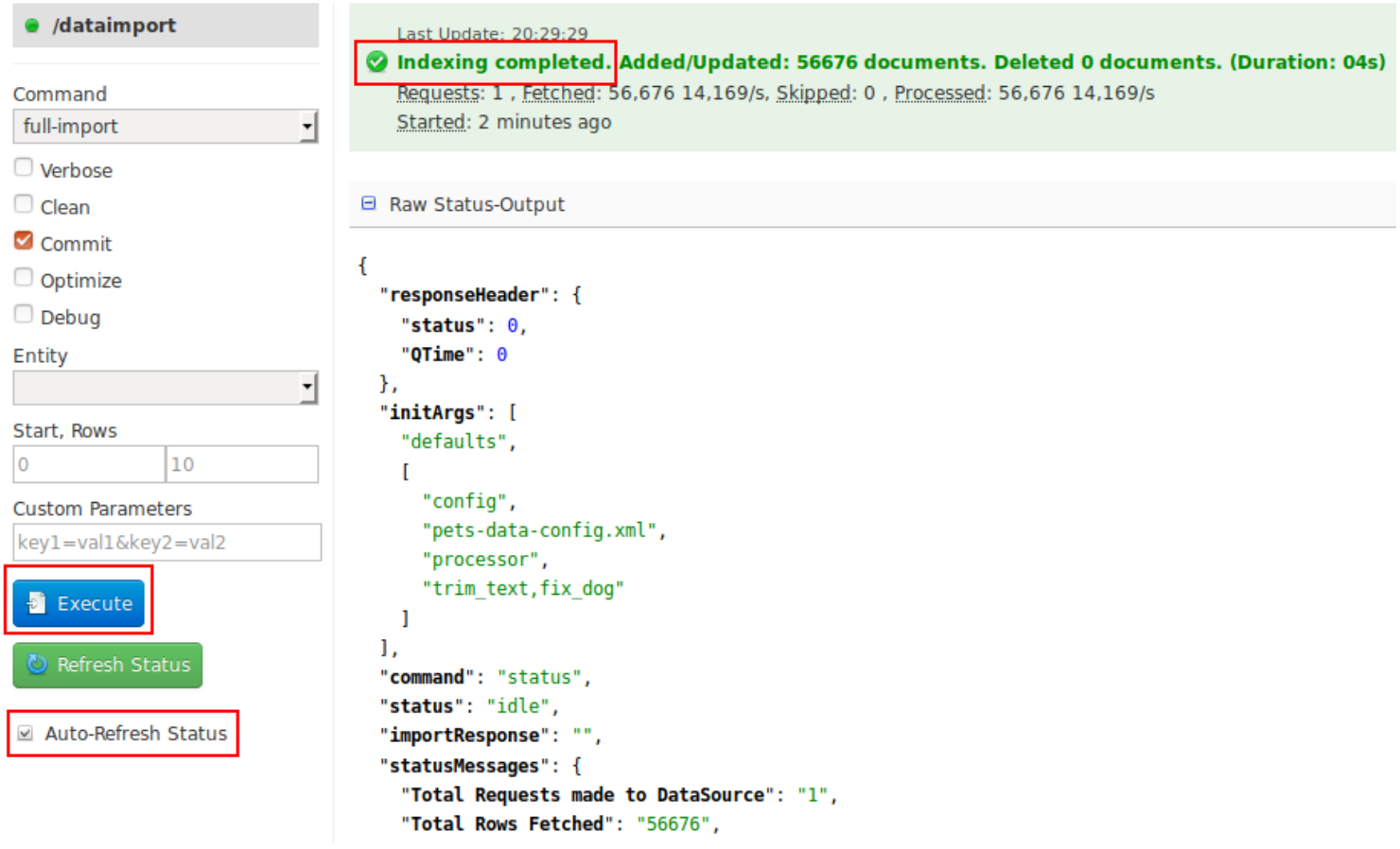
        </entity>

    </document>
</dataConfig>
```

Pets – create core

- **`bin/solr create -c pets -d …/pets-final`**
  - Create core in the running server with our custom configset
  - Active config files are now in *sroot/pets/conf*
  - Hack: you can edit these files and reload core
  - For more options: **`bin/solr create_core -h`**
- **`bin/solr delete -c pets`**
  - If you want to update configset and recreate core
  - Will delete modifications made while running

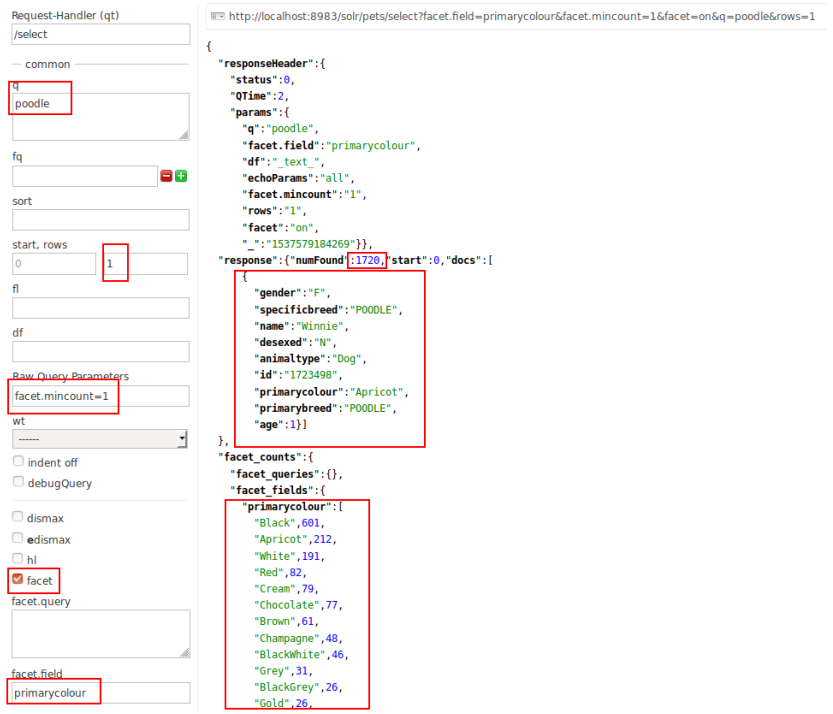
Import pets records with DIH



Basic search

http://localhost:8983/solr/pets/select?
facet.field=primarycolour&facet.mincount=1&facet=on&q=poodle&rows=1

- Case-insensitive search (poodle)
- Return only first record of 1720 found (of 56676 total)
- Show facets on *primarycolour* field
- Do not return facets with 0 matched documents



Merle puppy

- Merle – is a color
- There are several types of Merle
- Can we figure out types and popularity using **splitcolour** field?
  - q=splitcolour:**merle**
  - rows=0
  - facet=on
  - facet.field=primarycolour
  - facet.mincount=1

```
"facet_fields":{
  "primarycolour":[
    "BlueMerle",98,
    "Merle",67,
    "RedMerle",15]},
```

By Ted Van Pelt (Flickr, CC BY 2.0): https://www.flickr.com/photos/bantam10/5580029980/

Advanced analytics

- Basic queries and facets are good to start
- Recent Solr supports:
  - JSON Request API:
    https://lucene.apache.org/solr/guide/7_4/json-request-api.html
  - JSON Query DSL:
    https://lucene.apache.org/solr/guide/7_4/json-query-dsl.html
  - JSON Facets:
    https://lucene.apache.org/solr/guide/7_4/json-facet-api.html
  - Allows multi-leveled facets, analytics, (start of) semantic knowledge graph
- Can do very complex queries

Complex JSON query

```
{
  query: "splitcolour:gray",
  filter: "age:[0 TO 20]"
  limit: 2,
  facet: {
    type: {
    type: terms,
    field: animaltype,
    facet : {
      avg_age: "avg(age)",
      breed: {
        type: terms,
        field: specificbreed,
        limit: 3,
        facet: {
          avg_age: "avg(age)",
          ages: {
            type: range,
            field : age,
            start : 0,
            end : 20,
            gap : 5
}}}}}}}
```

- For all animals with a variation of gray colour
- Limited to those of age between 0 and 20 (to avoid dirty data docs)
- Show first two records and facets
- Facet them by animal type (Cat/Dog)
  - Then by the breed (top 3 only)
    - Then show counts for 5-year brackets
- On all levels, show bucket counts
- On bottom 2 levels, show average age

JSON Facets results

```
"facets":{
  "count":3168,
  "type":{
    "buckets":[{
      "val":"Cat",
      "count":1891,
      "avg_age":6.687...,
```

```
"breed":{
  "buckets":[{
    "val":"DOMSH",
    "count":951,

"avg_age":6.270241850683491,
      "ages":{
      "buckets":[
        {"val":0, "count":387},
        {"val":5, "count":358},
        {"val":10, "count":164},
        {"val":15, "count":41}]}},
```

```
{
  "val":"Dog",
  "count":1277,
  "avg_age":7.372748629600626,
  "breed":{
    "buckets":[{
      "val":"MALTESEX",
      "count":131,
      "avg_age":8.587786259541986,
      "ages":{
        "buckets":[{
          "val":0,
          "count":17},
        {
          "val":5,
          "count":62},
        {
          "val":10,
          "count":45},
        {
          "val":15,
          "count":7}]}},
```

THANK YOU

Alexandre Rafalovitch

@arafalov

arafalov@apache.org