# Hidden Gems

**Getting More Out Of Apache Solr**

**ApacheCon 2014 NA - 2014-04-08**

https://people.apache.org/~hossman/ac2014na

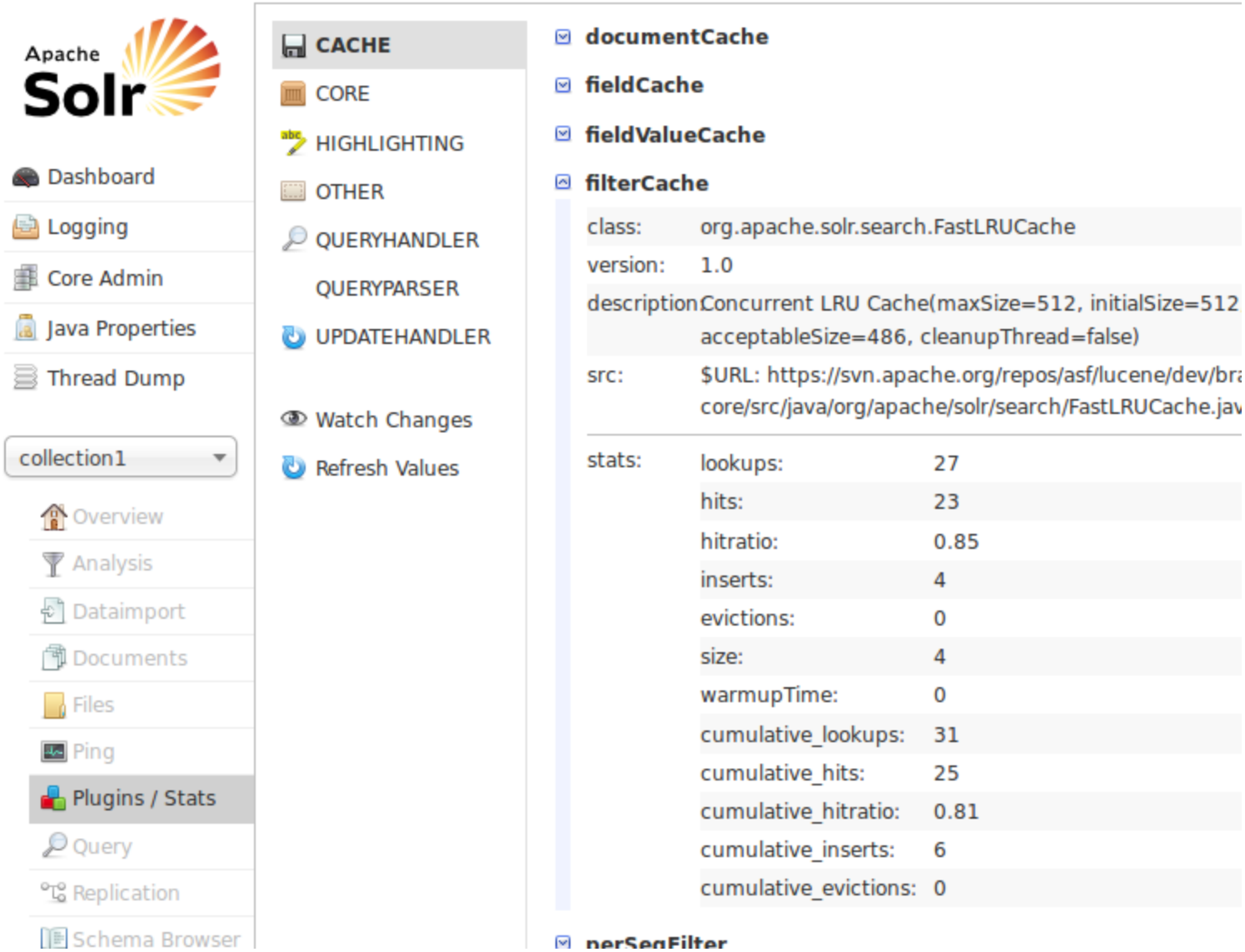
https://twitter.com/_hossman

http://www.lucidworks.com/



# Monitoring

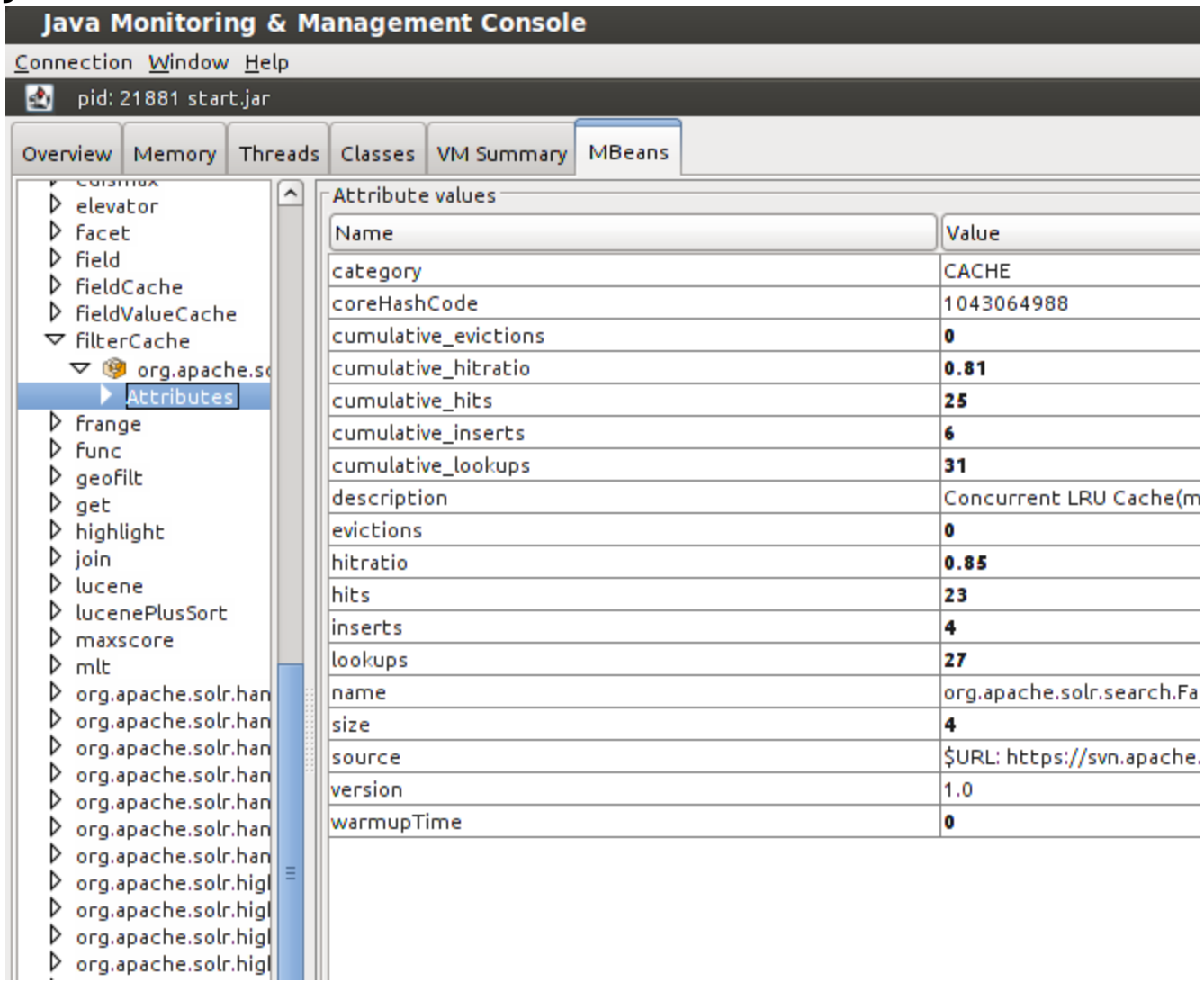
## Admin UI


Apache Solr
Dashboard
Logging
Core Admin
Java Properties
Thread Dump
collection1
Overview
Analysis
Dataimport
Documents
Files
Ping
Plugins / Stats
Query
Replication
Schema Browser
CACHE
CORE
HIGHLIGHTING
OTHER
QUERYHANDLER
QUERYPARSER
UPDATEHANDLER
Watch Changes
Refresh Values
documentCache
fieldCache
fieldValueCache
filterCache
class: org.apache.solr.search.FastLRUCache
version: 1.0
description: Concurrent LRU Cache(maxSize=512, initialSize=512
acceptableSize=486, cleanupThread=false)
src: $URL: https://svn.apache.org/repos/asf/lucene/dev/bra
core/src/java/org/apache/solr/search/FastLRUCache.jav
stats:
lookups: 27
hits: 23
hitratio: 0.85
inserts: 4
evictions: 0
size: 4
warmupTime: 0
cumulative_lookups: 31
cumulative_hits: 25
cumulative_hitratio: 0.81
cumulative_inserts: 6
cumulative_evictions: 0

Anything information you see in the Admin UI, is available programmatically to remote clients via Request Handlers or JMX.

Stefan Matheis provided a Great overview of how the Admin UI works @ LuceneRevolution 2013 (with Video).

## JMX


Java Monitoring & Management Console
Connection Window Help
pid: 21881 start.jar
Overview Memory Threads Classes VM Summary MBeans
elevator
facet
field
fieldCache
fieldValueCache
filterCache
org.apache.s
Attributes
frange
func
geofilt
get
highlight
join
lucene
lucenePlusSort
maxscore
mlt
org.apache.solr.han
org.apache.solr.han
org.apache.solr.han
org.apache.solr.han
org.apache.solr.han
org.apache.solr.han
org.apache.solr.han
org.apache.solr.higl
org.apache.solr.higl
org.apache.solr.higl
org.apache.solr.higl
Attribute values
Name
Value
category
CACHE
coreHashCode
1043064988
cumulative_evictions
0
cumulative_hitratio
0.81
cumulative_hits
25
cumulative_inserts
6
cumulative_lookups
31
description
Concurrent LRU Cache(m
evictions
0
hitratio
0.85
hits
23
inserts
4
lookups
27
name
org.apache.solr.search.Fa
size
4
source
$URL: https://svn.apache.
version
1.0
warmupTime
0

More details about using JMX in the Solr Reference Guide.

## HTTP Admin APIs


```
{
  "filterCache":{
    "stats":{
      "lookups":27,
      "hits":23,
      "hitratio":0.85,
      "inserts":4,
      "evictions":0,
      "size":4,
      "warmupTime":0,
      "cumulative_lookups":31,
      "cumulative_hits":25,
      "cumulative_hitratio":0.81,
      "cumulative_inserts":6,
      "cumulative_evictions":0}},
```

# facet.method

More details about using `facet.method` in the Solr Reference Guide.

## `fc` vs. `fcs`

- Both iterate over the matching documents and increment counters per-term
- `fc` (Default)
  - Single FieldCache (or UnInvertedField) over entire index
  - Typically faster look-ups than `fcs` once un-inverted structure is built
- `fcs`
  - FieldCache per index segment
  - Typically faster to re-build than `fc` in NRT situations -- only modified segments need built
  - Per-segment FieldCache is also used in sorting -- re-use in faceting may reduce total heap usage.

As with most things related to performance -- your experience will almost certainly vary from the observations of others. Always do some comparisons yourself using real data and realistic update/query patterns.

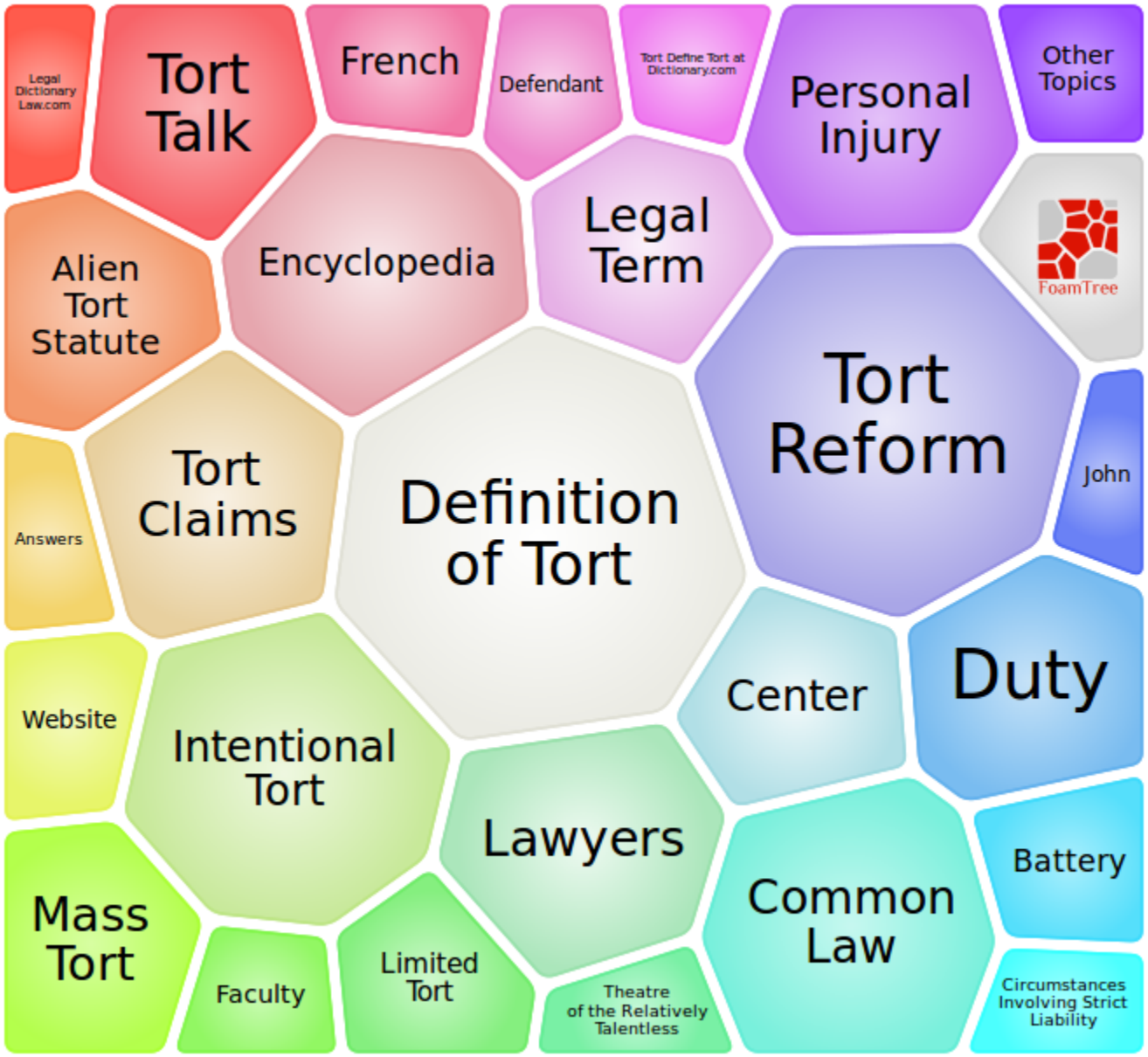
The most important thing folks should remember regarding the performance of `fc` & `fcs` is that using DocValues is probably a better choice than either of them.

## `enum`

- Enumerates all terms in the field and computes a set intersection with the matching documents
- Leverages the `filterCache`
- Small Cardinality Fields
  - Cached document sets may use less RAM than FilterCache
  - `fq` constraints on the same field will re-use the cached document sets
- High Cardinality Fields
  - FieldCache / UnInvertedField may not fit in RAM
  - Slower `enum` can still be used to get counts
  - Use `facet.enum.cache.minDf` to minimize `filterCache` churn
  - Can be used for faceting on full-Text fields to build tag clouds

# Result Clustering

aka: Dynamic Faceting


Legal Dictionary Law.com
Tort Talk
French
Defendant
Tort Define Tort at Dictionary.com
Personal Injury
Other Topics
Alien Tort Statute
Encyclopedia
Legal Term
FoamTree
Tort Reform
Tort Claims
Definition of Tort
John
Answers
Website
Intentional Tort
Center
Duty
Lawyers
Common Law
Battery
Mass Tort
Faculty
Limited Tort
Theatre of the Relatively Talentless
Circumstances Involving Strict Liability

Screenshot taken from the Carrot2's online demo of FoamTree's Voronoi treemap visualization tool.

## Clustering Component


```
{
  "clusters":[{
      "labels":["Environmental"],
      "score":6.393107732455205,
      "docs":["9781901362930", "9781841130903",
              "9781841130897", "9781841133607"]},
    {
      "labels":["Human Rights"],
      "score":15.667620783438327,
      "docs":["9781841130354", "9781841134574",
              "9781841136530"]},
    {
      "labels":["Anatomy of Tort Law"],
      "score":11.181459329239996,
      "docs":["9781901362091", "9781901362084"]},
    {
      "labels":["Litigation"],
      "score":8.560711128059928,
      "docs":["9781841132983", "9781841134574"]},
  ...
```


Details about Configuring & using Result Clustering in the Solr Reference Guide.

# Function Boosting & Personalized Scoring

At ApacheCon 2012 EU, I talked in depth about "Boosting & Biasing" using domain knowledge & user analytics (Video).

## Basic Function Boosting

```
      q = Nightfall Isaac Asimov
defType = edismax
  boost = div( popularity, add(1,price) )
```

```
      q = {!boost b=$my_func v=$qq}
     qq = +title:Nightfall author:"Isaac Asimov"
my_func = div( popularity, add(1,price) )
```

The `edismax` QParser has explicit support for a `boost` param that can be used to apply a multiplicity function boost, but the `boost` QParser can also be used to wrap any query type you can imagine.

If you aren't familiar with the "`{!qparser_name param=$variable}...`" syntax, you should take a look at "Local Params" in the Solr Reference Guide.

## Custom Category Boosts Per User

- Accumulate data on how much each of your users like/dislike various categories
- Batch process for every user:
  - A normalized "Z-Score" preference for each category
  - Record the 3 most significant (ie: greatest absolute value Z-Score) categories
- At query time:
  - Look-up the user's 3 most significant category scores
  - Use the Z-Scores as exponents in a boost function over those category queries

```
    qq = ...search terms...
     q = {!boost b=$b v=$qq}
     b = prod(pow( query($cat1), $z_cat1),
              pow( query($cat2), $z_cat2),
              pow( query($cat3), $z_cat3))
  cat1 = category:action                    # The user's 3 most significant categories,
z_cat1 = 1.48                               # ... and their Z-scores
  cat2 = category:comedy
z_cat2 = 1.33
  cat3 = category:kids
z_cat3 = -1.7
```

This specific example of personalized scores using categorical preferences comes from long time Solr user Amit Nithian via solr-user mailing list.

Amit is giving a talk tomorrow that I suspect will go into a lot more details on this sort of thing.

# Defaults, Appends, Invariants, ... Oh My!

## Lots of Options = Long URLs ?

```
http://server:8983/solr/collection_name/select?defType=edismax
  &qf=title^4+authors^3+description&pf2=title,author
  &boost=div(popularity,add(1,price))&sort=score+desc,+price+desc
  &fl=id,title,description,price&fq=instock:true
  &rows=100&start=0&q=Nightfall+Isaac+Asimov
```

## Lots of Options ≠ Long URLs !

```
http://server:8983/solr/collection_name/select?q=Nightfall+Isaac+Asimov
```

```
<lst name="defaults">
  <str name="defType">edismax</str>
  <str name="qf">title^4 authors^3 description</str>
  <str name="pf2">title, author</str>
  <str name="boost">div(popularity,add(1,price))</str>
  <str name="sort">score desc, price desc</str>
  <str name="fl">id,title,description,price</str>
  <str name="fq">instock:true</str>
  <int name="rows">100</int>
  <int name="start">0</int>
</lst>
```

Configuring request parameter `defaults` in your `solrconfig.xml` instead of in your clients, helps centralize your business logic about *how* you want your searches to work in a single place, so it's easier to change with out needing to modify all of your potential search client code. It also reduces the size of the HTTP requests, which may result in noticeable impacts on your network load & query throughput.

More details about using Default request parameters in the Solr Reference Guide.

## Prevent Client Mistakes

```
http://server:8983/solr/collection_name/select?q=Nightfall+Isaac+Asimov
```

```
<lst name="invariants">
  <str name="defType">edismax</str>
  <str name="qf">title^4 authors^3 description</str>
  <str name="pf2">title, author</str>
  <str name="boost">div(popularity,add(1,price))</str>
  <str name="sort">score desc, price desc</str>
  <int name="rows">100</int>
</lst>
<lst name="appends">
  <str name="fq">instock:true</str>
</lst>
<lst name="defaults">
  <str name="fl">id,title,description,price</str>
  <int name="start">0</int>
</lst>
```

Besides the benefits mentioned above regarding parameter `defaults`, Using `appends` and `invariantss` lets you enforce rules that client developers might "forget" to enforce themselves, or might not realize are important from a business / performance standpoint.

## Hide Implementation Details

```
/select?shipping=free_to_members&cat=books&q=Nightfall+Isaac+Asimov

 <lst name="invariants">
   <str name="cat_filter">{!term f=category v=$cat}</str>
 </lst>
 <lst name="appends">
   <str name="fq">instock:true</str>
   <str name="fq">{!switch case.any='*:*'
                           default=$cat_filter
                           v=$cat}
   </str>
   <str name="fq">{!switch case.any='*:*'
                           case.free_to_members='member_shipping:0.0'
                           case.free='shipping_cost:0.0'
                           v=$shipping}
   </str>
 </lst>
 <lst name="defaults">
   <str name="cat">any</str>
   <str name="shipping">any</str>
 </lst>
```

More details about using `switch` QParser in the Solr Reference Guide.

# Hierarchical Documents

(aka: Block Join)

A classic IR problem that is Block Joining solves is the idea of very large text documents (eg: textbooks), divided up into hierarchical chunks (eg: volume, part, section, chapter) and you want to be able to find books that contain a chapter with some specific criteria *and* another chapter with some different specific criteria.

## Nested Documents

```
<add>
  <doc>
    <field name="id">100</field>
    <field name="doctype">album</field>
    <field name="album_name">Wayne's World (soundtrack)</field>
    <doc>
      <field name="id">101</field>
      <field name="doctype">song</field>
      <field name="song_name">Bohemian Rhapsody</field>
      <field name="artist_name">Queen</field>
    </doc>
    <doc>
      <field name="id">102</field>
      <field name="doctype">song</field>
      <field name="song_name">Hot and Bothered</field>
      <field name="artist_name">Cinderella</field>
    </doc>
    ...
  </doc>
  ...
</add>
```

JSON Syntax for indexing document blocks was added by SOLR-5183 *after* Solr 4.7 was released, and will be included in Solr 4.8.

More details about using Block Joins in the Solr Reference Guide

## "Soundtrack" Albums

```
/select?q=soundtrack&fq=doctype:album
```


```
 { "response":{"numFound":3,"start":0,"docs":[
       {
         "id":"100",
         "album_name":"Wayne's World (soundtrack)"},
       {
         "id":"200",
         "album_name":"Empire Records (Soundtrack)"},
       {
         "id":"300",
         "album_name":"Reality Bites (Soundtrack)"}]
   }}
```

## "Love" Songs

```
/select?q=love&fq=doctype:song
```


```
 { "response":{"numFound":7,"start":0,"docs":[
       { "id":"114",
         "song_name":"Loud Love",
         "artist_name":"Soundgarden"},
       { "id":"112",
         "song_name":"Loving Your Lovin'",
         "artist_name":"Eric Clapton"},
       { "id":"406",
         "song_name":"One Year of Love",
         "artist_name":"Queen"},
       { "id":"503",
         "song_name":"Ready For Love",
         "artist_name":"Bad Company"},
       { "id":"532",
         "song_name":"Hammer of Love",
         "artist_name":"Bad Company"},
       ...,
```

## Soundtracks containing "Love" Songs

```
/select?q=soundtrack&fq={!parent which="doctype:album"}love
```


```
 { "response":{"numFound":2,"start":0,"docs":[
       {
         "id":"100",
         "album_name":"Wayne's World (soundtrack)"},
       {
         "id":"300",
         "album_name":"Reality Bites (Soundtrack)"}]
   }}
```

## "Love" Songs on Soundtracks

```
/select?q=love&fq={!child of="doctype:album"}soundtrack
```


```
 { "response":{"numFound":3,"start":0,"docs":[
       {
         "id":"114",
         "song_name":"Loud Love",
         "artist_name":"Soundgarden"},
       {
         "id":"112",
         "song_name":"Loving Your Lovin'",
         "artist_name":"Eric Clapton"},
       {
         "id":"314",
         "song_name":"Baby, I Love Your Way",
         "artist_name":"Big Mountain"}]
   }}
```

## Hiding the Details

```
/songs
/songs?song=love
/songs?album=soundtrack
/songs?song=love&album=soundtrack
```

```
<requestHandler name="/songs" class="solr.SearchHandler">
  <lst name="invariants">
    <str name="album_filter">doctype:album</str>
    <str name="songs_by_album">{!child of=$album_filter v=$album_query}</str>
    <str name="song_query">{!df=song_name_t v=$song}</str>
    <str name="album_query">{!df=album_name_t v=$album}</str>
    <str name="q">{!switch case='*:*'
                           default=$song_query
                           v=$song}
    </str>
  </lst>
  <lst name="appends">
    <str name="fq">{!switch case='doctype:song'
                            default=$songs_by_album
                            v=$album}
    </str>
  </lst>
</requestHandler>
```

Remember what I was saying about simplifying requests using `defaults` and the `switch` QParser?

- All Songs
- "Love" Songs
- Songs on "Soundtrack" Albums
- "Love" Songs on "Soundtrack" Albums

A similar `/albums` handler could be configured with the appropriate (reversed) rules.

## Block Join Caveats

- Fairly new feature, still evolving (SOLR-5142)
- Currently only supported as constant score queries
- Special `_root_` field needed to handle deletes when updating a block:
  - Currently has a bug if you "update" a parent document to have no children (SOLR-5211)
  - Doesn't play nicely with deleting by id -- need to use delete by query to ensure all children are removed
- Coming Soon: Option to include nested child documents in search results (SOLR-5285)

The constant score limitation should be easy to fix - I think it's just a hard coded simplification that needs a new local param option.

Deleting a child doc (that has no children of it's own) by ID should work fine -- the specific problem comes in when you try to delete a parent doc by id: it will orphan the children and lead to non-deterministic behavior.

# Update Processors

## Pipeline Of Reusable Tools

```
<processor class="solr.CloneFieldUpdateProcessorFactory">
  <arr name="source">
    <str>authors</str>
    <str>editors</str>
  </arr>
  <str name="dest">contributors</str>
</processor>
<processor class="solr.ConcatFieldUpdateProcessorFactory">
  <str name="delimiter">; </str>
  <str name="fieldName">contributors</str>
  </lst>
</processor>
<processor class="solr.CloneFieldUpdateProcessorFactory">
  <str name="source">authors</str>
  <str name="dest">primary_author</str>
</processor>
<processor class="solr.FirstFieldValueUpdateProcessorFactory">
  <str name="fieldName">primary_author</str>
</processor>
```

In this example, imagine that our input documents contain only the (multi-valued) authors and editors fields. We build up the (single valued) contributors string field by combining the two, and we populate the primary_author field using the first value from the authors field.

There are a lot more Update Processors like these. More info about these specific Update Processors:

- CloneFieldUpdateProcessorFactory
- ConcatFieldUpdateProcessorFactory
- FirstFieldValueUpdateProcessorFactory

## Script Your Own

```
<processor class="solr.StatelessScriptUpdateProcessorFactory">
  <str name="script">update-script.js</str>
  <lst name="params">
    <int name="min_popularity">42</int>
  </lst>
</processor>


// in update-script.js
function processAdd(cmd) {
  doc = cmd.solrDoc;  // org.apache.solr.common.SolrInputDocument

  if (params.get("min_popularity") < doc.getFieldValue("popularity")) {
    doc.addField("is_hot","true");
  }
}
```

If the existing Update Processors don't do what you need, you can write your own in Java -- or in any scripting language supported by your JVM.

# Q & A

- Me
  - https://twitter.com/_hossman
- My Company
  - http://www.lucidworks.com/
- These Slides
  - https://people.apache.org/~hossman/ac2014na
- Solr Docs
  - https://lucene.apache.org/solr/documentation.html
- Mailing Lists & IRC
  - https://lucene.apache.org/solr/discussion.html
- Join The Revolution in DC, November 11-14
  - http://www.lucenerevolution.org/2014/call-for-speakers