Lucene & Solr

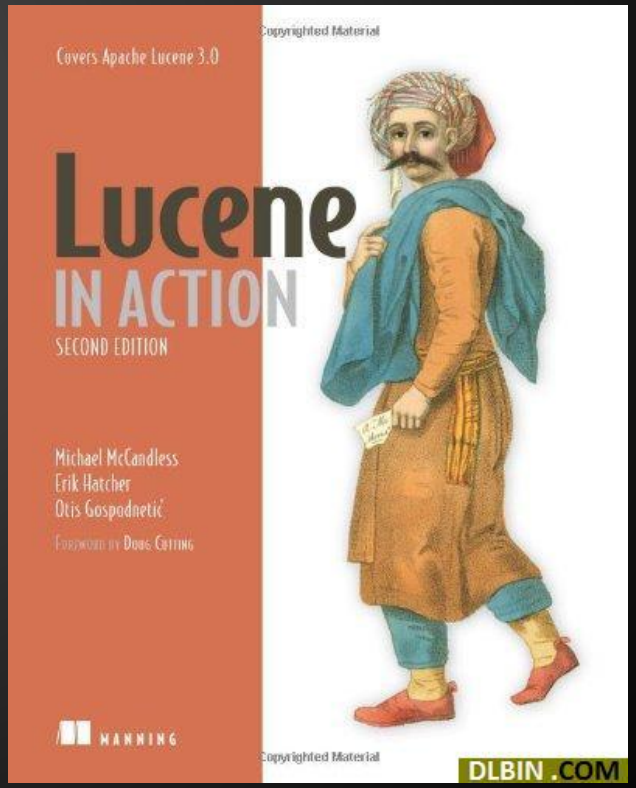
Overview

- Lucene
  - Indexing and Searching
  - Index Format
- Solr
  - Configuration
  - Administration
  - Clustering (SolrCloud)

Based on

Manning: Lucene in Action,

and Solr Refernce Guide



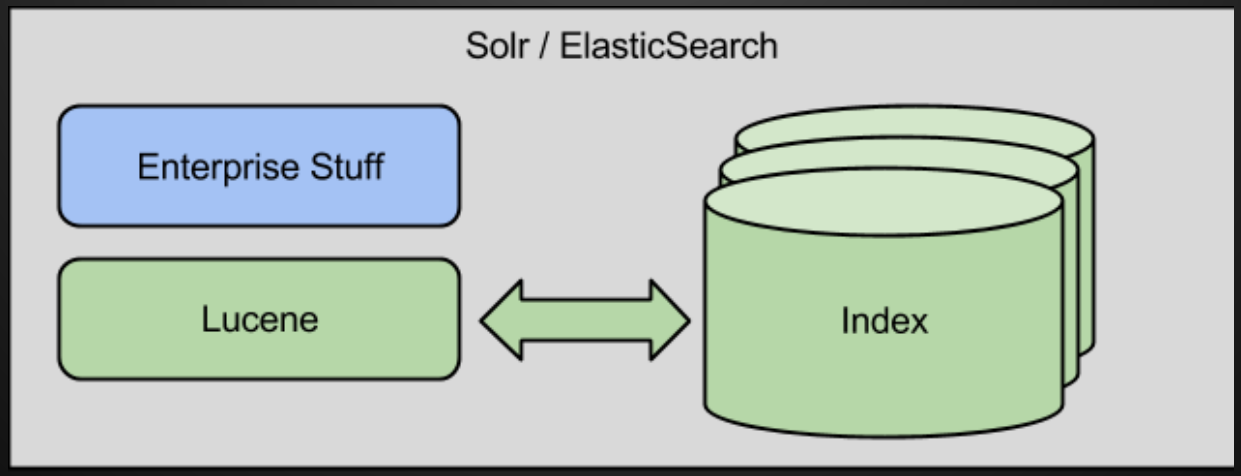
Demo Project

```
$ git clone git@github.com:serprime/solr-lucene-demo.git
```

- Lucene examples work out of the box
  - run the main() of each class from IDE
- Solr examples might need a running server
  - Download the solr distribution from:
    - http://lucene.apache.org/solr/mirrors-solr-latest-redir.html

Lucene, Solr, ElasticSearch?

- Lucene
  - just a library
  - handle index + query
- Solr and ElasticSearch
  - Enterprise Search Server
  - use Lucene



Solr VS. ElasticSearch

- Solr
  - by Yonik Seeley
  - 2007 graduated from incubating as Apache project
  - 2008 Solr 1.3
  - **2009 Solr 1.4**
  - 2014 Solr 4.10
  - 2015 Solr 5.0
- ElasticSearch
  - by Shay Banon
  - 2004 Compass started
  - 2010 ElasticSearch was released, as a rewrite of Compass

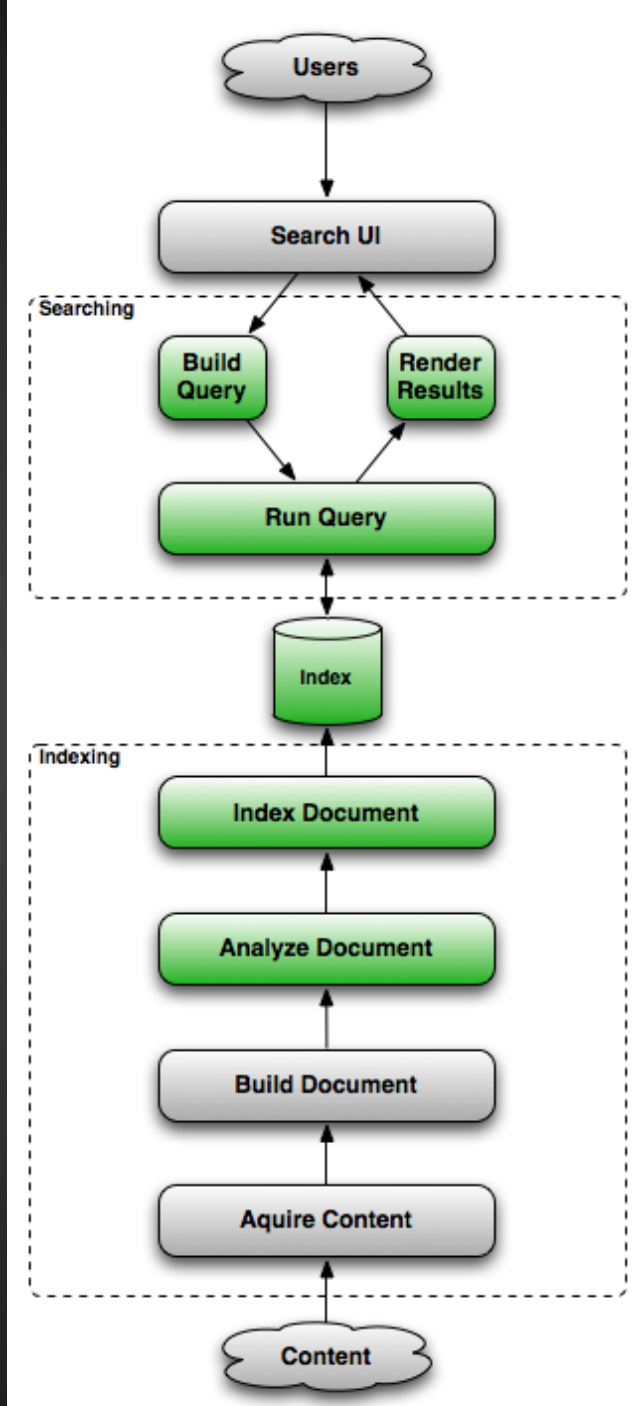
Part 1: Lucene

Project Details

- **Doug Cutting** started Lucene in 1999
  - Also started **Hadoop** and co-started **Nutch**
- Projects that use Lucene:
  - https://wiki.apache.org/lucene-java/PoweredBy
  - NetFlix
  - MySpace
  - LinkedIn
  - Fedex
  - Apple
  - PoolParty

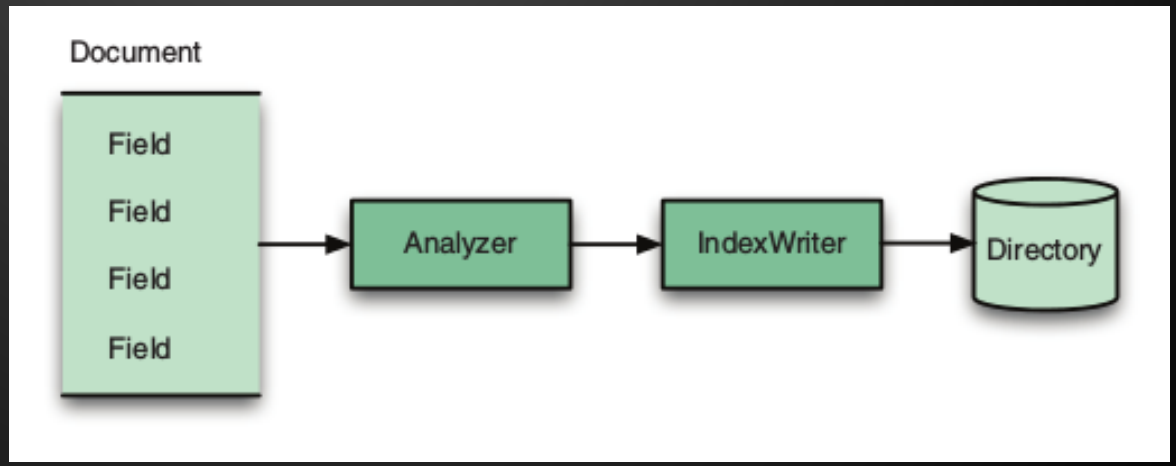
Components

- Lucene:
  - Indexing
    - Analyze Document
    - Index Document
  - Searching
    - Build Query
    - Run Query
    - Render Results



Indexing Classes

- Example: Lucene1Indexer
  - Document + Field
  - Analyzer
  - IndexWriter
  - Directory



Document & Field

- Document
  - atomic unit to store and retrieve from index
  - is a collection of fields
- Field
  - has a name
  - and a value
  - and a type
  - and options

Analyzer

- Analyzer
  - passed to IndexWriter
  - converts content to indexable format
    - stream of tokens
  - optional operations
    - filter stopwords
    - add synonyms
    - lower case
    - ngrams
    - ...

IndexWriter & Directory

- IndexWriter
  - create new index
  - open existing index
  - add, remove, update documents
  - writes to a Directory
- Directory
  - abstract class to access index files
  - other implementations:
    - RAM-based indices
    - indices stored in a database, via JDBC
    - index as a single file

Field Options

- Documents & Fields
  - indexed
    - indexing of fields is optional
    - to search for fields they must be indexed
  - term vectors
    - if indexed, term vectors might be stored
    - like an inverted index for one field
    - used for fast highlighting and “more like this”
  - stored
    - the plain unanalyzed value might be stored
    - not used for search
    - but to get the original value in the results

Schema

- none
  - each document can have different fields
  - with different options
- Denormalization
  - data has to be flattened
  - no joins like in relational dbs

Searching Classes

```
Directory dir = FSDirectory.open(new File("/tmp/index"));
IndexSearcher searcher = new IndexSearcher(dir);
Query q = new TermQuery(new Term("contents", "lucene"));
TopDocs hits = searcher.search(q, 10);
```

- IndexSearcher
  - opens an index read only
  - can evaluate a query against index

Searching Classes (2)

```
Directory dir = FSDirectory.open(new File("/tmp/index"));
IndexSearcher searcher = new IndexSearcher(dir);
Query q = new TermQuery(new Term("contents", "lucene"));
TopDocs hits = searcher.search(q, 10);
```

- Query
  - can be built manually
  - or by using QueryParser

Searching Classes (3)

```
Directory dir = FSDirectory.open(new File("/tmp/index"));
IndexSearcher searcher = new IndexSearcher(dir);
~~Query q = new TermQuery(new Term("contents", "lucene"));~~
QueryParser parser =
     new QueryParser("contents", new StandardAnalyzer());
Query q = parser.parse("lucene query");
TopDocs hits = searcher.search(q, 10);
```

- QueryParser
  - Defines default search field (`"contents"`)
  - Analyzer to prepare search tokens to match indexed tokens

Searching Classes (4)

```
Directory dir = FSDirectory.open(new File("/tmp/index"));
IndexSearcher searcher = new IndexSearcher(dir);
Query q = new TermQuery(new Term("contents", "lucene"));
TopDocs hits = searcher.search(q, 10);
```

- TopDocs
  - result set
  - references to top n results
  - with scores
- Example: Lucene2Searcher

Lucene Query Syntax

The query … matches documents that ...

- **java**
  - contain ‘java’ in the default field
- **java OR junit**
  - contain ‘java’ or ‘junit’ in the default field
- **+java +junit**
- **java AND junit**
  - contain ‘java’ and ‘junit’ in the default field
- **title:ant**
  - contain ‘ant’ in the field title
- **title:extreme –subject:sports**
- **title:extreme AND NOT subject:sports**
  - contain ‘extreme’ in title, but not ‘sports’ in the subject
- **(agile OR extreme) AND methodology**

Lucene Query Syntax (2)

The query … matches documents that ...

- **title:"junit in action"**
  - contains the phrase ‘junit in action’ in the title
- **title:"junit action"~5**
  - contains the terms ‘junit’ and ‘action’ within a range of 5 tokens
- **java***
  - contains ‘java’ but also: ‘javaspaces’, ‘javaserver’
- **java~**
  - contain ‘lava’ => fuzzy search
- **lastmodified:[1/1/09 TO 12/31/09]**
  - … date range query

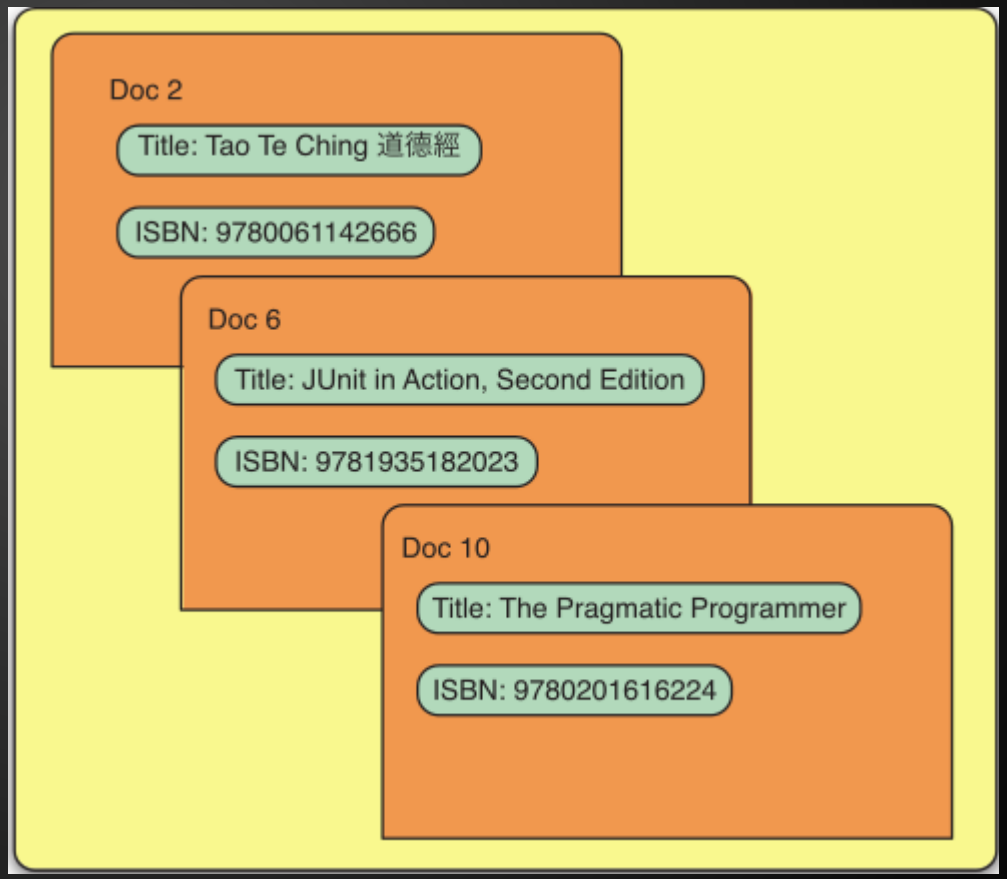
ACID

- Lucene supports full ACID
  - there can only be one writer
  - reader can only see added commited data
- Delete
  - deleted docs are marked
  - space is freed on optimize

Index Format

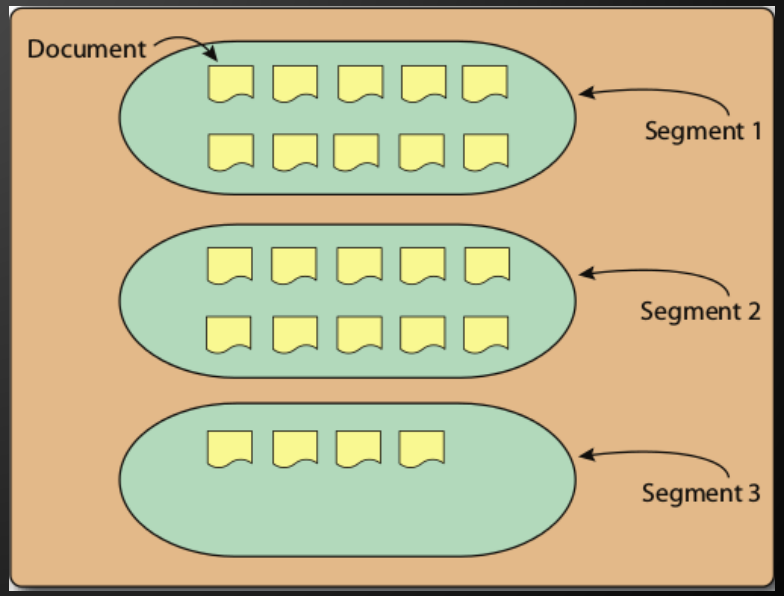
Logical View

- Documents
- with fields
- and values



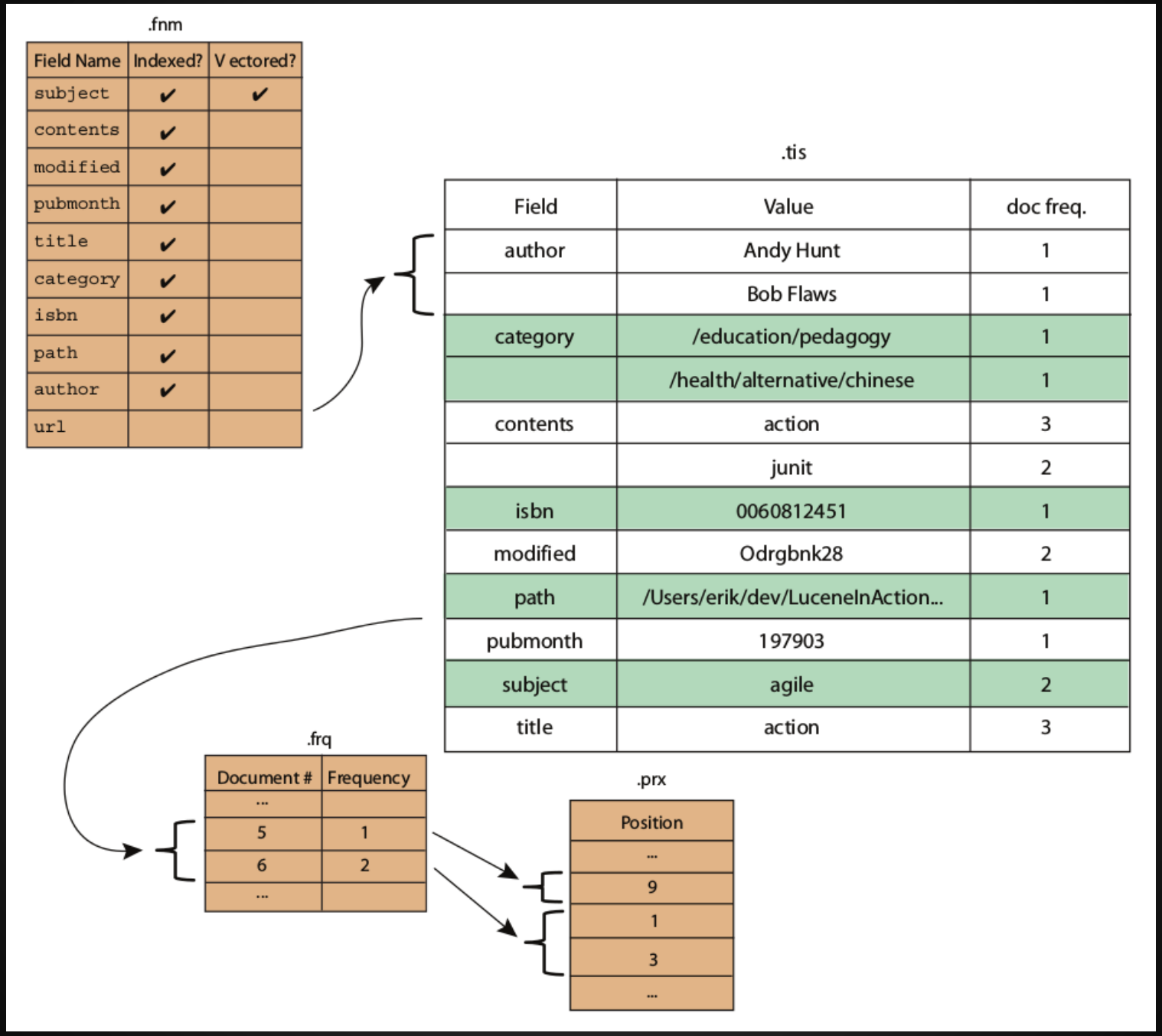
Index Segments

- Index consists of 1 or more segments
- A segment is (kind of) a subindex
- A segment contains one or more documents
- Enables incremental indexing
- A commit add a new segment
- Optimize merges segments



Index Files

- Field Names (**fnm**)
  - contains all used field names in a segment
  - stores options used on fields
    - stored? indexed? term vectors? ...
- Term Dictionary (**tis**)
  - all terms stored
  - sorted alphabetically by field name and value
- Frequencies (**frq**)
  - term frequency per document
- Positions (**prx**)


.fnm
Field Name
Indexed?
V ectored?
subject
contents
modified
pubmonth
title
category
isbn
path
author
url
.tis
Field
Value
doc freq.
author
Andy Hunt
1
Bob Flaws
1
category
/education/pedagogy
1
/health/alternative/chinese
1
contents
action
3
junit
2
isbn
0060812451
1
modified
Odrgbnk28
2
path
/Users/erik/dev/LuceneInAction...
1
pubmonth
197903
1
subject
agile
2
title
action
3
.frq
Document #
Frequency
...
5
1
6
2
...
.prx
Position
...
9
1
3
...

Scoring

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d) \right)$$

- Score is computed for each document **(d)** matching each term **(t)** in a query **(q)**
  - tf(t in d)
    - term freq. factor
  - idf(t)
    - inverse document freq. - measure for uniqueness
  - t.getBoost()
    - index time boost value of field
  - norm(t,d)
    - normalization: shorter fields get bigger boosts

Scoring (2)

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d) \right)$$

- Score is computed for each document **(d)** matching each term **(t)** in a query **(q)**
  - coord(q,d)
    - the more matching **t**s, the higher the boost for **d**
  - queryNorm(q)
    - normalization: sum of squared weights
- Score is:
  the relevancy of a document to a query

Explain

- Run Lucene3Explain
- Output

```
docs/Solr.txt
0.30901012 = (MATCH) weight(contents:solr in 0)
  [DefaultSimilarity], result of:
  0.30901012 = fieldWeight in 0, product of:
    4.690416 = tf(freq=22.0), with freq of:
      22.0 = termFreq=22.0
    1.4054651 = idf(docFreq=9, maxDocs=15)
    0.046875 = fieldNorm(doc=0)
```

Analyzer

- Converts content to indexable tokens
  - TokenStream
- Core Analyzers
  - WhitespaceAnalyzer
  - SimpleAnalyzer
  - StopAnalyzer
  - StandardAnalyzer

Analyzer (2)

“The quick brown fox jumped over the lazy dog”

- WhitespaceAnalyzer:
  [The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]
- SimpleAnalyzer:
  [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]
- StopAnalyzer:
  [quick] [brown] [fox] [jumped] [over] [lazy] [dog]

Analyzer (3)

“XY&Z Corporation - xyz@example.com”

- WhitespaceAnalyzer:
  [XY&Z] [Corporation] [-] [xyz@example.com]
- SimpleAnalyzer:
  [xy] [z] [corporation] [xyz] [example] [com]
- StandardAnalyzer:
  [xy&z] [corporation] [xyz@example.com]

Analyzer Demo

Example: Lucene4Analyzer

- different Analyzers create different TokenStreams
- Custom Analyzer: EdgeNGramAnalyzer
  - extends Analyzer
  - creates ClassicTokenizer from Reader
  - chains filter from source
    - ```
      TokenStream filter = new LowerCaseFilter(source);
      filter = new EdgeNGramTokenFilter(filter, 1, 10);
      ```
  - returns TokenStreamComponents
      ```

Analyzer Demo (2)

Output:

```
WhitespaceAnalyzer:
 1: [The]                  0->  3 word
 2: [quick]                4->  9 word
 3: [brown]               10-> 15 word
 4: [fox]                 16-> 19 word
 5: [jumped]              20-> 26 word
 6: [over]                27-> 31 word
 7: [the]                 32-> 35 word
 8: [lazy]                36-> 40 word
 9: [dog]                 41-> 44 word
```

Analyzer Demo (3)

Output:

```
NGramAnalyzer:
 1: [t]                    0->  3 word
 1: [th]                   0->  3 word
 1: [the]                  0->  3 word
 2: [q]                    4->  9 word
 2: [qu]                   4->  9 word
 2: [qui]                  4->  9 word
 2: [quic]                 4->  9 word
 2: [quick]                4->  9 word
```

Analyzer Demo (4)

Output:

```
ClassicAnalyzer:
 1: [i'll]                 0->  4 <APOSTROPHE>
 2: [email]                5-> 10 <ALPHANUM>
 3: [you]                 11-> 14 <ALPHANUM>
 5: [xyz@example.com]     18-> 33 <EMAIL>
```

Per Field Analyzer

To analyze each field differently

- Date, title, text body, numbers, emails, …
- Create PerFeldAnalyzerWrapper:

```
Map<String, Analyzer> analyzerPerField = new HashMap<>();
analyzerPerField.put("filename", new KeywordAnalyzer());
analyzerPerField.put("contents", new StandardAnalyzer());
PerFieldAnalyzerWrapper analyzer =
  new PerFieldAnalyzerWrapper(new SimpleAnalyzer());
```

Part 2: Solr

Solr Quick Start Guide

- Follow http://lucene.apache.org/solr/quickstart.html
- Overview
  - Get distribution
  - Start Solr
  - Visit admin dashboard
  - Index data
  - Searching
    - basics
    - facets
    - spatial

Quick Start

- Download solr distribution

  http://lucene.apache.org/solr/mirrors-solr-latest-redir.html

  sftp://file1.semantic-web.at/mnt/data/fileserver/Entwicklung/lucene-solr
- Unpack
- Start Solr

```
:$ jar                              ;check that jar is on PATH
:$ java -version                    ;check that java is >=1.7
:$ cd solr-4.10.3/
solr-4.10.2:$ bin/solr start -e cloud -noprompt
```

Quick Start (2)

- Other examples

```
-e <example>  Name of the example to run; available
examples:
      cloud:          SolrCloud example
      default:        Solr default example
      dih:            Data Import Handler
      schemaless:     Schema-less example
      multicore:      Multicore
```

Quick Start (3)

- Index Data

```
$ export CLASSPATH=dist/solr-core-4.10.2.jar
$ java -Dauto org.apache.solr.util.SimplePostTool
                example/exampledocs
```

Quick Start (4)

- go to http://localhost:8983/solr/collection1/browse
  - query
  - facets
  - pivot facets
  - range facets
  - geo spatial
- admin ui http://localhost:8983/solr/
  - analysis
  - query
  - schema browser

Solr Plugins

- Special search components
  - Spell checker
    - Did you mean …?
  - Highlighting
    - Foo Bar
  - Clustering
    - http://search.carrot2.org/stable/search?query=solr&results=100&source=web&view=foamtree
  - Phonetic search
    - cool, kool
    - similar names

Folder Structure

- Simple Solr folder
  - example/solr/
    - solr.xml
      - former definition of cores
    - zoo.cfg
      - config for Zookeeper in SolrCloud
    - collection1
      - folder for core “collection1”

Core Folder

- collection1
  - conf
    - solrconfig.xml
    - schema.xml
    - (many more for handlers)
  - data
    - index
      - lucene index
    - tlog
      - transaction log?
    - core.properties
      - (can be empty)

Core Discovery

- Since 4.4 cores are no longer defined in solr.xml
- At startup solr searches for cores in
  - `-Dsolr.solr.home`
- Each folder containing a cores.properties file will be added as core
- Process is recursive
- Subdirectories of cores will not be searched

Schema.xml

- Fields
  - fieldName
  - type
  - flags:
    - indexed
    - stored
    - multiValued
    - termVectors
    - …
  - dynamicField
  - copyField
  - uniqueKey

Schema.xml (2)

- FieldType
  - name
    - referenced by fields
  - class
    - solr.StrField
    - solr.DateField …
  - analyzer
    - tokenizer + list of filters
  - different query / indexing analyzer
    - <analyzer type=”index”>...</>
    - <analyzer type=”query”>...</>

Solrconfig.xml

- lucene Match Version
  - let solr mimic an older version (including bugs)
- query / index config
- Define Request Handler
  - Search
  - Update
  - Similarity
- Query Response Writer
  - json / xml / bin
- default Query
  - *:*

SolrJ

- Java client library for Solr
- Example code in Solr1Client.java
  - Simple query
  - Print total results
  - Iterate over results and print name field

Solrj POJOs

- @Field Annotation

```
class Foo {
  @Field String id;
  @Field(“fieldName”) String field;
}

solrServer.addBean(new Foo(...));
List<Foo> foos = solrServer.query().getBeans(Foo.class);
```

SolrJ UpdateHandler

- For adding many documents
- ConcurrentUpdateSolrServer
  - buffers all added documents
  - send in concurrent http connections

Fin