

# Information Access with Apache Lucene

Metodi per il Ritrovamento dell'Informazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Marco Polignano

[marco.polignano@uniba.it](mailto:marco.polignano@uniba.it)

# Code Repository & Requirements

## Code repository

[https://github.com/marcopoli/MRI\\_2022\\_23](https://github.com/marcopoli/MRI_2022_23)

## Requirements

- Java SDK 1.8+
- IDE: NetBeans, IntelliJ, Eclipse

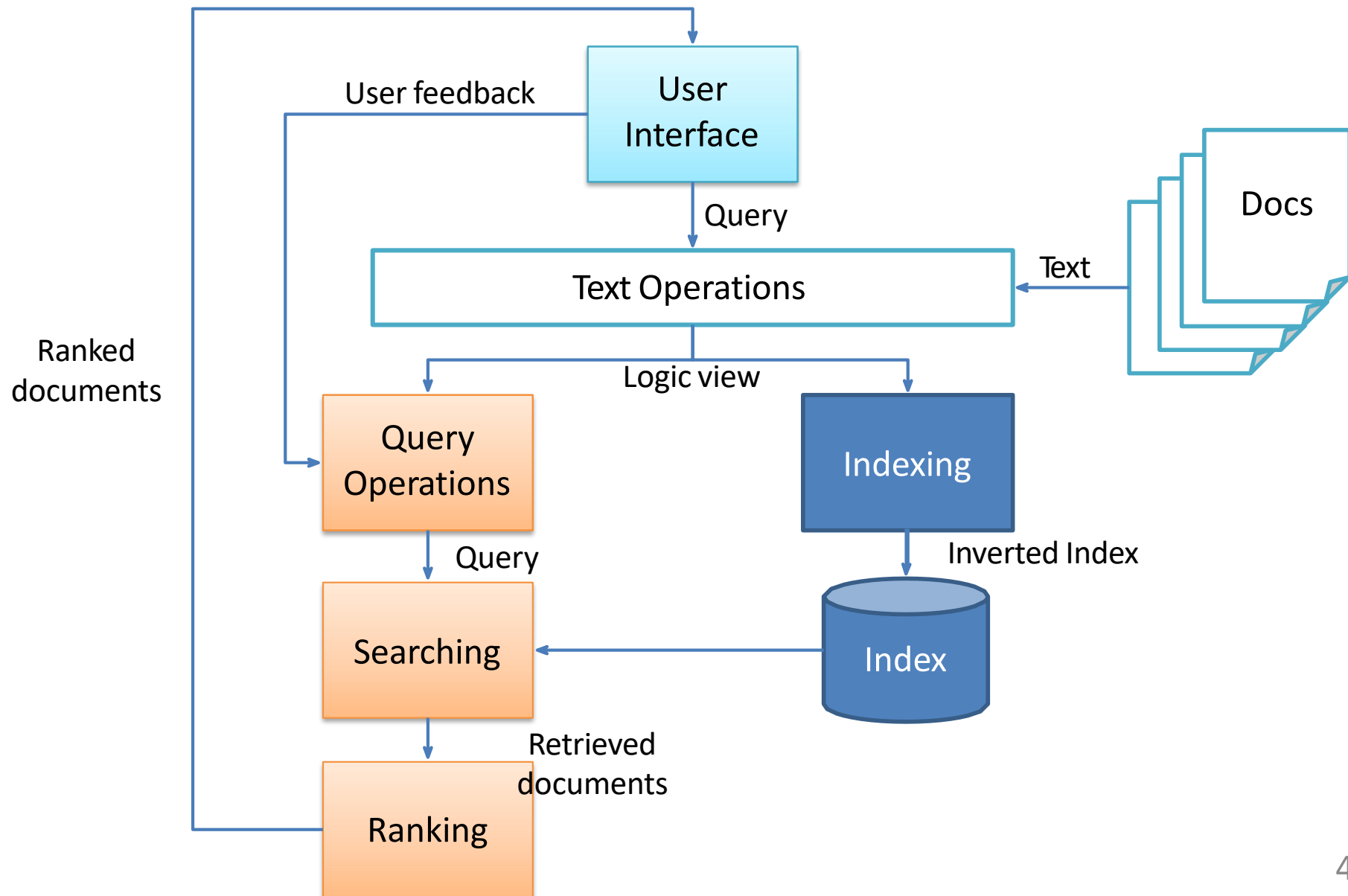
- **Maven:**

<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

Recap

# **SEARCH ENGINE**

# Information Retrieval Process



# Information Retrieval Model

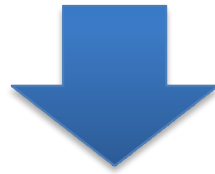
$\langle D, Q, F, R(q_i, d_j) \rangle$

- D: document representation
- Q: query representation
- F: query/document representation framework
- $R(q_i, d_j)$ : ranking function

# Bag-of-words representation

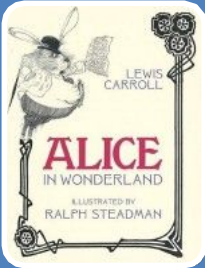
Document/query as unordered collection of words

John likes to watch movies. Mary likes too. John also likes to watch football games.

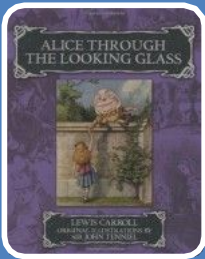


```
{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6,  
"football": 7, "games": 8, "Mary": 9, "too": 10}
```

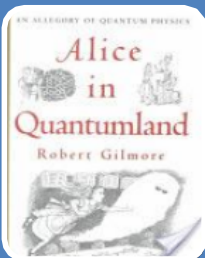
# Term-Document matrix



- 'It's a friend of mine — a **Cheshire Cat**,' said **Alice**: 'allow me to introduce it.'
- 'It's the oldest rule in the **book**,' said the **King**. 'Then it ought to be Number One,' said **Alice**.



- **Alice** watched the White **King** as he slowly struggled up from bar to bar, till at last she said, 'Why, you'll be hours and hours getting to the **table**, at that rate.'
- **Alice** looked round eagerly, and found that it was the Red **Queen**. 'She's grown a good deal!' was her first remark.



- In the pool of light was a billiards **table**, with two figures moving around it. **Alice** walked toward them, and as she approached they turned to look at her.
- **Alice** lay back, and closed her eyes. There was the Red **Queen** again, with that incessant **grin**. Or was it the **Cheshire cat's grin**?

# Term-Document matrix

	D1	D2	D3
Cheshire Cat	1	0	1
Alice	2	2	2
book	1	0	0
King	1	1	0
table	0	1	1
Queen	0	1	1
grin	0	0	2



# Term-Document matrix

	D1	D2	D3
Cheshire Cat	1	0	1
Alice	2	2	2
book	1	0	0
King	1	1	0
table	0	1	1
Queen	0	1	1
grin	0	0	2

Query:

Alice AND Queen



0
1
0
0
0
1
0

# Inverted Index

## Index

---

Page numbers in **bold face** refer to key term definitions

Page numbers in *italics* refer to images or diagrams

Page numbers followed by a "t" indicate a table

## A

absolute temperature scale, **350–351**

absolute zero, **351**

acceleration of gravity, A.23t

accuracy, A.5

acetic acid ( $\text{CH}_3\text{COOH}$ )

buffers, 575–576, 581–582

conjugate acid-base pairs, 540

ionization constant, 553, 554t

manufacture of, 451

titrations, 590–592

as weak acid, 144t, 145, 551–552

acid-base pairs, conjugate, **540–544**

acid-base reactions, **538**

autoionization of water, 545–547

gas-forming exchange, 150–151

net ionic equations for, 148–150

neutralization, 146–150, 561–566t

of salts, 146–151, 561–566

air, 342–343, 366–370, 380–381, 706

alkyl groups, **70–71**

alcohols, 64, 505–507

aldehydes, 278–279

alkali metals, **55**, 106

alkaline batteries, 670

alkaline earth metals, **55**

alkaline fuel cells, 674

alkalosis, 576

alkanes, **68–71**, 277–278, A.25–A.26

alkenes, **280–283**, A.26–A.27

alkyl group, **70–71**, A.25–A.26

alkynes, **281**, A.27

allotropes, **23–24**, 208, 403–405

alpha particles, 38–39, **693–696**, 697, 699–700

alpha radiation, **693**

alpha rays, 36–37

aluminum (Al), 7, 8t, 103, 634–635, 682

amines, 544–**545**

ammonia ( $\text{NH}_3$ )

amines, 545

Brønsted-Lowry base, 538–539

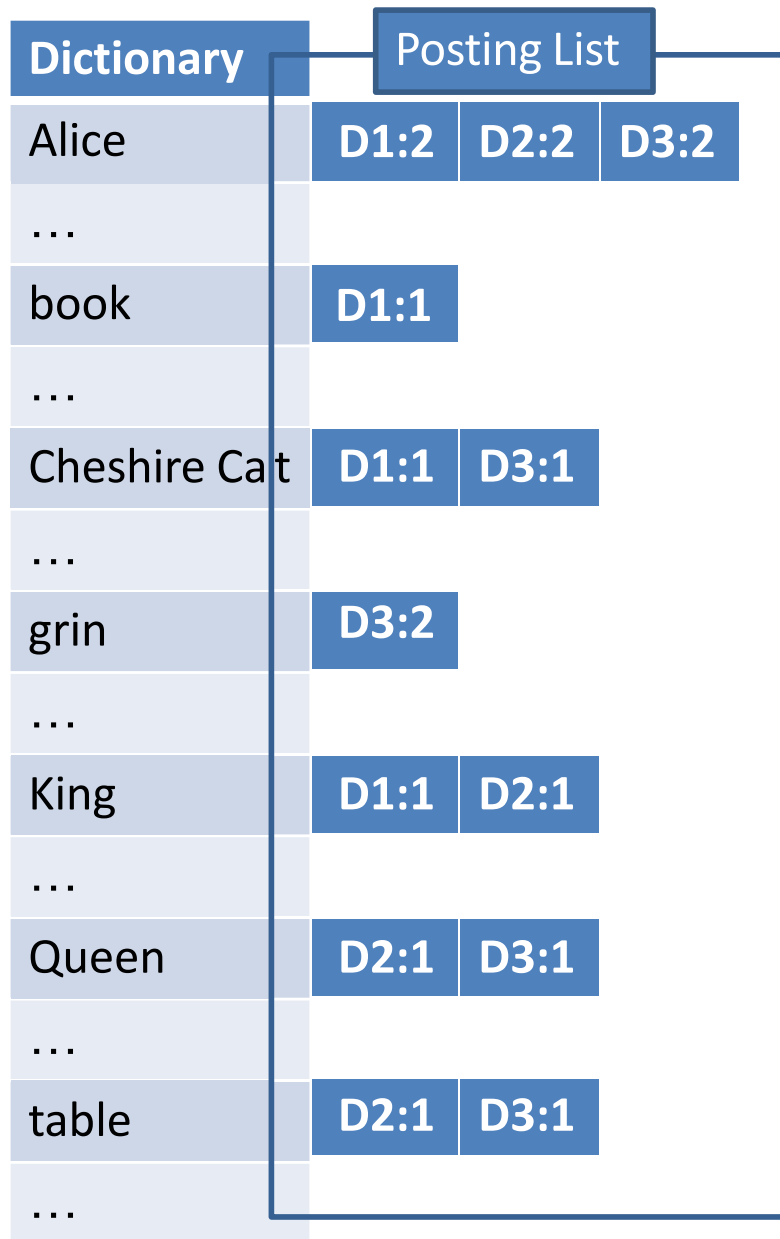
complex ions, 567

ionization constant, 554, 561

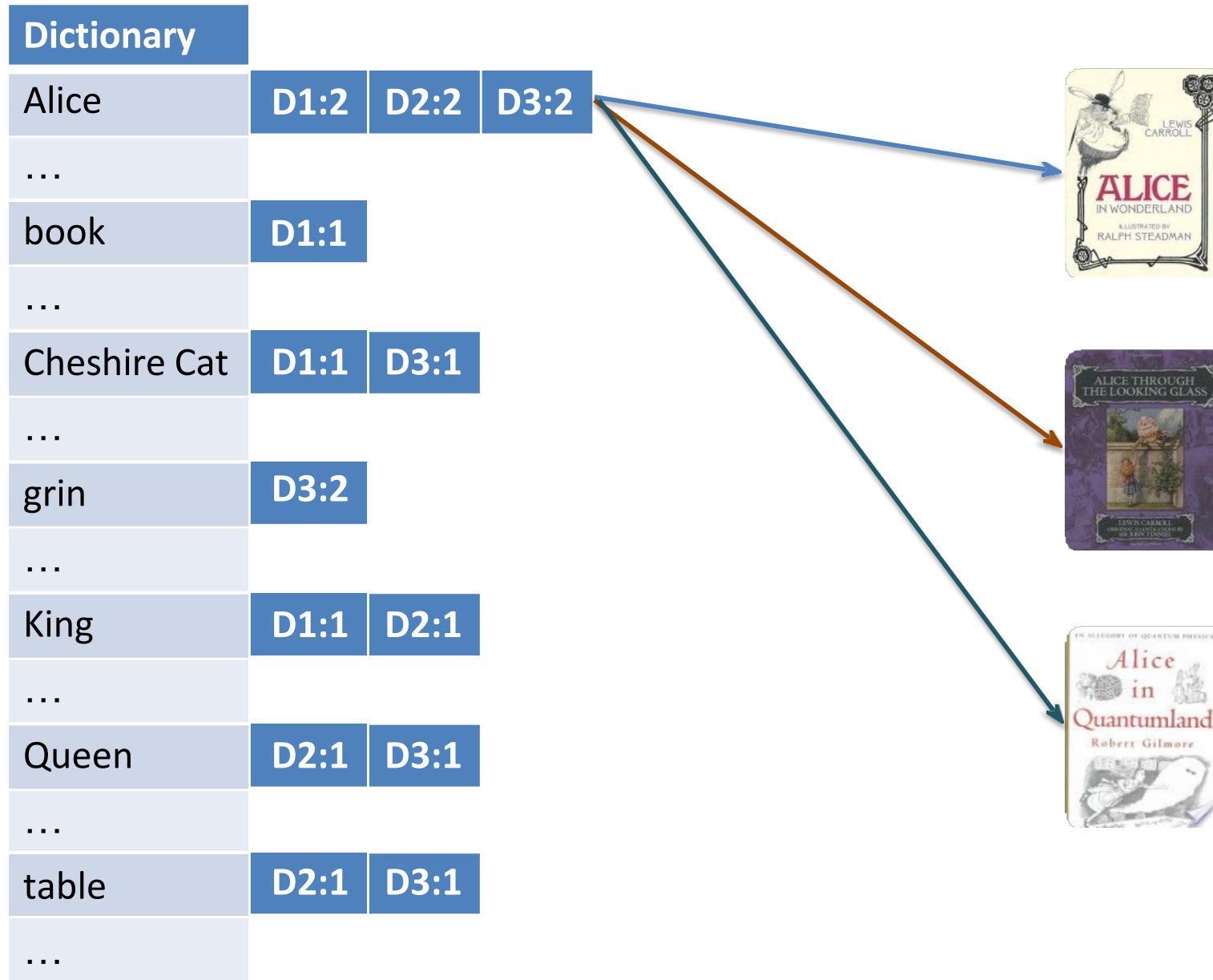
# Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			

# Inverted Index

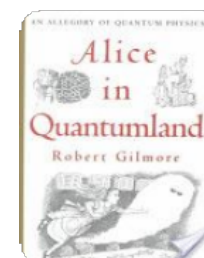
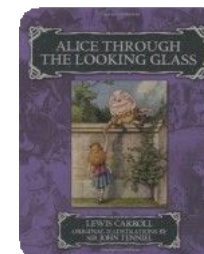
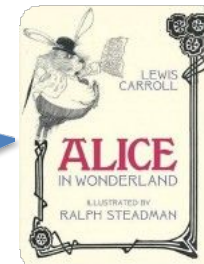


# Inverted Index

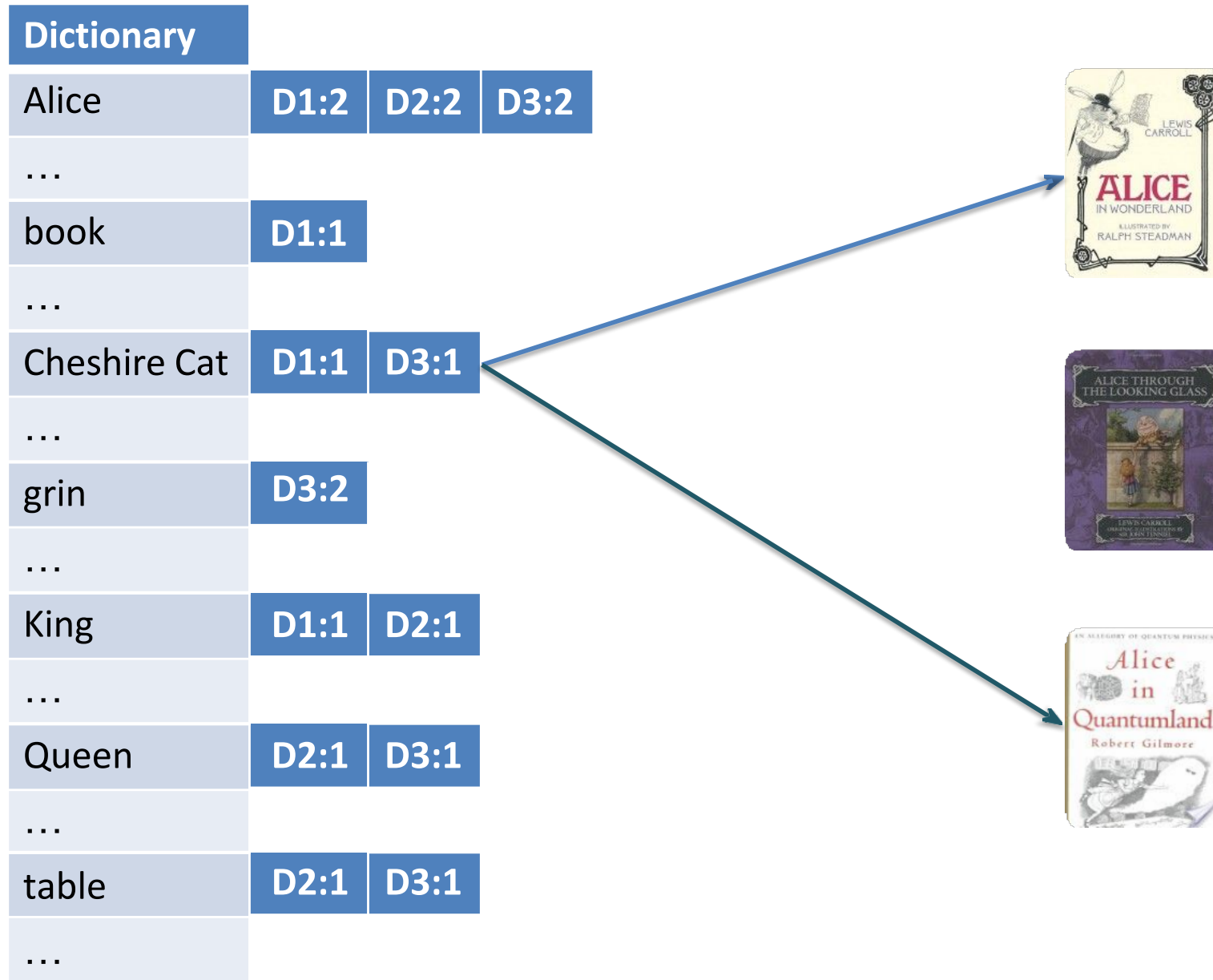


# Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			

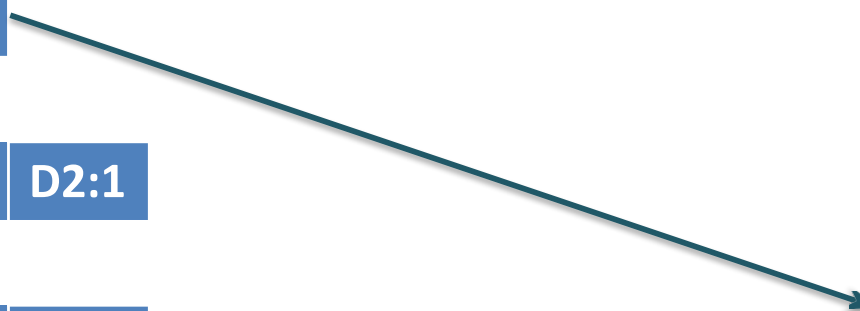
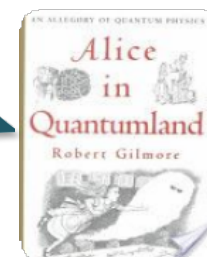
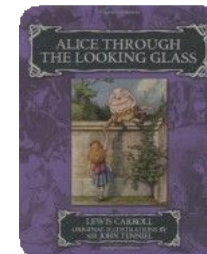
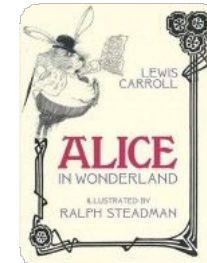


# Inverted Index



# Inverted Index

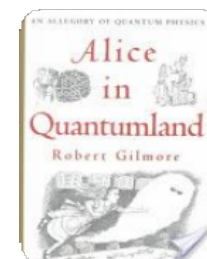
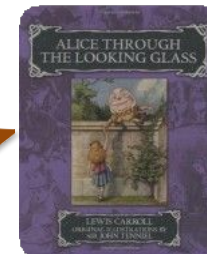
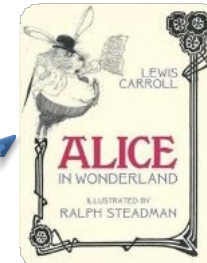
Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			





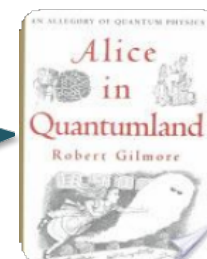
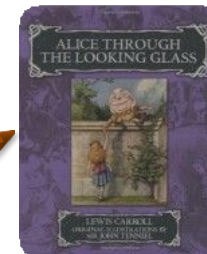
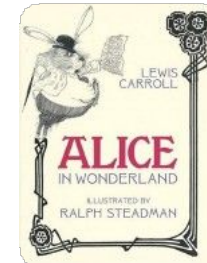
# Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			



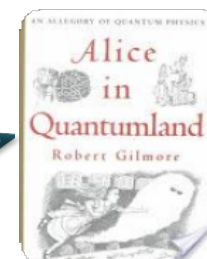
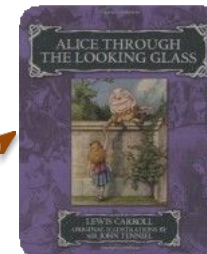
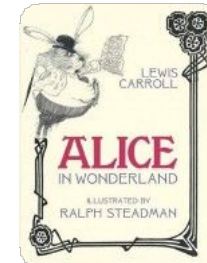
# Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			

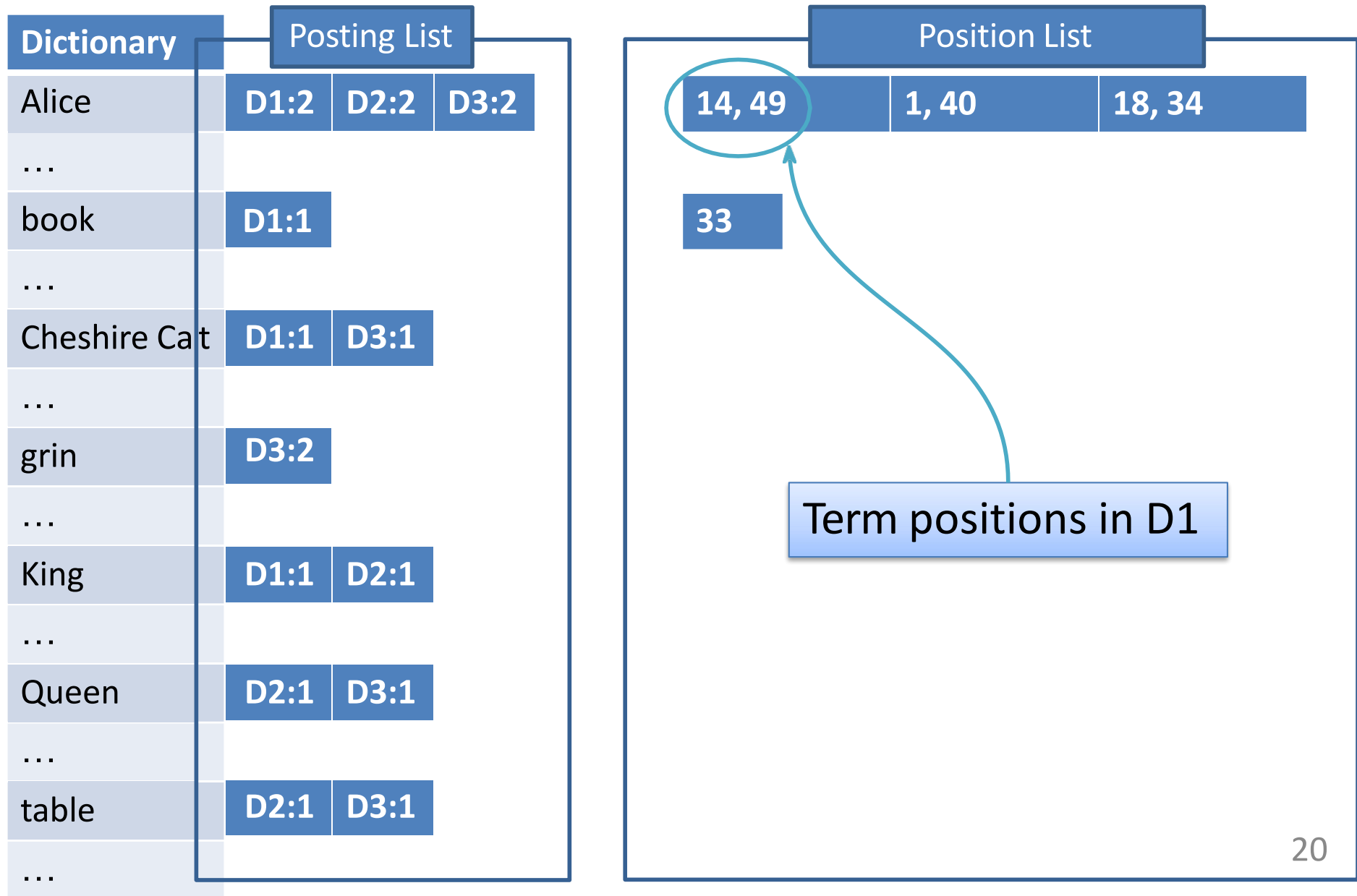


# Inverted Index

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			



# Inverted Index



# Inverted Index: query processing

Dictionary	
Alice	D1:2 D2:2 D3:2
...	
book	D1:1
...	
Cheshire Cat	D1:1 D3:1
...	
grin	D3:2
...	
King	D1:1 D2:1
...	
Queen	D2:1 D3:1
...	
table	D2:1 D3:1
...	

AND (intersection  $\cap$ )

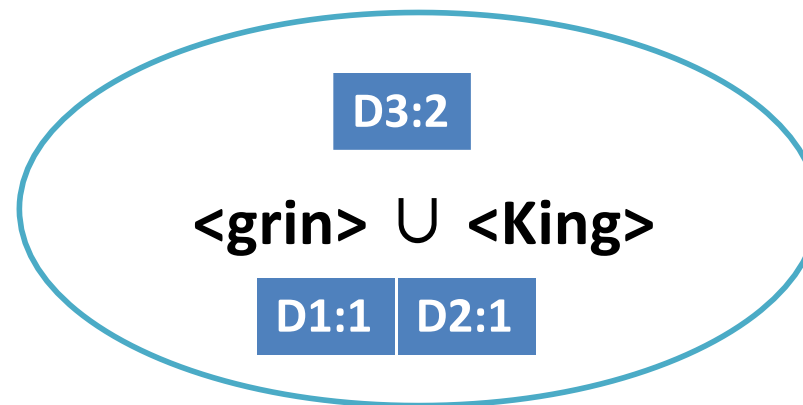


Alice AND King  $\rightarrow$  (D1, D2)

# Inverted Index: query processing

Dictionary			
Alice	D1:2	D2:2	D3:2
...			
book	D1:1		
...			
Cheshire Cat	D1:1	D3:1	
...			
grin	D3:2		
...			
King	D1:1	D2:1	
...			
Queen	D2:1	D3:1	
...			
table	D2:1	D3:1	
...			

OR (union  $\cup$ )

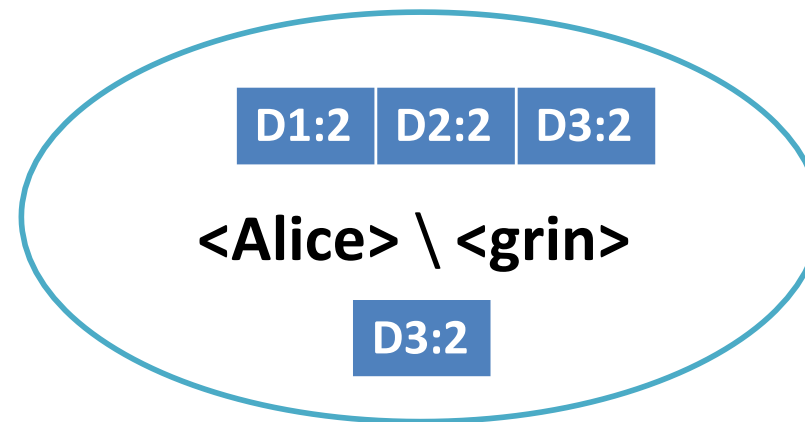


Alice OR King  $\rightarrow$  (D1, D2, D3)

# Inverted Index: query processing

Dictionary	
Alice	D1:2 D2:2 D3:2
...	
book	D1:1
...	
Cheshire Cat	D1:1 D3:1
...	
grin	D3:2
...	
King	D1:1 D2:1
...	
Queen	D2:1 D3:1
...	
table	D2:1 D3:1
...	

NOT (complement \)



Alice NOT grin -> (D1, D2)

# Term-weight

- Measures the term relevance in a document
  - component value in the document representation
  - TF\*IDF
    - TF (term frequency): term occurrences in the document
    - IDF (inverse document frequency): inverse to the number of documents in which the term occurs

$$tf * idf(t, d) = tf(t, d) * \log \frac{|D|}{|\{d \in D : t \in d\}|}$$



# Term-weight

- Measures the term relevance in a document
  - component value in the document representation
  - TF\*IDF
    - TF (term frequency): term occurrences in the document
    - IDF (inverse document frequency): inverse to the number of documents in which the term occurs

$$tf * idf(t, d) = tf(t, d) * \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Number of documents in the collection

idf

# TF\*IDF insights

- **increases proportionally** to the term frequency in the document
- **decreases** to the number of documents in which the term belongs
  - common words are generally more frequent in the collection
- **IDF depends on the collection, TF on the document**

# Inverted index/TF\*IDF

- TF: computed by term occurrences in the posting list
- IDF: computed by the posting list cardinality

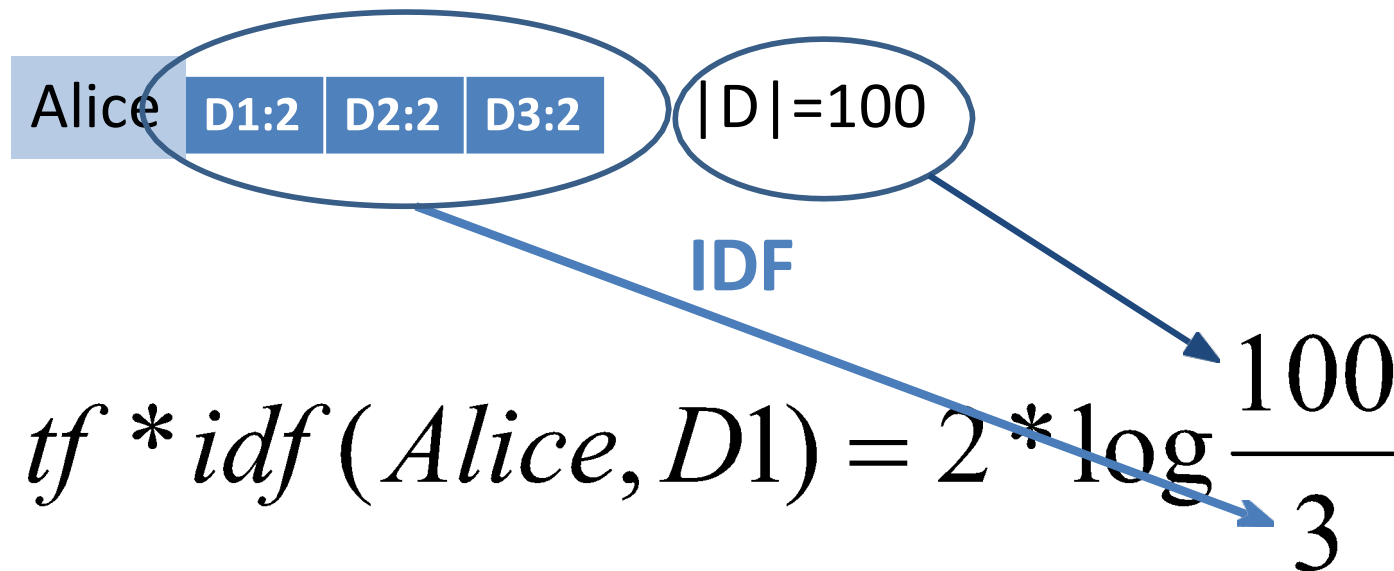


TF

$$tf * idf(Alice, D1) = 2 * \log \frac{100}{3}$$

# Inverted index/TF\*IDF

- TF: computed by term occurrences in the posting list
- IDF: computed by the posting list cardinality





<http://lucene.apache.org>

**APACHE LUCENE**

**Documentation**

[https://lucene.apache.org/core/8\\_11\\_2/index.html](https://lucene.apache.org/core/8_11_2/index.html)

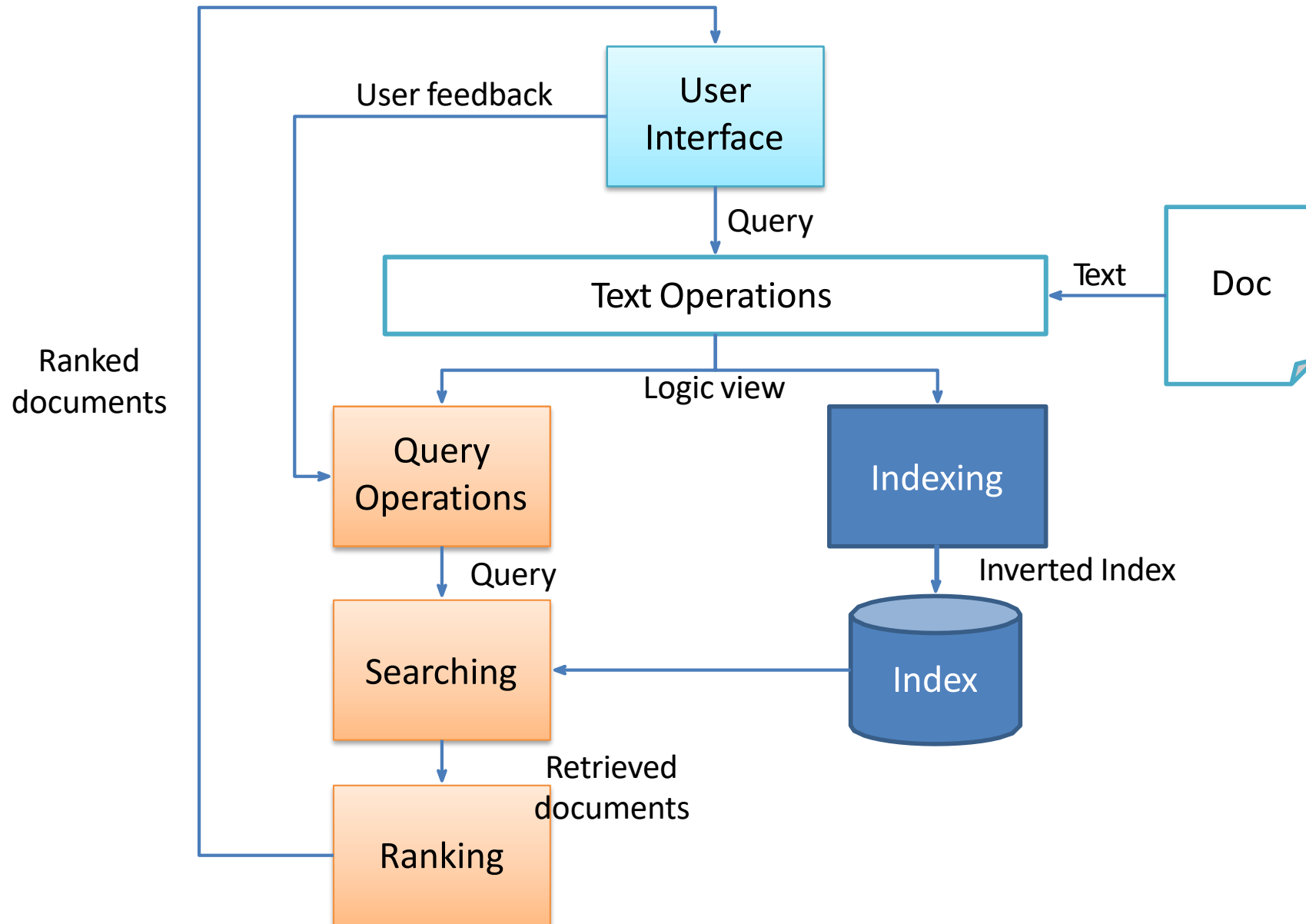
# Apache Lucene

- Apache Software Foundation project
  - <http://lucene.apache.org>
- What Lucene is
  - Search library: indexing and searching Application Programming Interface (API)
- What Lucene is not
  - Search engine (no crawling, server, user interface, etc.)

# Lucene

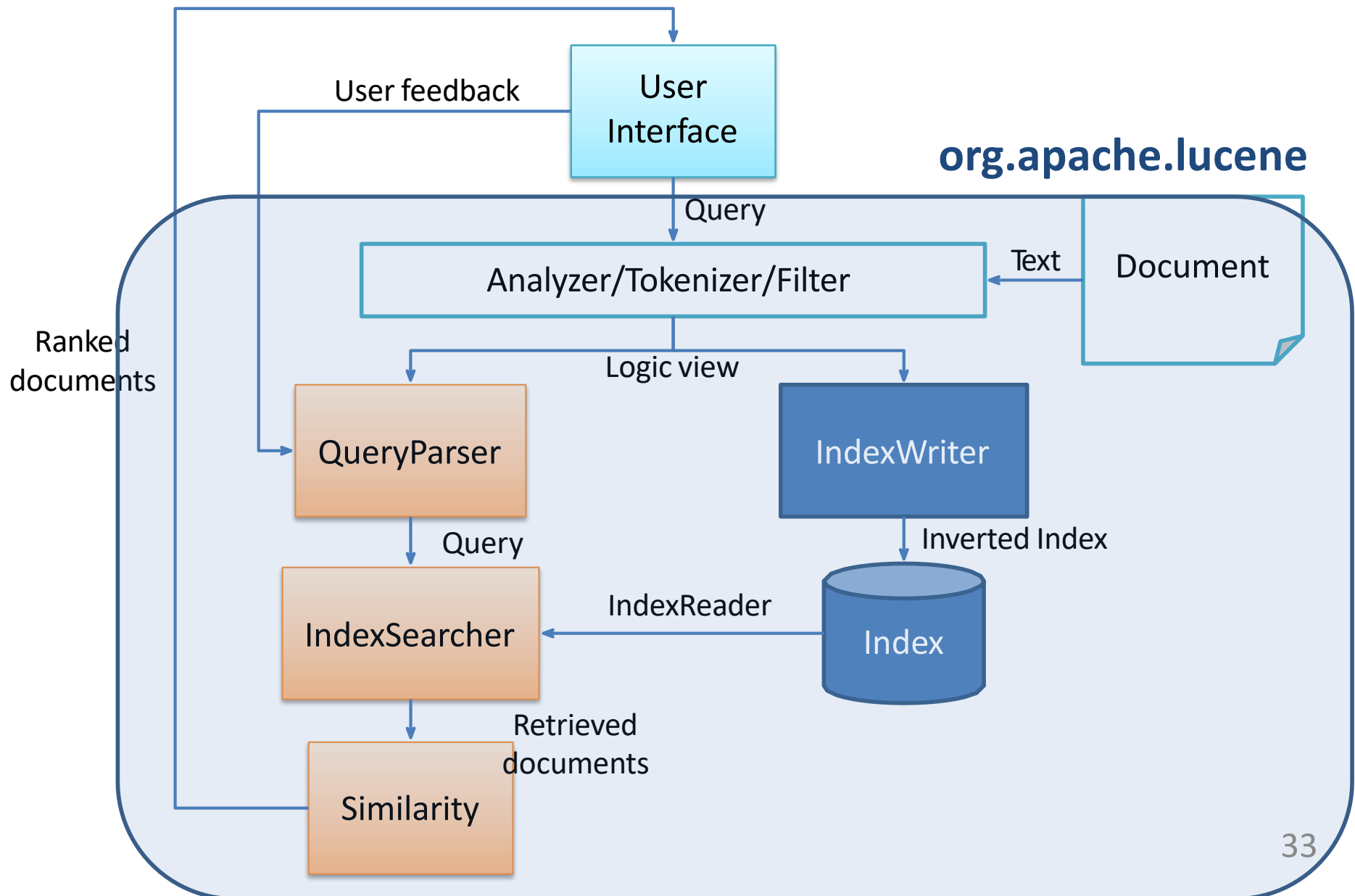
- Full-text search
- High performance
- Scalable
- Cross-platform
- 100%-pure Java
- Porting in other languages:
  - Perl, C#, C++, Python, Ruby e PHP

# Information Retrieval Process





# Information Retrieval Process



# Lucene Model 1/2

- Based on **Vector Space Model**
- Features:
  - **multi-field** document
  - **tf-term** frequency: number of matching terms in field
  - **lengthNorm**: number of tokens in field
  - **idf**: inverse document frequency
  - **coord**: coordination factor, number of matching
- Terms
  - field boost
  - query clause boost

# Lucene Model 1/2

$$score(q,d) = coord(q,d) \cdot queryNorm(q,d) \cdot \sum_{t \in q} (tf \cdot idf^2 \cdot boost(t) \cdot norm(t,d))$$

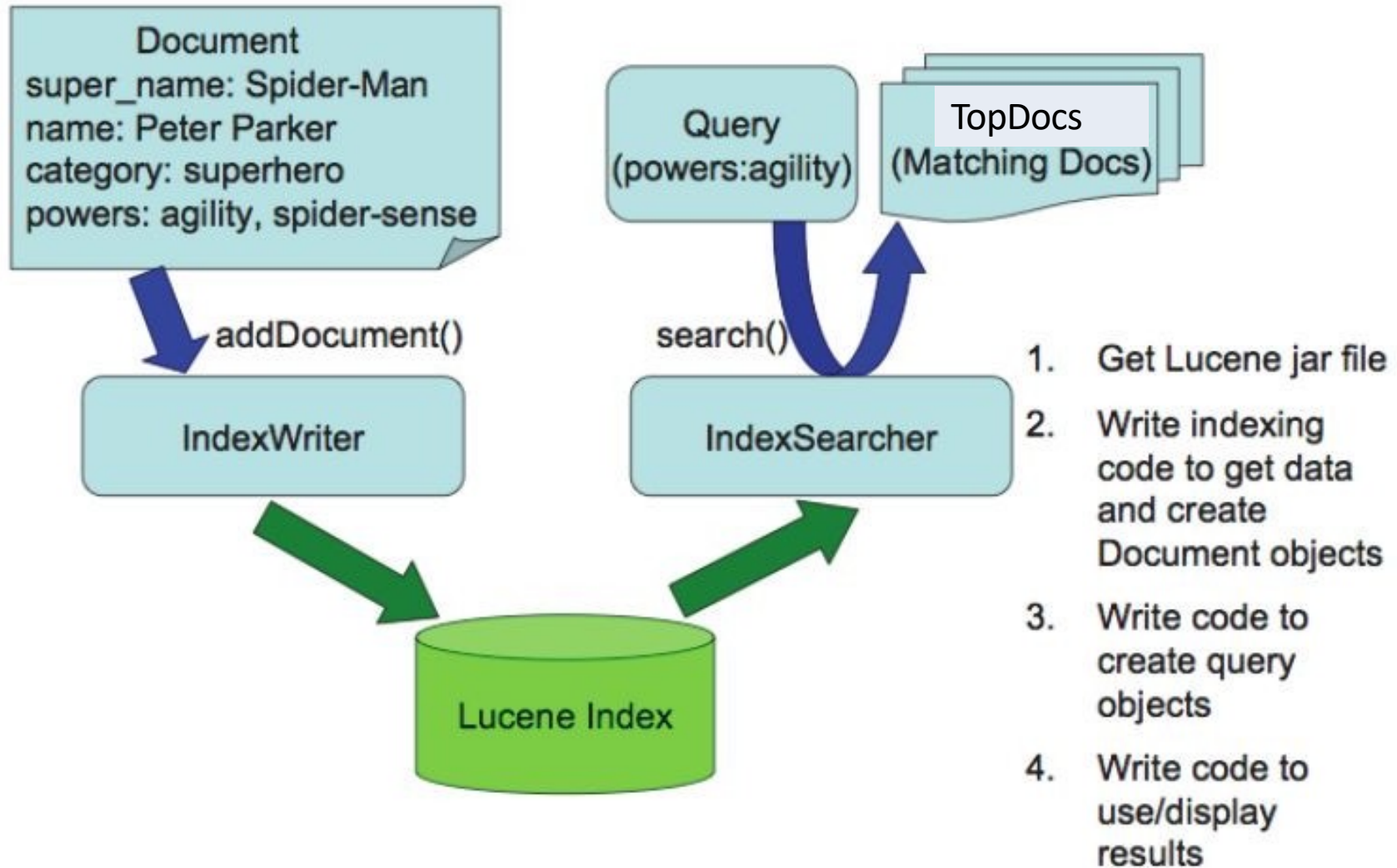
$$tf(t) = (\# occur(t))^{1/2}$$

$$idf(t) = 1 + \log\left(\frac{numDocs}{docFreq + 1}\right)$$

- $coord(q,d)$  score factor based on how many query terms are found in the specified document
- $queryNorm(q,d)$  is a normalizing factor used to make scores between queries comparable
- $boost(t)$  term boost
- $norm(t,d)$  encapsulates a few (indexing time) boost and length factors

More info [https://lucene.apache.org/core/8\\_6\\_2/core/index.html](https://lucene.apache.org/core/8_6_2/core/index.html)

# Lucene



# Document: Fields Representation

<http://www.bbc.co.uk/news/technology-15365207>

**NEWS TECHNOLOGY**

Home | UK | Africa | Asia-Pac | Europe | Latin America | Mid-East | South Asia | US & Canada | Business |

19 October 2011 Last updated at 11:55 GMT

41 [Share](#) [f](#) [t](#) [e](#)

**Rory Cellan-Jones**  
Technology correspondent  
[More from Rory](#) | [Follow Rory on Twitter](#)



## Android serves up its Ice Cream Sandwich

[COMMENTS](#) (62)

October could prove a key month in the smartphone wars. Last week saw the launch of the latest iPhone, next week we expect to see the first Nokia Windows Phone 7 handsets - and today Google has unveiled the latest version of its Android operating system.

At first glance, Ice Cream Sandwich looks as though it will set a new benchmark for what a clever phone should be able to do.

"Ice Cream Sandwich?" I hear non-Android users ask.

Each version of the operating system is named after a popular dessert, so you have to get used to bizarre statements like "People are at the heart of Ice Cream Sandwich", as [Google's blog said this morning](#).

The delights of ice cream, and the Samsung Galaxy Nexus phone designed to showcase the system, were unveiled at an event in Hong Kong, and there's [a YouTube video showing what it's about](#).



The new system uses facial recognition to unlock the phone instead of the traditional passcode to unlock the phone.

### Comments

Sign in or register to comment and rate comments.

All posts are **reactively-moderated** and must obey the **house rules**.

#### All Comments (62)

Order by: [Oldest First](#) [Highest Rated](#) [Lowest Rated](#)

1. **GrahamZ**

19TH OCTOBER 2011 - 13:14

At Last !!

Can't wait for it to be rolled out across the majority of handsets (ultimately unifying them). Hopefully it won't be too long before I can update my Galaxy....

# Document: Fields Representation

<http://www.bbc.co.uk/news/technology-15365207>

URL: StringField

Date: StringField (see DateTools)

Authors: TextField

Android serves up its Ice Cream Sandwich

Title: TextField

October could prove a key month in the smartphone wars. Last week saw the launch of the latest iPhone, next week we expect to see the first Nokia Windows Phone 7 handsets - and today Google has unveiled the latest version of its Android operating system.

Abstract: TextField

At first glance, Ice Cream Sandwich looks as though it will set a new benchmark for what a clever phone should be able to do.

"Ice Cream Sandwich?" I hear non-Android users ask.

Each version of the operating system is named after a popular dessert, so you have to get used to bizarre statements like "People are at the heart of Ice Cream Sandwich", as [Google's blog](#) said this morning.

The delights of ice cream, and the Samsung Galaxy Nexus phone designed to showcase the system, were unveiled at an event in Hong Kong, and there's a [YouTube video](#) showing what it's about.

Content: TextField

## Comments

Sign in or register to comment and rate comments.

All posts are **reactively-moderated** and must obey the **house rules**.

All Comments (62)

Order by: ■ Oldest First ▼ Highest Rated ▲ Lowest Rated

1. GrahamZ

19TH OCTOBER 2011 - 13:14

At Last !!

Can't wait for it to be rolled out across the majority of handsets (ultimately unifying them). Hopefully it won't be too long before I can update my Galaxy....

Comments: TextField

# Document

- A collection of *Fields* which give structure to the document
- Each Field is added to the document
- A Field has three parts
  - Name
  - Type
  - Value
    - text (String, Reader or pre-analyzed TokenStream)
    - binary (byte[])
    - numeric (a Number)
- A Field may be stored in the index (TYPE\_STORED), so it may be returned with hits on the document.
- Use FieldType for personalized types

# Hello World 1/3

- Setup a Java Maven project in your IDE

```
<dependencies>
  <dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-core</artifactId>
    <version>8.11.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-analyzers-common</artifactId>
    <version>8.11.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-queryparser</artifactId>
    <version>8.11.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.lucene</groupId>
    <artifactId>lucene-queries</artifactId>
    <version>8.11.2</version>
  </dependency>
</dependencies>
```



# Hello World 2/3

See the class `di.uniba.it.mri2223.lucene.HelloWorld`

```
//Open a directory from the file system (index directory)
FSDirectory fsdir = FSDirectory.open(new File("./resources/helloworld").toPath());
//IndexWriter configuration
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
//Index directory is created if not exists or
//overwritten
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
//Create IndexWriter
IndexWriter writer = new IndexWriter(fsdir, iwc);
//Create document and add
//fields
Document doc = new Document();
doc.add(new TextField("super_name", "Spider-Man", Field.Store.NO));
doc.add(new TextField("name", "Peter Parker", Field.Store.NO));
doc.add(new TextField("category", "superhero", Field.Store.NO));
doc.add(new TextField("powers", "agility, spider-sense",
Field.Store.NO));
//add document to index
writer.addDocument(doc);
//close IndexWriter
writer.close();
```

# Hello World 3/3

```
//Create the IndexSearcher
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
//Create the query parser with the default field and analyzer
QueryParser qp = new QueryParser("name", new StandardAnalyzer());
//Parse the query
Query q = qp.parse("name:parker powers:agility");
//Search
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

# Field Constructors

- `Field(String name, byte[] value, IndexableFieldType type)`  
Create field with binary value.
- `Field(String name, byte[] value, int offset, int length, IndexableFieldType type)`  
Create field with binary value.
- `Field(String name, BytesRef bytes, IndexableFieldType type)`  
Create field with binary value.
- `Field(String name, Reader reader, IndexableFieldType type)`  
Create field with Reader value.
- `Field(String name, CharSequence value, IndexableFieldType type)`  
Create field with CharSequence value.
- `Field(String name, TokenStream tokenStream, IndexableFieldType type)`  
Create field with TokenStream value.

# Adding documents to an Index

- `addDocument(Iterable<? extends IndexableField> doc)`

Adds a document to this index.

- `addDocuments(Iterable<? extends Iterable<? extends IndexableField>> docs)`

Atomically adds a block of documents with sequentially assigned document IDs, such that an external reader will see all or none of the documents.

# Exercise 1

- Index the famous novel **Alice In Wonderland** in a Lucene searchable index.
- You should index each **Chapter** as a separate Lucene Document.
- Each Document should contain:
  1. The Title of the Book
  2. The Author of the Book
  3. The title of the Chapter
  4. The text of the Chapter
- The text of the Chapter should store the Term Vector and tokens offsets.

# Exercise 1

- Start from the project Stored on GitHub!

**[https://github.com/marcopoli/MRI\\_2022\\_23/exercises/](https://github.com/marcopoli/MRI_2022_23/exercises/)**

# Analizers

- KeywordAnalyzer
  - Returns a single token
- WhitespaceAnalyzer
  - Splits tokens at whitespace
- Simple Analyzer
  - Divides text at non letter characters and lowercases
- StopAnalyzer
  - Divides text at non letter characters, lowercases, and removes stop words
- StandardAnalyzer
  - Tokenizes based on sophisticated grammar that recognizes e-mail addresses, acronyms, etc.; lowercases and removes stop words (optional)

# Text Operations

- Tokenization
  - split text in token
- Stop word elimination
  - remove common words and closed word class (e.g. function words)
- Stemming
  - reducing inflected (or sometimes derived) words to their stem

More info

[https://lucene.apache.org/core/8\\_6\\_2/analyzers-common/index.html](https://lucene.apache.org/core/8_6_2/analyzers-common/index.html)



# Analizers

The quick brown fox jumped over the lazy dogs

- WhitespaceAnalyzer:

[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]

- SimpleAnalyzer:

[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]

- StopAnalyzer:

[quick] [brown] [fox] [jumped] [lazy] [dogs]

- StandardAnalyzer:

[quick] [brown] [fox] [jumped] [over] [lazy] [dogs]

# Analizers

"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]

# Tokenizers

- Tokenization is the process of breaking input text into small indexing elements, e.g. tokens
- A **Tokenizer** is a **TokenStream** and is responsible for breaking up incoming text into tokens. Usually Analyzer will use a Tokenizer as the first step in the analysis process

source string: “full-text lucene.apache.org”

- StandardTokenizer
  - “full” “text” “lucene.apache.org”
- WhitespaceTokenizer
  - “full-text” “lucene.apache.org”
- LetterTokenizer
  - “full” “text” “lucene” “apache” “org”

# Text Analysis

- **Analyzer:** encapsulates the analysis process
  - Tokenize a text by performing any number of operations on it
  - Responsible for building the `TokenStream` consumed by the indexing and searching processes
- **Tokenizer:** is a `TokenStream`
  - Responsible for breaking up incoming text into tokens
  - Is a `TokenStream` whose input is a `Reader`
- **TokenFilter:** is also a `TokenStream` and is
  - responsible for modifying tokens that have been created by the `Tokenizer`
  - whose input is another `TokenStream`.

# TokenFilters

- A TokenFilter is also a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer.
- LowerCaseFilter
- StopFilter
- LengthFilter
- PorterStemFilter
  - stemming: reducing words to root form
  - rides, ride, riding => ride
  - country, countries => countri

# Test Analyzer

See the class `di.uniba.it.mri2223.lucene.TestAnalyzer`

```
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
    IOException {
    List<String> tokens = new ArrayList<>();
    TokenStream tokenStream = analyzer.tokenStream("text", reader);
    tokenStream.reset();
    CharTermAttribute cattr = tokenStream.addAttribute(CharTermAttribute.class);
    while (tokenStream.incrementToken()) {
        String token = cattr.toString();
        tokens.add(token);
    }
    tokenStream.end();
    return tokens;
}
```

# Custom Analyzer

See the class `di.uniba.it.mri2223.lucene.MyAnalyzer`

```
public static final CharArraySet STOP_WORDS;

static {
    final List<String> stopWords = Arrays.asList("a", "an", "and", "are",
    "the", "is", "but", "by");
    final CharArraySet stopSet = new CharArraySet(stopWords, false);
    STOP_WORDS = CharArraySet.unmodifiableSet(stopSet);
}
```

**@Override**

```
protected TokenStreamComponents createComponents(String fieldName) {
    Tokenizer source = new LetterTokenizer();
    TokenStream filter = new LowerCaseFilter(source);
    filter = new StopFilter(filter, STOP_WORDS);
    return new TokenStreamComponents(source, filter);
}
```

# Lucene Search

- **IndexReader**: interface for accessing an index
- **QueryParser**: parses the user query
- **IndexSearcher**: implements search over a single IndexReader
  - search(Query query, int numDoc)
  - TopDocs -> result of search
    - TopDocs.scoreDocs returns an array of retrieved documents (ScoreDoc)



# Lucene QueryParser

- Example:
  - `queryParser.parse("name:SpiderMan");`
- good human entered queries, debugging
- does text analysis and constructs appropriate queries
- not all query types supported

# QUERY SYNTAX 1/2

- title:"The Right Way" AND text:go
  - **Phrase query + term query**
- **Wildcard Searches**
  - tes\* (test - tests - tester)
  - te?t (test – text)
  - te\*t (tempt)
  - tes?
- **Fuzzy Searches** (Levenshtein Distance) (TERM)
  - roam~ (foam – roams)
  - roam~0.8
- **Range Searches**
  - mod\_date:[20020101 TO 20030101]
  - title:{Aida TO Carmen}
- **Proximity Searches** (PHRASE)
  - "jakarta apache"~10

# QUERY SYNTAX 2/2

- **Boosting a Term**
  - jakarta^4 apache
  - "jakarta apache"^4 "Apache Lucene"
- **Boolean Operator**
  - NOT, OR, AND
  - + required operator
  - - prohibit operator
- Grouping by ( )
- Field Grouping
- title:(+return +"pink panther")
- Escaping Special Characters by \

# Lucene Search

See the class `di.uniba.it.mri2223.lucene.TestSearch1`

```
FSDirectory fsdir = FSDirectory.open(new
File("./resources/helloworld").toPath()); IndexSearcher searcher = new
IndexSearcher(DirectoryReader.open(fsdir));
//Single term query
//Query q = new TermQuery(new Term("name", "parker"));
//Boolean query
BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("name", "parker")), BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("powers", "agility")),
BooleanClause.Occur.SHOULD); Query q = qb.build();
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

# QUERY (BY CODE) 1/2


```
TermQuery query = new TermQuery(new Term("name", "Spider-Man"))
```

- explicit, no escaping necessary
- does not do text analysis for you
- Query
  - TermQuery
  - BooleanQuery
  - PhraseQuery
  - FuzzyQuery / WildcardQuery /
- See **TestSearch1** for examples of TermQuery and BooleanQuery

# QUERY (BY CODE) 2/2

```
TermQuery tq1 = new TermQuery(new Term("name","pippo"))  
TermQuery tq2 = new TermQuery(new Term("name","pluto"))
```

```
BooleanQuery.Builder qb = new BooleanQuery.Builder();  
qb.add(new TermQuery(new Term("name", "parker")),  
    BooleanClause.Occur.SHOULD);  
qb.add(new TermQuery(new Term("powers", "agility")),  
    BooleanClause.Occur.SHOULD);
```



SHOULD -> OR  
MUST -> AND  
MUST\_NOT -> NOT

# Exercise 2

- Use the Index of **Alice In Wonderland** for making queries about the following needs. Search all paragraphs that talks about the:
  1. «**The Mad Hatter**»
  2. «**The Mad Hatter**» AND the «**Cheshire Cat**»
  3. («**The Mad Hatter**» OR the «**Cheshire Cat**») AND '**ALICE**'
  4. «**King**» or «**Queen**» «**of hearts**»
  5. «**Queen**» «**of hearts**» AND '**Alice**' with a max distance of 10 terms
  6. Something like '**Alice**' with a similarity higher than 0.7
  7. "**Alice**" but not «**The Mad Hatter**»
  8. "**Alice**" Or «**The Mad Hatter**» with a boost of 2 for them containing "**Alice**"
  9. ...

# Exercise 3

Build a “little” search engine that indexes and searches text files into a folder

- main class **IndexSE** takes the folder name and the index directory name as arguments
  - the class indexes all the “.txt” into the folder
- main class **SearchSE** takes the index directory name and the query as arguments
  - the class searches the index



# Exercise 3

See classes:

```
. di.uniba.it.mri2223.lucene.se.IndexSE  
. di.uniba.it.mri2223.lucene.se.SearchSE
```

# IndexSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();
for (File file : files) {
    if (file.isFile() && file.getName().endsWith(".txt")) {
        Document doc = new Document();
        doc.add(new StringField("id", file.getAbsolutePath(),
Field.Store.YES));
        doc.add(new TextField("text", new FileReader(file)));
        writer.addDocument(doc);
    }
}
writer.close();
```

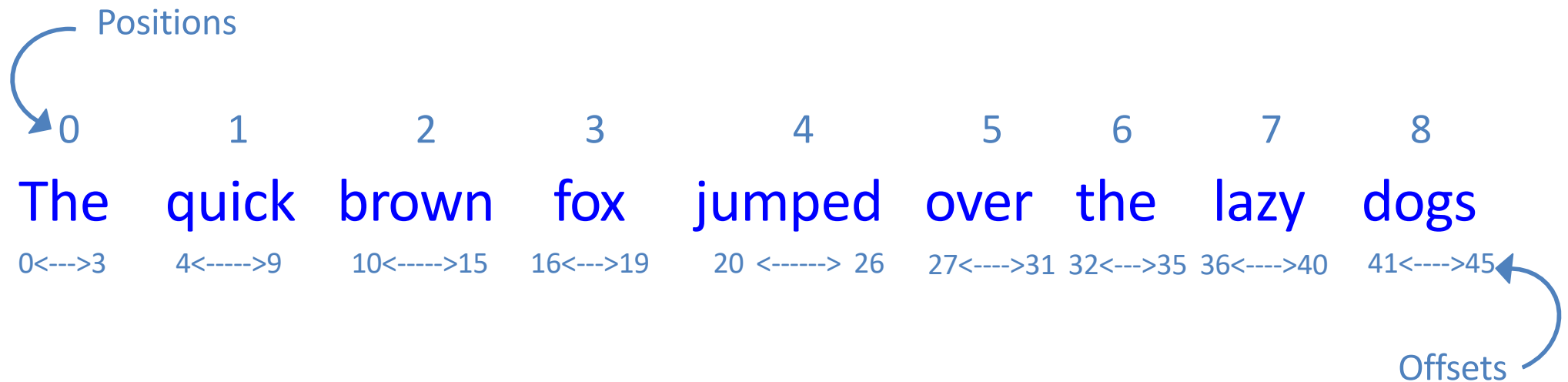
# SearchSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[0]).toPath());
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
//Create the query parser with the default field and analyzer
QueryParser qp = new QueryParser("text", new StandardAnalyzer());
//Parse the query
Query q = qp.parse(args[1]);
//Search
TopDocs topdocs = searcher.search(q, 10);
for (ScoreDoc sdoc : topdocs.scoreDocs) {
    System.out.println("Found doc, path=" +
searcher.doc(sdoc.doc).get("id") + ", score" + sdoc.score);
}
```

# Posting API

- Getting access to term **frequency, positions and offsets**
- Required for:
  - Relevance Feedback and “More Like This”
  - Clustering
  - Similarity between two documents
  - Highlighter
    - needs offsets info

# Term Vector: Positions and Offsets



# Store Posting 1/2

- During indexing, create a `Field` that stores `TermVectors` through `FieldType` (we used `TextField` as a predefined `IndexableField`)
  - `FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);`
  - `TYPE_STORED|TYPE_NOT_STORED`
- Invoke in cascade the following methods to set the field type:
  - `ft.setTokenized(true); //done as default`
  - `ft.setStoreTermVectors(true);`
  - `ft.setStoreTermVectorPositions(true);`
  - `ft.setStoreTermVectorOffsets(true);`
  - `ft.setStoreTermVectorPayloads(true);`

# Store Posting 2/2

See class: `di.uniba.it.mri2223.lucene.se.post.IndexPostExample`

```
//define a custom field type that stores post information
FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);
ft.setStoreTermVectors(true);
ft.setStoreTermVectorPositions(true);
ft.setStoreTermVectorOffsets(true);
...
Document doc1 = new Document();
doc1.add(new StringField("id", "1", Field.Store.YES));
doc1.add(new Field("text", new FileReader("./resources/text/es1.txt"), ft));
writer.addDocument(doc1);
```

# Posting API

- Getting the **BoW** of a document:

`di.uniba.it.mri2223.lucene.se.post.PostExample`

```
Fields fields = ireader.getTermVectors(docid);
for (String field : fields) {
    Terms terms = fields.terms(field);
    TermsEnum termsEnum = terms.iterator();
    BytesRef term = null;
    while ((term = termsEnum.next()) != null) {
        System.out.print(term.utf8ToString());
        PostingsEnum postings = termsEnum.postings(null, PostingsEnum.FREQS);
        while (postings.nextDoc() != DocIdSetIterator.NO_MORE_DOCS) {
            System.out.println(": " + postings.freq());
        }
    }
}
```

More info

[https://lucene.apache.org/core/8\\_6\\_2/core/org/apache/lucene/index/package-summary.html#postings](https://lucene.apache.org/core/8_6_2/core/org/apache/lucene/index/package-summary.html#postings)



# Posting API

- Getting positions and offsets:

`di.uniba.it.mri2223.lucene.se.post.PostExample`

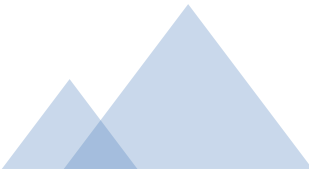
```
Fields fields = ireader.getTermVectors(docid);
for (String field : fields) {
    Terms terms = fields.terms(field);
    TermsEnum termsEnum = terms.iterator();
    BytesRef term = null;
    while ((term = termsEnum.next()) != null)
    {
        System.out.print(term.utf8ToString());
        PostingsEnum postings = termsEnum.postings(null,
            PostingsEnum.ALL); while (postings.nextDoc() !=
            DocIdSetIterator.NO_MORE_DOCS) {
            System.out.print(": " + postings.freq());
            for (int i = 0; i < postings.freq(); i++) {
                int position = postings.nextPosition();
                System.out.println("\t" + position + "\t{" +
                postings.startOffset( + ", " + postings.endOffset( + "}");
            }
        }
    }
}
```

# Exercise 4

- Take the IndexSE and add a third argument
  - the argument corresponds to different Field.TermVector... options
    - tv: term vectors
    - tvp: term vectors with positions
    - tvo: term vectors with positions and offsets
- Try Queries in SearchSE and highlight the found terms in the matching document **text** using positions and offsets.



# Interfaccia grafica: Luke

- **Luke** è un interfaccia grafica che permette di accedere agli indici creati da lucene ed eseguire una serie di operazioni:
    - Vedere i documenti indicizzati con i rispettivi campi
    - Eseguire e testare delle query complesse
    - Modificare l' indice cancellando documenti o modificando i dati già indicizzati.
    - E molto altro ...
- 

# LUKE!

Luke: Lucene Toolbox Project - v8.1.1

File Tools Help

Overview Documents Search Analysis Commits Logs

Index Path: /Users/kram/Desktop/lucene-8.1.1/primo\_indice2.index

Number of Fields: 5

Number of Documents: 3

Number of Terms: 21

Has deletions? / Optimized?: No / No

Index Version: 10

Index Format: Lucene 7.4 or later

Directory implementation: org.apache.lucene.store.MMapDirectory

Currently opened commit point: segments\_3 (generation=3, segs=3)

Current commit user data: {}

Select a field from the list below, and press button to view top terms in the field.

Available fields and term counts per field:

Name	Term count	%
testo_libero	8	38.10 %
name	6	28.57 %
id	3	14.29 %
super_name	3	14.29 %
category	1	4.76 %

Selected field: testo\_libero

Show top terms >

Num of terms: 50

Top ranking terms: (Double-click for more options.)

Rank	Freq	Text
1	2	superero
2	2	bravo
3	1	quasi
4	1	qesto
5	1	propr
6	1	massim
7	1	jhon
8	1	bomba

# LUKE!

Luke: Lucene Toolbox Project - v8.1.1

File Tools Help

Overview Documents Search Analysis Commits Logs

Browse terms in field:

testo\_libero

« First Term bravo Next

Hint:  
Edit the text field above and press Enter to seek to arbitrary terms.

Browse documents by term:

bravo

« First Doc 1 Next in 2 docs

Position	Offsets	Payload
4	26-31	

Browse documents by Doc # 0 in 3 docs

Copy values More like this Add document

(Select a row and double-click for more options.)  
(To copy all or arbitrary field value(s), unselect all rows or select row(s), and click 'Copy values' button.)

Field	Flags ? Help	Norm	Value
id	Idfp-N-S-----	1	Doc113
super_name	Idfp-N-S-----	1	Lop
name	Idfp-N-S-----	2	Wer Forest
category	Idfp-N-S-----	1	superhero
testo_libero	IdfpN-S----V-----	5	Qesto Supereroe è proprio bravo! Una Bomba...

# Query Results

Luke: Lucene Toolbox Project - v8.1.1

File Tools Help

Overview Documents **Search** Analysis Commits Logs

Query settings

Query Parser Analyzer Similarity Sort Field Values More Like This

☒ StandardQueryParser ☐ Classic QueryParser

Default field **testo\_libero** Default operator **OR**

☐ Enable position increments ☒ Allow leading wildcard (\*)

☒ Split on whitespace

Phrase query:

☒ Generate phrase query

☐ Generate multi term synonyms phrase query

Query expression ☐ Term Query

testo\_libero:" bomba bravo"~4

Parsed query

testo\_libero:"bomba bravo"~4

Parse ☐ rewrite

Search ☐ exact hits count

More Like This with doc # **0**

Search Results: Total docs: 1 hits **1 ~ 1** Delete Docs

(Select a row and double-click for more options.)

Doc ID	Score	Field Values
0	0,191	testo_libero=Questo Supereroe è proprio bravo! Una Bomba...; name=Wer Forest; id=Doc113; super_name=Lop; categor...

Implement an evaluation pipeline using the cranfield paradigm

# Evaluation

# Cranfield Experiment

[http://ir.dcs.gla.ac.uk/resources/test\\_collections/cran/](http://ir.dcs.gla.ac.uk/resources/test_collections/cran/)

- Implement an evaluation pipeline
- Given
  - A test collection
  - Relevance assessments
  - Evaluation MetricsFind out the best indexing/searching configurations
- Experiment with
  - Analyzers
  - Query construction
  - Field structuring
  - Term/field boosting



# Test collection

- All files are in `./resources/cran`
- Cranfield Collection: collection of abstracts
- **cran.all.1400.json**: documents collection in JSON
  - fields: *id, text, authors, title, biblio*
- **cran.qry.json**: topics (queries) in JSON
  - fields: *id, query*
- **cranqrel**: relevance judgments
  - a map between a query and the set of relevant documents

`<qid>` `<run_num>` `<docid>` `<relevance>`

# Pipeline

- **Index** the documents collection
- **Implement an evaluation class** that for each topic (query) stores the top 100 retrieved documents
  - see the output format in the next slide
- **Try** different analyzers, queries formulation, term boosting ... and **measure** the performance of your pipeline

# The output format

- Output format, six fields separated by a space char:
  - query\_id
  - run\_identifier: may be every number (it is used to keep reference to the experiment)
  - doc\_id
  - doc\_rank
  - doc\_score
  - exp\_name: exp\_name is a short reference string to the experiment
- `cran.out` contains an output example

# Trec\_Eval

- Program for running the evaluation
- Outputs many common evaluation measures:
  - Precision/recall
  - Number of [relevant] document retrieved
  - MAP, GMAP, P@K
- How to execute
  - `trec_eval relevance_judgment_file output_file`
- See: [http://trec.nist.gov/trec\\_eval/index.html](http://trec.nist.gov/trec_eval/index.html)

It must be compiled before using!

You should run the MAKE command into the shell.

# Trec\_Eval

[https://github.com/usnistgov/trec\\_eval](https://github.com/usnistgov/trec_eval)

```
((base) uniba@MacBook-Pro-4 cran % ./trec_eval cranqrel cran.out
runid                all      exp0
num_q                 all      225
num_ret               all      22500
num_rel               all      1837
num_rel_ret           all      1232
map                   all      0.3638
gm_map                all      0.2165
Rprec                 all      0.3579
bpref                  all      0.7100
recip_rank             all      0.7679
iprec_at_recall_0.00  all      0.7840
iprec_at_recall_0.10  all      0.7515
iprec_at_recall_0.20  all      0.6324
iprec_at_recall_0.30  all      0.5146
iprec_at_recall_0.40  all      0.4152
iprec_at_recall_0.50  all      0.3552
iprec_at_recall_0.60  all      0.2636
iprec_at_recall_0.70  all      0.2130
iprec_at_recall_0.80  all      0.1421
iprec_at_recall_0.90  all      0.0995
iprec_at_recall_1.00  all      0.0892
P_5                    all      0.4124
P_10                   all      0.2760
P_15                   all      0.2130
P_20                   all      0.1742
P_30                   all      0.1335
P_100                  all      0.0548
P_200                  all      0.0274
P_500                  all      0.0110
P_1000                 all      0.0055
(base) uniba@MacBook-Pro-4 cran % █
```