**Full Text Search (FTS)**

- More formally known as Information Retrieval (IR)
- Search LARGE amount of textual data

**Database Query**

- Find the posts regarding "SSHD login errors".
  
  ```sql
  select * from posts
  where content like '%SSHD login errors%';
  ```

  Here are the steps to take to fix the SSHD login errors:
  ...

  Please help! I got SSHD login errors!

**Search Text**

- Web search
- Desktop search
- Applications
  - Search posts in a bulletin board
  - Search product descriptions at an online retailer
  - ...

**Problems with Database Queries**

- Please help! I got an error when I tried to login through SSHD!

- There a problem recently discovered regarding SSHD and login. The error message is usually ...

- The solution for sshd/login errors: ...

- And how about performance??

**Characteristics of FTS**

- Vs. Databases
  - "Fuzzy" query processing
  - Relevancy ranking
Accuracy of FTS

\[
\text{Precision} = \frac{\text{# of relevant documents retrieved}}{\text{# of documents retrieved}}
\]

\[
\text{Recall} = \frac{\text{# of relevant documents retrieved}}{\text{# of relevant documents}}
\]

Journey of a Document

1. Stripping non-textual data
2. Tokenizing
3. Removing stop words
4. Stemming
5. Indexing

Document

- **Original**
  
  ```html
  <html>
  <body>
  <p>The solution for sshd/login errors: ...
  </p>
  </body>
  <html>
  ```

- **Text-only**
  
  The solution for sshd/login errors: ...  

Chinese Text Example

- **Text:** 今天天气不错。
- **Unigram:** 
  今天 
  天天 
  天气 
  不 站
- **Bigram:**
  今天天天
  天气不错
- **Grammar-based:**
  今天天气不错

Tokenizing

- [the] [solution] [for] [sshd] [login] [errors]

Stop Words

- **Words that do not help in search and retrieval**
  - Function words: a, an, and, the, of, for ...
- **After stop words removal:**
  
  solution sshd login errors

**Problem of stop word removal??**
Stemming
- Reduce a word to its stem or root form.
- Examples:
  - connection, connections → connect
  - connected, connecting → connect
  - connective
  - [solution] [sshd] [login] [errors] → [solve] [sshd] [login] [error]

Inverted Index
- cat
- dog
- keywords
- buckets
- documents
- 22, 137 234

Query Processing
- Query
  - tokenizing
  - Removing stop words
  - Stemming
  - Searching
  - results

Ranking
- How well the document matches the query
  - E.g. weighted vector distance
  - How “important” the document is
  - E.g. based on ratings, citations, and links

FTS Implementations
- MySQL: MyISAM tables only
- PostgreSQL (since 8.3)
- Oracle, DB2, MS SQL Server, ...
- Standard-alone IR libraries
  - Lucene, Egothor, Xapian, MG4J, ...

FTS from the Perspective of Application Developers
- Prepare data
- Create query
- Display result
- (Index)
- (Ranking)
Lucene Overview

- Originally developed by Doug Cutting
- THE full text search solution for Java applications
- Handles text only – needs external converters to convert other document types to text
- Java API - [http://lucene.apache.org/java/3_4_0/api/core/index.html](http://lucene.apache.org/java/3_4_0/api/core/index.html)

Example 1: Index Text Files

- Directory
- Document and Field
- Analyzer
- IndexWriter

Directory

- A place where the index files will be stored
- FSDirectory – file system directory
- RAMDirectory – virtual directory in memory

Analyzer

- Pre-processing the document or query text – tokenization, stop words removal, stemming ...
- Lucene built-in analyzers
  - WhitespaceAnalyzer, SimpleAnalyzer, StopAnalyzer
  - StandardAnalyzer
    - Grammar-based
    - Recognize special tokens such as email addresses
    - Handle CJK text

IndexWriter

- addDocument( Document )
- close()
- optimize()

Document

- A document consists of a number of user-defined fields
  - Title: FTS with Lucene
  - Author: Chengyu Sun
  - Content:
    - lots of words ...
    - lots of words ...
Types of Fields

- Indexed – whether the field is indexed
  - Analyzed
  - Not analyzed
- Stored – whether the original text is stored together with the index

Common Usage of Field Types

<table>
<thead>
<tr>
<th>Field</th>
<th>Indexed</th>
<th>Analyzed</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Large text file</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>ID, people’s name, date</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Non-searchable data</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>

Example 2: Search

- Query and QueryParser
- IndexSearcher
- TopDocs and ScoreDoc
- Document (again)

Queries

- full text search
- +full +text search
- +full +text -search
- +title:"text search"
- + (title:full title:text) -author:"john doe"

IndexSearcher

- search( Query, int n ) – returns the top n results for the query

TopDocs and ScoreDoc

- TopDocs contains an array of ScoreDoc, which has a document id and the relevancy score of the document
Factors in Lucene Score

- # of times a term appears in a document
- # of documents that contain the term
- # of query terms found
- length of a field
- boost factor - field and/or document
- query normalizing factor – does not affect ranking

See the API documentation for the Similarity class.

Document (again)

- Methods to retrieve data stored in the document
  - String get(String fieldName)

Handle Rich Text Documents

- HTML
  - NekoHTML
- PDF
  - PDFBox
- MS Word
- POI

Further Readings

- Lucene in Action (2ed Ed) by Michael McCandless, Erik Hatcher and Otis Gospodnetic

FTS in PostgreSQL

- Since 8.3
  - tsearch/tsearch2 module before 8.3

Text Search Configuration

- Specify the options to transform a document to a tsvector – tokenization, stop words removal, stemming etc.
- psql commands
  - \dF
  - show default_text_search_config;
  - set default_text_search_config=english;
- Change default text search configuration in $DATA/postgresql.conf
Sample Schema

```sql
create table messages {
    id      serial primary key,
    subject varchar(4092),
    content text,
    author  varchar(255)
};
```

Basic Data Types and Functions

- **Data types**
  - tsvector
  - tsquery
- **Functions**
  - to_tsvector
  - to_tsquery
  - plainto_tsquery

Query Syntax

- `plainto_tsquery` to `tsquery`
- `to_tsvector(text)` to `tsquery`
- `text @@ text`
- `to_tsvector(text) @@ plainto_tsquery(text)`

The Match Operator `@@`

- `tsvector @@ tsquery`
- `tsquery @@ tsvector`
- `text @@ tsquery`
- `to_tsvector(text) @@ tsquery`
- `text @@ text`
- `to_tsvector(text) @@ plainto_tsquery(text)`

Note that there is no `tsquery @@ text`.

Query Examples

- Find the messages that contain "computer programs" in the content
- Find the messages that contain "computer programs" in either the content or the subject

Create an Index on Text Column(s)

```sql
create index messages_content_index
on messages
using gin(to_tsvector('english',content));
```

- **Expression (function) index**
- The `language` parameter is required in both index construction and query
Use a Separate Column for Text Search
- Create a tsvector column
- Use a trigger to update the column

Create an Index on the tsvector Column
```
create index messages_tsv_index
on messages
using gin(tsv);
```
- The language parameter is no longer required

More Functions
- `setweight(tsvector, "char")`
  - A: 1.0
  - B: 0.4
  - C: 0.2
  - D: 0.1
- `ts_rank(tsvector, tsquery)`
- `ts_headline(text, tsquery)`

Function Examples
- Set the weight of subject to be "A" and the weight of content to be "D"
- List the results by their relevancy scores and highlight the query terms in the results

Using Native SQL in JPA
```
String sql = "select * from employees where id = ?";
entityManager.createNativeQuery(sql, Employee.class)
.setParameter(1, employeeId)
.getResultList();
```

Named Query in Entity Class
```
@NamedQueries(
    @NamedQuery( name="employee.findById",
        query="from Employee where id = :id" ),
    @NamedQuery( name="employee.findAll",
        query="select * from employees" ),
)
public class Employee {
    // ...
}
```
- A named query can be JPQL or SQL.
Named Query in Hibernate Mapping File

```xml
<sql-query name="message.search">
  <return class="Message" />
  <![CDATA[
    select * from messages
    where tsv @@ plainto_tsquery(?)
  ]]>}
</sql-query>
```

Using Named Query in DAO

```java
entityManager
  .createNamedQuery("employee.findAll", Employee.class)
  .getResultList();

entityManager
  .createNamedQuery("employee.findById", Employee.class)
  .setParameter("id", employeeId)
  .getSingleResult();
```

Example: Course Search in CSNS2

- Course
- CourseDao and CourseDaoImpl
- NamedQueries.hbm.xml
- csns-create.sql

FTS in Databases vs. Standalone Libraries

- Pros??
- Cons??