Full Text Search and Indexing in Languages With Two Alphabets

Tijana Talić

Paneuropean University APEIRON, Banja Luka, e-mail: tijana.t@apeiron-uni.eu

Case study

DOI: **10.7251/JIT1401041T** UDC: **004.738.5.057.4**

Abstract: The languages spoken in Bosnia and Herzegovina use both Cyrillic and Latin equally. This is an additional problem with indexing and full text searching. In this paper, we are analyzing this problem. Using the tools available on PostgreSQL and ispell dictionaries, we made a solution. As part of the solutions, we created a dictionary of stop words, adjusted the affix file for both alphabets and from the list of words made functional vocabularies for indexing and searching. We made a full search configuration which is useful for indexing texts in both alphabets.

Key words: Semantic full-text search; Indexing; Artificial intelligence

With the rapid development of information technology there has been an exponential increase of the available data. To process and use such information, we need a new and faster way of using the data. One of the current problems is text searching, which apart from the speed requires a minimum disk space. This technology has emerged as a necessity due to the presence of a large number of digital documents we have.

Full text search provides the ability to identify the documents in natural, spoken languages, that satisfy the search condition, and also it sorts them according to the query criteria. The most common type of search is: finding all documents containing specified query terms and returning them in order of similarity to your inquiry. Query terms and similarities are very flexible and depend on the particular program. The simplest search observes the query as a set of words, and it observes the similarity as the word frequency listed in the document.

This technology has an increasing importance, especially in archiving overall achievements of a particular community, or of all humanity, whether they are in domain of art or scientific research and achievements.

Operators for text search in databases have existed for years. Operators "Like (~, ~ *)" and "iLike" for text data types, are available in most databases, but they lack many essential features required for modern information systems:

- There is no linguistic support, even for the English language. Regular expressions are not enough, because they cannot process derived words easily. It is possible to use OR in the search for more derived forms, but this is demanding and inaccurate (some words can have several thousand derivatives);
- There is no ranking (sorting) of search results, which makes them ineffective, especially when it has been found over thousands of documents that contain the searched terms;
- They are slow because there is no support for indexes and they must process each document at each time.

When we talk about full text search system a document is defined as a unit of searching whether it is in terms of a data base column or text file.

Full text searching in documents can be made directly, without prior processing, i.e. without index-

ing. In this mode, text search is performed by successive reading and comparison with query criteria. The main feature of this search method is that it is very slow. For any criteria change we reiterate the reading and search again.

Full text indexing allows documents to be preprocessed and obtained indexes to be saved for later quick search. In this way, practically we have a processed document located in an index base in which we perform only the final comparison. This avoids constant reinterpretation of the document. Document Processing for full text comparison and indexing includes:

- Parsing document to tokens. It is useful for identifying different classes of tokens, for example, numbers, words, compound words, e-mail addresses. Each of these categories is treated differently in the further search. In general, the category tokens depend on the application. The most modern databases have a predefined category. For example, PostgreSQL has a defined category of tokens for the majority of searches.
- Conversion of tokens into lexemes. Lexeme is a string just as a token, but it is normalized so that different forms of the same word are equal. For example, the normalization almost always involves the conversion of large to small letters, and often includes the removal of extensions. This allows search engines to find various forms of the same word, without boring entering all possible versions. Normalization regularly includes the elimination of so-called stop words. Stop words are those words that frequently occur in the document and whose search is pointless, such as auxiliary verbs, conjunctions ... For this purpose there are special programs called dictionaries.
- Indexing involves saving pre-processed documents optimally adapted to search. Each document is presented as an organized row of normalized lexemes. For the purpose of doing the ranking range it is desirable with the lexeme to save the data about the place of their occurrence. A document that contains part in which the term appears frequently is ranked higher than the one in which the term is scattered in the text. This technology significantly uses *ispell* and *stem* dictionaries, which are, unfortunately for the languages spoken here, generally arranged poorly.
- Comparison occurs to this group of normalized lexemes. Criteria queries are processed in the same manner and the words within it are also converted to lexemes, then the comparison is made. This is the way in which the adjustment of the natural spoken language research is made. For the realization of this process we use:

Parsers

Parsers are responsible for dividing the document to tokens and for the recognition of the tokens' type. The set of possible token types is defined by the parser itself. It should be noted that the parser does not modify the text, but only determines the acceptable word barrier.

Dictionaries

Dictionaries are used to normalize words and remove words that should not be taken into account during the search (stop words). Normalization does not always have linguistic meaning, and generally depends on the semantics of the application.

Dictionary is a program that accepts input symbols and it returns:

- • *string* of lexemes if the token is known to the dictionary (one token can produce more lexemes)
  • an empty string if the token is known to the dictionary but it is recognized as a stop word
  • NULL if the dictionary did not recognize the symbol

The stop words are words that are very common, occurring in almost every document, and they are irrelevant to the search. Thus, they can be ignored in the context of the full text search.

Simple Dictionary converts the uppercase letters of the input token to lowercase and checks it in the dictionary of stop words. If the token is found in the file then it returns an empty string, or token is rejected. If not, then the token converted to lowercase returns as lexeme. Dictionary can be configured

to report all regular words as unrecognized, allowing them to be passed to the next dictionary in the list.

Synonym Dictionary is used to create dictionaries that replace the word to its synonym.

Thesaurus is a collection of words that contains information about the relationship of words and phrases.

Basically thesaurus replaces all non-priority terms to prioritize one and optionally keeps the original terms for indexing. The current implementation of PostgreSQL vocabulary thesaurus is a dictionary of synonyms expansion by adding support for the phrases.

Template ***Ispell dictionaries*** supports morphological dictionaries, which can normalize the different linguistic forms of the word in the same lexeme. For example, English Ispell dictionary can match all declinations and conjugations of the search term bank, for example: Banking, banked, banks, banks' and bank's. ***Snowball dictionary*** template is based on the Martin Porter's project. He is the creator of the popular Porter stemming algorithm for English. Snowball now provides stemming algorithms for many languages. Each algorithm understands how to reduce common variations of word forms to its base or stem, using the language spelling. Snowball dictionary requires a language parameter to identify which stemmer to use, and if it is necessary it may indicate the term of the stop word file that provides a list of words that should be removed.

Configuration

Functionality, dictionary combining and adjusting the needs in the PostgreSQL, is done by configuration. Configuration determines how the search is performed, by which dictionaries and in what order. Dictionary can have multiple different configurations and depending on the application, we can define different types of configurations. For example, for search mathematical texts we can define a special dictionary.

In this paper, for the purpose of archiving systems, we consider making a full text search configuration in conditions of the use of languages with two alphabets out of English and Russian speaking areas. On the territory of Bosnia and Herzegovina, Cyrillic and Latin alphabet are used equally. We want to create a configuration that will perform the search on both alphabets. For this purpose, we use PostgreSQL. Its built-in full text search system is very flexible. It allows the definition of the user dictionaries and making configuration of the text search by the combination of these dictionaries.

Parser RDBMS satisfies our needs.

The basis of every search is right dictionary. An indispensable element of every dictionary is a dictionary of stop words. In our case, this dictionary had to be made from the scratch, because in the available resources we have not found anything. Using grammar, spelling of the language and the method of text analysis in the spoken language, we defined a dictionary of stop words for general purposes. It looks like this:

a	ja	mojem	oni	ste
ako	je	mom	ono	su
ali	jer	na	onom	t
bi	jeste	nama	ova	ta
bismo	ji	naše	ovaj	taj
biste	k	ne	ove	tako
biti	ka	ni	ovi	te
će	kad	niti	ovim	tebi
ćemo	kako	njega	ovo	ti
ćeš	kao	njemu	ovom	to
ćete	kod	njihov	pa	tom
ću	koja	njihovi	po	u
da	koje	njihovom	pri	uz
do	koji	njima	s	vam
g	kojima	njoj	sa	vama
ga	koliko	o	sam	vaše
i	kom	od	se	vi
ih	kome	on	še	za
ili	li	ona	si	
iz	mi	one	smo	

Table 1. Stop words dictionary

For the forming of the configuration, we use ISPEEL configuration template. For the *affix* file, we exploit a file that is for the Croatian language made by Denis Lackovic [2]. List of words we took from the available *ispell* and *myspell* sources. We made two dictionaries, one for Latin and the other for Cyrillic. The basis of the Cy-

rillic *affix* file is taken from the previously mentioned file by Denis Lackovic.

Then, we process the list of words with this command:

```
munchlist -l ./bsh.affix bsh-list.dict > bsh.dict
munchlist -l ./bsh_c.affix bsh-list_c.dict > bsh_c.dict
```

In this way we get two configuration files for the vocabulary. We played with a thesaurus. For example, often in search, if we search for "bijelo", we want to find also "belo". Sometimes in the search for specific purposes we want to index some of the words equally in the Cyrillic and Latin text. For verifying, we made a dictionary of synonyms:

carina	carina
carinski	Carinski
carinske	Carinske
zakon	Zakon
pravilnik	Pravilnik
procedure	Procedure
postgresql	Pgsql
postgres	Pgsql
belo	Bijelo
mleko	mlijeko

Table 2. Synonym dictionary

Note that we only use dictionary for verifying functionality and there is no linguistic meaning.

Now we will create configurations of dictionaries:

```
CREATE TEXT SEARCH DICTIONARY bsh (
      TEMPLATE = ispell,
      dictfile = 'bsh', stopwords = 'bsh',
afffile = 'bsh');
CREATE TEXT SEARCH DICTIONARY bsh_c (
      TEMPLATE = ispell,
      dictfile = 'bsh_c', stopwords =
'bsh_c', afffile = 'bsh_c');
CREATE TEXT SEARCH DICTIONARY bsh_syn (
      TEMPLATE = synonym,
      synonyms = 'bsh');
```

We created configurations of dictionaries for two imaginary languages and a thesaurus. Now, we can configure full-text search.

```
CREATE TEXT SEARCH CONFIGURATION bsh (
PARSER = "default");
ALTER TEXT SEARCH CONFIGURATION bsh ADD MAP-
PING FOR asciihword WITH bsh_syn,bsh,bsh_c;
ALTER TEXT SEARCH CONFIGURATION bsh ADD MAP-
PING FOR asciiword WITH bsh_syn,bsh,bsh_c;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR email WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR file WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR float WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR host WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR hword WITH bsh_syn,bsh,bsh_c;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR hword_asciipart WITH bsh_
syn,bsh,bsh_c;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR hword_numpart WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD MAP-
PING FOR hword_part WITH bsh_syn,bsh,bsh_c;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR int WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR numhword WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR numword WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR sfloat WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR uint WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR url WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR url_path WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR version WITH simple;
ALTER TEXT SEARCH CONFIGURATION bsh ADD
MAPPING FOR word WITH bsh_syn,bsh,bsh_c;
```

Above we configured all types of tokens, although for our purposes it would be useful to exclude some types of tokens.

Let us look at the result of this configuration in the example of the noun "izvor" and its cases:

```
SELECT * FROM to_tsvector('izvor izvora
izvoru izvore izvorom');
```

```
          to_tsvector
---------------------------------
 'izvor':1,2,3,4 'izvori':2,3,4,5

SELECT * FROM to_tsvector('izvor izvora
izvoru izvore izvorom');
          to_tsvector
---------------------------------
 'izvor':1,2,3,4 'izvori':2,3,4,5
```

We can see that cases of the word "izvor" give as a result two lexemes: nominative of singular and nominative of plural, indicating that this dictionary works as expected.
Let us look at how our dictionary processes query condition:

```
SELECT * FROM to_tsquery('bsh', 'izvor &
izvoria');
          to_tsquery
---------------------------------
"'izvor' & 'izvorima'"
```

It is as we expected.
Now if we perform the comparison we got:

```
SELECT to_tsvector('bsh' , 'izvor izvora
izvoru izvore') @@ to_tsquery('bsh' , 'iz-
vor');
     ?column?
----------
```

From this, we can see that our configuration successfully handles with language, when is concerned more derivative of the same word. In this case, it successfully identifies the different word cases of the same word. Further, by detailed analysis, we determined that we should continue to work on developing *affix* configuration file, which is essential for machine recognition of linguistic features of spoken language.

CONCLUSION

In this paper, using the tools available in the RDBMS PostgreSQL and *ispell* system we made a configuration: such that each text is indexed in the alphabet by which was actually written.

If we would like to index all text in one alphabet, we had to work on the construction of custom parsers that in determining of the token perform transliteration. The process of transliteration is slow and tedious. The configuration that we composed in this paper, has the best use if we make application for searching that forward two queries. We transliterate the original query from one alphabet to another and forward to compare both queries. We merge the obtain result and display to the user. With this, we satisfied determined conditions for ensuring the preservation of the language purity. At the same time, the application provides an efficient search despite the alphabet of the language. The text indexes remain consistent with the original texts and enable an access to the index through another application in a different way.

Authorship statement
Author(s) confirms that the above named article is an original work, did not previously published or is currently under consideration for any other publication.

Conflicts of interest
We declare that we have no conflicts of interest.

LITERATURE

[1] Talic, T. (2014), Digitalno arhiviranje i upravljanje dokumentima, M.Sc. thesis, Banja Luka, Panevropski univerzitet Apeiron

Internet sorces:

[2] http://cvs.linux.hr/spell/ispell/croatian.aff, last accesed 18.05.2014, Lackovic, D. (2014), Croatian affix file for International Ispell,

[3] http://www.postgresql.org/docs/9.3/interactive/, last accesed 18.05.2014., The PostgreSQL Global Development Group, PostgreSQL 9.3.4 Documentation

[4]

Submitted: May 17, 2014.
Accepted: May 21, 2014.