RAWDATA Assignment 5 – Information Retrieval

The purpose of this assignment is to start working with IR as well as to indicate directions for treating IR-extensions in the Portfolio project. To start working, download the file words.sql and import it in MySQL to your local portfolio database.

Hand-in on Moodle no later than is April 16 at 16:00. One submission per group.

Question 1

- IIR Exercises 1.1, 1.2 and 1.3

Question 2

- IIR Exercises 2.1, 2.2 , 2.3 , 2.8 and 2.9

Question 3

Consider the words-table. It provides data that can be used to build inverted indexes. In principle the table also, as it is, comprise a combined inverted index for posts and comments. The columns tablename and what are used to distinguish what an entry refers to (either the title or body in posts or the text column in comments). The column id is the id of the comment or post where the word appear. The position of the word, given as sentence and word number, is encoded in sen and idx. Additionally, lexical info about part-of-speech (also called word category) and lemma is given in pos and lemma columns respectively.

a) Write two SQL-queries on words: In what posts does the word “millisecond” and in what does the word “instrumentation” appear?
b) Draw what corresponds to postings lists for each of the words in a).
c) What is the postings list corresponding to the Boolean query “millisecond AND instrumentation”?
d) Write an SQL query on the words table that retrieves the id for posts in the result of c).

Question 4

Consider again the words-table.

a) Notice that a reasonable assumption would be that a lemma always should be the same for a given word and a specific pos (word category). How can this assumption be verified against the actual content of the words table?
b) How can the assumption in e) be expressed as a functional dependency and what would a normalization of the words relation with respect to this dependency lead to?
c) Give candidate keys for the result of f).

Question 5

In this exercise, we will build and use a (simplified) positional inverted index on posts.

a) Use the table words as source to build a positional inverted index on the title column in posts (so ignore tablename and what columns). Store this index in a new table and call it **mwi** (my word index). You’ll need idx but you can ignore the sen column because we’ll assume that titles always consists of a single sentence. You can also ignore the lexical info from the words table (pos and lemma). Consider only words purely formed by letters, that is, any character of the word should be among A-Z or a-z. HINT: the simplest approach here is to use the “create table xx as select …” construct in SQL.

Use your new post title positional inverted index to answer the following queries. Don’t use any other tables to answer the queries.

b) What are the words starting with ‘data’ in the post titles?

c) What are the words appearing in titles together with the word 'database'?
(Hint: You'll need two copies of the mwi table. One approach is a construct like:
… **from** mwi m1, mwi m2 **where** m1.id=m2.id **and** …
d) What are the words appearing in titles after the word 'database'?
e) What are the words appearing in titles immediately after the word 'database'?
f) What are the words appearing in titles within a proximity of 3 words (no more than 3 words away) from the word 'database'?
g) What are the titles where the 'database' is followed by the word 'file'
(Hint: Elaborate on d) and join with posts to get title – for instance nested with **in**).

Question 6

In this exercise, we will use **mwi** again and build another inverted index **mwib** on the body column of posts. The main challenge here is to consider our small new database consisting of these two tables and to query and optimize this to efficiently process certain queries. (So we consider here inverted indexing as well as database indexing).

a) Use again the table words as source to build a positional inverted index on the body column in posts. Store this index in a new table and call it **mwib** (my word index on body). You'll need the idx as well as the sen column since body-values may include multiple sentences. You can again ignore the lexical info and restrict to words purely formed by letters as in **mwi**.

Notice that you now have two tables without any indexing. You can use the "create index …." command to add indexing to a table ("alter table" if you want to introduce primary key, but this is not necessary here). "Drop index" can be useful when experimenting.

b) Consider the following query that retrieves id, word and position (sen and idx) for all sentences where the word 'database' appears and try to process the query:
select m1.id, m2.word, m2.sen, m2.idx from mwib m1, mwib m2 where m1.id=m2.id and m1.sen =m2.sen and m1.word ='database';
You probably get bored waiting for the answer for this type of "words appearing together in sentences" queries. What can you do?
c) Consider the query:
select mwi.word from mwi
where mwi.word not in (select mwib.word from mwib where mwi.id=mwib.id);
Explain the answer to this. Try to process. Do you need further optimization to process this?
Why / Why not?
d) The following (rather weird) query joins mwi and mwib
select mwi.word from mwi, mwib
where mwi.word=mwib.word and mwi.word like "d%";
Try to process it. Do you need further optimization? And in that case, what?

Question 7

The section "Preparing the data" in the IR-slides gives an example of a minimalistic data preparation with the creation of the inverted index table wi.

a) Explain why the result conforms to the description on the slide and give your own description of what exactly is the result from executing this SQL-code.
b) Study the content of the words table and consider the filtering performed to reduce words to wi. Discuss alternatives to this filtering. Does it exclude to much? Can it be refined to exclude more useless words?
c) A word that appears in a post implies a single entry in wi – regardless of how many occurrences there are of the word in the post. How can you modify the creation of wi to make it include also the number of occurrences of the word in a post?
d) Download the file stopwords.sql from Moodle and import it to your database to get a set of candidate stopwords. How can you change the creation of the wi table so that you

take stopwords into consideration? What are the advantages and disadvantages of doing this?

e) The inverted indexing by wi uses words as they appear in posts as entries. Build an alternative lemmatized index li? Compare wi and li and discuss what is better.

f) In the inverted indexing by wi all posts, questions as well as answers, are individually indexed. Suppose you rather would prefer to index question posts only, but in such a way that every word from the question as well as the answers referring to this are entered in the index. How can you change the definition of wi to accommodate for this?

Question 8

Consider the bestmatch3 procedure code in the slides section "Ranking and Ordering".

a) What are the possible values for the rank?
b) Explain why the SQL expression lead to the desired ranking.
c) Modify the code such that it takes 5 rather than 3 keywords.

Question 9

The section in the IR-slides "Dynamic procedure – towards a solution" includes code to a stored procedure called **rewrite**. The dummy replacement substrings can be substituted by partial SQL expressions to make rewrite build an SQL-query string.

a) Try, with inspiration from the procedure **rewrite**, to develop a stored procedure bestmatch that can **generate a string representing an SQL-expression** as in bestmatch3, but with any number of arguments (2 or more).
b) Consider the example code for a simple procedure **find** in the IR-slides' section "Dynamic procedure – towards a solution". With inspiration from this, try to modify your code from a) so that a call of the procedure implies that firstly a dynamic SQL-expression is generated and secondly this expression is also executed.

Question 10

- IIR Exercises 6.9, 6.10 and 6.12

Question 11

Consider your database schema from Portfolio 1 with the addition of the words table and maybe a simplification like the wi table introduced in the slides.

a) How would you change the model if you were to include a *score*, indicating the relevance of words to posts by a weight between 0 and 1.
b) Consider again the code for bestmatch3 in the slides section "Ranking and Ordering". Given the *score* from a), how could you modify this code so that the weighting is taken into consideration?

Question 12

Given the introduction of *score* in question 12,

- What would be an approach to compute and store this?