ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Information Retrieval
Winter 2019/2020

Prof. Dr.-Ing. Klaus Berberich
Telefon: 06 81 58 67-243
klaus.berberich@htwsaar.de

Programming Assignment 3

The programming assignment will be discussed on **January 9**. To obtain bonus points, you have to submit your solution via e-mail by **January 7 at 12:00 (noon)**. Teams of up to three students are allowed.

Aufgabe 3.1 Inverted Index (2 Points)

In this exercise, you'll implement an inverted index based on the database created for the last programming assignment.

(a) Implement a class `Posting` which has the following attributes and provides suitable getter/setter methods:

- `long did`
- `int tf`

(b) Implement a class `InvertedIndex` which provides the following methods:

- `List<Posting> getIndexList(String term)` : returns the document-sorted posting list for the given term
- `int getDF(String term)` : returns the document frequency of the given term
- `int getSize()` : returns the size of the document collection
- `int getLength(long did)` : returns the length of the given document

(c) Using the `CREATE INDEX` command, create suitable indexes in your SQLite database on the tables `docs`, `tfs`, `dfs` to speed up the methods provided by `InvertedIndex`.

Hint: `CREATE INDEX idx ON t(a, b, c)` creates an index on the columns `a`, `b` and `c` of the table `t`. SQLite uses a B-Tree as an index structure to this end. Such an index allows you to efficiently retrieves rows for a specific value of `a`, a specific combination of values of `a` and `b`, and a specific combination of values for all columns. If no other columns from the table are involved in the query, the system can process the query solely by using the index.

Aufgabe 3.2 Query Processing (2 Points)

In this exercise, you'll use the inverted index to process queries.

(a) Implement a class `Accumulator` which has the following attributes and provides suitable getter/setter methods:

- `long did`
- `double score`

(b) Implement a class `QueryProcessor` which provides the following methods:

- `List<Accumulator> process(String query)` : returns the complete query result sorted in descending order of the attribute `score`.
- `List<Accumulator> process(String query, int k)` : returns the top-k query result, consisting of only the documents having the k highest scores, sorted in descending order of the attribute `score`.

As a retrieval model, we will use the following variant of tf.idf:

$$score(q, d) = \sum_{v \in q} tf(v, d) \cdot \log \frac{|D|}{df(v)}$$

Assume disjunctive query semantics and use Term-at-a-Time query processing to compute the query result. To this end, read each posting list in its entirety. When you first encounter a document, create an `Accumulator` for it and set its partial score. When the same document is encountered later on in another posting list, the score in its `Accumulator` needs to be updated.

Please remember to apply the same tokenization and normalization (cf. Programming Assignment 1), which you used for documents, to pre-process the query.

Once you've determined the full query result, sort it or extract the best k documents from it.

(c) Compute the top-5 query results for the following queries:

- `olympics opening ceremony`
- `denmark sweden bridge`
- `tokyo train disaster`