Information Retrieval with PostgreSQL

Alexander Hebel

Heidelberg University
Institute of Computer Science
Database Systems Research Group
vx228@uni-heidelberg.de

Mai 6, 2020

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Information Retrieval System (IRS)

Information Retrieval (IR)

- Information retrieval is the science of searching for information in a document
- An IR system is a software system that provides access to books, journals and other documents; stores and manages those documents.
- Web search engines are the most visible IR applications.

Full-Text-Search

The activity of searching through a collection of natural-language documents to locate those that best match a query

Objectives

To our best knowledge there exists no IRS based on a relational database.

- Finding different suitable database models
- Python api for the database creation and communication
- Crawl Wiki pages to gather text data
- Store and manage those text documents
- Support search querys containing the boolean operator AND
- Create a ranking function for the query results
- Compare the ranking of whole Wiki pages and their sections

Tsvector and Tsquery

- PostgreSQL provides **two** data types that are designed to support full text search
- The **tsvector** type represents a document in a form optimized for text search
- The **tsquery** type similarly represents a text query

Example Tsvector

```
SELECT to_tsvector('english', 'The Fat Rats');
{'fat':2 'rat':3}
```

Example Tsquery

```
SELECT to_tsquery('Fat & Cats');
{'fat' & 'cat'}
```

Tsvector Pro and Contra

Possibilities

- Full text search
- GIN-Index
- Automatic tokenization and lemmatization
- Adding weights
- Predefined ranking function

Limitations

- The number of lexemes must be less than 2^{64}
- Max position value: 16383
- No more than 256 positions per lexeme
- Relative small set of manipulation methods
- Ranking function must be customized

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Text Data Statistics

Document Definition

A document in my case is either a page title, seciton title or section text

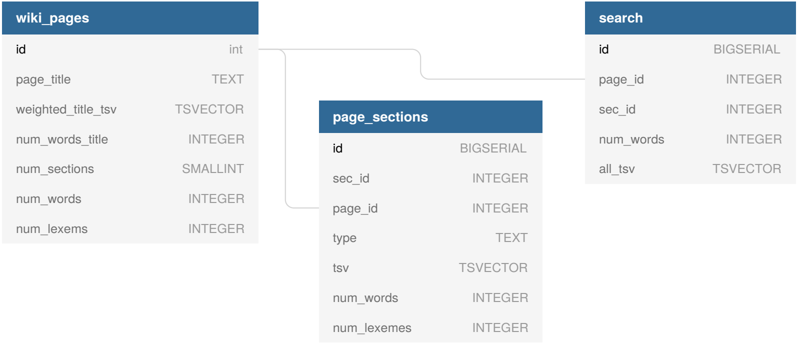
Wikipedia Category "Sports"

- Number of Wiki pages: 2000
- Number of sections (captions also count as section): 45756
- Total number of lexemes: 1969427

- Max number of words of a page: 124136
- Max number of lexemes of a page: **17787**

- Max number of words of a section: 24163
- Max number of lexemes of a section: **2785**

Database Model



wiki_pages	
id	int
page_title	TEXT
weighted_title_tsv	TSVECTOR
num_words_title	INTEGER
num_sections	SMALLINT
num_words	INTEGER
num_lexems	INTEGER

page_sections	
id	BIGSERIAL
sec_id	INTEGER
page_id	INTEGER
type	TEXT
tsv	TSVECTOR
num_words	INTEGER
num_lexemes	INTEGER

search	
id	BIGSERIAL
page_id	INTEGER
sec_id	INTEGER
num_words	INTEGER
all_tsv	TSVECTOR

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Reason to Extend PostgreSQL

Only a few manipulation methods for tsvectors exist

For example it is not possible to:

- count the occurrence of **all** lexemes in a tsvector (only the number of distinct lexemes)
- create a tsvector with an offset for the position index
- get the max index of a tsvector
- concatenate two tsvectors without changing the position values

Solution

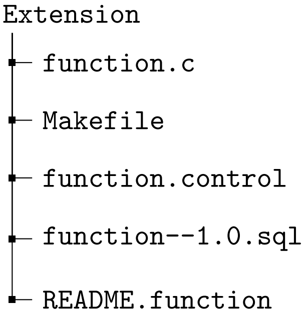
Write your own function either in plpgsql language or in C

Custom C-Functions in PostgreSQL

Prerequisites

- Developer version of PostgreSQL
- Installation of make
- Root privilege on database

Folder structure



```
Extension
├─ function.c
├─ Makefile
├─ function.control
├─ function--1.0.sql
└─ README.function
```

Steps

(1) make install

(2) CREATE EXTENSION ”extension”

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Evaluation Objectives

- Three different database models have been tested
- The model mainly using tsvector has proven most suitable
  - + GIN index for performance
  - + ts_rank() function for ranking
  - + ts_query boolean and even phraseto_tsquery() support

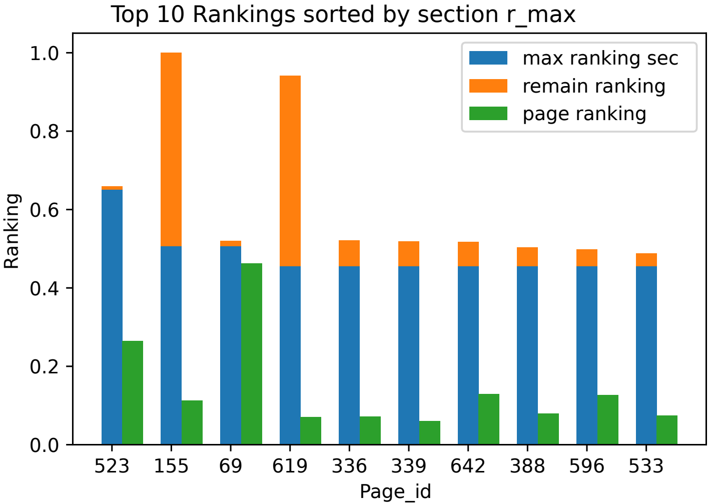
Calculation of Ranking

- Function ts_rank() of PostgreSQL
- Considers word occurrences and distance between words

Relationship Between Page and Section Ranking

- **section ranking**: ranking / num_words_of_section
- **page ranking**: sum_of_rankings / num_words_of_page

Query:"game", Sorted by Max Section Ranking



Distance Between Rankings

Calculating Distance

- Two sorted lists (section_rankings and page_rankings)
- Distance from first section in section_rankings to position of page in page_rankings containing this section and so on...

Query "game"

- result pages: 1176
- Top 10 - 289.5 avg(dist)
- Top 20 - 250.9 avg(dist)
- Top 30 - 183.2 avg(dist)
- Top 40 - 155.5 avg(dist)

Query "game & team & ball"

- result pages: 274
- Top 10 - 36.0 avg(dist)
- Top 20 - 40.6 avg(dist)
- Top 30 - 36.6 avg(dist)
- Top 40 - 34.0 avg(dist)

Outline

1. Introduction
2. Approach and Realizations
3. Adding Functionality to Tsvectors
4. Evaluation Objectives
5. Conclusion

Conclusion and Future Work

Conclusion

- An IR system made of a relational database is possible
- Rankings for sections and page return total different results
- Tsvector has a lot of potential
- PostgreSQL is easy customizable
- Solr gives a little bit different ranking results

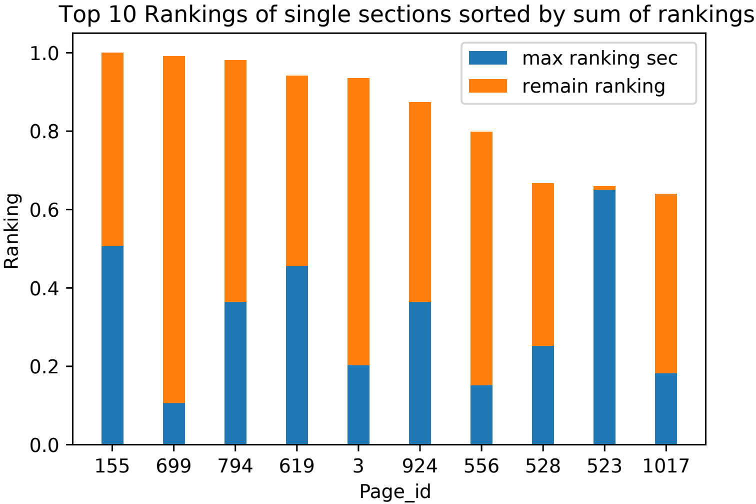
Future Work

- Improve the ranking algorithm with tf idf information (ts_stat)
- Boost certain terms of a query
- Proximity query (aka sloppy phrase query)
- Tests on big datasets

Questions

Questions

Query:"game", Sorted by Sum of Section Rankings


Top 10 Rankings of single sections sorted by sum of rankings
max ranking sec
remain ranking
Ranking
Page_id