Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Štempelj

Učinkovito preiskovanje polnotekstovnih podatkov v splošnonamenskih podatkovnih sistemih

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Iskanje in opravila nad splošnimi teksti postajajo vedno pomembnejši del analitičnih aplikacij. V ta namen obstajajo specializirani podatkovni sistemi (npr. Apache Lucene), na katerem temeljita preiskovalni infrastrukturi Solr in Elasticsearch. Po drugi strani pa večina sodobnih sistemov za upravljanje s podatkovnimi bazami (npr. PostgreSQL, MySQL/MariaDB, MongoDB) omogoča kreiranje posebnih (full text) indeksov nad tekstovnimi vsebinami in relativno močno (v smislu poizvedovanja) ter zelo hitro poizvedovanje v naravnem jezik. Raziščite možnosti prilagoditev sodobnih odprtokodnih SUPB za delo in polnotekstovno preiskovanje vsebin v slovenščini in jih preizkusite na praktičnem primeru kolokacije besed. Svoje rešitve ovrednotite po različnih scenarijih na več korpusih besedil v slovenskem jeziku.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Blaž Štempelj sem avtor diplomskega dela z naslovom:

Učinkovito preiskovanje polnotekstovnih podatkov v splošnonamenskih podatkovnih sistemih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. marca 2016 Podpis avtorja:

Zahvaljujem se vsem, ki so mi v času študija stali ob strani in me podpirali na moji poti. Hvala mentorju izr. prof. dr. Matjažu Kukarju za vso pomoč pri nastalih težavah med izdelavo diplomskega dela. Neskončno hvala celotni družini za izredno veliko spodbudo na celotni življenjski poti, hvala vsem prijateljem za nepozabne trenutke. Hvala tudi Simonu Kreku za pomoč pri delu s korpusi in Roku Rejcu iz podjetja Amebis za pomoč pri obrazložitivi delovanja korpusov Gigafida in Kres. Hvala tudi Alešu Permetu iz podjetja bolha.com za pomoč pri delu s podatkovnimi bazami in Maruši Grah prav tako iz podjetja bolha.com za vse spodbude in omogočen prosti čas za izdelavo diplomske naloge.

Mojim dragim staršem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
DBMS	database management system	sistem za upravljanje podatkovnih baz
GIN	generalized inverted index	posplošen obrnjen index
API	application programming interface	vmesnik za programiranje aplikacij
JSON	JavaScript object interface	objektni JavaScript vmesnik
BSON	binary JSON	binarni JSON

Slovar jezikoslovnih izrazov

izraz	definicija
koren besede	del besede, ki je vsem besedam iz iste jezikovne družine skupen po pomenu in zapisu
lema besede	osnovna oz. slovarska oblika besede
normalizacija besede	preslikava besede v njeno osnovno obliko (lemo)
leksem	množica vseh oblik besed, ki imajo enak pomen

Povzetek

Cilj diplomske naloge je pregled in vrednotenje možnosti, ki nam ji za delo s teksti v naravnem jeziku ponujajo splošnonamenski sistemi za upravljanje s podatkovnimi bazami. V prvem delu opišemo slovenska korpusa ccKres in ccGigafid, shemo shranjevanja tekstov iz korpusov v SUPB-je MariaDB, PostgreSQL in MongoDB ter polnotekstovnim indeksov v posameznem SUPB-ju. Vendar podpora slovenščini še vedno ni tam kjer bi želeli. MariaDB nam omogoča le definicijo seznama nepomembnih besed, medtem kot pri MongoDB še tega ne moremo narediti. Upanje ponuja PostgreSQL, kjer z izdevalo lastne konfiguracije lahko omogočimo uporabo leksemov in s tem boljše željene rezultate. V drugem delu testiramo performanse SUPB-jev na primeru kolokacije besed, kjer rezultate predstavimo tako tabelarično kot tudi z uporabo grafov. Rezultati pokažejo, da je za kolokacijo besed naša najboljša izbira MongoDB.

Ključne besede: MySQL, MariaDB, PostgreSQL, MongoDB, polnotekstovno iskanje.

Abstract

The goal of the thesis is the review and evaluation of options that database management systems support when working with natural language texts. In the first part we describe the slovenian corpuses ccKres and ccGigafida, the database structure of MariaDB, PostgreSQL, MongoDB and their use of full-text indexes. But the support the slovenian language still isn't all that great. MariaDB supports only the use of stop words, while MongoDB doesn't even support those. With a little work, PostgreSQL enables us to define custom made configurations which enable the use of lexemes and more fine tuned results. In the second part of this thesis we test the performance of each DBMS by using colocation. Results are presented by using tables and graphs. The final results also show that for colocation the best choice is to use MongoDB.

Keywords: MySQL, MariaDB, PostgreSQL, MongoDB, full-text search.

Poglavje 1

Uvod

Iskanje po tekstih v naravnem jeziku predstavlja izredno pomemben del marsikatere današnje aplikacije. Spletni iskalniki uporabljajo iskanje celo kot glavni način interakcije z uporabnikom. Veliko ljudi si namreč ne zna več predstavljati uporabe interneta brez spletnih iskalnikov kot so Google, Yahoo in Bing. Ali bi bil internet še vedno tako uporaben, če bi imeli samo dolg seznam spletnih strani, ki jih lahko obiščemo? Seveda ne. Zaradi tega se izredno veliko pozornosti posveča tehnologijam za iskanje in opravili nad splošnimi teksti. Ravno ta potreba je bila pobuda za razvoj specializiranih podatkovnih sistemov kot je Apache Lucene. Apache Lucene je odprtokodna javanska knjižnica za izredno močno in hitro iskanje po besedilu v naravnem jeziku.[36]

Tukaj se pojavi vprašanje, če je resnično potrebujemo specializiran sistem za iskanje po tekstu. Skozi leta so namreč tudi splošno-namenski sistemi za upravljanje s podatkovnimi bazami (v nadaljevanju SUPB) dodali podporo za napredno iskanje po tekstih. Tako so MySQL[1], MariaDB[2] in PostgreSQL[3] kot predstavniki relacijskih odprtokodnih podatkovnih baz, omogočili izdelavo tako imenovanih tekstovnih indeksov. Kasneje so enako podporo dodale tudi novejše nerelacijske podatkovne baze. Najbolj znan predstavnik nerelacijskih podatkovnih baz je MongoDB[52]. Tekstovni indeksi nam omogočijo uporabo veliko močnejših izrazov, s katerimi je iskanje po

tekstih učinkovitejše in hitrejše. Večjo učinkovitost nam omogočajo funkcije kot je iskanje po frazah ali iskanje z uporabo logičnim izrazov. Poleg tega pa z uporabo seznamov nepomembnih besed pridobimo tudi na hitrosti, saj se med samim iskanjem ne upoštevajo.[29]

Namen diplomske naloge je, raziskati ali lahko že uveljavljeni SUPB-ji nadomestijo ali dopolnjujejo uporabo specializiranih podatkovnih sistemov. To bomo ovrednotili tako, da bomo za vsak uporabljen sistem (MariaDB, PostgreSQl in MongoDB) shranili enake podatke. Nato bomo izdelali tekstovne indekse in z novimi funkcionalnostmi izvedli nekaj poizvedb in primerjali rezultate. Tukaj bo zelo pomembno kakšna je podpora slovenskemu jeziku, saj so podatki s katerimi operiramo zbirka slovenskih besedil. Besedila so izšla med leti 1990 in 2011, zbrana pa so bila v okviru projekta *Sporazumevanje v slovenskem jeziku.*[4] Poleg tekstovnih indeksov bo izvedeno tudi testiranje hitrosti iskanja kolokacije besed. Kolokacija besed je pojem, ki opisuje besede, ki se pogosto pojavljajo skupaj.[11] Primeri so: *kristalno jasno, plastična operacija, krdelo volkov, ...*

V drugem poglavju bomo torej spoznali tri različne SUPB-je MariaDB, PostgreSQL in MongoDB. Predstavili bomo njihov nastanek in posebnosti. Nato sledi poglavje o testnih podatkih. V njem bosta predstavljena dva slovenska korpusa Kres in Gigafida, ki vsebujeta naše testne podatke. V četrtem poglavju o implementaciji bomo realizirali naš opis organizacije posameznega SUPB-ja. Peto poglavje bo namenjeno tekstovnim indeksom in je razdeljeno tako, da se osredotočimo na vsak SUPB posebej. Šesto poglavje vsebuje poizvedbe in meritve poizvedb na podlagi katerih bomo v sedmem poglavju določili najprimernejši SUPB za iskanje kolokacije besed v slovenskem jeziku.

Poglavje 2

Platforme

V nadaljevanju bomo pregledali tri SUPB-je, v katere bomo shranili naše podatke. Poseben poudarek bomo dali opisu funkcionalnosti za delo s podatki v naravnem jeziku in podpori za slovenščino. Najprej si bomo ogledali MariaDB, saj se je razvil iz MySQL in bo njegova sintaksa večini najbolj poznana. Sledi PostgreSQL, ki prav tako kot MariaDB spada med relacijske podatkovne baze. Ker imata MariaDB in PostgreSQL za seboj že bogato zgodovino razvoja, bomo že tukaj predvideli, da bosta primernejša za naš problem. Za njiju je namreč na voljo veliko več dokumentacije in podpore. Vendar se zna zgoditi, da nas bo MongoDB prenenetil. Ima namreč drugačen pristop za shranjevanje in poizvedovanje podatkov, ki je morda primernejši za delo z naravnim jezikom. Spada med nerelacijske podatkovne baze, saj ne uporablja koncepta tabel, ampak podatke shranjuje v dokumente.

Vse poizvedbe in testi so bili izvedeni na operacijskem sistemu Ubuntu 14.04. Razlog za izbiro starejše različice je MongoDB. To je namreč zadnja različica Ubuntu, kjer je zagotovljeno njegovo popolno delovanje. Uporabljena različica MariaDB je bila 10.1, PostgreSQL 9.3 in MongoDB 3.2[1] (izšla v času pisanja). Pri vsakem SUPB-ju bomo predstavili kratko zgodovino in

[1] V različici 3.2 MongoDB pridobiva nov mehanizem za shranjevanje imenovan WiredTiger, nova verzija tekstovnih indeksov (neobčutljivost na dialekte (ne razlikuje npr. med é, É, e, in E), ...) v vseh podprtih jezikih in podpora za še več jezikov.

glavne značilnosti. Pri tem bomo pozorni na podporo operacijam nad teksti v naravnem jeziku in slovenščini, saj delamo z besedili v slovenskem jeziku.

Ker je to delo z nestrukturiranimi podatki, ne moremo uporabljati enakih poizved kot smo jih vajeni pri delu s strukturiranimi podatki. Primer pogostege naloge je identifikacija paragrafov, stavkov in posameznih besed. Če za primer vzamemo stavek *Prst je preperel del Zemljine skorje*, morajo funkcije v SUPB za delo v naravnem jeziku ugotoviti ali je z besedo *prst* mišljen del roke ali noge, ali pa geološki izraz za tla. Ljudje ponavadi ob pravem kontekstu s tem nimamo težav, računalniku pa to predstavlja dokaj velik izziv. Zato morajo funkcije za procesiranje naravnega jezika (v nadaljevanju NLP iz ang. *natural language processing*) podpirati vsaj nekaj izmed spodnjih funkcionalnosti[37]:

- Segmentacija stavkov (sposobnost določitve konca in začetka stavka),
- razdelitev na žetone (identifikacija posameznih besed),
- odstranjevanje končnic,
- identifikacija besednih vrst (samostalnik, glagol, pridevnik, ...),
- identifikacija sintakse,
- identifikacija lokacij, časa, zaimkov, ...

Poleg opisanih funkcionalnosti pa pomembno tudi, da zna SUPB delati s slovenskim jezikom. Delo z naravnim jezikov namreč zahteva specifikacijo jezika, saj ima vsak jezik svoje posebnosti. S tem, ko specificiramo jezik, morajo funkcije za delo z naravnim jezikom prilagoditi delitev besed na žetone (npr. v primeru azijskih jezikov ni presledkov kot ločilo med besedami), drugače zaznati končnice, besedne vrste itd. Ker delamo s slovenskimi besedili mora SUPB, ki ga želimo uporabljati, zagotavljati podporo slovenščini oz. vsaj omogočiti, da jo ročno dodamo. V nadaljevanju bomo spoznali kateri izmed naših treh kandidatov pokrije večino naših zahtev.

Za delo z naravnim jezikom imamo z uporabo SQL tri načine (z uporabo MariaDB). Prvi način je uporaba operatorja `LIKE`

```
SELECT * FROM besedila WHERE stavek = '%zemlja%';
```

Na tak način poiščemo vse zapise besedil, kjer polje *stavek* vsebuje besedo *zemlja*. Z vprašaji povemo, da se beseda lahko nahaja kjerkoli v besedilu in ne samo na začetku ali koncu. Drugi način je uporaba *regularnih izrazov*. Ti od uporabnika zahtevajo že nekaj dodatnega znanja, vendar lahko našo poizvedbo naredimo veliko bolj specifično. Poiščemo lahko torej stavek, ki ima vsaj eno besedo dolžine štiri

```
SELECT * FROM besedila WHERE stavek REGEXP = '\b\w{4}\b';
```

Operator
`b` označi konec ali začetek besede, odvisno kje stoji. V našem primeru oboje.
Operator
`w` pa označi alfanumerični znak. Če podamo zraven v zavitih oklepajih še število omejimo dolžino iskanih znakov, oz. besedo specificirane dolžine. Najmočnejši način za delo z naravnim jezikom pa dosežemo z uporabo polnotekstovnega iskanja. V MariaDB to naredimo z uporabo sintakse `MATCH ... AGAINST`, ki deluje skupaj s tekstovnimi indeski

```
SELECT * FROM besedila WHERE MATCH(stavek)
AGAINST ('-Planet +zemlja' IN BOOLEAN MODE);
```

Prikazana je uporaba logičnega iskanja. Iščemo namreč stavek, ki vsebuje besedo *zemlja*, ne pa besede *planet*. Hkrati pa se avtomatično velike črke spremenijo v male.

2.1 MariaDB

MariaDB je izboljšana, drop-in[2] zamenjava za MySQL. Razvija jo skupnost MariaDB community nad katero bdi organizacija MariaDB Foundation.[12] Razvijalci MariaDB so se zavezali k temu, da vzdržujejo kar se le da popolno združljivost z MySQL. Tako je prehod na MariaDB instanten, brez

[2]Ob namestitvi MariaDB se obstoječa MySQL baza zamenja z MariaDB.

nekompatibilnosti. Hkrati to pomeni, da so vsi ukazi, vmesniki, knjižnice in API-ji, ki obstajajo v MySQL združljivi z MariaDB. Varnosti popravki, ki so bili implementirani v MySQL se prenesejo tudi v MariaDB. Po potrebi pa se nad temi popravki dodajo tudi lastni popravki specifični za MariaDB. Če se pojavi kakšna izredno kritična varnostna luknja se v najkrajšem možnem času izda nova verzija, ki vsebuje potrebni varnostni popravek.[12] Dodatni varnostni popravki niso edina prednost MariaDB pred MySQl. Ima tudi dodane funkcionalnosti poleg tistih, ki jih podpira MySQL. Na primer dodatni shranjevalni mehanizni kot sta Aria in XtraDB, izboljšana hitrost MyISAM in razni dodatki. Glavna lastnost, ki pritegne največ uporabnikov pa je popolna odprtokodnost. MariaDB nima v lasti nobena večja korporacija, kot je Oracle pri MySQL.[13]

Za delo s teksti v naravnem jeziku nam MariaDB omogoča izdelavo tekstovnih indeksov.[46] Uporaba tekstovnih indeksov doda več opcij za iskanje po tekstovnih poljih. Nove opcije uporabljamo s pomočjo nove sintakse `MATCH AGAINST`. Poleg tega nam omogoča definiranje seznama nepomembnih besed za naš (npr. slovenski) jezik. Besede na tem seznamu se pri iskanju ne upoštevajo. Več o tekstovnih indeksih v MariaDB bomo izvedeli v četrtem poglavju.

2.2 PostgreSQL

PostgreSQL temelji na projektu z vzdevkom POSTGRES, različici 4.2, ki je bila razvita na University of Californija (Kalifornijska univerza) v Berkeley Computer Science Department (Berkeley oddelek za računalništvo)[38]. Je odprtokoden in podpira večino SQL standardov nad katere doda lastne funkcionalnosti. Je objektno-relacijski SUPB, odprtokodne narave, ki je popolnoma zastonj. Razvija ga skupnost razvijalcev, ki živijo po celem svetu. Ne lasti si ga nobeno podjetje, ima pa vedno od 5 do 7 glavnih razvijalcev, ki večino skrbijo za glavne razvojne cikle.

Vsebuje veliko funkcionalnosti, ki jo ima konkurenca, hkrati pa ima tudi

svoje lastne. Primer le-teh so podatkovni tipi, ki jih uporabnik lahko sam definira. Torej imamo lahko lasten tip za črkovne znake, ki ima veliko večji razpon. Hkrati ga odlikuje tudi odlična uporabniška podpora. Zaradi svoje odprte narave lahko razvijalcem programa direktno pošljemo email (ta možnost ni bila uporabljena) in nam tako osebno odgovorijo. Ker so sami razvijali programsko opremo lahko tako dobimo veliko boljši in natančnejši odgovor, kot če bi govori s kakšno zunanjo podporo. PostgreSQL je znan tudi po tem, da ima izredno stabilne razvojne cikle, saj se ima vsak večji cikel najmanj enomesečno dobo testiranja.[32]

Tudi PostgreSQL podpira delo s teksti v naravnem jeziku. Za razliko od MariaDB uporaba tekstovnih indeksov ni potrebna, saj funkcije za delo s teksti v naravnem jeziku delujejo tudi brez njih. Je pa njihova uporaba izredno priporočljiva, saj skupaj z `GIN (Generalized Inverted Index, sl. Posplošen obrnjen indeks)`[17] indeksi pohitrijo poizvedbe. Za razliko od MariaDB, ki podpira samo definicijo seznama nepomembnih besed, PostgreSQL omogoča tudi definiranje lastne konfiguracije slovarjev. Tako lahko sami implementiramo podporo za jezik, ki ga PostgreSQL v osnovi ne podpira pri delu s tekstom v naravnem jeziku.

2.3 MongoDB

MongoDB je odprtokodna, dokumentna nerelacijska podatkovna baza.[14] Odlikujeta jo predvsem enostavna uporaba in skalabilnost. Zapis v MongoDB je predstavljen kot dokument, ki je podoben JSON objektom. To pomeni, da ima obliko polje - vrednost. Vendar MongoDB interno ne uporablja JSON, temveč lastno implementacijo imenovano BSON. Razlika je v tem, da je BSON binarni format, medtem, ko je JSON tekstovni. To mu omogoča kompaktnejše shranjevanje, večjo hitrost in lažji prenos podatkov z različnimi programskimi jeziki. Ravno ta zasnova programerjem olajša delo, saj za razliko od relacijskih podatkovnih baz ni potrebe po preslikavi iz SQL v objekte. Hkrati pa se znebimo dragih stikov med tabelami. Več dokumen-

tov skupaj tvori zbirko (collection).[20] Zbirke že v fazi planiranja zasnujemo tako, da stiki med tabelami niso potrebni. Dokumenti lahko namreč vsebujejo sezname večih vrednosti. V primeru, da še vedno potrebujemo podatke iz različnih zbirk, jih enostavno družimo na aplikacijskem nivoju razvoja aplikacije.

MongoDB zamenja transkacijski model za hitrost. To pomeni, da ne zagotavlja konsistentnosti pri hranjenju podatkov. Zaradi tega ni najbolj primeren za klasične transakcijske aplikacije (npr. bančništvo). Omogoča nam zelo enostavno horizontalno skalabilnost. To pomeni, da ga je zelo enostavno razdeliti na več različnih strežnikov. Od verzije 3.2 naprej pa ima tudi novi standardni mehanizem za shranjevanje imenovan *WiredTiger*, ki se lahko pohvali z optimizacijo razporejanja podatkov za hranjenje na disku v pomnilniku. Pri podatkih shranjenih na disku pa uporablja tudi novi kompresijski API in s tem prihrani na prostoru.

MongoDB z različico 3.2 prinaša novosti za delo z naravnim jezikov. Na voljo imamo uporabo tekstovnih indeksov v 21 jezikih, med katerimi pa na žalost še vedno ni slovenščine. MongoDB ne podpira definicije seznama nepomembnih besed, saj je ta prednastavljen in vezan na izbran jezik.[24] Poleg tega različica 3.2 podpira tudi ignoriranje črk, ki vsebujejo diaktrične znake. To so naglasi nad črkami, ki nam povedo kako se besedo pravilno izgovori.[22] Pogost primer v slovenščini je beseda je, kjer brez naglasa ali konteksta ne vemo ali beseda pomeni *jesti* ali *biti*. Če pa besedo napišemo kot jé bralcu pomagamo pri pravilni izgovorjavi besede in identifikaciji njenega pomena. V tem primeru smo uporabili naglasno znamenje *krativec* in beseda pomeni *jesti*. Za MongoDB sta torej besedi *je* in *jé* enaki.

Poglavje 3

Prilagajanje slovenskemu jeziku

To poglavje bo služilo opisu kako omogočimo delo s slovenskimi teksti v posameznem SUPB-ju. Pred začetkom opisa prilagajanja slovenskemu jeziku pa moramo definirati dva pojma, in sicer leksem[53] in lema[43]. Lema predstavlja kanonsko obliko besede, ki jo predstavljajo še druge oblike, npr. *delati*, *delam*, *delaš*, *dela*, so vse oblike istega leksema (množica vseh oblik besede, ki imajo enak pomen), njihova lema pa je *delati*. Pomemben jezikoslovni izraz je tudi *koren*[57], ki predstavlja del besede, ki je vsem besedam iz iste jezikovne družine skupen po pomenu in zapisu. Pri vsakem SUPB-ju se to naredi nekoliko drugače, začeli pa bomo z *MariaDB*.

3.1 MariaDB in slovenščina

MariaDB na žalost nima lastne podpore za slovenski jezik. Tudi dodati je ne moremo na enostaven način. Ker pa je odprtokoden projekt lahko kdorkoli to implementira, če želi. Da bi bila podpora slovenskemu jeziku enaka, kot pri ostalih jezikih bi bilo potrebno implementirati vsaj lematizacijo besed in vnaprej nastavljen seznam nepomembnih slovenskih besed. Zaenkrat pa bomo delali s funkcijami, ki jih podpira sam SUPB, kot ga dobimo. To, da nima podpore za slovenski jezik pomeni, da ne zna delati s šumniki, ne zna odstraniti končnic iz besed in nima ostalih funkcionalnosti, ki jih ponujajo

NLP orodja. Edina stvar, ki jo lahko omogočimo je filtriranje nepomebnih besed.[25] To naredimo tako, da ustvarimo tekstovno datoteko, kjer so z nanizane besede, ki jih ne želimo videti v rezultatih, ločene z novo vrstico. Mehanizma za shranjevanje, ki podpirata uporabo seznama nepomembnih (tipično so te besede *in*, *ali*, *pred*, *po*, *nad*) besed sta *MyISAM* in *InnoDB*. Postopek za aktivacijo se pri obeh nekoliko razlikuje. Začeli bomo z MyISAM. MyISAM v osnovi uporablja seznam nepomembnih besed definiran na lokaciji `storage/myisam/ft_static.c`. Če želimo torej definirati svoj seznam nepomembnih besed moramo spremeniti katero datoteko upošteva. To naredimo tako, da nastavimo spremenljivko `ft_stopword_file` v datoteki `my.cnf`[26]. Datoteka `my.cnf` se nahaja v domačem direktoriju MariaDB, ki pa je vezan na operacijski sistem[39]

```
ft_stopword_file='home/user/Documents/stopwords.txt'
```

kjer je vrednost spremenljivke pot do datoteke. Sedaj imamo omogočen lasten seznam nepomebnih besed za MyISAM. Za InnoDB je postopek nekoliko drugačen. Tukaj je korak za kreiranje tekstovne datoteke poljuben, saj InnoDB podatke bere iz tabele z vnaprej fiksirano relacijsko shemo, ki jo kreiramo v samem MariaDB[27]

```
CREATE TABLE slo_stopwords (
value VARCHAR(255)
);
```

Tabelo lahko napolnimo tako kot vsako drugo s stavkom `INSERT INTO` ali pa uporabimo tekstovno datoteko

```
LOAD DATA INFILE '/home/user/Documents/slo_stopwords.txt'
INTO TABLE slo_stopwords;
```

Zadnja stvar, ki je še potrebna je nastavitev spremenljivke `INNODB_FT_USER_STOPWORD_TABLE` na sledeči način

```
SET INNODB_FT_USER_STOPWORD_TABLE='cckres/slo_stopwords';
```

kjer najprej specificiramo bazo oziroma shemo (v našem primeru *cckres*) in nato ime tabele.

3.2 PostgreSQL in slovenščina

Za prilagajanje *PostgreSQL* slovenskemu jeziku imamo veliko možnosti. PostgreSQL ima namreč možnost dodajanja lastnih slovarjev[40]. Najprej bomo naredili slovar nepomembnih slovenskih besed. Za to potrebujemo samo tekstovno datoteko v kateri so nanizane nepomembne slovenske besede, vsaka v svoji vrstici. To tekstovno datoteko moramo shraniti s končnico *.stop*, drugače jo PostgreSQL ne bo prepoznal. Lokacija kjer se mora nahajati datoteka je */usr/share/postgresql/9.3/tsearch_data/*, kjer je 9.3 verzija PostgreSQL, ki jo uporabljamo. Ko je seznam nepomembnih besed definiran in na pravi lokaciji je za njegovo delovanje potrebna izdelava enostavnega slovarja. Deluje tako, da žetone (posamezne prejete besede), spremeni v male črke in jih primerja s seznamom nepomembnih besed. Če se žeton ujema z besedo v seznamu, vrne prazen seznam, kar pomeni, da je žeton zavržen. Drugače je žeton vrnjen kot leksem[1]. Slovar izdelamo z naslednjim ukazom:

```
CREATE TEXT SEARCH DICTIONARY public.slovenian_simple (
    TEMPLATE = pg_catalog.simple,
    STOPWORDS = slovenian
);
```

Tukaj *slovenian* predstavlja dejansko ime naše konfiguracijske datoteke na lokaciji */usr/share/postgresql/9.3/tsearch_data/slovenian.stop*. Da preverimo ali ustvarjeni slovar deluje uporabimo funkcijo `ts_lexize`, ki vrne seznam leksemov, če je žeton, ki ga testiramo, znan slovarju, s katerim ga testiramo. Če je beseda na seznamu nepomembnih besed, vrne prazen seznam. V primeru, da beseda ni znana slovarja vrne `NULL`.

```
> SELECT ts_lexize('public.slovenian_simple', 'na');
 ts_lexize
-----------
 {}
```

[1] Najmanjša enota v pomenskem sistemu nekega jezika.[53]

Novo kreirani slovar je besedo *na* ignoriral, saj je na seznamu nepomembnih besed. Poleg enostavnega slovarja je mogoče je izdelati tudi navadni ali *thesaurus* slovar sopomenk. Teh slovarjev ne bomo naredili, bomo pa se lotili izdelave *Ispell* slovarja, ki nam omogoča natančno definiranje normalizacije besed v leme. Za izdelavo *Ispell* slovarja potrebujemo dve konfiguracijski datoteki. Datoteko s končnico *dict* in datoteko s končnico *affix*. Obe datoteki lahko dobimo na uradni strani odprtokodnega pisarniškega paketa *Open Office*[33]. Slovar ustvarimo s pomočjo naslednjega ukaza, pred tem pa moramo datoteki *dict* in *affix* shraniti na lokacijo */usr/share/postgresql/9.3/tsearch_data/*.

```
CREATE TEXT SEARCH DICTIONARY slovenian_ispell (
    TEMPLATE = ispell,
    DictFile = slovenian,
    AffFile = slovenian,
    StopWords = slovenian
);
```

Vrednosti *slovenian* so imena datotek s končnicami *dict*, *affix* in *stop*. Da preverimo, ali slovar deluje ponovno uporabimo funkcijo `ts_lexize()`:

```
> SELECT ts_lexize('slovenian_ispell', 'Pogumnejši');
         ts_lexize
---------------------------
 {pogumnejši,pogumnejšega}
```

Slovar *Ispell* je tokrat poleg besede z malimi črkami, vrnil tudi še en zadetek. In sicer še besedo *pogumnejšega*, ker ima enak leksem kot beseda *pogumnejši*.

Sedaj je potrebna samo še izdelava konfiguracijske datoteke[41], ki vse prej definirane slovarje združi v en sam paket. Lastno konfiguracijo lahko naredimo na dva načina. Lahko uporabimo že obstoječo konfiguracijo, npr. za angleški jezik

```
CREATE TEXT SEARCH CONFIGURATION public.slovenian
(COPY = pg_catalog.english);
```

ali pa kreiramo novo prazno konfiguracijo

```
CREATE TEXT SEARCH CONFIGURATION public.slovenian
(parser='default');
```

Da je konfiguracija res prazna lahko preverimo z naslednjim ukazom

```
cckres=# \dF+ slovenian
Did not find any text search configuration named "slovenian".
```

Da lahko začnemo uporabljati slovarje, ki smo jih kreirali v prejšnjih korakih moramo to PostgreSQL-u povedati. To naredimo tako, da za vse vrste žetonov povemo katere slovarje želimo uporabiti in v kakšnem vrstnem redu:

```
ALTER TEXT SEARCH CONFIGURATION slovenian
ALTER MAPPING FOR asciiword, asciihword,
hword_asciipart, word, hword, hword_part
WITH slovenian_simple, slovenian_ispell;
```

Če navedemo *simple* slovar na prvo mesto moramo uporabiti še en ukaz, ki poskrbi, da če simple slovar ne najde besede vrne `NULL` in se za njim uporabi naslednji slovar. Brez te nastavitve bi vse ostale slovarje enostavno preskočili:

```
ALTER TEXT SEARCH DICTIONARY slovenian_simple (accept = false);
```

Če želimo novo ustvarjeno konfiguracijo uporabljati kot standardno konfiguracijo uporabimo naslednji ukaz

```
SET default_text_search_config = 'public.slovenian';
```

3.3 MongoDB in slovenščina

Za razliko od MariaDB, kjer lahko definiramo seznam nepomembnih besed, in PostgreSQL (definiranje lastnih konfiguracij slovarjev) pri MongoDB ne moremo narediti čisto ničesar, da omogočimo podporo slovenskemu jeziku.

Tekstovni indeksi v MongoDB so namreč vezani na že vnaprej implementirane funkcije za posamezne jezike, ki jih v času pisanja podpira[24]. Edina možnost je spreminjanje izvorne kode[28], kar pa zahteva vzdrževanje ob vsaki novi različici. Pri nadaljnem delu smo uporabljali funkcije vezane na angleški jezik.

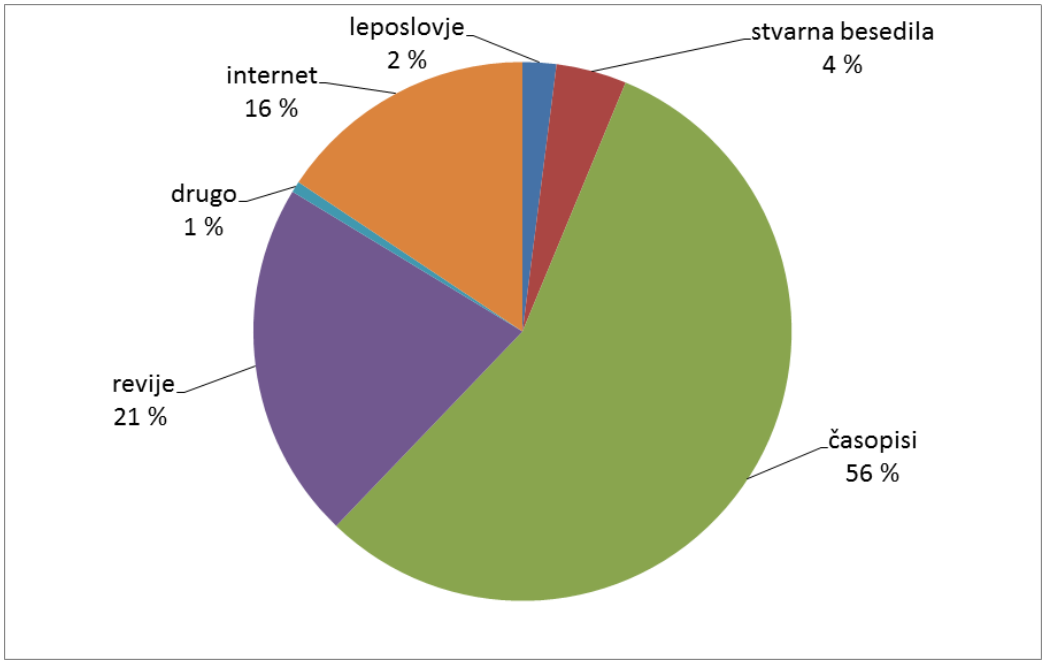
Poglavje 4

Podatki in organizacija

Preden se lahko lotimo kakršne koli analize različnih podatkovnih baz potrebujemo podatke, ki jih bomo shranili v te baze. Pomagali s spletnim portalom *slovenscina.eu.*[4] Tukaj namreč lahko najdemo elektronski zbirki avtentičnih besedil Kres[5] in Gigafida[7]. Tako zbrana besedila imenujemo tudi korpusi. Najprej bomo spoznali korpus Gigafida.

4.1 Korpus Gigafida

Gigafida[7] je spletni korpus, elektronska zbirka avtentičnih besedil, ki so bila zbrana pod določenimi merili z določenim ciljem. Gigafida vsebuje natančno 1.187.002.502 besed, kar je skoraj 1,2 milijarde besed. Za iskanje po besedilih zbranih v Gigafidi uporabljamo spletni konkordančnik.[8] Gigafida vsebuje besedila, ki so izšla med letoma 1990 in 2011, vključen pa je tudi predhodni referenčni korpus FidaPLUS (2006). Zbrana besedila so izšla v tiskanem ali elektronskem formatu. Tiskana besedila so lahko knjige, revije ali časopisi. Pridobljena elektronska besedila pa so predvsem iz novičarskih portalov večjih slovenskih podjetij in ustanov.

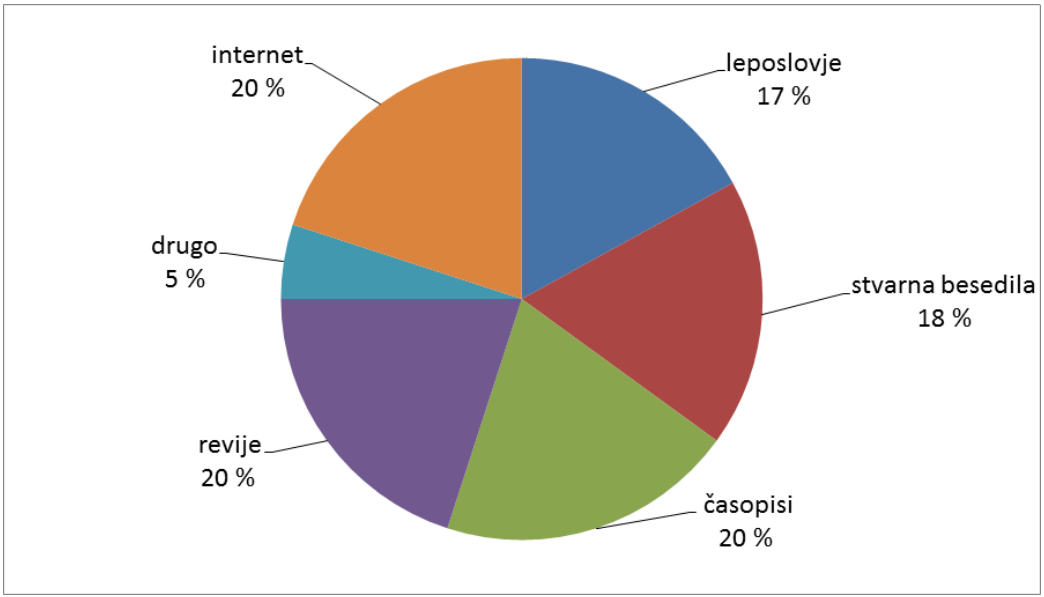


Slika 4.1: Relativne frekvence besedil v posamezni kategoriji v korpusu Gigafida.

Za diplomsko delo smo uporabili podkorpus ccGigafida, ki predstavlja 9 odstotkov del korpusa Gigafida, 31.722 dokumentov. Podkorpus ccGigafida je za razliko od celotne Gigafide, kamor so vključena tudi avtorsko zaščitena dela, prosto dostopen.[9]

4.2 Korpus Kres

Prav tako kot Gigafida je korpus Kres[5] nastal v okviru projekta Sporazumevanje v slovenskem jeziku. Projekt se je odvijal v obdobju od 2008 do 2012. Korpus Kres vsebuje natančno 99.831.145 besed, torej skoraj 100 milijonov besed. Kres je iz Gigafide vzorčeni uravnoteženi podkorpus. Ker Gigafida vsebuje 77 odstotkov besed iz periodike (časopisi, revije) in le dobrih 6 odstotkov besed iz knjig (leposlovje, stvarna besedila), je bil ustvarjen Kres. Njegovo uravnoteženost prikazuje naslednji graf:



Slika 4.2: Relativne frekvence besedil v posamezni kategoriji v korpusu Kres.

Tudi tukaj bomo uporabili podkorpus ccKres, ki predstavlja 9 odstotkov del korpusa Kres. To pomeni, da je v njem 9.376 dokumentov. Razlog za uporabo manjše zbirke je ponovno njena prosto dostopnost.[6]

4.3 Sestava korpusov

Oba uporabljena korpusa (ccKres in ccGigafida) sestavljajo XML datoteke. Vsaka XML datoteka poleg besedila vsebuje tudi informacije o viru (npr. Mladina, Delo, Dnevnik), leto nastanka, vrsto besedila (npr. leposlovje, revija), naslov in avtorija. V nekaterih primerih je avtor neznan.

Poleg dodatnih podatkov o besedilu je vsaka beseda jezikovno označena. Torej ima vsaka beseda še dva dodatna podatka. Prvi je osnovna oblika besede ali lema (npr. ribe, ribi, ribam = riba), drugi podatek pa je oblikoskladenjska oznaka. S to oznako vemo v katero besedno vrsto spada beseda (samostalnik, glagol, pridevnik itd.) in njene lastnosti (npr. spol, število, sklon). Za razdelitev na žetone, oblikoslovno označbo in določitev leme be-

sed je bil uporabljen statistični označevalnik Obeliks[10], ki je bil prav tako izdelan med projektom Sporazumevanje v slovenskem jeziku.

4.4 Organizacija podatkov

Podatki bodo organizirani v dve podatkovni bazi. Ena bo uporabljena za korpus ccKres, druga za korpus ccGigafida. Struktura tabel v obeh podatkovnih bazah bo enaka, razlikovali se bosta le po velikosti. Z razdelitvijo korpusov v dve bazi bomo lažje testirali hitrosti različnih poizvedb, saj velikost podatkov na to zelo vpliva. Z večanjem števila podatkov se namreč večajo tudi indeksi, kar podaljša iskanje. Indeksi so obvezni že pri bazi ccKres, čeprav vsebuje le okoli 1 GB podatkov. Iskanje se namreč z uporabo indeksov v nekaterih primerih zmanjša iz 8 sekund na 1,20 sekunde.

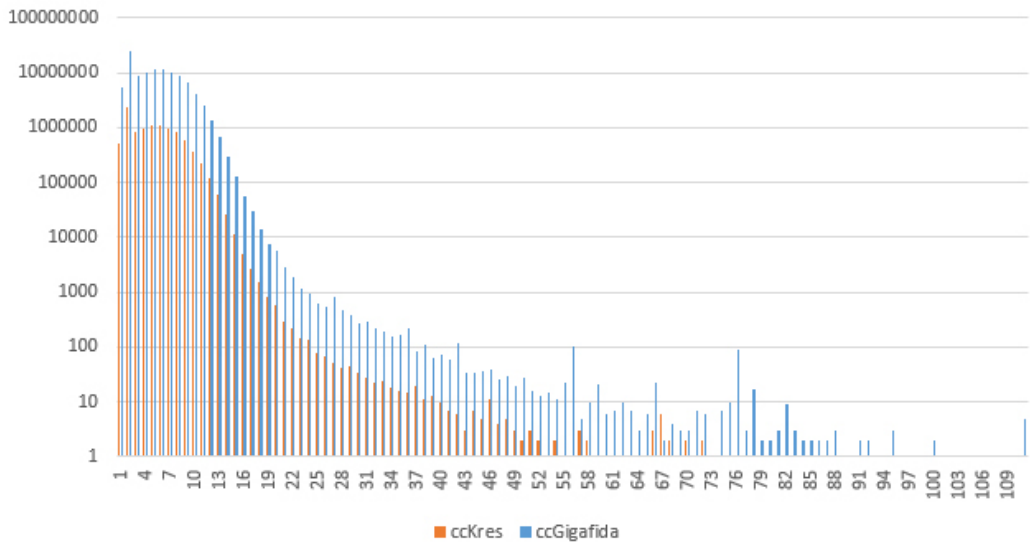
Že na nivoju podatkovne baze bomo omogočili podporo za slovenščino. To pomeni, da bomo uporabili UTF-8[42] nabor znakov, ki je sposoben prikaza celotnega nabora znakov. Tukaj so vključeni vsi jeziki, kot tudi razni posebni znaki (npr. puščice) in seveda slovenski šumniki. Podatkovni bazi bosta vsebovali po dve tabeli. Ena bo namenjena samim besedilom člankov in meta podatkom[1]. Druga tabela bo tabela vseh besed v posameznem članku. Vsaka beseda bo imela enoličen identifikator sestavljen iz dve polj. Prvo polje z imenom `aid`, bo vezan na enoličen identifikator članka, drugo polje z imenom `wid` pa bo pozicija besede v samem članku. Torej, če bo `wid` enak 1, pomeni, da je to prva beseda v članku. Za vsako besedo bomo shranili tudi njeno lemo. Poleg leme bomo shranili tudi besedno vrsto. Najbolj znane besedne vrste so samostalnik, glagol in pridevnik. Rezultati poskusov bi pokazali, da je najbolj pogosta sosednja beseda v slovenščini beseda *in*, česar pa v večini primerov ne želimo. Zato bomo dodali še en dodaten opis besede, in sicer stolpec, ki nam bo povedal ali je beseda nepomembna beseda. Tako bomo lahko kasneje pri iskanju sosednjih besed povedali, da želimo izpustiti

[1]Meta podatki so podatki o podatkih. Torej v primeru člankov so to dodatki podatki o člankih, kot so avtor, naslov članka, založnik in datum izdaje.

nepomembne besede. Ko bomo imeli vse besede shranjene v podatkovni bazi bomo izdelali še tekstovne indekse na tabeli s članki in navadne indekse na tabeli z besedami. Tabela s članki bo imela vsa polja indeksirana, saj želimo iskati tudi po meta podatkih. Tabela z besedami pa bo imela indeksirana samo polja kjer je shranjena beseda v obliki, ki se pojavi v članku in polje z njeno lemo.

Pri določitvi dolžine polj za besedo in lemo smo najprej preverili, kakšna je najdaljša beseda v korpusih. Za korpus ccKres se je izkazalo, da je najdaljša beseda dolga 155 znakov. To besedo predstavlja neka ogljikovodikova kemijska formula. Najverjetneje je program, ki je zbiral besedila za Gigafido, naletel na to formulo in jo interpretiral kot besedo. Zaradi zanimanja pa smo poiskali tudi ostale zelo dolge besede. Končni rezultat je pokazal, da avtorji korpusa niso ignorirali spletnih naslovov, razpršenih (hash) funkcij in komentarjev uporabnikov pod članki. Ena opcija bi bila, da tako dolge "besede"enostavno zavržemo. Za orientacijo bi lahko vzeli najdaljšo slovensko besedo. To je pridevnik dialéktičnomaterialístičen, ki ima vsega skupaj 26 črk in pomeni nanašajoč se na dialektični materializem (dialéktičnomaterialístičen pogled na življenje)[30]. Vendar bi s tem mogoče zavrgli tudi kako tujko, ki je mogoče celo daljša oz., če bi v bližnji prihodnosti pridobili v slovenski jezik novo najdaljšo besedo, bi bila tudi ta ignorirana. Ker pa nismo želeli zavreči nobene besede smo se odločili, da jih še vseeno shranimo in tako kot velikost polja, kjer bo shranjena beseda vzamemo dolžino 200, ki nam pokrije spodnji mejo 155 (dolžina najdaljše besede) in nam hkrati, da tudi nekaj rezervnega prostora.

Opisana organizacija podatkov v tem poglavju velja za vse tri SUPB-je, razliko podrobnostih bomo pokazali v naslednjem poglavju o implementaciji. Poseben poudarek si zasluži MongoDB, saj ima dinamično tipiziranje, kar pomeni, da sam prireja tipe podatkom, ko jih vnesemo v zbirko. Sam torej določi ali je vnešeni podatek število, niz, ipd. Poleg tega nam ni potrebno poskrbeti za dolžino polj, ker za to skrbi sam SUPB. Poleg tega bomo pri MongoDB uporabili nekoliko drugačno strukturo baze kot jo imata MariaDB



Slika 4.3: Grafični prikaz dolžine besed in njihovih ponovitev od 1 do 155 v korpusih ccKres in ccGigafida. Skala na y-osi je logaritmična.

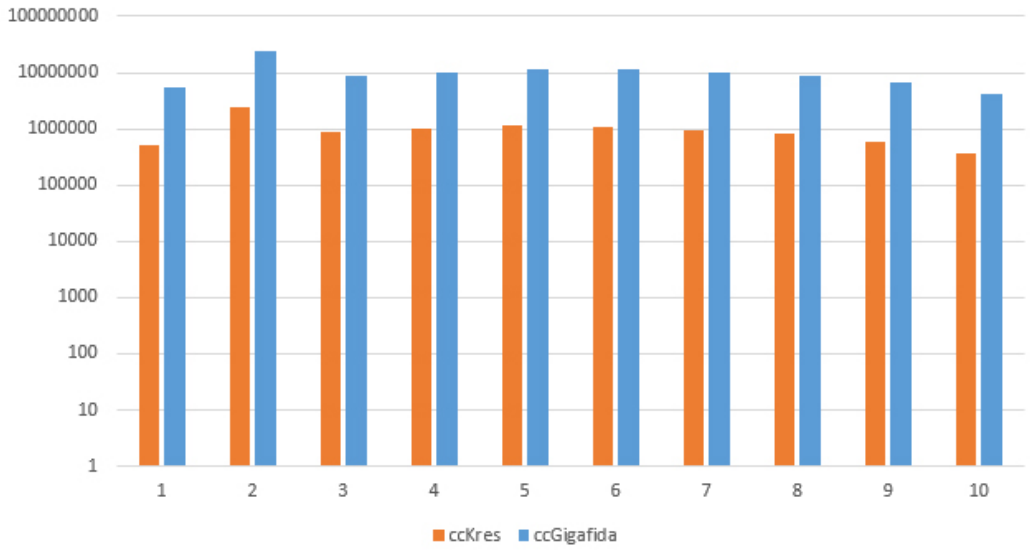
in PostgreSQL. MongoDB nam omogoča definiranje seznamov v zbirki. Tako bomo definirali seznam naslednjih in prejšnjih besed za vsako besedo posebej. Omejitev bo le dolžina seznamov, ki bo največ deset naslednjih in deset prejšnjih besed. S tem se bo tudi drastično pospešil čas poizvedb.

dolžina	ponovitve (ccKres)	ponovitve (ccGigafida)
1	513669	5540321
2	2349927	24184682
3	855727	8742191
4	976454	10167715
5	1117025	11864124
6	1086682	11482390
7	939870	10039283
8	820720	8907683
9	595215	6601309
10	376808	4217032
11	230977	2540289
12	119372	1333595
13	60475	684208
14	26711	290827
15	11815	131285
16	5158	58153
17	2709	30342
18	1507	13943
19	807	7696
20	565	5861
21	291	2862
22	227	1840
23	143	1206
24	134	923
25	77	645
26	70	531
27	50	805
28	41	483
29	44	372

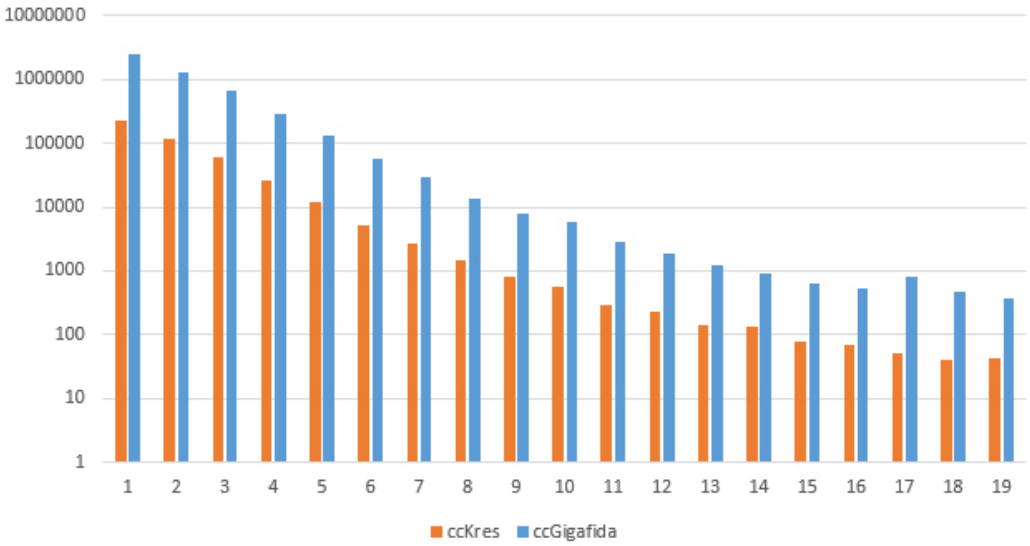
Tabela 4.1: Tabelarični prikaz dolžine besed (do 29) in njihovih ponovitev v korpusih ccKres in ccGigafida

Dolge besede
radioelektronsko
elektroenergetike
literarnozgodovinskih
hahahahahahahaaahahaha
http://www.shrani.si/f/
http://www.surveymonkey.com/s/
http://v1.nedstatbasic.net/stats?ABeGQg61hoiAm6/
YUDS8GZFATYAAACAAAAAAAAAAEQAYQB0AG EAAAAAA-AAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAA
Ðdghspace180Ðdgvspace180Ðdghorigin 1701Ðdgvorigin1984Ðdghshow1Ðdgvshow 1ÐjexpandÐviewkind4Ðviewscale117Ðpg brdrheadÐpgbrdrfoot
lxxxCxxxCH2xxxCH2xxxCH2xxxCH2xxxCxxxN xxxCH2xxxCH2xxxCH2xxxCH2xxxCH2xxxCH2 xxxNxxxCxxxCH2xxxCH2xxxCH2xxxCH2xxxC xxxNxxxCH2xxxCH2xxxCH2xxxCH2xxxCH2 xxxCH2xxxNxxx

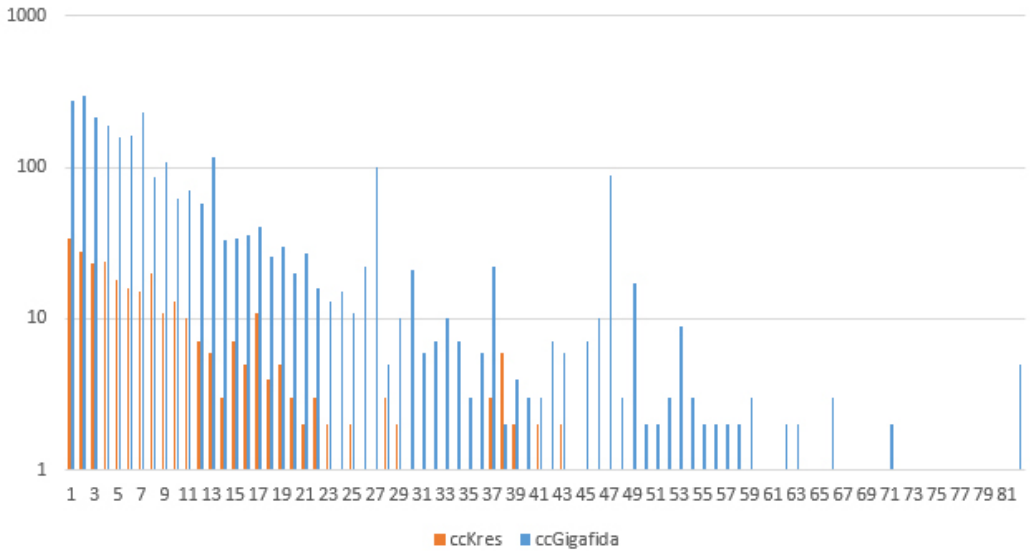
Tabela 4.2: Primeri zelo dolgih besed v korpusu ccKres



Slika 4.4: Grafični prikaz dolžine besed in njihovih ponovitev od 1 do 10 v korpusih ccKres in ccGigafida



Slika 4.5: Grafični prikaz dolžine besed in njihovih ponovitev od 11 do 29 v korpusih ccKres in ccGigafida



Slika 4.6: Grafični prikaz dolžine besed in njihovih ponovitev od 30 do 155 v korpusih ccKres in ccGigafida

Poglavje 5

Implementacija sheme organizacije podatkov

V tem poglavju bomo pokazali kako se opis iz prejšnjega poglavja izvede za posamezni SUPB. MariaDB in PostgreSQL bosta imela zelo podobno sintakso, saj oba spadata med relacijske SUPB-je. Zaradi tega se bodo ukazi le malenkostno razlikovali. MongoDB pa bo sam poskrbel za kodiranje znakov, dolžino polj in tipe polj. Narejen je bil namreč na principu, da je razvijalcu aplikacije potrebno čim manj razmišljati o predstavitvi podatkov v podatkovni bazi.

5.1 Implementacija v MariaDB

Podatkovni bazi bomo kreirali z naslednjima ukazoma:

```
CREATE DATABASE cckres
CHARACTER SET = 'utf8'
COLLATE = 'utf8_slovenian_ci';

CREATE DATABASE ccgigafida
CHARACTER SET = 'utf8'
COLLATE = 'utf8_slovenian_ci';
```

Pri tem ni dovolj samo ukaz CREATE DATABASE, potrebno je namreč dodati podporo za UTF-8 nabor znakov, kjer so tudi slovenski šumniki. Z ukazom CHARACTER SET povemo, kateri nabor znakov oz. črk želimo uporabljati. Z ukazom COLLATE pa povemo v katerem jeziku naj delujejo sortirne funkcije. Tukaj seveda izberemo sloveščino. Sedaj je omogočena podpora slovenskemu jeziku na samem nivoju podatkovne baze. Če smo v dvomih ali je ukaz deloval oz. ali so se spremembe uveljavile lahko to preverimo z ukazom `SHOW CREATE DATABASE`. Sledi ustvarjanje novega uporabnika

```
CREATE USER demo@localhost
IDENTIFIED BY '123';

GRANT ALL ON cckres.*
TO 'demo'@'localhost'
IDENTIFIED BY '123';

GRANT ALL ON ccgigafida.*
TO 'demo'@'localhost'
IDENTIFIED BY '123';
```

Ime novega uporabnika je *demo*, njegovo geslo je *123* in ima dodeljene vse pravice nad podatkovnima bazama cckres in ccgigafida. Kot smo že prej omenili bo sama struktura podatkovnih baz enaka, razliko bo le v velikosti. Zato naslednji ukazi veljajo tako za cckres, kot za ccgigafida. Podatkovni baza bosta vsebovali le dve tabeli. Prva bo namenjena shranjevanju celotnih člankov in informacij o avtorjih, založniku in letu izida.

```
CREATE TABLE articles (
    id          INT UNSIGNED PRIMARY KEY,
    title       VARCHAR(200),
    author      VARCHAR(100),
    date        DATE,
    publisher   VARCHAR(100),
    text        TEXT
);
```

Polje *id* predstavlja enoličen identifikator vsakega članka. Polja *title*, *author*, *date* in *publisher* so namenjena meta podatkom o naslovu članka, avtorju, datumu izida in založniku. Vsa ta polja razen polja za datum izida imajo kot tip *VARCHAR*[45], kar pomeni, da se dolžina polja prilagodi glede na dolžino niza, ki ga vnesemo v polje. Hkrati pa je specificirana maksimalna dolžina niza, ki ga lahko vnesemo. Polje za datum izida pa uporablja tip *DATE*[44], kjer je datum shranjen v obliki leto-mesec-dan, primer 2016-01-20. Zadnje polje je namenjeno celotnemu besedilu članka in ima tip *MEDIUMTEXT*[46] v katerega lahko shranimo izredno dolga besedila. Omejitev dolžine polja je 16,777,215 bajtov.

```
CREATE TABLE words (
    aid         INT UNSIGNED,
    wid         INT UNSIGNED,
    msd         VARCHAR(50),
    word        VARCHAR(200),
    lemma       VARCHAR(200),
    stop        BOOL,
    PRIMARY KEY (aid, wid),
    INDEX fk_Words_Articles_idx (aid ASC),
    CONSTRAINT fk_Words_Articles
    FOREIGN KEY (aid)
    REFERENCES Articles(id)
);
```

Pri tabeli besed je enolični identifikator sestavljen iz dveh polj, in sicer *aid* in *wid*. Polje aid nam pove, kateremu članku pripada beseda in je enak polju id v tabeli člankov (articles). S poljem wid pa določimo vrstni red besede v članku. Polje *msd* nam pove v katero besedno vrsto[47] uvrščamo besedo, ki je shranjena v polju *word*. Polje *lemma* pa vsebuje njeno lemo. Zadnje polje *stop* označuje ali je beseda obravnavana kot nepomembna med iskanjem, če to želimo.

Po vnosu podatkov v tabeli je potrebno zgraditi še indekse. Tega nismo storili že prej, ker indeksi upočasnijo vnos podatkov v tabelo. Najprej zgradimo tekstovne indekse na vseh poljih v tabeli člankov, z izjemo polja z datumom. S tem omogočimo polnotekstovno iskanje (ang. full-text search) ne samo po besedilu članka, temveč tudi po meta podatkih.

```
CREATE FULLTEXT INDEX title_ftidx
ON articles(title);

CREATE FULLTEXT INDEX author_ftidx
ON articles(author);

CREATE FULLTEXT INDEX publisher_ftidx
ON articles(publisher);

CREATE FULLTEXT INDEX body_ftidx
ON articles(text);
```

MariaDB nam omogoča tudi izdelavo enega polnotekstovnega indeksa, ki zajame več polj naekrat. Ker pa je potem potrebno pri vsaki poizvedbi navesti vsa polja, ki jih zajema ta indeks smo se odločili, da raje naredilo štiri polnotekstovne indekse. Tabela besed pa zahteva izdelavo navadnih indeksov, saj imamo v posameznih poljih samo po eno vrednost oz. besedo.

```
CREATE INDEX word_idx
ON words(word);

CREATE INDEX lemma_idx
ON words(lemma);
```

5.2 Implementacija v PostgreSQL

Organizacija podatkov v PostgreSQL bo enaka kot v MariaDB. Ponovno bomo zgradili dve različni podatkovni bazi, in sicer *cckres* in *ccgigafida*. Ker pa gre tokrat za PostgreSQL bo seveda sintaksa malo drugačna. Najprej moramo narediti podatkovni bazi:

```
CREATE DATABASE cckres
ENCODING 'utf8';

CREATE DATABASE ccgigafida
ENCODING 'utf8';
```

Poleg imena podatkovne baze navedemo tudi uporabo UTF-8 nabora znakov. Izdelava novega uporabnika pri PostgreSQL je nekoliko drugačna. Da lahko dodajamo in brišemo uporabnike, moramo najprej popraviti konfiguracijske datoteke[48]. Te se nakajajo na lokaciji */etc/postgresql/current/main/pg_hba.conf*. V tej datoteki dodamo naslednjo vrstico

```
# TYPE  DATABASE    USER    IP-ADDRESS    IP-MASK           METHOD
host    all         all     10.0.0.0      255.255.255.0     md5
```

V konfiguracijski datoteki */etc/postgresql/current/main/postgresql.conf* navedemo, da naj PostgreSQL spremlja promet na vseh povezavah

```
listen_addresses = '*'
```

Z naslednjimi ukazi naredimo novega uporabnika z imenom *demo*, podatkovni bazi cckres in gigafida, hkrati pa dodelimo tudi lastništvo nad bazama uporabniku

```
sudo -u postgres createuser demo
sudo -u postgres createdb -O demo cckres
sudo -u postgres createdb -O demo ccgigafida
```

V podatkovno bazo se prijavimo tako, da povemo ime uporabnika in ime podatkovne baze na katero se želimo prijaviti

```
psql -U demo -d cckres
```

V našem primeru je to uporabnik *demo* in podatkovna baza *cckres*. Če želimo zamenjati podatkovno bazo to storimo z ukazom
`c ime_baze`, npr.

```
\c ccgigafida
```

Sledi izdelava samih tabel. Najprej tabela člankov (ang. articles), kjer je sintaksa izdelave zelo podobna sintaksi za MariaDB.

```
CREATE TABLE articles (
    id          integer PRIMARY KEY,
    title       varchar(200),
    author      varchar(100),
    date        date,
    publisher   varchar(100),
    text        text
);
```

Tudi tukaj kreiramo enolični identifikator članka *id*, polja za meta podatke (*title*, *author*, *date*, *publisher*) in polje *text* za besedilo članka. Vsa polja za meta podatke razen polja za datum imajo tudi tukaj tip niz, katerega dolžina se prilagaja dolžini vhodnega niza. Polje za datum ima enak zapis kot pri MariaDB, polje text pa nam ponovno omogoča shranjevanje zelo dolgih besedil. Za razliko od MariaDB, polje text v PostgreSQL nima omejitve dolžine. Sledi tabela za besede (ang. words)

```
CREATE TABLE words (
    aid         integer NOT NULL,
    wid         integer NOT NULL,
    msd         varchar(50)  NOT NULL,
    word        varchar(200) NOT NULL,
    lemma       varchar(200) NOT NULL,
    stop        boolean NOT NULL,
    PRIMARY KEY (aid, wid)
);
```

Ponovno zelo podobno, vendar je nekaj razlik. Najprej moramo omeniti, da PostgreSQL ne podpira tipa `UNSIGNED INT`[49]. Sledijo tekstovni indeksi, ki pa jih je prav tako kot pri MariaDB priporočljivo ustvariti šele po uvozu podatkov

```
CREATE INDEX title_idx
ON articles
USING gin(to_tsvector('slovenian', title));

CREATE INDEX author_idx
ON articles
USING gin(to_tsvector('slovenian', author));

CREATE INDEX publisher_idx
ON articles
USING gin(to_tsvector('slovenian', publisher));

CREATE INDEX text_idx
ON articles
USING gin(to_tsvector('slovenian', text));
```

Razlika v sintaksi je očitna. Z ukazom `gin()` ustvarimo GIN indeks. GIN pomeni splošni inverzni indeks (ang. Generalized Inverted Index). Primer

njihove uporabe so npr. dokumenti, kjer želimo iskati specifične besede, kar je točno to, kar želimo v našem primeru početi. Indeks vrednosti shranjuje kot par sestavljen iz ključa in seznamov id-jev vrstic, kjer se ključ pojavi.[17] Naslednji ukaz je `to_tsvector()`. Omogoča nam shranjevanje dokumentov, po katerih lahko iščemo in računamo pomembnost ter omogoča uporabo tekstovnih indeksov. Prvi argument, ki ga sprejme je konfiguracijska datoteka za posamezni jezik. V našem primeru je to slovenščina. Podrobneje o konfiguracijah bomo izvedeli v poglavju o tekstovnih indeksih v PostgreSQL na strani 48. Drugi argument pa je stolpec, nad katerim naredimo tekstovni indeks. Prav tako kot pri MariaDB smo tukaj izdelali štiri indekse, saj nam omogočajo fleksibilno iskanje tudi po posameznih poljih. Drugače bi morali pri vsaki poizvedbi vnesti vsa polja, ki jih vsebuje indeks, da bi deloval pravilno. Na koncu naredimo še navadne indekse po besedi in lemi.

```
CREATE INDEX word_idx
ON words(word).

CREATE INDEX lemma_idx
ON words(lemma).
```

5.3 Implementacija v MongoDB

Drastična razlika v sintaksi se pokaže šele pri uporabi MongoDB. Tukaj člankov in besed ne bomo shranjevali v tabele, temveč v tako imenovane zbirke (ang. collection). Tudi zbirk nam ni potrebno izdelati vnaprej, temveč samo začenemo vstavljati podatke in zbirka se izdela sama. Novo bazo izdelamo kar z ukazom `use ime_baze` na sledeči način

```
use cckres
```

tako kreiramo podatkovno bazo za cckres in ccgigafida

```
use ccgigafida
```

Kot že omenjeno se bosta zbirki za članke in besede izdelali sami takoj, ko začnemo vnašati podatke

```
db.articles.insert_one(
  {
    'title': title,
    'author': author,
    'date': date,
    'publisher': publisher,
    'text': text
  }
)
```

S tem ukazom vstavimo en zapis v zbirko imenovano `articles`. Kot že omenjeno, zbirke nismo izdelali vnaprej, temveč se je ustvarila sama takoj, ko smo vanjo shranili prvi zapis [50]. Tukaj nam ni potrebno skrbeti za primerno dolžino polj, pravilen tip polj in enolične ključe. Za vse to poskrbi MongoDB sam. Vse kar moramo narediti sami so tekstovni indeksi po izdelavi samih zbirk, imena polj v zbirki in njihove vrednosti. Struktura zbirke člankov je skoraj enaka kot pri MariaDB in PostgreSQL. Nato izdelamo še zbirko za besede

```
self.db.words.insert_one(
  {
    'aid': aid,
    'wid': wid,
    'msd': msd,
    'word': text,
    'lemma': lemma,
    'stop': stop,
    'next': next,
    'prev': prev
  }
```

```
)
```

Vse je enako, razen dveh zadnjih polj. Polji *next* in *prev* sta seznama deset naslednjih in deset prejšnjih besed. Zajete so vse besede, zato filtriranje po nepomembnih besedah ni možno. Po izdelavi obeh zbirk so potrebni samo še tekstovni in navadni indeksi. Za zbirko člankov bomo uporabili tako imenovani *wildcard* index, ki naredi tekstovne indekse na vseh poljih, ki vsebujejo nize

```
db.articles.createIndex({ "$**": "text" })
```

Za zbirko besed pa kreiramo dva navadna indeksa na poljih kjer sta shranjena beseda in lema

```
db.words.createIndex({word: 1})
db.words.createIndex({lemma: 1})
```

5.4 Implementacija poizvedb za iskanje kolokacije besed

Za namene testiranja sta bili uporabljeni naslednji poizvedbi. Prva poizvedba je bila uporabljena tako v MariaDB, kot tudi v PostgreSQL. Oba namreč uporabljata SQL standard za pisanje poizvedb. Poizvedba za MongoDB je nekoliko drugačna, ker upošteva spremenjeno strukturo, poleg tega pa MongoDB uporablja lastno sintakso oz. JavaScript z lastnimi funkcijami.

```
SELECT w1.word AS beseda, COUNT(*) AS ponovitve
FROM words w1
INNER JOIN (
  SELECT aid, wid
  FROM words
  WHERE tip = 'iskana besede'
) AS w2
```

```
ON w1.aid = w2.aid
AND (w1.wid BETWEEN w2.wid - razdalja
     AND w2.wid + razdalja)
AND w1.wid != w2.wid
GROUP BY w1.word
ORDER BY ponovitve DESC
LIMIT 20;
```

Najprej je potrebno omeniti, da sta v poizvedbi dve spremenljivki. Prva je *tip*, ki jo najdemo v notranji poizvedbi, s katero določimo ali bomo iskali po *besedi* ali *lemi* (v bazi *word* in *lemma*). V niz *iskalna beseda* vnesemo besedo po kateri bi radi iskali. Druga spremenljivka je *razdalja*. Z njo povemo na kakšni razdalji od iskane besede naj se iščejo vse ostale sosednje besede. Pri testiranju je bila najmanjša uporabljena vrednost ena, največja pa deset. Tako se sklada s poizvedbo in strukturo pri MongoDB. Poizvedba deluje tako, da najprej poišče enolični ključ sestavljen iz zaporedne številke članka *aid* in zaporedne številke besede v tem članku *wid*, za vsako besedo v tabeli *words* (notranja poizvedba). Nato ta seznam združimo z obstoječo tabelo *words* z uporabo INNER JOIN in primerjamo ključe tako, da je poiščemo vse besede kjer je njihov wid manjši ali večji za največ razdaljo (ukaz BETWEEN), ki smo jo izbrali (vključno z vrednostjo razdalje). Z ukazom CAST preprečimo pojavitev napake, če je slučajno rezultat wid - razdalja negativen. Poleg tega pa izključimo iskano besedo (w1.wid != w2.wid). Nato dobljeni rezultat sortiramo po besedi z GROUP BY, kjer je beseda z največ ponovitvami (ukaz COUNT) na vrhu (ukaz ORDER BY in DESC). Na koncu pa omejimo število prikazanih vrstic na dvajset z LIMIT.

Sledi poizvedba uporabljena v MongoDB. Ker je bil MongoDB s podobno poizvedbo in enako strukturo dokumentov prepočasen je bila potrebna sprememba. MongoDB nam omogoča definiranje seznamov v zbirki. Tako bomo definirali seznam naslednjih in prejšnjih besed za vsako besedo posebej. Omejitev bo le dolžina seznamov, ki bo največ deset naslednjih in deset prejšnjih besed. S tem se bo tudi drastično pospešil čas poizvedb.

```
db.words.aggregate([
  { '$match': { 'tip': 'iskana beseda' } },
  { '$project': { 'words': { '$concatArrays': [
    { '$slice': ['$prev', razdalja] },
    { '$slice': ['$next', razdalja] }
  ] } } },
  { '$unwind': '$words' },
  { '$group': { '_id': '$words', 'ponov': { '$sum': 1 } } },
  { '$sort': { 'ponov': -1 } },
  { '$limit': 20 }
])
```

Čeprav je sintaksa povsem drugačna, poizvedba deluje na podoben način. Tudi tukaj imamo enaki dve spremenljivki *tip* in *razdalja*, ki imata tudi enaki funkciji. Uporabljen je MongoDB-jev *aggregation framework*[59], ki deluje na konceptu cevovoda za procesiranje podatkov. Dokumenti potujejo skozi različne faze cevovoda, ki podatke spreminjajo in združujejo. Prva faza v poizvedbi imenovana *$match* je iskanje podane besede. Nato s pomočjo *razdalje* določimo koliko sosednjih besed nas zanima tako, da po želji vzamemo manj besed iz seznama prejšnjih *$prev* ali naslednjih *$next* besed. Tudi tukaj je najmanjša možna vrednost spremenljivke *razdalja* ena in največja deset (celotna velikost seznamov). Rezultat nato združimo v nov seznam s pomočjo *$concatArrays* in ga s *$project* shranimo pod imenom *words*. Nato ta seznam odpremo z *$unwind* tako, da dobimo posamezne dokumente za vsako besedo v seznamu. Te besede združimo z *$group* po številu ponovitev s pomočjo *$sum*. Na koncu sortiramo s *$sort* tako, da so besede z največ ponovitvami na vrhu (parameter -1) in vrnemo dvajset rezultatov.

Poglavje 6

Tekstovni indeksi

Poglavje je namenjeno temu, da spoznamo kaj so tekstovni indeksi, čemu služijo in kako jih uporabljamo. Zakaj sploh potrebujemo tekstovne indekse, če že imamo operatorje kot so `LIKE` in z njimi lahko čisto enostavno iščemo določene besede v tekstu. To je že res, vendar kaj kmalu naletimo na omejitve takih operatorjev. Recimo, da imamo v tabeli *articles*, ki smo jo definirali v drugem poglavju že shranjenih na tisoče člankov. Trenutno še nimamo tekstovnih indeksov, saj smo še vedno prepričani, da jih ne potrebujemo. Sedaj se odločimo, da bi radi našli niz "Iščemo najlepši dragulj.". Ker vemo, da niz vsebuje besedo *najlepši* se odločimo, da jo poiščemo niz z uporabo operatorja `LIKE`. Poizvedbi bi z uporabo MySQL ali MariaDB izgledala nekako takole:

```
SELECT body
FROM articles
WHERE body
LIKE '%najlepši%';
```

Ta poizvedba bi seveda delovala, saj je beseda najlepši obdana s procenti s katerimi povemo, da iskani niz vsebuje besedo dan, hkrati pa so lahko pred in po njen tudi ostale črke oz. besede. V primeru, da nismo prepričani ali je iskana beseda *najlepši* ali *najlepše* bi lahko uporabili operator `OR`

```
SELECT body
```

```
FROM articles
WHERE body
LIKE '%najlepši%'
OR body LIKE '%najlepše%';
```

Bolj elegantna rešitev bi bila uporaba regularnih izrazov, vendar v tem primeru naredimo našo poizvedbo veliko počasnejšo. V tem primeru mora biti regularni izraz v celoti preverjen na vsakem znaku v nizu. Pri tako veliki količini podatkov si tega žal ne moremo privoščiti. Ravno ta problem rešijo tekstovni indeksi. Pri posameznih besedah upoštevajo lekseme in s tem lahko najdejo skoraj vsako besedo tudi, če smo jo narobe vpisali. Poleg tega nam vedno vrnejo še ostale podobne rezultate in vse se skupaj razvrsti tako, da so najbolj verjetni rezultati na vrhu.

Ker ima vsak uporabljeni podatkovni sistem svojo sintakso in svoj način delovanj, bomo ločeno obravnavali uporabo za vsak sistem posebej. Pri uporabi tekstovnih indeksov se besedilo razdeli na žetone, torej se razbije na posamezne besede. Osnovni jezik, ki ga uporablja razčlenjevalnik je angleščina. Ker slovenščina prav tako kot angleščina za ločitev besed uporablja presledek, je osnovni razčlenjevalnik dovolj dober, da zna ločiti besede med seboj. Za bolj napredne funkcije pa je potrebna dodatna podpora določenemu jeziku znotraj samega sistema.

Ena izmed takih funkcij je tudi podpora za določitev nepomembnih besed (stopwords v angleščini). To je seznam besed, ki jih razčlenjevalnik (ang. parser) ignorira in se ne upoštevajo niti pri iskanju rezultatov. Med take besede spadajo predvsem vezniki (in, ker, če, ...) in predlogi (pod, nad, v, ...). V kolikor nam je pomembna hitrost, nam tak seznam lahko zelo pomaga, saj se večina takih besed pojavi tudi največkrat v besedilih. V slovenščini je namreč najbolj pogosta beseda *in*.

6.1 Tekstovni indeksi v MariaDB

Tekstovni indeksi v MariaDB so tipa `FULLTEXT`. Z njim pridobimo dodatne možnosti pri iskanju delov teksta v posameznem polju tabele. Na začetku so delovali le na tabelah *MyISAM* in *Aria*[1]. Z različico 10.0.15 pa lahko tekstovne indekse uporabljamo tudi s tabelami *InnoDB*. Polja na katerih lahko ustvarimo tekstovne indekse so omejena na `CHAR`, `VARCHAR` in `TEXT`. V dokumentaciji je omenjeno tudi, da naj v primeru, ko delamo z velikimi količinami podatkov, zgradimo tekstovne indekse šele po vnosu podatkov[15].

Kako pa je s podporo uporabe tekstovnih indeksov za ostale jezike poleg angleščine? Nas seveda najbolj zanima slovenščina. Tukaj lahko kaj hitro opazimo, da kot je bilo omenjeno že na začetku poglavja, slovenščina z razdelitvijo na žetone nima problemov. Besede so namreč ločene s presledki tako kot pri angleščini. Kaj pa, če bi želeli uporabiti tekstovne indekse na kakšnem izmed azijskih jezikov kot sta npr. kitajščina in japonščina. Oba jezika namreč ne ločita posameznih besed s presledki. V uradni dokumentaciji MariaDB[15] nikjer ne moremo zaslediti podatka, da bi obstajala kakršna koli podpora. V komentarjih na isti strani je točno ta problem omenil neznani uporabnik. Odgovor drugega uporabnika je, da MariaDB ne nudi podpore za azijske jezike. Lahko pa zasledimo, da MySQL to podporo ima[31]. Od različice 5.7.6 najprej je poleg osnovnega razčlenjevalnika besedila vgrajen tudi n-gram razčlenjevalnik, ki podpira kitajščino, japonščino in korejščino. Poleg tega pa je vgrajen tudi MeCab parser za japonščino, ki podpira tudi InnoDB tabele. S slovenščino torej ne bomo imeli problemov, v primeru, da pa nekdo želi delati z neevropskimi jeziki, naj raje za SUPB vzame MySQL.

Sintaksa, ki jo uporabljamo s tekstovnimi indeksi se imenuje MATCH() ... AGAINST() sintaksa. V MATCH() delu navedemo z vejico ločena imena stolpcev, AGAINST() pa prejme niz, po katerem iščemo, in modifikator, ki ni obvezen. Sintaksa izgleda takole

[1] Aria tabele pohitrijo poizvedbe, ki uporabljajo GROUP BY in DISTINCT, saj imajo boljše predpomnenje kot MyISAM

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

MariaDB pozna tri načine uporabe tekstovnih indeksov. Prvi način je `IN NATURAL LANGUAGE MODE`, kjer MariaDB razvrsti zadetke glede na relevantnost. Drugi način je `IN BOOLEAN MODE`, ki sintakso nadgradi z možnostjo uporabe logičnih operatorjem. Torej lahko povemo katerih besed ne želimo med rezultati. Tretji način pa je `WITH QUERY EXPANSION`, ki dvakrat naredi `IN NATURAL LANGUAGE MODE` in na koncu vrne najbolj primerne rezultate glede na obe poizvedbi. V vseh treh načinih so rezultati razvrščeni po relevantnosti in med samim iskanjem je upoštevan seznam nepomembnih besed.

Tekstovni indeksi v MariaDB že v osnovi ne upoštevajo nekaterih besed. Med te spadajo:

- besede krajše od 4 znakov
- besede daljše od 84 znakov
- besede na seznamu nepomembnih besed
- besede, ki se pojavijo v več kot polovici vrsticah

Dolžini besed, ki jih ne upoštevamo lahko nastavimo s sistemskima spremenljivkama *ft_min_word_length* (za InnoDB *innodb_ft_min_token_size*) in *ft_max_word_length* (za InnoDB *innodb_ft_max_token_size*). Ko definiramo tekstovni indeks, MariaDB zgradi B-drevo, ki ima dva nivoja. V prvem nivoju se nahajajo ključne besede, ki se zgradijo s razdelitivijo besedila na žetone. V drugem nivoju pa so asociativni kazalci na dokumente[29] (z njihovo pomočjo SUPB ve, kje v katerem dokumentu je shranjena beseda), ki vsebujejo te ključne besede. Prva popustljivost tekstovnih indeksov se vidi že tukaj. Problem je namreč v tem, da se pozicije ključnih besed ne shranjujejo. To je glavni razlog, da bomo kasneje naredili lastno rešitev za iskanje sosednjih besed.

Za prikaz uporabe tekstovnih indeksov bomo naprej zgradili manjšo tabelo citatov in vanjo vnesli nekaj citatov. Če eksplicitno ne povemo, kateri shranjevalni mehanizem bi uporabljali, MariaDB samodejno izbere MyISAM.

```
CREATE TABLE citati (citat VARCHAR(150) NOT NULL);

INSERT INTO citati
VALUES ("Skrivnost uspeha vsakega uspešnega človeka je,
         da je razvil drugačne navade od tistih, ki so neuspešni.");

INSERT INTO citati
VALUES ("Nihče ne more uspeti brez pričakovanj!");

INSERT INTO citati
VALUES ("Da bi v življenju uspeli, se moramo delati neumne,
         a biti modri.");

INSERT INTO citati
VALUES ("Samo eno pravilo ti bo vlilo pogum, in sicer pravilo,
         da nobeno zlo ne traja večno niti zelo dolgo.");

INSERT INTO citati
VALUES ("Edini pogum, ki kaj šteje, je tisti,
         ki te popelje od danes do jutri");

INSERT INTO citati
VALUES ("Kje je šola za pogum? Šola za pogum je,
         da narediš tisto, v kar verjameš!");

INSERT INTO citati
VALUES ("Življenje se meri po delih, ne po dnevih.");

INSERT INTO citati
VALUES ("Življenjski cilj vsakega posameznika je vedno isti:
         napredovanje v dobrem.");
```

Sedaj, ko imamo tabelo ustvarjeno in v njej osem citatov potrebujemo le še tekstovni indeks

```
CREATE FULLTEXT INDEX ftidx ON citati(citat);
```

6.1.1 Polnotekstovno iskanje v načinu naravnega jezika

MariaDB podpira tri načine iskanja z uporabo tekstovnih indeksov. Prvi način se imenuje **NATURAL LANGUAGE MODE**. V tem načinu ni nobenih posebnih operatorjev, saj iščemo s pomočjo ključnih besed, ki jih ločimo z vejico. Rezultati, ki jih dobimo so urejeni v padajočem vrstnem redu glede na relevantnost

```
> SELECT citat,
> MATCH(citat) AGAINST("pogum šola") AS pomembnost
> FROM citati
> WHERE MATCH(citat) AGAINST("pogum šola");
```

citat	pomembnost
Kje je šola za pogum? Šola za pogum je, da narediš tisto, v kar verjameš!	1.9940418004989624
Samo eno pravilo ti bo vlilo pogum, in sicer pravilo, da nobeno zlo ne traja večno niti zelo dolgo.	0.18144935369491577
Edini pogum, ki kaj šteje, je tisti, ki te popelje od danes do jutri	0.18144935369491577

Tabela 6.1: Rezultat poizvedbe polnotekstovnega iskanja (NATURAL LANGUAGE MODE) v MariaDB

Rezultat poizvedbe so vse tri vrstice, ki vsebujejo besedo *pogum*. Pomembnost se izračuna tako, da se deli število najdenih ključnih besed s frekvenco pojavitve teh besed v samem dokumentu. Mogoče je tudi iskanje po več stolpcih naenkrat. Seveda, morajo imeti vsi ti stolpci definirane tekstovne indekse. Primer bi bilo iskanje po citatu in avtorju citata. Moramo pa biti pozorni, da pri uporabi tekstovnih indeksov ne uporabljamo tudi ORDER BY. Ta ukaz namreč povzroči, da se dokumenti na koncu ne sortirajo po pomembnosti, vendar po sortiranju dokumentov (ang. filesort). Sintaksa pri tekstovnih indeksih ima še eno posebnost. Če dvakrat napišemo MATCH AGAINST, nič ne vpliva na hitrost. MariaDB sam opazi, da je stavek ponovljen in ga enostavno ignorira.

Enako rezultat lahko dosežemo tudi z `LIKE` poizvedbo, vendar dokumenti niso sortirani na noben način. Poleg tega poizvedba zahteva veliko več previdnosti, saj moramo uporabiti tudi `OR` operator in paziti, da besedi zapremo v `%`.

```
> SELECT citat
> FROM citati
> WHERE citat LIKE '%pogum%'
> OR citat LIKE '%šola%';
```

Z uporabo regularnih izrazov [51] tudi lahko naredimo poizvedbo, ki vrne enake rezultate. Vendar tudi tukaj ni nikakršnega sortiranja. Poleg tega od nas zahteva dodatno znanje regularnih izrazov.

```
> SELECT citat
> FROM citati
> WHERE citat REGEXP '(šola)|(pogum)';
```

6.1.2 Polnotekstovno iskanje z uporabo logičnih izrazov

Drugi način je **BOOLEAN MODE** in nam omogoča uporabo posebnih operatorjev. Rezultati, ki jih dobimo s tem načinom niso urejeni po pomembnosti.

Sedaj pa pokažimo še kako se uporablja

Operator	Opis
+	Beseda je se mora pojaviti v vseh vrnjenih vrsticah.
-	Besede se ne sme pojaviti v nobeni vrnjeni vrstici.
<	Beseda, ki sledi ima manjšo pomembnost kot ostale besede.
>	Besede, ki sledi ima večjo pomembnost kot ostale besede.
()	Uporablja se za grupiranje podizrazov.
~	Beseda, ki sledi negativno pripomore k pomembnosti vrstice.
*	"Wildcard", ki označuje 0 ali več črk. Pojavi se lahko samo na koncu besede.
"	Karkoli zapremo v narekovaje označuje celoto. Uporabno za fraze.

Tabela 6.2: Tabela vseh podprtih operatorjev v načinu BOOLEAN MODE

```
> SELECT citat,
> MATCH(citat) AGAINST("-edini pogum" IN BOOLEAN MODE) AS pomembnost
> FROM citati
> WHERE MATCH(citat) AGAINST("-edini pogum" IN BOOLEAN MODE);
```

V tem primeru smo dobili vse rezultate, ki imajo v nizu besedo pogum in hkrati nimajo tudi besede edini. Iz tabele je razvidno, da lahko iščemo tudi po celotnih frazah z uporabo narekovajev. Vendar je tukaj potrebno omeniti to, da je takšno iskanje izredno počasno. Z uporabo operatorja `LIKE` bi za enak rezultat napisali poizvedbo

```
> SELECT citat
```

citat	pomembnost
Življenjski cilj vsakega posameznika je vedno isti: napredovanje v dobrem.	1.997020959854126
Da bi v življenju uspeli, se moramo delati neumne, a biti modri.	0.18144935369491577
Življenje se meri po delih, ne po dnevih.	0.18144935369491577

Tabela 6.3: Rezultat poizvedbe polnotekstovnega iskanja (BOOLEAN MODE) v MariaDB z uporabo operatorja >

```
> FROM citati
> WHERE citat LIKE '%pogum%'
> AND citat NOT LIKE '%edini%';
```

Z uporabo regularnih izrazov taka poizvedba zelo hitro postane neberljiva.

```
> SELECT citat
> FROM citati
> WHERE citat REGEXP '^(?!.*Edini).*pogum.*$';
```

Naslednje poizvedbe pa z uporabo `LIKE` operatorja ali regularnih izrazov ne moremo narediti. Lahko iščemo besedo *življenj**, vendar ne moremo pa uporabiti operatorja `>` za sortiranje.

```
> SELECT citat,
> MATCH(citat) AGAINST("življenj* >cilj" IN BOOLEAN MODE) AS pomembnost
> FROM citati
> WHERE MATCH(citat) AGAINST("življenj* >cilj" IN BOOLEAN MODE);
```

Tukaj pa so rezultati vsi nizi, ki vsebujejo besedo s korenom *življenj*. Na vrhu pa je rezultat, ki vsebuje tudi besedo *cilj*.

citat	pomembnost
Kje je šola za pogum? Šola za pogum je, da narediš tisto, v kar verjameš!	0.36289870738983154
Samo eno pravilo ti bo vlilo pogum, in sicer pravilo, da nobeno zlo ne traja večno niti zelo dolgo.	0.18144935369491577
Edini pogum, ki kaj šteje, je tisti, ki te popelje od danes do jutri	0.18144935369491577

Tabela 6.4: Rezultat poizvedbe polnotekstovnega iskanja (NATURAL LANGUAGE MODE) v MariaDB

6.1.3 Polnotektovno iskanje v načinu dvojne poizvedbe

Tretji način pa je **WITH QUERY EXPANSION**. Ta način je modifikacija NATURAL LANGUAGE MODE načina. Deluje namreč tako, da se najprej izvede iskanje z NATURAL LANGUAGE MODE. Takoj po tem se izvede še eno iskanje, kjer se uporabijo vse besede, ki so bile vrnjene v vrsticah prvega rezultata. Poizvedba nato vrne vrstice, ki jih dobimo v drugem krogu iskanja.

```
> SELECT citat,
> MATCH(citat) AGAINST("pogum") AS pomembnost
> FROM citati WHERE MATCH(citat) AGAINST("pogum");
```

```
> SELECT citat,
> MATCH(citat) AGAINST("pogum" WITH QUERY EXPANSION) AS pomembnost
> FROM citati
> WHERE MATCH(citat) AGAINST("pogum" WITH QUERY EXPANSION);
```

citat	pomembnost
Samo eno pravilo ti bo vlilo pogum, in sicer pravilo, da nobeno zlo ne traja večno niti zelo dolgo.	10.783881187438965
Kje je šola za pogum? Šola za pogum je, da narediš tisto, v kar verjameš!	6.0718994140625
Edini pogum, ki kaj šteje, je tisti, ki te popelje od danes do jutri	5.890450477600098

Tabela 6.5: Rezultat poizvedbe polnotekstovnega iskanja (WITH QUERY EXPANSION) v MariaDB

Sedaj pa lahko vidimo, da sta se vrstna reda prve in druge vrstice zamenjala, hkrati pa so se spremenile tudi vrednosti v stolpcu *pomembnost*.

Poleg vseh prednosti pa imajo tekstovni indeksi tudi svoje slabosti. Nekaj smo jih že omenili med samim opisom. Poleg že naštetih, je slabost tudi to, da ne moremo specificirati lastnega algoritma za sortiranje po pomembnosti. Vedno se uporabi vgrajeni. Velika slabost iskanja s tekstovnimi indeksi pa je tudi njihova počasnost, če so le-ti preveliki, da bi se shranili v pomnilnik. Poleg tega pa zelo hitro pride do fragmentacije indeksa. Rešitev je uporaba ukaza OPTIMIZE TABLE, ali pa enostavno ponovno zgradimo tekstovne indekse. Kot smo že omenili na začetku, je veliko hitreje, če indekse zgradimo po tem, ko imamo podatke že v bazi. Če pa smo indekse že vgradili vnaprej, podatkov pa še nismo vnesli, jih lahko začasno onemogočimo z DISABLE KEYS in kasneje ponovno omogočimo z ENABLE KEYS[29]. Seveda po tem, ko smo podatke že uvozili v bazo. Ta pristop deluje samo na MyISAM tabelah. Za InnoDB tabele potrebujemo drugačen pristop, ena možnost je, da izkloplimo *autocommit*

```
SET autocommit=0;
... SQL INSERT stavki ...
COMMIT;
```

Če uporabljamo `UNIQUE` na tujih ključih lahko začasno izklopimo preverjanje enoličnosti

```
SET unique_checks=0;
... SQL INSERT stavki ...
SET unique_checks=1;
```

V enako lahko storimo v primeru uporabe `FOREIGN KEY`

```
SET foreign_key_checks=0;
... SQL INSERT stavki ...
SET foreign_key_checks=1;
```

Za konec pa omenimo še eno slabost tekstovnih indeksov[29]. Te indeksi imajo vedno prednost uporabe pred navadnimi indeksi. In to do te mere, da če uporabimo tekstovni indeks s sintakso MATCH AGAINST se normalni indeksi enostavno ignorirajo (velja za polja, ki imajo oba indeksa). Primer:

```
... WHERE MATCH(citat) AGAINST("ključna beseda")
AND avtor = "Janez";
```

Najprej bi se iskali vsi citati, ki vsebujejo besedi *ključna* ali *beseda*, šele na to bi se iskalo po avtorju. Poleg tega se navadni indeks na polju *avtor* ne bi upošteval, kar bi upočasnilo celotno iskanje.

6.2 Tekstovni indeksi v PostgreSQL

Tako kot MariaDB tudi PostgreSQL podpira tekstovne indekse. Vendar PostgreSQL funcionalnosti tekstovnih indeksov zelo razširi[40]. Ne podpira

samo razdelitve na žetone in seznama nepomembnih besed (stopwords), ampak podpira tudi pretvarjanje žetonov v lekseme, uporabo slovarja Ispell[2], Thesaurus[3] in Snowball[4].

Vse te dodatne funkcionalnosti omogočimo z uporabo že prednastavljenih konfiguracij za tekstovne indekse, ali pa naredimo čisto svojo. Lastno konfiguracijo bomo naredili za slovenski jezik, saj PostgreSQL tekstovni indeksi ne podpirajo slovenščine. Tekstovni indeksi v PostgreSQL podpirajo štiri tipe konfiguracij:

- tekstovni razčlenjevalniki (text search parsers) - dokument razbijejo na žetone in klasificirajo vsak žeton (npr. beseda, številka)
- tekstovni slovarji (text search dictionaries) - normalizirajo vsak žeton in zavrnejo nepomembne besede
- tekstovne predloge (text search templates) - zagotovijo funkcije uporabljenih slovarjev
- tekstovne konfiguracije (text search configurations) - izberejo razčlenjevalnik in zbirko slovarjev za normalizacijo žetonov, ki jih je ustvaril razčlenjevalnik

Polnotekstovno iskanje v PostgreSQL deluje z uporabo operatorja @@, ki vrne drži ali nedrži tako, da primerja `tsvector` (dokument) z `tsquery` (poizvedba). Vrstni red ni pomemben.

```
> SELECT 'Edini pogum, ki kaj šteje,
          je tisti, ki te popelje od
          danes do jutri'::tsvector
> @@ 'danes & jutri'::tsquery;

 ?column?
----------
 t
```

[2]Pregledovalnik pravopisa
[3]Slovar sopomenk
[4]Algoritmi za pretvarjanje besed v njihove leme

Kot vidimo iz primera lahko uporabljamo tudi logične izraze. V našem primeru smo hoteli izvedeti ali podani niz vsebuje tako besedo *danes*, kot tudi besedo *jutri*. Ta način ima eno pomankljivost. Besede v nizu niso v svoji osnovni obliki (lemi) in ločila tudi niso odstranjena. Če bi hoteli poiskati besedo *pogum* naletimo na težavo

```
> SELECT 'Edini pogum, ki kaj šteje,
          je tisti, ki te popelje od
          danes do jutri'::tsvector
> @@ 'pogum'::tsquery;

 ?column?
----------
 f
```

Poizvedba bo delovala šele, ko dodamo poleg besede *pogum* tudi vejico

```
> SELECT 'Edini pogum, ki kaj šteje,
          je tisti, ki te popelje od
          danes do jutri'::tsvector
> @@ 'pogum,'::tsquery;

 ?column?
----------
 t
```

Zaradi tega je veliko bolje uporabiti funkciji `to_tsvector` in `to_tsquery`

```
> SELECT to_tsvector('Edini pogum, ki kaj šteje,
                     je tisti, ki te popelje od
                     danes do jutri')
> @@ to_tsquery('pogum & tisti');

 ?column?
```

```
----------
 t
```

Tako kot pri MariaDB bomo tudi tukaj zgradili tabelo citatov. Sintaksa ostaja enaka, paziti je potrebno le pri narekovajih, saj se PostgreSQL ne razume z dvojnimi narekovaji

```
CREATE TABLE citati (citat VARCHAR(150) NOT NULL);

INSERT INTO citati
VALUES ('Skrivnost uspeha vsakega uspešnega človeka je,
         da je razvil drugačne navade od tistih, ki so neuspešni.');

INSERT INTO citati
VALUES ('Nihče ne more uspeti brez pričakovanj!');

INSERT INTO citati
VALUES ('Da bi v življenju uspeli, se moramo delati neumne,
         a biti modri.');

INSERT INTO citati
VALUES ('Samo eno pravilo ti bo vlilo pogum, in sicer pravilo,
         da nobeno zlo ne traja večno niti zelo dolgo.');

INSERT INTO citati
VALUES ('Edini pogum, ki kaj šteje, je tisti,
         ki te popelje od danes do jutri');

INSERT INTO citati
VALUES ('Kje je šola za pogum? Šola za pogum je,
         da narediš tisto, v kar verjameš!');

INSERT INTO citati
```

```
VALUES ('Življenje se meri po delih, ne po dnevih.');

INSERT INTO citati
VALUES ('Življenjski cilj vsakega posameznika je vedno isti:
         napredovanje v dobrem.');
```

Polnotekstovno iskanje deluje na podoben način kot pri MariaDB z eno manjšo razliko. Deluje namreč tudi brez tekstovnih indeksov. PostgreSQL funkcionalnosti polnotekstovnega iskanja ponuja preko funkcij in ne preko tekstovnih indeksov. Indeksi namreč le pohitrijo poizvedbe. Razdelitev na žetone in lekseme, primerjava sopomenk in ostale funkcionalnosti opravijo funkcije same. Zaradi tega spodnja poizvedba deluje

```
> SELECT citat
> FROM citati
> WHERE to_tsvector(citat) @@ to_tsquery('življenje');
                   citat
-------------------------------------------
 Življenje se meri po delih, ne po dnevih.
```

Uspešno smo dobili rezultat, vendar zakaj samo enega. Ali ni tako, da funkciji `to_tsvector` in `to_tsquery` normalizirata besede (iščeta po leksemih)? To seveda drži, vendar mi smo vnesli slovenske besede. Če kot prvi argument pri funkciji `to_tsvector` ne specificiramo konfiguracije za jezik v katerem je naš dokument, se bo uporabil jezik, ki je nastavljen v konfiguraciji *default_text_search_config*, ki jo najdemo v datoteki *postgresql.conf*. Osnovna prednastavljena konfiguracija, ki jo dobimo ob namestitvi PostgreSQL je namreč konfiguracija za angleščino. To torej pomeni, da je funkcija želela normalizirati slovenske besede z angleško konfiguracija, kar seveda ne deluje. Da bomo lahko uporabljali slovenščino bo potrebno narediti lastno konfiguracijsko datoteko. Izdelava konfiguracijske datoteke je bila opisana v tretjem poglavju *Prilagajanje slovenskemu jeziku*.

Po izdelavi slovenske konfiguracije lahko začnemo uporabljati polno funk-

cionalnost polnotekstovnega iskanja v PostgreSQL in lahko kreiramo tekstovni indeks

```
> CREATE INDEX ftidx
> ON citati
> USING gin(to_tsvector('slovenian', citat));
```

Tukaj omenimo še to, da lahko tekstovne indekse izdelamo tudi nad več polji naenkrat. Z uporabo ukaza `to_tsvector` si lahko podrobneje ogledamo kako slovenska konfiguracija razdeli besede na žetone in vrne lekseme

```
> SELECT to_tsvector('Delovanje računalniškega programa
je odvisno od njegove izvedbe.');
                          to_tsvector
-------------------------------------------------------------
 'delovanj':1 'izvedb':8 'je':4 'njegov':7 'od':6 'odvisno':5
 'programa':3 'računalniškega':2
```

Kot lahko vidimo smo dobili žetone za vse besede, razen *in*. Ta je namreč v našem seznamu nepomembnih besed. Žetoni so razporejeni po abecednem vrstnem redu, vsak pa ima tudi zapisano mesto kjer se nahaja v nizu. Beseda *Delovanje* je bila normalizirana v besedo *delovanj*. Delovanje funkcije je zelo odvisno od definiranega slovarja. Če torej v slovarju ni zapisa za lematizacijo besede *računalniškega*, je funkcija ne zna lematizirati. S funkcijo `setweight` lahko posameznim poljem dodamo težo. To pomeni, da lahko določimo katero polje je bolj pomembno od drugega. Teže so predstavljene v obliki črk A, B, C in D.

Tudi funkcija `ts_query`, ki uporabljamo kot iskalno polje za iskanje po dokumentih (`ts_vector`), razdeli besede na žetone in vrne lekseme

```
> SELECT to_tsquery('Življenje | danes');
     to_tsquery
---------------------
 'življenj' | 'dane'
```

Prav tako lahko našim ključnim besedam po katerih iščemo dodamo tudi težo

```
> SELECT to_tsquery('Življenje & danes:A');
      to_tsquery
-----------------------
 'življenj' & 'dane':A
```

Če besed ne ločimo z operatorji ali jih ne zapremo v narekovaje bo `ts_query` vrnil napako

```
> SELECT to_tsquery('Življenje danes');
ERROR:  syntax error in tsquery: "Življenje danes"
```

Temu se izognemo tako, da uporabimo funkcijo `plainto_totsquery`

```
> SELECT plainto_tsquery('Življenje pogum');
   plainto_tsquery
----------------------
 'življenj' & 'pogum'
```

Vendar ta ne zmore prepoznati boolean operatorjev, tež in predpon.

6.3 Tekstovni indeksi v MongoDB

MongoDB kot dokumentni nerelacijski SUP zveni primerni za delo s teksti v naravnem jeziku in je z verzijo 2.4 začel eksperimentalno podpirati tekstovne indekse. Od verzije 2.6 naprej pa so del samega MongoDB-ja. V času pisanju pa je izšla tudi nova verzija 3.2, kjer so predstavljeni tekstovni indeksi verzije 3.[21] Prednosti, ki jih prinašajo so naslednje

- Izboljšana podpora za velike in male črke (npr. ne razlikuje med é, É, e, and E),
- neobčutljivost na znake nad črkami (npr. ne razlikuje med é, ê, and e),
- dodatna ločila za tokenizacijo (pomišljaj, vezaj, vzorec, narekov, končno ločilo in presledek).

MongoDB pa na žalost tudi z verzijo 3 tekstovnih indeksov ne podpira slovenščine. To pomeni, da bomo morali v primerih uporabe polnotekstovnega iskanja primorani uporabljati angleščino. Le tako bodo lahko uporabili vse funkcionalnosti polnotekstovnih indeksov kot so neobčutljivost na velike in male črke, neobčutljivost na diaktične znake, ignorairanje nepomembnih besed itd. Jeziki, ki jih trenutno podpira so

- angleščina
- arabščina
- danščina
- dari
- finščina
- francoščina
- italijanščina
- kitajščina (poenostavljena)
- kitajščina (tradicionalna)
- madžarščina
- nemščina
- nizozemščina
- norveščina
- perzijščina
- portugalščina
- romunščina
- ruščina
- turščina
- urdu
- španščina
- švedščina

Če bi želeli sami implementirati podporo slovenščini to lahko naredimo. Celotna izvorna koda MongoDB je na voljo na Github-u[28]. Za implementacijo novega jezika moramo dodati programsko kodo v funkcije namenjene delu z različnimi jeziki pri uporabi tekstovnih indeksov. Programska koda se nahaja v podmapi *src/mongo/db/fts*[54]. Ker MongoDB podpira dinamične poizvedbe lahko začnemo vnašati podatke v bazo ne da bi prej definirali zbirko. Enako kot pri MariaDB in PostgreSQL bomo za prikaz delovanja tekstovnih indeksov uporabili zbirko citatov.

```
db.citati.insert({
    "avtor":"A. Jackson King",
    "citat":"Skrivnost uspeha vsakega uspešnega človeka je,
             da je razvil drugačne navade od tistih,
             ki so neuspešni."
})

db.citati.insert({
  "avtor":"Ralph Charell",
  "citat":"Nihče ne more uspeti brez pričakovanj!"
})

db.citati.insert({
  "avtor":"Montes",
  "citat":"Da bi v življenju uspeli,
           se moramo delati neumne, a biti modri."
})

db.citati.insert({
  "avtor":"Epikur",
  "citat":"Samo eno pravilo ti bo vlilo pogum,
           in sicer pravilo,
           da nobeno zlo ne traja večno niti zelo dolgo."
})

db.citati.insert({
  "avtor":"Mignon Mclaughlin",
  "citat":"Edini pogum, ki kaj šteje,
           je tisti, ki te popelje od danes do jutri."
})
```

```
db.citati.insert({
  "avtor":"El Cordobes",
  "citat":"Kje je šola za pogum? Šola za pogum je,
           da narediš tisto, v kar verjameš!"
})

db.citati.insert({
  "avtor":"Pietro T. Metastasio",
  "citat":"Življenje se meri po delih, ne po dnevih."
})

db.citati.insert({
  "avtor":"Lev Nikolajevič Tolstoj",
  "citat":"Življenjski cilj vsakega posameznika je
           vedno isti: napredovanje v dobrem."
})
```

Tekstovni indeks naredimo podobno kot navadni indeks. Povemo polje na katerem želimo izdelati indeks in uporabimo ključno besedo `text`:

```
> db.citati.createIndex({"citat":"text"})
```

Z ukazom `getIndexes` lahko pogledamo vse indekse v

```
> db.quotes.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "test.quotes"
  },
```

```
  {
    "v" : 1,
    "key" : {
      "_fts" : "text",
      "_ftsx" : 1
    },
    "name" : "quote_text",
    "ns" : "test.quotes",
    "weights" : {
      "quote" : 1
    },
    "default_language" : "english",
    "language_override" : "language",
    "textIndexVersion" : 3
  }
]
```

Torej imamo dva indeksa. Prvi indeks MongoDB samodejno izdela glede na enoličen ključ za vsak zapis v kolekciji. Drugi indeks pa je ravnokar izdelani tekstovni indeks. Iz opisa indeksa lahko tudi vidimo, da je prednastavljeni jezik angleščina in da resnično uporabljamo tekstovni indeks verzije 3. Sedaj pa testirajmo naš indeks še s poizvedbo:

```
> db.citati.find({$text: {$search: "Pogum ŠOLA"}})
{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b57"),
  "avtor" : "Epikur",
  "citat" : "Samo eno pravilo ti bo vlilo pogum,
             in sicer pravilo, da nobeno zlo ne
             traja večno niti zelo dolgo."
}

{
```

```
  "_id" : ObjectId("56a62bc97c70ca4f09f79b58"),
  "avtor" : "Mignon Mclaughlin",
  "citat" : "Edini pogum, ki kaj šteje, je tisti,
             ki te popelje od danes do jutri."
}

{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b59"),
  "avtor" : "El Cordobes",
  "citat" : "Kje je šola za pogum? Šola za pogum je,
             da narediš tisto, v kar verjameš!"
}
```

Dobili smo tri rezultate. Čeprav iskane besede vsebujejo tudi velike črke, smo še vedno dobili rezultate, ki smo jih želeli. MongoDB namreč pred iskanjem vhodne besede normalizira (torej jih spremeni v korene in male črke). Ker pa delamo s slovenskimi besedili na angleškem tekstovnem indeksu korenjenje ne deluje najbolje. Če bi radi videli kako MongoDB uvrsti rezultate dodamo `score:{$meta:"textScore"}`:

```
> db.citati.find(
  {$text: {$search: "Pogum ŠOLA"}},
  {score: {$meta: "textScore"}})
  .sort({score: {$meta: "textScore"}}
)
```

Dobimo rezultate:

```
{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b59"),
  "avtor" : "El Cordobes",
  "citat" : "Kje je šola za pogum? Šola za pogum je,
             da narediš tisto, v kar verjameš!",
  "score" : 1.7
```

```
}

{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b58"),
  "avtor" : "Mignon Mclaughlin",
  "citat" : "Edini pogum, ki kaj šteje, je tisti,
             ki te popelje od danes do jutri.",
  "score" : 0.5384615384615384
}

{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b57"),
  "avtor" : "Epikur",
  "citat" : "Samo eno pravilo ti bo vlilo pogum,
             in sicer pravilo, da nobeno zlo ne traja
             večno niti zelo dolgo.",
  "score" : 0.5277777777777778
}
```

Sedaj smo prikazali številke na podlagi katerih MongoDB razvršča rezultate po pomembnosti. Hkrati pa smo jih tudi ročno prikazali od največjega proti najmanjšemu. Drugi in tretji rezultat imata oba samo po eno ponovitev besede *pogum*, vendar je njun rezultat različen. To zgodi zaradi tega, ker na rezultat ne vplivajo samo ponovitve, vendar tudi dolžina zapisa v polju. To lahko vidimo, če pogledamo v izvorno kodo MongoDB[58] v metodo _scoreStringV2 na vrstici 185. Torej, če je št. ponovitev besede pri obeh 1 (spremenljivki data.count in data.freq), na koeficient (vrstica 219), ki se upošteva pri končnem rezultatu, vpliva samo dolžina žetonov (spremenljivka numTokens). Tako kot MariaDB in PostgreSQL tudi MongoDB podpira iskanje z logičnimi operatorji. Če enostavno navedemo tekst v narekovajih potem se izvede iskanje z operatorjem **OR**. To pomeni, da vrne vse rezultate kjer je vsaj ena beseda enaka kot v besedilu. Če bi želeli iskati z operatorjem

AND bi morali uporabiti še ene narekovaje:

```
> db.citati.find({$text: {$search: "\"vlilo pogum\""}})
{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b57"),
  "avtor" : "Epikur",
  "citat" : "Samo eno pravilo ti bo vlilo pogum,
             in sicer pravilo, da nobeno zlo ne traja
             večno niti zelo dolgo."
}
```

Na tak način torej lahko iščemo fraze. Lahko pa iščemo tudi tako, da povemo katere beseda naj ne bo v besedilu:

```
> db.citati.find({$text: {$search: "-vlilo pogum šola"}})

{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b58"),
  "avtor" : "Mignon Mclaughlin",
  "citat" : "Edini pogum, ki kaj šteje, je tisti,
             ki te popelje od danes do jutri."
}

{
  "_id" : ObjectId("56a62bc97c70ca4f09f79b59"),
  "avtor" : "El Cordobes",
  "citat" : "Kje je šola za pogum? Šola za pogum je,
             da narediš tisto, v kar verjameš!"
}
```

Ko si želimo ogledati kako delujejo normalizacija, fraze in logičnimi operatorji lahko uporabimo ukaz `explain` in si ogledamo del rezultata s ključem `parsedTextQuery`:

```
> db.citati.find({$text: {$search: "\"vlilo pogum\" -ŠOLA"}}).explain(true)
"parsedTextQuery" : {
"terms" : [
"pogum",
"vlilo"
],
"negatedTerms" : [
"sola"
],
"phrases" : [
"vlilo pogum"
],
"negatedPhrases" : [ ]
},
```

Polnotekstovno iskanje pa ima tudi eno pomanjkljivost, podpira namreč le en tekstovni indeks na zbirko. Poleg tega omogoča tudi kreiranje indeksa na več poljih. V kolikor smo že prej kreirali indeks, ga moramo najprej odstaniti

```
> db.citati.dropIndex("citat_text")
{ "nIndexesWas" : 2, "ok" : 1 }
```

in nato zgraditi novega, kjer vanj vključimo več polj:

```
> db.citati.createIndex({"avtor":"text", "citat":"text"})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

MongoDB nam omogoča, da poljem, ki jih indeksiramo, določimo tudi težo. To pomeni, da želimo nekemu polju določiti večjo prioriteto kot ostalim. V

našem primeru lahko damo polju `quote` težo 10, polju `author` pa težo 5, vendar moramo to storiti že pri kreaciji indeksa:

```
> db.citati.createIndex({"avtor":"text", "citat":"text"},
{"weights": {avtor: 5, citat: 10}})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

Če bi želeli dodati tekstovni indeks na vsako tekstovno polje (tipi polj se določijo samodejno) lahko to storimo z operatorjem `$**` na sledeči način:

```
> db.citati.createIndex({"$**":"text"})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

MongoDB ima torej kar dobro podporo za polnotekstovno iskanje. Vendar na žalost ne podpira slovenščine, kar pomeni, da je v našem primeru najmanj primerna baza. Za razliko od PostgreSQL, funkcionalnosti polnotekstovnega iskanja ne moremo enostavno razširiti z lastnimi slovarji in seznamom nepomembnih besed. Poleg tega ne shranjuje pozicije besed in ne podpira sinonimov. Tudi iskanje fraz zna biti dokaj počasno.

Poglavje 7

Rezultati in analiza

V tem poglavju se bomo lotili testiranja učinkovitosti posameznih SUPB-jev na problemih kolokacij besed. Vse meritve bodo izvedene na korpusu ccKres. Pri MariaDB bomo testirali tudi različne shranjevalne mehanizme. Poudarek bo na Aria, MyISAM in InnoDB shranjevalnih mehanizmih. PostgreSQL ima samo en shranjevalni mehanizem, vendar pa podpira različne indekse. Po pregledu dokumentacije smo ugotovili, da je so za besedila najprimernejši *B indeksi*, ker podpirajo vse operacije primerjave rezultatov ($>$, $\geq$, $=$, $\leq$, $<$) [55]. Pri MongoDB bomo prav tako kot pri PostgreSQL uporabili samo en indeks in en shranjevalni mehanizem. Pri vseh meritvah so rezultatati podani v sekundah in zaokroženi na tri decimalke, poleg tega pa je bil podatkovni strežnik pred vsako meritvijo ponovno zagnan (rezultati niso v predpomnilniku in zato ne pospešijo poizvedb). Vsak test je bil izveden na sto izbranih besed iz baze. Pri vsaki besedi se je testirana razdalja sosednjih besed povečevala od ena do deset. To pomeni, da je bila najprej izvedena poizvedba, ki je poiskala prejšnjo in naslednjo besedo po iskani. To najlažje prikažemo na primeru stavka. Če imamo stavek *"Danes je lep dan."* in iščemo sosednje besede na razdalji ena od besede *lep*, potem sta to besedi *je* in *dan*. Pri večji razdalji se upoštevajo tudi vse besede na manjših razdaljah. Vse skupaj je bilo torej izvedenih po tisoč ponovitev na posamezni test.

Poglavje je razdeljeno na štiri podpoglavja, ki prikazujejo rezultate tako v tabelarični kot grafični podobi. Vsaka tabela vsebuje z vrsticami ločene rezultate za posamezne SUPB-je. V stolpcih so rezultati ločeni glede na število klientov povezanih na podatkovni strežnik (ena, pet ali dvajset klientov hkrati). Grafi so prav tako kot tabele razdeljeni tako, da prikazujejo vsak SUPB na horizontalni osi, na vertikali pa so logaritemske skale hitrosti podane v sekundah. Vsak SUPB vsebuje tri stolpce, ki z različnimi barvami prikazujejo čase, glede na število hkrati povezanih klientov na podatkovni strežnik. Prav tako, kot pri tabeli so vrednosti za ena, pet in dvajset klientov.

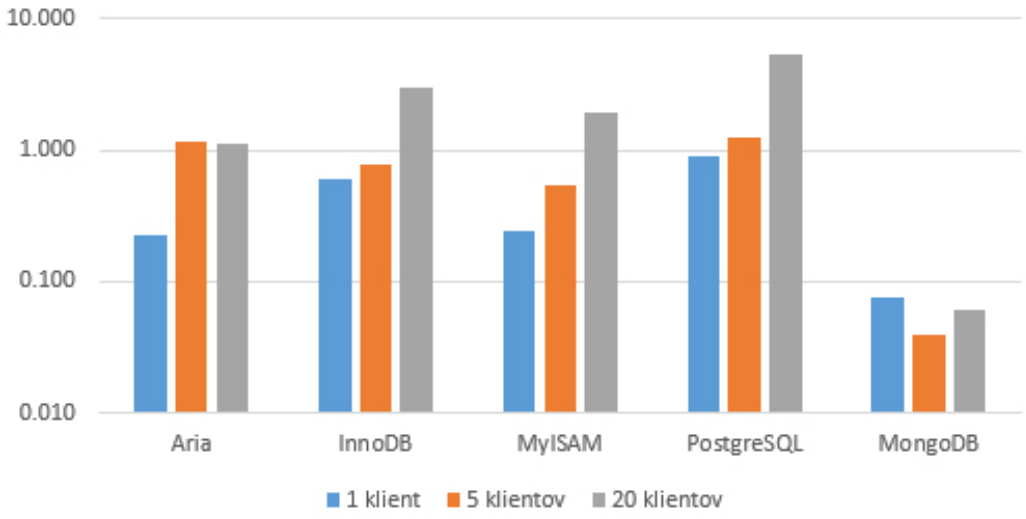
7.1 Povprečje hitrosti iskanja po besedi in lemi

Baza	1 klient	5 klientov	20 klientov
Aria	0,224	1,166	1,119
InnoDB	0,605	0,773	2,961
MyISAM	0,244	0,549	1,969
PostgreSQL	0,892	1,247	5,430
Mongo	0,075	0,040	0,060

Tabela 7.1: Tabela povprečja meritev časov (v sekundah) iskanja po besedi

Pri povprečnih hitrostih smo ugotovili, da je najhitrejši MongoDB, kjer je presenteljivo najhitrejši čas pri petih hkrati povezanih klientih (0,040s). Druga ugotovitev je, da je Aria shranjevalni mehanizem hitrejši od MyISAM shranjevalnega mehanizna (MariaDB). Pri enem klientu ima čas 0,224s, MyISAM pa 0,244s. Pri dvajsetih klientih vodi z 1,119s v primerjavi z MyISAM (1,969s). MyISAM je hitrejši samo pri petih klientih z 0,549s, Aria pa ima 1,116s. Za najpočasnejšega se je izkazal PostgreSQL, kjer je čas pri dvajsetih klientih 5,430s. Na predzadnjem mestu je pristal InnoDB, kjer je čas pri

petih klientih celo hitrejši od Arie z 0,773s proti 1,166s.



Slika 7.1: Graf povprečja meritev časov (v sekundah) iskanja po besedi

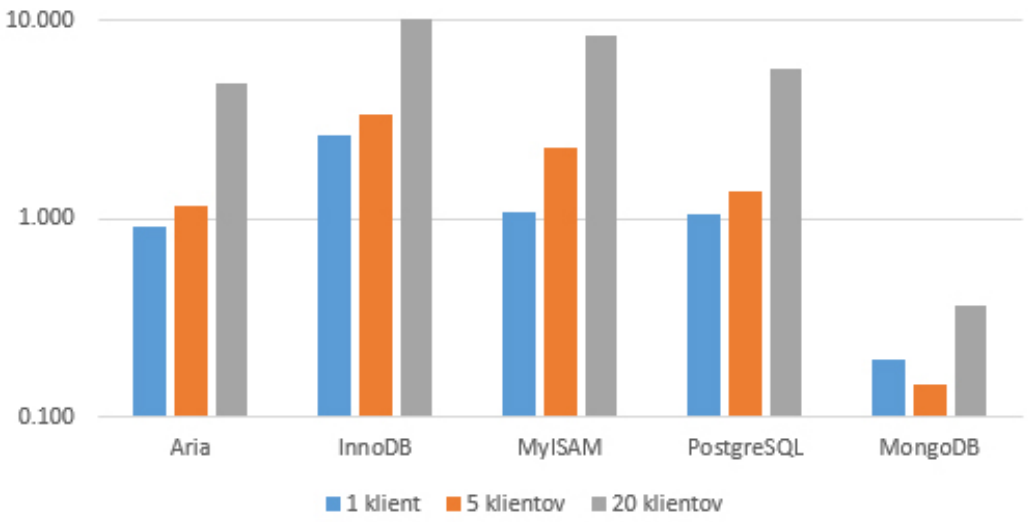
Sledi test povprečnih hitrostih, kjer se je za iskanje namesto besede uporabilo iskanje po lemi. To pomeni več zadetkov, kar hkrati pomeni daljši časi trajanja poizvedb.

Baza	1 klient	5 klientov	20 klientov
Aria	0,903	1,166	4,864
InnoDB	2,661	3,338	12,494
MyISAM	1,070	2,262	8,346
PostgreSQL	1,044	1,362	5,784
Mongo	0,194	0,147	0,364

Tabela 7.2: Tabela povprečja meritev časov (v sekundah) iskanja po lemi

Še vedno vodi MongoDB. Najhitrejši čas je dosegel pri petih klientih z 0,147s, hkrati pa so časi vseh treh meritev še vedno pod polovico sekunde. Tudi tokrat je Aria shranjevalni mehanizem hitrejši od MyISAM pri vseh treh

meritvah. Pri enem klientu je čas Arie pod sekundo z 0,903s, kjer ima MyISAM malo nad sekundo z 1,070s. Pri petih klientih je razlika med njima več kot sekundo z 1,146s. Pri dvajsetih klientih pa razlika skoči na 3,482s. Pri iskanju po lemi je PostgreSQL celo boljši od MyISAM v vseh treh testih. Na zadnjem mestu je InnoDB, ki ima pri dvajsetih klientih celih 12,494s.



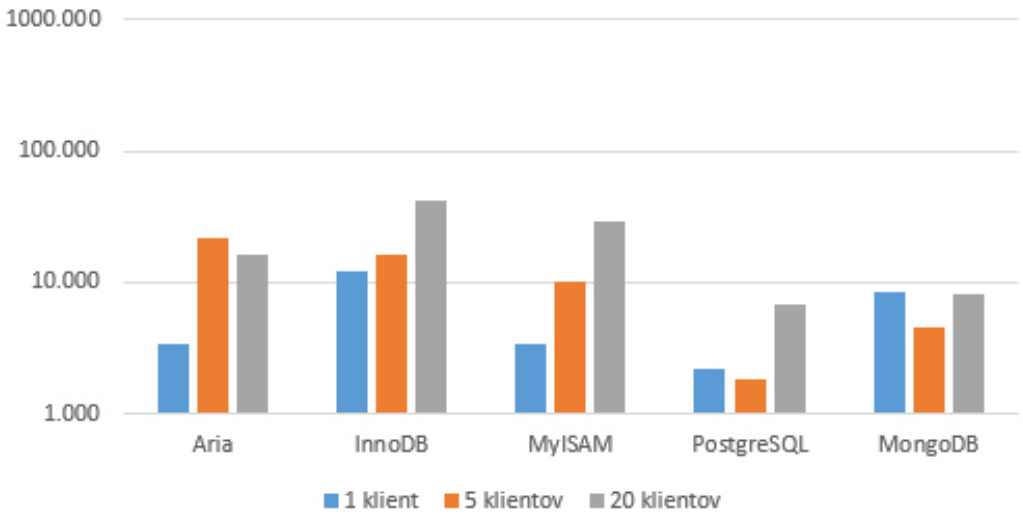
Slika 7.2: Graf povprečja meritev časov (v sekundah) iskanja po lemi

7.2 Najslabši časi iskanja po besedi in lemi

Baza	1 klient	5 klientov	20 klientov
Aria	3,420	22,043	16,328
InnoDB	12,093	16,350	42,502
MyISAM	3,431	10,011	28,677
PostgreSQL	2,218	1,859	6,812
Mongo	8,394	4,537	8,059

Tabela 7.3: Tabela najslabših časov meritev (v sekundah) iskanja po besedi

Tu so prikazani maksimalni časi poizvedb. Pri enem klientu vodita Aria z 3,420s in MyISAM z 3,431s. Pri petih klientih je od MyISAM hitrejši za 5,474s, od Arie pa za celih 17,506s. MyISAM je od Arie hitrejši pri petih klientih, vendar je pri dvajsetih klientih počasnejši za 12,349s. Presenetljivo je InnoDB hitrejši od Arie pri petih klientih z 16,350s v primerjavi z 22,043s. Vendar pri dvajstetih klientih potrebuje 26,174s več od Arie. Še bolj presenetljivo je, da je PostgreSQL na prvem mestu, sledi mu MongoDB, kjer je čas pri dvajsteih klientih slabši za 1,247s.

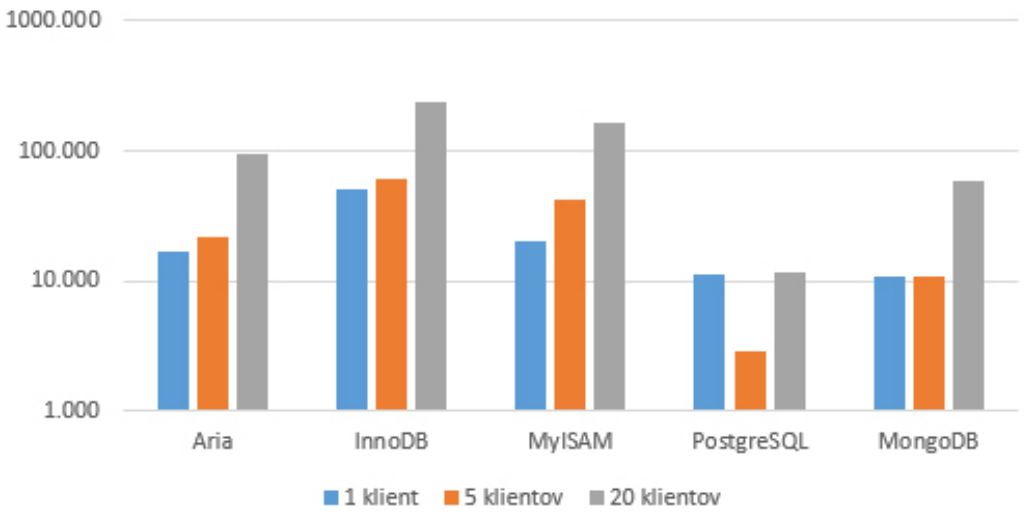


Slika 7.3: Graf najslabših časov meritev (v sekundah) iskanja po besedi

Baza	1 klient	5 klientov	20 klientov
Aria	16,861	22,043	94,855
InnoDB	50,058	60,215	240,447
MyISAM	20,154	41,935	162,161
PostgreSQL	11,321	2,898	11,732
Mongo	10,750	10,810	57,643

Tabela 7.4: Tabela najslabših časov meritev (v sekundah) iskanja po lemi

Sledijo rezultati iskanja po lemi. Tudi tukaj je najhitrejši PostgreSQL, ki ima pri enem klientu čas 11,321s (samo pri tem času je MongoDB hitrejši od njega), presenetljivo ima pri dvajstetih klientih skoraj enak čas kot pri enem z 11,732s. Hkrati pa je s tem časom najhitrejši pri dvajsetih klientih. Drugi je ponovno MongoDB, kjer je njegov najboljši čas pri enem klientu (10,750s) primerljiv s časov od PostgreSQL pri dvajsetih klientih. Aria je tudi tokrat hitrejša od MyISAM v vseh treh meritvah. Njen čas je pri enem klientu za 3,293s boljši. Pri dvajsetih klientih pa je razlika za 67,306s. Zadnji je InnoDB, ki pri dvajsetih klientih potrebuje celih 240,447s.



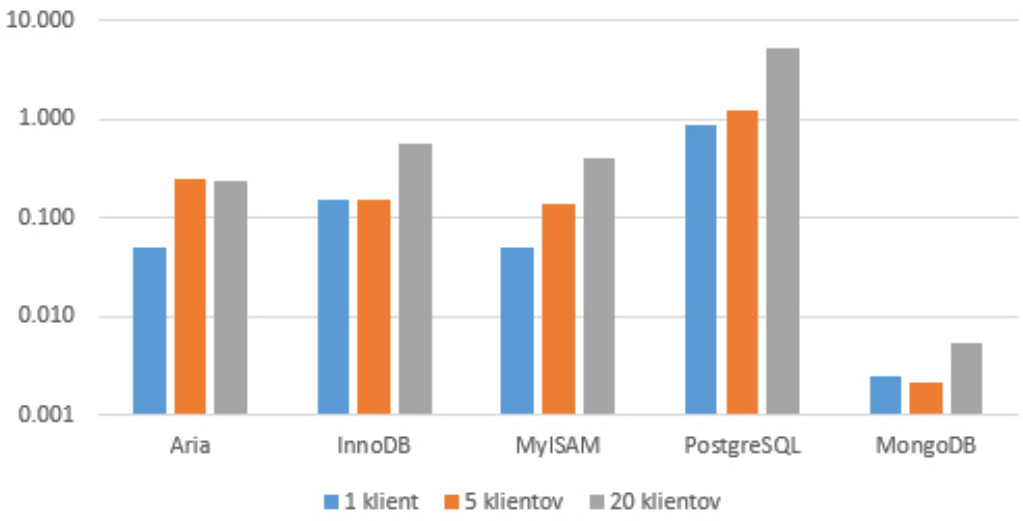
Slika 7.4: Graf najslabših časev meritev iskanja po lemi

7.3 Mediana hitrosti iskanja po besedi in lemi

Baza	1 klient	5 klientov	20 klientov
Aria	0,051	0,253	0,234
InnoDB	0,154	0,152	0,566
MyISAM	0,050	0,142	0,395
PostgreSQL	0,863	1,242	5,391
Mongo	0,002	0,002	0,005

Tabela 7.5: Tabela median meritev časov (v sekundah) iskanja po besedi

Tukaj so prikazane mediane meritev. Z mediano podamo srednjo vrednost danega zaporedja. Mediane vseh rezultatov se gibljejo pod sekundo, kjer je izjema PostgreSQL pri petih in dvajsetih klientih. Pri petih klientih je mediana malo nad sekundo z 1,242s, pri dvajsetih pa 5,391s. Najmanjšo mediano ima MongoDB, ki je pri enem in pri petih klientih enaka z 0,002s. Pri vseh testih je vedno prvi ali drugi, zato je tudi mediana tako manjhna.



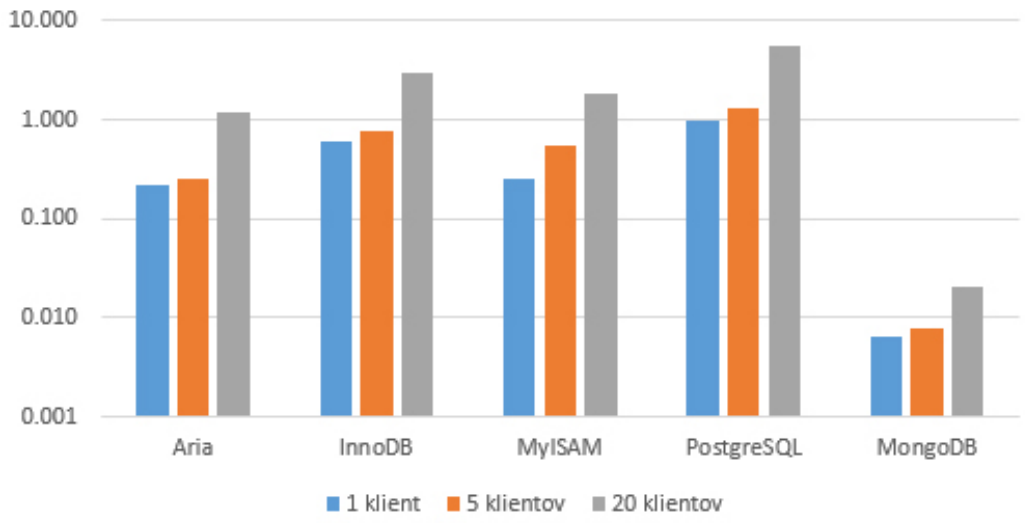
Slika 7.5: Graf median meritev časov (v sekundah) iskanja po besedi

Sedaj pa mediane iskanja po lemi

Baza	1 klient	5 klientov	20 klientov
Aria	0,216	0,253	1,165
InnoDB	0,609	0,768	2,942
MyISAM	0,259	0,558	1,813
PostgreSQL	0,961	1,328	5,629
Mongo	0,006	0,008	0,020

Tabela 7.6: Tabela median meritev časov (v sekundah) iskanja po lemi

Ponovno najbolj izstopa PostgreSQL, vendar so vrednosti dokaj podobne kot pri iskanju po besedi. Vrednosti so seveda višje, ker iskanje po lemah vrne več rezultatov, kar pomeni daljši časi izvajanja poizvedbe. Pri vseh ostalih pa so časi pri dvajsetih klientih narasli na več kot sekundo, razen seveda MongoDB. Aria vodi z 1,165s, drugi je MyISAM z 1,813s, tretji pa InnoDB z 2,942s, kar je skoraj 3s.



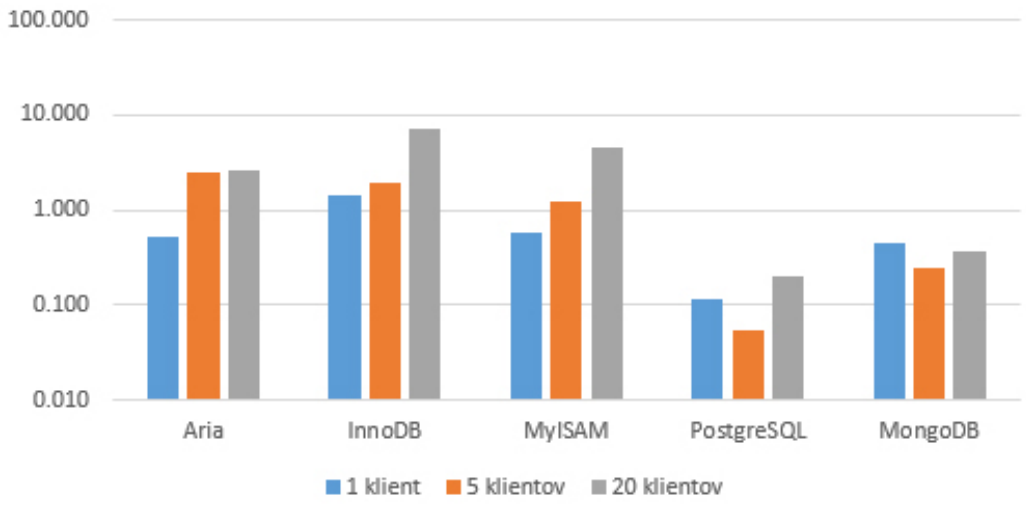
Slika 7.6: Graf median meritev časov (v sekundah) iskanja po lemi

7.4 Standardni odklon hitrosti iskanja po besedi in lemi

Baza	1 klient	5 klientov	20 klientov
Aria	0,514	2,550	2,630
InnoDB	1,419	1,887	6,984
MyISAM	0,565	1,236	4,662
PostgreSQL	0,117	0,053	0,197
Mongo	0,448	0,242	0,364

Tabela 7.7: Tabela standarnih odklonov meritev časov (v sekundah) iskanja po besedi

Standardni odklon nam pove za koliko vrednosti odstopajo od povprečja. Najmanjše odstopanje ima PostgreSQL, ki je pri petih klientih samo 0,053s. Drugi je InnoDB z 6,984s, prav tako pri dvajsteih klientih. Vsa odstopanja pri MongoDB so pod polovico sekunde. Največje je pri enem klientu z 0,448s. Aria in MyISAM sta pri enem klientu primerjljiva, pri petih klientih pa je MyISAM boljši za 1,314s. Pri dvajsetih klientih pa je Aria boljša za 2,032s.



Slika 7.7: Graf standardnih odklonov meritev časov (v sekundah) iskanja po besedi

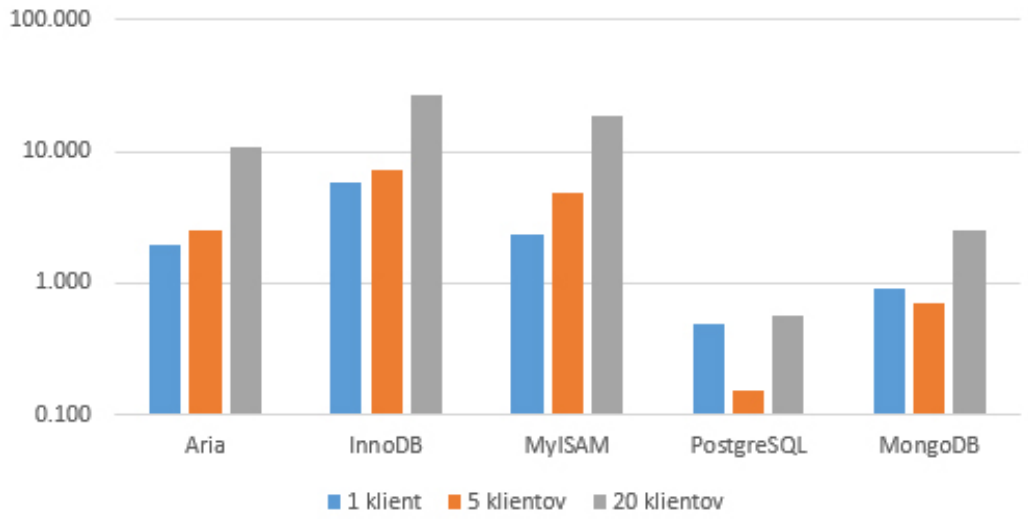
Zadnji pa je standardni odklon pri iskanju z lemo

Baza	1 klient	5 klientov	20 klientov
Aria	1,951	2,550	10,655
InnoDB	5,924	7,285	27,283
MyISAM	2,327	4,907	18,770
PostgreSQL	0,484	0,151	0,566
Mongo	0,901	0,701	2,523

Tabela 7.8: Tabela standardnih odklonov meritev časov (v sekundah) iskanja po lemi

Ponovno najmanj odstopa PostgreSQL z najboljšim rezultatom pri petih klientih z 0,151s. Vse vrednosti PostgreSQL so namreč pod eno sekundo. MongoDB sekundo preseže šele pri dvajsetih klientih s časom 2,523s. Aria in MyISAM se pri enem klientu razlikujeta za 0,376s, pri petih klientih za 2,550s, pri dvajsetih klientih pa za 8,115s. V vseh primerih je hitrejši Aria.

Zadnje mesto zaseda InnoDB, kjer se čas pri enem klientu začne pri skoraj šestih sekundah z 5,9924s, pri dvajsetih klientih pa je 27,283s.



Slika 7.8: Graf standardnih odklonov meritev časov (v sekundah) iskanja po lemi

Poglavje 8

Sklepne ugotovitve

Sodobni SUPB-ji nam z uporabo polnotekstovnega iskanja omogočajo izredno močno iskanje po tekstih, vendar je njihova uporaba za slovenščino vprašljiva. Kdor, bi si želel samo namestiti SUPB, shraniti vanj slovenske tekste, ustvariti polnotekstovne indekse in pričakoval popolno delovanje, bo razočaran. Že v osnovi brez dodatnega dela namreč noben sistem slovenščine ne podpira. Pri MariaDB je največ kar lahko naredimo to, da ustvarimo seznam nepomembnih besed. To pomeni, da ne moremo pričakovati, da bo sistem vedno našel besedo, ki smo jo vpisali. Nima namreč nikakršne podpore za korenjenje ali lematizacijo besed v slovenščini. Pri PostgreSQL lahko z nekaj dodatnega dela ustvarimo lastno konfiguracijo, ki bo znala ignorirati nepomembne besede in vrniti lekseme besed v iskalnem nizu. Pod pogojem, da imamo seveda datoteke, ki specificirajo navodila leksimacije besed v slovenščini. Paketi, ki jih za to ponuja OpenOffice so namreč le zadovoljivi. Če bi pa radi uporabljali najnovešjo tehnologijo kot je MongoDB, pa na podporo slovenščine lahko kar pozabimo, saj ne podpira niti dodajanje lastnega seznama nepomembnih besed. Ni pa nemogoče narediti podpore slovenščini, saj je celotna izvorna koda MongoDB dosegljiva na spletu. To pomeni, da lahko naredimo lastno prirejeno verzijo MongoDB, ki vključuje podporo za slovenščino. Pri tem moramo le poskrbeti, da bomo z nadgradnjo podporo obdržali.

Če pa želimo implementirati kolokacijo besed, kot je opisana v diplomski nalogi, je MongoDB najboljša izbira. V večini ima najhitrejše čase in najmanjšo odstopanje v hitrostih pri večjemu številu hkrati povezanih uporabnikov. Vendar je tukaj potrebno omeniti, da bi najverjetneje s spremenjeno strukturo tudi ostale baze lahko imele rezultate primerljive z MongoDB. Z enako strukturo je MongoDB namreč prepočasen. V okviru MariaDB je shranjevalni mehanizem Aria prehitel shranjevalni mehanizem MyISAM. Torej drži, da je Aria, kot trdijo avtorji MariaDB, izboljšana različica MyISAM. Za kolokacijo pa je najmanj primeren shranjevalni mehanizem InnoDB, saj je bil skoraj v vseh rezultatih na zadnjem mestu.

Kot vidimo, imamo na področju polnotekstovnega iskanja po tekstih v naravnem jeziku tudi v splošnonamenskih SUPB precej možnosti. Le podpora slovenščini je precej skromna, z izjemo PostgreSQL. Očitno bi bila na področju dela s teksti v naravnem jeziku potrebna iniciativa, s katero bi omogočili vključitev podpore slovenskemu jeziku v pomembnejše odprtokodne SUPB.

Literatura

[1] MySQL. [Online]. Dosegljivo: https://www.mysql.com/ [Dostopano 15. 1. 2016].

[2] MariaDB. [Online]. Dosegljivo: https://mariadb.org/ [Dostopano 15. 1. 2016].

[3] PostgreSQL. [Online]. Dosegljivo: http://www.postgresql.org/ [Dostopano 15. 1. 2016].

[4] Sporazumevanje v slovenskem jeziku. [Online]. Dosegljivo: http://www.slovenscina.eu/ [Dostopano 15. 1. 2016].

[5] Korpus Kres. [Online]. Dosegljivo: http://www.slovenscina.eu/korpusi/kres [Dostopano 18. 1. 2016].

[6] ccKres. [Online]. Dosegljivo: https://www.clarin.si/repository/xmlui/handle/11356/1034 [Dostopano 18. 1. 2016].

[7] Korpus Gigafida. [Online]. Dosegljivo: http://www.slovenscina.eu/korpusi/gigafida [Dostopano 18. 1. 2016].

[8] Spletni konkordančnik Gigafida. [Online]. Dosegljivo: http://www.gigafida.net/ [Dostopano 18. 1. 2016].

[9] ccGigafida. [Online]. Dosegljivo: https://www.clarin.si/repository/xmlui/handle/11356/1035 [Dostopano 18. 1. 2016].

[10] Označevalnik Obeliks. [Online]. Dosegljivo: http://www.slovenscina.eu/tehnologije/oznacevalnik [Dostopano 18. 1. 2016].

[11] Kolokacija. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Kolokacija [Dostopano 15. 1. 2016].

[12] About MariaDB. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/about-mariadb/ [Dostopano 20. 9. 2015].

[13] MariaDB versus MySQL - Features. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/ [Dostopano 20. 9. 2015].

[14] What is MongoDB. [Online]. Dosegljivo: https://www.mongodb.com/what-is-mongodb [Dostopano 17. 01. 2015].

[15] Fulltext Index Overview. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/fulltext-index-overview/ [Dostopano 28. 9. 2015].

[16] What is PostgreSQL?. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.4/interactive/intro-whatis.html [Dostopano 28. 9. 2015].

[17] GIN Indexes. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.4/static/gin-intro.html [Dostopano 23. 12. 2015].

[18] Full Text Search - Introduction. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.4/interactive/textsearch-intro.html [Dostopano 28. 9. 2015].

[19] The MongoDB 3.2 Manual. [Online]. Dosegljivo: https://docs.mongodb.org/manual/ [Dostopano 18. 1. 2016].

[20] Introduction to MongoDB. [Online]. Dosegljivo: https://docs.mongodb.org/manual/core/introduction/ [Dostopano 28. 9. 2015].

[21] Text Indexes. [Online]. Dosegljivo: https://docs.mongodb.org/manual/core/index-text/ [Dostopano 29. 12. 2015].

[22] $text. [Online]. Dosegljivo: https://docs.mongodb.org/v3.2/reference/operator/query/text/#op._S_text [Dostopano 18. 1. 2016].

[23] Wildcard Text Indexes. [Online]. Dosegljivo: https://docs.mongodb.org/manual/core/index-text/#text-index-wildcard [Dostopano 29. 9. 2015].

[24] Text Search Languages. [Online]. Dosegljivo: https://docs.mongodb.org/v3.2/reference/text-search-languages/ [Dostopano 18. 1. 2016].

[25] MariaDB Stopwords. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/stopwords/ [Dostopano 19. 1. 2016].

[26] MySQL Full-Text Stopwords. [Online]. Dosegljivo: http://dev.mysql.com/doc/refman/5.7/en/fulltext-stopwords.html [Dostopano 19. 1. 2016].

[27] InnoDB Fulltext Stopwords. [Online]. Dosegljivo: https://mariadb.com/blog/mysql-56-innodb-fulltext-round-2-stopwords [Dostopano 19. 1. 2016].

[28] MongoDB izvorna koda. [Online]. Dosegljivo: https://github.com/mongodb/mongo [Dostopano 20. 1. 2016].

[29] B. Schwartz, P. Zaitsev, V. Tkachenko *High Performance MySQL, 3rd Edition*, O'Reilly Media, 2012, str. 154.

[30] Najdaljša slovenska beseda. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Najdalj%C5%A1a_slovenska_beseda [Dostopano 15. 12. 2015].

[31] Full-Text Restrictions. [Online]. Dosegljivo: http://dev.mysql.com/doc/refman/5.7/en/fulltext-restrictions.html [Dostopano 20. 12. 2015].

[32] PostgreSQL FAQ. [Online]. Dosegljivo: https://wiki.postgresql.org/wiki/FAQ [Dostopano 22. 12. 2015].

[33] OpenOffice wiki. [Online]. Dosegljivo: https://wiki.openoffice.org/wiki/Dictionaries [Dostopano 25. 12. 2015].

[34] D. Hows, P. Membrey, E. Plugge, T. Hawkins *The Definitive Guide to MongoDB, Third Edition*, Apress, 2015, str. 181-189

[35] Full-Text Search in MongoDB. [Online]. Dosegljivo: http://code.tutsplus.com/tutorials/full-text-search-in-mongodb–cms-24835 [Dostopano 29. 12. 2015].

[36] Apache Lucene. [Online]. Dosegljivo: https://lucene.apache.org/core/ [Dostopano 15. 01. 2016].

[37] 3 Key Capabilities Necessary for Text Analytics & Natural Language Processing in the Era of Big Data. [Online]. Dosegljivo: https://blog.pivotal.io/data-science-pivotal/products/3-key-capabilities-necessary-for-text-analytics-natural-language-processing-in-the-era-of-big-data. [Dostopano 20. 1. 2016].

[38] What is PostgreSQL?. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.4/interactive/intro-whatis.html [Dostopano 20. 1. 2016].

[39] Configuring MariaDB with my.cnf. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/configuring-mariadb-with-mycnf/ [Dostopano 20. 1. 2016].

[40] PostgreSQL Dictionaries. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.3/interactive/textsearch-dictionaries.html [Dostopano 20. 1. 2016].

[41] PostgreSQL Fulltext Configuration. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.3/interactive/textsearch-configuration.html [Dostopano 20. 1. 2016].

[42] UTF-8. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/UTF-8 [Dostopano 20. 1. 2016].

[43] Lema. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Lema [Dostopano 20. 1. 2016].

[44] MariaDB: DATE. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/date/ [Dostopano 20. 1. 2016].

[45] MariaDB: VARCHAR. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/varchar/ [Dostopano 20. 1. 2016].

[46] MariaDB: MEDIUMTEXT. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/mediumtext/ [Dostopano 20. 1. 2016].

[47] Seznam besednih vrst. [Online]. Dosegljivo: http://nl.ijs.si/jos/msd/html-sl/msd.index.categories.html [Dostopano 21. 1. 2016].

[48] Ubuntu PostgreSQL Help Wiki. [Online]. Dosegljivo: https://help.ubuntu.com/community/PostgreSQL [Dostopano 21. 1. 2016].

[49] PostgreSQl Numeric Types. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.3/static/datatype-numeric.html [Dostopano 21. 1. 2016].

[50] MongoDB CRUD Introduction. [Online]. Dosegljivo: https://docs.mongodb.org/v3.0/core/crud-introduction/ [Dostopano 21. 1. 2016].

[51] MariaDB REGEXP. [Online]. Dosegljivo: https://mariadb.com/kb/en/mariadb/regexp/ [Dostopano 24. 1. 2016].

[52] MongoDB. [Online]. Dosegljivo: https://www.mongodb.com/ [Dostopano 2. 2. 2016].

[53] Leksem. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Leksem [Dostopano 3. 2. 2016].

[54] MongoDB Fulltext Search Source. [Online]. Dosegljivo: https://github.com/mongodb/mongo [Dostopano 5. 2. 2016].

[55] PostgreSQL Index Types. [Online]. Dosegljivo: http://www.postgresql.org/docs/9.3/static/indexes-types.html [Dostopano 10. 2. 2016].

[56] Lemma (linguistics). [Online]. Dosegljivo: https://simple.wikipedia.org/wiki/Lemma_(linguistics) [Dostopano 6. 3. 2016].

[57] Koren. [Online]. Dosegljivo: http://gradiva.txt.si/slovenscina/slovenscina-za-triletne-sole/jezik-in-besedilne-vrste/besedoslovje/besedoslovje/053_pomenska_razmerja-2/obravnava-584/ [Dostopano 6. 3. 2016].

[58] MongoDB Fulltext Spec. [Online]. Dosegljivo: https://github.com/mongodb/mongo/blob/master/src/mongo/db/fts/fts_spec.cpp [Dostopano 8. 3. 2016].

[59] Aggregation. [Online]. Dosegljivo: https://docs.mongodb.org/manual/aggregation/ [Dostopano 14. 3. 2016].