UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Primož Skale

Načrtovanje in razvoj iskalnika za potrebe policije

MAGISTRSKO DELO

Mentor: prof. dr. Tomaž Pisanski

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 146-MAG-RI/2012
Datum: 16. 04. 2012

Primož Skale, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Načrtovanje in razvoj iskalnika za potrebe policije**

Implementation and Development of Internal Search Engine for Slovenian Police

Tematika naloge:

V okviru magistrskega dela podrobneje načrtujte in izdelajte iskalnik (v poljubnem programskem okolju), ki bo omogočal iskanje in dostop do koristnih informacij v *internih* virih podatkov slovenske policije. Kot primarni vir podatkov vzemite IBM Lotus Notes zbirke.
Izdelajte in implementirajte varnostni model, ki bo učinkovito in hitro filtriral vrnjene zadetke poizvedb glede na uporabnikove pravice dostopa v primarnem viru podatkov.
Izdelajte in implementirajte modul za pridobitev podatkov iz virov in njihovo posredovanje v odprto-kodno rešitev Apache Solr/Lucene.
Teoretično in s primeri pojasnite delovanje indeksiranja in iskanja kot ga izvaja Apache Solr/Lucene. V praktičnem delu naloge predstavite namen in prednosti uporabe takega iskanja za potrebe policije.

M e n t o r:

prof. dr. Tomaž Pisanski

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

D e k a n:

prof. dr. Nikolaj Zimic

Sem vstavite **originalni** list teme magistrske naloge!

Izjava o avtorstvu

Spodaj podpisani **Primož Skale**,

z vpisno številko **63080467**,

sem avtor magistrskega dela z naslovom:

Načrtovanje in razvoj iskalnika za potrebe policije

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom **prof. dr. Tomaža Pisanskega**,
- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.), ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,

in

- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 24. 9. 2012 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju, prof. dr. Tomažu Pisanskemu, za vso pomoč in navodila pri nastajanju tega magistrskega dela.

Enako se zahvaljujem svoji družini, ženi Tanji in hčerki Neži, za vso podporo in spodbujanje, ko sem ob pisanju dela že omahoval in ga končno le dokončal – brez vaju ne bi videl cilja.

Če smo obrnjeni v pravo smer,
moramo v tej smeri samo naprej.

Budistična modrost

Kazalo

Povzetek

V zadnjih desetletjih smo priča neizmernemu širjenju svetovnega spleta in informacijskih storitev, ter s tem tudi količine informacij, ki jih morajo posamezniki – predvsem pa družbe in organizacije – obdelati za uspešno delovanje. Na svetovnem spletu je na voljo večje število zmogljivih in pogosto uporabljenih spletnih iskalnikov, ki – vsaj za svetovni splet – dobro opravljajo svoje delo. Podatki niso shranjeni le na spletu, temveč imajo razne organizacije (banke, zavarovalnice, državne ustanove, ipd.) svoje zbirke elektronskih podatkov po katerih dnevno iščejo. Vprašanje je, ali je tako iskanje uspešno, kvalitetno, hitro in predvsem varno?

V ta namen nekatere družbe za razvoj programske opreme (IBM, Microsoft, ipd.) ponujajo plačljive sisteme, ki te podatke ustrezno indeksirajo in pripravijo za hitro in učinkovito iskanje. Problemi, s katerimi se soočajo organizacije, so predvsem, kako dostopati v pravem trenutku do potrebnih informacij na varen način.

V policiji smo zato s pomočjo dveh prosto dostopnih, odprto-kodnih rešitev (Apache Lucene in Apache Solr), ki že vsebujeta funkcionalnosti sodobnih iskalnikov, razvili lastni iskalnik po internih podatkih policije, kjer smo kot primarni vir podatkov vzeli zbirke IBM Lotus Notes. Uspešno smo implementirali overitev uporabnikov in zagotavljanje varnostne politike dostopa do iskalnika in podatkov.

Magistrska naloga predstavlja opis sistemov za izbiranje in iskanje informacij (angl. *Information Retrieval*) v neposredni povezavi z rešitvama Apache Lucene in Solr, ter razvoj in uvedbo internega iskalnika policije v realno okolje.

Ključne besede: pridobivanje informacij, sistemi za izbiranje in pridobivanje informacij, iskalniki, indeksiranje, Apache Solr, Apache Lucene, vektorsko-prostorski model, razvoj iskalnikov

Abstract

The amount of information that modern society receives and processes is, simply said, enormous. Individuals, groups and organizations are faced with the daily overload of information from which they want to extract only the useful and quality information. On the World Wide Web are available many powerful and widely used Web search engines that - at least for the Internet - perform their work well. The data is not only stored on the Internet, but different organizations (banks, insurance companies, government institutions, etc.) have their own private data warehouses on which employees perform many daily queries. The question is whether such queries return quality information in a secure manner?

To this end, some software development companies (IBM, Microsoft, etc..) are offering paid solutions that prepare and index this information for fast and efficient searching. Organizations are faced with problems as how to prevent access by unauthorized persons and to ensure a high quality of the returned information.

At police we have developed our own internal search engine that is based on two freely available, open source solutions (Apache Lucene and Apache Solr) which already contain the functionality of modern search engines. For the primary source of data we took IBM Lotus Notes databases. We have successfully implemented the authentication of users and the provision of security policy and access to search data.

This Master Thesis presents a description of the systems for information retrieval in direct connection with solutions Apache Lucene and Apache Solr, and the development and implementation of an internal search engine into production for use in Police.

Keywords: information retrieval systems, search engines, indexing, Apache Solr, Apache Lucene, vector-space model, development of enterprise search engine

1 Uvod

V zadnjih desetletjih smo priča neizmernemu širjenju svetovnega spleta in s tem tudi količine informacij, ki nam je v vsakem trenutku na voljo preko raznih spletnih iskalnikov ter podobnih storitev. Večina sistemov zagotavlja iskanje informacij na spletu, le peščica pa omogoča iskanje znotraj organizacij, kot so banke, zavarovalnice, državne ustanove, javne organizacije, ipd. S povečevanjem količine internih podatkov, ki so shranjeni v različnih virih in oblikah, se povečuje tudi problem iskanja koristnih informacij. Zahteve po zmogljivih sistemih za iskanje in izbiranje informacij (angl. *Information Retrieval* – IR), ki lahko te informacije učinkovito in hitro obdelajo, so zato čedalje glasnejše. Hitra obdelava pomeni zmanjšanje časa in prostora, ki ga sistem potrebuje za obdelavo podatkov, učinkovita obdelava pomeni pravilno prepoznavo koristnih informacij za uporabnika. V praksi je težko združiti učinkovitost in hitrost, zato je v načrtovanje in razvoj takih sistemov vloženega veliko truda, ki to vrzel zmanjša do te mere, da sistem zadovoljivo deluje na obeh področjih [2].

Problema, s katerima se soočamo sta, kako hitro in učinkovito iskati po potrebnih virih na enem mestu in obenem zagotoviti, da do teh podatkov ne bodo dostopale nepooblaščene osebe. Za ustrezno zaščito podatkov je potrebno varnostno politiko virov po katerih iščemo, združiti v enotni varnostni model, katerega politiko bo izvajal iskalnik in s tem onemogočil zlorabo oz. uhajanje občutljivih informacij. Razvoj varnostnega mehanizma ni edini manjkajoči del, temveč je potrebno podatke iz primarnih virov ustrezno prebrati in jih posredovati sistemu za indeksiranje podatkov, ki vhodne podatke prevede v ustrezno obliko in jih zapiše na način, da je do njih možno hitro in učinkovito dostopati.

V policiji smo se lotili razvoja lastnega iskalnika, ki bo sprva pridobival podatke iz zbirk IBM Lotus Notes in jih shranil v indeks za poznejše iskanje preko spletnega vmesnika. Na tak način smo omogočili zaposlenim na policiji hiter dostop do želenih informacij, kjer smo odpravili potrebo po odpiranju in iskanju več različnih zbirk Lotus Notes in izvajanje iskanja po vsaki zbirki posebej znotraj odjemalca Lotus Notes. S tem smo prihranili na času, ki je bil potreben, da so zaposleni pridobili ustrezno informacijo in omogočili učinkovitejše opravljanje delovnih procesov na policiji.

Programske rešitve, ki ustrezajo zgornjim zahtevam (tj. zahtevi po varnostnem mehanizmu, hitremu iskanju, pridobivanju podatkov iz različnih tipov virov, ipd.), vsekakor že obstajajo, tako v plačljivi obliki (npr. Google Search Applicance, Microsoft FAT, IBM Omnifind, ipd.) kot v brezplačni obliki (npr. Apache Lucene, Apache Solr, ipd.) [10]. Glavna slabost plačljivih rešitev je njihova cena, večinoma ponujajo preveč funkcionalnosti oz. ne zadostujejo nekaterim posameznim zahtevam (npr. varnostnemu mehanizmu).

V policiji smo preizkusili rešitev IBM Omnifind, ki se je izkazal za izjemno kompleksnega in težavnega za razvoj dodatnih modulov in funkcionalnosti, zato smo kasneje preizkusili odprto-kodno rešitev Apache Lucene, ki je prenosljiva med sistemi in omogoča hitro indeksiranje podatkov (do 95 GB/uro na sodobnih računalnikih), ima majhne zahteve po delovnem spominu, učinkovito dostopa do indeksa (inkrementalno indeksiranje je enako hitro

kot prvo indeksiranje), in omogoča napredne funkcije iskanja (razvrstitev seznama zadetkov po pomembnosti, različni tipi poizvedb, razvrstitev seznama zadetkov glede na poljuben podatek v indeksu, porazdeljeno iskanje po več indeksih hkrati, ipd.) [11].

Ker Apache Lucene ni samostojna rešitev, ki jo uporabniki preprosto namestijo in pričnejo uporabljati, smo kasneje uporabili podobno odprto-kodno rešitev Apache Solr, ki v ozadju uporablja Apache Lucene in vključuje večino funkcionalnosti, ki jih zahtevamo. Z uporabo te rešitve nam je bil prihranjen razvoj nekaterih funkcionalnosti (npr. porazdeljeno iskanje, replikacija indeksa, definicija indeksnih polj, ipd.) in omogočena takojšnja uporaba teh naprednih funkcionalnosti, brez poseganja v programsko kodo. Uporaba odprto-kodnih rešitev je omogočila hiter razvoj iskalnika, saj je bilo na voljo dovolj dokumentacije, ki podrobno obravnava posamezne procese sistema Apache Solr.

Iskalnik, ki smo ga razvili, je prenosljiv tudi v druge organizacije, ki za vire podatkov uporabljajo zbirke Lotus Notes. Okrnjena različica iskalnika kriminalističnim inšpektorjem omogoča hitro iskanje po zaseženih podatkih in s tem tudi hitrejšo obdelavo primerov. Določeni primeri so prikazali, da so kriminalisti do ključnih informacij prišli v nekaj minutah – pred uporabo iskalnika je bilo do teh informacij potrebno priti preko ročnega pregledovanja dokumentacije oz. zaseženih podatkov.

Zagotavljanje dostopa do ustreznih in celovitih informacij preiskovalcem in analitikom je ključ do uspešnega reševanja kriminalističnih primerov in preprečevanja kriminalnih dejanj. V preteklosti – in tudi danes – je nezmožnost povezljivosti različnih informacijskih sistemov med seboj in kvalitete izmenjave informacij med raznimi preiskovalnimi agencijami, pripeljala do izgube človeških življenj [18].

5. novembra 2009 je major ameriške vojske Nidal Malik Hasan, ki je v oporišču Fort Hood v Teksasu delal kot psihiater, ubil 12 vojakov sodelavcev in 29 ranil. Kasnejša preiskava dejanja je pokazala, da se je Hasan dopisoval z ameriškim radikalnim islamistom živečim v Jemnu, Umarjem F. Abdulmutallabijem, ki je bil kasneje povezan s terorističnim dejanjem (v kartuše laserskih tiskalnikov je z željo strmoglavljenja tovornih letal namenjenih v Združene države namestil eksplozivna sredstva). FBI je že slabo leto pred napadom Hasana na sodelavce vedela za korespondenco med njim in Abdulmutallabijem, vendar je zaradi napake v komunikaciji med FBI-jem in ostalimi agencijami in zastarelih informacijskih sistemov, ter nemožnosti uporabe naprednih tehnik iskanja po zaseženih podatkih, preiskava zastala do samega napada [18].

New yorška policija (NYPD) je leta 2009 uvedla inovativni informacijski sistem za upravljanje z informacijami, ki se osredotoča na zagotavljanje ustreznih in celovitih informacij policistom od trenutka, ko pričnejo z delom na novem primeru. NYPD uporablja programsko rešitev IBM Infosphere Data Warehouse za zagotavljanje ključnih informacij policistom in preiskovalcem takoj po prejetju informacije o kriminalnem dejanju – kratek odzivni čas sistema in posredovanje ustrezne informacije omogoča izvrševanje morebitnih hitrih nadaljnjih preiskav in aretacij ključnih osumljencev. Policisti imajo dostop do dnevnikov klicev na številko za nujne primere, 911, datotek pridržanj, kazenskih ovadb in aretacij, ter kriminalističnih poročil – vse na nacionalnem nivoju. Zmožnost lažje preučitve obstoječih informacij preko analitike in

vpogleda v različna podatkovna skladišča, omogoča policistom hitreje ukrepati in s tem reševati življenja ljudi [19].

Enako je New yorška policija letos (2012) s sodelovanjem Microsofta začela uvajati nov projekt, *Domain Awarness System*, ki bo razvit za potrebe združevanja in analize obstoječih sistemov javne varnosti v realnem času. Sistem bo združeval prejete informacije iz več kot 3000 različnih video-nadzornih sistemov, 100 sistemov za prepoznavo registrskih tablic in detektorjev sevanja, nameščenih na področju spodnjega Manhattna (angl. *Lower Manhattan*). Sistem bo prav tako prejemal informacije iz sistemov drugih agencij in jih ustrezno povezal z informacijami v lastnem sistemu. Združeval bo podatke o klicih na številko za nujne primere, prepoznavo registrskih tablic, kriminalističnih poročil, ipd. Ker se bodo video-nadzorne vsebine hranile za obdobje 30 dni, bo npr. lažje ugotoviti, kdo je namestil sumljiv paket. Preiskovalci, ki bodo naleteli na sumljiv paket, postavljen na javnem kraju, bodo identificirali ustrezne video-nadzorne sisteme, ki so posneli paket in s pomočjo sistema identificirali osebo, ki je paket namestila. Zaradi varovanja osebnih podatkov sistem ne bo avtomatsko prepoznaval obrazov ljudi in bo namenjen le nadzoru javnih površin. Vrednost celotne investicije se ocenjuje na 90 milijonov ameriških dolarjev [20, 21].

Varnostne, policijske in obveščevalne agencije po vsem svetu lahko v današnjem času informacijske preobremenjenosti s podatki uspešno rešujejo primere le, ko jih je omogočeno učinkovito in hitro iskanje ključnih informacij.

1.1 Struktura naloge

V nadaljevanju naloge predstavimo splošno delovanje sistemov za izbiranje in iskanje informacij, kjer opišemo proces indeksiranja podatkov in iskanja po indeksu, ki ga tak proces pripravi.

Nadalje, preko primerov, teorijo povežemo z odprto-kodno rešitvijo Apache Lucene, v nadaljevanju na kratko opišemo še eno odprto-kodno rešitev, Apache Solr, ki deluje v neposredni povezavi z Apache Lucene.

Predstavimo tudi primarni vir podatkov, ki sestoji iz zbirk IBM Lotus Notes, ter varnostni model, ki ga uporablja sistem Lotus.

V zadnjih dveh poglavjih opišemo interni iskalnik, ki smo ga razvili za potrebe policije. Opišemo posamezne komponente sistema ter njihovo delovanje. Podrobno predstavimo delovanje varnostnega modela, ki ga iskalnik uporablja in ki v celoti imitira varnostni model sistema Lotus.

V zaključku povzamemo še glavne rezultate in ugotovitve.

2 Sistemi za izbiranje in iskanje informacij

V tem razdelku bomo predstavili splošno delovanje sistemov za izbiranje in iskanje informacij. Predstavili bomo proces pridobivanja informacij iz virov podatkov in kako ter na kakšen način se te informacije obdelajo ter shranijo v indeks, od koder jih nato sistemi IR črpajo, ko uporabniki izvršujejo poizvedbe. Podrobno bomo razložili proces indeksiranja, izgradnjo fizične datoteke, ki predstavlja indeks, porazdeljeno indeksiranje in obdelavo vhodnih podatkov pred samim procesom shranjevanja. V nadaljevanju bomo razložili kako se izvedejo poizvedbe uporabnikov ter predstavili odprto-kodni rešitvi Apache Lucene in Apache Solr, ter vektorsko-prostorski model (angl. *vector-space model*) za določanje pomembnosti dokumentom in iskanje podobnih dokumentov.

2.1 Izbiranje in iskanje informacij

Področje izbiranja in iskanja informacij zajema predstavitev, shranjevanje in dostopanje do informacij. Akademsko področje bi področje IR definiralo kot [4]:

> *Izbiranje in iskanje informacij je iskanje nestrukturirane (tekstovno besedilo) vsebine (dokumentov) iz neke velike kolekcije (ki je shranjena na računalnikih) tako, da ta zadostuje potrebi po informacijah.*

Sprva je bilo iskanje informacij na način, ki je definiran zgoraj, omejeno na ozka področja znotraj knjižnic, univerz, ipd. Sodobne sisteme IR danes dnevno uporablja nekaj milijonov ljudi, ki izvedejo nekaj milijard poizvedb dnevno. Razlog razmaha sistemov IR tudi na druga področja je neverjeten razvoj interneta oz. svetovnega spleta (angl. *World Wide Web* – WWW). Količina informacij, ki je shranjena in na voljo uporabnikom na svetovnem spletu dnevno narašča, in s tem se tudi povečuje problem uspešnega iskanja koristnih informacij. Če so bili pred dvajsetimi leti ročno grajeni seznami spletnih strani po področjih (npr. Yahoo) dovolj dobri za širšo množico ljudi, temu dan danes ni več tako. Število spletnih strani se povečuje hitreje, kot bi jih ljudje lahko ročno pregledovali. Zato so spletni iskalniki, ki preiščejo celotni splet in najdene informacije indeksirajo za kasnejše hitro iskanje, zelo popularni in na veliko uporabljeni (npr. Google, Yahoo!, Bing, DuckDuckGo, ipd.). Osrednji pomen sodobnih sistemov IR (predvsem spletnih) je ta, da zadetke uredijo po pomembnosti. Najtežji del procesa za pridobivanje informacij je ravno odločanje kateri dokumenti ustrezajo oz. so sorodni določeni poizvedbi. Sistemi IR dosežejo največjo uspešnost, ko so najbolj ustrezni dokumenti najvišje na seznamu zadetkov, najmanj ustrezni pa nižje na seznamu. Kot že rečeno v uvodu, pa uspešnost sistemov IR ni določena samo s tem, kako dobro razvrstijo dokumente na seznamu zadetkov, temveč tudi od tega kako hitro se poizvedba izvrši in se uporabniku vrne seznam zadetkov.

Razmerje med pravilnostjo zaporedja seznama zadetkov in hitrostjo vrnjenih zadetkov določa kvaliteto sistema IR.

Sisteme IR ločimo na tri področja na katerih delujejo [4]:

- Iskanje po spletu
- Iskanje po osebnih informacijah (angl. *personal information retrieval*)
- Iskanje po internih informacijah (angl. *enterprise, institutional, domain-specific search*)

Na področju spletnega iskanja mora sistem zagotoviti iskanje med milijardami dokumentov, ki so shranjeni na več milijonih računalnikov. Zagotoviti je potrebno ustrezne procese, ki te dokumente pridobijo, obdelajo in nato shranijo v indeks, ne glede na velikost področja, ki ga predstavlja splet. Prav tako morajo taki procesi pravilno obravnavati strani, ki jih upravitelji ustvarijo z namenom, da bi zvišali njihovo pomembnost in bi bile take strani na seznamu zadetkov višje kot sicer.

Področje iskanja po osebnih informacijah je ravno nasprotje iskanja po svetovnem spletu, saj so informacije po katerih uporabniki iščejo shranjene lokalno na njihovih delovnih postajah. Rešitve, ki tako iskanje omogočajo so npr. Google Desktop Search ali Windows Instant Search oz. so neposredno vgrajene npr. v odjemalce elektronske pošte. Količina informacij, ki jih te rešitve obdelujejo so neprimerno manjše od količine informacij, ki jih ponuja splet, zato je obdelava takih informacij lažja in so zato rešitve, ki to omogočajo strojno manj zahtevne in lahko brez težav tečejo tudi na manj zmogljivih sistemih. S tem uporabniki pridobijo možnost lažjega iskanja informacij po lokalnem sistemu brez zaznavne upočasnitve sistema.

Tretje področje leži med zgoraj omenjenima področjema in predstavlja področje iskanja po internih podatkih raznih podjetij oz. družb. Interne informacije oz. podatki segajo od elektronske pošte zaposlenih, raznih relacijskih baz, datotečnih sistemov do video vsebin in zvočnih zapisov.

To magistrsko delo obravnava sisteme IR, ki segajo na to – tretje – področje.

2.1.1 Indeksiranje

Da lahko sistemi IR zagotovijo hitro in kvalitetno iskanje po informacijah, uporabljajo mehanizem indeksiranja, ki primarne vire informacij prebere in nato te informacije na ustrezni način shrani v indeks. Najpopularnejša varianta podatkovne strukture indeksa, ki ga uporabljajo sistemi IR je t.i. obrnjena oz. invertirana datoteka (angl. *inverted file*), ki je obrnjena predstavitev prvotne kolekcije dokumentov organizirane v sezname (angl. *postings lists*) [4]. Vsak vpis v obrnjeni datoteki vsebuje informacijo o posameznem izrazu iz kolekcije dokumentov. Drugače rečeno, iz vseh dokumentov kolekcije sistem pridobi posamezne izraze (le-ti predstavljajo posamezne besede) in za vsak tak dokument izgradi seznam vseh izrazov, ki se v njem pojavljajo. Za hitro iskanje po takem seznamu je potrebno le-tega »obrniti« in sicer

tako, da se ustvari seznam individualnih izrazov in poleg vsakega izraza seznam dokumentov, kjer se ta izraz pojavi. Nekateri izrazi se pojavljajo v veliki količini dokumentov, drugi morda samo v enem ali dveh dokumentih. Tak seznam se lahko predstavi v sledeči obliki:

$$< t; df_t; d_1, d_2, \dots, d_{f_i} >,$$

kjer df_t predstavlja dokumentno frekvenco izraza (angl. *document frequency*) t (tj. število dokumentov v katerih se pojavi t) in d_i številko dokumenta (predpostavimo, da so dokumenti unikatno oštevilčeni). Seznam lahko vključuje tudi dodatne informacije o izrazu t, npr. kolikokrat se tak izraz pojavi v določenem dokumentu d_i, in njegovem položaju v tem dokumentu glede na začetek besedila [4].

Izgradnja indeksa je veriga procesov, ki obsega vse od identifikacije in prepoznave vhodnih podatkov do samega ustvarjanja končne vsebine, ki bo zapisana v indeks. V razdelku 2.2, kjer bomo predstavili Apache Lucene, bomo enake procese predstavili še iz vidika kot jih vidi Lucene, ter prikazali nekaj preprostih programskih primerov [2].

– Prvi korak indeksiranja predstavlja zajem podatkov iz virov podatkov in prenos le-teh v indekser[1]. Viri podatkov so lahko različni: datoteke na datotečnem sistemu, spletne strani, podatkovne baze, elektronska sporočila, ipd., prav tako tudi procesi zajemanja podatkov, kjer preprostejši delujejo le na en zapis vira hkrati (npr. datoteko), kompleksnejši pa morajo upoštevati, da en zapis vira vodi v drugega (npr. spletne strani, kjer spletna povezava vodi na drugo stran – zajeti je potrebno obe strani, če tako določajo nastavitve globine indeksiranja). Večinoma zajem podatkov iz virov ni trivialen in zahteva uporabo naprednih orodij (tudi plačljivih) v poslovnih okoljih, kjer so viri podatkov različni in vsebujejo različne vrste varnostnih mehanizmov.

– Drugi korak indeksiranja predstavlja prepoznavo zaporedja znakov v posameznem dokumentu. Dokumente se indekserju predstavi kot tok bajtov, ki jih slednji mora pretvoriti v znake. Kritično za dobro delovanje indeksiranja je, da se pretvorba izvrši učinkovito in predvsem pravilno. Dokumenti so lahko kreirani v različnih kodnih tabelah ali jezikih in indekser mora vsebino vseh dokumentov, ki mu jih posredujemo ustrezno pretvoriti v obliko, ki bo obdelana enako ne glede na prvotno obliko zapisa. Tipično indekserji najlažje obdelujejo vsebino, ki je zapisana v obliki UTF-8. Vhodni dokumenti so lahko najrazličnejših formatov (npr. PDF, Microsoft Word, TIFF, HTML, ipd.). Nekateri formati vsebujejo posebne znake, ki jih moramo pred obdelavo odstraniti, spet pri drugih pa moramo samo vsebino najprej pridobiti preko drugih orodij (npr. Apache Tika) ali preko vmesnika API (angl. *Application Programming Interface*) [12].

– Tretji korak predstavlja določitev razdrobljenosti (angl. *granularity*) indeksa. V indeks se lahko zapiše samo podatek o številu pojavitev posameznih izrazov v dokumentih, vendar pa

[1] Z besedo indekser označujemo stroj oz. stroje (npr. računalnike), ki izvajajo indeksiranje nad neko kolekcijo dokumentov.

je v določenih primerih potrebna tudi informacija, ki posamezni izraz umesti v določeno polje, stavek ali odstavek.

– Četrti korak predstavlja pretvorbo posameznih besed dokumentov v žetone (angl. *tokens*), ki jih nato indeksiramo. Na žetone lahko delujemo z analizatorjem, ki jih lahko skrajša, pretvori v drugo obliko, odstrani določene znake, ipd. Nekateri indekserji kot žeton vzamejo zaporedje več besed – ti so znani kot k-gram indekserji, kjer k označuje število besed v enem žetonu.

– Zadnji proces indeksiranja predstavlja izgradnjo dejanskega indeksa, kjer se izvrši inverzija kolekcije in se ustvarita dve podatkovni strukturi: slovar (angl. *dictionary*) in datoteka seznamov (angl. *postings file*). Prva vsebuje vse indeksirane žetone in njihove statistične podatke. Druga vsebuje informacijo o tem v katerih dokumentih se posamezni izraz nahaja, ter posledično tudi v katerih poljih in njihov položaj. Slovar se pogosto hrani v glavnem pomnilniku za hiter dostop, datoteko seznamov pa na trdem disku, saj je velikost slovarja nekaj redov velikosti manjša od velikosti datoteke seznamov. Glavna naloga slovarja je preslikava vsakega indeksiranega žetona z ustreznim položajem tega žetona v datoteki seznamov. Vsak zapis žetona v slovarju mora vsebovati najmanj sledeče tri informacije:

- Dejanski žeton, predstavljen kot niz znakov
- Število dokumentov v katerih se žeton pojavi
- Kazalec na zapis v datoteki seznamov, kjer so shranjene podrobnejše informacije o žetonu

Razlog hranjenja slovarja v glavnem pomnilniku sistema za hitri dostop je predvsem v tem, da ko sistemu pošljemo poizvedbo se posamezne besede poizvedbe (oz. žetoni) primerjajo s seznamom žetonov v slovarju. Taka primerjava poizvedbe in slovarja omogoča hitro določitev besed, ki jih sistem nato uporabi pri dejanskem iskanju dokumentov v indeksu.

2.1.2 Iskanje

Večina uporabnikov, ki uporabljajo spletne ali lokalne iskalnike, se ne obremenjuje s teoretičnimi modeli na katerih sloni proces pridobivanja relevantnih dokumentov iz indeksa glede na vhodno poizvedbo, ter določitev njihove pomembnosti in urejenosti v seznamu zadetkov. Navajeni so, da stvari preprosto delujejo, ne oziraje se na kompleksnost zalednih algoritmov in sistemov.

2.1.2.1 Boolov model

Pred dvema desetletjema so uporabniki izvajali samo t.i. Boolovo iskanje, kjer so se vhodne poizvedbe upoštevale kot izraz Boolove logike [17]. Model, ki opisuje tako iskanje se imenuje

Boolov model. Iskanje po takem modelu je iskanje točnih zadetkov, kar pomeni, da se dokument smatra kot zadetek, le in samo če, ustreza Boolovi logiki poizvedbe. Operatorji, ki se pri tej logiki uporabljajo so trije: `AND`, `OR` in `NOT`. Slabost uporabe Boolovega modela leži v tem, da le-ta vrne neurejen seznam zadetkov – tj. seznam zadetkov, kjer dokumenti niso urejeni po pomembnosti, saj se le-ta ne določi s pomočjo Boolovega modela. To pogosto pomeni, da je lahko najbolj relevanten zadetek šele na zadnjem mestu. Kljub tej očitni slabosti modela, se je le-ta uporabljal praktično v vseh iskalnikih do sredine 90. let. Če je uporabnik izvedel poizvedbo, ki je vrnila veliko število zadetkov, je preprosto razširil oz. natančneje določil poizvedbo in tako zmanjšal število zadetkov. Za določitev najbolj relevantnih zadetkov je bilo nato potrebno ročno pregledati celoten seznam zadetkov. Pomembno je tudi poudariti, da Boolov model ne upošteva strukture dokumentov, ki jih preiskuje (čeprav nekateri iskalniki omogočajo iskanje besed tudi v istem odstavku) ter ne upošteva število pojavitev besed v dokumentu. Z drugimi besedami, dokument je enako relevanten (se smatra kot zadetek) če vsebuje eno pojavitev iste besede, ali večkratno pojavitev iste besede. V nadaljevanju bomo videli, da sodobni modeli za iskanje dokumentov po indeksu upoštevajo tudi število pojavitev besede v posameznih dokumentih ter število pojavitev iste besede v vseh dokumentih kolekcije skupaj [4].

Za primer prikaza iskanje z Boolovo logiko vzemimo tri dokumente (oštevilčene kot d_0, d_1 in d_2), ki vsebujejo sledečo vsebino:

Dokument	**Vsebina dokumenta**
d_0	danes je lep dan
d_1	ali ni dan lep
d_2	lep pozdrav

Vsebino dokumentov razdelimo na posamezne izraze:

»danes«, »je«, »lep«, »dan«, »ali«, »ni«, »pozdrav«

Nekatere izraze bi lahko odstranili (npr. »je« in »ali«) saj se redno pojavljajo v slovenskem besedilu, vendar jih bomo za ponazoritev primera obdržali. Vsebina dokumentov je bila tako izbrana, da *normalizacija* izrazov ni potreba. Normalizacija izraza je proces pri katerem se določi osnovna oblika izraza (lematizacija (angl. *lemmatisation*) ali krnjenje (angl. *stemming*)), črke izraza se spremeni v velike oz. male, odstrani se naglasila, ločila, ponavljajoče znake in zaustavitvene besede (angl. s*top words*) [13]. Normalizacija povzroči, da sistem IR ne razlikuje med besedami »pozdrav«, »pozdrave«, »pozdravi«, … Posledica tega je manjša velikost indeksa, kar doprinese k hitrosti izvajanja poizvedb. Druga prednost normalizacije je tudi ta, da uporabnikom ni potrebno dobesedno navajati točnih besed, saj bo tudi poizvedba, ki jo uporabnik vpiše v iskalnik pred samo izvedbo iskanja, ustrezno normalizirana.

Sedaj, ko imamo posamezne izraze iz vseh treh dokumentov, naredimo seznam vseh dokumentov, kjer se ti izrazi pojavijo (le-ta predstavlja datoteko seznamov):

Izraz	Dokument
danes	d_0
je	d_0
lep	d_0, d_1, d_2
dan	d_0, d_1
ali	d_1
ni	d_1
pozdrav	d_2

Če bi izvedli preprosto poizvedbo[2] `lep dan` bi dobili kot seznam zadetkov množico dokumentov:

$$\{d_0, d_1, d_2\} \cap \{d_0, d_1\} = \{d_0, d_1\},$$

saj se obe besedi pojavita le v prvem (d_0) in drugem dokumentu (d_1). Na tem mestu je pomembno poudariti, da vrstni red izrazov v poizvedbi in dokumentih ni pomemben, pomembno je le to, da dokument vsebuje te izraze. Če bi želeli iskati po dejanskem izrazu `'lep dan'`, kjer želimo, da je vrstni red izrazov v dokumentih enak kot v poizvedbi, moramo v indeksu poleg informacije o vsebovanosti izraza v dokumentu, zabeležiti tudi položaj izraza v dokumentu. Za omenjeni dve besedi bi bila vsebina indeksa sedaj takšna:

Izraz	Dokumenti in položaji izrazov
lep	$\{(d_0, 2);\ (d_1, 3);\ (d_2, 0)\}$
dan	$\{(d_0, 3);\ (d_1, 2)\}$

kjer prva vrednost v oklepaju označuje številko dokumenta, druga pa položaj izraza v tem dokumentu glede na začetek vsebine dokumenta. Položaj prvega izraza v dokumentu je enak 0, položaj drugega je enak 1, itd.

Če uporabnik išče točno besedno zvezo `'lep dan'`, kjer želi, da se vrstni red besed ohranja, potem tej poizvedbi ustreza samo dokument d_0. Če vrstni red besed ni pomemben, potem – tako kot prej – poizvedbi ustrezata dokumenta d_0 in d_1.

Boolov model pridobivanja relevantnih dokumentov iz indeksa torej upošteva le pojavitev določene besede, ki jo iščemo, ne upošteva pa število njenih pojavitev v dokumentu in kolekciji dokumentov kot celoti. Zadetki zato niso urejeni po pomembnosti. V nadaljevanju bomo

[2] Privzamemo, da med izrazoma nastopa operator AND.

predstavili vektorsko-prostorski model, ki to število pojavitev upošteva, v poglavju 2.2 pa tudi razširjen model, ki ga za ocenjevanje zadetkov uporablja Apache Lucene.

2.1.2.2 Vektorsko-prostorski model

Vektorsko-prostorski model predstavlja prvi poskus definicije modela za izbiranje in iskanje informacij, ki ne temelji na Boolovi logiki. Predstavljen je bil že leta 1975 [4]. V tem modelu se množica dokumentov predstavi v obliki vektorjev, kjer je dimenzija vektorjev določena s številom izrazov v slovarju. Ker model sam ne določa kaj naj bi komponente vektorjev predstavljale, lahko iskalniki, ki model integrirajo le-tega poljubno razširijo. Za osnoven prikaz ocenjevanja zadetkov vzemimo, da posamezne komponente vektorjev predstavljajo število pojavitev posameznih izrazov v dokumentih. Vsakemu izrazu v dokumentu bomo torej določili utež, ki je odvisna od števila pojavitev tega izraza v dokumentu. Končni namen je izračun ocene med izrazi t poizvedbe in dokumentom d, glede na utež izraza t v d. Najpreprosteje je utež določiti kot število pojavitev izraza t v dokumentu d. Tej utežni shemi pravimo frekvenca izraza (angl. *term frequency*) in jo označimo kot $\mathrm{tf}_{t,d}$. Uporaba take uteži ima prednosti pred navadnim Boolovim modelom, saj upošteva število pojavitev izraza v dokumentu in ne le samo njegovo pojavitev. Enako pa uporaba take uteži ne upošteva vrstnega reda besed v dokumentih in zato bi bila po tej poti dokumenta z vsebino »Jaka je višji kot Matej« in »Matej je višji kot Jaka« ovrednotena z isto oceno.

Slabost uporabe frekvence izrazov je v tem, da so vsi izrazi enako pomembni. V resnici pa niso vsi izrazi enakovredni, temveč je potrebno upoštevati tudi število njihovih pojavitev v celotni kolekciji dokumentov. Ideja na tem mestu je ta, da zmanjšamo veljavo (tj. utež) tistim izrazom, ki imajo veliko število pojavitev čez celotno kolekcijo. S tem definiramo pojem frekvence kolekcije (angl. *collection frequency*) in jo označimo z cf. Z njeno pomočjo zmanjšamo vrednost uteži tf izraza za določen faktor, ki se povečuje s frekvenco kolekcije tega izraza.

Namesto frekvence kolekcije raje definirajmo frekvenco dokumentov (angl. *document frequency*), ki predstavlja število vseh dokumentov v kolekciji v katerih se pojavi izraz t in jo označimo z df_t. Frekvenca dokumentov se neposredno uporabi pri definiciji inverzne frekvence dokumentov (angl. *inverse document frequency*) izraza t [4]:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t},$$

kjer je N število vseh dokumentov v kolekciji. Vrednost idf_t bo velika za majhno število pojavitev izraza t in majhna za veliko število pojavitev.

Z združitvijo frekvence izraza $\mathrm{tf}_{t,d}$ in inverzne frekvence dokumentov idf_t definiramo utež izraza t v posameznem dokumentu d kot [4]:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{1}$$

Uporaba enačbe 0 povzroči, da je utež izraza t v dokumentu d:

- največja, ko se t pojavi mnogokrat v majhnem številu dokumentov,
- manjša, ko se t pojavi malokrat v dokumentu oz. se pojavi v velikem številu dokumentov,
- najmanjša, ko se t pojavi praktično v vseh dokumentih.

Vsak dokument v kolekciji lahko sedaj predstavimo kot vektor, kjer vsaka komponenta vektorja ustreza uteži izrazov v slovarju. Za izraze, ki ne nastopajo v dokumentu je utež enaka 0. Ker je tipično število izrazov v slovarju veliko večje od števila izrazov v posameznem dokumentu, je veliko komponent vektorjev enakih 0.

Vrednost oz. pomembnost dokumenta lahko za začetek opišemo kot seštevek uteži po vseh izrazih v poizvedbi q [4]:

$$\text{score}(q,d)=\sum_{t\in q}\text{tf-idf}_{t,d} \tag{2}$$

Z drugimi besedami: pomembnost dokumenta d je enaka vsoti uteži vseh izrazov poizvedbe, ki nastopajo v dokumentu. Z enačbo (2) torej določimo vsakemu dokumentu v kolekciji pomembnost glede na vhodno poizvedbo.

Podobno lahko tudi izračunamo podobnost (angl. *similarity*) med dokumenti oz. med poizvedbo in dokumenti v kolekciji. Označimo z $\vec{V}(d)$ vektor dokumenta d, kjer komponente vektorja sestavljajo posamezni izrazi v slovarju ali njihove uteži. Taka predstavitev dokumentov pomeni, da vektorski pristop vsebuje toliko dimenzij kolikor je različnih izrazov v slovarju. Vrstni red izrazov v dokumentu ni pomemben, kot smo poprej že prikazali s primerom. Standardna pot za izračun podobnosti med dokumenti je izračun kosinusne podobnosti (angl. *cosine similarity*) med vektorji kot [4]:

$$\text{sim}(d_1,d_2)=\frac{\vec{V}(d_1)\cdot\vec{V}(d_2)}{\left|\vec{V}(d_1)\right|\left|\vec{V}(d_2)\right|}=\frac{\sum_{i=1}^{n}v_i(d_1)v_i(d_2)}{\sqrt{\sum_{i=1}^{n}v_i^2(d_1)}\sqrt{\sum_{i=1}^{n}v_i^2(d_2)}}, \tag{3}$$

kjer $\vec{V}(d_i)=(v_1(d_i),\dots,v_n(d_i))$.

Kje pravzaprav ležijo prednosti izračuna podobnosti med dokumenti? Ena izmed prednosti je vsekakor ta, da nam tak način omogoča uporabo naprednih funkcionalnosti iskanja, kot npr. iskanja podobnih dokumentov – *more like this*, ki je v zadnjem času del praktično vsakega naprednejšega iskalnika. Največja prednost iskanja podobnosti po enačbi (3) pa je primerjava dokumentov v kolekciji z vhodno poizvedbo, ki jo predstavimo kot (zelo majhen) dokument oz. vektor. Glavna ideja na tem mestu je, da vsakemu dokumentu v kolekciji priredimo pomembnost, ki je enaka:

$$sim(d_i, q) = \frac{\vec{V}(d_i) \cdot \vec{V}(q)}{\left|\vec{V}(d_i)\right|\left|\vec{V}(q)\right|}, \tag{4}$$

kjer q predstavlja vektor poizvedbe, d_i pa vektor i-tega dokumenta v kolekciji. Dokumenti, ki se najbolj ujemajo s poizvedbo, so tudi najvišje ocenjeni (pomembni).

Preko enačbe (4) dosežemo dvoje:

- dokument se lahko zelo dobro ujema s poizvedbo četudi ne vsebuje vseh izrazov poizvedbe
- dokumenti, ki najbolj ustrezajo poizvedbi imajo večjo pomembnost oz. podobnost.

2.2 Apache Lucene

V tem delu bomo predstavili odprto-kodno rešitev Apache Lucene [5]. Pojasnili bomo kaj nam rešitev omogoča, kakšne so njene zmogljivosti, kako se jo uporablja ter strukturo indeksa, ki ga izgradi. Opisali bomo ključne komponente oz. razrede, ki jih uporablja za indeksiranje podatkov ter iskanje po indeksu, in pojasnili model za vrednotenje pomembnosti zadetkov. Uporabo bomo prikazali s pomočjo primerov, kjer bomo poljubno besedilo indeksirali s pomočjo nekaterih Lucene razredov (ter tako ustvariti indeks) in po tem indeksu tudi iskali.

2.2.1 Kaj je Lucene?

Lucene je javanska[3] knjižnica, ki omogoča visoko-zmogljivo in skalabilno izbiranje in iskanje informacij ter enostaven razvoj in implementacijo iskalnih možnosti v katerokoli aplikacijo. Razvoj Lucene se je pričel kot osebni projekt Douga Cuttinga[4], ki je leta 2000 na spletni strani SourceForge objavil prvo verzijo. Kmalu za tem, septembra 2001, je Lucene postala članica fundacije Apache Software in štiri leta kasneje ena izmed najpomembnejših odprto-kodnih projektov Apache. Enostavna implementacija in učinkovito delovanje Lucene sta ena izmed glavnih razlogov, da se Lucene uporablja na različnih spletnih straneh (npr. LinkedIn, Fedex, AOL, Instagram, Netflix, CNET, SourceForge, NASA Planetary Data System, Cisco, ipd.) in namiznih aplikacijah [5].

En izmed glavnih razlogov za veliko priljubljenost Lucene je tudi njena preprostost - tako po velikosti, kot po enostavni uporabi. Javanska arhivska knjižnica, ki vsebuje vse razrede, ki so potrebni za implementacijo je velika le 1 MB in ne zahteva nobenih drugih knjižnic za svoje delovanje. To tudi pomeni, da delovanje Lucene zasede malo prostora v Java kopici in zato lahko hitro in učinkovito deluje tudi v manj zmogljivih napravah (npr. mobilni telefoni, starejši

[3] Lucene je primarno pisana v programskem jeziku Java, vendar pa se vzporedno razvija tudi v drugih programskih jezikih, kot so C/C++, C# (.NET), Ruby, Perl, Python, PHP in ostali.

[4] Rešitev je poimenoval po svoji ženi.

računalniki, ...). Čeprav je Lucene preprosta za uporabo, pa zato njene zmožnosti pridobivanja in izbiranja informacij niso nič kaj preproste, temveč uporabljajo napredne mehanizme in modele iz področja pridobivanja informacij, o katerih bomo govorili kasneje. Ravno zato je razvojni vmesnik (API) grajen na način, da uporabniku omogoča uporabo teh mehanizmov brez podrobnega razumevanja kako Lucene pravzaprav v ozadju deluje. Videli bomo, da za implementacijo preprostega iskalnika uporabniku ni potrebno uporabiti več kot nekaj razredov v pravem vrstnem redu.

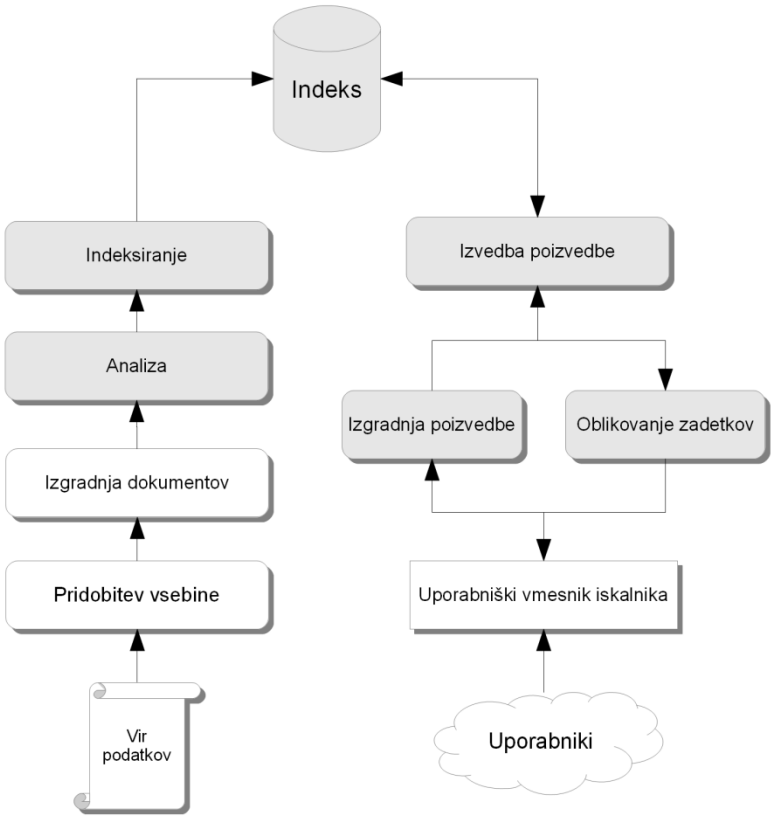
Pomembno je še enkrat opozoriti, da Lucene ni aplikacija kot npr. iskalnik datotek, spletni pajek ali spletni iskalnik, ki bi jo uporabnik namestil in pričel uporabljati, temveč le programska knjižnica, ki omogoča indeksiranje in iskanje podatkov, ne glede na njihov vir podatkov, obliko ali celo jezik. Dokler so podatki predstavljeni v obliki besedila in tako tudi podani v indeksiranje, za Lucene ni pomembno, ali so ti podatki pravzaprav spletne strani, lokalni dokumenti, tekstovne datoteke, Microsoft Office dokumenti, dokumenti PDF, ali celo video in zvočne datoteke (npr. meta podatki). Uporabniki lahko po tako indeksiranih podatkih iščejo na enak način kot so ga vajeni iz različnih spletnih iskalnikov, npr. `Apple -pie +Tiger` ali `naslov:policija AND kraj:ljubljana`, itd.

2.2.2 Glavne komponente iskalnika

Opozorili smo že, da Lucene ni iskalnik, temveč le programska knjižnica, zato si bomo v tem razdelku pogledali glavne komponente iskalnika in pojasnili katere komponente vsebuje Lucene in katere komponente mora uporabnik implementirati oz. razviti sam. Tipične komponente iskalnika prikazuje slika 2.1, kjer sivo obarvane komponente predstavljajo dele, ki jih izvaja Lucene, belo obarvane komponente pa mora uporabnik implementirati sam [5].

Iskalnik sestoji iz dveh delov: indeksiranja in iskanja. Indeksiranje in iskanje nista samostojni komponenti temveč verigi različnih procesov, ki se morajo izvršiti za uspešno delovanje. Da uporabniki lahko sploh izvajajo iskanje, se morajo podatki najprej indeksirati. Proces indeksiranja se prične tako, da se najprej pridobi vsebino poljubnega vira. Ta vsebina se nato pretvori v besedilo in dokument se indeksira. Ko je indeks enkrat izgrajen in uporabniki izvedejo iskanje po tem indeksu (preko uporabniškega vmesnika), se najprej izgradi poizvedba (tako, da jo Lucene razume), nato se ta poizvedba izvede nad indeksom, zadetki poizvedbe se ustrezno preoblikujejo in končno vrnejo uporabniku v pregled preko uporabniškega vmesnika.

V nadaljevanju bomo postopoma predstavili delovanje vsake izmed komponent na sliki 2.1, začenši s procesom indeksiranja, ki pomeni pretvorbo prvotnih podatkov v obliko, ki omogoča hitro iskanje.



Slika 2.1: Diagram tipičnih komponent iskalnika. Sivo obarvane komponente so del knjižnic Lucene, belo obarvane komponente mora uporabnik implementirati sam.

2.2.2.1 Komponente indeksiranja podatkov

Za primer vzemimo množico poljubnih tekstovnih datotek po katerih želimo iskati določeno besedo ali frazo. Iskanje bi sprva lahko izvedli tako, da bi postopoma odprli vsako izmed datotek in preverili ali beseda oz. fraza, ki jo iščemo obstaja v njej. Če le-ta obstaja, postavimo datoteko na seznam zadetkov, sicer nadaljujemo z naslednjo datoteko, dokler ne zaobjamemo celotne množice datotek. Tak postopek iskanja bi vsekakor deloval, vendar pa ima svoje pomanjkljivosti. Najočitnejša je ta, da bi bilo tako iskanje dolgotrajno, če bi bila množica datotek velika oz. če bi posamezne datoteke vsebovale velike količine besedila.

Da se tej pomanjkljivosti izognemo, besedilo datotek najprej indeksiramo in ga pretvorimo v obliko, ki nam omogoča hitro iskanje, brez nepotrebnega zaporednega preiskovanja datotek. Ta proces imenujemo indeksiranje, rezultat indeksiranja pa indeks o katerem bo govora več kasneje, kjer bomo predstavili fizično in logično strukturo indeksa, ki ga izgradi Lucene.

Indeks si lahko predstavljamo kot podatkovno strukturo, ki omogoča hitri naključni dostop do besed oz. podatkov, ki jih vsebuje. Analogni primer indeksu bi bilo indeksno kazalo, ki ga najdemo na koncu vsake knjige in omogoča hitro iskanje strani, ki vsebujejo določeno temo.

Kot smo že omenili je indeksiranje veriga procesov. Prvi proces v tej verigi predstavlja pridobitev vsebine.

Pridobitev vsebine

Prvi korak indeksiranja je pridobitev ustrezne vsebine iz virov podatkov, ki jih želimo indeksirati. Procesov, ki pridobijo vsebino virov je različno mnogo. Najbolj znani so spletni pajki oz. preiskovalniki, uporabimo pa lahko katerokoli metodo, ki nam ustreza, saj je proces pridobivanja vsebine virov v naši domeni in ni del Lucene. Pomembno je, da je proces pridobivanja vsebine učinkovit tudi, ko pridobiva vsebino, ki nastaja spotoma (npr. vsebina podatkovne baze, ki se neprestano dodaja – Twitter, ipd.). Tak proces mora implementirati metodo, ki učinkovito pridobi vsebino, ki je bila posodobljena od zadnjega indeksiranja. V primeru pridobivanja vsebin spletnih strani lahko spletni strežnik vrne le tiste strani, ki so bile posodobljene od nekega datuma dalje; v primeru relacijskih baz je ta parameter lahko unikatni ključ, ki se samodejno povečuje in ki ga uporabimo pri poizvedbi SQL, itd.

Obstaja številno odprto-kodnih preiskovalnikov, med njimi:

- Solr – sestrski projekt, ki nastaja vzporedno z razvojem Apache Lucene in omogoča pridobivanje vsebin relacijskih podatkovnih baz, datotek XML in različnih podatkovnih formatov (Microsoft Office dokumenti, dokumenti PDF, meta podatki slikovnih, video in zvočnih gradiv, ipd) z uporabo rešitve Apache Tika. Rešitev Apache Solr predstavljamo v naslednjem razdelku, saj je bil uporabljen pri razvoju internega iskalnika policije.
- Nutch – spletni pajek oz. preiskovalec, ki omogoča visoko-skalabilno in učinkovito pridobivanje vsebin spletnih strani.
- Aperture – preiskovalec, ki pridobiva vsebino spletnih strani, datotečnih sistemov, poštnih baz in omogoča indeksiranje in pridobivanje teksta.
- The Google Enterprise Connector Manager – ponuja različne povezovalnike (tudi za vire Lotus Notes) za številne nespletne vire.

V kasnejših poglavjih bomo tudi predstavili lastni proces pridobivanja vsebine iz zbirk Lotus Notes, ki smo jih tudi vzeli za primarni vir podatkov za indeksiranje.

Ko vsebino vira pridobimo, je naslednji korak izgradnja enot, ki jih imenujemo dokumenti (tj. dokumenti, ki jih prepozna Lucene, in *ne* dokumenti v klasičnem pomenu besede).

Izgradnja dokumentov

Dokumenti, ki jih izgradimo s tem procesom in ki se tudi shranijo v indeks, če tako zahtevamo, pogosto vsebujejo poimenovana polja s pripadajočo vsebino, npr. *naslov*, *avtor*, *datum*, *url*, ipd. Pomembno je, da natančno določimo vlogo posameznih polj in njihovo vsebino, saj bo od tega odvisno kako natančne zadetke bo iskalnik vračal uporabnikom. V primeru indeksiranja dokumentov PDF oz. spletnih strani je očitno, da bo en dokument v indeksu predstavljal en fizični dokument PDF ali spletno stran z ustreznimi polji. V drugih primerih, npr. zbirkah Lotus Notes, to ni takoj očitno, saj lahko dokumenti Lotus Notes poleg besedila vsebujejo tudi priponke. Vprašanje je, ali te priponke indeksirati skupaj z vsebino dokumenta Lotus Notes, ali jih indeksirati posebej in jih tudi prikazati v rezultatu poizvedbe kot ločene zadetke? V našem primeru smo ubrali prvo možnost in vsebino morebitnih priponk upoštevamo kot, da je del vsebine nosilnega dokumenta.

Posamezno polje, ki ga definiramo v (indeksnem oz. Lucene) dokumentu vsebuje *ime*, ki ga določa, *tekstovno* ali *binarno* vsebino, in vrsto podrobnih nastavitev, ki določajo kaj naj Lucene s tem poljem počne, ko dokument doda v indeks. Lucene lahko ob indeksiranju izvrši tri operacije nad poljem:

- *Vsebina polja je lahko indeksirana ali ne.* Polje mora biti označeno kot indeksirano, če želimo po njem tudi iskati. Iskati je možno le po tekstovnih poljih, binarna polja je mogoče le shraniti v indeks (npr. slike ali datoteke). Ko je polje indeksirano, se vsebina polja razdeli na žetone s pomočjo procesa, ki ga imenujemo *analiza*, in nato se ti žetoni (ki so lahko dalje preoblikovani preko raznih filtrov, ki jih Lucene omogoča) shranijo v indeks.

- *Če je vsebina polja indeksirana*, se opcijsko lahko shranijo tudi vektorji izrazov (angl. *term vectors*), ki predstavljajo nekakšen miniaturen invertirani indeks samo za to polje s katerim lahko pridobimo vse žetone polja. To omogoča napredne funkcije iskalnika kot je npr. iskanje dokumentov s podobno vsebino.

- *Kopija vsebine polja je lahko shranjena v indeks* v nespremenjeni obliki in ki jo kasneje lahko iz indeksa tudi preberemo. Na tem mestu posebej poudarimo, da se vsebina polja, ki jo indeksiramo, ne shrani v celoti v indeks, če na takem polju ni posebej določeno, da se mora vsebina (poleg vsebine, ki jo indeksiranje proizvede) shraniti. Z drugimi besedami: ko dokument, ki vsebuje taka polja preberemo iz indeksa, bodo v dokumentu prisotna le polja za katera je bilo določeno, da se njihova vsebina shrani.

Polje je v Lucene okolju predstavljeno z razredom `Field` (paket `org.apache.lucene.document`). Ta razred vsebuje posamezne dejansko vsebine, ki se indeksirajo. Polje ustvarimo tako, da ustvarimo primerek tega razreda, ki mu podamo želeno ime polja, vrednost polja, ter posamezne nastavitve, ki smo jih predstavili zgoraj. Kreiranje enega polja prikazuje spodnji

primer v javanski kodi, kjer ustvarimo polje z imenom »naslov«, ki vsebuje vrednost »Univerza v Ljubljani«, katera bo indeksirana (`Field.Index.ANALYZED`) in shranjena v indeks (`Field.Store.YES`).

```
Field polje = new Field("naslov",
                        "Univerza v Ljubljani",
                        Field.Store.YES,
                        Field.Index.ANALYZED
                       );
```

Kot smo že prikazali na primeru je določanje posameznih nastavitev polja enostavno. Polje sprejme tri vrste nastavitev:

1. Ali naj bo polje indeksirano, ter kako naj bo indeksirano?
2. Ali naj bo vsebina polja shranjena v indeks?
3. Ali naj se izgradi tudi vektor izrazov vrednosti polja?

– Prva možna nastavitev (`Field.Index.*`) določa kako bo po vrednosti polja mogoče iskati preko invertiranega indeksa. Možnosti za to nastavitev je pet:

- `Index.ANALYZED` – vsebina polja se preko analize razdeli na posamezne žetone in po posameznem žetonu je tudi kasneje možno iskati. Ta opcija nastavitve indeksiranja je najbolj primerna za tekstovna polja (npr. naslovi, povzetki, telesa besedila, ipd.)

- `Index.NOT_ANALYZED` – vsebina polje se indeksira, vendar pa se ne razdeli na žetone. Tako se vsebina upošteva kot en žeton po katerem je možno iskati. Ta opcija nastavitve je najbolj primerna za polja po katerih bi radi iskali, vendar pa bi razdelitev vsebine na žetone povzročilo neuspešne oz. nepravilne zadetke. Polja, ki jim nastavimo to opcijo vsebujejo spletne naslove, poti do datotek na datotečnem sistemu, datume, unikatne ključe, EMŠO, ipd...

- `Index.ANALYZED_NO_NORMS` – podobna nastavitev kot `Index.ANALYZED`, vendar pa ne shrani informacij o normah v indeks. Norme shranjujejo informacijo o pomembnosti posameznih polj o kateri govorimo v naslednjih odstavkih. Uporaba norm zahteva večje pomnilniške zmogljivosti sistema pri iskanju, zato se ta možnost uporabi pri iskalnikih, ki so vdelani v manj zmogljive naprave.

- `Index.NOT_ANALYZED_NO_NORMS` – podobna nastavitev `kot Index.NOT_ANALYZED`, vendar pa ne shrani informacij o normah v indeks. Ta opcija se pogosto uporablja v primeru, ko želimo zmanjšati velikost indeksa in porabo pomnilnika pri iskanju.

- `Index.NO` – vsebine polja ne bo mogoče iskati.

– Druga nastavitev (`Field.Store.*`) določa ali se bo dejanska vsebina polja shranila v indeks in katero bomo ob iskanju lahko pridobili:

- `Store.YES` – Shrani dejansko (nespremenjeno) vrednost polja v indeks. Ta možnost se uporablja za shranjevanje vsebine polj, ki jih želimo prikazati na strani z zadetki (npr. naslovi, spletne povezave, ključi na zapise v primarnih virih podatkov, ipd.) in se jo praviloma uporablja za polja, ki ne vsebujejo velike količine teksta, saj shranjevanje vrednosti polja posledično pomeni tudi večjo zasedenost prostora za shranjevanje, ki ga zavzema indeks.

- `Store.NO` – Ne shrani vrednosti polja v indeks. Ta možnost se pogosto uporablja skupaj z možnostjo `Index.ANALYZED`. Vsebina polja se indeksira in je po njej možno iskati, vendar pa je ni možno v nespremenjeni obliki pridobiti iz indeksa, temveč le iz primarnega vira (npr. besedila spletnih strani, dokumenti, ipd.).

– Tretja nastavitev (`TermVector.*`) spada med naprednejše nastavitve in določa ali naj se za posamezen dokument izgradi tudi vektor izrazov vrednosti polja. Kaj točno to pomeni? V določenih primerih bi pri iskanju dokumentov radi pridobili seznam vseh unikatnih izrazov, ki nastopajo v poljih dokumenta za potrebe dodatnih funkcionalnosti iskalnika, npr. označevanje najdenih izrazov, možnost iskanja podobnih dokumentov, kategorizacija dokumentov, ... V ta namen uporabimo vektorje izrazov, ki so mešanica indeksiranih in shranjenih polj. Vektorji izrazov vsebujejo posamezne izraze, ki jih proizvede analiza polja in omogočajo, da pridobimo seznam vseh izrazov posameznih polj in frekvenco pojavitev teh izrazov znotraj dokumenta. Pri nastavitvah polja je možno določiti, da se shrani položaj in odmik posameznih izrazov polja v indeks.

Spodnji seznam prikazuje pet možnosti, ki jih lahko določimo za to nastavitev:

- `TermVector.YES` – Shrani seznam unikatnih izrazov polja v vektor izrazov dokumenta, vendar pa ne shrani njihovih položajev in odmikov.

- `TermVector.WITH_POSITIONS` – Shrani seznam izrazov polja ter njihove položaje, vendar brez odmikov.

- `TermVector.WITH_OFFSETS` – Shrani seznam izrazov polja ter njihov odmik od začetka polja (položaj začetnega in končnega znaka izraza), vendar brez položajev.

- `TermVector.WITH_POSITIONS_OFFSETS` – Shrani seznam izrazov polja ter njihove položaje in odmike.

- `TermVector.NO` – Ne shrani informacije o izrazih v indeks (če možnosti o shranjevanju vektorja izrazov ne podamo v konstruktor razreda `Field` – kot v primeru na strani 19 – se privzame ta možnost). Če konstruktorju razreda `Field` podamo možnost `Index.NO`, lahko podamo samo `TermVector.NO`.

Tabela 2.1 prikazuje primere uporabe zgornjih nastavitev za posamezne tipe (vrednosti) polj.

Omenimo še možnost povečanja oz. zmanjšanja pomembnosti (angl. *boost*) posameznih polj oz. dokumentov ob izgradnji ali ob iskanju. Lucene privzeto poveča pomembnost poljem, ki nosijo krajšo vsebino kot daljšo. Z drugimi besedami: če poizvedba sestoji iz besede, ki je bila najdena v dokumentu z veliko vsebine, ima tak dokument manjšo pomembnost kot pa dokument, kjer je bila ista beseda najdena v dokumentu, ki vsebuje le nekaj besed.

Pomembnost poljem in dokumentov pa lahko preprosto določimo tudi sami. Tako bi npr. polju, ki vsebuje besedilo naslova lahko pripisali večjo pomembnost, kot pa polju, ki vsebuje življenjepis avtorja. Če bi poizvedba uporabnika vrnila tako en kot drugi dokument in bi prvi zadetek ustrezal poizvedbi po naslovu, drugi pa po življenjepisu, bi ob taki nastavitvi prvi dokument imel večjo pomembnost od drugega. O teoretičnih modelih, ki se uporabljajo za določitev pomembnosti smo že govorili, v nadaljevanju pa bomo tudi predstavili model, ki ga uporablja Lucene za določitev pomembnosti dokumentom.

Pomembnost lahko določamo tako poljem kot dokumentom. Privzeto polja in dokumenti nimajo določene pomembnosti oz. imajo pomembnost nastavljeno na faktor 1.0. Pomembnost ročno določimo preko metode `setBoost(float)`, ki je del vmesnika API in se jo uporablja enako za določitev pomembnosti poljem kot dokumentom. Spodnji primer prikazuje kako dokumentu povečamo pomembnost:

```
// Kreiramo objekt, ki predstavlja Lucene dokument
Document doc = new Document();

// ... V dokument vstavimo poljubno število polj ...
doc.add(new Field(...));

// Dokumentu povečamo pomembnost za 30%
doc.setBoost(1.3F);
```

Pri iskanju bo Lucene povečala oz. zmanjšala pomembnost dokumenta glede na ročno nastavitev, ki smo jo določili. Če določimo le pomembnost dokumenta, Lucene v ozadju postavi pomembnost poljem, ki smo jih dodali na dokument, na enako vrednost kot jo ima dokument. Če želimo, da imajo znotraj dokumenta, določena polja večjo veljavo kot ostala, lahko poleg določitve pomembnosti dokumenta ročno določimo tudi pomembnost ostalih polj, kot prikazuje naslednji primer:

```
// Kreiramo objekt, ki predstavlja Lucene dokument
Document doc = new Document();

// V dokument vstavimo dve poljubni polji
Field naslov = new Field("naslov", "Univerza v Ljubljani", Field.Store.YES,
                         Field.Index.ANALYZED));
doc.add(naslov);
doc.add(new Field("kraj", "Ljubljana", Field.Store.YES, Field.Index.ANALYZED));

// Polju 'naslov' povečamo pomembnost za 50%.
// Polje 'kraj' in dokument obdržita privzeto vrednost 1.0.
naslov.setBoost(1.5F);
```

Na tem mestu je pomembno opozoriti, da četudi dokumentom oz. poljem povečamo pomembnost za nek faktor, to ne pomeni, da bodo v seznamu zadetkov le-ti na vrhu. Lucene za določitev pomembnosti posameznih dokumentov uporablja modificiran vektorsko-prostorski model, ki sicer upošteva ročne nastavitve pomembnosti, vendar pa je potrebno za dobre rezultate poizvedb nekaj eksperimentiranja z ročnimi nastavitvami pomembnosti, saj imajo krajša polja privzeto višjo pomembnost kot daljša.

Index.*	**Store.***	**TermVector.***	**Primer vsebine polja**
`ANALYZED`	`YES`	`WITH_POSITIONS_OFFSETS`	Naslov dokumenta, povzetek
`ANALYZED`	`NO`	`WITH_POSITIONS_OFFSETS`	Vsebina
`NOT_ANALYZED`	`NO`	`NO`	Skrita vsebina oz. polja
`NOT_ANALYZED_NO_NORMS`	`YES`	`NO`	Imena datotek, primarni ključi, telefonske številke, EMŠO, spletni naslovi, osebna imena, datumi, polja za namen ureditve vrstnega reda
`NO`	`YES`	`NO`	Tip dokumenta, primarni ključi (če se jih uporablja pri iskanju)

Tabela 2.1: Nekaj primerov uporabe nastavitev `Field.Index.*`, `Field.Store.*` in `TermVector.*` glede na vrednosti polj.

Analiza dokumenta

Ko je dokument pripravljen za indeksiranje (tj. vsebuje polja, ustrezne nastavitve, ...) se ga pošlje v analizo. Tam se vsebina polj na dokumentu razdeli na žetone, ki približno ustrezajo posameznih besedam vsebine. Zakaj približno? Pomembno se je namreč zavedati, da je lahko

vsebina dokumenta ali polja v kateremkoli jeziku, morda opisuje matematično enačbo v kodi LaTeX, ali pa vsebuje pravopisne napake. Lucene vsebuje veliko število filtrov (angl. *filters*), ki spreminjajo vsebino posameznih žetonov. Filter `LowerCaseFilter` – na primer – spremeni vse črke v žetonu v male in s tem povzroči, da je iskanje neodvisno od velikosti črk. Podobno filter `StopFilter` povzroči, da se ne indeksirajo žetoni (t.i. *stop words*), ki ustrezajo besedam na določenem seznamu; npr. besede, ki se pogosto pojavljajo v besedilu (*in*, *da*, *ter*, *ko*, *da*, ...).

Analizator sestavlja množica vhodnih žetonov in veriga filtrov, ki delujejo na posamezne žetone v množici. Lucene tudi omogoča razvoj lastnih analizatorjev in filtrov, tako, da jih uporabniki lahko prilagodijo za delovanje na različnih jezikih in oblikah. V zgornjem primeru smo govorili samo o filtrih, Lucene pa vsebuje štiri preproste analizatorje (ki so žal najbolje prilagojeni za delo na angleškem jeziku):

- `WhitespaceAnalyzer` razdeli vsebino polja na žetone tam, kjer se pojavi presledek. Žetoni nadalje niso nič spremenjeni.

- `SimpleAnalyzer` razdeli žetone na mestih, kjer nastopajo ne črkovni znaki (npr. &, @, ...), spremeni črke žetonov v male in zavrže vse numerične znake.

- `StopAnaylzer` deluje enako kot `SimpleAnalyzer` s to razliko, da odstrani besede, ki jih posredujemo preko seznama.

- `StandardAnalyzer` je analizator, ki se ga najpogosteje uporablja in omogoča napredno prepoznavo vsebine žetonov (npr. elektronski naslovi, imena krajev, ...). Črke žetonov spremeni v male, odstrani besede iz seznama in vsa ločila.

Za prikaz delovanja zgornjih štirih analizatorjev vzemimo analizo besedila:

XY&Z Ministrstvo - info@policija.si

Rezultat analize besedila prikazuje Tabela 2.2. Rezultat na primeru analizatorja `StopAnalyzer` je enak kot pri primeru analizatorja `SimpleAnalyzer` zato, ker nismo podali seznama besed, ki bi jih radi odstranili. Če bi npr. podali seznam besed »policija« in »info« bi analizator proizvedel množico žetonov [xy], [z], [ministrstvo], [si].

Kasneje si bomo v poglavju, kjer obravnavamo odprto-kodno rešitev Apache Solr tudi podrobneje pogledali kako se analizatorje in pripadajoče filtre definira in njihovo delovanje na primerih.

Analizator	Žetoni
WhitespaceAnalyzer	[XY&Z], [Ministrstvo], [-], [info@policija.si]
SimpleAnalyzer	[xy], [z], [ministrstvo], [info], [policija], [si]
StopAnalyzer	[xy], [z], [ministrstvo], [info], [policija], [si]
StandardAnalyzer	[xy&z], [ministrstvo], [info@policija.si]

Tabela 2.2: Nekaj primerov uporabe analizatorjev na besedilu »XY&Z Ministrstvo - info@policija.si«.

Dodajanje dokumenta v indeks

Ko je vhodni dokument analiziran, se ga doda v indeks. Lucene za shranjevanje podatkov v indeks uporablja invertirani indeks, ki smo ga že predstavili. Dokument se v indeks doda preko razreda `IndexWriter` s klicem metode `addDocument(Document, [Analyzer])`, kjer prvi vhodni parameter predstavlja (analiziran) dokument, drugi vhodni parameter pa morebitni analizator, s katerim želimo delovati na dokument. Na tem mestu je pomembno poudariti, da v primeru uporabe drugega analizatorja, moramo enak analizator uporabiti tudi pri iskanju, saj sicer lahko pride do napačnih zadetkov.

Indeks, ki ga ustvari Lucene ni ena sama datoteka, temveč je skupek več različnih datotek, katerih zapisi so med seboj povezani. V podrobnosti sestave indeksa se ne bomo spuščali, omenimo le nekaj pomembnejših dejstev:

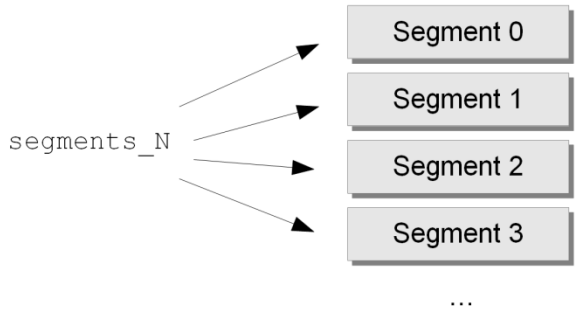
- Indeks sestoji iz enega ali več segmentov (Slika 2.2), kjer je vsak segment samostojni indeks, ki vsebuje neko podmnožico vseh indeksiranih dokumentov (logično sestavo indeksa predstavlja Slika 2.3).

- Novi segment se ustvari, ko `IndexWriter` izravna medpomnilnik z dodajanjem oz. brisanjem dokumentov.

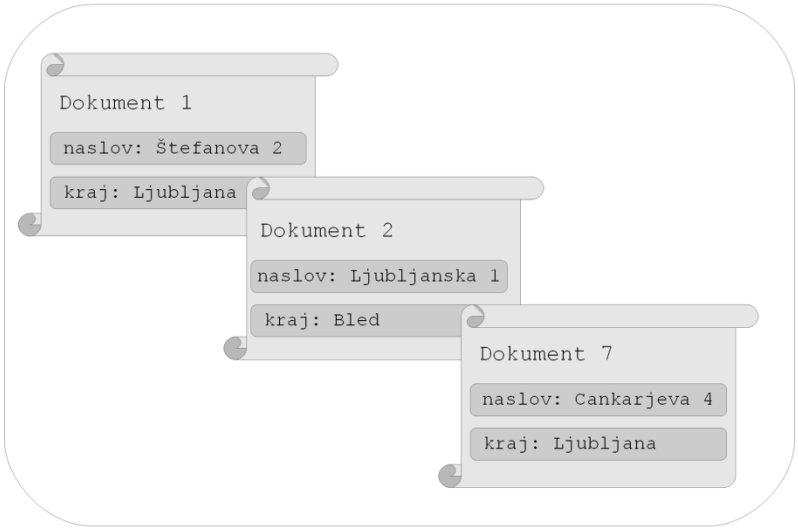
- Ob iskanju se preišče vsak segment posebej in seznami posameznih zadetkov se združijo v skupen seznam.

- Vsak segment sestavlja več datotek v obliki `_X.<končnica>`, kjer `X` označuje ime segmenta in `<končnica>` kateri del indeksa fizična datoteka predstavlja. Posamezni deli indeksa (vektorji izrazov, shranjena polja, invertirani indeks, ipd.) se namreč shranijo v ločene datoteke. Datoteke je tudi možno združiti v skupno datoteko oblike `_X.cfs` na račun počasnejšega indeksiranja in iskanja (v večini primerov razlika ni omembe vredna) z ukazom `IndexWriter.setUseCompoundFile(true)`.

- Segmente »združuje« datoteka `segments_<N>`, ki povezuje vse aktivne segmente. Lucene ob iskanju najprej odpre to datoteko in nato vsak segment, ki ga ta datoteko povezuje. Vrednost `<N>`, imenovana *generacija*, je celoštevilska vrednost, ki se poveča vsakič, ko spremembe (dodane in/ali brisane dokumente, ipd.) zapišemo v indeks z ukazom `IndexWriter.commit()`.



Slika 2.2: Segmentna struktura invertiranega indeksa. Datoteka `segments_N` povezuje vse aktivne segmente (segment 0, segment 1, ...). Posamezni segment je samostojni indeks, ki vsebuje neko podmnožico vseh indeksiranih dokumentov. Pri iskanju se preišče vsak posamezen segment in seznami posameznih zadetkov se nato združijo v skupen seznam zadetkov, ki se vrne uporabniku [5].



Slika 2.3: Logična sestava indeksa oz. posameznega segmenta. Indeks vsebuje dokumente, kateri vsebuje posamezna polja z vsebino. Polje je sestavljeno iz imena in pripadajoče vsebine (npr: kraj:Ljubljana, kjer je »kraj« ime polja, »Ljubljana« pa vsebina polja). Po posameznih poljih je

možno tudi iskati, če v poizvedbi navedemo ime polja ter želeni iskalni niz. Ime polja in iskalni niz ločimo z dvopičjem.

2.2.2.2 Komponente iskanja po podatkih in indeksu

Ko so podatki indeksirani lahko po njih iščemo. Sedaj smo se pomaknili na desno stran Slika 2.1, kjer bomo predstavili komponente, ki predstavljajo izvajanje iskanja po indeksiranih podatkih.

V prvi vrsti imamo uporabnike (lahko tudi avtomatske sisteme), ki izvajajo poljubne poizvedbe. Poizvedbe uporabniki vnesejo v uporabniški vmesnik (spletni, programski, telefonski, ...), kjer se le-te lahko preuredijo, da ustrezajo standardu iskalnega sistema in nato pošljejo v komponento oz. proces izvedbe poizvedbe. Takrat se pridobijo vsi dokumenti (ali podmnožica teh), ki ustrezajo poizvedbi in se preko komponente oblikovanja zadetkov (npr. za primeren prikaz uporabniku) vrnejo nazaj na uporabniški vmesnik.

Lucene ne vsebuje uporabniškega vmesnika za iskanje, zato je izgradnja le-tega v domeni uporabnika. Vgradnja preprostega iskalnika je z uporabo Lucene programskih knjižnic preprosta, saj je potrebnih – kot bomo prikazali na primerih – le nekaj vrstic programske kode. Poizvedbo lahko kreiramo programsko (tj. v obliki izrazov s pomočjo razreda `TermQuery`) ali pa jo ustvarimo z uporabo razreda `QueryParser`, kjer se prosti vhodni tekst (npr. `Apple -pie +Tiger`) ustrezno transformira.

Iskanje po Lucene indeksu se izvaja preko razreda `IndexSearcher` s pomočjo metode `search(Query)`, katere vhodni razred `Query` predstavlja poizvedbo, ki jo želimo izvesti.

Spodnji primer prikazuje izvedbo preproste poizvedbe s katero želimo pridobiti vse dokumente iz indeksa, ki vsebujejo polje z imenom »kraj« in katerega vsebina je »ljubljana«.

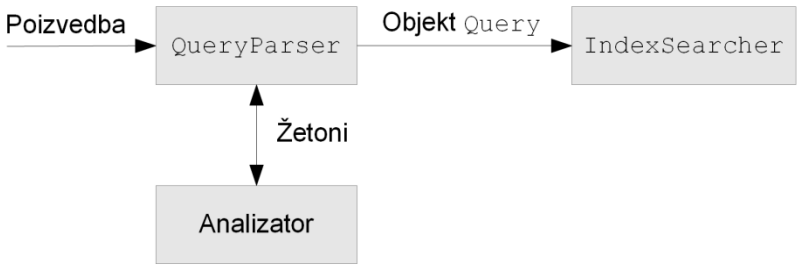
```
// Pridobimo primerek razreda IndexSearcher, preko
// katere izvajajmo poizvedbe nad indeksom, ki se
// nahaja v mapi oz. lokaciji, ki jo podamo kot
// vhodni parameter konstruktorja
IndexSearcher searcher = new IndexSearcher(...);

// Kreiramo primerek razreda Term, ki predstavlja
// izraz, ki ga iščemo
Term term = new Term("kraj", "ljubljana");

// Kreiramo primerek razreda Query, ki je interna
// predstavitev poizvedbe, ki jo Lucene razume
Query query = new TermQuery(term);

// Vrnemo seznam zadetkov (največ maxreturned)
TopDocs docs = searcher.search(query, maxreturned);
```

Opozorimo na neko pomembno dejstvo: pri iskanju s `TermQuery` nikjer ne navedemo analizatorja, ki je bil uporabljen pri indeksiranju. V ta namen smo iskani niz (tj. »ljubljana«) namenoma napisali z malo začetnico, saj smo privzeli, da smo ob indeksiranju uporabili tak analizator (npr. `SimpleAnalyzer`), ki vsebino posameznih žetonov spremeni v male črke. Če bi na primeru uporabili iskalni niz »Ljubljana«, bi bil rezultat iskanja prazen seznam zadetkov. Kritično je torej, da izraze, ki jih vnesemo v poizvedbo, vnesemo na isti način kot so predstavljeni v indeksu (tj. male oz. velike črke, krajšanje besed, ipd.).

Zgornji primer je prikazal zelo enostavno izvedbo poizvedbe, ki pa je uporabniki načeloma ne izvajajo več v taki obliki, saj sodobne rešitve omogočajo uporabniku bolj prijazen vnos poizvedbe in uporabo logičnih operatorjev (`AND`, `OR`, `NOT`, `+`, `-`, ipd...). Vseeno bi pa radi obdržali funkcionalnost iskanja po vsebini posameznih polj ter imeli omogočeno iskanje po datumskih intervalih, delnih izrazih, določenih nizih, ...

Razred `QueryParser` omogoča, da uporabniki vpišejo poljubno poizvedbo, `QueryParser` pa nato to poizvedbo prevede oz. razčleni v obliko (primerek razreda `Query`), ki jo Lucene razume. Proces prevedbe poizvedbe iz objekta `QueryParser` v objekt `Query` prikazuje Slika 2.4. Tabela 2.3 pa prikazuje nekaj primerov poizvedb, ki jih `QueryParser` prepozna.

Za razliko od iskanja s `TermQuery`, `QueryParser` vzame kot vhodni parameter analizator s katerim želimo delovati na poizvedbo. Priporoča se, da je vhodni analizator isti kot je bil uporabljen pri indeksiranju, saj bo le na tak način zagotovljeno pravilno (pravično) iskanje. Omenimo, da Lucene ne postavlja omejitev glede uporabe različnih analizatorjev tako pri indeksiranju kot pri iskanju. Poljubno število dokumentov se lahko indeksira z enim analizatorjem, drugo število pa z drugim. Lucene bo vsebino teh dokumentov prevedla z izbranim analizatorjem in jo ustrezno indeksirala. Enako lahko pri iskanju uporabimo poljuben analizator, vendar pa zaradi različnih prevedb posameznih indeksiranih besed (oz. žetonov) lahko pride do zgrešenih oz. napačnih zadetkov.



Slika 2.4: Postopek prevedbe uporabnikove poizvedbe v obliko, ki jo razume Lucene. `QueryParser` poizvedbo analizira z izbranim analizatorjem in nato iz prevedene poizvedbe generira objekt `Query` na katerega deluje `IndexSearcher` [5].

Primer poizvedbe	**Kateri dokumenti ustrezajo poizvedbi?**
`policija`	...ki vsebujejo izraz *policija* v privzetem polju

`policija ljubljana`
 	
`policija OR ljubljana`
	...ki vsebujejo izraz *policija* ali *ljubljana*, ali oba v privzetem polju[5]

`+policija +ljubljana`
 	
`policija AND ljubljana`
	...ki vsebujejo oba izraza *policija* in *ljubljana* v privzetem polju

`kraj:ljubljana`	...ki vsebujejo izraz *ljubljana* v polju »kraj«

`kraj:ljubljana -naslov:stefanova`
 	
`kraj:ljubljana AND NOT naslov:stefanova`
	...ki vsebujejo izraz *ljubljana* v polju »kraj« in ne vsebujejo izraza *stefanova* v polju »naslov«

`(policija OR policist) AND ljubljana`	...ki vsebujejo izraz *ljubljana* in morajo vsebovati tudi vsaj enega izmed izrazov *policija* in *policist*, vse v privzetem polju
`naslov:'univerza v ljubljani'`	...ki vsebujejo niz (ne izraze) 'univerza v ljubljani' v polju »naslov«
`polic*`	...ki vsebujejo izraze, ki se pričnejo z *polic*, npr: *policija*, *polica*, *policist*, ipd., ter seveda izraz *polic* sam
`datum:[1/1/12 TO 31/12/12]`	...katerih polje »naslov« vsebuje datumski zapis med 1. jan. 2012 in 31. dec. 2012.

Tabela 2.3: Nekaj primerov preprostih poizvedb, ki jih lahko uporabimo pri iskanju s pomočjo razreda `QueryParser`, ki uporabnikovo poizvedbo prevede v obliko, ki jo razume Lucene.

Najbolje, da to ponazorimo s kratkim primerom. Vzemimo, da indeksiramo besedo »Ljubljana«, enkrat z analizatorjem `WhitespaceAnalyzer` drugič z analizatorjem `SimpleAnalyzer`. Bistvena razlika med obema je ta, da prvi ne spremeni velike začetnice v malo, drugi pa jo. Če bi sedaj indeksirali dva dokumenta z isto vsebino, npr. »Ljubljana«, vsakega s svojim analizatorjem, bi v indeksu imeli prvi dokument s poljem `polje:Ljubljana`, in drugi dokument s poljem `polje:ljubljana`. Pri iskanju uporabimo `QueryParser`, kjer prvič kot analizator podamo `WhitespaceAnalyzer`, drugič pa `SimpleAnalyzer` ter poizvedbo `Ljubljana` (pozor: z veliko začetnico). Pričakovali bi, da bo poizvedba vrnila dva dokumenta, v resnici pa v obeh primerih vrne le enega. V prvem primeru vrne prvi dokument (zadetek »Ljubljana«), v drugem pa drugi dokument (zadetek »ljubljana«). Izbira analizatorja, tako pri

[5] Privzeti operator je `OR`. Tega se da spremeniti preko metode `QueryParser.setDefaultOperator(...)`.

procesu indeksiranja, kot pri procesu iskanja, je torej pomembna in od te izbire je odvisna uspešnost iskanja.

Povrnimo se nazaj k uporabi razreda `QueryParser` in razložimo vhodne parametre konstruktorja tega razreda. Primerek razreda `QueryParser` ustvarimo tako, da mu podamo tri obvezne vhodne parametre: oznako različice Lucene, ime privzetega polja in analizator.

– Oznaka različice Lucene (npr. `LUCENE_29` ali `LUCENE_30`) označuje do katere različice nazaj naj `QueryParser` upošteva nastavitve in delovanje (v nekaterih primerih »simulira« tudi napačno delovanje). Ta parameter je pomemben iz vidika sestave indeksa, saj ni nujno, da vsaka nova različica Lucene podpira zapis indeksa, ki ga je ustvarila prejšnja različica. Če torej želimo z novejšo različico Lucene (npr. 3.6) brati indeks, ki ga je ustvarila neka prejšnja različica Lucene (npr. 3.0) moramo kot različico Lucene v konstruktorju podati vrednost `Version.LUCENE_30`.

– Ime privzetega polja določa v katerem polju se bo iskani niz iskal, če v poizvedbi eksplicitno ne določimo polja. Zaradi lažjega izvajanja poizvedb (uporabnikom tako ni potrebno poznati imen polj, ki so zapisana v indeks) se vsebino vseh polj pri indeksiranju združi v skupno polje in ime tega polja se nato poda kot vhodni parameter konstruktorja razreda `QueryParser`.

– Tretji parameter konstruktorja predstavlja izbrani analizator, ki bo deloval na poizvedbo, predno se le-ta prevede v objekt `Query`. Analizatorjev je več na voljo, možno pa je razviti tudi lastne analizatorje.

Spodnji primer prikazuje izvedbo preproste poizvedbe (`policija AND kraj:Ljubljana`), s katero želimo pridobiti dokumente, ki vsebujejo besedo *policija* v privzetem polju in besedo *ljubljana* v polju z imenom »kraj«. Privzeli bomo, da smo ob indeksiranju teh dokumentov uporabili analizator `StandardAnalyzer`, zato ga bomo uporabili tudi pri iskanju. V poizvedbi smo namenoma zapisali *Ljubljana* z veliko začetnico, saj bo analizator to začetnico spremenil v malo.

```
// Pridobimo primerek razreda IndexSearcher, preko
// katere izvajajmo poizvedbe nad indeksom, ki se
// nahaja v določeni mapi
IndexSearcher searcher = new IndexSearcher(...);

// Kreiramo primerek razreda QueryParser, kjer mu
// podamo ime privzetega polja in analizator
QueryParser parser = new QueryParser(Version.LUCENE_30,
                                     "privzetopolje",
                                     new StandardAnalyzer()
                                     );

// Kreiramo primerek razreda Query, ki je interna
// predstavitev poizvedbe, ki jo Lucene razume
Query query = parser.parse("policija AND kraj:Ljubljana");

// Vrnemo seznam zadetkov (največ maxreturned)
TopDocs docs = searcher.search(query, maxreturned);
```

Na kratko omenimo še objekt `TopDocs`, ki smo ga prikazali v nekaterih primerih, vendar z njim nismo nič počeli. Večina metod razreda `IndexSearcher` vrne objekt `TopDocs`, ki predstavlja množico zadetkov (tj. dokumentov, ki so ustrezali poizvedbi). Ta razred vsebuje tri relevatne metode:

- metodo `totalHits`, ki vrne število vseh dokumentov, ki so ustrezali poizvedbi.
- metodo `scoreDocs`, ki vrne seznam objektov `ScoreDoc`, ki predstavljajo posamezni zadetek.
- metodo `getMaxScore()`, ki vrne pomembnost (angl. *score*) najvišjega zadetka.

Objekt `TopDocs` torej v resnici ne vsebuje posameznih zadetkov, temveč jih posredno naslavlja preko objekta `ScoreDoc`. Ker `ScoreDoc` vsebuje unikatno oznako dokumenta, lahko ta dokument pridobimo iz indeksa preko klica `IndexSearcher.document(id)`.

2.2.2.3 Kako Lucene določi pomembnost dokumentom?

Sedaj, ko smo spoznali oba teoretična modela za pridobivanje in ocenjevanje dokumentov iz indeksa (poglavje 2.1.2), ter osnovne gradnike iskalnika je čas, da predstavimo kako Lucene določi pomembnost dokumentom.

Lucene ob iskanju najprej uporabi Boolov model za pridobivanje informacij, s katerim iz indeksa pridobi vse relevantne dokumente, ki ustrezajo poizvedbi (poglavje 2.1.1). Nato uporabi vektorsko-prostorski model s katerim določi pomembnost vsakemu zadetku. Množica vrnjenih zadetkov je pogosto urejena po pomembnosti v padajočem vrstnem redu.

Lucene uporablja sledečo formulo za izračun pomembnosti [5]:

$$\text{score}(q, d_i) = \text{coord}_{q,d_i} \cdot \text{norm}_q \cdot \sum_{t \in q} \left[\text{tf}_{t,d_i} \cdot \text{idf}_t^2 \cdot \text{boost}_t \cdot \text{norm}_{t,d_i} \right] \tag{5}$$

kjer d_i predstavlja vektor i-tega dokumenta iz množice vrnjenih zadetkov, ki jih določi Boolov model, q vektor poizvedbe in t posamezni izraz v poizvedbi.

Formula (5) kot rezultat vrne realno število, ki je ≥ 0. Večji kot je rezultat, večja je podobnost med q in d_i. Nekateri členi enačbe so nam znani že od začetka tega poglavja, ostale pa moramo še razložiti.

Ker Lucene omogoča, da dokumentu in njegovim poljem ročno povečamo oz. zmanjšamo pomembnost, formula (5) to tudi upošteva preko člena boost_t, ki predstavlja faktor povečanja oz. zmanjšanja pomembnosti polja v katerem je bil najden izraz t. Pojasnili smo že, da imajo privzeto vsa polja ob indeksiranju to vrednost postavljeno na 1, zato za privzete vrednosti ta člen ne vpliva na izračun pomembnosti dokumenta. Če ob indeksiranju dokumentu nastavimo faktor povečave pomembnosti, se na to vrednost postavijo tudi faktorji povečave vseh polj, ki jih dodamo v dokument. Ko še dodatno nekaterim poljem v dokumentu nastavimo faktor

povečave pomembnosti, se ta faktor množi s faktorjem povečave pomembnosti, ki smo jo nastavili dokumentu. Drugače razloženo: če dokumentu nastavimo faktor povečave 1.5, vsa polja, ki jih dodamo (oz. so že dodana) na dokument privzamejo tak faktor povečave pomembnosti. Če polja na i-tem dokumentu označimo z F_j, kjer $1 \leq j \leq n$ in n število vseh polj na dokumentu, sledi:

$$\mathrm{boost}_{d_i} = 1.5$$

$$\mathrm{boost}_{F_j} = \mathrm{boost}_{d_i} \times \mathrm{boost}_{F_j} = 1.5 \times 1 = 1.5 \tag{6}$$

Nadalje lahko samo nekaterim poljem spremenimo faktor povečave pomembnosti. Izberimo si polji F_2 in F_4, ki jima nastavimo faktor na 2. Iz enačbe (6) sledi:

$$\mathrm{boost}_{F_2} = \mathrm{boost}_{d_i} \times \mathrm{boost}_{F_2} = 1.5$$

Podobno sledi še za polje F_4. Poljem, katerim dodatno nastavimo faktor pomembnosti, prispeva tudi faktor pomembnosti dokumenta.

Pomembnost dokumenta se neposredno poveča tudi preko velikost posameznih polj, ki jih formula (5) upošteva. Pojasnili smo že, da polja, ki so krajša, prispevajo več k pomembnosti dokumenta, kot pa polja, ki so daljša (seveda se pri izračunu upoštevajo le polja v katerih so bili najdeni izrazi poizvedbe). Ta prispevek je del člena $\mathrm{norm}_{t,d}$.

Člen $\mathrm{coord}_{q,d}$ enako neposredno poveča pomembnost dokumentom pri katerih je bilo najdenih več izrazov poizvedbe kot pa pri tistih v katerih je bilo najdenih manj izrazov.

Člen norm_q predstavlja normalizacijo poizvedbe in je enak $\sqrt{(\sum_{t \in q} w_t^2)}$, kjer je w_t utež posameznega izraza t poizvedbe [5].

Lucene lahko izračun pomembnosti posameznega dokumenta v množici zadetkov tudi »razloži« preko metode `IndexSearcher.explain(q, d)`, kjer je `q` objekt `Query` in predstavlja poizvedbo, `d` pa objekt `Document`.

2.3 Apache Solr

Apache Solr je odprto-kodna spletna platforma namenjena podjetjem in zasebnikom za izvajanje indeksiranja in iskanja po podatkih, ki v ozadju uporablja Apache Lucene. Za razliko od Lucene je Solr rešitev, ki uporabnikom omogoča izvajanje indeksiranja in iskanja po podatkih praktično brez dodatnega programskega razvoja. Solr vključuje raznovrstne funkcije, kot so [10]:

- Razširjeno iskanje (angl. *full-text search*)
- Označevanje zadetkov
- Izvajanje strukturiranih in tekstovnih poizvedb
- Komunikacija preko vmesnika API ali protokola HTTP v različnih formatih (XML, JSON, CSV)
- Upravljanje s sistemom preko spletnega vmesnika
- Replikacija indeksov med različnimi strežniki Solr
- Porazdeljeno iskanje
- Indeksiranje relacijskih podatkovnih baz
- Možnost prepoznave in indeksiranja več različnih formatov vhodnih datotek (Microsoft Office, PDF, XML, ...)

Solr je leta 2004 ustvaril Yonik Seeley kot interni projekt podjetja CNET Networks, ki ga je uporabil za dodatno zmogljivost pri iskanju po spletnih straneh podjetja. Dve leti kasneje je bila izvorna koda darovana skupini Apache Software Foundation. Od leta 2010 dalje razvoj rešitev Lucene in Solr poteka znotraj istega projekta – Apache Lucene/Solr. Tako kot Lucene je tudi Solr pisan v programskem jeziku Java in teče kot samostojni iskalni strežnik v aplikacijskih strežnikih, ki podpirajo izvajanje tehnologije Java Servlet (npr. Apache Tomcat, JBoss, IBM WebSphere, ...).

2.3.1 Nastavitve polj, filtrov in analizatorjev

Solr za razliko od Lucene omogoča uporabniku preprosto nastavitev lastnosti polj, ki se shranjujejo v indeks, in filtrov ter analizatorjev, ki delujejo na vhodno besedilo. Nastavitve se definirajo v dveh datotekah (`schema.xml` in `solrconfig.xml`) za vsak fizični indeks, ki ga Solr ustvari.

V prvi datoteki (`schema.xml`) definiramo tipe polj (tekstovno polje, datumsko polje, numerično polje, splošno polje, ipd.), seznam filtrov in analizatorjev, ki delujejo na vsebino posameznih polj (posebej za proces indeksiranja in proces iskanja), ter imena in pripadajoče nastavitve polj, ki so del dokumentov v indeksu.

Slika 2.5 prikazuje primer definicije dveh tekstovnih polj v datoteki `schema.xml`. Prvo polje z imenom `id` je tekstovno (*string*) polje, katerega vsebina se upošteva kot celota (v indeks se shrani dejanska vsebina polja, brez morebitnih razčlenitev vsebine). Polje se indeksira (da je

po njem možno iskati), vsebina se shrani v indeks (da je vsebino polja možno ob iskanju tudi prikazati), in polje mora biti obvezno prisotno na vsakem dokumentu, ki se pošlje v indeksiranje.

Drugo polje z imenom `besedilo` je tekstovno (*text*), katerega vsebina se filtrira in analizira pred indeksiranjem. Vsebino polja ne shranimo, vendar pa jo indeksiramo, da bo po njem možno iskati. Polje mora biti prav tako obvezno prisotno na vsakem indeksiranem dokumentu. Na to polje (bolje rečeno na tip tega polja) delujemo s filtrom `WhitespaceTokenizerFactory`, ki vsebino polja razdeli na posamezne žetone glede na prazna polja (npr. presledki, tabulatorji) in črke posameznih žetonov spremeni v male (`LowerCaseFilterFactory`). Enak analizator uporabimo tako pri indeksiranju (*index*) kot pri iskanju (*query*).

V tej datoteki definiramo tudi nekaj preostalih nastavitev, kot so definicija polja, ki vsebuje unikatne ključe dokumentov (npr. polje `id`), definicija privzetega operatorja med vnesenimi besedami v poizvedbi (`AND`, `OR`), ter definicija privzetega polja za iskanje, da uporabnikom ni potrebno v poizvedbi eksplicitno navesti polja po katerem želijo iskati[6].

Druga nastavitvena datoteka `solrconfig.xml` je obsežnejša in omogoča vrsto različnih nastavitev. Omenimo le nekatere:

- Navedbo poti do javanskih knjižnic razširitev (npr. indeksiranje relacijskih baz, razvrščanje dokumentov po kategorijah, ipd.)

- Nastavitev količine delovnega spomina, ki je na voljo za začasno hrambo podatkov predno se zapišejo v indeks (podatki se začasno hranijo v spominu zaradi zmogljivosti sistema)

- Nastavitev politik brisanja in zaključevanja podatkov v indeksu

- Nastavitev privzetega procesa iskanja in definicija morebitnih dodatnih polj, ki se pripnejo k vsaki poizvedbi

- Nastavitev procesov za inkrementalno indeksiranje za vsak vhodni format podatkov posebej (XML, JSON, Java)

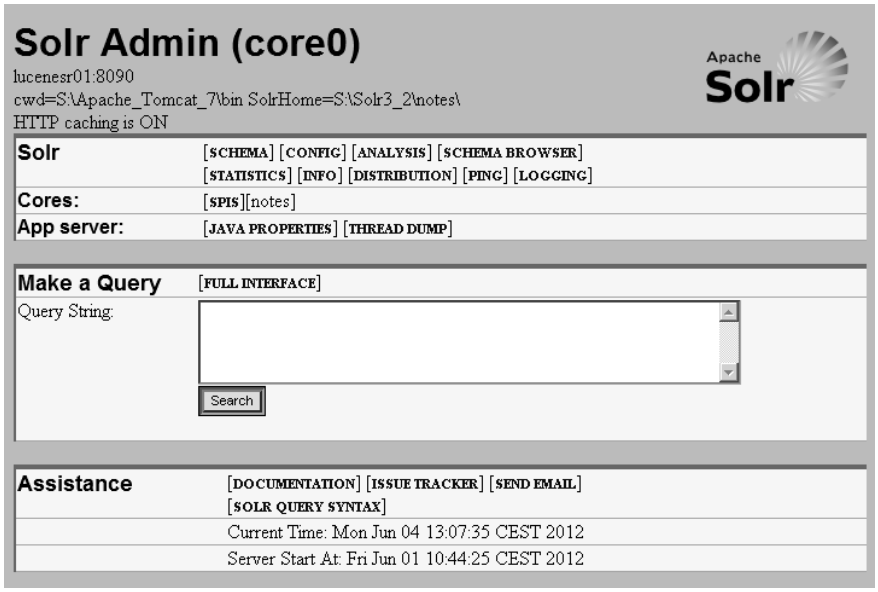
- Nastavitev posameznih razširitev iskalnika (preverjanje jezika – *spellchecker*, kategorizacija dokumentov – *clustering*, označevanje zadetkov – *highlighting*)

[6] Pogosto se v ta namen vsebina vseh polj na dokumentu prepiše v skupno polje, ki je namenjeno splošnemu iskanju. Tudi to nastavitev definiramo v datoteki `schema.xml` preko uporabe `<copyField source='ime_izvornega_polja' dest='ime_skupnega_polja'>`.

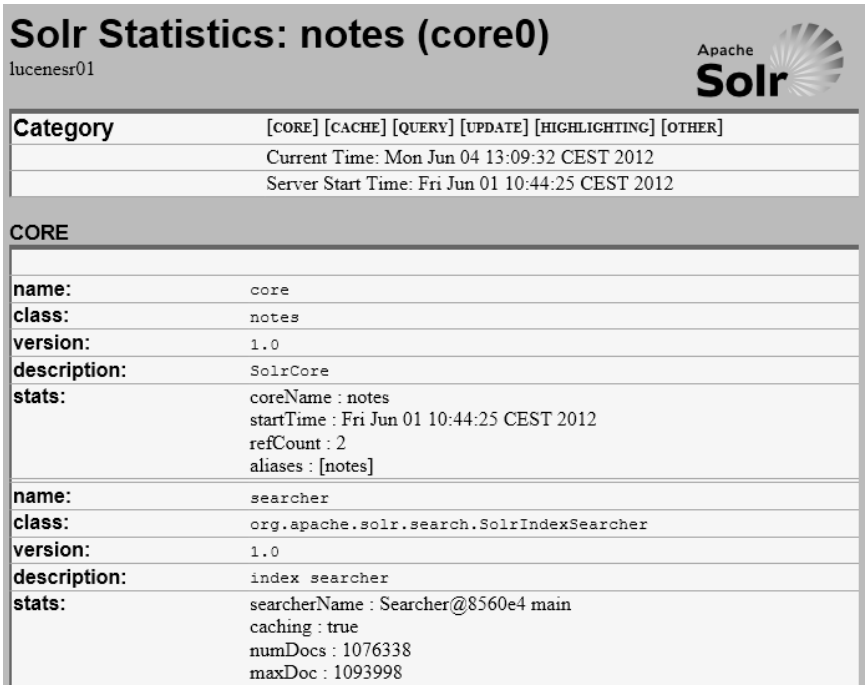
```
<schema name="indeks" version="1.1">
    <types>
           <!-- Tip polja string  povzroči, da se     -->
           <!-- vsebina polja upošteva kot en žeton   -->
           <!-- in se zato ne razdeli na več delov    -->
           <fieldtype name="string" class="solr.StrField"
                        sortMissingLast="true" omitNorms="true"/>

           <!-- Tip polja text povzroči, da se vsebina -->
           <!-- polja ustrezno filtrira in analizira    -->
           <fieldType name="text" class="solr.TextField" positionIncrementGap="100">
                  <!-- Definicija analizatorja za indeksiranje -->
                  <analyzer type="index">
                         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                         <filter class="solr.LowerCaseFilterFactory"/>
                  </analyzer>
                  <!-- Definicija analizatorja za iskanje -->
                  <analyzer type="query">
                         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                         <filter class="solr.LowerCaseFilterFactory"/>
                  </analyzer>
           </fieldType>
    </types>

    <fields>
           <field name="id" type="string" indexed="true" stored="true"
                  required="true" />
           <field name="besedilo" type="text" indexed="true"  stored="false"
                 required="true" />
    </fields>

    <!-- Ime polja, ki vsebuje unikatni ključ dokumentov -->
    <uniqueKey>id</uniqueKey>

    <!-- Ime polja po katerem izvajamo splošno iskanje -->
    <defaultSearchField>besedilo</defaultSearchField>

    <!-- Definicija privzetega operatorja med izrazi v poizvedbi -->
    <solrQueryParser defaultOperator="AND"/>
</schema>
```

Slika 2.5: Primer nastavitvene datoteke `schema.xml`, kjer definiramo dva tipa polj (*string* in *text*) ter ju povežemo z ustreznima poljema (`id` in `besedilo`). Polje `id` je definirano kot polje, ki vsebuje unikatne ključe dokumentov v indeksu, polje `besedilo` pa predstavlja splošno polje za iskanje, da v poizvedbi ni potrebno eksplicitno navesti imena polj po katerih iščemo. Primer prikazuje tudi definicijo analizatorjev in filtrov pri polju besedilo ter definicijo privzetega operatorja, ki bo deloval med besedami v poizvedbi (operator `AND`).

Pregled nastavitev je možen tudi preko skrbniškega spletnega vmesnika (Slika 2.6), kjer imajo upravitelji sistema na enem mestu pregled na nastavitvami indeksiranja in posameznimi indeksi, podrobni pregled statistike (Slika 2.7), ter možnost izvajanja poizvedb.



Slika 2.6: Spletni vmesnik strežnika Solr, kjer skrbniki sistema izvajajo pregled nad nastavitvami indeksiranja in posameznimi indeksi. Nastavitve polj, filtrov in analizatorjev predstavlja povezava »Schema«, nastavitve indeksiranja pa povezava »Config«. Prva neposredno prikaže vsebino datoteke `schema.xml`, druga pa vsebino datoteke `solrconfig.xml` (posebej za vsak izbrani indeks). Izbrani indeks predstavlja povezava v rubriki »Cores«. Preproste poizvedbe se izvaja v rubriki »Query String«, naprednejše poizvedbe pa so možne pod povezavo »Full interface«.



Slika 2.7: Podrobna statistika indeksa. Prikazana sta trenutni čas in čas, ko je bil strežnik Solr nazadnje uspešno zagnan. V spodnjem delu slike je prikazana statistika indeksa, kjer je navedeno trenutno število aktivnih dokumentov (tj. dokumentov, ki jih je možno iskati) v indeksu (*numDocs*), ter število vseh dokumentov v indeksu (*maxDocs*). Ko dokument pobrišemo iz indeksa se le-ta fizično ne izbriše dokler indeksa ne »optimiziramo« s posebnim ukazom. Ker je operacija optimizacije indeksa zahtevna, se jo izvaja šele, ko je le-ta nujna zaradi zaznavne upočasnitve iskanja ali zaradi prevelikega števila datotek, ki sestavljajo indeks. Spomnimo, da se ustvari nov segment vsakič, ko izvedemo potrditev indeksiranih dokumentov (*commit*). Z naraščanjem števila segmentov je tudi iskanje počasnejše. Optimizacija zmanjša število segmentov na najmanjšo možno število oz. jo zmanjša na število, ki jo določi skrbnik sistema.

2.3.2 Komunikacija s strežnikom Solr

Komunikacija s strežnikom Solr lahko poteka preko protokola HTTP ali preko vmesnika API. Oba imata svoje prednosti in slabosti, zato si naprej poglejmo nekaj primerov komunikacije.

2.3.2.1 Komunikacija preko protokola http

Komunikacija s strežnikom Solr preko protokola HTTP poteka tako, da strežniku v obliki ukazov URL posredujemo navodila za indeksiranje, iskanje ali izvajanje določenih opravil (potrditev – *commit*, ali optimizacija – *optimize* indeksa).

Vsebino dokumentov se strežniku Solr posreduje v treh oblikah:

- XML (Extensible Markup Language)
- JSON (JavaScript Object Notation)
- CSV (Comma Separated Values)

Kot primer bomo prikazali obliko XML – delovanje ostalih dveh oblik je podobno. Vzemimo dva preprosta dokumenta, ki vsebujeta dve polji (id in besedilo) in ustrezata shemi iz Slika 2.5. Vsebino obeh dokumentov prikazuje Slika 2.8.

```
<add>
    <doc>
           <field name="id">0001</field>
           <field name="besedilo">Vsebina prvega dokumenta</field>
    </doc>

    <doc>
           <field name="id">0002</field>
           <field name="besedilo">Vsebina drugega dokumenta</field>
    </doc>
</add>
```

Slika 2.8: Primer datoteke XML, ki vsebuje podatke o vsebini dveh dokumentov. Oba dokumenta dodamo v indeks tako, da vsebino datoteke posredujemo strežniku Solr preko ukaza URL `/update`.

Dokumenta se strežniku Solr posreduje v indeksiranje tako, da vsebino datoteke vključimo v ukaz URL. V ta namen lahko uporabimo brezplačni program cURL[7], ki je namenjen prenašanju podatkov preko ukazov URL in podpira vrsto različnih protokolov (HTTP, HTTPS, FTP, LDAP, POP3, ...). V nadaljevanju bomo prikazali posredovanje ukazov URL s pomočjo tega programa.

[7] Program je dostopen na naslovu http://curl.haxx.se.

Preprost ukaz, ki posreduje oba dokumenta v indeksiranje bi bil sledeči:

```
curl http://<naslov_IP_strežnika_Solr>:<št_vrat>/<ime_indeksa>/update?commit=true -H
     "Content-Type: text/xml" --data-binary '<vsebina_datoteke_XML>'
```

ali kot specifičen primer:

```
curl http://localhost:8090/solr/update?commit=true -H "Content-Type: text/xml" --data-binary
     '<add><doc><field name="id">0001</field>[...]</doc></add>'
```

kjer smo z [...] označili vsebino datoteke XML iz Slika 2.8, ki je nismo specifično navedli v ukazu. Tak ukaz bi povzročil, da bi strežnik Solr prejel vsebino obeh dokumentov in ju glede na nastavitve dodal v indeks. Ukaz prav tako naroča strežniku, da po končanem indeksiranju izvede zaključek procesa (`commit=true`) in s tem povzroči, da bosta oba dokumenta na voljo za iskanje. Če opcije `commit` ne uporabimo, jo lahko izvedemo kasneje, ko dodamo vse želene dokumente, preko ukaza:

```
curl http://<naslov_IP_strežnika_Solr>:<št_vrat>/<ime_indeksa>/update?commit=true
```

Namesto ukaza `commit=true` je smiselno uporabiti ukaz `commitWithin`, ki povzroči, da so dokumenti, ki jih posredujemo v indeksiranje potrjeni v določenem intervalu. Spodnji primer prikazuje dodajanje istih dveh dokumentov v indeks, pri čemer navedemo strežniku Solr, da ju mora zaključiti v roku 15 sekund po preteku obdelave:

```
curl http://localhost:8090/solr/update?commitWithin=15000 -H "Content-Type: text/xml"
     --data-binary '<add><doc><field name="id">0001</field>[...]</doc></add>'
```

Dokumente lahko iz indeksa brišemo na dva načina:

- Z navedbo polja in njegove vsebine
- Z navedbo poizvedbe

Sledeči primer prikazuje vsebino XML, ki jo posredujemo strežniku Solr, da izvede brisanje obeh, prej dodanih, dokumentov iz indeksa, kjer prvega brišemo preko navedbe polja, drugega pa preko navedbe poizvedbe:

```
<delete>
       <id>0001</id>
</delete>

<delete>
       <query>id:0002</query>
</delete>
```

Brisanje preko navedbe polja bo zajelo le tiste dokumente, ki vključujejo to polje in vsebujejo vsebino, ki jo navedemo (npr. »0001«). Brisanje preko poizvedbe povzroči, da Solr najprej nad indeksom izvede poizvedbo, ki jo navedemo (poizvedbe so lahko kompleksnejše kot ta, ki smo jo navedli kot primer), nato pa pobriše iz indeksa vse dokumente, ki ustrezajo tej poizvedbi. Če želimo izbrisati vse dokumente uporabimo poizvedbo `*:*`.
Iskanje dokumentov poteka na podoben način, vendar s to razliko, da poizvedbo pošljemo strežniku Solr kot parameter v ukazu URL. Poleg poizvedbe lahko navedemo dodatne parametre kot so: število vrnjenih vrstic (`rows`), seznam vrnjenih polj iz indeksa (`fl`), format vrnjenega seznama zadetkov (`wt=[xml|json|csv]`), ipd. Edini zahtevani parameter je poizvedba (`q`). Sledeči primer prikazuje izvedbo splošne poizvedbe nad določenim indeksom:

```
curl http://<naslov_IP_strežnika_Solr>:<št_vrat>/<ime_indeksa>/select?q=<poizvedba>
```

Vrnjeni izpis bo v formatu XML, saj je to privzeti izpis. Privzeto število vrnjenih vrstic je 10.
Solr lahko izvaja tudi porazdeljeno iskanje po več indeksih hkrati. Ti indeksi se lahko nahajajo na istem strežniku Solr ali pa se nahajajo na različnih strežnikih. Porazdeljeno iskanje izvajamo tako, da ukazu URL dodamo parameter `shards`, kjer navedemo imena vseh strežnikov in indeksov, ki jih želimo preiskati. Najbolje, da porazdeljeno iskanje prikažemo s primerom. Kot referenčno točko si izberimo poljubni strežnik Solr, ki mora biti delujoč in vsebuje dva indeksa – *indeks1* in *indeks2*. Poleg teh dveh indeksov želimo še preiskati indeks *indeks3* na nekem drugem strežniku. Primer iskanja je sledeči:

```
curl http://<streznik1>:<vrata1>/<indeks1>/select?q=<poizvedba>
    &shards=<streznik1>:<vrata1>/<indeks2>,<streznik2>:<vrata2>/<indeks3>
```

Solr bo izvedel poizvedbo na vseh treh indeksih in posamezne sezname zadetkov ustrezno združil in vrnil uporabniku v pregled. Porazdeljeno iskanje se iz stališča uporabnika izvaja enako kot iskanje samo po enem indeksu.

2.3.2.2 Komunikacija preko vmesnika API

Strežnik Solr omogoča komunikacijo z zunanjimi odjemalci tudi preko vmesnika API. Odjemalci, ki lahko koristijo funkcije vmesnika API so lahko pisani v Javi, Pythonu, PHPju, .NETu, ipd. Ker je Solr pisan v Javi si bomo pogledali nekaj preprostih primerov kar s pomočjo javanske kode. Sledili bomo enakim primerom kot smo jih navedli v razdelku 2.3.2.1.

Java odjemalec, ki omogoča komunikacijo s Solrjem preko vmesnika API se imenuje Solrj in vsebuje java vmesnik za dodajanje, posodabljanje, brisanje in izvajanje poizvedb nad indeksom. Solrj ni nič drugega kot skupek raznih JAR (Java ARchive) datotek.

Dodajanje dokumentov v indeks je tudi v primeru uporabe vmesnika API preprosto. Najprej je potrebno ustvariti povezavo s strežnikom Solr (objekt `HttpSolrServer`), nato pa za vsak dokument, ki ga želimo dodati v indeks ustvarimo objekt `SolrInputDocument`, na njem

definiramo polja in njihovo vsebino, ter dokument pošljemo strežniku. Spodnji primer prikazuje dodajanje istih dveh dokumentov kot sta definirana na Slika 2.8, le da v tem primeru ne potrebujemo datoteke XML, temveč vso vsebino definiramo v sami kodi:

```
// Ustvarimo povezavo s strežnikom Solr
HttpSolrServer solr =
            new HttpSolrServer("http://<naslov_strežnika>:<vrata>/<indeks>");

// Ustvarimo oba dokumenta
SolrInputDocument dok1 = new SolrInputDocument();
dok1.addField("id", "0001");
dok1.addField("besedilo", "Vsebina prvega dokumenta");

SolrInputDocument dok2 = new SolrInputDocument();
dok2.addField("id", "0002");
dok2.addField("besedilo", "Vsebina drugega dokumenta");

// Ju dodamo na kolekcijo dokumentov
Collection<SolrInputDocument> dokumenta = new ArrayList<SolrInputDocument>();
dokumenta.add(dok1);
dokumenta.add(dok2);

// In ju pošljemo v obdelavo
solr.add(dokumenta);

// Ter zaključimo postopek indeksiranja
solr.commit();
```

Brisanje dokumentov je enako preprosto. Na voljo imamo možnost brisanja glede na vsebino polja določenega kot nosilca unikatnih ključev dokumentov (nastavitev `uniqueKey` na Slika 2.5), ali preko poizvedbe. Primer prikazuje brisanje prvega dokumenta iz indeksa preko polja in drugega preko poizvedbe:

```
// Ustvarimo povezavo s strežnikom Solr
HttpSolrServer solr =
            new HttpSolrServer("http://<naslov_strežnika>:<vrata>/<indeks>");

// Izbrišemo prvi dokument
solr.deleteById("0001");

// Izbrišemo drugi dokument
solr.deleteByQuery("id:0002");

// Ter zaključimo postopek brisanja
solr.commit();
```

Izvajanje poizvedb vršimo na podoben način. Po vzpostavitvi povezave s strežnikom Solr, ustvarimo objekt `SolrQuery`, ki predstavlja poizvedbo in njene nastavitve (npr. število vrnjenih vrstic, sortirano polje, ipd.), nato pa to poizvedbo izvedemo. Sledeči primer prikazuje izvedbo poizvedbe, ki iz indeksa pridobi vse dokumente in jih uredi po polju id v padajočem vrstnem redu:

```
// Ustvarimo povezavo s strežnikom Solr
HttpSolrServer solr =
             new HttpSolrServer("http://<naslov_strežnika>:<vrata>/<indeks>");

// Kreiramo objekt poizvedbe
SolrQuery query = new SolrQuery();
// Nastavimo poizvedbo
query.setQuery("*:*");
// Dodamo sortirano polje
query.addSortField("id", SolrQuery.ORDER.desc);

// Izvedemo poizvedbo
QueryResponse response = solr.query(query);

// Pridobimo seznam dokumentov
SolrDocumentList hits = response.getResults();
```

2.4 IBM Lotus Notes & Domino

Za primarni vir podatkov smo vzeli zbirke IBM Lotus Notes. Ideja in razvoj programske opreme za skupno sodelovanje in komuniciranje Lotus Notes in Domino ima svoj začetek že v začetku 70. let prejšnjega stoletja, ko je bilo na univerzi v Illinoisu narejeno orodje PLATO Notes za sledenje napakam na programski opremi. Le nekaj let kasneje je bila funkcionalnost orodja PLATO Notes razširjena in marsikatero funkcionalnost najdemo v programski opremi Lotus še danes [14].

Leta 1984 se je pričel razvoj odjemalca Lotus Notes pod okriljem družbe Iris Associates Inc. Začetni razvoj je trajal do leta 1986, ko je bila notranjim uporabnikom predstavljena prva delujoča verzija. Prva verzija je na trg prišla leta 1989 in v prvem letu je bilo prodanih 35 000 kopij. Sistem je bil sestavljen iz dveh delov: odjemalca (Notes) in strežnika (Domino), ki sta oba delovala na operacijskem sistemu Microsoft DOS oz. IBM OS/2. Že prva verzija je vsebovala sisteme za poštno komuniciranje, diskusije, imenike, prav tako pa je bilo uporabnikom na voljo okolje za izgradnjo lastnih programskih rešitev.

Lotus Notes je za tisti čas na trg prinesel funkcionalnosti, ki praktično niso bile prisotne v ostalih primerljivih orodjih. Omogočeno je bilo šifriranje, podpisovanje in ovrednotenje z uporabo tehnologije RSA. Prav tako so bile vključene napredne možnosti oddaljenega povezovanja preko klicnih povezav, uporabe elektronske pošte, pomoči, ipd. Skrbnikom okolja

je bilo omogočeno preprosto dodajanje in urejanje uporabnikov, kreiranje certifikatov in dodajanje le-teh v skupni imeniški sistem. Ne nazadnje je bil definiran varnostni model, ki v podobni obliki obstaja še danes.

Slika 2.9: Prva različica programske opreme Lotus Notes iz leta 1989 [14].

Glavnino programske opreme Lotus Notes in Domino tvori podatkovna zbirka. Vsi podatki, ki spadajo v eno podatkovno zbirko Domino so fizično shranjeni v eni datoteki s končnico NSF (*Notes Storage File*). Branje, vpisovanje in urejanje podatkov poteka preko uporabe storitve *Notes Storage Facility*, ki definira vse funkcionalnosti, ki so uporabniku (nekatere morda samo v okrnjeni različici) na voljo.

2.4.1 Varnostni model

Sistem Lotus Domino vsebuje zmogljiv varnostni model [3], ki dostope do podatkov preverja na več ravneh. Ker iskalnik upošteva varnostni model, kot ga izvaja Notes, bomo v nadaljevanju na kratko pojasnili kako le-ta deluje.

Sistem Lotus Domino že od samega začetka vključuje močan varnostni model (Slika 2.10), ki sestoji iz šestih nivojev [3]:

- *Omrežje* – varnost omrežja skrbi za preprečitev nepooblaščenega dostopa do omrežja na katerem se nahajajo strežniki Lotus Domino. Dostop do omrežja se tipično nadzoruje in uveljavlja s pomočjo ustrezne omrežne strojne in programske opreme. Sistem Lotus dodatno omogoča šifriranje podatkov, ki se pretakajo med posameznimi strežniki Domino in s tem onemogoča branje podatkov na omrežju z raznimi analizatorji omrežnih protokolov (npr. Wireshark). Podatke se šifrira tako, da vključimo možnost šifriranja na vseh vratih (angl. *port*), ki jih strežnik Domino uporablja, ali pa z uporabo protokola SSL (angl. *Secure Sockets Layer*).

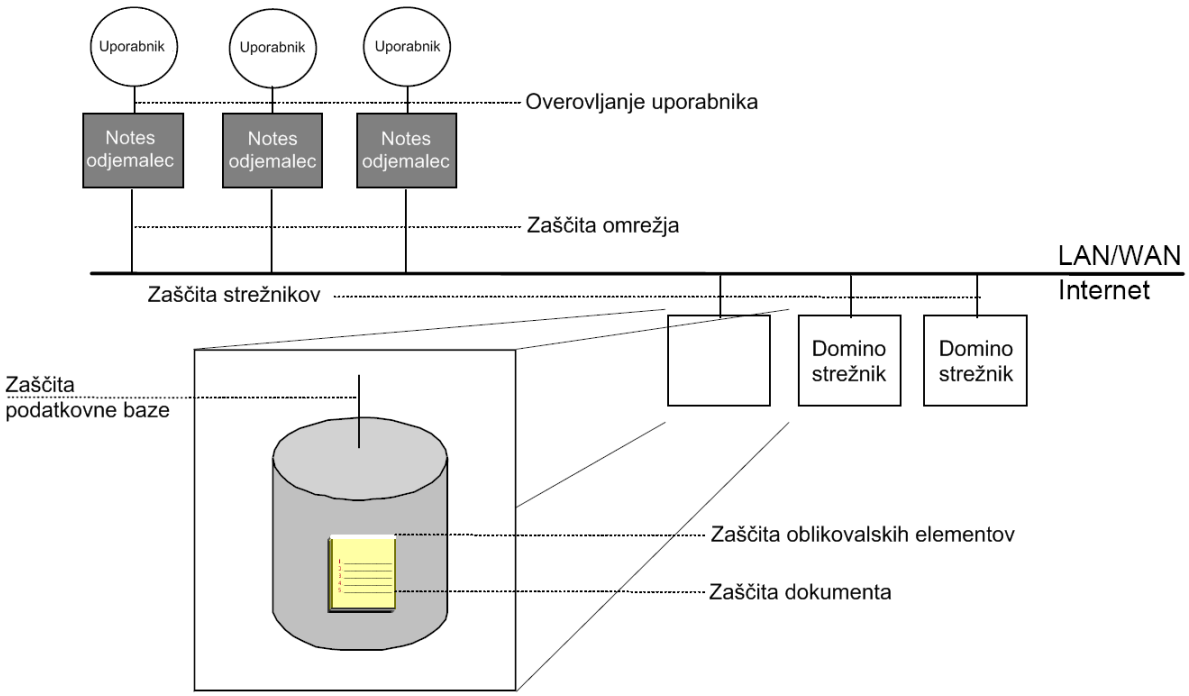
- *Overovitev uporabnikov* – je proces, ki se izvaja med odjemalcem Notes in strežnikom Domino, ki omogoča potrditev in overovitev identitete uporabnika *in* strežnika, ko uporabnik dostopa do strežnika. Odjemalec in strežnik uporabita certifikate, ki so shranjeni v posebnih datotekah (datoteke *Notes ID*) za medsebojno potrditev in overovitev. Ko uporabniki dostopajo do strežnika preko spletnega brskalnika, se overovitev uporabnikov izvrši s pomočjo certifikatov X.509 ali uporabe uporabniškega imena in gesla, ki ju posreduje uporabnik.

- *Strežniki* – dostop do strežnika se določi šele, ko je uporabnik ustrezno overovljen s strani strežnika. Takrat se uporabnikovo uporabniško ime primerja s posebnih seznamom (angl. *Deny Access Groups*) in če uporabnik *ni* naveden na tem seznamu, mu je dostop do strežnika omogočen.

- *Zbirke* – če ima uporabnik dostop do strežnika, na katerem se nahaja zbirka do katere želi dostopati, se preveri seznam dostopov (angl. *Access Control List* – ACL), ki je del vsake zbirke. Zbirke, ki se nahajajo lokalno (tj. uporabnik do njih ne dostopa preko strežnika), so lahko tudi šifrirane na določenega uporabnika, ki bo do nje lahko dostopal le s pomočjo ustrezne datoteke Notes ID. Če uporabnik izgubi certifikat oz. geslo in le-teh ni možno več obnoviti, so vsi šifrirani podatki nedostopni – tudi skrbnikom sistema ne.

- *Oblikovni elementi* – varnost oblikovnih elementov (angl. *design elements*) skrbi za določanje dostopa do obrazcev (angl. *form*), pogledov (angl. *view*) in map (angl. *folder*), ki so del posameznih zbirk. Uporabniku mora biti najprej omogočen dostop do zbirke predno mu je omogočen dostop do posameznih oblikovnih elementov znotraj zbirke. Uporabnikom se lahko omogoči dostop do le nekaterih elementov, do drugih pa ne. Obrazce je možno zavarovati preko seznama dostopov ali z uporabo šifrirnih ključev. Pogledi in mape vsebuje le seznam dostopov.

- *Dokumenti* – dostop do dokumentov se določa glede na celotni dokument (z uporabo bralskih polj), ali na posamezne dele dokumenta preko uporabe posebnih pravil (angl. *hide-when formulas*). Dostop do posameznih polj na dokumentu je možno urejati z uporabo šifrirnih ključev.

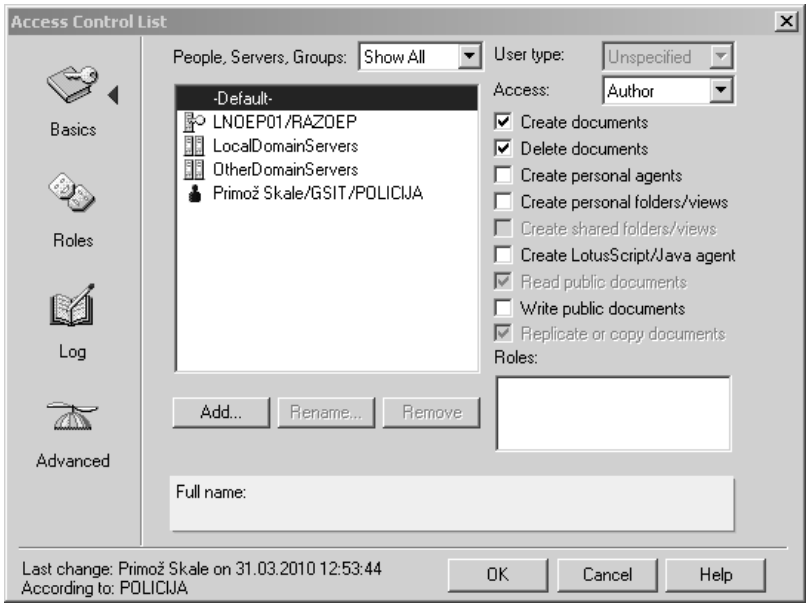


Slika 2.10: Šest plasti varnostnega modela sistema Lotus Domino. Zadnje tri plasti zadevajo samo podatkovno zbirko. Varnost zbirke je omogočena le v primeru, ko se le-ta nahaja na strežniku - lokalni dostop do zbirke obide varnostni model. Zbirko je lokalno mogoče zavarovati le s šifriranjem.

2.4.2 Nivoji dostopa do zbirke Lotus Notes

Vsaka zbirka, ki je del Domino sistema, vsebuje seznam dostopov, ki določa kateri uporabniki (in strežniki – za primer replikacije) imajo dostop do vsebine zbirke (Slika 2.11). Seznam dostopov je v zbirki shranjen kot dokument, ki pa ni viden v običajnih pogledih dokumentov. Do njega je mogoče dostopa tako preko menijev kot preko programske kode – seveda z ustreznimi pravicami. Varnostni nivoji, ki so uporabniku oz. strežniku dodeljeni, so med seboj enaki, vendar pa pri uporabniku določajo do katerih vsebin lahko dostopa, pri strežniku pa katere vsebine lahko replicira.

Seznam ACL prav tako vključuje dva pomembna dostopa: anonimni (angl. *anonymous*) in privzeti (angl. *default*) dostop. Anonimni dostop določa varnostni dostop neprijavljenim uporabnikom (npr. uporabnikom, ki do vsebine dostopajo neprijavljeni preko spletnega brskalnika), privzeti dostop pa določa varnostni dostop prijavljenim in neprijavljenim uporabnikom – za slednje, le če seznam ACL ne vključuje definicije varnosti anonimnega dostopa. Seznam ACL vsake zbirke mora vsebovati definicijo varnosti privzetega dostopa – anonimni dostop je poljuben.



Slika 2.11: Seznam dostopov določa kateri uporabniki in strežniki imajo dostop do vsebine zbirke.

Seznamu ACL je možno poleg poimenskih uporabniških imen dodati tudi skupine (angl. *groups*) in vloge (angl. *roles*).

Namen skupin je združevanje več uporabnikov z istimi pravicami na posamezni zbirki. Skupino in njene člane definiramo globalno v imenskem imeniku. Pravice, ki jih uporabniki skupine posedujejo definiramo posebej za vsako zbirko, kateri skupino dodamo na seznam dostopov. Ista skupina uporabnikov ima lahko tako različne pravice na različnih zbirkah.

Uporabnik je lahko naveden v več skupinah naenkrat. Če so te skupine navedene na seznamu dostopov zbirke z različnimi pravicami, se uporabniku dodeli pravice skupine, ki ima najvišji dostop. Če je uporabnik naveden tako v skupinah kot poimensko na seznamu dostopov se vedno upošteva pravice, ki so dodeljene poimenskemu uporabniku.

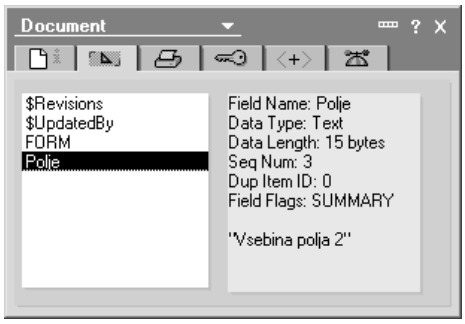
Namen vlog je urejanje dostopa do oblikovnih elementov in funkcionalnosti zbirke. Vloga definira množico uporabnikov oz. strežnikov. Vloge se obnašajo podobno kot skupine, le, da se jih definira za vsako zbirko posebej. Vloge se lahko dodeli uporabnikom, strežnikom in skupinam, ki so navedeni na seznamu dostopov zbirke. Praviloma se jih uporablja v primerih, ko želimo določene dele dokumentov skriti pred uporabniki, ali pa različnim uporabnikom (npr. glede na njihovo delovno mesto) omogočiti uporabo ustreznih funkcionalnosti zbirke.

Zbirke omogočajo sedem nivojev dostopa do vsebin: *upravitelj*, *načrtovalec*, *urednik*, *avtor*, *bralec*, *dodajalec* in *ni dostopa*. Dodatno je možno določiti še pravice ustvarjanja in brisanja dokumentov, kreiranja lastnih in deljenih kazal ter map, agentov, ipd. Nekatere dodatne možnosti niso na voljo za vse nivoje dostopa (npr. če ima uporabnik le nivo dostopa bralec – torej ima le pravico branja dokumentov – je dodatna možnost, ki mu omogoča brisanje dokumentov, nesmiselna).

Tabela 2.4 prikazuje vseh sedem nivojev dostopa ter podroben opis dovoljenj uporabnikov in strežnikov s temi nivoji.

V seznamu ACL je pomembno določiti tudi ustrezne nivoje dostopa posameznim strežnikom, saj lahko sicer pride do nepopolnih replikacij. Če ima strežnik na zbirki, v katero vpisuje spremenjene podatke le dostop bralca, sprememb ne bo mogel vpisati. Spremembe bodo lahko vpisane le, če bo drugi strežnik iste spremembe prepisal k sebi (če ima vsaj dostop bralca na oddaljeni repliki, ter dostop urednika oz. avtorja na lokalni repliki). Strežnikom se splošno na lokalnih replikah vedno doda najvišje pravice, saj se uporabnik na zbirko ne more prijaviti s certifikatom strežnika – to lahko stori le strežnik[8]. S tem preprečimo morebitne težave, ki bi se lahko pojavile pri replikaciji.

Tabela 2.4 prikazuje le prvega izmed treh nivojev dostopa, ki jih zbirka omogoča (in jih strežnik izvršuje). Drugi nivo dostopa določa, kateri oblikovni elementi so posameznim uporabnikom na voljo. Če določen uporabnik nima pravice dostopa do obrazca s katerim vnese podatke, jih tudi ne bo mogel – razen programsko. Ker so podatki in oblikovni elementi v zbirkah ločeni, se lahko uporabnikom tudi onemogoči ogled določenih dokumentov – tj. dokumentov, ki uporabljajo te oblikovne elemente za prikaz. Seveda pa to ni prava varnost, saj si vsak uporabnik, ki ima dostop do dokumenta, lahko vsebino ogleda v lastnostih dokumenta preko kazal (Slika 2.12).



Slika 2.12: Vsebino dokumenta se lahko ogleda preko za to namenjenega oblikovnega elementa (*form*) oz. v lastnostih dokumenta, ki vsebuje seznam vseh elementov dokumenta ter njihovo vsebino.

Polja v dokumentu je možno tudi šifrirati na posamezne uporabnike – v tem primeru se podatki dešifrirajo šele, ko uporabnik dokument pregleda preko obrazca. Vsebine polj, ki so šifrirana, ni mogoče pregledati v lastnostih dokumenta. Zbirke omogočajo tudi omejitev dostopa do posameznih kazal in map. Pomembno je poudariti, da se s tem ne prepreči dostop do dokumentov, saj si vsak uporabnik lahko vedno kreira svoj pogled oz. mapo.

[8] To sicer ni čisto res, vendar bi v ta namen moral uporabnik napisati ločen program, ki bi se vedel kot strežnik - prav tako bi moral imeti dostop do strežniškega certifikata, ki pa je tipično dobro varovan.

Dostop do dokumentov se ureja z dvema posebnima poljema: poljem bralci (angl. *readers*) in poljem avtorji (angl. *authors*). Ker ti dve polji igrata pomembno vlogo v tretjem nivoju dostopa do vsebin, si ju poglejmo natančneje.

Nivo dostopa	**Katera dovoljenja imajo uporabniki in strežniki?**
Upravitelj	Urejanje seznama ACL, šifriranje podatkovne baze, nastavitev replikacijskih nastavitev, brisanje celotne baze in izvrševanje vseh opravil, ki jih omogočajo nižji nivoji dostopa.
Načrtovalec	Urejanje vseh oblikovnih elementov, ustvarjanje indeksa in izvrševanje vseh opravil, ki jih omogočajo nižji nivoji dostopa.
Urednik	Ustvarjanje in urejanje vseh dokumentov, tudi tistih, ki so jih ustvarili drugi. Branje vseh dokumentov, razen tistih, ki imajo omejitev branja glede na določene uporabnike oz. skupine. Urejanje dokumenta je mogoče le, če je dokument viden za branje.
Avtor	Ustvarjanje dokumentov (če je vključena dodatna možnost »ustvarjanje dokumentov«). Urejanje le tistih dokumentov, ki vsebujejo posebno polje *authors* in je trenutni uporabnik na njem naveden. Branje vseh dokumentov, razen tistih, ki imajo omejitev branja glede na določene uporabnike oz. skupine.
Bralec	Branje vseh dokumentov, razen tistih, ki imajo omejitev branja glede na določene uporabnike oz. skupine.
Dodajalec	Ustvarjanje dokumentov – uporabniki s to pravico dostopa lahko dokumente le kreirajo ne morejo pa jih brati (npr. podobno deluje poštni nabiralnik. Pošto je možno ustvariti, ko pa je oddana v nabiralnik, jo ni mogoče več prevzeti).
Ni dostopa	Brez dostopa do vsebine zbirke, razen do dokumentov, ki so označeni kot javni (posebno polje).

Tabela 2.4: Opis vseh sedmih nivojev dostopa do zbirke. Najvišji nivo dostopa upravitelj se po navadi določi skrbniku zbirke, saj omogoča urejanje seznama dostopov, šifriranje vsebine zbirke in brisanje celotne vsebine [3].

2.4.3 Dokumentni dostop

Dokumentni dostop določa zadnjo plast varnostnega modela. Pri določanju dostopov do podatkov (oz. dokumentov) igrata največjo vlogo polji *bralci* in *avtorji*. Vsebina teh dveh polj lahko vsebuje imena uporabnikov, skupin in vlog. Če dokument vsebuje polje bralcev bodo do

njega lahko dostopali le uporabniki (oz. strežniki), ki so bodisi navedeni v tem polju, bodisi so del skupine oz. vloge, ki je navedena v tem polju. Pomembno je poudariti, da uporaba teh dveh polj lahko zmanjša zmogljivost strežnika, saj mora strežnik pri odprtju pogleda, le-tega vsakič sproti ustvariti za posameznega uporabnika. Dokumenti, do katerih uporabnik nima dostopa, se na pogledu sploh ne prikažejo.

Polje avtorjev ima podobno vlogo kot polje bralcev, s to razliko, da določa kateri uporabniki lahko določen dokument urejajo. Poudarimo, da lahko dokumente urejajo vsi uporabniki z glavnim dostopom vsaj *urednik* (Tabela 2.4). Uporabniki, ki imajo glavni dostop nastavljen na *avtor* lahko le kreirajo dokumente. Urejanje dokumentov jim je omogočeno le, če dokument vsebuje polje avtorjev in so v tem polju tudi navedeni (bodisi preko imena, skupine oz. vloge).

Poleg uporabe polj bralcev in avtorjev se na dokumentni ravni lahko zaščiti tudi posamezne dele besedila in posamezna polja. Za določena polja lahko definiramo tudi, da mora imeti uporabnik za urejanje vsaj pravico urednika – pravica avtorja v tem primeru ne bo dovolj.

Najmočnejša zaščita, ki jo zbirka nudi je šifriranje[9]. Sistem Lotus Domino že sam po sebi vključuje močan sistem šifriranja, ki je za uporabnika transparenten. Dokumente je možno tako podpisovati kot šifrirati. Šifriranje poteka v povezavi z imenskim imenikom ali z javnimi ključi shranjenimi v uporabnikovem certifikatu. Uporabnik, ki želi določen dokument (in polja na njem) šifrirati za druge uporabnike, imena šifrirnih ključev navede v za to namenjeno polje. Če dokument šifrira brez uporabe posebnega polja, se dokument šifrira samo na trenutnega uporabnika. Ko se tak dokument odpre, odjemalec pregleda seznam uporabnikov za katere je bil dokument šifriran. Če je uporabnik, ki dokument odpira, na tem seznamu naveden, odjemalec prevzame uporabnikov zasebni ključ (iz certifikata) in z njim dešifrira podatke na dokumentu. Če šifrirani podatki nisi namenjeni trenutnemu uporabniku oz. pride pri dešifriranju do napake, se dokument odpre, uporabniku pa se javi, da nekateri deli dokumenta ne bodo vidni.

[9] V času, ko so ostali ponudniki (Microsoft, Netscape, ipd) nudili uporabnikom 40- oz. 56-bitno zaščito, je Domino podatkovna baza nudila uporabniku kar 64-bitno zaščito. Ta zaščita je veljala samo za ameriške uporabnike. Ostali uporabniki so sicer res imeli podatke šifrirane z 64-bitnimi ključi, vendar se je del ključa (zadnjih 24 bitov) dodatno šifriral z javnim ključem, ki je bil v lasti ameriške NSA (National Security Agency). Napadalci, ki niso imeli dostopa do tega dela ključa so morali ugotoviti vseh 64 bitov ključa, NSA pa le 40 bitov [15].

3 Opis rešitve

V tem poglavju bomo opisali rešitev, ki smo jo razvili za potrebe policije. Predstavili bomo razloge za potrebo uporabe relacijske baze in replikacijo nekaterih podatkov iz imenskega imenika Lotus Domino; varnostni model, ki ga iskalnik uporablja; proces zajema podatkov iz zbirk Lotus Notes; proces interakcije indekserja s strežnikom Solr, in izvajanje porazdeljenih poizvedb ter urejanje zadetkov po določenih poljih.

3.1 Uporaba relacijske baze

Rešitev za uspešno delovanje zahteva uporabo relacijske baze, ki mora biti stalno dostopna. Razlogi za uporabo relacijske baze so različni, in segajo od shranjevanja informacij o virih podatkov do izvajanja replikacije podatkov iz obstoječega imenskega imenika. Rešitev trenutno uporablja 13 relacijskih tabel, ki so združene v skupni bazi. Sledeči seznam predstavlja nazive in opis delovanja posamezne tabele:

- DOMACCESS – vsebuje informacije o dostopu do posamezne zbirke Lotus Notes, ki jo sistem indeksira.
- DOMCRED – vsebuje prijavne podatke in naslove strežnikov na katere se proces indeksiranja overi, ko prične z indeksiranjem posamezne zbirke. Prijava je lahko oddaljena preko protokola DIIOP, lokalna preko lokalno nameščenega odjemalca Lotus Notes, ali preko protokola LDAP.
- DOMDIR – vsebuje seznam imenskih imenikov Lotus Domino za potrebe overitve uporabnikov ter replikacije skupin, ki so navedene v posameznem imenskem imeniku. Ta tabela je povezana s tabelo DOMCRED preko skupnega ključa.
- DOMREPL – vsebuje replicirane podatke o skupinah v imenskem imeniku in vlogah na zbirkah zaradi hitrejšega dostopanja do uporabnikovih pravic ob prijavi uporabnika.
- DOMLOGIN – vsebuje podatke o domenah v katere se uporabnik lahko prijavi. Ime domene je vezano na imenski imenik preko katerega se uporabniki overijo. Ta tabela je povezana s tabelo DOMDIR preko skupnega ključa.
- MAPNOTESWWW – določa kako se posamezne povezave do dokumenta preslikajo (velja za preslikavo iz povezave Notes v spletno povezavo).
- MAPWWWNOTES – določa kako se posamezne povezave do dokumenta preslikajo (velja za preslikanje iz spletne povezave v povezavo Notes).
- SOLR – vsebuje seznam vseh strežnikov Solr, ki izvajajo indeksiranje.
- SOLRSEARCH – vsebuje seznam vseh strežnikov Solr, ki izvajajo iskanje.
- SRCNOTES – vsebuje seznam vseh zbirk, ki jih indeksiramo (razen imenskih imenikov, ker so le-ti že navedeni v tabeli DOMDIR).
- SYSLOG – naslovi IP sistemov kamor se pošiljajo sporočila *syslog* o delovanju indeksiranja in iskanja.

- FRMFIELDS – vsebuje preslikave polj na obrazcih (*form*) zbirk v ustrezno polje v indeksu in tip preslikave za vsak vir posebej.
- JSPVIEW – vsebuje seznam polj in njihove opise, kateri se prikažejo na strani zadetkov za vsako zbirko posebej.

Nekatere tabele so med seboj povezane preko skupnih ključev, druge so samostojne. V nadaljevanju bomo podrobno predstavili zgradbo in delovanje posamezne tabele.

3.1.1 Opis in uporaba tabel

Sistem uporablja relacijsko bazo za shranjevanje podatkov in nastavitev zaradi njene preproste uporabe, hitrosti dostopa do podatkov ter možnosti replikacije na druge sisteme. V prejšnjem razdelku smo na kratko predstavili tabele, ki jih sistem uporablja, v tem delu pa bomo opis razširili tako, da bomo opisali s primerom delovanje posamezne tabele ter njeno shemo.

3.1.1.1 Tabela DOMACCESS

Tabela DOMACCESS vsebuje informacije o dostopu uporabnikov do posameznih zbirk. Zgradbo tabele prikazuje Tabela 3.1. Vsaka zbirka ima že vgrajen sistem za določanje dostopov katerega smo predstavili v razdelku 2.4.1. Uporabniki imajo na zbirki različne pravice dostopa. Prav tako pa lahko uporabnike, ki jim določamo dostop, postavimo v tri skupine:

- Privzeto (*default*) – če uporabnik ni implicitno (tj. kot del skupine) oz. eksplicitno (tj. kot poimensko) naveden na seznamu za dostopov spada v *privzeto* skupino. Sistem vedno najprej preveri, ali je uporabnik implicitno ali eksplicitno naveden na seznamu za dostop, šele nato privzame privzeti dostop do zbirke. Omenimo še, da ima uporabnik lahko popolni dostop do zbirke, vendar pa, če upravitelji sistema določijo, da uporabnik nima dostopa do strežnika (to se določi v imenskem imeniku) na katerem se zbirka nahaja, tak uporabnik do zbirke ne bo mogel dostopati – varnostna politika sistema Lotus Domino deluje v slojih, kjer prvi sloj določa dostop do strežnika, zadnji sloj pa določa dostop do posameznega polja na dokumentu znotraj določene zbirke.
- Skupina – uporabnik je lahko del skupine, ki se jo navede na seznamu dostopov. Uporabniku se določijo pravice, kot jih ima skupina. Če je uporabnik naveden tako implicitno kot eksplicitno, se vedno upošteva eksplicitni dostop (tj. če je uporabnik naveden v skupini, ki ima dostop do zbirke in dodatno še poimensko, kjer nima dostopa, se mu dostop onemogoči, saj eksplicitne pravice prevladajo nad implicitnimi).
- Uporabnik – uporabnik je poimensko naveden na seznamu dostopov in se mu dodelijo pravice, kot jih je določil upravitelj zbirke.

Ime atributa	Tip	PK?	Null?	Opis
ID	int	Da	Ne	Primarni ključ – samodejno povečevanje
REPLID	char(16)	-	Ne	Številka replike
USRNAME	varchar(200)	-	Ne	Ime uporabnika
DBACCESS	char(1)	-	Ne	Ali ima uporabnik dostop?

Tabela 3.1: Zgradba tabele DOMACCESS, ki jo sistem uporablja za določanje dostopov uporabnikov do posameznih zbirk. Atribut REPLID vsebuje številko replike zbirke, atribut USRNAME vsebuje polno uporabniško ime ali niz -Default-, in atribut DBACCESS vrednost 0 (ni dostopa) ali 1 (dostop).

Sistem za indeksiranje do posameznih zbirk dostopa z uporabniškim računom, ki ima neomejen dostop in je s tem sistemu omogočeno neomejeno indeksiranje vseh podatkov v taki zbirki. Vendar pa je to dvorezni meč, saj povzroči, da uporabniki, ki po tako pripravljenem indeksu izvajajo poizvedbe, vidijo tudi dokumente, ki jim v okolju Lotus ne bi bili dostopni. Zato poleg vsake poizvedbe, ki jo uporabnik izvede, sistem uporabi t.i. *filter poizvedbe* (angl. *FilterQuery*), ki uporabniku vrne le tiste dokumente do katerih ima dostop.

Sistem določi filter poizvedbe ravno s pomočjo tabele DOMACCESS, ki mu pove, ali ima uporabnik dostop do posamezne zbirke. V indeksu vsak dokument vsebuje podatek o tem del katere zbirke je (indeksno polje BatchID), ki vsebuje kot vrednost številko replike.

Številko replike zbirke predstavlja unikatna številka, ki se določi vsaki novo kreirani zbirki – zbirke, ki so replike ena druge imajo isto številko replike. Replika je 16-mestna številka, ki vsebuje tako številke kot črke.

Tabela DOMACCESS je pravzaprav prepisan seznam dostopov posameznih zbirk, ki jih indeksiramo. S tem je omogočena hitra izgraditev filtra, ker se podatki pridobijo iz relacijske baze in ne neposredno iz posameznih zbirk. Neposredno branje seznama dostopov zbirk ob vsaki prijavi uporabnika bi povzročilo zaznavne zakasnitve prijave ter obremenitev strežnikov sistema Lotus Domino.

Tabela vsebuje o vsaki zbirki, ki jo indeksiramo, podatek o dostopu posameznega uporabnika – le eksplicitne, saj se morebitne skupine razčlenijo na uporabnike. Določenemu uporabniku, ki izvede uspešno overitev iskalnik, se najprej po tej tabeli določi dostop do posameznih zbirk oz. dokumentov v indeksu. Primer delovanja bo najbolje ponazorjen s spodnjim primerom.

Denimo, da tabela DOMACCESS vsebuje zapise, kot jih prikazuje Tabela 3.2.

Atribut REPLID	**Atribut** USRNAME	**Atribut** DBACCESS
C1257034004CC5AB	CN=Jaka Racman/O=POLICIJA	0 – (brez dostopa)
C1257034004CC5AB	CN=Primož Skale/OU=GSIT/O=POLICIJA	1 – (dostop)
C1257034004CC5AB	-Default-	1
C125790B002332CF	CN=Jaka Racman/O=POLICIJA	0
C125790B002332CF	CN=Primož Skale/OU=GSIT/O=POLICIJA	1
C125790B002332CF	-Default-	0

Tabela 3.2: Primer vsebine tabele DOMACCESS, ki določa dostop do dveh zbirk Lotus Notes.

Zapisi v tabeli določajo dostope uporabnikov do dveh zbirk, saj stolpec REPLID vsebuje le dve unikatni vrednosti – številki replike posameznih baz (...AB in ...CF). Do prve zbirke (poimenujmo jo kar AB, po zadnjih dveh črkah replike) lahko dostopajo vsi uporabniki razen uporabnika Jake Racmana, saj ima le-ta vrednost 0 v stolpcu DBACCESS. Vidimo, da imajo uporabniki, ki niso navedeni na seznamu dostopov za zbirko AB polni dostop do te zbirke (-Default- ima vrednost atributa DBACCESS postavljen na 1).

Drugače je pri drugi zbirki (poimenujmo jo CF). Vidimo, da ima do nje dostop le ena uporabnik (tj. Primož Skale), ostali pa do zbirke ne morejo dostopati, saj je privzeti dostop nedovoljen, ostali uporabniki pa imajo eksplicitno prepovedan dostop do zbirke.

S pomočjo teh podatkov bi sistem ob izvedbi poizvedbe določenih uporabnikov kreiral filter poizvedbe, ki bi določil do katerih dokumentov ima uporabnik dostop (glede na vsebino indeksnega polja `BatchID`). Seveda pa to še ni vse, saj varnostno shemo vsebuje tudi dokument, kjer se lahko določi bralce. Tudi to mora sistem ob vrnitvi zadetkov poizvedbe upoštevati. Tabela DOMACCESS torej določa le kdo ima dostop do posamezne zbirke, ne določa pa dostopov do posameznih dokumentov znotraj zbirke – to je delo tabele DOMREPL, ki jo bomo predstavili v nadaljevanju.

3.1.1.2 Tabela DOMCRED

Pred dostopom do katerekoli zbirke se morajo uporabniki overiti s strežnikom[10]. Za uspešno overitev s strežnikom se uporabnik najprej prijavi lokalno[11]. Nato se iz datoteke `user.id` (ki se izgradi ob registraciji uporabnika v imenskem imeniku Lotus Domino) prebere certifikat

[10] Razen, če strežnik dovoljuje anonimne dostope, kar pa v praksi ni pogosto.

[11] Vsi Lotus uporabniki izvajajo prijavo lokalno, kar je v nasprotju z današnjo politiko, kjer se uporabniki prijavljajo v oddaljeni sistem. Lotus Notes je bil predstavljen že leta 1989, ko omrežne povezave niso bile nekaj vsakdanjega in so se uporabniki na oddaljene sisteme povezovali le, ko so podatke replicirali – sicer so delali lokalno.

uporabnika in se le-ta primerja s certifikatom, ki ga posreduje strežnik. Če je uporabnikov certifikat soroden s strežnikovim certifikatom ima uporabnik dostop do strežnika (nadaljnji dostop pa je odvisen od pravic uporabnika na tem strežniku).

Iskalnik uporablja podatke iz te tabele, ko izvaja prijavo na posamezne strežnike, kjer se nahajajo zbirke za indeksiranje. Tabela vsebuje podatke o strežniku na katerega sistem izvede prijavo, uporabniško ime ter geslo, ter ali se prijava izvaja lokalno preko lokalno nameščenega odjemalca Lotus Notes, oddaljeno preko protokola DIIOP ali preko storitve LDAP (Tabela 3.3).

Ime atributa	**Tip**	**PK?**	**Null?**	**Opis**
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`LGNSRV`	`varchar(50)`	-	Ne	Naslov IP ali ime strežnika za prijavo
`ISLOCAL`	`varchar(1)`	-	Ne	Vrsta prijave (lokalna, oddaljena, LDAP)
`UNAME`	`varchar(30)`	-	Da	Uporabniško ime za prijavo v imenski imenik
`UPASS`	`varchar(50)`	-	Ne	Geslo za prijavo v imenski imenik

Tabela 3.3: Zgradba tabele DOMCRED, ki jo sistem uporablja za prijavo na strežnike, kjer se nahajajo zbirke za indeksiranje. Prijava na strežnik je mogoča preko lokalno nameščenega Lotus Notes odjemalca (atribut ISLOCAL je takrat postavljen na 1), preko protokola DIIOP (ISLOCAL enak 0) oz. preko protokola LDAP (ISLOCAL enak 2).

3.1.1.3 Tabela DOMDIR

Infrastruktura Lotus Domino lahko vsebuje več domen. Vsako domeno določa imenski imenik v katerem so navedeni uporabniki, skupine, strežniki ter nastavitve. V domeno je lahko povezanih več strežnikov, ki vsi vsebujejo isto repliko imenskega imenika. Strežnik je lahko del le ene domene. Strežnika, ki sta vsak v svoji domeni Lotus vsebujeta različna imenska imenika (privzeto je to datoteka `names.nsf`).

Dostop do dokumentov znotraj zbirk Lotus Notes se določi preko bralskih polj (angl. *readers fiela*) kamor se navede uporabnike, skupine in vloge, ki imajo dostop do tega dokumenta. Skupine so navedene v imenskem imeniku, vloge pa so del vsake baze (in so pomembne le za to bazo). Uporabniki so lahko del tako skupine kot vloge. Omenimo še dejstvo, da imata lahko dve različni zbirki navedeno isto vlogo, vendar z različnimi člani. Isto pa ne velja za skupino.

Podatke o vlogah in skupinah ter njihovemu članstvu se (periodično) prepiše v tabelo DOMREPL za hitrejši dostop do podatkov.

Tabela DOMDIR (zgradbo tabele prikazuje Tabela 3.4) vsebuje seznam vseh imenskih imenikov Lotus Domino do katerih ima sistem dostop (vsaj en na domeno). Pri vsakem

navedenem imeniku določimo še podatke za prijavo na strežnik (atribut DCID, ki se veže na tabelo DOMCRED); ali naziv skupine vsebuje poševnico (/), kar nam lahko prihrani nekaj časa pri indeksiranju, saj imajo le uporabniki poševnice v nazivu in tako ni potrebno vsakič preverjati ali sistem zapisuje uporabnika ali skupino (vloge lahko vsebujejo poševnico, vendar se vedno začnejo z znakom '[' in končajo z ']' – npr. `[VlogaA]`); kdaj je bil imenik nazadnje prepisan (atribut LASTINDEX), ipd.

Ime atributa	Tip	PK?	Null?	Opis
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`DDSRV`	`varchar(50)`	-	Ne	Naslov IP ali ime strežnika Lotus Domino
`DDNMS`	`varchar(20)`	-	Ne	Pot do in ime imenskega imenika Lotus Domino
`DCID`	`int`	-	Ne	Podatki za overjanje
`GRPSLH`	`varchar(1)`	-	Ne	Ali skupine vsebujejo poševnico?
`REPLID`	`char(16)`	-	Da	Številka replike imenika
`LASTINDEX`	`datetime`	-	Da	Kdaj je bil imenik nazadnje prepisan v relacijsko bazo?

Tabela 3.4: Zgradba tabele DOMDIR, ki jo sistem uporablja za prijavo na strežnike, kjer se nahajajo zbirke za indeksiranje, ter kot vir imenskih imenikov, katerih vsebina se periodično prepisuje v relacijsko tabelo DOMREPL za hitrejše določanje uporabniških prisotnosti v skupinah in vlogah. Tabela je povezana s tabelo DOMCRED, ki vsebuje podatke za overjanje na posamezni imenski imenik.

3.1.1.4 Tabela DOMREPL

Vlogo te tabele smo pojasnili že pod opisom tabele DOMDIR. Ta tabela vsebuje prepisane podatke o članstvu skupin posameznih imenskih imenikov, ter članstva vlog posameznih zbirk, ki jih indeksiramo. Tabela 3.5 prikazuje zgradbo te tabele.

3.1.1.5 Tabela DOMLOGIN

Vsebuje seznam domen (oz. imenskih imenikov) s katerimi se uporabniki pri prijavi v iskalnik lahko overijo. Ob prijavi se v navedeni imenik domene pošlje podatek o vpisanemu uporabniškemu imeni in geslu. Če imenik vsebuje uporabnika s temi podatki, se uporabniku dovoli vstop v iskalnik (in s tem se mu tudi določi ustrezne pravice).

Tabela je povezana s tabelo DOMDIR preko ključa DDID, ki vsaki domeni dodeli ustrezni imenski imenik.

Zgradbo tabele prikazuje Tabela 3.6.

Ime atributa	Tip	PK?	Null?	Opis
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`REPLID`	`char(16)`	-	Ne	Številka replike
`GRPNAME`	`char(100)`	-	Ne	Ime skupine oz. vloge
`USRNAME`	`varchar(200)`	-	Ne	Ime uporabnika

Tabela 3.5: Zgradba tabele DOMREPL, ki vsebuje prepisane podatke iz imenskih imenikov. V tabeli so zajeta članstva posameznih uporabnikov v skupinah in vlogah.

Ime atributa	Tip	PK?	Null?	Opis
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`DOMAIN`	`varchar(30)`	-	Ne	Poljubno ime domene
`DDID`	`int`	-	Ne	Povezava na imenski imenik preko katerega izvedemo overitev uporabnika

Tabela 3.6: Zgradba tabele DOMLOGIN, ki vsebuje seznam domen v katere se uporabniki lahko prijavijo. Ena domena je vezana na posamezno domeno (tj. en imenski imenik).

3.1.1.6 Tabeli MAPNOTESWWW in MAPWWWNOTES

Dostop do dokumentov v zbirkah je mogoč na dva načina - preko odjemalca Lotus Notes ali spletnega brskalnika. Sistem ob indeksiranju zapiše v indeks obe povezavi - tj. povezavo za dostop preko odjemalca in povezavo za dostop preko brskalnika. Problem se pojavi pri načinu dostopa (lokalno, oddaljeno) do baze, ki jo indeksiramo. Če dostopamo lokalno preko odjemalca Lotus Notes, potem vedno pridobimo povezavo preko odjemalca, kjer je spletna povezava prazna in obratno. To težavo smo rešili tako, da smo povezavi kopirali eno v drugo in spremenili začetni `notes://` v `http://` in obratno. Novi problem se je pojavil pri tem, da povezava Notes vrne npr. `notes://streznik@policija/...`, ki pa jo spletni brskalnik ne more odpreti. Zato v tabelah MAPNOTESWWW in MAPWWWNOTES definiramo preslikave posameznih povezav.

Ponazorimo delovanje preslikave s primerom. Preslikavi `notes://` → `http://` in `streznik@policija` → `streznik.policija.si` omogočata, da spletni brskalnik povezavo uspešno odpre saj iz povezave `notes://streznik@policija/...` naredi povezavo `http://streznik.policija.si/...` glede na ti dve preslikavi.

Tabela 3.7 prikazuje zgradbo obeh tabel. Obe tabeli bi lahko tudi združili v eno, vendar smo ju zaradi praktičnih razlogov pustili kot dve neodvisni tabeli.

Ime atributa	Tip	PK?	Null?	Opis
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`MAPFROM`	`varchar(100)`	-	Ne	Iskani niz
`MAPTO`	`varchar(100)`	-	Ne	Niz, ki nadomesti iskani niz

Tabela 3.7: Zgradba tabel MAPNOTESWWW in MAPWWWNOTES, ki vsebujeta definicije preslikav povezav do posameznih dokumentov, ki jih ob indeksiranju vrne sistem.

3.1.1.7 Tabeli SOLR in SOLRSEARCH

Ti tabeli vsebujeta naslove URL strežnikov Solr za indeksiranje in iskanje (tudi po posameznih indeksih – *core*). Sistem lahko naredimo tako, da en strežnik izvaja indeksiranje in se nato ta indeks replicira na drug (iskalni) strežnik, saj tako ne obremenjujemo iskalnega sistema, ko se izvaja indeksiranje (oz. optimizacija indeksa).

Uporabniki svoje poizvedbe pošiljajo na strežnike Solr, ki so navedeni v tabeli SOLRSEARCH. Če je v tabeli naveden več kot en strežnik, gre za porazdeljeno iskanje po več indeksih in iskalnik to tudi upošteva.

Tabela SOLR vsebuje naslove URL strežnikov kamor indekser pošlje podatke v indeksiranje. Vsak vir v tabeli SRCNOTES ima naveden strežnik Solr kamor se podatki pošljejo. Zgradbo obeh tabel prikazuje Tabela 3.8.

Ime atributa	Tip	PK?	Null?	Opis
`ID`	`int`	Da	Ne	Primarni ključ – samodejno povečevanje
`URL`	`varchar(100)`	-	Ne	Naslov URL do strežnika Solr in pripadajočega indeksa

Tabela 3.8: Zgradba tabel SOLR in SOLRSEARCH, ki vsebujeta naslove URL strežnikov Solr, ki jih sistem uporablja pri indeksiranju (tabela SOLR) in iskanju (tabela SOLRSEARCH).

3.1.1.8 Tabela SRCNOTES

Ta tabela vsebuje seznam vseh zbirk Lotus Notes, ki jih sistem indeksira (zbirka je lahko aktivna ali neaktivna). Sistem indeksira samo zbirke, ki so trenutno aktivne. Tabela vsebuje podatke o zbirki in kdaj je bila nazadnje uspešno indeksirana. Zgradbo tabele prikazuje Tabela 3.9. Tabela je povezana s tabelo SOLR (preko ključa SOLRID) in tabelo DOMDIR (preko ključev PDDID in SDDID).

Ime atributa	Tip	PK?	Null?	Opis
ID	int	Da	Ne	Primarni ključ – samodejno povečevanje
SRV	varchar(50)	-	Ne	Ime ali naslov IP strežnika
DB	varchar(200)	-	Ne	Pot do in ime zbirke
SOLRID	int	-	Ne	Referenca na strežnik Solr v tabeli SOLR kamor naj se pošljejo podatki
PDDID	int	-	Ne	Referenca na imenski imenik Lotus Domino za določanje članstva v skupinah in vlogah
SDDID	int	-	Da	Referenca na sekundarni imenski imenik Lotus Domino za določanje članstva v skupinah in vlogah
MAXDOCS	int	-	Ne	Največje število dokumentov, ki se lahko indeksirajo iz te zbirke ob vsakršnem zagonu procesa indeksiranja (v produkcijskem okolju je vrednost tega atributa po navadi nastavljena na 0 – tj. vsi dokumenti)
SKIPCONF	varchar(1)	-	Ne	Ali indekser preskoči dokumente, ki so označeni kot konfliktni?
FORCESEC	varchar(1)	-	Ne	Ali indekser upošteva bralce dokumentov (tj. vsebino bralskih polj)?
INDEXATT	varchar(1)	-	Ne	Ali indekser indeksira tudi morebitne priponke dokumentov?
REPLID	char(16)	-	Da	Številka replike zbirke

FORMS	varchar(1000)	-	Ne	Seznam imen obrazcev te zbirke, ki jih želimo indeksirati
TYPE	varchar(20)	-	Ne	Ime zbirke (unikaten glede na ostale zapise)
LASTINDEX	datetime	-	Da	Datum in ura zadnjega uspešnega indeksiranja zbirke
ACTIVE	varchar(1)	-	Ne	Ali se zbirka indeksira ali ne?

Tabela 3.9: Zgradba tabele SRCNOTES, ki vsebuje podatke o indeksiranih zbirkah.

3.1.1.9 Tabela FRMFIELDS

Tabela FRMFIELDS je namenjena definiciji vseh polj, ki jih iz posameznega obrazca zbirke (imena obrazcev so vezana na posamezno zbirko in so definirana v atributu FORMS v tabeli SRCNOTES) želimo shraniti v indeks. Glavni razlog za tako početje je v tem, da s tako obidemo potrebo po definiranju teh polj v sami programski kodi. Rešitev tako deluje univerzalno za vse zbirke ne glede na njihov namen. Upravitelj sistema v tabeli FRMFIELDS definira za vsako zbirko posebej obrazce in pripadajoča polja, ki se morajo iz zbirke prebrati in nato shraniti v indeks iskalnika.

Tabela 3.10 prikazuje zgradbo te tabele.

Ime atributa	**Tip**	**PK?**	**Null?**	**Opis**
ID	int	Da	Ne	Primarni ključ – samodejno povečevanje
SOURCE	varchar(20)	-	Ne	Ime zbirke (neposredno povezan z atributom TYPE tabele SRCNOTES)
FORM	varchar(100)	-	Ne	Ime obrazca
NOTESNAME	varchar(100)	-	Ne	Ime polja na obrazcu zbirke
SOLRNAME	varchar(100)	-	Ne	Ime polja v indeksu (definiran v datoteki schema.xml)

TYPE	varchar(20)	-	Ne	Tip polja na obrazcu (String[12], Date[13], Summary[14])

Tabela 3.10: Zgradba tabele FRMFIELDS, ki je namenjena definiciji polj, ki jih iz zbirke Lotus Notes želimo shraniti v indeks.

3.1.1.10 Tabela JSPVIEW

Tabela JSPVIEW je nasprotje tabele FRMFIELDS in definira opise in polja za vsak obrazec zbirke, ki ga prikažemo na seznamu zadetkov. V tej tabeli torej za vsako zbirko in obrazec definiramo katera polja želimo prikazati pod posameznim zadetkom in pripadajoči opis (ter zaporedje polj na zaslonu, kjer lahko definiramo tudi preskok v novo vrstico – privzeto se sicer polja prikažejo v isti vrstici).

Tabela 3.11 prikazuje zgradbo tabele JSPVIEW.

Ime atributa	**Tip**	**PK?**	**Null?**	**Opis**
ID	int	Da	Ne	Primarni ključ – samodejno povečevanje
SOURCE	varchar(20)	-	Ne	Ime zbirke (neposredno povezan z atributom TYPE tabele SRCNOTES)
FORM	varchar(100)	-	Ne	Ime obrazca
DESCR	varchar(100)	-	Ne	Opis polja
SOLRNAME	varchar(100)	-	Ne	Ime polja v indeksu (definiran v datoteki schema.xml)
CNT	int	-	Ne	Zaporedna številka prikaza

Tabela 3.11: Zgradba tabele JSPVIEW, ki je namenjena definiciji polj in pripadajočih opisov, ki se prikažejo ob posameznem dokumentu na seznamu zadetkov.

[12] Vsebina polja na obrazcu Lotus Notes zbirke se prebere kot niz znakov.

[13] Vsebina polja na obrazcu Lotus Notes zbirke se prebere kot datumsko polje.

[14] Vsebina polja na obrazcu Lotus Notes zbirke se prebere kot niz znakov v vektor.

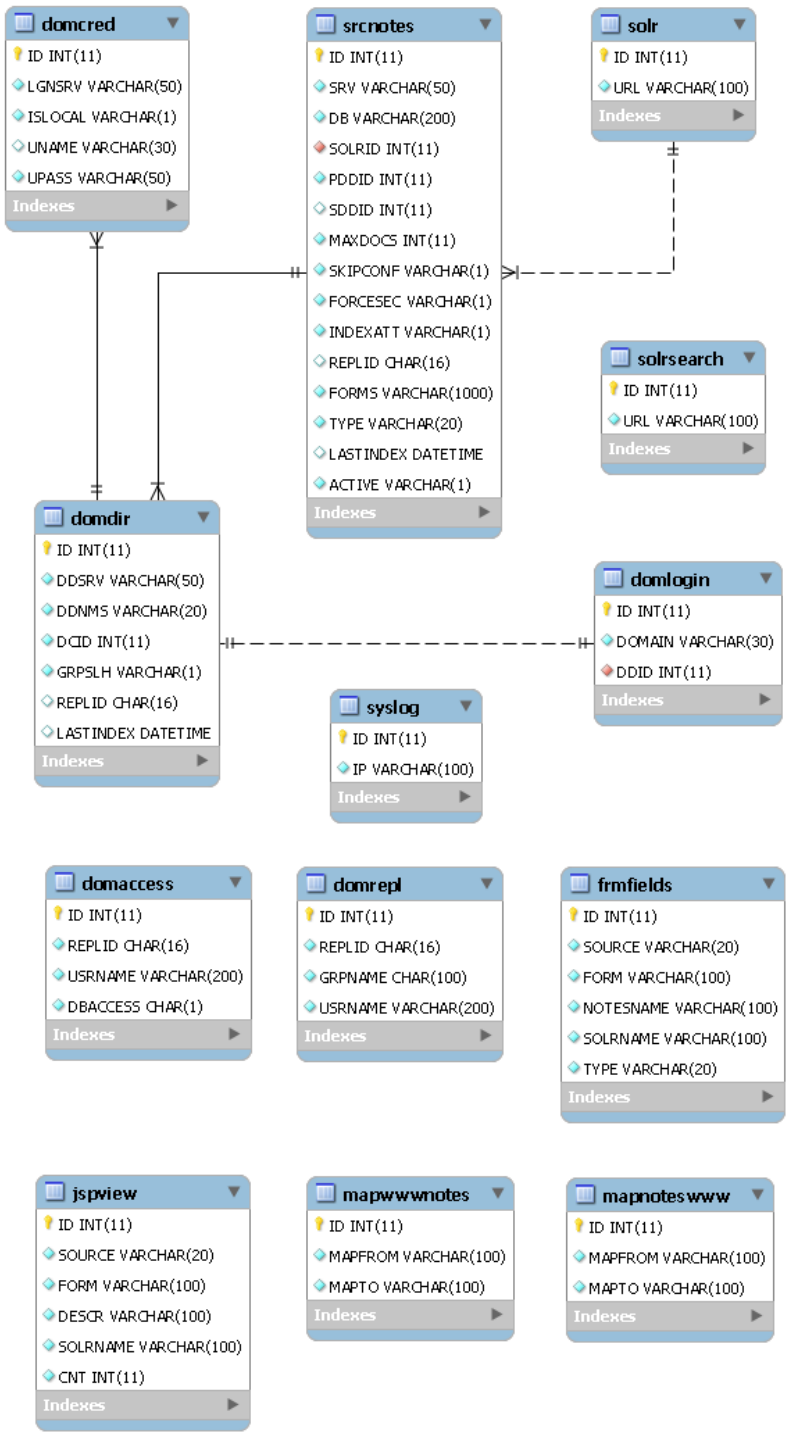
3.1.2 Povezave med tabelami

Pod opisi posameznih tabel, ki sestavljajo relacijsko bazo, ki jo uporablja iskalnik, smo že navedli, da so nekatere tabele med seboj povezane preko ključev. Tabele, ki so med seboj povezane so: SRCNOTES, DOMLOGIN, DOMCRED, DOMDIR in SOLR.

Še enkrat na kratko ponovimo povezave:

- Tabela SRCNOTES se preko ključa SOLRID povezuje s tabelo SOLR.
- Tabela SRCNOTES se preko ključev PDDID in SDDID povezuje s tabelo DOMDIR.
- Tabela DOMDIR se preko ključa DCID povezuje s tabelo DOMCRED.
- Tabela DOMLOGIN se preko ključa DDID povezuje s tabelo DOMDIR.

Slika 3.1 prikazuje diagram entitetno-relacijskega modela (E-R model), ki predstavlja logično predstavitev podatkovnega modela baze, ki jo iskalnik uporablja.



Slika 3.1: Entitetno-relacijski diagram podatkovne baze, ki jo iskalnik uporablja. Nekatere tabele so med seboj povezane, druge so neodvisne druga od druge.

3.2 Varnostni model iskalnika

Varnostni model predstavlja najpomembnejši sestavni del iskalnika in realno imitira varnostni model, ki ga implementira sistem Lotus Domino. Uporabnikom, ki izvajajo iskanje po indeksu, ne smejo biti prikazani dokumenti na seznamu zadetkov do katerih nimajo dostopa v primarni zbirki. Iskalnik varnostni model implementira preko uporabe *filtra poizvedbe* (angl. *filter query* – FQ), ki je nič drugega kot dodatna poizvedba, ki definira kateri dokumenti se vrnejo uporabnikom, brez vpliva na izračun pomembnosti dokumentov [16]. Filter poizvedbe bi se lahko uporabil tudi kot del same poizvedbe, vendar pa njegova ločena uporaba pohitri iskanje, saj se filter poizvedbe hrani v delovnem spominu strežnika in se ponovno uporabi ob izvedbi iste poizvedbe (npr. ko uporabnik želi prikazati naslednje strani zadetkov).

Primarna naloga varnostnega modela iskalnika je torej omejevanje števila vrnjenih dokumentov uporabnikom glede na njihove pravice dostopa. V razdelku 3.1.1 smo omenili, da se pravice uporabnikov zapisujejo v dve tabeli: DOMACCESS in DOMREPL.

Prva določa pravico dostopa do zbirke, druga pa določa članstvo uporabnikov v skupinah (glede na vse zbirke) in vlogah (glede na posamezno zbirko). Filter poizvedbe izgradimo s pomočjo informacij, ki jih podajata ti dve tabeli.

3.2.1 Izgraditev filtra poizvedbe

Uporabniki se morajo pred uporabo iskalnika prijaviti preko imenskega imenika Lotus Domino. Če je prijava uspešna, sistem vsakega uporabnika identificira s polnim uporabniškim imenom (npr.: »CN=Jaka Racman/OU=GSIT/O=POLICIJA«). Postopek določitve filtra poizvedbe je proces treh korakov:

- Iz tabele DOMACCESS se pridobi seznam vseh številk replik (tj. zbirk) do katerih ima uporabnik dostop.
- Iz tabele DOMREPL se pridobi seznam vseh skupin katerih član je uporabnik.
- Iz tabele DOMREPL se pridobi seznam vlog po posameznih zbirkah katerih član je uporabnik.

V nadaljevanju bomo podrobno pojasnili izvedbo vseh treh točk.

– Seznam vseh številk replik do katerih ima uporabnik dostop se izgradi tako, da najprej iz tabele DOMACCESS pridobimo seznam vseh zbirk na katerih je uporabnik naveden – na tem mestu ne upoštevamo vrednosti atributa za dostop DBACCESS. Sistem uporabi dve tabeli v kateri zapiše številko replike zbirke, če ima (`userAccess`) uporabnik dostop (DBACESS je enak 1) in če nima dostopa (`userNoAccess`). S tem smo določili seznam zbirk do katerih ima uporabnik dostop ne glede na privzetega uporabnika (tj. -Default-). V nadaljevanju preverimo še dostop do zbirk na katerih uporabnik *ni* eksplicitno oz. implicitno naveden, temveč lahko do zbirke dostopa preko privzetega uporabnika. V ta namen iz tabele DOMACCESS pridobimo

seznam številk replik zbirk do katerih ima privzeti uporabnik dostop. Iz tega seznama nato dodamo na seznam zbirk za dostop le tiste, ki niso navedene ne v tabeli `userAccess`, ne v tabeli `userNoAccess` – z drugimi besedami, če uporabnik nima navedenega dostopa do te zbirke, privzeti uporabnik pa dostop ima, to pomeni, da uporabnik do zbirke lahko dostopa.

Prvi del FQja definiramo kot:

$$\mathrm{FQ}_{\mathrm{repl}} = \mathrm{BatchID} : (stRepl_1 \text{ OR } stRepl_2 \text{ OR } \ldots \text{ OR } stRepl_N),$$

kjer elementi *stRepli* predstavljajo številke replik zbirk iz tabele `userAccess` – do teh zbirk uporabnik lahko dostopa. Polje `BatchID` je polje v indeksu, ki je del vsakega dokumenta in identificira zbirko iz katere dokument prihaja.

Za primer vzemimo vsebino Tabela 3.2, kjer upoštevamo, da se je uporabnik »Primož Skale« ravnokar prijavil v sistem. Sistem najprej za tega uporabnika pridobi seznam vseh zbirk v katerih je na seznamu dostopov uporabnik naveden. Sistem v tabelo `userAccess` zapiše vrednosti ['C1257034004CC5AB', 'C125790B002332CF']. Tabela `userNoAccess` ostane prazna. Sistem nato pridobi še dostope privzetih uporabnikov in jih primerja z ostalima dvema tabelama. V našem primeru sta definirani le dve zbirki in obe smo že dodali na seznam za dostop. Na tej točki lahko za tega uporabnika izgradimo vrednost:

$$\mathrm{FQ}_{\mathrm{repl}} = \mathrm{BatchID} : (\text{'C1257034004CC5AB' OR 'C125790B002332CF'})$$

Če bi uporabili samo tak filter poizvedbe bi uporabnik na seznamu zadetkov videl le dokumente, ki so del teh dveh zbirk, ne glede na morebitno ujemanje ostalih dokumentov iz drugih zbirk. Seveda pa tak filter ne zadostuje vseh varnostnim vidikom, saj so lahko znotraj teh dveh zbirk pred pogledom varovani tudi posamezni dokumenti. Slika 3.2 prikazuje diagram procesa, ki izvaja prvo točko na konkretnem primeru.

Izgraditev dodatnih členov FQ je v domeni naslednje točke:

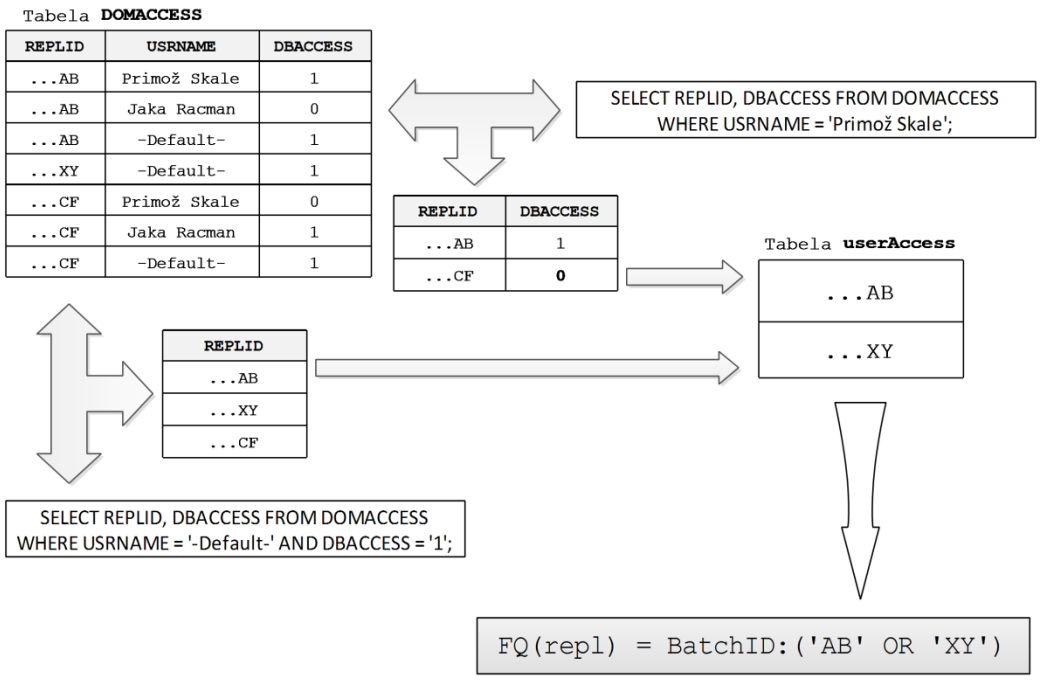
– Prejšnji korak je torej določil seznam vseh zbirk do katerih uporabnik lahko dostopa. V tej točki varnostni vidik prestavimo stopnjo nižje – na nivo posameznih dokumentov znotraj zbirk. Kot smo že omenili dokumenti lahko vsebujejo posebna polja, kjer navedemo nazive skupin, vlog ali uporabnikov, ki te dokumente lahko vidijo in posledično odprejo. Ob indeksiranju se vsebina teh polj prepiše v indeksno polje `AccessMembers`. Še enkrat pojasnimo, da so imena indeksnih polj (npr. `BatchID`, `AccessMembers`, ipd.) poljubna in bi jih lahko zapisali tudi drugače – mi smo pač vzeli take kot so.

Naloga te točke je torej izgraditev seznama vseh skupin imenskega imenika Lotus Domino katerih član je uporabnik. Sistem izvede poizvedbo na tabeli DOMREPL, kjer pridobi imena skupin (atribut GRPNAME) pod pogojem da niz ni skupina (se ne začne z znakom [in konča z znakom]) in da je uporabnik naveden v atributu USRNAME.

S temi informacijami izgradimo drugi del FQja, ki je enak:

$$\mathrm{FQ}_{\text{skupine}} = \text{AccessMembers} : (\textit{skupina}_1 \text{ OR } \textit{skupina}_2 \text{ OR } \ldots \text{ OR } \textit{skupina}_M),$$

kjer elementi $\textit{skupina}_i$ predstavljajo posamezna imena skupin katerih član je uporabnik.



Slika 3.2: Diagram procesa, ki iz tabele DOMACCESS pridobi seznam številk replik (zbirk) do katerih ima določen uporabnik dostop. V tem primeru želimo izgraditi $\mathrm{FQ}_{\text{repl}}$ za uporabnika »CN=Primož Skale/OU=GSIT/O=POLICIJA«, ki ga krajšamo v »Primož Skale«.

Poleg skupin pa globalno na dostop deluje tudi uporabniško ime – tako v polni obliki, kot v skrajšani obliki. Ponazorimo prejšnji stavek s primerom.

Vzemimo uporabnika »CN=Jaka Racman/OU=GSIT/O=POLICIJA«[15]. Če v bralsko polje na dokumentu vpišemo ta niz bo imel samo in le ta uporabnik dostop do tega dokumenta (privzamemo, da ima dostop do zbirke, kjer se dokument nahaja in da poleg tega vpisa v bralsko polje ni drugih vpisov oz. drugih bralskih polj, ki niso prazna). Če v polje nato zapišemo npr. »*/OU=GSIT/O=POLICIJA« bodo do dokumenta lahko dostopali vsi uporabniki katerih uporabniško ime se konča s tem nizom. Ali še bolj splošno, »*/O=POLICIJA«. Če želimo dovoliti dostop vsem uporabnikom lahko zapišemo le »*«[16]. Take

[15] Uporabniško ime smo namenoma zapisali v dolgi obliki.

[16] Ob indeksiranju se vrednost »*« ne zapiše v indeks, temveč se namesto nje zapiše vrednost »-All Access-«. Enako velja tudi, če so vsa bralska polja na dokumentu prazna.

oblike mora iskalnik upoštevati, zato poleg seznama skupin v $FQ_{skupine}$ dodamo še te možnosti in dobimo:

$$FQ_{skupine} = \text{AccessMembers} : (skupina_1 \text{ OR } skupina_2 \text{ OR } \ldots \text{ OR } skupina_M \text{ OR '–All Access-' OR } uporIme \text{ OR } uporIme_1* \text{ OR } \ldots \text{ OR } uporIme_k*),$$

kjer element *uporIme* predstavlja polno uporabniško ime, elementi $uporIme_i*$ pa dele uporabniškega imena glede na poševnice[17].

– V tretjem koraku določimo članstvo uporabnika v posameznih vlogah glede na zbirke do katerih ima dostop (tabela `userAccess`). Iz tabele DOMREPL sistem pridobi zapise v katerih je uporabnik naveden v atributu USRNAME in kjer se vrednost niza v atributu GRPNAME prične z znakom [in konča z znakom]. Ta korak je torej isti kot pri 2. točki, s to razliko, da sedaj želimo pridobiti seznam vlog in ne skupin. Sistem po končani poizvedbi nato shrani seznam vlog katerih član je uporabnik glede na posamezno zbirko in izgradi N (kjer je N število zbirk do katerih ima dostop in hkrati velikost tabele `userAccess`) dodatnih FQjev – enega na zbirko – v obliki:

$$FQ^i_{vloge} = \text{BatchID} : stRepl_i \text{ AND AccessMembers} : (vloga^i_1 \text{ OR } \ldots \text{ OR } vloga^i_{K_i}),$$

kjer $1 \leq i \leq N$ in K_i število posameznih vlog katerih član je uporabnik glede na i-to zbirko.

S tem smo končali izgraditev posameznih delov filtra poizvedbe, sedaj pa te dele lahko združimo v celoto kot:

$$FQ = (FQ_{repl} \text{ AND } FQ_{skupine}) \text{ OR } FQ^i_{vloge} \text{ OR } \ldots \text{ OR } FQ^N_{vloge}$$

Slika 3.3 prikazuje diagram procesov 2. in 3. točke. S tem smo definirali celotni filter poizvedbe skozi katerega sistem filtrira seznam zadetkov, ki ustrezajo uporabnikovi poizvedbi. Za konec ponovimo – filter poizvedbe se izgradi za vsakega uporabnika posebej ob njegovi prijavi v iskalnik s pomočjo informacij dveh tabel: DOMACCESS in DOMREPL. Tabela DOMACCESS se osveži ob procesu indeksiranja posamezne zbirke, tabela DOMREPL pa se periodično osvežuje glede na pogostost zagona indekserja.

Na tem mestu se poraja vprašanje, kako dobro filter poizvedbe (FQ) zagotavlja varen dostop do informacij v indeksu? Z drugimi besedami – sprašujemo se, ali lahko uporabnik izvede takšno poizvedbo, ki bo kljub uporabi filtra poizvedbe vrnila uporabniku informacijo o podatkih v zbirkah do katerih nima dostopa. Ob iskanju se namreč najprej poizvedba izvrši nad celotnim indeksom, šele nato se seznam zadetkov filtrira skozi filter poizvedbe. Uporabniku se torej končno vrnejo le tisti zadetki, ki ustrezajo tako poizvedbi kot filtru poizvedbe.

[17] Vzemimo uporabniško ime CN=Jaka Racman/OU=GSIT/OU=SRA/O=POLICIJA. Krajšanje uporabniškega imena glede na poševnice bi potekalo takole: ... OR '-All Access-' OR 'CN=Jaka Racman/OU=GSIT/OU=SRA/O=POLICIJA' OR '*/OU=GSIT/OU=SRA/O=POLICIJA' OR '*/OU=SRA/O=POLICIJA' OR '*/O=POLICIJA'.

S preprostim primerom lahko pokažemo, da uporabnik ne more dostopati do informacij do katerih nima dostopa, saj se FQ vede kot mehanizem, ki navidezno razdeli indeks na dva dela – del, ki vsebuje dokumente, ki so uporabniki vidni, in del, ki vsebuje ostale dokumente.

Uporabnik ima sicer več možnosti, da do indeksa dostopa brez uporabe filtra poizvedbe. Ena izmed možnosti je, da uporabnik razvije lastni modul za izvajanje iskanja (angl. *search handler*), ki ga nato implementira v strežnik Solr. Ta možnost zahteva neposredni dostop do strežnika in s tem posledično tudi neposredni dostop do indeksa. Ker ima v tem primeru uporabnik že dostop do indeksa, ga lahko pregleda brez omejitev z ustreznimi programskimi orodji (npr. Luke).

Druga možnost je direktna izvedba poizvedbe preko ukaza URL na strežnik Solr pri katerem pa ne podamo filtra poizvedbe kot enega izmed parametrov ukaza. Ker je izvajanje direktnih poizvedb na strežnik tipično varovano z geslom, so take poizvedbe omogočene le tistim, ki geslo za dostop poznajo.

Vzemimo indeks v katerem so skupno shranjeni trije dokument iz dveh zbirk (označene kot A in B), kot jih prikazuje spodnja tabela:

Dokument	**Zbirka**	**Podatek**
A1	A	javno
B1	B	javno
B2	B	skrito

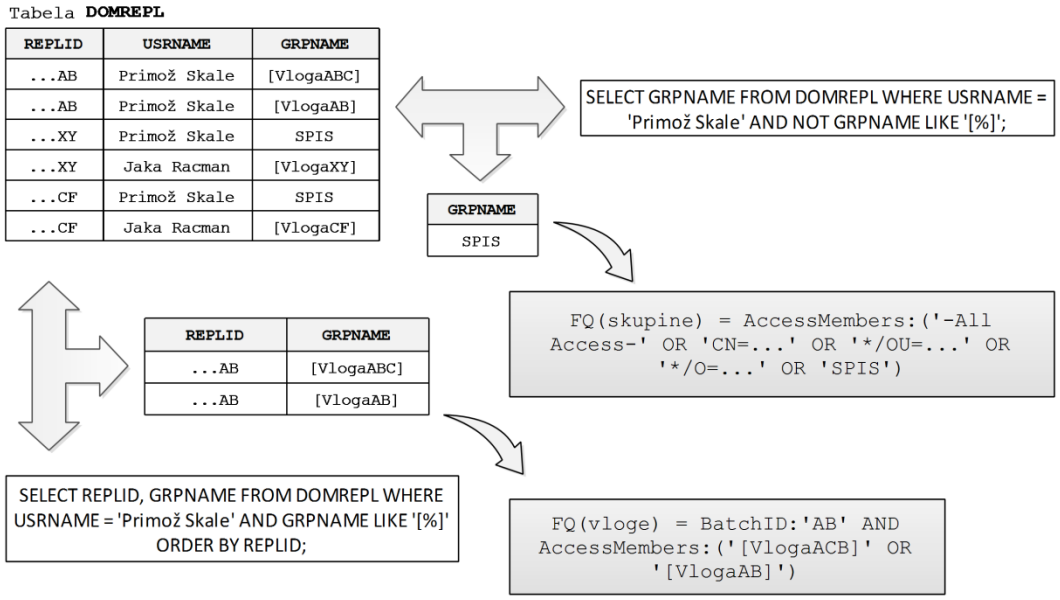
Uporabnik ima dostop samo do zbirke A in lahko zato neomejeno pregleda vse dokumente le v tej zbirki. Vzemimo, da uporabnika zanima, ali se v zbirki B nahaja podatek `javno`. Uporabnik nato vpiše poizvedbo `javno AND (zbirka:A OR zbirka:B)`. Sistem bi brez uporabe filtra poizvedbe (`zbirka:A`) vrnil seznam zadetkov na katerem bi bila dokumenta A1 in B1. Ker iskalnik poleg poizvedbe pošlje strežniku Solr tudi filter poizvedbe, seznam zadetkov v resnici vsebuje le dokument A1. Uporabnik je na ta način ugotovil, da se podatek `javno` nahaja le v zbirki A. Podobno se dogodi, če uporabnik išče podatek, ki se *ne* nahaja v zbirki A (`skrito`), vendar pa želi vedeti, ali se podatek morda nahaja v zbirki B. Če sedaj izvede poizvedbo `skrito AND (zbirka:A OR zbirka:B)`, sistem vrne prazen seznam zadetkov, saj filter poizvedbe uspešno filtrira tiste zadetke, ki sicer ustrezajo zgornji poizvedbi (dokument B2), vendar pa ne ustrezajo filtru poizvedbe.

3.2.2 Prednosti in slabosti varnostnega modela

Prednost uporabe filtra poizvedbe je ta, da le-tega lahko hitro izgradimo saj imamo vse potrebne informacije v relacijski bazi do katere hitro in učinkovito dostopamo. Filter poizvedbe je tudi neodvisen od sprememb članstva uporabnikov v skupinah oz. vlogah, ter bralskih poljih na dokumentih. Če pride do spremembe na dokumentih, bo tak dokument označen in se bo ob naslednjem indeksiranju ponovno zapisal v indeks. Enako velja tudi za članstvo v skupinah oz. vlogah. Ob indeksiranju posameznih zbirk se v tabeli DOMACCESS in DOMREPL na novo

zapiše vsebina seznama dostopov, v tabelo DOMREPL pa se prepišejo spremembe članstva skupin imenskega imenika. Posledica tega je, da se filter poizvedbe izgradi vedno enako ne glede na dinamiko spreminjajočih se dostopov. Kvaliteta oz. veljavnost filtra poizvedbe je odvisna le od tega kako pogosto izvajamo indeksiranje in kako pogosto prepisujemo imenski imenik v tabelo DOMREPL.

Slabost uporabe filtra poizvedbe je predvsem v tem, da velikost (tj. kako dolg je niz) vpliva na hitrost izvajanja poizvedbe. Dolžina niza filtra poizvedbe je odvisna od treh dejavnikov: števila zbirk do katerih ima uporabnik dostop, števila skupin in števila vlog na posamezno zbirko, katerih član je uporabnik.



Slika 3.3: Diagram procesa, ki iz tabele DOMREPL pridobi seznam skupin in vlog katerih član je uporabnik. V tem primeru želimo izgraditi FQ_{skupine} in FQ_{vloge} za uporabnika »CN=Primož Skale/OU=GSIT/O=POLICIJA«, ki ga krajšamo v »Primož Skale«.

3.2.3 Druge izvedbe varnostnega modela

Zgoraj opisani primer varnostnega model pa ni edini možni način le-tega. Prvotno smo varnostni model implementirali brez uporabe tabel DOMACCESS in DOMREPL na način, da smo seznam dostopov zbirk in bralska polja posameznih dokumentov razširili tako, da smo v indeksno polje `AccessMembers` zapisali samo uporabniška imena – tj. če je bila v bralskem polju navedena skupina ali vloga smo morali najprej pridobiti seznam članstva le-te in ta seznam nato dodati v indeksno polje. Tak način sicer poenostavi oz. skrajša dolžino filtra poizvedbe, ki bi bil v tem primeru enak:

FQ = AccessMembers : ('-All Access-' OR *uporIme* OR *uporIme*$_1$* OR ... OR *uporIme*$_k$*)

FQ sedaj ne vsebuje informacij o skupinah ter vlogah, saj te niso več potrebne.

Slabost uporabe takega varnostnega modela leži v tem, da po nepotrebnem povečuje velikost indeksa. Ponazorimo to s primerom. Vzemimo poljubno zbirko, ki vsebuje milijon dokumentov. Vsak izmed teh ima v bralskem polju navedeno skupino, ki vsebuje 500 članov. V tem primeru bo v indeksnem polju `AccessMembers` vsakega dokumenta zapisanih 500 uporabnikov, kar glede na število dokumentov pomeni 500 milijonov zapisov v poljih `AccessMembers`. Če sedaj tak model primerjamo z modelom iz razdelka 3.2.1 vidimo, da bi pri prvem v polje `AccessMembers` zapisali le ime skupine (skupno milijon zapisov) in članstvo skupine v tabelo DOMREPL (skupno 500 zapisov). Razmerje zapisov v indeksu je torej 1:500, kar pomeni občuten prihranek prostora, ki ga zaseda indeks – posledično to pomeni, da lahko v določenih primerih celotni indeks sistem hrani v delovnem spominu in s tem poveča hitrost izvedbe poizvedb.

Druga slabost uporabe takega varnostnega modela je v tem, da ni prilagodljiv hitro spreminjajočim se pravicam dostopov. Ker poleg vsakega dokumenta v indeksu zapišemo združene pravice seznama dostopov in bralskih polj na dokumentu pomeni, da že sprememba v seznamu dostopov povzroči, da mora sistem ponovno izvesti popolno indeksiranje vseh dokumentov v zbirkah, kjer se je seznam dostopov spremenil. Odveč je omeniti, da je tak postopek ne samo zamuden, temveč tudi izredno obremenjuje sistem. Tak poenostavljen varnostni model je torej najprimernejši za okolja, kjer se pravice dostopov ne spreminjajo pogosto oz. sploh ne.

3.3 Zajem podatkov iz zbirk Lotus Notes

Podatke iz zbirk Lotus Notus je možno pridobiti na dva načina: i) preko uporabe knjižnic DLL (angl. *dynamic link libraries*), in ii) preko uporabe knjižnic JAR (angl. *Java ARchive*). Naša rešitev uporablja slednjo možnost, saj se je izkazala za preprostejšo.

Zgradba procesa pridobivanja podatkov (oz. indeksiranja) je preprosta in sestoji iz dveh glavnih delov:

- Pridobitev vsebine ustreznih dokumentov iz zbirke.
- Zapis vsebine v primerno obliko za namen prenosa dokumenta v indeks.

– Prvi korak sestoji iz pridobivanja seznama vseh aktivnih zbirk za indeksiranje iz tabele SRCNOTES. Proces za vsako zbirko prebere zadnji datum indeksiranja (atribut LASTINDEX) in nato nad vsako zbirko izvede iskanje ustreznih dokumentov. Če je datum zadnjega indeksiranja neobstoječ (tj. `null`) se iz zbirke pridobijo vsi dokumenti, ki so bili ustvarjeni z ustreznimi obrazci (atribut FORMS). Če je datum zadnjega indeksiranja vpisan, se iz zbirke

pridobijo vsi novo kreirani oz. spremenjeni dokumenti na in po tem datumu, ne glede na obrazec s katerim so bili ustvarjeni. Razlog v različnem pridobivanju dokumentov (enkrat se preverja obrazec s katerim so bili dokumenti ustvarjeni, drugič ne) je v tem, da je proces pridobivanja novo kreiranih oz. spremenjenih dokumentov po nekem datumu hitrejši od pridobivanja istih dokumentov po tej metodi, kot po prvi. V nadaljevanju se tiste dokumente, ki niso bili ustvarjeni z želenimi obrazci, preprosto zavrže. Ker se proces indeksiranja celotne zbirke izvaja redko (v realnem okolju morda enkrat letno), lahko dopustimo, da se seznam dokumentov pridobiva dalj časa – npr. 5 minut. Izvajanje inkrementalnega indeksiranja pa zahteva hitro pridobitev spremenjenih dokumentov – npr. najkasneje v 10 sekundah.

– Ko proces ustvari seznam dokumentov, ki jih mora dodati v indeks, je potrebno pridobiti vsebino dokumentov in pripadajočih priponk. Za vsak dokument zbirke Lotus Notes se ustvari pripadajoči dokument `SolrDocument`, kamor se v ustrezna polja prepiše vsebina. Nekatera polja so vedno del dokumenta vstavljenega v indeks (ime obrazca s katerim je bil dokument ustvarjen, datum kreiranja dokumenta, ipd.), druga polja so določena preko vnosov v tabeli FRMFIELDS. Če nastavitve določajo, da se indeksirajo tudi morebitne priponke, ki so del dokument zbirke, se njihova vsebina pridobi s pomočjo odprto-kodne rešitve Apache Tika, ki zmore pridobiti vsebino (v tekstovni obliki) več kot 200 različnih oblik datotek.

– Ko je dokument pripravljen za prenos v indeks, ga proces prenese v ustrezni indeks in nadaljuje z naslednjim dokumentom na seznamu. Ko so v indeks dodani vsi dokumenti s seznama, proces sporoči strežniku Solr naj zaključi proces indeksiranja in s tem so novo dodani dokumenti na voljo za iskanje. Proces prav tako v tabelo SRCNOTES poleg ustrezne zbirke zapiše trenutni datum v atribut LASTINDEX in nadaljuje na naslednjo zbirko na seznamu, kjer se ves proces pridobivanja vsebine dokumentov in prenos le-teh v indeks ponovi.

3.4 Izvajanje poizvedb

Iskalnik izvaja poizvedbe tako, da preko vmesnika API komunicira s strežnikom Solr. Iskanje poteka podobno kot smo že prikazali v razdelku 2.3.2.2, le da strežniku pošljemo več parametrov – npr. filter poizvedbe (FQ), število zadetkov na eni strani, katero stran zadetkov želimo prikazati, ipd.

Iskalnik omogoča dva načina iskanja: osnovno in napredno iskanje. Pri osnovnem iskanju ima uporabnik možnost vpisa proste poizvedbe in izbiro polja po katerem se ureja seznam zadetkov. Pri naprednem iskanju uporabnik določi datumski interval nastanka dokumentov. Uporabnik ima tako možnost prikazati le dokumente, ki so bili ustvarjeni[18] v določenem datumskem intervalu in ustrezajo vpisani poizvedbi. Napredno iskanje se izvrši tako, da se izbrana datuma zapiše v ustrezno obliko in se jo doda v filter poizvedbe. Če je uporabnik zahteval le dokumente, ki so bili ustvarjeni v letu 2012 (tj. od 1. jan. 2012 do 31. dec. 2012) bi sistem izgradil sledečo poizvedbo:

[18] Mišljeno v primarnem viru podatkov (npr. zbirkah Lotus Notes), ne v indeksu.

```
Date:[2012-01-01T00:00:00Z TO 2012-12-31T23:59:59Z]
```

Če uporabnik vnese le končni datum (npr. 20. jun. 2012) bi bila poizvedba takšna:

```
Date:[* TO 2012-06-20T23:59:59Z]
```

V primeru vpisa le začetnega datuma (npr. 20. jun. 2012) pa bi bila poizvedba takšna:

```
Date:[2012-06-20T00:00:00Z TO NOW]
```

Spodnji primer prikazuje izvajanje porazdeljenega iskanja, kjer želimo prikazati 5. stran zadetkov (10 zadetkov na stran). Iskanje poteka po treh indeksih hkrati (glej parameter `shards`). Datum nastanka dokumentov omejimo na leto 2012, po katerem tudi urejamo seznam zadetkov v padajočem vrstnem redu. Ta primer torej prikaže zadetke 50 – 59 (če seveda obstaja tako vsaj tolikšno število dokumentov v indeksu, sicer se za izbrano stran vrne prazen seznam zadetkov).

```
// Ustvarimo povezavo s strežnikom Solr
HttpSolrServer solr = new HttpSolrServer("http://localhost:8983/core0");

// Kreiramo objekt poizvedbe
SolrQuery query = new SolrQuery();
// Nastavimo poizvedbo
query.setQuery("*:*");

// Dodamo polje za ureditev zadetkov
query.addSortField("Date", SolrQuery.ORDER.desc);
// Dodamo datumski interval
query.addFilterQuery("Date:[2012-01-01T00:00:00Z TO 2012-12-31T23:59:59Z]");
// Seznam vrnjenih polj
query.setField("*, score");
// Porazdeljeno iskanje
query.setParam("shards",
         "localhost:8983/core0,localhost:8983/core1,streznikX:8983/core0");
// 5. stran zadetkov --> 5 * rows
query.setStart(50);

// Število zadetkov na eno stran
query.setRows(10);

// Izvedemo poizvedbo
QueryResponse response = solr.query(query);

// Pridobimo seznam dokumentov
SolrDocumentList hits = response.getResults();
```

3.4.1 Porazdeljeno iskanje

Izvajanje porazdeljenega iskanja je preprosto, saj pomeni le nastavitev dodatnega parametra (`shards`) pri izvajanju poizvedb. Porazdeljeno iskanje pomeni iskanje po več indeksih hkrati, ki se lahko nahajajo na istem strežniku ali pa so porazdeljeni po več strežnikih. Iskalnik informacijo o lokaciji indeksov po katerih naj izvaja iskanje najde v tabeli SOLRSEARCH (razdelek 3.1.1.7). Če tabela vsebuje več kot en zapis, gre za porazdeljeno iskanje. Sistem v zgoščeni obliki hrani v delovnem spominu vrednost parametra `shards` in ga pošlje strežniku ob vsaki izvedeni poizvedbi. Če tabela SOLRSEARCH vsebuje le en zapis, parametra `shards` ni potrebno posredovati strežniku Solr, saj zapis v tej tabeli definira strežnik na katerega pošljemo poizvedbo.

Sistem neodvisno od tega ali gre za iskanje po enem indeksu ali več, vedno pošlje poizvedbo na strežnik, ki je naveden v prvem zapisu tabele SOLRSEARCH. Če ta strežnik ni dosegljiv, se poizvedba ne more izvesti (četudi so ostali strežniki dosegljivi). V ta namen je potrebno strežnike združevati v gruče (angl. *clustering*) in s tem poskrbeti za nemoteno delovanje celotnega sistema.

Pogoj za izvajanje porazdeljenega iskanja je ta, da morajo imeti vsi dokumenti definirano polje za unikatni ključ, ki je shranjeno v indeks (nastavitev *stored*='true' v datoteki `schema.xml`). Ključi dokumentov naj bi bili unikatni za celotno množico indeksov po katerih iščemo. Če iskalnik najde dva različna dokumenta z istim unikatnim ključem, ju bo sicer postavil na seznam zadetkov, vendar pa je obnašanje takega iskanja nepredvidljivo (ni nujno, da bi ista poizvedba kratek čas kasneje vrnila enak vrstni red zadetkov).

Vrednost parametra `shards` je definirana kot:

<naslov_IP_strežnika_Solr_1>:<št_vrat_1>/<ime_indeksa_1>[,*<naslov_IP_strežnika_Solr_n>:<št_vrat_n>/<ime_indeksa_n>*]*

Nujen je le prvi del, sledi pa poljubno število ostalih strežnikov Solr.

Omenimo še kratko zanimivost. Porazdeljeno iskanje lahko v določenih primerih povzroči, da strežnik Solr preneha izvajati poizvedbe. To se dogodi v primeru, ko je strežnik HTTP na katerem teče Solr nastavljen tako, da je število možnih niti, ki prejemajo vhodne zahteve manjše od števila vseh možnih vhodnih zahtev. Ponazorimo zadnji odstavek s kratkim primerom.

Vzemimo dva strežnika HTTP, ki sta nastavljena tako, da lahko naenkrat prejmeta le *eno* vhodno zahtevo. Recimo, da oba strežnika istočasno prejmeta zahtevo po izvršitvi porazdeljenega iskanja, ki želi preiskati indeksa obeh strežnikov hkrati. Ko prvi strežnik (strežnik A) prejme vhodno zahtevo za iskanje, jo dodeli v obdelavo edini niti za vhodne zahteve, ki je tako zasedena. Strežnik A torej ne more prejeti nove vhodne zahteve za iskanje, dokler se nit ne sprosti (tj. se izvede iskanje do konca). Strežnik A nato pošlje zahtevo za iskanje drugemu strežniku (strežnik B), ki pa ima edino možno nit za obdelavo že zasedeno zaradi obdelave porazdeljene zahteve za iskanje, ki jo je prejel istočasno kot strežnik A. Ker si

strežnika ne moreta odgovoriti, uporabniki, ki so zahtevali izvedbo porazdeljene poizvedbe, ne prejmejo seznama zadetkov. Strežnika preprosto čakata drug na drugega na odgovor, ki pa nikoli ne pride, ker ni na voljo dovolj niti za obdelavo vhodnih zahtev.

3.4.2 Urejanje seznama zadetkov

Uporabniki imajo možnost urejanja seznama zadetkov po: pomembnosti, času nastanka dokumenta in času spremembe dokumenta. Urejanje seznama zadetkov poteka tako, da poleg poizvedbe strežniku Solr posredujemo ime indeksnega polja po katerem želimo urejati seznam in smer urejanja (naraščajoče, padajoče). Privzeto je seznam zadetkov urejen po pomembnosti (polje `score`, ki se kreira dinamično) v padajočem vrstnem redu (dokumenti, ki so ocenjeni najbolje so pri vrhu).

Čas nastanka in spremembe dokumenta označuje čas, ko je bil dokument ustvarjen oz. spremenjen v zbirki Lotus Notes. Slabost take izbire je v tem, če v zbirki Lotus Notes naredimo kopijo dokumentov, stare pa izbrišemo[19]. Dokumenti so fizično isti, sistem pa nato javlja, da so bili ustvarjeni kasneje kot v resnici. Temu se lahko ognemo, če definiramo drugo polje, ki ga sistem uporablja za urejanje seznama zadetkov.

[19] To se lahko dogodi pri prehodu iz ene različice aplikacije v drugo, kjer aplikacija zaradi novega zapisa naredi kopijo obstoječih dokumentov in stare pobriše.

4 Uvedba v realno okolje

V tem razdelku bomo predstavili uvedbo iskalnika v realno okolje, kjer bomo predstavili:

- Funkcionalne in nefunkcionalne zahteve, ki jim sistem mora zadostiti
- Fizično in logično arhitekturo sistema
- Akterje, ki izvajajo interakcijo s sistemom
- Diagrame aktivnosti uporabnikov v vmesniku iskalnika

4.1 Zahteve sistema

Sistem za iskanje po podatkih policije mora zadostovati določitvam zahtev, ki jih tak sistem nudi skrbnikom in uporabnikom.

V projekt je vpletenih večje število ljudi: razvijalci, »beta« uporabniki oz. preizkuševalci, načrtovalci ter končni uporabniki. Končni uporabniki bodo uslužbenci policije (administratorji ter kriminalisti).

Funkcionalne zahteve sistema so:

- Prijava in overitev uporabnika (prijava se preveri preko že obstoječega imenskega imenika Lotus Domino).
- Določitev filtra poizvedbe uporabnika ob prijavi glede na pravice, ki jih ima v okolju Lotus.
- Vloge uporabnikov: upravitelj sistema, splošni uporabnik.
- Upravitelju sistema je potrebno omogočiti spreminjanje nastavitev in delovanje iskalnega sistema.
- Uporabniku je potrebno omogočiti nemoteno izvajanje poizvedb, kreiranje periodičnega obveščanja o novih zadetkih, ter 24-urno dostopnost do sistema.
- Uporabnik vidi zadetke v skladu z varnostno shemo.

Nefunkcionalne zahteve sistema so:

- Sistem temelji na odprto-kodnih rešitvah Apache Lucene in Apache Solr.
- Razvojni jezik je (Oracle) Java.
- Sistem je nameščen na aplikacijskem strežniku Apache Tomcat (ali IBM WebSphere Application Server).
- Nastavitve uporabnikov ter njihove pravice se shranjujejo v relacijsko bazo MySQL (ali IBM DB2).
- Operacijski sistem je Microsoft Windows ali poljubna ustrezna distribucija sistema Linux.

- Interakcija sistema z uporabniki in skrbniki poteka preko brskalnika (Internet Explorer, Firefox, Opera, …).
- Sistem je razširljiv, saj je mogoče dodati nov iskalni oz. indeksni strežnik, ki se nato povezuje z ostalimi. Uporabniki lahko izvajajo porazdeljeno iskanje po več iskalnih strežnikih hkrati.
- Dosegljivost sistema – sistem mora biti dosegljiv 24 ur na dan. To bo omogočeno s pravilnim načrtovanjem in postavitvijo strežnikov.
- Zanesljivost sistema – zaradi že vgrajene funkcionalnosti replikacije ter izvajanja varnostih kopij, sistem nudi visoko zanesljivost delovanja.
- Uporabnost – sistem omogoča uporabo več uporabnikom hkrati brez upočasnitve izvajanja poizvedb, saj se poizvedbe lahko porazdeli na več iskalnih strežnikov.

4.2 Identifikacija uporabnikov oz. akterjev

Akterji so vsi uporabniki, ki neposredno izvajajo dejavnosti na sistemu – bodisi preko uporabe iskalnika, bodisi preko izvajanja skrbništva nad sistemov. Akterje lahko strogo ločimo v več vrst:

- Upravitelj imenskega imenika Lotus Domino:
    - Skrbi za dodajanje, urejanje in brisanje uporabnikov iz imenika.
- Upravitelj spletnega strežnika:
    - Skrbi za nemoteno delovanje aplikacijskega/spletnega strežnika.
    - Skrbi za namestitev sistema in nastavitve.
- Upravitelj iskalnega strežnika Apache Solr:
    - Skrbi za pravilno nastavitev iskalnega strežnika Solr.
    - Skrbi za posodobitve shem.
    - Skrbi za varnostne kopije indeksov.
    - Skrbi za učinkovito delovanje indeksiranja podatkov ter stanje trenutnega indeksa.
- Upravitelj podatkovne baze:
    - Skrbi za pravilno nastavitev podatkovne baze.
    - Skrbi za ustvarjanje in urejanje podatkovnih baz in tabel.

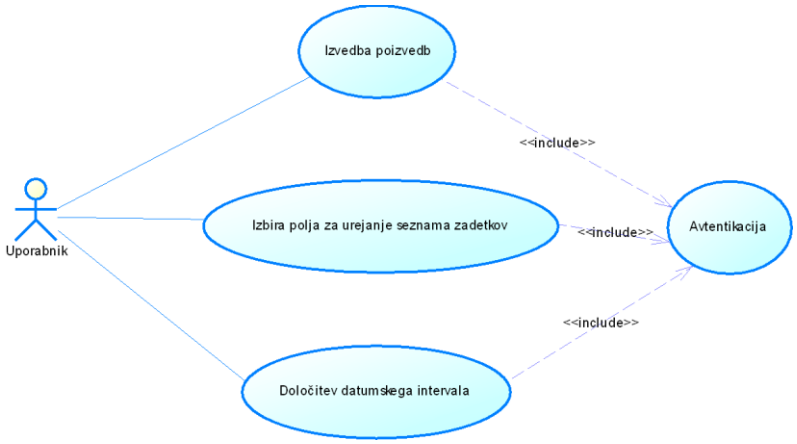
- Skrbi za integriteto podatkov.
- Skrbi za varnostne kopije in replikacijo baze.

- Uporabniki:

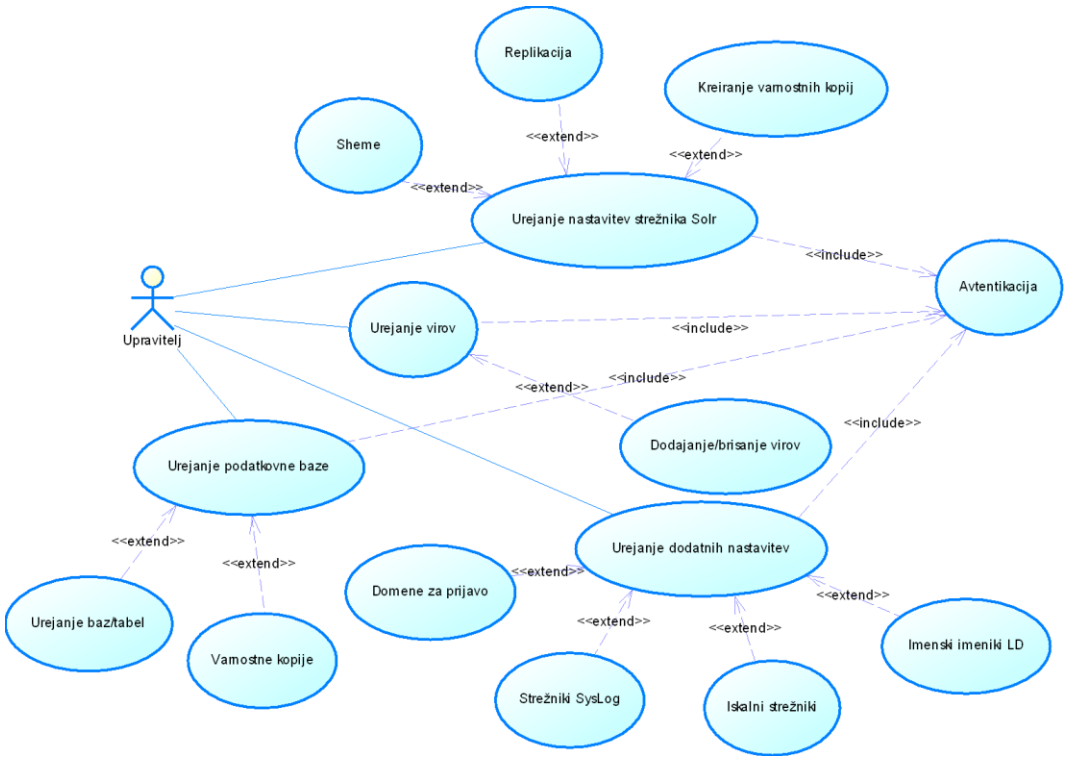
  - Izvajajo prijavo v sistem.
  - Poizvedujejo po informacijah s pomočjo vpisanih poizvedb.

Zgornja delitev prikazuje ločitev upraviteljev na strogo ločena področja upravljanja sistema. Realno pa se akterje deli le na dve vrsti: upravitelje in uporabnike. V tem primeru upravitelji skrbijo za pravilno in stalno delovanje sistema, uporabniki pa izvajajo poizvedbe nad podatki v indeksu.

4.2.1 Diagrami uporabe sistema



Slika 4.1: Diagram uporabe iskalnika, ki jo izvajajo uporabniki.



Slika 4.2: Diagram uporabe sistema, ki jo izvajajo upravitelji.

4.2.2 Diagrami aktivnosti prijave uporabnika

Uporabniki, ki želijo izvajati poizvedbe v iskalniku se morajo predhodno prijaviti. Sistem vpisana uporabniško ime in geslo preveri v izbranem (preko domene) imenskem imeniku Lotus Domino. Potek prijave je prikazan na Slika 4.3.

Potek prijave v iskalnik poteka takole:

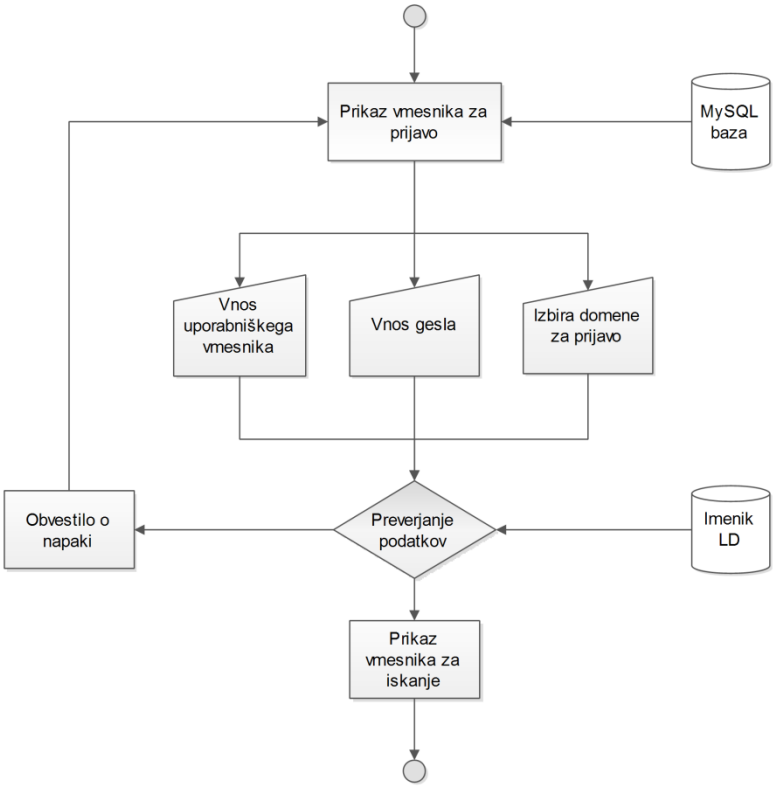
1. Pred odprtjem strani za prijavo, sistem iz relacijske baze prebere vse domene v katere se uporabniki lahko prijavijo.

2. Uporabniku se prikaže prijavna stran (Slika 4.4), kjer uporabnik vnese svoje uporabniško ime in geslo ter izbere domeno za prijavo.

3. Uporabnik pritisne gumb Prijavi.

4. Sistem pošlje prijavne podatke strežniku, ki v izbrani domeni (tj. imenskem imeniku Lotus Domino) preveri pravilnost vnesenih podatkov.

5. Če uporabnik v izbranem imeniku obstaja, mu sistem dovoli vstop v iskalnik, sicer ga z opisom napake vrne na prijavno stran.



Slika 4.3: Diagram prijave uporabnika v iskalnik. Pred izvajanjem poizvedb je nujna uspešna prijava.

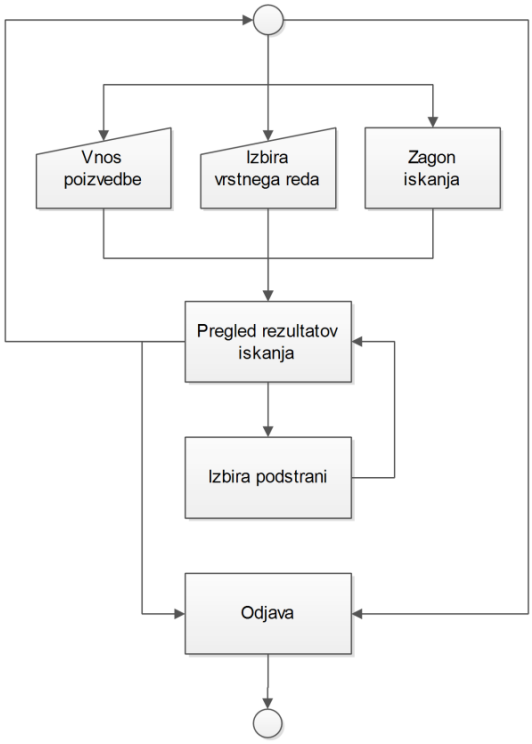
Slika 4.4: Prijavna stran. Pred izvajanjem poizvedb se morajo uporabniki prijaviti.

4.2.3 Diagram aktivnosti uporabnika v vmesniku za iskanje

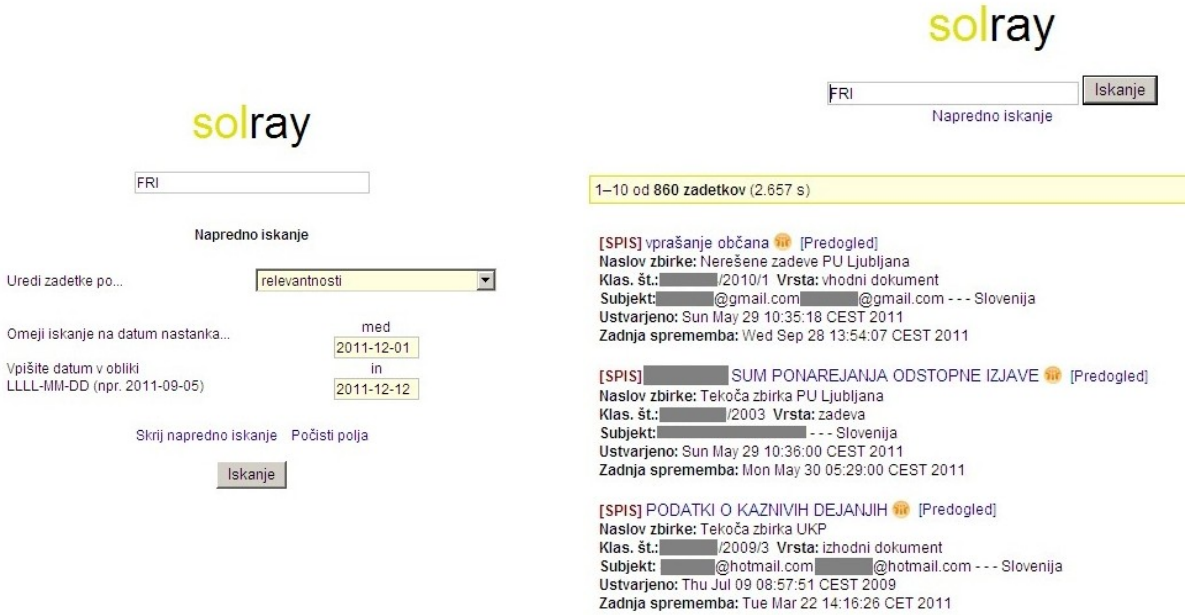
Po uspešni prijavi, se uporabniku odpre stran za izvajanje poizvedb. Na tej strani uporabnik v vnosno polje vnese iskalni niz in pritisne gumb *Išči*. Rezultate poizvedbe lahko sistem uredi glede na relevantnost oz. po datumu zadnje spremembe. Slika 4.5 in Slika 4.6 prikazujeta stran za iskanje ter diagram aktivnosti, ki jih uporabnik lahko izvaja.

Slika 4.5: Stran za iskanje, ki se prikaže uporabniku po uspešni prijavi v sistem. Uporabnik lahko izvede preprosto in napredno iskanje, urejanje zadetkov in odjavo iz sistema.

Uporabnik ima po prikazu vmesnika za iskanje dve možnosti: i) vnos poizvedbe, ii) izbiro naprednejših možnosti iskanja. Za izvajanje iskanja uporabnik vnese poizvedbo, izbere parameter za urejanje rezultatov iskanja (Slika 4.7 – levo), ter pritisne gumb *Išči*. Ko strežnik Solr konča iskanje vrne rezultate iskanja ((Slika 4.7 – desno), ki jih isti vmesnik prikaže (privzeto po 10 na stran). Uporabnik rezultat iskanja pregleda, izbere določen vpis ali nadaljuje na naslednjo podstran rezultatov.



Slika 4.6: Diagram aktivnosti uporabnika v vmesniku za iskanje.



Slika 4.7: Napredno iskanje in rezultat iskanja oz. poizvedbe. Napredno iskanje omogoča urejanje rezultata poizvedbe po različnih parametrih (relevantnosti, datumu kreiranja, datumu zadnje spremembe, itd.). V rezultatu je navedeno število vseh zadetkov, čas trajanja poizvedbe ter sami zadetki (*Opomba*: Nekateri podatki so zaradi varovanja podatkov zamaskirani).

4.3 Fizična arhitektura

Slika 4.8 prikazuje fizično arhitekturo sistema. Sistem sestoji iz treh celot: strežnikov Lotus Domino, kjer se nahajajo zbirke, ki jih sistem indeksira; aplikacijskih strežnikov HTTP in Solr; ter strežnikov, kjer se nahaja podatkovna baza za potrebe iskalnika. Slika prikazuje splošno fizično arhitekturo sistema, kjer je deloma upoštevana redundanca sistemov (podvojena strežnika za podatkovno bazo, podvojeni oz. potrojeni aplikacijski strežniki, ipd...). Za doseganje popolne zanesljivosti in dosegljivosti sistema bi bilo potrebno upoštevati tudi več fizičnih vhodnih točk v iskalnik, ter možnost nahajanja strežnikov na različnih oddaljenih lokacijah. Taka postavitev tudi omogoča porazdelitev poizvedb na različne strežnike (vozlišča) in s tem manj obremenjeno omrežje. Uporabniki, ki so fizično bližje enemu vozlišču bodo avtomatično dostopali samo do tega vozlišča (seveda pod pogojem, če je sistem pravilno postavljen, ter če je dosegljiv), drugi spet k drugemu vozlišču. Vsa vozlišča so med seboj enakovredna in podatki med njimi se periodično replicirajo (indeks, vsebina podatkovne baze, ipd...).

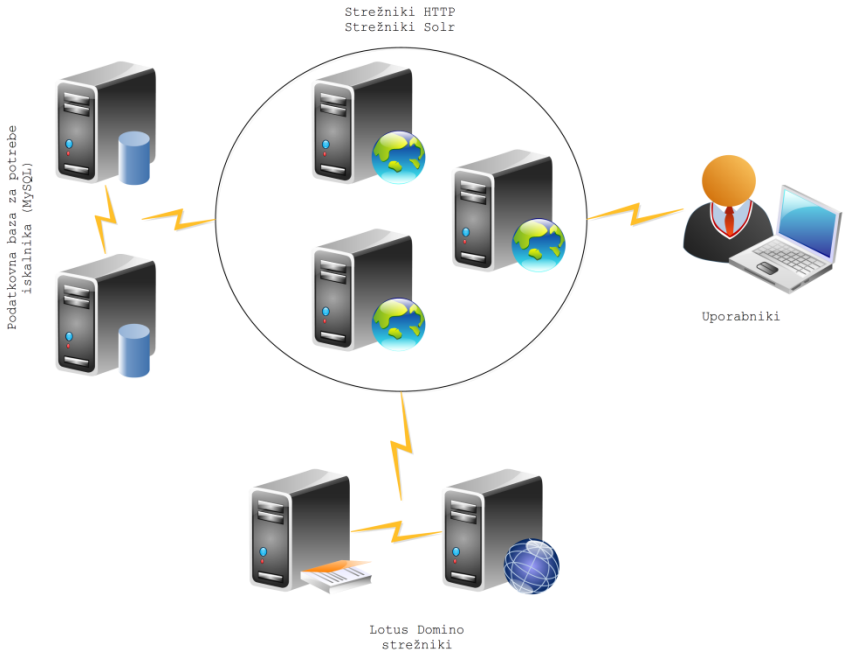
Sistem, ki je trenutno v uporabi v policiji je preprostejši. Sestoji iz dveh delov: strežnika na katerem tečejo aplikacijski strežnik HTTP, strežnik Solr in strežnik podatkovne baze, ter ločenih strežnikov Lotus Domino. S tem smo omogočili preprostejše upravljanje s sistemom na račun morebitne slabše dosegljivosti sistema. Ker na strežniku teče le en proces aplikacijskega strežnika HTTP in strežnika Solr, lahko napaka v delovanju le-teh povzroči začasno nedosegljivost sistema. Rešitev pogosto zahteva ponovni zagon fizičnega oz. aplikacijskega strežnika.

Sistem je za dostope iz zunanjega omrežja ustrezno varovan in do iskalnika lahko dostopajo le uslužbenci policije oz. tisti, ki so del policijskega informacijskega sistema.

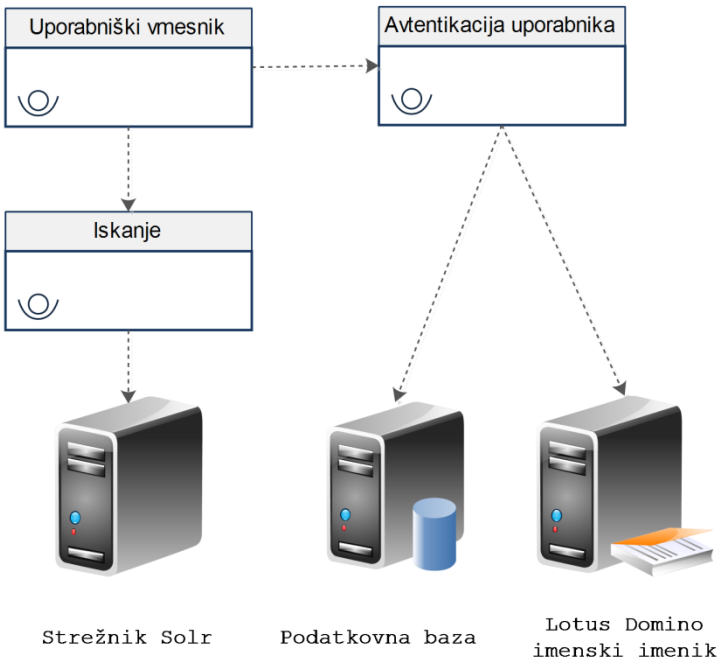
4.4 Logična arhitektura

Slika 4.9 prikazuje logično arhitekturo sistema. Uporabniki morajo pred uporabo iskalnika v uporabniškem vmesniku izvesti prijavo, ki poteka preko imenskega imenika Lotus Domino. Po uspešni prijavi jim je omogočen dostop do iskalnega uporabniškega vmesnika. Poizvedbe, ki jih uporabniki izvajajo se preusmerijo na iskalni[20] strežnik Solr, kateri vrne seznam zadetkov uporabnikom nazaj na uporabniški vmesnik. Seznam zadetkov se filtrira preko filtra poizvedbe (FQ), ki se izgradi za vsakega uporabnika posebej ob njegovi uspešni prijavi v sistem s pomočjo podatkov v podatkovni bazi.

[20] Posebej smo poudarili iskalni, saj sistem lahko sestoji iz več strežnikov Solr, kjer so eni namenjeni le indeksiranju podatkov (definirani v tabeli SOLR), drugi pa za izvajanje iskanja po indeksu (definirani v tabeli SOLRSEARCH).



Slika 4.8: Fizična arhitektura sistema. Sistem sestoji iz treh celot: strežnikov Lotus Domino, kjer se nahajajo zbirke, ki jih sistem indeksira; aplikacijskih strežnikov HTTP in Solr; ter strežnikov, kjer se nahaja podatkovna baza za potrebe iskalnika. Sistemi so med seboj lahko še dodatno porazdeljeni tako, da jih ločimo še glede na lokacijo in s tem omogočimo še boljšo zanesljivost sistema. Sistem, ki je trenutno v uporabi v policiji je preprostejši. Sestoji iz dveh delov: strežnika na katerem tečejo aplikacijski strežnik HTTP, strežnik Solr in strežnik podatkovne baze, ter ločenih strežnikov Lotus Domino. S tem smo omogočili preprostejše upravljanje s sistemom na račun morebitne slabše dosegljivosti sistema.



Slika 4.9: Logična arhitektura sistema. Uporabniki pred vstopom v iskalnik izvedejo prijavo v sistem (sistem uporabnika overi s pomočjo imenskega imenika Lotus Domino). Poizvedbe uporabnikov se pošljejo na iskalni strežnik Solr, ki nato uporabniku vrne rezultat poizvedbe. Podatkovna baza vsebuje pravice uporabnikov in članstvo v skupinah ter vlogah za vsak vir posebej.

5 Zaključek

V magistrskem delu smo opisali delovanje sistemov za izbiranje in iskanje informacij in predstavili načrtovanje in razvoj iskalnika po zbirkah IBM Lotus Notes za potrebe slovenske policije. Opisali smo lastnosti in namen uporabe dveh odprto-kodnih rešitev, Apache Lucene in Solr, na katerih temelji iskalnik. S primeri smo prikazali izvajanje indeksiranja podatkov in iskanja po indeksu. Predstavili smo tudi dva modela (Boolov in vektorsko-prostorski model), ki jih sodobni sistemi IR uporabljajo pri iskanju informacij in teoretično pojasnili določanje pomembnosti dokumentov kot ga izvaja Lucene.

V policiji smo uspešno izdelali in implementirali v realno okolje interni iskalnik, ki policistom in kriminalistom omogoča iskanje po več kot 60 zbirkah glavnega dokumentnega sistema v državi upravi (SPIS) ter operativnih zbirkah za izmenjavo podatkov med državami Evropske unije (SIRENE, Interpol, Europol). Podobno programsko rešitev, ki temelji na opisani tehnologiji, kriminalisti uporabljajo za iskanje in analizo zaseženih podatkov in s tem hitrejšo obdelavo primerov.

Razvoj lastnega iskalnika se je izkazal za koristno rešitev, saj smo z njim omogočili zaposlenim na policiji prihranek na času, ki je bil potreben, da so iz različnih zbirk pridobili ustrezno informacijo in s tem tudi učinkovitejše opravljanje delovnih procesov.

Iskalnik deluje hitro, hitrost iskanja se zmanjša le, ko ima uporabnik dostop do večjega števila zbirk oz. je član večjega števila skupin oz. vlog na zbirkah. Večina uporabnikov takih pravic nima, izjema so le upravitelji posameznih zbirk oz. sistema Lotus, ki imajo praviloma neomejen dostop. Vseeno iskalnik tudi v teh primerih omogoča zadovoljivo iskanje po zbirkah Lotus Notes.

Iskalnik upošteva in realno imitira varnostni model sistema Lotus Domino preko izgradnje filtra poizvedbe, ki povzroči, da uporabniki na seznamu zadetkov ne vidijo dokumentov do katerih nimajo dostopa v primarni zbirki dokumentov. Z uporabo filtra poizvedbe se indeks, po katerem uporabniki izvajajo poizvedbe, navidezno razdeli na dva dela: del, ki vsebuje dokumente, ki so uporabnikom dostopni za iskanje; in del, ki vsebuje vse ostale dokumente. Četudi se uporabnikova poizvedba najprej izvrši čez celotno množico dokumentov v indeksu (tudi tistih do katerih uporabnik nima dostopa), se nato nad seznamom zadetkov izvrši še poizvedba, ki je enaka filtru poizvedbe. Iskalnik tako na preprost način uspešno zagotavlja varen dostop do dokumentov.

Glavne funkcionalnosti iskalnika so:

- Izvajanje prijave v iskalnik in overitve uporabnikov.
- Zapis uspešnih in neuspešnih overitev uporabnikov v dnevnik prijav.
- Periodično indeksiranje novo dodanih dokumentov v zbirke Lotus Notes.
- Izvajanje varnega dostopa do dokumentov.
- Uporaba preprostih in naprednih oblik poizvedb.

- Urejanje seznama zadetkov po pomembnosti, datumu nastanka in datumu spremembe dokumentov.
- Zapis izvršenih poizvedb v dnevnik za potrebe izvajanja kontrole uporabe iskalnika (če se v javnosti pojavi informacija skrite narave je možno preko pregleda dnevnika izvršenih poizvedb določiti množico uporabnikov, ki bi to informacijo lahko posredovali javnosti).

Pri razvoju smo uporabili samo prosto dostopne ter brezplačne rešitve in s tem tudi rešili problem licenciranja programske opreme[21]. S tem smo prihranili tako na finančnem področju, kot na področju urejanja nakupa rešitev.

Razvoj procesa za pridobivanje podatkov iz zbirk Lotus je omogočil izgradnjo indeksa, ki je osnova za uvedbo naprednih tehnik upravljanja z znanjem v policiji. V prihodnje bo iskalnik tako mogoče nadgraditi z novimi naprednimi funkcionalnostmi, kot so: popravljanje vnesenih poizvedb (angl. *Did you mean?*), predpostavljanje oz. nadaljevanje vnesenih poizvedb (angl. *Auto-complete*), urejanje zadetkov v skupine (angl. *clustering*), filtriranje dokumentov po ključnih besedah, uvedba opomnikov, ki bodo uporabnike samodejno obvestili o najdbi želenih dokumentov, izvajanje statistike, ekstrakcijo določenih pojmov iz dokumenta (imena, nazivi, registrske številke, telefonske številke, ipd.), iskanje podobnih zadetkov (angl. *More like this*), itd.

[21] Sem se ne upošteva licenciranje programske opreme, ki skrbi za vodenje raznih virov podatkov (Lotus, DB2, Oracle, …), ki so praviloma plačljiva.

6 Literatura

[1] N. Fuhr, »Models in Information Retrieval«, *Lectures on Information Retrieval*, št. 1980, str. 21–50, Springer Berlin, 2011

[2] R. B. Gonzáles, »Index Compression for Information Retrieval Systems«, Univerza v La Coruñi, doktorska disertacija, 2008

[3] IBM, *Inside Lotus: The Architecture of Notes and the Domino Server*, IBM Press, 2000

[4] C. D. Manning, P. Raghavan, H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008

[5] M. McCandless, E. Hatcher, O. Gospodnetić, *Lucene in Action*, Manning Publications Co., 2010

[6] S. Melnik, S. Raghavan, B. Yang, H. Garcia-Molina, »Building a Distributed Full-Text Index for the Web«, *ACM Transactions on Information Systems*, št. 19, zv. 3, str. 217–241, 2001

[7] G. Salton, C. Buckley, »Term-weighting approaches in automatic text retrieval«, *IP&M*, št. 25, zv. 5, str. 513–523, 1988

[8] G. Salton, A. Wong, C. S. Yang, »A Vector Space Model for Automatic Indexing«, *Communications of the ACM*, št. 18, zv. 11, str. 613–620, 1975

[9] J. Zobel, A. Moffat, »Inverted files for text search engines«, *ACM Computing Surveys*, št. 38, zv. 2, 2006

[10] Apache Solr. Dostopno na: http://lucene.apache.org/solr

[11] Apache Lucene. Dostopno na: http://lucene.apache.org

[12] Apache Tika. Dostopno na: http://tika.apache.org

[13] Normalizacija besedil – Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/Text_normalization

[14] Zgodovina razvoja Lotus Notes in Domino. Dostopno na: http://www.ibm.com/developerworks/lotus/library/ls-NDHistory/

[15] »*Only NSA can listen, so that's OK*«. Dostopno na: http://www.heise.de/tp/artikel/2/2898/1.html

[16] Apache Solr – *filter query*. Dostopno na: http://wiki.apache.org/solr/CommonQueryParameters#fq

[17] Standardni Boolov model – Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/Standard_Boolean_model

[18] »*How FBI technology woes let Fort Hood shooter slip*«. Dostopno na: http://arstechnica.com/information-technology/2012/07/how-fbi-technology-woes-let-fort-hood-shooter-slip-by/

[19] »*Information Insight: From the Department to the Enterprise with IBM InfoSphere Warehouse*«. Dostopno na: ftp://ftp.boulder.ibm.com/software/cn/data/download/IBM_InfoSphere_Warehouse_9.7_Whitepaper.pdf

[20] »*NYPD and Microsoft Create a Next Generation Law Enforcement Big Data Solution*«. Dostopno na: http://smartdatacollective.com/node/64511

[21] Lower Manhattan Security Initiative – Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/Lower_Manhattan_Security_Initiative