UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jure Adlešič

Optimizacija delovanja in povečanje obiska na spletni strani

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Tomaž Dobravec

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00466/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JURE ADLEŠIČ**

Naslov: **OPTIMIZACIJA DELOVANJA IN POVEČANJE OBISKA NA SPLETNI STRANI**

PERFORMANCE OPTIMIZATION AND INCREASE OF WEBSITE VISITS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

V okviru diplomskega dela izdelajte učinkovito spletno stran. Pri izdelavi uporabite odprtokodno programsko opremo (PHP, MySQL). Stran naj bo zasnovana tako, da lahko prikažete postopke za hiter dostop aplikacije do podatkovne baze. Predstavite tudi nevarnosti, ki se pri tem pojavijo in načine, kako se pred njimi zavarujemo. Pri izdelavi opišite načine, ki vplivajo na boljšo uvrstitev spletne strani v iskalnikih. Načine izboljšanega delovanja predstavite in analizirajte na konkretnih primerih.

Aplikacija naj bo preprosta in namenjena širokemu krogu internetnih uporabnikov. Zasnovana naj bo tako, da jo bodo lahko uporabljali vsi internetni uporabniki, brez predhodnega izobraževanja na tem področju. Aplikacija mora delovati optimalno, ne glede na brskalnik, s katerim uporabnik dostopa do vsebine.

Mentor:

doc. dr. Tomaž Dobravec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Jure Adlešič,

z vpisno številko 63040412,

sem avtor/-ica diplomskega dela z naslovom:

Optimizacija delovanja in povečanje obiska na spletni strani

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom doc. dr. Tomaža Dobravca.
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. 12. 2009 Podpis avtorja/-ice:

Zahvala

Za pomoč pri nastajanju diplomske naloge se zahvaljujem mentorju, doc. dr. Tomažu Dobravcu.

Zahvaljujem se staršem, Darji in sinu Taju za potrpežljivost in razumevanje ter Urošu in celotni ekipi Dominatus d. o. o., ki sodeluje pri trženju produkta.

Zahvaljujem se tudi vsem sošolcem za lepo preživete študentske dni (Ulfu, Tkalčiču, Anesu, Đoniju, Romanu in vsem ostalim, ki so z mano igrali poker).

Kazalo

Seznam uporabljenih kratic in simbolov

DOS - Denial of Service.
XSS - Cross Site Scripting.
SQL - Structured Query Language.
SEO - Search Engine Optimization.
MD5 - Message Digest algorithm 5.
CRC - Cyclic redundany check.
BDB - Berkeley Database.
ISAM - Indexed Sequential Access Method.

Povzetek

Aplikacije ločimo na namizne in spletne. Spletne aplikacije so namenjene široki uporabi, kjer vsi uporabniki obdelujejo iste podatke. Ti podatki so shranjeni v podatkovni bazi, ki jo mora programer oziroma načrtovalec zasnovati, tako da so podatki hitro dostopni in je podatkovna baza odzivna.

Večje kot je število uporabnikov, bolj previdni in skrbni moramo biti, saj se z večanjem prepoznavnosti in konkurenčnosti veča tudi nevarnost vdora in kraje podatkov v našem sistemu.

Zato sem v diplomski nalogi izpostavil področji, ki sta pomembni tudi za mojo spletno stran Spletni-slovar.com. Vsaka še tako dobro narejena, hitra in varna spletna stran, lahko ostane neobiskana, če je uporabniki ne poznajo oziroma je ne najdejo.

Optimizacija za iskalnike je ključnega pomena pri obisku strani. Z vloženim trudom in pridobljenim znanjem sem v času nastanka diplomske naloge na svojo stran privabil že preko 85.000 različnih mesečnih uporabnikov.

V diplomski nalogi sem opisal postopke, s katerimi poskrbimo za hitrost branja podatkov, varnost uporabnikov in načine, s katerimi izboljšamo uvrstitev naše spletne strani v iskalnikih. Vse praktične primere sem prikazal na delovanju spletne strani Spletni-slovar.com.

Ključne besede:

MySQL, optimizacija, optimizacija spletnih strani, varnost, varnost aplikacij, vdori, slovar, slovarji, Spletni-slovar.com

Abstract

Applications are divided to desktop and web applications. Web applications are intended for wide use where every user processes the same data. This data is saved in a database which the programmer or the designer has to plan in such manner that the data can be accessed quickly and that the database is responsive.

The larger the number of users, the more careful and thorough must be the designer because the increase of recognition and competence also increases the danger of intrusion and data loss in our system.

That is why I highlighted two fields in this diploma thesis that are also important for my webpage Spletni-slovar.com. Every webpage, even if it contains quality, speed and safety can remain without visits if users cannot find it.

Search engine optimization is crucial for webpage visits. During my writing of this diploma thesis and with the invested effort and acquired knowledge I attracted over 85,000 unique monthly users.

In this diploma thesis I described procedures which are used to improve data read speed, user safety and classification of our webpage in search engines. I showed all practical examples on the example of the Spletni-slovar.com webpage.

Key words:

MySQL, optimization, search engine optimization, security, application security, intrusions, dictionary, dictionaries, Spletni-slovar.com

Poglavje 1

Notranja zgradba MySQL-a

1.1 Groba shema zgradbe

Ker je MySQL odprtokodna podatkovna baza, lahko kadarkoli vzamete v roke izvorno kodo programa in jo prilagodite svojim potrebam. Morate se zavedati, da je izvorna koda zelo zapletena in predstavlja izziv tudi najbolj izkušenim C/C++ programerjem.

Osnovni tipi podatkovnih baz:

1. Predmetno usmerjene podatkovne baze
   Podatki so predstavljeni kot objekti in so priporočljive, ko potrebujemo hitro procesiranje zapletenih zbirk podatkov. Nekatere so namenjene uporabi v standardnih programskih jezikih (Phyton, Java, C/C++), druge uporabljajo lastne programske jezike. Poizvedbe so večinoma hitrejše, saj posameznih tabel ne združujemo kot pri relacijskih podatkovnih bazah. Poizvedbe so podobne, včasih celo popolnoma enake kot pri relacijskih podatkovnih bazah. Objektno orientirane podatkovne baze kot rezultat poizvedbe vrnejo objekt oziroma zbirko objektov, za razliko od relacijske podatkovne baze, ki kot rezultat vrne tabelo.

2. Predmetno relacijske podatkovne baze
   Podobne so relacijskim podatkovnim bazam, uporabljajo pa tehnike predmetno usmerjene podatkovne baze. Poizvedbe z združevanjem so poenostavljene, saj za to skrbi podatkovna baza sama.

3. Relacijske podatkovne baze
   Trenutno so dominantne na mnogih področjih. Relacijska podatkovna

baza je zbirka relacij, ki jih imenujemo tabele. Relacije so zbirke vrstic, ki vsebujejo enake atribute. Zasnovane so, tako da so uporabniku razumljive. Vse podatkovne tabele so sestavljene iz glavne vrstice, ki predstavlja podatke za vsak zapis v tabeli in podatkovnih vrstic, v katerih so zapisani podatki. Do podatkov v relacijskih podatkovnih bazah dostopamo s pomočjo povpraševalnih jezikov (SQL). MySQL je poleg mnogih drugih tipičen primer relacijske podatkovne baze, ki si ga bomo ogledali bolj podrobno.

Primer: Poizvedba v relacijski podatkovni bazi:

```
SELECT oseba.ime
FROM oseba, naslov
WHERE oseba.naslovid = naslov.id
 AND naslov.kraj = 'LJ'
```

Primer: Poizvedba v objektno relacijski podatkovni bazi [10]:

```
SELECT Forma1(o.ime)
FROM oseba o
WHERE o.naslovid = 'LJ'
```

1.2 MySQL pospeševalnik

MySQL ima vgrajen pospeševalnik (MySQL Optimizer), ki skrbi za hitro izvajanje poizvedb. Naloga pospeševalnika je analiza in optimizacija podane poizvedbe. Pospeševalnik preračuna različne izvedbene plane ter vzame tistega, ki se glede na rezultate najbolje obnese.

Primer - Osnovni primer poizvedbe:

```
SELECT * FROM  tbl_name WHERE 0;
```

V tem primeru pospeševalnik najprej preveri pogoj. Ker ve, da noben rezultat ne bo zadostil pogoju, se poizvedba izvrši zelo hitro, ne glede na to, koliko vrstic vsebuje tabela. V tem primeru MySQL sploh ne išče po tabeli, vendar takoj vrne rezultat.

Najpomembnejša, vendar ne edina naloga pospeševalnika, je izbira najboljšega kazalca v tabeli, če kazalec obstaja. Pravi kazalec za našo poizvedbo je tisti, ki najhitreje odstrani vrstice, ki jih ne potrebujemo. Poizvedbe so hitrejše, če se najprej izvede pogoj, ki odstrani največje število vrstic.

Primer - Osnovno iskanje po imenu in priimku:

```
SELECT naslov
FROM oseba
WHERE ime = 'Janez'
   AND priimek = 'Novak';
```

V primeru, da imamo v tabeli 2000 Janezov in 500 Novakov, Janezov Novakov pa je le 50, je bolje, če najprej zadostimo drugemu pogoju, ker bo poizvedba hitrejša. Poizvedba bo hitrejša zato, ker bomo pri naslednji primerjavi iskali po manjšem številu zapisov.

Kako pomagamo pospeševalniku pri hitrejši izvedbi?

1. **Primerjamo stolpce, ki imajo isti podatkovni tip** [2]
   Primerjamo dva stolpca s kazalcem in istim podatkovnim tipom. Pri tem moramo paziti, da je podatkovni tip dejansko isti (varchar(10) != varchar(12)). Če primerjamo dve poizvedbi z istim podatkovnim tipom, lahko vidimo, katera se izvrši hitreje.

2. **Naj bodo indeksi samostoječi**
   Če uporabljamo kazalec v funkciji ali kot del bolj kompleksnega izraza, ga MySQL ne more uporabljat. Včasih pretvorba poizvedbe ni mogoča, v večini primerov se lahko izognemo uporabi takih poizvedb.

   Primer - Preračunavanje poizvedbe zaradi matematičnih operacij:

   ```
   WHERE hista_st * 2 < 4
   WHERE hisna_st < 4/2
   ```

   V drugem primeru pospeševalnik vzame kazalec, saj prirejanje ni potrebno. Prvi primer je bistveno slabši, saj se za vsak zapis v tabeli preračuna rezultat in uporaba kazalca ni mogoča.

   Primer - Preračunavanje poizvedbe zaradi uporabe vgrajenih funkcij:

   ```
   SELECT * FROM oseba WHERE year(datum_roj) < 1990
   SELECT * FROM oseba WHERE datum_roj < '1990-01-01'
   ```

   Ta primer je zelo podoben. V prvem primeru je potreben izračun leta rojstva za vsako vrstico posebej, zato kazalec spet ne pride do izraza.

Primer - Več primerov preračunavanja podatkov [1]:

```
WHERE TO_DAYS(datum_vnosa) - TO_DAYS(CURDATE()) < 30
WHERE TO_DAYS(datum_vnosa) < 30 + TO_DAYS(CURDATE())
WHERE datum_vnosa < DATE_ADD(CURDATE(),
        INTERVAL 30 DAY)
```

V prvem primeru je potreben izračun datuma, ko je bila beseda vnešena, v dneve. Potem je potrebno še odšteti današnji datum. Drugi primer je boljši, saj se desna stran izračuna, konstanta na desni strani pa je vedno ista. Na levi strani je še vedno potrebno preračunavanje v dneve, zato uporaba kazalca ni mogoča. Tretji primer je najboljši, saj na levi strani uporabimo kazalec, desna stran pa se izračuna samo enkrat.

3. **Ne uporabljamo % na začetku LIKE ukaza**
   Po potrebni ga uporabimo, ne smemo pa ga uporabljati iz navade.

4. **Pomagamo pospeševalniku bolje izkoristiti kazalec**
   Poženemo ukaz ANALYZE_TABLE. S tem ukazom pospeševalnik analizira in shrani podatke o tabeli. Na podlagi teh podatkov lahko boljše izvaja poizvedbe, saj so njegove informacije o stanju tabel in indeksov lahko že zastarele. Ta ukaz lahko poženemo le nad BDB, MyISAM in InnoDB podatkovnimi tipi tabel. Tabele so v času analize zaklenjene za branje, InnoDB pa za pisanje. Stanje kazalcev po analizi lahko preverimo z ukazom SHOW INDEX.

5. **Uporabljamo ukaz EXPLAIN**
   Ta ukaz nam pove, katere kazalce bo uporabil pospeševalnik. Podatki, ki nam jih vrne so uporabni tudi pri analizi poizvedb in primerjavi zahtevnosti med različnimi poizvedbami.

6. **Svetujemo pospeševalniku, če je to potrebno**
   Pospeševalniku lahko svetujemo na različne načine. Svetujemo mu v primeru, ko se sam ne zna odločiti najbolje. Uporabimo lahko ukaz STRAIGHT_JOIN, ki izvede pridružitev tabele v takem vrstnem redu, kot mu določimo v WHERE stavku. Svetujemo mu lahko tudi pri izbiri kazalca z ukazi FORCE_INDEX, USE_INDEX in IGNORE_INDEX.

7. **Izogibamo se pretvorbi podatkovnih tipov**
   MySQL zna pretvarjati podatkovne tipe med seboj. Vendar vsaka pretvorba vzame nekaj časa, v skrajnem primeru lahko pride tudi do neuporabe kazalca.

   Primer - Pretvorba podatkovnih tipov:

```
SELECT * FROM oseba WHERE starost >'25'
SELECT * FROM oseba WHERE starost > 25
```

   V prvem primeru MySQL opravi pretvorbo podatkovnega tipa, ki ni bila potrebna.

1.3 Poizvedbe v predpomnilniku

1. 0 - izključimo poizvedbe v predpomnilniku.
2. 1 - vključimo poizvedbe v predpomnilniku.
3. 2 - ročno vključimo, ko je to potrebno.

Ko neko poizvedbo poženemo prvič, bo trajala dlje, kot bi trajala, če bi imeli poizvedbe v predpomnilniku izključene. Do tega pride, ker mora MySQL poizvedbo shraniti v predpomnilnik. Vse nadaljne poizvedbe, do naslednjega zapisa, se izvršijo bistveno hitreje.

Primer - Poizvedba prebere vse besede iz angleško slovenskega slovarja in jih uredi po slovenskem prevodu:

```
SELECT *
FROM anglesko_slovenski_slovar
ORDER BY slovenska_beseda
```

- 0.794 - Trajanje poizvedbe ob prvem zagonu
- 0.005 - Trajanje ob drugem zagonu

Ko je poizvedba že v predpomnilniku, je njeno izvajanje lahko več kot 100 krat hitrejše.

Nekatere podatkovne baze shranijo v predpomnilnik cele izvedbene plane za ponavljajoče poizvedbe. MySQL uporablja drugačen način shranjevanja. Pri MySQL-u je poizvedba v predpomnilniku shranjena kot množica rezultatov, ki jih vrne nek izbirni (select) stavek. Ko vstavimo v tabelo nove podatke ali spremenimo stare, se rezultat v predpomnilniku, ki uporablja to tabelo, označi za napačnega, čeprav bi lahko vrnil pravilne rezultate. Vendar je to način, ki predstavlja zelo malo dodatnega dela za strežnik in se zelo dobro obnese pri obremenjenih sistemih. Iz aplikacijskega vidika je enako, če MySQL vrne podatke iz tabele ali iz predpomnilnika.

Način, kako MySQL preveri, če se neka poizvedba že nahaja v predpomnilniku je preprost. Poizvedbe so shranjene v tabelo. Ključ, po katerem najdemo željeno poizvedbo je preprost kodiran zapis podatkovne baze, poizvedbe in nekaterih drugih podatkov, ki lahko vplivajo na izbiro. Zaradi primerjave poizvedbe, ki jo želimo izvršiti in poizvedbe v predpomnilniku lahko hitro pride do razlike in MySQL ne prepozna shranjene poizvedbe. Do teh razlik pride že zaradi presledkov, komentarjev in zamenjav malih črk z velikimi. Vse vgrajene funkcije, ki vračajo rezultate glede na časovno obdobje, prijavljenega uporabnika itd. (now, current_date, connection_id), preprečijo shranjevanje poizvedbe v predpomnilnik. MySQL zavrne shranjevanje, takoj ko naleti na ukaz, ki to prepreči. Rezultat takšne poizvedbe se ne shrani.

Primer - Poizvedba, ki se ne shrani v predpomnilnik:

```
SELECT *
FROM tabela1
WHERE datum = CURRENT_DATE;
```

Primer - Poizvedba, ki se shrani v predpomnilnik:

```
SELECT *
FROM tabela1
WHERE datum = '2009-06-06';
```

Poizvedbe v predpomnilniku lahko povečajo hitrost, vendar je to odvisno od aplikacije. Če omogočimo poizvedbe v predpomnilniku, se moramo zavedati, da to povzroča nekaj dodatnega dela strežniku pri branju in pisanju:

1. Pri branju moramo preveriti predpomnilnik pred izvedbo.
2. Če je poizvedba primerna za shranjevanje, vendar je še nimamo v predpomnilniku, je potrebno shraniti rezultate po izvedbi.
3. Pri novih zapisih in spremembah v neki tabeli je potrebno označiti stare zapise, ki uporabljajo to tabelo, kot neprimerne.

Brisanje poizvedb iz predpomnilnika lahko traja kar nekaj časa, če smo določili prevelik kos predpomnilnika v nastavitvah MySQL-a.

V predpomnilniku niso shranjeni samo rezultati poizvedbe, ampak je predpomnilnik organiziran podobno kot datotečni sistem. Sistem uporablja strukture, ki vsebujejo podatke o tekstu poizvedbe, rezultatih poizvedbe, kateri bloki pomnilnika so prosti in katere tabele so povezane z določenimi poizvedbami. Posamezna stuktura je sestavljena iz blokov, ki so lahko različnih tipov. En blok lahko vsebuje rezultat poizvedbe, seznam tabel, ki jih uporablja poizvedba, tekst poizvedbe ... Vsi tipi blokov so obravnavani na enak način.

Ko se strežnik zažene, rezervira celotno količino pomnilnika kot en blok. Odšteti je potrebno le en majhen del pomnilnika, ki se uporablja za urejanje pomnilnika. Strežnik rezervira blok določene velikosti, preden začne shranjevati. Zato ne more vedeti, kako velik bo blok na koncu. MySQL sam upravlja s pomnilnikom in rezervacijo blokov. Vendar pa je to vseeno počasen proces, saj mora sistem preveriti vse proste bloke in izbrati takšnega, ki bo dovolj velik.

Shranjevanje poizvedb v predpomnilnik se izkaže za koristno, če je prihranek časa večji od časa, ki ga strežnik porabi za iskanje shranjenih poizvedb, brisanje poizvedb in shranjevanje poizvedb. Če izključimo poizvedbe v predpomnilniku, strežnik izvede vsako poizvedbo in vrne rezultate poizvedbe. Ker tega pri večini aplikacij ne želimo, moramo biti pozorni na dodatno delo, ki ga poizvedbe v predpomnilniku povzročajo strežniku. Ko poženemo neko poizvedbo, mora MySQL preveriti, če ima rezultate te poizvedbe že shranjene. Če so rezultati že v predpomnilniku in niso označeni kot napačni, jih vrne. Če rezultatov ni v predpomnilniku in je poizvedba primerna za shranjevanje, jo izvede, shrani rezultat ter ostale meta podatke in vrne rezultat aplikaciji. Vsak vstavljajoč

(insert) in posodobitveni (update) stavek v tabelo shrani nov zapis in označi vse shranjene poizvedbe nad to tabelo za nepravilne.

Vedno je težko določiti, ali se nam shranjevanje poizvedb splača ali ne. Ne obstajajo točno določena pravila o koristnosti shranjevanja poizvedb v predpomnilnik, ker je ta odločitev odvisna od aplikacije in operacij, ki jih izvaja. Najbolj se splača shranjevanje takšnih poizvedb, ki izvajajo zahtevne operacije nad velikimi količinami podatkov, vrnejo pa prostorsko majhen rezultat. Tipičen primer takšne poizvedbe je preštevanje števila rezultatov, kjer je rezultat neko naravno število. Kljub temu se splača shranjevati tudi veliko drugih rezultatov poizvedb.

Najbolj enostaven način za ugotavljanje koristnosti poizvedb v predpomnilniku je stopnja zadetkov (hit rate). Ta podatek nam pove, kolikšen odstotek poizvedb se vrne iz predpomnilnika in kolikšen odstotek mora obdelati strežnik. Za računanje tega podatka ima MySQL strežnik vgrajeni dve spremenljivki - Qcache_hits in Com_select. Če nek izbirni stavek vrne podatke iz predpomnilnika, se poveča prva spremenljivka, če rezultate poišče strežnik pa druga. Stopnjo zadetkov lahko izračunamo po formuli: Qcache_hits/(Qcache_hits + Com_select). Primerna stopnja zadetkov ni točno določena. Če je ta stopnja nizka in nam prihrani izvajanje najtežjih poizvedb, se splača, saj dodatno delo strežnika predstavlja zelo majhno dodatno obremenitev.

Vsak izbirni stavek, ki ga MySQL ne vrne iz predpomnilnika, je zgrešitev (miss). Vzroki za zgrešitev:

1. Poizvedba ni primerna za shranjevanje. Vzrok za to je lahko uporaba nedeterminističnih funkcij (now, current_date ...) ali preobsežen rezultat. V obeh primerih se v MySQL-u poveča vgrajena spremenljivka - "Qcache_not_cached".

2. Strežnik te poizvedbe še ni izvedel, zato je tudi ni mogel shraniti.

3. Strežnik je poizvedbo izvedel in shranil njen rezultat, vendar pa se je vsebina tabele kasneje spremenila in rezultat je označen kot nepravilen.

Če imamo na strežniku veliko zgrešitev, je razlog eden od naštetih:

1. Strežnik še ni ogret. Če smo šele pognali shranjevanje poizvedb, je pomnilnik še prazen in se mora najprej napolniti. Enak rezultat dobimo tudi pri ponovnem zagonu strežnika.

2. Strežnik ima vedno drugačne poizvedbe. Če nimamo veliko ponavljajočih poizvedb, potem shranjevanje ni smiselno.

3. Imamo veliko sprememb v tabelah, ki označijo shranjene rezultate kot nepravilne.

Označevanje nepravilnih poizvedb je zelo pomemben dejavnik pri odločanju o uporabi shranjevanja poizvedb. Če za primer vzamemo dve tabeli in imamo nad prvo samo izbirne stavke, nad drugo pa izvajamo insert, update operacije ter predpostavimo, da se nad drugo tabelo izvede precej več operacij kot nad prvo, se shranjevanje poizvedb ne splača več, saj je dodaten napor strežnika pri razveljavljanju poizvedb nad drugo tabelo prevelik. V tem primeru bomo lahko imeli zelo visoko stopnjo zadetkov, kljub temu pa je strežnik bolj obremenjen. Na podoben primer naletimo, če se velikokrat spreminjajo podatki v isti tabeli, nad katero poganjamo izbirne stavke. Vsaka sprememba podatkov v tabeli označi shranjene rezultate kot neveljavne. Strežnik v tem primeru vedno shranjuje in potem briše shranjene poizvedbe, uporabi pa jih lahko le redko. Najslabši primer shranjevanja je, če shranjene rezultate označimo za neprimerne, preden jih prvič uporabimo.

Obvezno moramo spremljati, kako strežnik dejansko uporablja predpomnilnik. Če ne izkorišča celotnega prostora, ki smo mu ga podali, je potrebno ta prostor zmanjšati. Lahko se zgodi, da je predpomnilnik premajhen in se nekateri rezultati brišejo, ker zmanjka prostora. V tem primeru je potrebno predpomnilnik povečati. Idealna velikost bloka, znotraj pomnilnika bi bila enaka velikosti večini rezultatov najpogostejših poizvedb. Ker nekatere poizvedbe vrnejo zelo majhne rezultate, druge pa velike količine podatkov, je določanje prave velikosti bloka problem. Povprečno velikost rezultatov poizvedbe v predpomnilniku lahko izračunamo, če velikost zasedenega predpomnilnika delimo s številom rezultatov poizvedb v predpomnilniku (Qcache_query_in_cache).

V nastavitvah MySQL-a imamo le nekaj spremenljivk, s katerimi nadzorujemo predpomnilnik, namenjen za shranjevanje [7]:

1. query_cache_type - Vključimo ali izključimo poizvedbe v predpomnilniku. Tretja možnost je še, da predpomnilnik vključimo le na zahtevo.
2. query_cache_size - Velikost predpomnilnika, ki ga namenimo poizvedbam v predpomnilniku.
3. query_cache_min_res_unit - Najmanjša velikost posameznega bloka.
4. query_cache_limit - Velikost največjega rezultata, ki ga bomo shranili. Rezultati, ki zasedejo več prostora od določenega, se ne shranijo.
5. query_cache_wlock_invalidate - Branje shranjenih rezultatov nad tabelami, ki so jih zaklenile druge poizvedbe. Po privzetih nastavitvah MySQL-a je ta spremenljivka izključena.

Shranjevanje poizvedb lahko prenesemo tudi na nivo aplikacije. To naredimo, tako da iz celotne strani zgeneriramo nov dokument. Pri tem načinu se moramo odločiti, koliko časa želimo, da so rezultati shranjeni in kdaj jih bo potrebno ponovno zgenerirati. Ta način bi bil primeren za prvo stran na slovarju, saj prikazujemo le število prijavljenih in aktivnih uporabnikov. Ker se dnevno registrira le 10 do 50 uporabnikov, bi lahko celotno vsebino strani shranili kot html dokument in jo na novo zgradili le enkrat na uro, saj se vsebina ne menja pogosto. Zelo pozorni moramo biti na aktualnost podatkov. Ta primer ne bi bil primeren za prikaz števila vpisanih besed uporabnika na moji spletni strani. Uporabnik bi namreč vpisoval nove besede, mi pa bi mu prikazovali podatek o vnešenih besedah na eno uro. Pri tem bi uporabnik dobil občutek, da naša aplikacija ne deluje in bi izgubil motivacijo.

Poglavje 2

Optimizacija poizvedb s strani uporabnika

2.1 Vrste tabel in lastnosti

MySQL pozna več različnih vrst tabel. Bistvena razlika med njimi je v tem, da nekatere omogočajo upravljanje transakcij, druge pa ne. Glavni tipi tabel v MySQL-u so:

1. MyISAM (razvil se je iz ISAM, ki se ne uporablja več).
2. Heap (podatke shranjuje v RAM, zato se izbrišejo, ko ponovno zaženemo strežnik).
3. BDB
4. InnoDB

Poleg zgoraj omenjenih tipov tabel, poznamo še tip Merge, ki deluje enako kot MyISAM, le da obravnava eno tabelo tipa Merge enako kot več MyISAM tabel.

Najbolj pomembna razlika pri tipih tabel je v njihovem odnosu do transakcij. Tabele, ki omogočajo transakcije, imajo vgrajen mehanizem, ki določene podatke zaklene med postopkom obdelave. Če se transakcija uspešno izvede, se podatki odklenejo, sicer je pred tem potrebno vrniti vse podatke v prvotno stanje, pred izvedbo transakcije. Transakcije podpirajo podatkovni tipi BDB in InnoDB, MyISAM in HEAP pa transakcij ne podpirajo.

Razlika med tipi tabel je tudi v zaklepanju podatkov (lock granuality). Ko ena aplikacija dostopa do podatkov in jih zaklene, druga aplikacija ne more obdelati teh podatkov. Vse vrste podatkovnih tabel omogočajo zaklepanje, razlika pa je v velikosti blokov, ki jih določena vrsta zaklene. Zaklenemo lahko le določeno vrstico, eno stran ali celotno tabelo. Prednost zaklepanja ene vrstice je v tem, da lahko druga aplikacija prosto dostopa do vseh podatkov razen do zaklenjenih vrstic. Če zaklenemo celotno tabelo, mora druga aplikacija počakati, da prva konča z delom in odklene celotno tabelo. Vendar se moramo pri tem zavedati, da je zaklepanje vrstic počasnejše od zaklepanja celotne tabele, saj mora strežnik hraniti podatke o vseh zaklenjenih vrsticah in preverjati, če so le te zaklenjene ali ne. MyISAM in HEAP zakleneta celotno tabelo, BDB zaklene eno stran (približno 8 KB), InnoDB zaklene le eno vrstico.

Podatkovne tabele se razlikujejo tudi po načinu shranjevanja podatkov na disku. HEAP hrani podatke v pomnilniku, zato lahko do njih zelo hitro in učinkovito dostopa. MyISAM hrani podatke v nekaj ločenih datotekah (podatki ločeni od indeksov), BDB ima za vsako tabelo svojo datoteko. Poseben način shranjevanja pa uporablja InnoDB, ki podatke hrani v posebnih tabelah (tablespaces).

2.2 Optimizacija poizvedb

Pri optimizaciji podatkovnih baz je njena zasnova prva stvar, na katero moramo pomisliti. Slaba zasnova podatkovne baze vpliva na hitrost in tudi samo težavnost izdelave aplikacije. Slabo zasnovane podatkovne baze je kasneje, na delujoči aplikaciji težko, če ne celo nemogoče popravljati. Vendar pa sama zasnova ni edina stvar, ki vpliva na hitrost. Če imamo slabo zasnovane poizvedbe, nam tudi optimalne podatkovne baze ne bodo delovale hitro.
Za hitrost podatkovne baze in posledično aplikacije so bistvene tri stvari, ki se medsebojno dopolnjujejo:

1. Optimizacija poizvedb.

2. Postavitev indeksov.

3. Zasnova podatkovne baze.

Ko pride do težav in se aplikacija odziva počasneje, kot je normalno, so poizvedbe prva stvar, ki jo preverimo.

Poizvedbe nad velikimi količinami podatkov

Najpogostejši razlog za dolgo trajanje poizvedb je velika količina podatkov. Nekatere poizvedbe morajo obdelati veliko količino podatkov, da vrnejo željen rezultat. Za te poizvedbe je normalno, da trajajo dalj časa. Včasih pa lahko poizvedbe predelamo, tako da obdelujejo manjšo količino podatkov.
Slabo zasnovane poizvedbe rešujemo v dveh korakih:

1. Ugotovimo, kje se v naši aplikaciji uporablja več podatkov, kot jih potrebujemo. Večinoma gre za preveč prebranih vrstic, včasih pa tudi stolpcev.
2. Poiskati moramo poizvedbe, pri katerih MySQL strežnik analizira več vrstic, kot je potrebno.

Nekaj tipičnih napak pri pisanju poizvedb:

1. Izberemo vse stolpce v neki tabeli (uporaba *).
2. Izberemo podatke brez uporabe omejitve (LIMIT). V aplikaciji pa preberemo le prvih nekaj zapisov.

V nekaterih primerih je bolje, da poizvedbe delamo nad večjimi količinami podatkov, kot pa je potrebno. Včasih lahko uporabimo bolj potratne poizvedbe zato, da lahko isto kodo uporabljamo na več mestih. Vedno pa moramo premisliti, koliko nam tak pristop poenostavi programiranje aplikacije in kakšni so stroški takih poizvedb.

Bolj potratne poizvedbe lahko uporabimo v primeru, ko imamo enako poizvedbo že v predpomnilniku. V takem primeru postane bolj potratna poizvedba hitrejša, saj vrne rezultate iz predpomnilnika in ni potrebno njeno ponovno izvajanje.
Popolna poizvedba bi analizirala enako število zapisov, kot bi jih vrnila, vendar je to v praksi zelo redko izvedljivo.

Enostavne meritve zahtevnosti poizvedb

Za zahtevnost poizvedbe uporabljamo kazalce:

1. Izvedbeni čas (Execution time).
2. Število pregledanih vrstic (Number of rows examined).
3. Število vrnjenih vrstic (Number of rows returned).

Noben od zgoraj naštetih kazalcev ni popoln pokazatelj zahtevnosti poizvedbe. Pri časovno potratnih poizvedbah se stanja vseh treh kazalcev in tekst poizvedbe zapišejo v dnevnik (slow query log). Dnevnik je najboljši seznam poizvedb, ki v aplikaciji trajajo predolgo.

Testiranje poizvedb

Poizvedbe testiramo z uporabo funkcije razloži (Explain). Z uporabo te funkcije vidimo tudi način dostopa do tabele. Poznamo kar nekaj načinov dostopa:

1. Iskanje po celi tabeli (full table scan).
2. Iskanje po kazalcu (index scan).
3. Iskanje po določenem območju (range scan).
4. Iskanje po enoličnih kazalcih (unique index lookups).
5. Iskanje po konstantah (constants).

Pri tem si načini dostopa sledijo po vrsti od najbolj potratnega do najhitrejšega. Najhitrejši način je tisti, ki potrebuje najmanj podatkov. Za poizvedbe, ki uporabljajo bolj potraten način, je najboljša rešitev dodajanje kazalca nad tabelo.

V spodnjem primeru bomo prikazali zahtevnost poizvedbe nad tabelo uporabnikov z in brez uporabe indeksa. V tabeli imamo 5073 uporabnikov. Parameter 'count' predstavlja število vpisanih besed za določenega uporabnika. Poiskali bomo tiste uporabnike, ki so vpisali deset različnih besed. Imamo deset takšnih uporabnikov.

Primer - Prikaz dveh poizvedb z uporabo funkcije razloži (EXPLAIN):

Poizvedba brez uporabe kazalca:

```
EXPLAIN SELECT * FROM uporabniki WHERE counter = 10
id: 1
select_type: SIMPLE
table: users
type: all
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 5073
Extra: Using where
```

Poizvedba z uporabo kazalca:

```
EXPLAIN SELECT * FROM uporabniki WHERE counter = 10
id: 1
select_type: SIMPLE
table: users
type: ref
possible_keys: count
key: count
key_len: 2
ref: const
rows: 10
Extra: Using where
```

Kot vidimo iz zgornjega primera, je MySQL pri prvem načinu moral pregledati vse zapise v podatkovni bazi, da je lahko izločil tiste, ki niso ustrezali pogoju.

MySQL lahko uporabi pogojni stavek (where) na tri možne načine:

1. Prvi in najboljši način je uporaba pogojnega stavka za iskanje po kazalcih.

2. Drugi način je uporaba kazalcev za pridobitev podatkov in filtriranje nepravilnih zadetkov.

3. Tretji in najslabši način je izbira vseh vrstic iz tabele in filtriranje nepravilnih zadetkov. Ta način predstavlja naš prvi primer poizvedbe brez uporabe kazalca.

Razgradnja poizvedb

Pod razgradnjo poizvedb razumemo večkratno poganjanje poizvedbe nad manjšim številom podatkov. Ta način je najbolj uporaben pri brisanju zastarelih podatkov iz podatkovne baze. Če želimo zbrisati zelo velike količine podatkov lahko podatkovno bazo zaklenemo za daljši čas in s tem onemogočimo izvajanje operacij hitrim poizvedbam. Namesto da poženemo izbris vseh podatkov, brišemo le po nekaj tisoč podatkov, dokler jih ne zmanjka.

Razgradnja pridruženih poizvedb

Včasih je uporabno pridružene (join) poizvedbe nad večimi tabelami razgraditi, tako da poženemo posamezne poizvedbe nad vsako tabelo posebej.

Primer - Razgradnja pridruženih poizvedb (za testni primer bomo izbrali urednika pod zaporedno številko deset, število njegovih vpisanih besed (iz tabele uporabniki) in izpis vseh besed tega uporabnika v angleško slovenskem slovarju (tabela anglesko_slovenski)):

```
SELECT *
FROM uredniki
JOIN uporabniki ON uredniki.uporabnik_id=uporabniki.id
JOIN anglesko_slovenski
WHERE uredniki.active=10;

SELECT * FROM uredniki WHERE id=10; #dobimo test_user, user_id
    =21
SELECT * FROM uporabniki WHERE id=21; #dobimo število vseh besed
    , ki jih je vpisal ta uporabnik
SELECT * FROM ang_slo WHERE uporabniki_id = 21;
```

Na prvi pogled izgleda prva poizvedba z uporabo pridruževanja hitrejša in v večini primerov to tudi drži. Kljub temu je lahko drugi način v določenih primerih hitrejši:

1. Določene poizvedbe se že lahko nahajajo v predpomnilniku in njihovo izvajanje ni potrebno.
2. Zaklepanje tabel se izvrši le nad posamezno tabelo in ne nad vsemi tremi hkrati (zato lahko ostale poizvedbe nemoteno dostopajo do drugih tabel).
3. Lažji dostop do tabel, če so le te razdeljene med več strežniki.
4. Izognemo se možnosti podvajanja poizvedb zaradi napake v poizvedbi.

5. Če lahko pridružitev zamenjamo s seznamom (IN) v velikih tabelah.

2.2.1 Omejitve MySQL pospeševalnika

Delovanje pospeševalnika smo si pogledali že v drugem poglavju. Sedaj bomo to znanje nadgradili s teorijo o omejitvah, ki jih ima. Na srečo so takšni primeri zelo redki, zato lahko MySQL v večini primerov pospeši vašo poizvedbo.

Soodvisnost poizvedb

Primer - Soodvisne poizvedbe:

```
SELECT *
FROM slovensko_angleski
WHERE id IN
    ( SELECT id
      FROM anglesko_slovenski
      WHERE usr_sug = 10);
```

Pričakovali bi, da MySQL izvede najprej notranjo poizvedbo in potem zunanjo. Ker so poizvedbe z IN načeloma zelo hitre, bi pričakovali, da se bo poizvedba izvršila tako:

```
SELECT *
FROM slovensko_angleski
WHERE id IN (1,2,33,432,541) ...
```

Zgodilo se bo ravno nasprotno. MySQL bo za vsak zapis v tabeli slovensko_angleski izvršil notranji del poizvedbe. Pri manjših tabelah s tem ne bomo imeli težav. Če prva tabela (slovensko_angleski) zraste, lahko izgubimo ogromno hitrosti podatkovne baze. Take poizvedbe lahko sami pospešimo na različne načine (prisilimo v izvajanje notranjega dela najprej - GROUP_CONCAT(), shranimo notranji del poizvedbe v predpomnilnik). To ne pomeni, da ne smemo uporabljati odvisnih poizvedb. V določenih primerih so lahko takšne poizvedbe tudi zelo hitre.

Primer - Poizvedba, ki jo lahko pospešimo z odvisno poizvedbo:

```
SELECT DISTINCT uporabniki.id
FROM uporabniki
 INNER JOIN slovensko_angleski
 ON uporabniki.id = slovensko_angleski.uporabniki_id;

SELECT uporabniki.id
FROM uporabniki
WHERE EXISTS (
   SELECT * FROM anglesko_slovenski WHERE uporabniki.id =
       anglesko_slovenski.uporabniki_id
);
```

V drugem primeru se izognemo ukazu za izbor različnih vrednosti (DISTINCT), ki je lahko tako velik, da naredi začasno tabelo na disku. Temu se izognemo z uporabo ukaza EXISTS. Zato je druga poizvedba precej hitrejša od prve. Če nismo prepričani o uporabi vgnezdene poizvedbe, je najbolje, da oba primera testiramo.

Uporaba unije

Če pri uporabi unije (union) uporabljamo omejitev (limit), moramo omejiti obe poizvedbi. V primeru, da imamo dve zelo veliki tabeli in le eno omejimo na prvih nekaj zadetkov, bo MySQL obe tabeli shranil na disk in nato izbral prvih nekaj zadetkov. Temu se lahko izognemo, tako da omejimo obe poizvedbi na prvih nekaj zadetkov.

Združevanje kazalcev

V MySQL-u lahko uporabljamo več kazalcev (index) nad eno tabelo v eni poizvedbi. To je mogoče šele od verzije 5.0 dalje. Večinoma deluje dobro, včasih lahko pride do prevelike izrabe pomnilnika in procesorja. To se zgodi, takrat ko vsi od uporabljenih kazalcev izločijo zelo malo podatkov. Taka poizvedba je lahko počasnejša od iskanja po celi tabeli. Če naletimo na tak problem, se mu lahko izognemo, tako da onemogočimo kazalec (IGNORE INDEX).

Vzporedno izvajanje

MySQL ne omogoča poganjanja ene poizvedbe na več procesorskih enotah, kot to omogočajo nekatere druge podatkovne baze.

Izgubno iskanje po kazalcih

Kazalci v MySQL-u omejijo podatke od neke začetne točke do končne točke. Ko pri poizvedbi uporabljamo kazalec, je potrebno preiskati celotno področje kazalca. MySQL ne omogoča iskanja samo po nekem manjšem delu kazalca.

Iskanje najmanjšega in največjega elementa

Iskanje najmanjšega (MIN()) in največjega (MAX()) elementa ni dobro optimizirano v MySQLu.

Primer - Iskanje prvega registriranega uporabnika z imenom Janez:

```
SELECT MIN(users.id)
FROM users
WHERE ime='Janez';
```

Ker nimamo kazalca na polju ime, bo taka poizvedba preiskala celotno tabelo. Teoretično bi se lahko taka poizvedba ustavila, takoj ko najde prvi zadetek, ki zadostuje pogoju, saj bi primarni ključi morali biti urejeni po vrstnem redu. Tako poizvedbo bi lahko pospešili z uporabo omejitve:

```
SELECT users.id
FROM users USE INDEX(PRIMARY)
WHERE ime='Janez' LIMIT 1;
```

Izbor in spremembe iste tabele

MySQL ne omogoča izbora (SELECT) in spremembe (UPDATE) iste tabele hkrati.

2.2.2 Optimizacija specifičnih tipov poizvedb

V tem poglavju bomo obravnavali najbolj tipične poizvedbe, ki obremenjujejo hitrost podatkovne baze, če niso pravilno sestavljene.

Optimizacija preštevanja

Preštevanje (COUNT) zadetkov v podatkovni bazi je pogosto narobe razloženo. Tako lahko na internetu najdemo veliko virov, ki opisuje napačen pristop. Preštevanje lahko deluje na dva različna načina. Lahko preštevamo dejansko število vseh vrstic ali le vrstice, ki imajo določene vrednosti (NOT NULL).

Če podamo v operacijo preštevanja ime nekega stolpca v podatkovni bazi, nam operacija vrne število vrstic, ki ima določeno vrednost. Pri uporabi vgrajenega ukaza 'COUNT(*)' ali če polje, po katerem štejemo zadetke, ne omogoča prazne vrednosti, vrne MySQL samo število vseh zadetkov. Drugi način zelo pospeši delovanje takšne poizvedbe.

Če MySQL ve, da neko polje ne more biti prazno, lahko tako poizvedbo predela v poizvedbo tipa 'COUNT(*)'. Po različnih gradivih lahko najdemo tudi veliko napačnih razlag, kako hitra operacija je preštevanje. Hitra je samo, če jo uporabljamo na pravi način, drugače lahko zavira delovanje aplikacije [9].

Če uporabimo pri preštevanju pogojni stavek, lahko s tem precej obremenimo poizvedbo.

Primer - Uporaba pogojnega stavka pri preštevanju:

```
SELECT COUNT(*)
FROM users
WHERE id>20;
# poizvedba pregleda vse vrstice v podatkovni bazi

SELECT (
      SELECT COUNT(*)
      FROM users
) - COUNT(*)
FROM users
WHERE id<20;
# poizvedba pregleda le prvih dvajset vrstic
```

Prva poizvedba pregleda vse vrstice in primerja polje s pogojem. Druga poizvedba je precej hitrejša, saj uporablja hitrejši način preštevanja vseh zadetkov, ki odšteje prvih dvajset uporabnikov. Ker je polje id primarni ključ, lahko hitro najdemo prvih dvajset zadetkov. Moramo biti pozorni, da polje, po katerem iščemo z drugim načinom, vsebuje kazalec, drugače sta oba načina enakovredna.

Optimizacija ukazov grupiranja in sortiranja

MySQL obravnava grupiranje (GROUP BY) in sortiranje (ORDER BY) podobno. Oba tipa poizvedb lahko zelo pospešimo z uporabo kazalcev. Uporaba

kazalcev je edini pravi način za optimizacijo teh tipov poizvedb. Brez uporabe kazalcev, ali če je množica rezultatov prevelika bo MySQL uporabil začasno tabelo (temporary table) ali sortiranje datotek (filesort). Pri tem se moramo zavedati, da tabele, ki jih zgenerira MySQL ne vsebujejo kazalcev in so zelo počasne.

Primer - Grupiranje uporabnikov po starosti z uporabo kazalcev in po številu prijav (logins), ki ne vsebuje kazalcev:

```
SELECT COUNT(*), logins
FROM users
GROUP BY logins; #za sortiranje uporablja za"casno tabelo.

SELECT COUNT(*), starost
FROM users
GROUP BY starost; #za sortiranje uporablja kazalec.
```

Pri združenih poizvedbah lahko grupiranje in sortiranje pospešimo z odvisnimi poizvedbami.

Optimizacija poizvedb z združevanjem

Optimizacija poizvedb z združevanjem (JOIN) je ena najbolj zapletenih stvari v MySQL-u. Hitrost teh operacij je odločilnega pomena, ne samo kot pogoj za hitro delovanje ampak kot pogoj za delovanje aplikacije nasploh. Pri takih operacijah je potrebno vedno paziti, da določenih ukazov ne uporabljamo nad njimi, če je to le mogoče (ORDER BY, GROUP BY). Vedno je dobro imeti tudi kazalec na polju, po katerem združujemo.

Če združujemo tabeli uporabniki in slo_ang je dobro imeti kazalec v drugi tabeli, po kateri združujemo. Na prvi tabeli tak kazalec ni potreben.

Optimizacija omejitve in izravnave

Poizvedbe z uporabo omejitve (LIMIT) in izravnave (OFFSET) so zelo pogoste pri sistemih, ki uporabljajo ostranjevanje (PAGING). Večina takih poizvedb uporablja tudi sortiranje za prikaz relevantnih zadetkov. Zato je nujno uporabljati kazalce na poljih, po katerih sortiramo, sicer obremenjujemo strežnik precej bolj, kot je potrebno.

Če hočemo iz podatkovne baze dobiti tudi druge podatke in ne samo tiste,

na katerih imamo kazalce, pri velikih izravnavah vržemo stran veliko več podatkov, kot jih uporabimo. Če za primer vzamemo del poizvedbe 'LIMIT 1000, 20' mora MySQL pregledati 1020 zadetkov, od katerih jih takoj 1000 zavrže. Ta poizvedba je zato precej bolj potratna, kot bi želeli. Če predpostavimo, da so poizvedbe enakomerno razdeljene po celotni tabeli, lahko ugotovimo, da v povprečju za vsako poizvedbo pregledamo polovico tabele.

Primer - Poizvedba s kazalcem na polju starost:

```
SELECT starst, ime, priimek
FROM users
ORDER BY starost
LIMIT 10000, 20;

SELECT starost
FROM users
ORDER BY starost
LIMIT 10000, 20;
```

Druga poizvedba lahko uporablja kazalec, zato je hitrejša od prve. Če pogledamo zahtevnost obeh poizvedb, lahko opazimo, da druga poizvedba pregleda 100.020 vrstic in uporablja kazalec, medtem ko prva poizvedba pregleda celotno tabelo (380.000 zapisov), ki jih predhodno še sortira. V drugem primeru se poizvedba izvrši v 0.03 sekundah, v prvem pa traja 1.5 sekunde. Ker v večini primerov potrebujemo več podatkov iz podatkovne baze, lahko drugo poizvedbo pospešimo z uporabo združenih poizvedb. Pri tem primeru jo bomo spremenili, tako da bo vrnila zadetke sortirane po polju id, ki je primarni ključ. Ostalih podatkov ne moremo omejevati na tak način.

Primer - Uporaba kazalca pri odvisnih poizvedbah:

```
SELECT ime, priimek, id
FROM users
ORDER BY id
LIMIT 1000000, 20;

SELECT ime, priimek
FROM users
INNER JOIN
    ( SELECT id
      FROM users
      ORDER BY id
      LIMIT 100000, 20
  ) AS lim USING(id);
```

Spet smo omogočili MySQL-u uporabo kazalca s posebno odvisno poizvedbo. Druga poizvedba je hitrejša, ker v odvisni poizvedbi izberemo pravilne zadetke, nato pa z uporabo združevanja poizvedb izberemo še ostala polja, ki jih potrebujemo.

Včasih lahko namesto omejevanja poizvedbe uporabimo iskanje po določenem območju (RANGE SCAN).

Primer - Iskanje po določenem območju:

```
SELECT ime, priimek, id
FROM users
ORDER BY id
LIMIT 100000, 20;

SELECT ime, priimek, id
FROM users
WHERE id
BETWEEN 100000 AND 100020;
```

Podobno kot v prejšnjih dveh primerih, smo pospešili poizvedbo. Druga poizvedba lahko uporabi kazalec primarnega ključa, zato mora pregledati samo 24 zapisov.

Pri izdelavi in vzdrževanju podatkovne baze in aplikacije je potrebno testiranje različnih poizvedb, saj se vsaka obnaša drugače in je odvisna od podatkov v podatkovni bazi.

Optimizacija ukaza SQL_CALC_FOUND_ROWS

Ukaz je zelo priljubljen pri ostranjevanju. Če poizvedba vrne zelo malo zadetkov, mora kljub temu pregledati celotno tabelo in prešteti število vseh zapisov. Ostale zadetke zavrže. Uporaba tega ukaza je zelo obremenjujoča za podatkovno bazo. Ko potrebujemo n vrstic, je bolje, da izberemo n+1 vrstic. Če nam poizvedba vrne n+1 vrstic, lahko zgeneriramo povezavo na naslednjo stran, sicer smo prišli do konca. Drugi način je izbira več strani naenkrat, npr. 10 strani in shranjevanje le teh v predpomnilnik. V tem primeru lahko v aplikaciji napišemo, da smo našli več kot 10 strani.

Če nas noben od alternativnih načinov ne zadovolji, je še vedno bolje uporabljati ločeno poizvedbo z ukazom preštevanja (COUNT(*)).

2.2.3 Vodenje MySQL pospeševalnika

MySQL ima nekaj vgrajenih ukazov, s katerimi lahko spreminjamo delovanje pospeševalnika, če nismo zadovoljni z rezultati ali delovanjem poizvedbe.

HIGH_PRIORITY in LOW_PRIORITY

S tem ukazom nastavimo prioritete določeni poizvedbi. Če nastavimo visoko prioriteto, se bo poizvedba izvršila pred ostalimi, ki se takrat poganjajo, z nizko prioriteto pa se postavi na konec čakalne vrste. Prioriteto lahko nastavljamo vsem tipom poizvedb, vendar moramo biti pazljivi. V določenih primerih se lahko namreč izbirni stavek izvrši, preden vstavimo podatke, od katerih je odvisen.

DELAYED

Ukaz uporabljamo pri operacijah vstavljanja novih zapisov v podatkovno bazo. Ukaz povzroči zakasnitev poizvedbe, dokler ni nobenega ukaza, ki bi imel prednost, v vrsti. Ukaz je primeren takrat, ko uporabnik izvede neko akcijo, ki povzroči veliko novih zapisov ali sprememb že obstoječih zapisov v podatkovni bazi, nima pa potrebe po čakanju rezultatov akcije.

STRAIGHT_JOIN

Z njim ukažemo MySQL-u izvajanje združevanja tabel v takem vrstnem redu, kot smo ga sami zapisali. Ukaz je primeren, ko MySQL ne izbere pravega vrstnega reda združevanja ali če pospeševalnik potrebuje preveč časa za določanje vrstnega reda.

SQL_SMALL_RESULT in SQL_BIG_RESULT

Uporabljamo ga pri izbirnih stavkih. Z njim povemo pospeševalniku, kdaj naj uporablja začasne tabele in sortiranje pri poizvedbah z grupiranjem ali izborom različnih vrednosti (DISTINCT). Prvi ukaz pove, da bo množica rezultatov majhna in ni potrebe po začasnih tabelah, ravno obratno pa je pri drugem ukazu.

SQL_CACHE in SQL_NO_CACHE

Ukaz se uporablja za shranjevanje poizvedb v predpomnilnik. Če želimo uporabljati ta dva ukaza, moramo imeti v nastavitvah MySQL določeno ročno

shranjevanje poizvedb v predpomnilnik.

2.3 Uporaba kazalcev

Osnove in tipi kazalcev

Kazalci so najbolj pomembna stvar pri optimizaciji podatkovne baze. Od njih je v veliki meri odvisna hitrost podatkovne baze in posledično aplikacije. Če na njih pozabimo ali jih narobe uporabimo, bomo imeli težave. Te težave verjetno ne bodo nastopile takoj, ampak se bodo pojavile kasneje, ko bo podatkovna baza že vsebovala večjo količino podatkov. Če se pri rasti podatkovne baze hitrost aplikacije nenadoma kritično zmanjša, je to pokazatelj slabo postavljenih kazalcev.

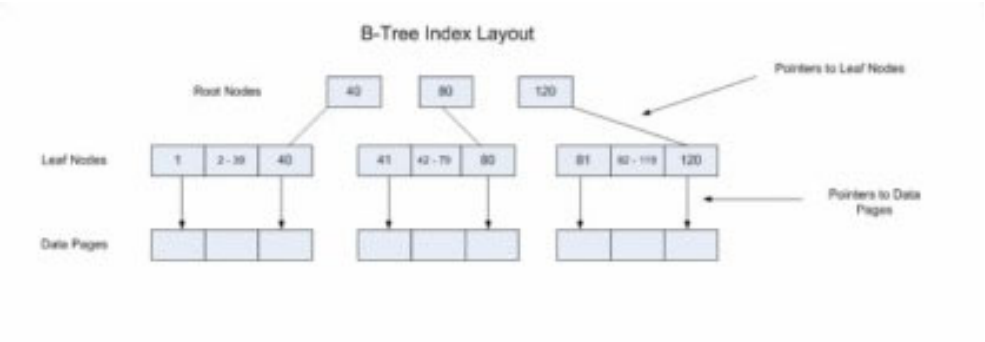
Kazalce si najlažje predstavljamo kot telefonski imenik ali kazalo v knjigi. Kot mi pogledamo kazalo in poiščemo željeno stran v knjigi, tako tudi MySQL razporedi svoje podatke. Ko MySQL najde kazalec, ki kaže na iskano vrednost, lahko neposredno dostopa do te vrednosti. Kazalec vsebuje vrednosti določenega polja ali več polj v tabeli. Če naš kazalec vsebuje več polj v podatkovni bazi, je zelo pomemben vrstni red polj v kazalcu. MySQL lahko uspešno pospeši iskanje podatkov samo za prvo polje v kazalcu. Kazalec nad dvema poljema v tabeli ni enak dvema kazalcema nad istimi polji.

B-Tree kazalec

Ko govorimo splošno o kazalcih v MySQLu, verjetno mislimo B-Tree kazalec. Je največkrat uporabljan kazalec, saj je podprt pri skoraj vseh tipih tabel.

Ideja tega kazalca je, da so vse vrednosti shranjene v zaporedju od leve proti desni in enako oddaljene od korena. Poznamo tudi posebno obliko B-Tree kazalca, ki se imenuje B+Tree. Razlika je v tem, da bloki na zadnjem nivoju (listi), vsebujejo kazalec na naslednji blok. Listi se razlikujejo od drugih blokov po tem, da vsebujejo kazalce na dejanske podatke v podatkovni bazi. Globina takšne drevesne strukture je odvisna od velikosti tabele.

Ker ima ta tip kazalca urejene podatke po vrstnem redu, je uporaben za iskanje določenega območja podatkov (npr. uporabnike, ki so starejši od 20 in mlajši od 30 let).



Slika 2.1: B-Tree - prikaz iskanja po kazalcu

Primeri:

Imamo tabelo z naslednjimi podatki:

1. ime
2. priimek
3. st_besed
4. ...
5. key(ime, priimek, st_besed)

Zadnji podatek (key) je kazalec tipa B-Tree. Kot primer bomo uporabili tabelo s preko 1.000.000 zapisi.

Iskanje po celotni vrednosti:

```
SELECT id
FROM uporabniki
WHERE ime='Janez'
AND priimek='Krajnski'
AND st_besed=105 ;
```

Če nad to poizvedbo uporabimo vgrajen ukaz za razlaganje poizvedb (EXPLAIN) ugotovimo, da bo v primeru kazalca MySQL preiskal v najslabšem primeru 1535 zapisov, kar je velikost posameznega bloka. V primeru, da tega kazalca ne bi imeli, bi poizvedba v najslabšem primeru preiskala celotno tabelo.

Ujemanje z levim delom:

```
SELECT id
FROM uporabniki
WHERE ime='Janez' or ime LIKE('Janez\%');
```

Zahtevnost poizvedbe je enaka kot v zgornjem primeru. Pri uporabi kazalca MySQL pregleda en blok podatkov, sicer celotno tabelo. To deluje zato, ker iščemo po imenu uporabnika, ki je prvi argument kazalca. Če namesto imena uporabnika iščemo po priimku, nam kazalec ne pomaga več.

Ujemanje določenega območja:

```
SELECT id
FROM uporabniki
WHERE ime > 'C'
    AND ime < 'D'
```

Tudi v tem primeru nam kazalec pomaga. Poizvedba zato z uporabo kazalca namesto milijona vrstic pregleda le 55298 vrstic (30 blokov). Tudi ta poizvedba deluje le v primeru, ko iščemo po imenu uporabnika kot prvemu argumentu kazalca. V primeru, da bi iskali po priimku ali vpisanih besedah, nam kazalec ne koristi.

Ujemanje po delu drugega argumenta:

```
SELECT id
FROM uporabniki
WHERE ime='Janez'
 AND priimek LIKE('Kra\%')
```

Poizvedba uporabi celoten prvi argument in del drugega argumenta. Ker sta oba argumenta na začetku, lahko MySQL koristno uporabi kazalec in pospeši poizvedbo.

Za večje aplikacije na spletu je običajno, da naredimo več kazalcev z istimi argumenti in različnim vrstnim redom, da zadovoljimo vse poizvedbe.

Hash kazalec

Kazalec je sestavljen iz zgostitvene tabele (hash table). Uporaben je le za specifične poizvedbe, ki uporabljajo vse argumente kazalca (exact lookups).

Za vsako vrstico MySQL izračuna zgostitveno kodo, ki je majhna in se po vsej verjetnosti razlikuje od ostalih kod z drugačnimi podatki. Pri MySQL-u omogoča to vrsto kazalcev le spominski (HEAP) tip podatkovnih tabel. Kljub temu ima lahko ta tip tabel tudi druge tipe kazalcev. Izjema MySQL-a je, da je lahko koda dveh različnih podatkov enaka. V tem primeru MySQL preveri dobljen podatek z zahtevanim.

Omejitve kazalca:
Kazalec nam ne pomaga pri sortiranju, saj podatki niso urejeni. Ne moremo iskati samo po delu ključa ampak po celem ključu.

Dostop do podatkov je načeloma zelo hiter. Dostop se upočasni v primeru, če imamo veliko podvojitev (collision). Če imamo veliko podatkov z isto kodo, je potrebno vsak dobljen podatek preveriti z iskanim podatkom.

Če naš tip podatkovne tabele ne podpira tega kazalca, lahko kreiramo svojo zgostitveno kodo z uporabo zgostitvenih funkcij (MD5, CRC32 ...). Ker je to polje načeloma krajše od drugih tekstovnih polj, lahko nad njim zgeneriramo B-Tree kazalec. Ta način je uporaben pri iskanju daljših nizov v podatkovni bazi. Uporabimo ga lahko za iskanje internetnih naslovov, e-mail naslovov itd. V primeru, da bi B-Tree kazalec zgenerirali nad dolgimi nizi (e-mail, internetni naslov), bi bil nekajkrat večji. Če postane kazalec prevelik, je iskanje preveč potratno. V skrajnem primeru bi potrebovali kazalec na kazalce. Pri uporabi tega kazalca se moramo zavedati, da lahko pride do podvojitev.

Primer - Uporaba zgostitvenih kod:

```
SELECT *
FROM users
WHERE crc_email = CRC32('janez.kranjski@mail.si');

SELECT *
FROM users
WHERE crc_email = CRC32('janez.kranjski@mail.si')
   AND email = 'janez.kranjski@mail.si';
```

Pri prvi poizvedbi lahko pride do več enakih zgostitvenih kod. Zaradi tega lahko prva poizvedba vrne več rezultatov, med katerimi bo samo eden pravilen. Temu se lahko izgognemo tako, kot prikazuje druga poizvedba. Zaradi prvega pogoja bo poizvedba delovala hitro, drugi pogoj pa bo izločil napačne podatke z isto zgostitveno kodo.

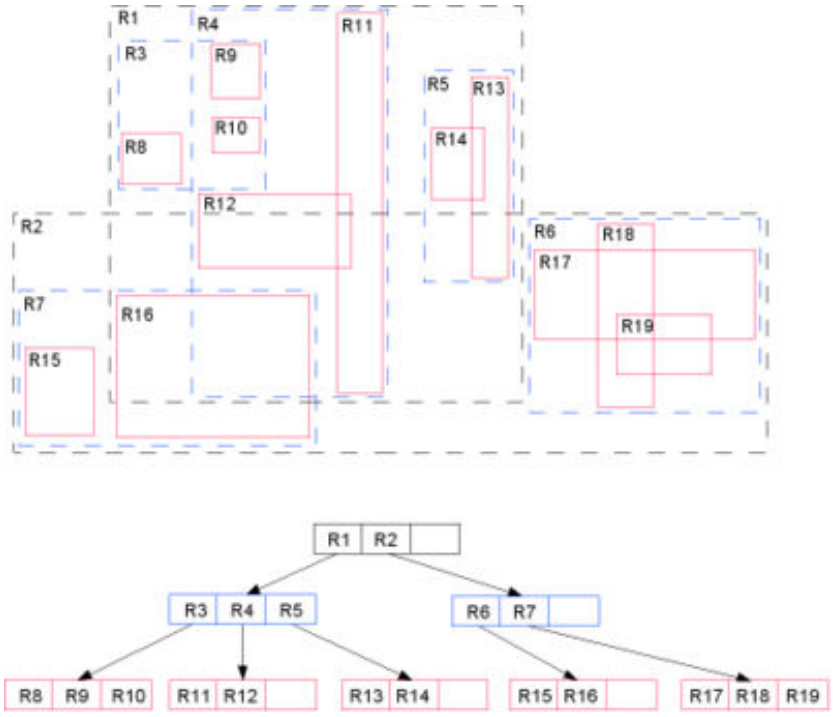
R drevo ali prostorski kazalec

R drevesa (R-Tree) [2] so po delovanju podobna B drevesom. Namenjena so iskanju po večdimenzionalnem polju. Delujejo po sistemu popolnoma uravnoteženega drevesa. Vsako vozlišče, ki ni na zadnjem nivoju (list), vsebuje vsaj dva naslednika.

Tipične lastnosti R dreves:

1. Vsako notranje vozlišče ima vsaj dva, lahko pa tudi več naslednikov.
2. Drevo je popolnoma uravnoteženo, listi so na isti višini.
3. Vsak končni list vsebuje kazalec na objekt.

R drevesa so zelo učinkovita. Uporabljajo se pri obdelavi geoloških baz podatkov, kjer se ostali principi ne obnesejo tako dobro.



Slika 2.2: Prikaz R-Tree drevesa

Kazalec po celotnem besedilu

Kazalec po celotnem besedilu (Full Text Index) je poseben tip kazalca, katerega uporaba je mogoča le v MyISAM tabelah. Uporablja se za iskanje ključnih besed v tekstovnih poljih. Podatke izbira po pomembnosti (relevance) in ne po primerjavi iskanih besed. Zahtevajo posebno obliko poizvedb in tudi delovanje iskanja s tem kazalcem se precej razlikuje od ustaljene prakse MySQL-a. Spletni iskalniki večinoma ne uporabljajo relacijskih podatkovnih baz, vendar delujejo po podobnih načelih. S tem kazalcem lahko iščemo samo po tekstovnih (text) in znakovnih (char, varchar) poljih v podatkovni bazi.

Sintaksa poizvedbe, ki uporablja kazalec po celotnem besedilu se rahlo razlikuje od standardne sintakse MySQL-a. Kazalec ne hrani informacije o tem, v katerem stolpcu se iskana beseda nahaja, zato ne moremo določiti pomembnosti posameznih stolpcev. Lahko pa nastavimo vrednost posamezne besede pri sortiranju in s tem delno odpravimo težavo. V primeru spletnih iskalnikov je ta informacija zelo pomembna, saj je vrstni red prikazanih povezav odvisen od tega, če se iskana beseda nahaja v domeni, naslovu strani ali kje drugje v tekstu.

Iskanje po celotnem besedilu se izvede, tako da MySQL določi ustreznost posameznega zadetka glede na poizvedbo. Besede, ki se v podatkovni bazi ne pojavljajo pogosto, so bolj pomembne. Besede, ki se pojavljajo prepogosto, MySQL izloči iz kazalca in jih ne išče. Za prepogoste besede šteje tiste, ki se v podatkovni bazi pojavljajo v več kot polovici zapisov.

Poznamo tudi seznam neustreznih besed (stopword list), ki na iskanje nimajo nobenega vpliva. To so vezniki, predlogi, členki, medmeti itd., ki so standardni sestavni deli nekega jezika in za iskanje niso pomembni. V MySQL-u so te besedne vrste določene samo za angleški jezik, zato moramo te besede dodati v seznam neustreznih besed. Dodajanje v seznam neustreznih besed ni obvezno, saj lahko kljub temu uspešno poganjamo poizvedbe, vendar ima velik vpliv na hitrost poizvedbe.

Primer - Poizvedbe za besedne fraze, ki vsebujejo besedi “going“ in “kako“ v slovenskem ali angleškem prevodu z uporabo kazalca po celotnem besedilu:

```
SELECT beseda, prevod,
    MATCH(beseda, prevod)
    AGAINST ('going kako')
    AS pomembnost
FROM anglesko_slovenski
WHERE MATCH (beseda, prevod) AGAINST ('going kako')
```

Če ta primer poženemo nad podatkovno bazo slovarja bomo dobili naslednje rezultate:

Rezultati poizvedbe		
How are you? (polite form)	kako se imate? kako ste?	9.93891906738281
How are you?	kako se imaš, kako si?	8.85460758209229
how	kako	7.12220478057861
what is your name	kako ti je ime?	7.12220478057861
how is it going	kako ti gre?	7.12220478057861
what’s your name	kako ti je ime?	7.12220478057861
what is your family name	kako se pišeš	7.04214096069336

Tabela 2.1: Rezultati uporabe kazalca po celotnem besedilu.

Ukaz ujemanja (MATCH) izračuna pomembnost zadetka za iskano poizvedbo. Ukaz ujemanja smo v poizvedbi uporabili dvakrat, vendar to ne vpliva na njeno hitrost, saj MySQL prepozna, da gre za enak ukaz in ga ne izvaja dvakrat. Velik vpliv na pomembnost zadetka ima število ponovitev iskane besede v podatkovni bazi. Manjkrat, ko se določena beseda ali besedna zveza ponovi, večja je njena teža in višja pomembnost zadetka. Vrstni red besed v ukazu ujemanja mora biti enak vrstnemu redu v kazalcu.

Če želimo, da ima neka beseda večjo težo od druge, lahko priredimo poizvedbo.

Primer poizvedbe, pri kateri je beseda vredna dvakrat več kot prevod:

```
SELECT beseda, prevod,
      MATCH(beseda, prevod)
      AGAINST ('going kako')
      AS pomembnost
FROM ang_slo
WHERE MATCH (beseda, prevod)
      AGAINST ('going kako')
ORDER BY (2 * MATCH(beseda)
                AGAINST ('going')
          ) +
      MATCH(prevod) AGAINST ('kako') DESC;
```

V tem primeru je angleška beseda dvakrat bolj pomembna kot slovenski prevod, zato ima beseda dvakrat večjo težo pri razvrščanju rezultatov.

Posebna oblika kazalca po celotnem besedilu

Posebna oblika kazalca po celotnem besedilu (Boolean) se rahlo razlikuje od navadnega kazalca po celotnem besedilu. Še vedno uporablja seznam neustreznih besed, ne uporablja pa nastavitev za najmanjšo (ft_min_word_len) in največjo dolžino (ft_max_word_len) iskanega niza. Ta oblika nam omogoča uporabo nekaterih dodatnih parametrov za iskanje:

1. Niz - rezultate, ki vsebujejo to besedo uvrstimo višje.
2. niz - rezultate, ki vsebujejo to besedo uvrstimo nižje.
3. +niz - rezultat mora vsebovati besedo 'niz'.
4. -niz - rezultat ne sme vsebovati besede 'niz'.
5. Niz* - rezultate, ki se začnejo na besedo 'niz' uvrstimo višje.

Sedaj lahko z uporabo dodatnih parametrov iz zgornje poizvedbe izločimo zadetke, ki vsebujejo besedo 'family':

```
SELECT beseda, prevod,
   MATCH(beseda, prevod)
   AGAINST ('going kako')
   AS pomembnost
FROM anglesko_slovenski
   WHERE MATCH (beseda, prevod)
     AGAINST ('+going +kako -family' IN BOOLEAN MODE)
```

Rezultati poizvedbe		
How are you? (polite form)	kako se imate? kako ste?	9.93891906738281
How are you?	kako se imaš, kako si?	8.85460758209229
how	kako	7.12220478057861
what is your name	kako ti je ime?	7.12220478057861
how is it going	kako ti gre?	7.12220478057861
what's your name	kako ti je ime?	7.12220478057861

Tabela 2.2: Rezultati uporabe posebne oblike kazalca po celotnem besedilu.

Kot lahko opazimo v tabeli, so dobljeni rezultati skoraj enaki prejšnji poizvedbi. Manjka samo zadnji rezultat, ki smo ga izločili iz zadetkov.

Pri tem kazalcu je lahko spreminjanje, brisanje in dodajanje novih zapisov zelo potratno. Spreminjanje zapisa, ki vsebuje sto besed, ne zahteva spremembe kazalca le za en zapis, ampak povzroči spremembo stotih zapisov. Kazalci po celotnem besedilu so izpostavljeni razdrobljenosti (fragmentaciji) in niso stabilni. Zato moramo tabele s temi kazalci velikokrat optimizirati (z ukazom OPTIMIZE TABLE). Če želimo uporabljati urejanje in obdržati hitrost poizvedbe, je edina možnost urejanje po pomembnosti. Če urejamo po katerem drugem podatku, MySQL uporabi sortiranje datotek (filesort).

Poizvedbe lahko pospešimo, tako da določimo dober seznam neustreznih besed ali povečamo najmanjše dovoljeno število znakov (ft_min_word_len). Če povečamo najmanjše dovoljeno število znakov, bo poizvedba prišla hitreje do rezultatov, saj bo število zadetkov manjše. S tem lahko izločimo tudi tiste rezultate, ki jih iščemo. Če za primer vzamemo LCD televizorje in minimalno štiri znake za iskanje, bomo pri iskanju LCD televizorjev dobili tudi vse druge televizorje.

Če v tabelo uvažamo večjo količino podatkov, je smiselno začasno kazalec izključiti. Ko je uvažanje končano, ga ponovno vključimo. Zaradi potratnega vstavljanja je tak pristop precej hitrejši.

2.4 Pogledi

Pogledi (VIEWS) so uporabni takrat, ko neko zapleteno poizvedbo potrebujemo na več mestih v aplikaciji. Pri MySQL-u jih imamo možnost uporabljati

šele v zadnji verziji (MySQL 5.0). Pogled si lahko predstavljamo kot virtualno tabelo. Na pogled so enaki tabelam, vendar se moramo zavedati, da ne hranijo podatkov. Hranijo le poizvedbo, na podlagi katere se kreirajo dinamično. Sestavljeni so lahko iz ene ali večih tabel ali pogledov. Poznamo dva algoritma - algoritem združevanja (MERGE) in algoritem začasnih tabel (TEMP TABLE).

Algoritem združevanja je boljši in ga MySQL uporabi, če je le možno. Pri tem algoritmu MySQL združi našo poizvedbo s poizvedbo pogleda in jo požene nad fizično tabelo podatkov. Če MySQL ne more uporabiti algoritma združevanja, uporabi algoritem začasnih tabel. Ta algoritem je bistveno počasnejši in večinoma uporablja vpeljane (derived) tabele. V tem primeru se naša poizvedba izvede ločeno od poizvedbe pogleda. Podatki pogleda se shranijo v začasni tabeli, nad katero se požene naša poizvedba.

MySQL uporabi algoritem začasnih tabel v primeru uporabe ukazov, ki ne omogočajo povezave ena na ena med pogledom in fizično tabelo (DISTINCT, GROUP BY, UNION ...).

Primer - Pogled, ki vrne število vpisanih besed za posameznega uporabnika:

```
SELECT user, count(*) as cnt
FROM anglesko_slovenski
GROUP BY user
```

Če sedaj preverimo zahtevnost poizvedbe za število besed točno določenega uporabnika, dobimo naslednje rezultate:

SELECT * FROM tmp_view WHERE user = 1							
id	select_type	table	type	key_len	ref	rows	extra
1	PRIMARY	derived2	ALL	NULL	NULL	78	Using where
2	DERIVED	ang_slo	ALL	NULL	NULL	25573	where, temp, file

Tabela 2.3: Uporaba pogleda, pri prikazu števila besed točno določenega uporabnika.

Podatke v pogledih lahko tudi spreminjamo, brišemo ali dodajamo, enako kot v tabele. Te operacije lahko izvajamo le nad pogledi, ki uporabljajo algoritem združevanja. Izvajamo jih lahko tudi nad pogledi, ki združujejo več tabel (JOIN), spreminjamo pa lahko le podatke, ki so vezani na eno samo tabelo. Pri tem se moramo zavedati, da pogledi niso fizične tabele, zato se bodo vse

SELECT COUNT(*) FROM ang_slo WHERE user = 1 GROUP BY user							
id	select_type	table	type	key_len	ref	rows	extra
1	SIMPLE	ang_slo	ALL	NULL	NULL	25573	Using where

Tabela 2.4: Zahtevnost poizvedbe brez uporabe pogleda.

spremembe nad pogledi poznale tudi v tabeli in vseh drugih pogledih, ki so vezani na ta podatek.

Ukaz CHECK OPTION - poseben ukaz, ki ga lahko uporabimo, ko naredimo nov pogled. S to opcijo preprečimo poizvedbam, da bi v pogledu spremenile podatek, tako da ta ne bi več ustrezal pogojem poizvedbe pogleda.

Pri uporabi pogledov moramo biti zelo previdni, saj lahko slabo zasnovani pogledi zelo poslabšajo hitrost podatkovne baze. Dobro zasnovani pogledi lahko pospešijo delovanje podatkovne baze.

2.5 Sprožilci

Sprožilci (triggers) nam omogočajo izvajanje neke kode, preden poženemo poizvedbo. Za vsako posamezno akcijo (vstavljanje, spreminjanje, brisanje) nad določeno tabelo lahko naredimo samo en sprožilec [2]. Pri sprožilcih lahko uporabljamo dva ukaza (NEW, OLD). Z njima določimo, na katero vrstico se nanaša. Z ukazom 'OLD' se sklicujemo na obstoječi zapis, preden izvedemo brisanje ali spreminjanje. Z ukazom 'NEW' pa se sklicujemo na trenutni zapis, preden ga vstavimo ali spremenimo. Pri vstavljanju novega zapisa lahko uporabimo samo ukaz 'New', ker pred zapisom novega podatka, le ta ne obstaja v podatkovni bazi. MySQL vsebuje sprožilce šele od verzije 5.0.2 dalje.

Poglavje 3

Izvedba optimizacije - www.spletni-slovar.com

Spletni slovar je internetna aplikacija, namenjena iskanju prevodov besed in dodajanju novih besed, kar je osnovna ideja. Omogoča tudi prevajanje daljših besedil s pomočjo orodja Google Translate in izgovorjavo besed in besedil (TTS - Text to speech) z orodjem vozMe, kar je novost na našem tržišču. Prevajanje besedil in izgovorjava besed je dodana vrednost, ki pomaga uporabnikom pri prevajanju ali učenju tujih jezikov in ni bila zajeta v osnovni ideji.

Slovar je zasnovan interaktivno. Za iskanje prevodov registracija ni potrebna. Za uporabnike, ki želijo v podatkovno bazo dodati nove prevode, je registracija obvezna. Za vsako besedo, ki jo je uporabnik dodal, hranimo tudi unikatno število (unique id), ki predstavlja uporabnika v podatkovni bazi. Tako lahko za vsakega uporabnika hitro preverimo primernost dodanih besed in jih odstranimo, če niso primerne. Uporabniška imena uporabnikov, ki so dodali največ novih besed prikazujem na prvi strani slovarja. S tem želim dodatno motivirati ostale uporabnike za vnos novih prevodov.

V posebni tabeli hranimo tudi seznam angleških besed in fraz, ki jih prikazujemo na prvi strani. Uporabniku prikažemo naključno besedo v angleščini in ga sprašujemo po slovenskem prevodu. Če ne pozna prevoda prikazane besede, lahko besedo zamenja. Ko uporabnik predlaga prevod besede ali fraze, se le ta izključi (deaktivira). Pri tem načinu dodajanja besed registracija ni potrebna, zato vse vnose pregleda urednik. Potrjene vnose shranimo, slovenske prevode pa prikazujemo pri drugih slovarjih (italijanski, hrvaški, španski ...), kjer sprašujemo po prevodu te besede v jeziku, ki ga je uporabnik izbral.

Pregledovanju vnosov je namenjen uredniški kotiček. V uredniškem kotičku prikažemo iskane besede in podane prevode. Če je prevod pravilen, ga urednik potrdi, sicer ga lahko popravi ali zavrne. Če urednik prevod zavrne, se v tabeli angeških besed in fraz prevod znova vključi (aktivira) in je na voljo za prikaz na prvi strani.

Pri iskanju prevodov lahko uporabnik prijavi napačen prevod. Napako shranimo, da lahko besedo kasneje preverimo. Besedo do pregleda prikazujemo enako kot prej, v posebnem stolpcu pa prikazujemo število prijavljenih napak.

Poleg vseh navedenih možnosti, ki jih slovar nudi, nameravamo v prihodnje njegovo delovanje izboljšati. Prva stvar, ki jo bomo spremenili, bo nov izgled (design) spletne strani, saj je bila celotna stran narejena kot osnova za diplomsko nalogo in izgled strani ni imel bistvenega pomena.

Stran je že odlično obiskana, njen obisk še raste, zato bo treba izgled prilagoditi in izboljšati uporabniško izkušnjo (user experience) na strani.

Veliko internetnih uporabnikov ne pozna tehnologije Ajax, ki besede prevaja sprotno, ko uporabnik tipka. Zato smo naredili manjšo raziskavo na desetih uporabnikih in spremljal njihove odzive pri prvem obisku te strani. Devet od desetih se na strani ni znašlo in niso opazili, da se je prevod iskane besede že pojavil spodaj, ampak so vztrajno iskali gumb, s katerim bi prevedli iskano besedo. Zato bomo z novim izgledom tudi delno ukinili Ajax. Uporaba slovarja po starem načinu bo še vedno mogoča. To bomo naredili, tako da si bo lahko registriran uporabnik izbral uporabo sprotnega prevajanja. Pri tej izbiri se mu bo v računalnik naložil piškotek (cookie). Ko bo prišel naslednjič na spletno stran, bo njegov izbrani način še vedno deloval, ne da bi se moral za to prijaviti v sistem.

3.1 Rešitve s strani aplikacije

Naključni izbor neke vrednosti iz podatkovne baze predstavlja veliko oviro pri hitrem delovanju aplikacije. Kljub temu smo hoteli na prvi strani prikazati popolnoma naključno besedo iz tabele angleških besed in fraz, ki zajema preko 180.000 zapisov. Taka poizvedba je precej potratna in se izvede vsakič, ko nek uporabnik odpre prvo stran.

Poizvedba:

```
SELECT angleska_beseda
FROM anglesko_hrvaski
WHERE preveden=0
ORDER BY rand() LIMIT 1;
```

Opomba: anglesko_hrvaski je tabela angleško hrvaških prevodov, iz katere prebiramo angleške besede in fraze, ki jih uporabniki prevajajo v slovenščino. Ko je beseda prevedena v slovenščino, vprašamo uporabnika še po drugih prevodih (italijanskem, francoskem ...).

Če zgornjo poizvedbo poženemo nad dejansko podatkovno bazo s 180.000 prevodi, traja poizvedba povprečno 0.4 sekunde. Ker se poizvedba izvede vsakič, ko uporabnik prikaže prvo stran, je prezahtevna. Če vzamemo povprečno število prikazov prve strani (9 000 prikazov), lahko izračunamo, da bi izvajanje takih poizvedb skupno trajalo eno uro. Poizvedbe ne moremo shraniti v predpomnilnik, ker uporablja funkcijo RAND, ki vrne vsakič drugačen zadetek. V tem primeru nam tudi kazalec ne pomaga, saj je večina prevodov zaenkrat še neprevedenih.

V tem primeru je bilo najbolje del rešitve prenesti na aplikacijo. Tako smo v php kodi določili spodnjo in zgornjo mejo, kjer bomo izbirali naključno besedo. Poizvedba je tako spremenjena in zajema le 30 besed.

Poizvedba:

```
SELECT angleska_beseda
FROM anglesko_hrvaski
WHERE preveden=0
    AND id>97291
    AND id<97321
ORDER BY rand() LIMIT 1;
```

Uspešno smo skrajšali povprečni čas poizvedbe iz 0.4 sekunde na 0.02 sekunde.

Ker je med besedami manj kot 10 % prevedenih, bi bilo smiselno tako poizvedbo še izboljšati. Če določimo naključno število v aplikaciji, lahko iščemo željeno besedo po tem ključu. V večini primerov bi dobili nazaj besedo, ki še ni prevedena.

Poizvedba:

```
SELECT angleska_beseda
FROM anglesko_hrvaski
WHERE preveden=0
    AND id = 97299 LIMIT 1
```

Povprečen čas trajanja te poizvedbe je 0.0003. Zato bi bilo smiselno tako poizvedbo tudi ponoviti, če v prvem iskanju ne bi prišli do rezultata. Slovar bomo torej spremenili, tako da bo iskal z zadnjo poizvedbo, vendar največ trikrat. Če v treh poizkusih ne bomo prišli do rešitve, bomo sprožili iskanje po območju. Ta rešitev se bo trenutno najbolje obnesla, vendar jo bo potrebno spremeniti, ko bo prevedenih besed v podatkovni bazi preveč.

Poleg branja neprevedenih besed smo v aplikaciji naredili tudi seznam slovarjev. Edina prednost seznama slovarjev v podatkovni bazi je v tem, da bi bilo dodajanje, urejanje in brisanje slovarjev enostavneje, saj bi te vrednosti lahko spremenili v podatkovni bazi ali pa s pomočjo urejevalnika (CMS). Ker se spremembe dogajajo redko in so slovarji v večini enaki, kot so bili ob zagonu spletne strani, ni potrebe po tem. Slabost takšnega pristopa bi bila nova poizvedba, ki bi se izvedla vsakič, ko bi uporabnik obiskal vstopno stran. Poizvedba bi bila sicer hitra, a bi po nepotrebnem obremenjevala strežnik. Kljub temu pa ima slovar seznam slovarjev v podatkovni bazi. Seznam je namenjen urednikom za preverjanje novih besed, pregledovanje prijavljenih napak itd. Če na stran dodamo nov slovar, vpišemo nekaj podatkov v bazo (kateri slovar, katera polja ...) in slovar je že viden v uredniškem delu strani.

3.2 Poizvedbe v predpomnilniku

Za iskanje prevodov se uporablja ogromno število različnih poizvedb, saj besede iščemo sprotno, za vsako črko, ki jo uporabnik vnese. Tako lahko pridemo do izračuna v tabeli 3.1, za slovensko abecedo (25 črk), kjer lahko uporabnik vnese isto črko večkrat zapored.

Če predvidimo, da uporabnik ni omejen pri vnosu, pridemo do zaključka, da je število možnih poizvedb neskončno. Kljub temu lahko s shranjevanjem poizvedb v predpomnilnik prihranimo nekaj dela strežniku. Shranjevanje lahko vključimo samo za tiste poizvedbe, za katere je to smiselno. Da to naredimo, moramo v nastavitvah vključiti možnost shranjevanja na zahtevo. Na splet-

Slika 3.1: Trenutni izgled strani.

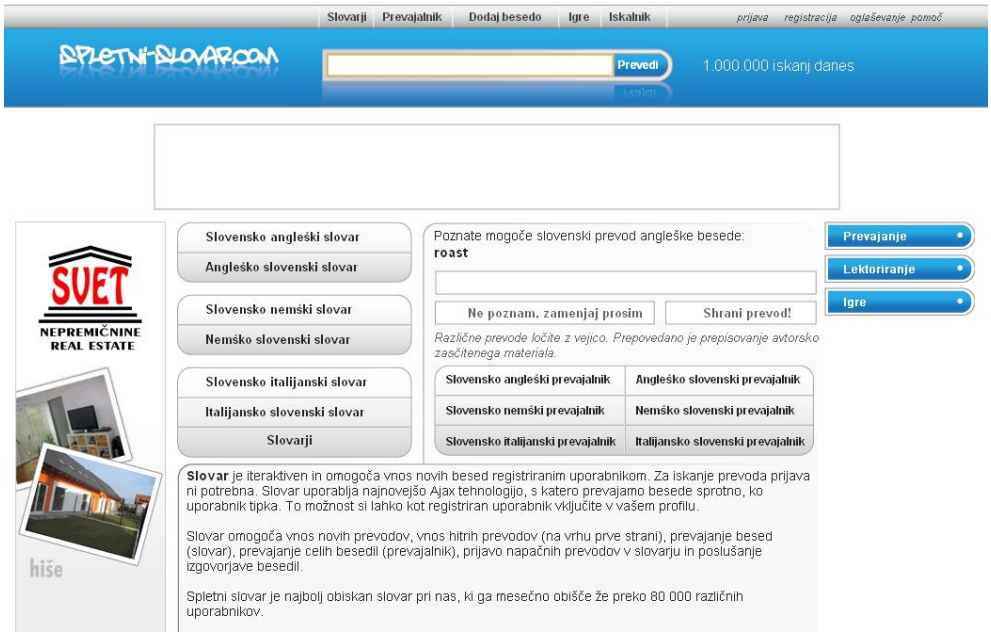
nem slovarju lahko delovanje izboljšamo s shranjevanjem poizvedbe za število registriranih uporabnikov, prikaz najbolj aktivnih uporabnikov in za prikaz števila vnesenih prevodov na vsakem posameznem slovarju. Te poizvedbe so primerne za shranjevanje, saj se ne spreminjajo in so enake za vsakega obiskovalca na strani. Novi podatki se v podatkovno bazo ne dodajajo pogosto.

Iskanje prevodov po podatkovni bazi bi lahko izboljšali, tako da se v primeru, ko uporabnik ne najde željenega prevoda, pri naslednji črki poizvedba nad podatkovno bazo sploh ne bi izvedla.

Primer - Izvajanje nepotrebnih poizvedb:

- *Uporabnik se zatipka in vpiše hišš namesto hiša.*
- *Ker ne opazi, vnese še a in je iskani niz hišša.*

Ker smo že v prvem primeru ugotovili, da iskanega niza ni v podatkovni



Slika 3.2: Nov izgled strani.

Izračun števila poizvedb	
2	$25 * 25 = 625$
3	$25 * 25 * 25 = 15625$
10	2384185791015625

Tabela 3.1: Število možnih kombinacij, glede na število vnešenih črk.

bazi, ni potrebno ponovno iskanje pri dodani črki.

3.3 Sestava tabel v podatkovni bazi

Vse tabele spletnega slovarja so MyISAM, kar je privzeta izbira MySQL-a. MyISAM smo izbrali zato, ker se dobro obnese pri branju in pisanju podatkov. Ker večina poizvedb bere iz podatkovne baze, je to prava izbira. Slabost MyISAM tabel je v tem, ker ne podpirajo transakcij. Transakcij na slovarju ne potrebujemo.

Za shranjevanje prevodov imamo na izbiro dve možnosti. Prva možnost je

shranjevanje besed vseh jezikov v eni tabeli. Pri takem pristopu potrebujemo v tabeli še eno polje, ki določa, za kateri slovar gre. Dobra lastnost takšnega pristopa je v tem, da za kreiranje novega slovarja ne potrebujemo nove tabele v bazi. Takšna rešitev nam omogoča lažje urejanje strani z orodjem za upravljanje vsebine (CMS - Content Management System). Zavedati se moramo, da v primeru vseh slovarjev v eni tabeli, tabela hitro postane velika. Iskanje po taki tabeli je časovno precej zamudno. Zato je bolje prestaviti vsak slovar v svojo tabelo. V našem primeru imamo šestnajst slovarjev. Povprečna velikost posamezne tabele je torej šestnajstkrat manjša od skupne tabele, omenjene v prvem pristopu.

Primer - nad tabelama poženemo poizvedbo:

```
SELECT id, prevod
FROM slovar
WHERE beseda LIKE (a%)
ORDER BY beseda
LIMIT 10
```

Prikaz zahtevnosti	
Št. preiskanih vrstic	Povprečno trajanje poizvedbe
291161	0.1531
25623	0.0160

Tabela 3.2: Časovna primerjava poizvedb nad posamezno in skupno tabelo.

Za primer smo uporabili tabelo, kjer so shranjeni vsi slovarji in tabelo slovensko angleškega slovarja. Vse poizvedbe iščejo samo po desni strani, saj so takšni zadetki bolj primerni. Če nekdo vpiše niz 'miz', verjetno išče mizo in ne omizje. Poleg tega, da so zadetki bolj primerni, pa je iskanje po desni strani hitrejše od iskanja po obeh straneh niza. Trajanje zgornje poizvedbe, pri iskanju po obeh straneh, je enkrat daljše. Še večje razlike pa se pojavijo pri uporabi kazalcev.

V tabeli uporabnikov so shranjeni osebni podatki o uporabniku. Za shranjevanje gesla je potrebno uporabiti kodiranje. Najbolj pogosta izbira, ki smo jo uporabili tudi pri slovarju, je MD-5. Kodirni algoritem spremeni niz, tako da se ga ne da odkodirati (enosmerno kodiranje). Na tak način pri nezaželjenem vdoru v bazo ne moremo izkoristiti gesla in uporabniškega imena. Če do

podatkovne baze dostopa več ljudi, ki dela na projektu, je kodiranje gesla nujno, da zagotovimo varnost uporabniku. Poleg uporabniškega imena in gesla hranimo še ime, priimek, naslov, e-mail, število vnešenih besed, nivo uporabnika (uporabnik, urednik, administrator), stanje (aktiven, neaktiven) in polje email_poslan, ki ga potrebujemo za pošiljanje sporočil. Pri tekstovnih poljih, za shranjevanje besedil (ime, priimek, beseda, prevod, ...), imamo na voljo dve možnosti. Prva in najbolj pogosta možnost je VARCHAR. Ta podatkovni tip je podprt v vseh relacijskih podatkovnih bazah. Večina podatkovnih baz podpira tudi podatkovni tip TEXT. Razlika med njima je v MySQL-u skoraj ničelna. Preverimo razlike na konkretnem primeru:

Primer poizvedbe:

```
SELECT *
FROM anglesko-hrvaski-slovar
WHERE angleska-beseda LIKE 'n%';
```

Trajanje poizvedbe	
Podatkovni tip	Povprečen čas
TEXT	0.07
VARCHAR	0.07

Tabela 3.3: Primerjava zahtevnosti pri različnih podatkovnih tipih.

Po primerjavi časov ugotovimo, da je poizvedba v obeh primerih enako potratna. V zgornjem primeru smo iskali po poljih brez kazalcev. Za primerjavo dodamo B-TREE kazalec na obe tabeli. Nad poljem tipa TEXT dodamo kazalec velikosti 255.

Primer poizvedbe:

```
SELECT *
FROM anglesko-hrvaski
WHERE angleska-beseda LIKE 'n%'
```

Poizvedba je še vedno enako zahtevna v obeh primerih. Rezultat poizvedb je posledica prilagajanja. Oba podatkovna tipa se namreč znata prilagoditi glede na velikost niza v polju. Če preverimo še velikost tabele “anglesko-hrvaski-slovar“ v obeh primerih, pridemo do rezultatov, ki so prikazani v tabeli.

Trajanje poizvedbe	
Podatkovni tip	Povprečen čas
TEXT	0.0008
VARCHAR	0.0008

Tabela 3.4: Primerjava zahtevnosti med različnimi podatkovnimi tipi (2).

Velikost tabele anglesko-hrvaski-slovar	
Podatkovni tip	Velikost tabele v Mb
TEXT	13,7
VARCHAR	13,5

Tabela 3.5: Primerjava velikosti tabel.

Opazimo lahko, da pri prvem podatkovnem tipu (TEXT) podatki zavzamejo nekaj več prostora. Oba podatkovna tipa rezervirata nekaj prostora za podatek o dolžini niza. Pri prvem podatkovnem tipu ta podatek zavzame dva bajta, pri drugem pa samo enega (včasih tudi dva). Razlika med obema je premajhna, da bi kakorkoli vplivala na zahtevnost poizvedb. Slabost prvega podatkovnega tipa je tudi v tem, da ga vse relacijske podatkovne baze ne podpirajo in bi lahko imeli probleme pri menjavi podatkovne baze. Ker do tega skoraj zagotovo ne bo prišlo, lahko zaključimo, da je za delovanje aplikacije vseeno, kateri podatkovni tip izberemo. Vseeno se je pri izdelavi internetnih aplikacij treba posvetiti tudi takšnim primerom in izbrati takšnega, ki se bo bolje obnesel, čeprav je razlika minimalna. Če imamo v eni tabeli več takih primerov in več tabel s podobnimi lastnostmi, je lahko razlika precej večja.

Pretvorba podatkovnih tipov je lahko hujša težava. Če imamo kazalec na določenem polju, ki je številčnega tipa, lahko pri poizvedbi s pretvorbo podatkovnega tipa pride do neuporabe kazalca. Zato je dobro, da imamo to v mislih že pri izdelavi aplikacije in da se te težave zavedamo, če naletimo na takšen primer v seznamu počasnih poizvedb (MySQL slow queries).

Seznam počasnih poizvedb se shranjuje v posebni datoteki na strežniku. Pot do te datoteke je določena v nastavitvah MySQL-a. MySQL v seznam počasnih poizvedb zapiše vse tiste poizvedbe, ki trajajo dlje (spremenljivka long_query_time), kot je določeno v nastavitvah. Do verzije 5.1.6 so se počasne poizvedbe lahko shranjevale le v datoteko. Od te verzije naprej, se shranjujejo tudi v

tabelo.

Glede podatkov, ki se nanašajo na uporabnika, smo opazili še nekaj možnosti, ki bi jih bilo treba na strani dodati. Nujen podatek, ki ga bomo hranili, je datum, ko se je uporabnik zadnjič prijavil v sistem. Če se uporabnik v sistem ne prijavi vsaj enkrat v roku dveh let, bomo njegov uporabniški račun izbrisali iz baze. Pri registraciji uporabnika bomo dodali kodo, s katero bo uporabnik potrdil, da gre za dejanskega uporabnika, ki se želi registrirati in ne za napad na spletno stran. Več o tem bo govora v nadaljevanju. Po uspešno opravljeni registraciji bo uporabnik dobil e-mail, s katerim bo lahko vključil svoj uporabniški račun. To bomo dodali zato, ker je trenutno v bazi veliko e-mail naslovov, ki so neuporabni, ker se je uporabnik zatipkal ali namerno vpisal napačen naslov. Zato pri pošiljanju sporočil vsem uporabnikom (mass mail), pride do nepotrebne obremenitve strežnika, saj se nedostavljena sporočila vračajo nazaj in poskušajo ponovno poslati.

3.4 Postavitev kazalcev v tabelah

Pri izbiri kazalca za iskanje prevoda v podatkovni bazi, se kot primeren izkaže samo B-TREE kazalec. Kazalec vedno postavimo na polje, po katerem iščemo oziroma sortiramo. V našem primeru je polje, na katerega bomo dodali kazalec, beseda, katere prevod iščemo.

Primer: Iskanje prevoda besede 'miza' v slovensko angleškem slovarju.

SELECT prevod FROM s-a WHERE beseda LIKE 'miza%' ORDER BY beseda	
B-TREE kazalec	Povprečen čas
Brez	0.1
Na prevodu	0.1
Na besedi	0.0008

Tabela 3.6: Zahtevnost iskanja z uporabo kazalca.

Če pogledamo razlago poizvedb (EXPLAIN), lahko opazimo, da nam kazalec na prevod nič ne koristi. Zato bomo uporabili kazalec na iskani besedi, saj je takšna poizvedba na slovarju tudi najbolj pogosta izbira. Z razlago

poizvedbe ugotovimo, da smo območje iskanja pravega elementa iz prejšnjih 180.000 vrstic, kar obsega vse prevode v tej tabeli, omejili na 50 vrstic.

Pri registriranih uporabnikih se pogosto izvaja poizvedba, ki prešteje vse registrirane uporabnike in izbere prve tri. To so tisti, ki so do sedaj v slovar vpisali največ prevodov. V tem primeru kazalec ni potreben, saj so te poizvedbe statične, torej se ne spreminjajo in jih lahko shranimo v predpomnilnik.

3.5 Izbira gostovanja (hosting)

Pri izbiri primernega gostovanja za spletno stran http://www.spletni-slovar.com smo morali biti pozorni na mnoge kriterije. Dokler je slovar uporabljalo le okrog 20.000 različnih uporabnikov na mesec, to ni bil tak problem in sem lahko strežnik vzdrževal sam. Ker sem imel strežnik doma, so uporabniki do njega dostopali po eni sami internetni liniji. Največji problem takšnega gostovanja je v tem, da se velikokrat zgodi kaj nepričakovanega (izpad električne energije, izpad interneta ...). Pri vsakem tovrstnem dogodku, ko stran nekaj časa ni dosegljiva, uporabniki pa potrebujejo storitev, jo nato poiščejo pri konkurenci.

Po pregledu večjega števila gostovanj, ki jih preko interneta ponujajo domača in tuja podjetja, sem izločil bistvene stvari, ki jih označujejo kot pomembne pri izbiri gostovanja. Kot glavne stvari navajajo prostor na disku, promet (bandwidth), število e-mail naslovov, velikost podatkovne baze in hitrost strežnika. Vendar pa nihče od teh podjetij, predvsem tujih, ne navaja, da je strežnik dobro imeti v domači državi, kar sem si sam določil za enega izmed glavnih pogojev, pri izbiri gostovanja. V primeru, da je strežnik v tujini, potrebujemo več časa, če je večina uporabnikov iz Slovenije (v primeru slovarja 97 %). Poleg tega se lahko vedno zgodijo nepričakovane težave po poti iz domače države do strežnika, npr. prekinjeni glavni internetni vodi med državama. Eden od glavnih pogojev pri odločitvi je tudi zaupanje ponudniku, ki gostovanje ponuja. Ponudba je že samo v Sloveniji ogromna in z nizko ceno. Veliko teh ponudnikov vam ponuja neomejen prostor na disku, neomejen mesečni prenos itd., vendar to drži samo za predstavitvene spletne strani, ki navadno ne uporabljajo podatkovne baze in ne presegajo sto različnih uporabnikov na mesec. Pomemben dejavnik pri izbiri gostovanja je dosegljivost vzdrževalcev in njihova odzivnost. Cene nikakor ne smemo določiti za pomemben pogoj pri izbiri, ko gre za dobro obiskano stran, ki ima veliko konkurence.

Poglavje 4

Varnost spletnih aplikacij - preprečevanje zlorab

Najbolj enostavna in primitivna tehnika neželjenega vdora je kraja uporabniškega imena in gesla. Ta kraja je lahko fizična v primeru, da nekdo vidi geslo, ko ga vpisujete, najde vaše podatke zapisane na listku ali pa se izvrši preko interneta. Z uporabo istega gesla in uporabniškega imena na več straneh si povečamo možnosti vdora v aplikacije, ki jih na internetu uporabljamo. Večina uporabnikov noče za vsako spletno stran, ki zahteva registracijo, uporabljati različnega uporabniškega imena in gesla, saj je takih strani ogromno in bi imel uporabnik preveč različnih podatkov.

Zato moramo za varnost osebnih podatkov poskrbeti pri izdelavi spletne strani. Tudi nam se lahko zgodi, da nam vdrejo v sistem. Pri tem je naša odgovornost še toliko večja, saj hranimo osebne podatke za več tisoč uporabnikov in ne samo svojih. Če nekdo vdre v naš sistem in prebere osebne podatke uporabnikov, dobi veliko količino gesel. Če predpostavimo, da velika količina uporabnikov uporablja za vse spletne strani in storitve iste podatke, se moramo zavedati, da lahko vsem tem uporabnikom naredimo veliko škodo. V večini primerov morajo za varnost svojih podatkov skrbeti uporabniki sami, izdelovalci in administratorji spletnih aplikacij pa moramo to področje dobro poznati, da lahko zaščitimo uporabnika, kjer je to možno.

4.1 Vrinjene poizvedbe

Spletne aplikacije nudijo uporabniku možnost prijave v sistem, dodajanja, brisanja in spreminjanja svojih prispevkov. V večini primerov se ti podatki

shranjujejo v podatkovno bazo. Pri teh procesih vedno obstaja možnost, da oseba vdre v sistem z različnimi, v večini primerov škodljivimi nameni. Z vrivanjem SQL poizvedb (SQL injections) se lahko vsiljivec okoristi z zaupnimi informacijami, spremeni shranjene podatke ali sesuje celoten sistem [5, 6].

Preprečevanje in odpravljanje ranljivih poizvedb je enostavno, vendar je lahko v aplikaciji veliko število vnosnih mask, ki jih je potrebno preveriti. Priporočljivo je vedno uporabljati poševnice (slash), ker s tem MySQL strežniku povemo, da gre za podatke.

Primer: Primer vrinjene poizvedbe pri prijavi uporabnika:

Programska koda:

```
$oUser = mysql_fetch_object(mysql_query("SELECT userid, numwords
    FROM users
    WHERE active=1
    AND username='".$_POST['up-ime'])."'
    AND password='".$_POST['geslo']."'"));

if($oUser->username) {
  //Uporabnik je logiran
} else {
  //Napacno uporabnisko ime in geslo
}
```

Vnosna maska za prijavo je navadno prva in najbolj pogosta tarča vsiljivcev. Primer prikazuje ranljivo PHP kodo, kjer se lahko vsiljivec enostavno okoristi, saj ne nudi nobene zaščite pred vdorom. Če namesto uporabniškega imena in gesla vsiljivec vpiše nek niz, npr. ' OR 1=1, se v MySQL poizvedbo doda niz, ki predstavlja nov pogoj. V tem primeru se vsiljivec na enostaven način prijavi, kot prvi registriran uporabnik. V tem primeru se požene MySQL poizvedba:

```
SELECT userid, numwords
FROM users
WHERE active=1
 AND username='' OR 1=1
```

Poznavanje strukture podatkovne baze nudi vsiljivcu še večje možnosti dostopa. Vsiljivec, ki pozna strukturo podatkovne baze, lahko do tega znanja pride na različne načine. To so lahko bivši zaposleni, ki poznajo tako strukturo baze, kot tudi ranljivosti v sistemu. Še večjo nevarnost predstavljajo programski paketi, katerih koda je javno dostopna in odprtokodni programi

za urejanje vsebin (Joomla, Drupal ...). Ravno odprto kodni programi so največkrat tarča napadov, saj so njihove slabosti javno znane. Pri teh programih moramo zato skrbeti za sprotno posodabljanje in odpravljanje lukenj. Strukturo podatkovne baze lahko vsiljivec ugotovi tudi preko imen vnosnih polj, skritih polj in parametrov, ki se prenašajo.

Če za zgornji primer predpostavimo, da vsiljivec pozna strukturo podatkovne baze, lahko prebere podatke vseh uporabnikov v sistemu, oziroma se prijavi z njihovimi podatki. To naredi čisto enostavno, ker ve, da so uporabniki oštevilčeni s poljem 'id'. Zato namesto prejšnjega niza vpiše ' OR id=n. N je število uporabnika, pod katerem je shranjen v sistemu. Vse tehnologije in programski jeziki so ranljivi za te napade.

Vrinjene poizvedbe lahko preprečujemo na različne načine. Najbolj enostaven način je brisanje narekovajev. Vendar je ta način za slovar neprimeren (tuje besede vsebujejo narekovaje) in poslabša uporabniško izkušnjo.

S strani strežnika je najbolj priljubljena uporaba funkcije 'magic_quotes_gpc'. Ta funkcija, če je vključena na strežniku, doda poševnice vsem parametrom, ki se prenašajo preko 'GET', 'POST' ali 'COOKIE'. Slabost tega pristopa je v tem, ker se vsi ti podatki ne shranjujejo v podatkovno bazo. Po drugi strani pa v podatkovno bazo shranjujemo tudi podatke, ki niso zajeti v teh treh spremenljivkah, zato ne smemo kar slepo zaupati temu pristopu. V nekaterih podatkovnih bazah je problem tudi to, da poševnica ni znak za ubežne sekvence. Največja težava tega pristopa je v tem, da bo kmalu ukinjen.

Dober način je tudi uporaba funkcije 'addslashes'. Tudi pri tem načinu je težava v tem, da poševnica ni znak za ubežno sekvenco v vseh podatkovnih bazah. Vendar pri uporabi v MySQL-u s tem ni težav. PHP nam nudi tudi funkcijo 'mysql_real_escape_string', ki očisti poizvedbo vseh neprimernih znakov. Problem nastane, če želimo prestaviti na drugo podatkovno bazo.

Številčni podatki ne vsebujejo narekovajev, oziroma jih ne smejo vsebovati, kot smo povedali že v poglavju o MySQL-u. Zato vse zgoraj naštete funkcije ne učinkujejo več. Pri številih je najboljša rešitev preverjanje vnosov in ugotavljanje, če gre res za številčni podatek.

Pomemben dejavnik za varnost podatkov v podatkovni bazi je tudi v omejevanju dovoljenj. Navadno spletne aplikacije uporabljajo popoln (root) dostop

do podatkovne baze. S tem je aplikacija izpostavljena vdorom še bolj, saj dovoljuje brisanje podatkov oziroma brisanje celotne podatkovne baze. Zato je potrebno omejiti dostope aplikaciji, oziroma njenim uporabnikom, kolikor se le da.

4.2 Križno izvajanje skript

Križno izvajanje skript (Cross site scripting oziroma XSS) je problem dinamičnih spletnih strani. Pri statičnih straneh se s tem problemom ne srečamo. Pri XSS-u napadalec doda del kode na našo spletno stran. Nova koda se kasneje prikaže uporabnikom, ko iščejo neko informacijo na naši strani. Ko uporabnik vpiše potrebne podatke, se izvede tudi nezaželjena koda, ki te podatke posreduje vsiljivcu. V večini primerov uporabnik sploh ne opazi napada, saj vsiljivci uporabijo različne zvijače.

Težavi se lahko izognemo, tako da preprečimo vnos neželjene kode na našo spletno stran. Slabši način za brisanje neželjenih vsebin je brisanje določenih znakov ali nizov (npr. <, >, <script>)[5, 6]. Boljši način za preprečevanje XSS-a je uporaba določenih funkcij, ki poskrbijo za varnost aplikacije.

V PHP-ju lahko uporabimo funkciji 'htmlentities' ali 'htmlspecialchars' pred vpisom podatkov v bazo ali pred izpisom podatkov uporabniku.

Poznamo dve vrsti XSS napadov, glede na tehniko vstavljanja neprimerne kode v aplikacijo:

1. Odsevni napad XSS.
2. Trajni napad XSS.

Pri prvem načinu mora vsiljivec prepričati uporabnika, da klikne na nek spletni naslov. V parametrih spletnega naslova se skriva zlonamerna koda, ki izkoristi slabosti aplikacije. Drugi način XSS napadov se izvede, tako da uporabnik s svojim vnosom naloži škodljivo kodo. Če mu to uspe, ni več potrebno, da uporabnik klikne nek spremenjen spletni naslov, saj se škodljiva koda že prikazuje na strani aplikacije.

Poznamo tudi lokalne XSS napade, ki so se razvili z razvojem WEB 2.0 tehnologije. Pri tej tehnologiji se veliko programske kode izvede na lokalnem

računalniku oziroma spletnem brskalniku in ne več na strežniku.

Težavam se lahko delno izognemo z preverjanjem vhodnih podatkov. Uporabimo lahko principe belega (white list) ali črnega seznama (black list). Z uporabo teh seznamov lahko določimo vse dovoljene znake, oziroma vse prepovedane znake, ki jih lahko uporabnik vnese.

Ranljivost neke spletne strani lahko ugotovimo, tako da vstavimo enostavno javascript kodo, ki izpiše neko sporočilo.

4.3 Napadi zavračanja storitve

DOS (denial of service) napadi so metoda napada spletnih strani, kjer napadalec generira ogromno količino omrežnega prometa. Količina omrežnega prometa je tako velika, da strežnik ne uspe zadovoljiti vseh podanih zahtev. Pred to vrsto napadov je klonilo že dosti velikih spletnih strani (Microsoft, Amazon, Bolha ...). Pred napadom mora napadalec okužiti zadostno število računalnikov, ki so lahko razdeljeni po celem svetu. Ko ima zadostno količino okuženih računalnikov (zombie) začne iz vseh okuženih sistemov pošiljati zahtevke na točno določeno spletno stran. Večinoma pošilja zahtevke na točno določeno skripto spletne strani. Metode za preprečevanje DOS napadov se v zadnjih letih niso kaj dosti razvile. Zato je DOS napad ena največjih nevarnosti uporabnikom, organizacijam in celo celotni spletni infrastrukturi. Vsak internetni sistem oziroma vsaka spletna stran je izpostavljena DOS napadom.

Ko pride do napada je malo možnosti, da bi ga hitro ustavili. Obstaja nekaj načinov, s katerimi si lahko pomagamo in ublažimo moč DOS napada. Če je napad usmerjen na samo en računalnik je najbolje zamenjat njegov IP naslov. V primeru, da je napad usmerjen samo na eno skripto oziroma podstran naše spletne strani, je najboljše onemogočiti izvajanje te skripte. Pri tem bomo onemogočili tudi legalno uporabo programa, vendar je to navadno majhna žrtev, ki omogoči delovanje drugih funkcij aplikacije.

Poznamo kar nekaj preventivnih metod za zaščito pred DOS napadi. Zaščititi se moramo na nivoju strežnika in ne na nivoju aplikacije. Za zaščito se uporabljajo orodja, ki preverijo in analizirajo dobljene pakete. Ta orodja preverijo izvor paketov, njihov namen in njihov cilj. Delno se lahko zaščitimo s požarnimi zidi in razdeljevanjem strežniških zmogljivosti na različne omrežne

lokacije. Proti dobro zamišljenim, organiziranim in izpeljanim DOS napadom se dejansko ne moremo zaščititi.

4.4 Napadi botov

Internetni boti ali spletni roboti so aplikacije, ki avtomatsko izvajajo določena opravila na svetovnem spletu. Takšni roboti so tudi iskalni pajki, ki jih uporabljajo iskalniki (Google.si, Bing.com, Najdi.si), saj z njimi preiščejo strukturo in vsebino spletne strani. Med takšne robote lahko štejemo tudi okužene računalnike (zombie), ki se uporabljajo pri DOS napadih.

Pri omembi spletnih robotov oziroma botov pomislimo najprej na robote, ki izvajajo sicer dovoljene operacije (registracija, vnos novih besed ...), vendar v veliko večjem obsegu in neprimerljivo hitreje, kot bi jih izvajal uporabnik. S takimi napadi lahko naredijo veliko škode, saj onesnažijo podatkovno bazo z napačnimi, lažnimi informacijami.

Najbolj učinkovita zaščita pred tovrstnim napadom je uporaba varnostne kode (CAPTCHA). Varnostna koda je zgenerirana kot slika. V tej sliki se nahaja tekst, ki ga moramo prepisati v tekstovno polje. S tem potrdimo, da gre za pravega uporabnika in ne za robota, saj robot ne more prebrati teksta, ki se nahaja v sliki. Zaradi hitrega razvoja OCR (optical charachter recognition) programov slika ne sme biti jasna. Paziti moramo, da slike ne zakrijemo preveč, saj mora biti razumljiva uporabniku.

Največji problem varnostnih kod je omejevanje uporabnika, saj mora le ta izvesti še eno dodatno operacijo. Na slovarju varnostne kode nismo uporabili, ker smo želeli pridobiti veliko število registriranih uporabnikov. Z uporabo varnostne kode bi izgubili nekaj uporabnikov, ki se ne bi registrirali, zaradi dodatnega koraka. Največji problem varnostne kode je prav v tem, da se pokvari uporabniška izkušnja. V primeru registracije to ni velik problem, če pa bi varnostno kodo uporabil pri vnosu novih besed, bi bila težava precej večja. Uporabnik bi namreč za vsak vnos nove besede porabil še enkrat toliko časa za vnos varnostne kode.

Namesto uporabe varnostne kode lahko redno delamo kopije (backup) podatkovne baze. To možnost smo tudi sami izbrali za slovar. V prihodnje bomo pri registraciji varnostno kodo namestili, saj je sedaj stran že dobro obiskana

in to ni več tak problem, kot je bil na začetku.

4.5 Vdor pri zadnjih vratih

Računalniški vdor pri zadnjih vratih (backdoor attack) je nezakonit vdor do računalnika, kadar gre za pomanjkljivo varnost računalnika, za katero običajno uporabnik ne ve. S pomočjo virusa oziroma trojanskega konja lahko napadalec spremlja ves promet v omrežju. Tako si lahko pridobi vse osebne podatke, ki jih potrebuje za nadzor nad žrtvinimi sistemi. Vsak uporabnik internetnih storitev (bančne storitve, spletni nakupi ...) mora skrbeti za varnost svojih podatkov, sicer si lahko povzroči ogromno škode. Skrbniki spletnih storitev moramo svoje podatke hraniti še bolj skrbno kot navadni uporabniki.

Pred vdori v naš osebni računalnik se lahko učinkovito zaščitimo z uporabo požarnih zidov in protivirusnih programov. Dobro se lahko zaščitimo že s protivirusnimi programi in požarnimi zidovi, ki jih dobimo brezplačno na spletu.

Tudi sam sem doživel primer vdora v računalnik. Ker nisem računalnika pregledal dovolj pogosto, mi je trojanski konj prebral vse podatke, ki so se prenašali po omrežju. Tako je prišel do uporabniških imen in gesel, ki sem jih uporabljal za urejanje slovarja. Ko je dobil te podatke, je na spletno stran vstavil škodljivo kodo, ki je preprečevala preusmeritev iz iskalnikov, oziroma je brskalnik obvestil uporabnika, da je prišel na stran s škodljivo vsebino. Ker sem napako hitro zaznal, sem jo lahko odpravil brez večjih izgub. Posledice pa bi bile lahko precej hujše, če napake ne bi opazil ali če bi program izbrisal podatkovno bazo.

Nevarnosti vdorov se moramo zavedati v celotnem procesu izdelave in vzdrževanja spletne aplikacije. Poskrbeti moramo za varnost in za kopije obstoječih podatkov, saj se na spletu pojavljajo vedno novi načini zlorab.

4.6 Trendi

Razvoj načinov za vdiranje se razvija hitreje in je vedno korak pred razvojem sistemov za zaščito. Najbolj zanimivo področje v prihodnjih letih bodo mobilne aplikacije [3]. Vse več spletnih strani ima podstran, namenjeno mobilnim aplikacijam. To je svetovni trend, ki mu moramo razvijalci spletnih strani slediti, če želimo ostati konkurenčni. Pri tem se moramo zavedati, da

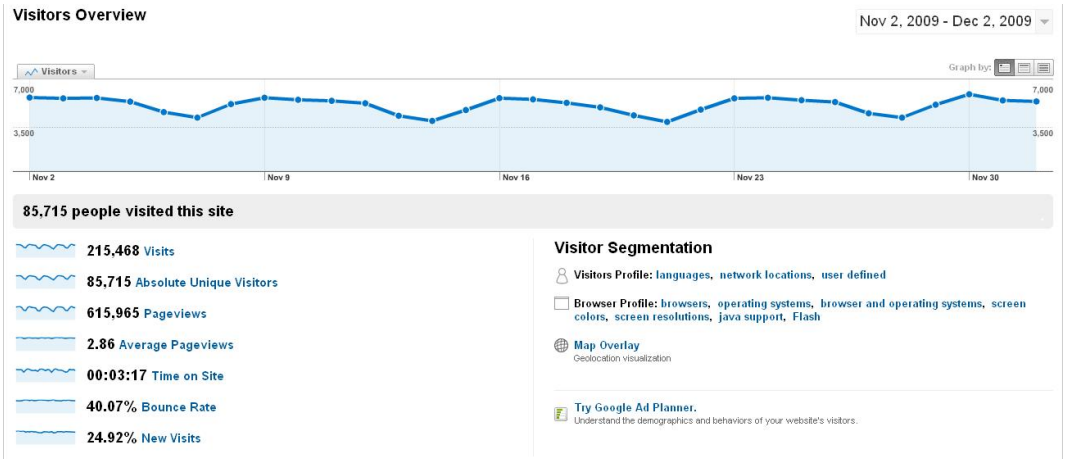
so mobilni telefoni še bolj občutljivi na vdore, saj zaščita na tem področju še ni razvita, oziroma se večina nevarnosti niti ne zaveda.

Poglavje 5

Optimizacija spletne strani (SEO)

5.1 O optimizaciji

Spletne strani, ki niso obiskane s strani uporabnikov, so nekoristne in ne služijo svojemu namenu. Za takšno spletno stran ni potrebe po zaščiti niti po optimizaciji MySQL poizvedb. Nihče si ne želi takšne spletne strani. Da privabimo obiskovalce oziroma potencialne stranke na svojo stran, moramo stran prilagoditi iskalnikom. Temu postopku rečemo optimizacija spletne strani. 80 % uporabnikov si namreč pogleda le prvo stran na iskalnikih. Torej, če nas ni na prvi strani, nas ni. Optimizacija spletne strani je bila ključnega pomena za



Slika 5.1: Mesečni obisk na Spletni-slovar.com.

spletni slovar, saj brez nje ne bi dosegli 80.000 različnih mesečnih uporabnikov [8]. Z določenimi postopki, ki jih bomo še opisali v nadaljevanju, smo spletno stran pripeljali na prvo pozicijo iskalnikov.

Ločimo med dvema različnima vrstama optimizacije glede na postopke, ki jih opravljamo:

1. Optimizacija na spletni strani (on-site).
2. Gradnja dohodnih povezav (off-site).

Optimizacijo ločimo na dovoljene postopke (white hat SEO) in prepovedane postopke (black hat SEO). S prepovedanimi postopki lahko preslepimo iskalnik. Na kratek rok bo verjetno tak pristop deloval, vendar bo sčasoma iskalnik ugotovil prevaro in bo vašo spletno stran kaznoval z zelo slabimi uvrstitvami. Ko je spletna stran kaznovana, je potrebno veliko več časa in truda, da se zopet pojavi na prvi strani.

Optimizacijo na spletni strani lahko razdelimo na različna področja. V osnovi jo lahko razdelimo na dve glavni področji:

1. Analiza ključnih besed.
2. Optimizacija spletne strani.

Prva stvar, ki sicer ni vključena pod optimizacijo spletne strani, vseeno pa spada v to področje, je ideja, na podlagi katere se razvije spletna stran. Pri razvijanju ideje spletne strani, moramo vedno misliti na uporabnika oziroma na število uporabnikov, ki jih ta ideja lahko doseže.

5.2 Analiza ključnih besed

Analiza ključnih besed je proces, s katerim določite, katere ključne besede najbolje opisujejo vašo spletno stran. Pri analizi ključnih besed moramo preveriti vse različne možnosti, da dobimo tiste besede, ki so najbolj iskane. Z različnimi možnostmi mislim predvsem na različne oblike ključnih besed (sopomenke) in uporabo ednine ali množine. Pri analizi ključnih besed za slovar, sem prišel do ugotovitve, da je ključna beseda slovar nekajkrat bolj iskana od ključne besede slovarji. Če tega ne bi preveril in bi optimiziral pod ključno besedo slovarji, bi imel dosti manjše možnosti za uspeh.

Analiza ključnih besed je pomemben dejavnik pri optimizaciji spletne strani. Če želimo na svojo stran privabiti uporabnike, ki potrebujejo slovar, moramo stran optimizirati pod tistimi ključnimi besedami, ki so za našo stran primerne in so iskane. Če nihče ne išče pod ključno besedo, ki jo optimiziramo, potem se tudi obisk na spletni strani ne bo povečal [4].

Pri analizi ključnih besed za slovar, sem izbral ključne besede, ki so najbolj iskane in najbolje predstavljajo spletno stran. Seznam ključnih besed in uvrstitev na iskalniku Google.si:

1. Slovar (1).
2. Slovensko-angleški slovar (1).
3. Angleško-slovenski slovar (1).
4. Angleški slovar (2).
5. Nemško-slovenski slovar (1).
6. Slovensko-nemški slovar (1).
7. Prevajalnik (4).

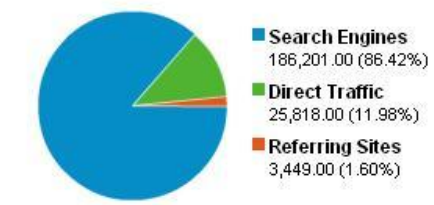
Poleg teh ključnih besed, sem stran optimiziral še pod drugimi ključnimi besedami, ki predstavljajo vseh štirinajst slovarjev na strani. Pod vsemi kombinacijami slovarjev, se stran nahaja na prvem ali drugem mestu Google.si. Podobne rezultate sem dosegel tudi s prevajalnikom.

Ključne besede, pod katerimi uporabniki največkrat obiščejo spletno stran Spletni-slovar.com, vidite na sliki.

5.3 Osnovni elementi optimizacije

Po dokončani analizi ključnih besed moramo stran prilagoditi iskalnikom, tako da so izbrane ključne besede izpostavljene. Izpostavimo jih, tako da se nahajajo v meta oznakah, vsebini strani, naslovu, povezavah in slikah. Pomemben je tudi položaj besed na strani, oziroma v kodi sami, saj so besede, ki se nahajajo višje v kodi spletne strani tudi boljše ovrednotene.

Naslov spletne strani je eden od pomembnih dejavnikov pri uvrščanju vaše



Keywords	Visits	% visits
angleško slovenski slovar	40,279	21.63%
slovar	33,954	18.24%
slovensko angleški slovar	24,418	13.11%
slovarji	6,617	3.55%
nemško slovenski slovar	6,009	3.23%

view full report

Slika 5.2: Odstotek obiskovalcev glede na iskano ključno besedo.

strani. Naslov spletne strani naj bo kratek in predstavlja naj bistvo strani. Priporočena dolžina naslova je nekje med desetimi in šestdesetimi črkami, vendar po lastnih izkušnjah lahko povem, da se krajši naslovi bolje obnesejo (npr. do 20 črk). Naslov spletne strani na slovarju je slovar, saj je to nadpomenka vsem podstranem, ki se na moji strani nahajajo. Naslov spletne strani in naslovi vseh podstrani se ne smejo ponavljati. Ponavljati se ne smejo niti meta oznake na naši strani.

Opis (description) in ključne besede (keywords) so pomembne meta oznake pri optimizaciji. Opis spletne strani mora vsebovati ključne besede, pod katerimi optimiziramo. O najboljši dolžini opisa so mnenja med različnimi strokovnjaki na tem področju precej deljena. Sam menim, da dolžina opisa ne sme presegati 160 znakov, čeprav lahko marsikje zasledimo 240 znakov. Manj pomemben dejavnik so ključne besede, ki jih nekateri iskalniki več ne upoštevajo. Nekateri iskalniki lahko vašo stran kaznujejo s slabšimi pozicijami, če se ključna beseda ne nahaja v vsebini strani. Zato naj velja, da so vse ključne besede, ki jih naštejete v meta oznakah, napisane tudi v vsebini spletne strani, torej tistem delu, ki je viden uporabniku.

Primer slabega opisa:

```
Obiščite našo spletno stran in si oglejte, kaj vse vam ponujamo
    pri nas. Čakajo vas tudi bogate nagrade, če na stran
    prispevate kakšno besedo.
```

Primer dobrega opisa:

```
Brezplačni slovarji. Angleški, nemški, hrvaški, slovenski,
    španski, francoski in italijanski slovar.
```

Tudi vsebina, ki jo predstavimo na spletni strani je pomemben dejavnik. Vsebina mora vsebovati vse ključne besede, ki jih optimiziramo. Ključne besede se morajo nahajati čim višje na strani, saj so tako bolje ovrednotene. Pogostost ključnih besed mora biti visoka, vendar ne preveč. Če se ključne besede v vsebini spletne strani pojavljajo prepogosto, lahko tako spletno stran iskalnik kaznuje, ker sklepa, da ga skušamo prevarati. Določitev primerne vsebine, ki je prilagojena iskalnikom ter razumljiva in pregledna uporabnikom, je največji izziv pri izdelavi oziroma optimizaciji spletne strani. Gostota ključnih besed nikoli ne sme preseči 10 %. Gostota ključne besede slovar na prvi strani slovarja je 8.71 %. Spletne strani, ki so v celoti narejene iz slike, flash animacije ali javascript kode so nekoristne, saj ne vsebujejo nobene vsebine, ki bi jo prikazali iskalniku. Take spletne strani so slabo obiskane, saj iskalnik ne ve, kaj spletna stran predstavlja. Te tehnologije lahko uporabimo v manjšem obsegu, na primer v delu spletne strani, ki izboljša uporabniško izkušnjo.

Povezave na spletni strani (URL strani) naj bodo opisne. Vsebujejo naj ključne besede, pod katerimi optimiziramo. Tekst v povezavah na strani, kakor tudi tekst v povezavah iz drugih strani, ki kaže na našo stran, naj vsebuje ključno besedo, ki je za nas pomembna.

Primer slabih spletnih naslovov (URL):

```
/izpis.php?slo-id=1
/izpis.php?slo-id=2
/izpis.php?slo-id=3
```

Primer dobrih spletnih naslovov (URL):

```
/slovensko-angleski-slovar
/anglesko-slovenski-slovar
/nemsko-slovenski-slovar
```

Slike na spletni strani so iskalnikom neberljive. Iskalniki iz slike, ki ne vsebuje opisa (alt oznak), ne morejo določiti vsebine. Zato moramo vedno pri prikazu

slik na spletni strani uporabiti opis (alt).

Na trgu lahko najdemo ogromno število podjetij, ki se ukvarjajo z izdelavo spletnih strani. Večina od njih nam ponuja tudi optimizacijo strani po izjemno nizkih cenah. Vendar optimizacija strani ni postopek, ki bi ga lahko končali v nekaj dneh, saj mora vsebovati natančne analize, da ne uničimo dobrih lastnosti spletne strani. V teh podjetjih nimajo zaposlenega nobenega strokovnjaka za optimizacijo in v večini primerov bolj škodijo, kot izboljšajo vašo spletno stran. Najbolj pogoste napake, ki jih naredijo podjetja ali posamezniki, ki proces optimizacije samo delno poznajo:

1. Prepogosto pojavljanje ključnih besed - temu pojavu lahko rečemo tudi besedno onesnaževanje. Take strani nimajo dodane vrednosti za uporabnika, ampak želijo s ponavljanjem ključnih besed zvišati svojo pozicijo na iskalnikih. Iskalniki vas lahko kaznujejo s slabšo pozicijo, ali pa jo v skrajnih primerih izključijo iz rezultatov iskanj.

2. Vstopna stran - vstopna stran je najpomembnejša stran. Veliko podjetij lahko po želji naročnika ali zaradi svojega načina dela vstopno stran naredi v flash animaciji, oziroma je vstopna stran vsebinsko skromna (npr. kliknite vstopi, izberite jezik, logotip podjetja ...).

3. Pretiravanje pri izbiri ključnih besed - eno spletno stran lahko uspešno optimiziramo pod skromnim številom ključnih besed. Na spletu lahko opazimo veliko strani, kjer s ključnimi besedami pretiravajo. Uspešno lahko neko stran oziroma podstran optimiziramo z največ petimi besedami. Boljše rezultate bomo dosegli z manj ključnimi besedami. Slovar je lep primer, kjer je vsaka podstran namenjena točno določeni ključni besedi.

4. Slike namesto teksta - veliko izdelovalcev spletnih strani rajši izdela sliko, kot da bi tekst na sliki oblikovali ločeno. Slike se velikokrat nahajajo v meniju, ki vsebuje povezave na podstrani. S tem izgubljamo vrednost, ki bi jo strani dodali z dobro izdelanimi povezavami.

5. Uporaba nepomembnih besed v povezavah - večina spletnih strani še vedno uporablja nepomembne besede v povezavah (npr. domov). Če za primer vzamemo slovar, opazimo, da so vse povezave iz podstrani, ki peljejo nazaj na prvo stran označene s tekstom 'Slovar'.

6. Prehitro pridobivanje povezav - iskalniki ne marajo strani, ki se želijo hitro prebiti na najboljše pozicije. Če bomo zunanje povezave pridobivali prehitro, bo iskalnik izključil vašo stran iz rezultatov iskanj.

5.4 Pridobivanje povezav

Ko zaključimo postopek optimizacije na spletni strani (on-site), začnemo z gradnjo dohodnih povezav. Iskalniki delujejo, tako da se preko povezav na druge spletne strani širijo in večajo svojo bazo (index). Več kot je povezav, ki kažejo na našo stran, boljše bo stran ovrednotena s strani iskalnikov.

Pri tem je zelo pomembna vrednost strani (Page Rank), ki jo iskalnik določi vaši strani in vsem stranem, ki linkajo na vašo stran. Višja kot je vrednost strani, ki vsebuje povezavo do vaše strani, večjo korist vam prinese. Vrednost vaše spletne strani lahko preverite na naslovu http://www.trustrank.org.

Postopek pridobivanja povezav je delo, ki ni nikoli opravljeno. Če želite ohraniti dobro pozicijo na iskalnikih, morate skrbeti za že obstoječe povezave, preverjati vrednost povezav, ki kažejo na vašo stran in pridobivati nove povezave. Če zaspimo in novih povezav nekaj časa ne pridobivamo, si lahko povzročimo veliko več težav, če nas konkurenca prehiti.

Povezave lahko pridobivamo na različne načine. Prvi in najbolj enostaven način je vpis v spletne imenike in iskalnike. S tem ne pridobimo veliko, saj so spletne strani, na katerih se nahaja povezava do naše spletne strani, večinoma nizke vrednosti. Malo bolj zamuden in ne veliko boljši način je pisanje komentarjev po forumih in blogih s sorodno tematiko. Tretji in najboljši način je izmenjava kvalitetnih povezav z drugimi spletnimi stranmi. Za izmenjavo povezav je dobro, če ste lastnik več spletnih strani, saj lahko tako izmenjate več povezav.

O kvaliteti povezav do vaše strani, ki so objavljene na drugih straneh, odloča več dejavnikov. Poleg vrednosti spletne strani je pomemben tudi položaj povezave do vaše strani. Povezave, ki se pojavljajo višje na spletni strani, imajo tudi višjo vrednost. Pomembno je tudi število vseh povezav, ki so objavljene na strani. Večje kot je število povezav, manjša je vrednost vsake povezave. Besedilo, ki se pojavlja na povezavi, mora vsebovati ključno besedo, pod katero optimiziramo. Pomemben dejavnik za vrednost zunanje povezave

je tudi vsebinska povezanost strani. Na primer, če imamo dve povezavi, ki kažejo na slovar. Prva povezava se nahaja na spletni strani, kjer prodajajo slovarje, druga pa na spletni strani, kjer prodajajo avtomobile. Prva povezava je v tem primeru, če so ostali dejavniki enaki (vrednost strani, pozicija ...), veliko boljša od druge povezave.

Povezave iz drugih spletnih strani pridobimo z namenom višjih pozicij v iskalnikih. Za izmenjavo se ne smemo odločati, ker je neka stran dobro obiskana in mislimo, da bo iz nje prišlo k nam veliko obiskovalcev. Uporabniki, ki vašo stran najdejo v iskalniku, pod tisto ključno besedo, pod katero ste stran optimizirali, so vaša potencialna stranka, saj iščejo prav tisto, kar vi ponujate. Odstotek obiskovalcev iz iskalnikov na Spletni-slovar.com lahko vidite na sliki.

Slika 5.3: Odstotek obiskovalcev glede na vir.

Poglavje 6

Zaključek

Cilj diplomske naloge je bil izdelati učinkovito spletno stran, ki bo odzivna, varna in dobro obiskana. To mi je z dobro idejo in izvedbo tudi uspelo. Ker se za vsako dobro idejo hitro najde tudi konkurenca, se je enako zgodilo tudi s slovarjem. Ko sem začel z izvedbo, je bilo na spletu samo nekaj slovarjev, ki so bili nepregledni in slabo obiskani. Razlog slabega obiska je bil v neizvedbi optimizacije in slabih uvrstitvah v iskalnikih. Ker sem imel tudi sam težave z iskanjem brezplačnega slovarja, sem hitro dobil idejo. Po dobrem obisku na strani se je ideja razširila in danes lahko brez težav najdemo preko deset slovenskih slovarjev različnih izvajalcev. Veliko teh izvajalcev ima za sabo veliko količino sredstev, ki jih lahko namenijo oglaševanju. Ker sam tega ne morem, je edina stvar, s katero lahko ostanem konkurenčen, to da je slovar boljši od ostalih. V prihodnosti nameravam uvesti še veliko izboljšav. Največja težava je v številu prevodov, ki so last posameznih avtorjev. Zato bom moral v prihodnosti svoje uporabnike še bolj motivirati za vpisovanje novih besed. To bom naredil z nagradnimi igrami, kjer bo nagrado prejel uporabnik, ki bo vpisal največ besed. Spremenil bom izgled spletne strani, kot sem že omenil v prejšnjih poglavjih. Izgled spletne strani bo bolj prijazen uporabniku. Sprotno prevajanje besed (Ajax) bo še vedno mogoče za uporabnike, ki to tehnologijo razumejo.

Uporabniku bom ponudil še več vsebine. Z novim izgledom bo slovar omogočal še igranje iger. Igre, ki jih bodo uporabniki igrali na slovarju, bodo namenjene izključno učenju tujega jezika in izobraževanju nasploh. Ker se vsak uporabnik, ko pridobi informacijo, ki jo išče, odpravi na nek iskalnik, s katerim išče drugo informacijo, bom na slovarju naredil tudi iskalnik. Iskalnik bo temeljil na že uveljavljenih algoritmih (Google, Bing, Yahoo).

Z uvedbo novega izgleda in dodatnimi možnosti, ki jih bo slovar vseboval, je cilj spletne strani v letu 2010, preseči 100.000 različnih mesečnih uporabnikov.

Slike

Tabele

Literatura

[1] D. Russell, "MySQL in a Nutshell", (Sebastopol, O'Reilly Media, 2008).

[2] B. Schwartz in drugi, "High Performance MySQL", (Sebastopol, O'Reilly Media, 2008).

[3] M. Cross, "Developer's Guide to Web Application Security", (Rockland, Syngress Publishing, 2007).

[4] J. Ledford, "Search Engine Optimization Bible", (Indianapolis, Wiley Publishing, 2008).

[5] G. Kozak, "Varnost in PHP", (PHP konferenca 2009).

[6] T. Uranjek, "Tehnike vdorov II", (Hirtgen d.o.o.).

[7] MySQL, [URL: http://www.mysql.com/].

[8] Google Analytics, [URL: http://www.google.com/analytics/].

[9] Blog o optimizaciji MySQL-a, [URL: http://www.mysqlperformanceblog.com/].

[10] Wikipedia, [URL: http://en.wikipedia.org/].