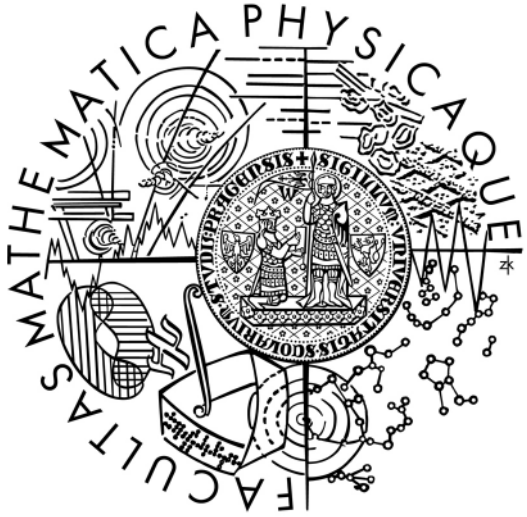
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta



BAKALÁŘSKÁ PRÁCE

Martin Baroš

Modulární fulltextový vyhledávač pro MySQL

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce:
Mgr. Jiří Semecký

Studijní program: Informatika, Programování
2007

Na tomto místě bych rád poděkoval vedoucímu bakalářské práce, Mgr. Jiřímu Semeckému za podporu, pomoc a věcné připomínky. Také bych rád poděkoval své rodině a přátelům za neocenitelnou podporu při psaní této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 31.5.2006 Martin Baroš

Obsah

Název práce: Modulární fulltextový vyhledávač pro MySQL

Autor: Martin Baroš

Katedra (ústav): Institute of Formal and Applied Linguistics

Vedoucí bakalářské práce: Mgr. Jiří Semecký

E-mail vedoucího: Jiri.Semecky@mff.cuni.cz

Abstrakt: Cílem bakalářského projektu je za pomoci databázového serveru MySQL vystavět fulltextový vyhledávač, který je schopen pracovat se specifickými vlastnostmi češtiny. Není snahou naprogramovat moduly pro zpracování dokumentů na vysoké úrovni, které budou řešit lingvistické otázky, ale vytvořit dobře definované rozhraní, v rámci kterého lze zasouvat další (jiné) moduly, jejichž rozhraní odpovídá definici. Cílovou platformou je Unix, programovacím jazykem C++.

Klíčová slova: fulltext, mysql, modulární vyhledávač

Title: Modular fulltext search for MySQL

Author: Martin Baroš

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Jiří Semecký

Supervisor's e-mail address: Jiri.Semecky@mff.cuni.cz

Abstract: An objective of the project is to develop a modular fulltext search engine using MySQL database server. The search engine should operate with the Czech language's specific attributes. There is no endeavor to develop high quality modules solving linguistic problems. Project should provide interface and ability to plug-in (plug-out) next modules. Project's software platform is Unix, programming language C++.

Keywords: fulltext, mysql, modular search

1. Popis prostředí/systému

1.1. Úvod

S rostoucím množstvím dokumentů v elektronické podobě bylo nutné začít používat flexibilnější techniku jejich správy. Přestaly dostačovat dosud používané postupy, jakými bylo například katalogové třídění – každý dokument byl zařazen do tematicky rozdělené několikaúrovňové hierarchie. Bylo nutné zodpovědět otázku, jak přesněji popisovat obsah dokumentu a automaticky rozpoznávat jeho zařazení v kontextu ostatních. Odpovědí se ukázalo být nasazení fulltextového vyhledávání.

Tato technika prokázala svou použitelnost i v tak dynamicky se měnícím prostředí, jakým jsou internetové stránky. Právě webový prostor je totiž ukázkou velmi heterogenního prostředí, ve kterém selhávají jiné specializované techniky správy dat, ačkoliv v užším kontextu jsou úspěšné. Příkladem může být kromě zmíněného katalogu tzv. tagování. Jedná se o techniku, kdy může být daný dokument ve více kategoriích současně. Ukázkou vhodného použití může být žánr u filmu.

Jedním z nejčastějších způsobů uchovávání většího a velkého množství dat jsou bezesporu databáze. Dnes jsou databázové stroje běžně používány k uchovávání celých dokumentů na rozdíl od dřívějšího přístupu, kdy se v databázi uchovávala jen informace o umístění a několik krátkých záznamů charakterizujících dokument. S rostoucím množstvím dat přichází opět problém s jejich vhodným uchováváním tak, aby je bylo možné následně efektivně využívat. Nezbytnou součástí takové správy dat je rychlé a dostatečně přesné vyhledávání dokumentů, ideálně s ohledem na celý obsah, pro což je s výhodou používán právě fulltext.

1.2. Cíl práce

Cílem bakalářské práce je navrhnout a implementovat rozhraní pro vkládání modulů umožňující přizpůsobení fulltextového vyhledávání. Tyto moduly v důsledku umožní získání kvalitnějších odpovědí na fulltextové dotazy. Možnost přizpůsobení je významná zvláště v souvislosti s jazyky jako např. čeština zejména kvůli komplikované morfologii.

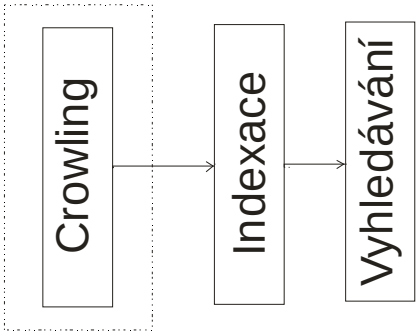
Významným kritériem je modularita celého řešení podpořená co nejsnazší implementací a přidáním dalších modulů. Práce by měla ukázat, jakým způsobem je možné další moduly vyvíjet a rozšiřovat (příp. měnit) jimi funkcionalitu vyhledávače. Není záměrem programovat zásuvné moduly řešící lingvistické otázky, ale je nutné počítat s jejich nároky při budoucí implementaci.

Implementace má být realizována pro databázový stroj MySQL, který je nejpoužívanější open source variantou databázové aplikace.

Primární cílovou platformou softwarového díla je UNIX, nicméně zdrojové kódy jsou psány a z velké části testovány i na Microsoft Windows.

1.3. Prostředí fulltextu

Funkčnost fulltextového vyhledávání je dnes možné získat dvěma základními cestami. Oběma cestám je společné, že práce s daty probíhá ve dvou resp. ve třech fázích. První fází je načtení dokumentů (indexace), druhou je samotné vyhledávání v zaindexovaných dokumentech. Indexaci v některých případech předchází získání dat – crowling.



1.3.1. Vyhledávání

První metodou fulltextového vyhledávání je využití samostatné aplikace, která zpravidla obsahuje modul pro indexování a vyhledávání. Tato varianta se využívá zpravidla pro případy, kdy se objem dat pohybuje v řádu desítek a stovek GB popř. terabytů. Pro takové objemy dat je nutné k zachování dostatečné rychlosti používat úzce specializované postupy, které prakticky vylučují připojení jiných funkčností. Samostatné fulltextové aplikace jsou využívány např. k indexování www stránek na internetu.

Výhodou tohoto řešení je vyšší rychlost a tím pádem možnost zpracovávat větší objem dat. V případě potřeby je možné vyhradit samostatný server pro zajištění této funkčnosti a tímto oddělením od ostatních služeb získat vysoký výkon při zpracovávání dat. Nevýhodou je často nutnost odborné instalace spolu se správou další samostatné aplikace.

Druhá metoda se týká uživatelů, kteří potřebují pracovat s menším a středním objemem dat – řádově jednotky gigabytů a méně. Vzhledem k tomu, že jsou informace zpravidla uloženy v databázi, je tato vrstva ideálním místem pro implementaci fulltextu.

Výhodou řešení je bezesporu jeho dostupnost – základní funkčnost je možné používat s minimálním úsilím, často i bez potřeby instalace dalších modulů. Indexování vkládaných dat probíhá automaticky po spuštění této funkčnosti (pokud již není aktivována ve výchozí konfiguraci). Dotazy na vyhledávání a jeho výsledky jsou zprostředkovány prostřednictvím ODBC[1], které je rošířeno o potřebná klíčová slova. Zpracování výsledků je tedy jednodušší – není nutné napojení na externí aplikaci přes nový interface. Stinnou stránkou je nižší rychlost práce s daty – reálný rozdíl je ale vzhledem ke zpracovávanému objemu dat často zanedbatelný. Větším problémem je omezení možného zpracovávaného objemu dat spolu s potenciálními

[1] ODBC (Open Database Connectivity) – definovaný aplikační interface (API) pro přístup k databázi

problémy v případě nárůstu zpracovávaných dat a potřebou přechodu na výkonnější variantu vyhledávání.

1.3.2. Indexace

Způsoby získávání dat k indexaci jsou různé dle typu aplikace. Je možné využít specializovanou aplikaci, která může opět běžet na samostatném počítači. V prostředí Internetu se ke stahování webových stránek využívá specializovaný robot (často nazývaný spider)[2]. V případě aplikace, jakou je databáze, jsou zpracovávaná data držena přímo aplikací a není tedy robot potřeba. Data k indexaci nemusí být jen textové materiály, ale také obrázky, videa, zvukové soubory apd. Pokud data nejsou v textové podobě lze postupovat tak, že je pro ně vytvořen text, který je popisuje. V této práci je zúžena otázka práce na zpracování materiálů v textové podobě.

Data, která jsou určena k indexaci jsou zpracována a je z nich vytvořen tzv. index. Vytvoření této specializované struktury vychází z požadavku, kterými je efektivní vyhledávání, jehož výstupem je odkaz na dokument, případně výstřižek[3] V indexu není udržován celý dokument, nicméně pokud je to požadováno, je možné zaindexovaný dokument udržovat v datovém skladu, popř. udržovat jiné související informace (např. pro knihy umístění v knihovně apd.).

Každý dokument v indexu má přidělen svůj identifikátor, označme jej DID. Dokument je parserem přeložen na tokeny[4]. Každé DID je svázáno se seznam termů, které dokument obsahuje. To umožní odpovídat na pokládané dotazy.

Token \ DID	DID_1	DID_2	DID_3
nejlevnější	1	0	1
autopůjčovna	0	1	0
praha	0	1	1

Tabulka 1 - Informace o termech v indexu

V předchozí Tabulce 1 je naznačeno uložení informací o tokenech. Pro bitové vektory pro každý term je následně možné snadno pomocí bitových operací provádět logické operace AND, OR, příp. NOT. V praxi se ale vzhledem k množství zpracovávaných dat využívá následujících optimalizací.

Dokument zpravidla obsahuje jen malou část slov ze všech možných, a proto je jeho bitmapa řídká. To vede k použití tzv. invertovaného indexu, jehož princip je popsán níže.

Ke každému termu se vytvoří seznam dokumentů, ve kterých se vyskytuje (seznam je tvořen DID a je seřazen vzestupně kvůli možnosti slévání). V případě řešení logické operace, je možné tuto operaci snadno vyřešit s využitím informace, že seznamy DID jsou vzestupně setříděné.

---

[2] Tento proces získávání dat se nazývá crowling.
[3] Krátký úsek textu, který obsahují hledaná klíčová slova. Zobrazuje se pro přiblížení kontextu, ve kterém jsou slova použita.
[4] Token si lze představit jako jedno slovo, případně slovní spojení

Implementačně se pro efektivnější práci s dokumenty uchovává v odděleném souboru seznam termů (slovník) a proud DID (záznamy). Slovník obsahuje termy a odkazy do záznamů spolu s informací o konci řady DID pro tento term (počet DID nebo offset do záznamů, kde řada DID končí). Soubor 2 obsahuje pouze proud DID. Záznamy mohou obsahovat položky standardizované délky (DID), což umožňuje snadnou manipulaci s daty.

Z technických důvodů je problematická aktualizace ve smyslu smazání nebo přidání DID do souboru záznamů. V takovém případě je totiž nutné aktualizovat jak záznamy, tak slovník. Proto je třeba provádět tyto úpravy s co nejmenší frekvencí, k čemuž se používají dodatečné malé indexy obsahující přidané či odstraněné DID. Výsledky z velkého indexu jsou zaktualizovány podle malého indexu a následně prezentovány. U velkých indexovacích strojů se udržují indexy paralelně. Dotaz následně probíhá paralelním dotazem do dostupných indexů a spojením vrácených výsledků. Díky tomu je možné rozložit zátěž a umožnit efektivní paralelní zpracování dotazů.

U velkého množství dat se provádí tvorba indexů paralelním zpracováním postupně. Nejdříve je vytvořen index pro každý dokument, označme jej *L0-index* (index úrovně 0). *L0-index* se následně slévá dle zvoleného faktoru *k*. Spojením *k* *L0-indexů* dostáváme jeden *L1-index*. Takto se spojuje vždy *k* indexů se stejnou úrovní a vznikají indexy vyšších úrovní, které se teprve napojují do centrálního indexu, nebo tvoří další paralelní. Spojení dvou indexů stejné úrovně lze provést v lineárním čase.

1.3.3. Získání dat – crowling

V případě, že jsou data pro fulltextového vyhledávání umístěná na jasně definovaném místě není nutné se starat o jejich získání. V některých nasazeních je ale nutné data pro indexování nejdříve získat. Příkladem může být internetový vyhledávač, který indexuje obsah www stránek.

Pro stažení obsahu webových stránek se obvykle využívá samostatná aplikace, která prochází webový prostor a stahuje obsah stránek. Samotné stahování naráží na několik technických a etických problémů.

Stahování webových stránek komplikuje z technické stránky odezva DNS[5], dynamicky se rozšiřující portfolio odkazů stránek spolu s jejich normalizováním a rozpoznáváním unikátnosti. Etické otázky se týkají chování robota, které by nemělo omezovat chod serverů, které poskytují webové služby.

Pro stažení nové stránky je - v případě, že není známa - nutné získat IP adresu serveru, kde je umístěna. K získání této adresy se využívá služba DNS[6]. Aktivní čekání na odpověď a následné zpracování výsledku by vedlo k výraznému zpomalení. Proto jsou dotazy řazeny do fronty a robot nečeká a zpracovává již vyřešené. Je nutné dát pozor na skutečnost, že DNS záznamy mají omezenou dobu

[5] Odezva se pohybuje v řádech milisekund, ale při průchodnosti crowlovacího stroje 1000 stránek za sekundu, je i toto relativně malé zpoždění problémem.

[6] Lze využít specializované aplikace (např. wget), které stažení stránek zajistí. V textu se předpokládá, že se pro zvýšení průchodnosti využije vlastní řešení, kdy sama aplikace odesílá a zpracovává http request.

platnosti pomocí TTL[7]. Situace je drobně zkomplikována tím, že TTL lze nastavit individuálně pro doménu. Řešením by mohla být fronta vyřešených dotazů, uchování struktury obsahující IP adresu, URL adresu a časovou značku[8], kdy vyprší (příp. host[9]). Před zpracováním záznamu se pouze zkontroluje, zda je časová značka větší než aktuální čas. Pokud je překlad neplatný, vloží se host znovu do fronty ke zpracování.

V drtivé většině případů nestačí pouze načíst jednu stránku z prozkoumávaného host. Stažený text obsahuje zpravidla řadu dalších URL linků na stránky na stejné či další doméně. Parsování dokumentu, které mimo jiné vyhledává obsažené odkazy, lze provádět na úrovni crowleru nebo až při indexování.

První varianta umožňuje snížit zátěž indexátoru a optimalizovat množství uložených dat. Neukládají se celé dokumenty s tagy[10], ale jen text zpravidla doplněný o několik málo speciálních znaků uchovávajících důležité sémantické informace vycházející z HTML[11]. Nevýhodou je využití většího množství paměti pro běh. Zparsované dokumenty se uloží do datového úložitě, odkud jsou dále zpracovávány indexátorem. V případě velkého množství dat se úložiště stejně jako následně indexy distribuují.

Indexátor připravující URL musí být dostatečně rychlý, aby se nestal slabým místem – crowler by zpracovával data rychleji, než by se připravovala nová. Výhody a nevýhody vyplývají z předchozího odstavce.

Dostáváme se k problémům souvisejícím se samotným zpracováním, URL adres. Adresu lze rozdělit na část za znakem otazník a před ní. Část za otazníkem obsahuje parametry pro skript stránky - z toho vyplývá, že jiné parametry mohou vést na jinou stránku a je tedy nutné je rozlišovat. Pro optimální chod crawleru je důležité rozpoznat duplicitní URL, a to jak ve frontě ke zpracování, tak mezi zpracovanými stránkami. Pro to, aby bylo možné znakové porovnání, se provádí úprava argumentů za otázníkem, která spočívá v:

1. přepis vhodných znaků na jejich escape sekvence (př. ^ na %5E)
2. seřazení argumentů dle abecedy (pořadí argumentů neovlivňuje chování korektně napsaného skriptu)

Druhým možným přístupem je odstranění celé části za otazníkem[12] a dvě takto shodné URL považovat za duplicitní.

I přes tyto úpravy je následně nutné provádět rozpoznávání duplicit obsahu, protože je běžným jevem, že více URL adres odkazuje na shodný webový dokument. Ale tato heuristika již nespadá do diskuze o URL.

Zajímavou otázkou je, jak často by měl robot navštěvovat webové stránky kvůli jejich aktualizaci. HTTP nabízí dvě informace související s touto otázkou:

---

[7] TTL – time to live
[8] Časovou značkou se rozumí uchovávání informace ve tvaru DD-MM-YYYY HH:MM:SS
[9] Doménové jméno serveru, které je obsaženo v URL
[10] Speciální značky, které nejsou brány jako text v HTML jsou uvozeny „<“ a uzavřeny „>“
[11] Text v tagách typu hX, b apod.
[12] Tento přístup spolu se snahou umístění vhodných klíčových slov do URL vede programátory webových stránek k využívání techniky URL rewritingu pro dynamicky generované stránky.

- *If-Modified-Since* – při dotazu na stránku je možné specifikovat datum a čas poslední verze dokumentu, která byla stažena. V případě, že od specifikovaného data nedošlo ke změně, je namísto stránky vrácen kód 304 (not modified), samotná stránka se neposílá.
- *Expire* – nastavení data, kdy má být obsah stránky v budoucnu aktualizován – eventuelně lze využít k označení, že po tomto datu již informace na této stránce nejsou aktuální.

1.3.4. Parsování a tvorba tokenů

Dokument, který se má zaindexovat, je na úrovni indexace příp. crowlování parsován za účelem získání textového obsahu. Textový obsah je převeden na tzv. tokeny, které se dále zpracují a výsledek se zanese do indexu. V této kapitole se budu zabývat tvorbou tokenů a jejich vhodným výběrem.

V nejjednodušším případě by mohlo být každé slovo bráno jako token, který se zanese do indexu. Tento přístup není z celé řady hledisek nejlepším řešením. Dokument ke zpracování zpravidla neobsahuje jen text, ale celou řadu speciálních znaků souvisejících s jeho formátováním a sémantikou (speciální znaky v PDF souborech , tagy v (X)HTML, XML). Pro rozpoznání textu je nutné určit formát souboru spolu se znakovou sadou či dokonce jazykem dokumentu.

V ideálním případě by měl být parser schopen rozpoznat nejen samotná slova, ale zároveň ovládat pokročilé techniky typu odhalování překlepů, poradit si s chybějící diakritikou, rozpoznat v textu cizojazyčnou pasáž či slovní spojení atd. Podívejme se nyní postupně na jednotlivé úkoly a na související problematiku.

Už samotné rozdělení textu na jednotlivá slova je netriviální. S rostoucí různorodostí textů nelze často jednoduše rozhodnout zda je nealfanumerický znak oddělovačem. Tečka může označovat konec věty, být součástí zkratky či oddělovat desetinné místo. Problematické jsou také různě zápisy téhož ovlivněné národními zvyklostmi. Příkladem může být datum – tři různé zápisy 1.2.2005, 2/1/2005 a 2005-2-1 označují stejný den.

Odhalování překlepů lze provádět relativně efektivně v prostředí, které není příliš různorodé, tzn. má dobře definovaný slovník, který se minimálně mění, např. databáze lékařkých záznamů. Zde lze překlep lokalizovat díky předpokladu, že se neočekává masivní výskyt zcela nových slov či slovních spojení. V heterogenním prostředí, které je dynamické, lze obtížně určit, zda není nalezené slovo novotvar. Podobný problém je v prostředí vícejazyčném, kdy může docházet k překrývaní slov z jednoho jazyka s překlepy z druhého, obzvláště pokud nelze s vysokou pravděpodobností specifikovat jazyk, ve kterém je dokument napsán. V důsledku těchto faktorů se v heterogenním dynamickém prostředí obvykle neprovádí korekce, ale při vyhledávání je analyzována query dotazu a při podezření na překlep je nabídnuta domnělá správná varianta.

Úspěšné zpracování textu s (částečně) chybějící diakritikou podléhá velmi obdobným omezujícím podmínkám jako odhalování překlepů. Mohlo by se v prvním přiblížení zdát, že by řešením problému bylo indexovat text bez diakritiky, ale takto upravená slova vedou i v rámci jednoho jazyka k celé řadě nejednoznačností. Příkladem může být slovo „vlada“, kterému v podobě s diakritikou odpovídá „Vláďa“ i „vláda“.

Rozpoznání pasáže textu v jiném jazyce je také obtížný úkol. Pokud je text krátký (jedna věta) může být část textu namapována na slovo v jazyce dokumentu[13], či vyhodnocena jako překlep[14]. Kvůli tomu bude pravděpodobnost výskytu cizojazyčné pasáže vyhodnocena jako nízká a slova věty indexována v jazyce dokumentu, protože budou považovány za novotvary. Delší text, který je ideálně sémanticky oddělen formátovacími prostředky dokumentu, zvyšuje šanci na korektní vyhodnocení.

V textu by bylo výhodné identifikovat slovní spojení a indexovat je jako jeden token, protože indexována jednotlivě částečně mění či úplně ztrácí svůj původní význam[15]. Jednou z metod identifikace tzv. frází je statistické zohlednění počtu výskytů zkoumaných slov vedle sebe. Výhodou této metody je odhalení nových frází. Cenou je potřebný výpočetní výkon na prohledávání textů[16] a zpracování informací o výskytu slov. Pro malé množství úzce zaměřených textů je možné vytvořit a použít seznam frází, ale s rostoucím množstvím dat se tento postup jeví jako neudržitelný.

V některých nasazeních se pro optimalizaci výsledků nasazují techniky odstraňování nevýznamových slov (stopwords), zpracování synonym (thesauri), ohýbání slov (stemming) atd.

Seznam nevýznamových slov (stoplist) lze určit na základě jejich statistického výskytu v textu. Seznam závisí na jazyce textu a je otázkou, kde je hranice, odkud už jsou slova považována za významová. Zmíněné aspekty tvoří významné milníky omezující nasazení této techniky.

Stejně jako stopwords jsou i synonyma závislá na jazykové mutaci. Není problémem pro daná slova sestavit slovník synonym, ale je nutné experimentálně ověřit, zda jeho nasazení nepřináší zkreslení výsledků vyhledávání, protože využití synonym při vyhledávání v heterogenním textu může vést k významnému posunu sémantického významu a v souvislosti s tím ke snížení počtu hitů[17].

Důležitou a často využívanou technikou je stemming. V základní variantě se jedná o transformaci slova na jeho základní tvar. Tím se dosáhne výrazného zmenšení indexu (zmenší se slovník) a zajištění, že bude nalezeno slovo v textu i v jiném než původním tvaru. V tomto případě je nutné, aby stejným procesem transformace prošla i slova zadaná k vyhledání. Druhou variantou je indexace všech slova a provádění inverzního postupu na slova zadaná k vyhledání, tzn. nahrazení jednoho tvaru slove všemi jeho variantami. Je zřejmé, že je pro efektivní normalizování slova ve smyslu tohoto odstavce nutné znát jazyk textu, což je jedna z komplikací, kterou je třeba vyřešit k úspěsnému nasazení. Tato technika je vyhledávači i přes veškerá úskalí úspěšně implementována, protože významně zvyšuje přesnost vrácených výsledků (precision).

---

[13] Pokud je obsažen mezinárodně používaný výraz (robot), příp. shodný výraz ve dvou jazycích („a“ – v češtině spojka, v angličtině neučitý člen)

[14] V českém dokumentu anglický neučitý člen „an“ může být považován za překlep spojky „na“

[15] Příklady slovních spojení měnících rozpojením svůj význam: „Ústí nad Labem“, „teorie strun“

[16] Prakticky při určování fráze neprobíhá prohledávání textu, ale informací v indexu, kde se často vedle informací o DID udržuje kolikáté se dané slovo v jakém dokumentu vyskytlo

[17] Hit – výsledek fulltextového vyhledávání, který odpovídá představě uživatele na odpověd pro zadaná klíčová slova

Faktorem, který se zohledňuje při tvorbě indexu, je kromě samotného výskytu tokenu jeho důležitost v dokumentu. Kritérii jsou frekvence výskytů, označení syntaktickými prostředky (dle formátu dokumentu) některého slova či slovního spojení jako významné atp.

Kromě samotných slov je v některých prostředích (jako www) počítáno hodnocení pro celý dokument. Nejznámějším algoritmem, který se využívá jako kritérium pro hodnocení kvality stránky je Pagerank, který určuje „popularitu" stránky mezi ostatními. Pagerank stránky se počítá z PageRanků stránek, které na ni odkazují. Vzorec pro výpočet PageRanku stranky A je

$$PR(A) = (1-d)/m + d * (PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n))$$

kde
PR je PageRank
d je dampening faktor (nastavený pravděpodobně na 0,85)
m je celkový počet zaindexovaných stránek
C(T) je počet odkazů vedoucích ze stránky T

1.4. Databáze MySQL a fulltext

Vzhledem k charakteru licence (GPL), pod kterou je šířena databáze MySQL, a s ohledem na skutečnost, že se jedná o nejpoužívanější stroj tohoto druhu, je nutné zajistit rošiřitelnost funkčností pro vývojáře. MySQL poskytuje možnost vytvoření klienta pro napojení k datům a jejich správě, a také možnost přizpůsobení samotného databázového serveru.

Pro vytvoření klienta je možné využít následujících variant:

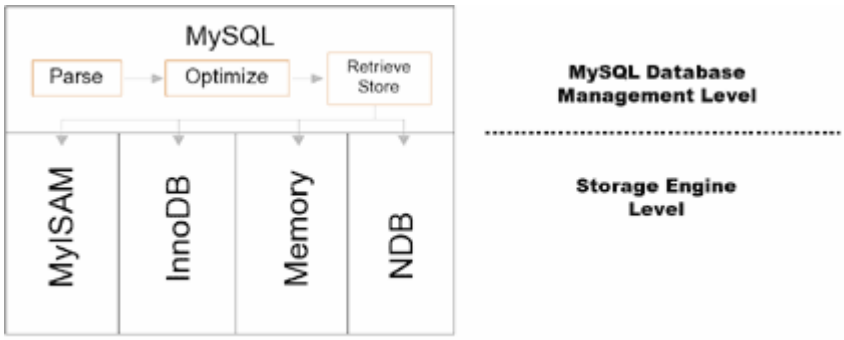
- Pro platformu Windows rodiny NT (NT, 2000, XP, 2003 nebo Vista) je možné využívat *named pipes*[18]. Od verze 4.1 je možné připojení využitím sdílené paměti[19].
- Využití Connector/ODBC (MyODBC) rozhraní, které zpřístupňuje použití ODBC (Open Database Connectivity). Díky tomu je možné např. napojení aplikací typu MS Access na MySQL server. Connector podporuje všechny funkce ODBC 2.5. Zmíněný přístup je možné použít na platformě Windows i UNIX.
- Využití jednoho z následujících protokolů, který je zvolen s ohledem na požadavky cílové platformy.
- Pro platformě nezávislý komunikační kanál lze volit TCP/IP.
- Implementace spojení prostřednictvím MySQL Connector/NET, poskytující rozhraní orientované na použití v aplikacích vyvíjených v jazycích .NET.
- Na UNIX-like systémech lze využít *socket files*.
- Volba programovacího jazyka modulu buďto přímo C/C++[20], nebo libovolný jazyk poskytující C bindings s připraveným API[21].

[18] Pro tuto variantu je nutné spustit server s parametrem *--enable-named-pipe*
[19] Tento přístup očekává použití parametru *--protocol=memory* na straně klienta a *--enable-named-pipe na straně serveru.*
[20] C, popř. C++ je programovacím jazykem, který používá MySql
[21] API je definováno pro programovací jazyky Eiffel, Java, Perl, PHP, Python, Ruby, Tcl a další.

Kromě tvorby klientů mohou být implementována rozšíření přímo na straně serveru. MySQL od sebe odděluje vrstvu práce s daty (Management level) a jejich způsob uložení (Storage engine). Úrovně jsou propojeny dobře definovaným rozhraním, což umožňuje tvorbu vlastního enginu, který je možné použít v řadě současných a velmi pravděpodobně i budoucích verzí.

Storage engine zajišťuje správu dat. Získává data k zaindexování z horní vrstvy rozhraním a následně realizuje dotazy nad těmito daty a vrací odpovědi. Naproti tomu Management level poskytuje pro uživatele rozhraní k práci s databází včetně získání a interpretace dotazů a následné s tím spojené předání výsledků.

Zmíněná architektura je mezi rozšířenými databázemi prakticky unikátem. V běžném přístupu jsou obě vrstvy spojeny a pro ukládání dat není možné storage engine změnit, případně připojit produkt třetí strany. Díky koncepci, kterou využívá MySQL, je možné podpořit optimalizovaný výkon databáze výběrem vhodného storage enginu s ohledem na charakter aplikace, která data používá. Z charakteru aplikace je v řadě případů známo, zda je vhodné klást větší důraz na rychlost vkládání, odstraňování či aktualizace dat, a jakou prioritu má v aplikaci rychlost vyhledávání. Dle frekvence využití následujících operací je poté možné volit enginy, které implementují nejvhodnější algoritmy pro prioritní operace.



Obr. 1 Architektura MySQL – oddělení vrstvy storage enginu

Dokonce není nutné používání jednoho storage enginu v rámci celé databáze – použitý engine lze definovat pro každou tabulku zvlášť. Engine se volitelně definuje přímo při vytváření tabulky použitím klíčového slova „ENGINE“. Následující příklad je ukázkou syntaxe pro tvorbu tabulky s použitím MEMORY storage enginu.

```
CREATE TABLE name_list
(ID int PRIMARY KEY, Name VARCHAR(50) )
ENGINE = MEMORY;
```

Ve verzi 5.1.12 je k dispozici celkem 5 storage enginů lišících se přístupem k ukládání dat v závislosti na požadavcích aplikace. Pro maximální rychlost práce s dotazy je možné využít MEMORY engine uchovávající data alokovaná v RAM paměti. Dalším zajímavým přístupem je FEDERATED storage engine, který umožňuje ukládání dat skrze transparentní rozhraní odděleně (např. na jiném MySQL

serveru). Souhrnné informace o těchto a dalších dostupných enginech[22] jsou k dispozici v [1].

V rámci vrstvy Management level je připraveno několik možností rozšiřování funkčnosti. Jedním z modelů je UDF (User Defined Function), který umožňuje rozšířit databázový systém napsáním vlastní rutiny. Kromě toho UDF umožňuje (od verze 5.0.3) vytvářet uživatelsky definované funkce pro manipulaci s daty v rámci SQL dotazu.

Další možností, která je k dispozici od verze 5.1 jsou pluginy. Pluginem se rozumí knihovna obsahující odpovídající API rozhraní. Plugin má řadu výhod oproti UDF. Jednou z nich je možnost instalace pluginu za běhu serveru bez nutnosti restartu. Instalace a používání pluginu je díky zvolenému přístupu intuitivní. Plugin se instaluje pomocí SQL příkazu INSTALL PLUGIN. K instalaci je nutné nahrát sdílenou knihovnu, která obsahuje přeložený kód pluginu, do adresáře pluginů.

Komplikace mohou nastat při neuváženém odstranění pluginu použitím SQL příkazu UNINSTALL PLUGIN v případě, že tabulka v databázi plugin používá. V důsledku odstranění dojde k nemožnosti práce dat v tabulce. Tabulku je poté možné odstranit pouze příkazem DROP TABLE – ostatní SQL příkazy nelze provádět.

2. Module manager

MySQL implementuje funkčnosti potřebné k využívání fulltextového vyhledávání v základní formě. Některé části fulltextu je nutné upravit s ohledem na mnoho faktorů, kterými jsou zejména jazyk vkládaného textu a zaměření dokumentů. Pro tuto přispůsobení je často nutné provádět složitější algoritmické postupy, zejména pokud se jedná o jazyk s komplikovanou morfologií, jakým je např. čeština.

Na základě potřeby mít možnost snadno a standardizovaně přidávat, resp. odebírat postupy pro zpracování textu před samotným uložením do storage enginu, příp. upravovat dotaz, jsem v rámci bakalářské práce navrhl a implementoval plugin poskytující potřebné rozhraní a snadnou rozšiřitelnost.

Plugin využívá pro komunikaci s MySQL standardizované API rozhraní, což by mělo zajistit jeho kompatibilitu i s budoucími verzemi programu. S tím jsou spojené výhody zmíněné výše, které se týkají snadné instalace a používání. Mimo to je plugin jako součást vrstvy Management level nezávislý na použitém storage enginu.

Pluginu zastává v procesu indexace funkci parseru. To znamená, že zajišťuje převedení dokumentu z jeho prvotní podoby na tokeny. Díky modularitě je tedy možné vytvořit fulltext parser zpracovávající prostý text, HTML či PDF, a k této funkčnosti přidat specifické vlastnost související s jazykem dokumentů, příp. jejich zaměřením.

Plugin se skládá z jednotlivých modulů, které lze volitelně řadit za sebe s tím omezením, že první musí stát vždy modul vytvářející z dokumentu tokeny. Ten uloží

[22] Přímý odkaz na část přehledově pojednávající o integrovaných enginech je dostupný na http://dev.mysql.com/tech-resources/articles/storage-engine/part_2.html

tokeny do sdílené struktury `tagStore`, která je následně předávána po řadě ke zpracování zahrnutým modulům.
Struktura `tagStore` je optimalizována pro rychlé vyhledávání s ohledem na snadnou rozšiřitelnost. Struktura přebírá logiku uspořádání částečně z XML, ale data jsou uloženy v datových strukturách volených pro optimální rychlost vyhledávání. Každý token – struktura `Word` může obsahovat další informace, které slouží k předávání dat mezi jednotlivými moduly.

2.1. Struktura MYSQL_FTPARSER_PARAM

Module manager (mmanager) pracuje s následující strukturou, která obsahuje query ke zpracování (parsování) a další rošiřující informace. První modul typicky zpracuje strukturu `MYSQL_FTPARSER_PARAM` a naplní interní `tagStore`, Následující moduly mají do struktury `MYSQL_FTPARSER_PARAM` přístup přes parametr funkce, ale ke své práci až na vyjímky využívají výhradně `tagStore`. Díky tomu je možné snadno předávat rozšiřující informace do dalších modulů či si informaci ponechat pro svůj modul ke zpracování při deinicializaci.

```
typedef struct st_mysql_ftparser_param
{
  int (*mysql_parse)(struct st_mysql_ftparser_param *,
                     char *doc, int doc_len);
  int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                        char *word, int word_len,
                        MYSQL_FTPARSER_BOOLEAN_INFO
*boolean_info);
  void *ftparser_state;
  void *mysql_ftparam;
  struct charset_info_st *cs;
  char *doc;
  int length;
  int flags;
  enum enum_ftparser_mode mode;
} MYSQL_FTPARSER_PARAM;
```

`int` `(*mysql_parse)()` **-** ukazatel na funkci, která slouží jako callback funkce, která zavolá interní parsovací funkci. Tato funkcionalita se ve fulltext pluginu používá v případě, že plugin slouží pouze jako frontend parser tzn. převádí textový formát na plain text. Využití je zamýšleno při přímém zpracování HTML dokumentů atp.

`int` `(*mysql_add_word)()` - ukazatel na funkci, která odesílá token k zaindexování. Tato funkce propojuje parser a server – tvoří komunikační kanál umožňující předání výsledků. Její volání je očekáváno ve funkci provádějící samotné parsování. O její zavolání se v mmanageru nestará žádný modul. Této funkci jsou po zavolání všech parse funkcí modulů předány tokeny obsažené ve struktuře `tagStore` konkrétně v tagu `current`. Další úpravy struktury `tagStore` už nemají vliv na data předaná serveru.

`void` `*ftparser_state` - obecný ukazatel, který server nevyužívá a je možné jej použít pro vlastní účely pluginu. mmanager tento ukazatel v současné verzi nevyužívá.

void *mysql_ftparam - do verze MySQL 5.1.12 byl tento ukazatel předáván do funkce mysql_parse() a mysql_add_word() jako první parametr. Nyní je předáván ukaztel na celou strukturu MYSQL_FTPARSER_PARAM. Ukazatel slouží pro interní potřeby serveru a plugin by jej neměl měnit.

struct charset_info_st *cs - ukazatel uchovávající informaci o kódování query. V případě, že je hodnota NULL není informace dostupná.

char *doc - ukazatel na začátek query. Je využíván prvním modulem pro přístup k textu, který je následně rozparsován a vzniklými tokeny je naplněna interní struktura tagStore.

int length - specifikuje délku query, která obsahuje platný text určený ke zpracovávání. Tento parametr stejně jako query zpracovává první modul.

void *ftparser_state - mmanageru vychází z interface fulltext parseru. Interface obsahuje pět funkcí, které jsou popsány v kapitole 2.2.2 Vytvoření vlastních funkcí.

int flags - parametr přidaný ve verzi MySQL 5.1.12, který zatím může obsahovat pouze jeden flag MYSQL_FTFLAGS_NEED_COPY. Jeho nastavení serveru indikuje, že si má text, na který ukazuje parametr ve funkci mysql_add_word(), zkopírovat (není dostačující uložit si ukazatel). Zamýšlené využití je v případě, že pointer ukazuje mimo query určenou pointerem doc.

enum enum_ftparser_mode mode - mód parseru. Tato proměnná může nabývat tří hodnot:

MYSQL_FTPARSER_SIMPLE_MODE - mód pro rychlé indexování, ve kterém nejsou na text aplikovány ze strany serveru žádné úpravy textu ve smyslu neindexování příliš krátkých slov či odstraňování stopwords. Je na parseru, aby v případě, že chce aplikovat tato či další omezení, příslušně upravil slova předávaná k zaindexování.

MYSQL_FTPARSER_WITH_STOPWORDS - server sám provádí odstraňování stopwords dle specifikovaného seznamu a aplikuje nastavený limit na minimální délku slov pro zaindexování. V tomto módu se očekává, že serveru budou předána všechna slova textu včetně stopwords atp.

MYSQL_FTPARSER_FULL_BOOLEAN_INFO - parser má za úkol rozpoznat kromě samotných slov i speciální znaky související s vyhledáváním v boolean módu. Tyto znaky mají být následně předány pomocí indexátoru struktury MYSQL_FTPARSER_BOOLEAN_INFO. Ukazatel na zmíněnou strukturu je třetím parametrem funkce mysql_parse() a je poslán ke zpracování spolu se souvisejícím tokenem.

2.2. Tvorba vlastního modulu

2.2.1. Struktura tagStore

Vzhledem k tomu, že uvnitř mmageru je spouštěno více modulů, které mezi sebou potřebují sdílet aktuální verzi tokenů a dalších informací, byla pro tento účel vytvořena struktura tagStore. Konceptuálně vychází z myšlenky XML. V hashování s klíčem identifikačního čísla slova jsou uchovávány ukazatele na

objekty třídy `Word`. `Word` obsahuje skupinu tagů, které mohou kromě své hodnoty obsahovat atributy. Počet tagů a atributů není omezen, což umožňuje uchovávat všechny potřebné informace ke sdílení mezi moduly.

Nejdůležitějšími třemi tagy v tagu `word` jsou `TAG_BASE`, `TAG_CURRENT`, `TAG_DELETED`. Tagy jsou vloženy do objektu `word` prvním modulem, který je také naplní příslušnými hodnotami. Všechny hodnoty jsou brány jako interně jako string. Tento datový typ byl zvolen kvůli snažší manipulaci s řetězci.

`TAG_BASE` - základní tvar tokenu, další moduly tento tag neupravují a slouží k udržení informace o původním tvaru slov(a). To může být výhodné pro moduly zabývající se složitější analýzou textu pro kterou se nehodí pracovat s normalizovanými tvary slov. `TAG_CURRENT` uchovává aktuální tvar tokenu. Při prvním naplnění tagu se hodnota zpravidla shoduje s `TAG_BASE`. Změny tohoto tagu se očekávají od modulů provádějících normalizaci. Posledním zmíněným je `TAG_DELETED`, který obsahuje hodnotu 0 jako indikátor, že tento tag `word` (konkrétně obsah tagu `TAG_CURRENT`) je určen k zaindexování. Analogický je význam hodnoty 1, která je nastavována moduly pro odstraňování stopwords atp.

V kapitole 2.2 Tvorba vlastního modulu je demonstrována práce s touto třídou. Přesný popis je obsahem dokumentace.

2.2.2. Vytvoření vlastních funkcí

Modul mmanageru vychází z interface fulltext parseru. Interface obsahuje následujících pět funkcí:

```
int modNewModuleInitPlugin (void *);
int modNewModuleDeinitPlugin (void *);

int modNewModuleInitParse (MYSQL_FTPARSER_PARAM * param);
int modNewModuleDeinitParse (MYSQL_FTPARSER_PARAM * param);

int modNewModuleParse (MYSQL_FTPARSER_PARAM * param);
```

Pro ilustraci bude v této kapitole vytvořen modul nazvaný newModule, který bude obsahovat funkce `modNewModuleInitParse()`, `modNewModuleDenitParse()` a `modNewModuleParse()`. Zbývající funkce nebudou implementovány.

Jména funkcí nejsou pevně stanovena, ale je vhodné dodržovat alespoň konvenci s prefixem `mod` pro jména funkcí, která jsou součástí interface. U větších modulů toto rozlišení přispívá k větší přehlednosti kódu. Těchto pět typů funkcí se volá mmanagerem společně pro všechny zaregistrované moduly v pořadí, v jakém jsou zaregistrovány. To znamená, že např. při inicializaci pluginu mmanageru jsou zavolány postupně všechny funkce jeho modulů pro inicializaci modulu. Analogicky platí tento postup i pro ostatní funkce. Nejsou tedy volány nejdříve všechny funkce jednoho modulu, ale bloky dle jednotlivých typů.

Funkce `modNewModuleInitPlugin()`, slouží k úvodní inicializaci. mmanager volá tuto funkci v okamžiku načtení pluginu. Analogicky je tomu u funkce `modNewModuleDeinitPlugin()`, která je volána při deinicializaci pluginu. V těchto funkcích se mohou připravit (či odstranit) potřebné struktury pro práci modulu, které se vždy, příp. velmi často používají. Díky tomu lze optimalizovat rychlost alokace při parsování query a urychlit tak celý proces. MySQL umožňuje uložit, příp. číst

data do souboru, který je uložen v adresáři `mysql/data`. Soubor otevřený bez uvedení cesty (jen jménem), jak je naznačeno v následující ukázce kódu, se automaticky mapuje do zmíněného adresáře.

```
FILE * fp;
fp = fopen('datafile.dat');
```

Následující tři funkce obsahují jako parametr ukazatel na strukruru `MYSQL_FTPARSER_PARAM`, která je popsána v odstavci 2.1 Struktura MYSQL_FTPARSER_PARAM `Struktura` MYSQL_FTPARSER_PARAM.

Funkce `modNewModuleInitParse()`je volána před každým zpracováním query. To umožňuje provést inicializaci (smazání, úvodní nastavení) pro připravené struktury, které byly naalokovány při inicializaci pluginu. V ukázce probíhá naplnění proměnné před samotným parsováním.

```
int modNewModuleInitParse(MYSQL_FTPARSER_PARAM * param) {
      texttodel.append("Ehmm");
      return 0;
}
```

Funkce `modNewModuleDeinitParse` slouží k analogickému účelu jako `modNewModuleInitParse()` a je také volána po každém zpracování query. Změny v `tagStore` provedené v této funkci se již neprojeví v datech zaslaných serveru. V ukázce probíhá odstranění obsahu použité proměnné.

```
int modNewModuleDeinitParse(MYSQL_FTPARSER_PARAM * param) {
      texttodel.clear();
      return 0;
}
```

Funkce `modNewModuleParse()` obsahuje zpravidla hlavní logiku modulu. Pro ilustraci: první modul pro parsování query uložené v `doc` v této funkci provádí vytváření tokenů a jejich uložení do `tagStore`. Obecně se předpokládá, že v této části kódu budou prováděny operace se strukturou `tagStore`. Ačkoliv je do všech modulů předáván jako parametr ukazatel na strukturu `MYSQL_FTPARSER_PARAM`, nemělo by být potřeba ji použít. Všechny informace by se měly předávat pomocí interní struktury `tagStore`. Následující kód nastavuje pro první slovo ze struktury v případě, že je „Ehmm“ příznak DELETED. To v důsledku znamená, že pokud některý z následujících modulů příznak nezmění, nebude slovo předáno k zaindexování.

```
int modNewModuleParse(MYSQL_FTPARSER_PARAM * param) {
      NSTagStore::Word * w = Global::allMyWords.getWord(0);
      NSTagStore::Tag * tc = w->getTagByName(TAG_CURRENT);
      if (tc->getValue() == texttodel) {
            NSTagStore::Tag * t = w->getTagByName(TAG_DELETED);
            t->setValue("1");
      }
      return 0;
```

2.2.3. Registrace vlastního modulu

Registraci modulu je nutné provést v souboru `mmanger/configModules.h`. Do tohoto souboru se přidá include hlavičky souboru modulu, která obsahuje funkce k zaregistrování, aby bylo možné kód přeložit.

```
#include "../modules/mod_rmStopWords/rmStopWords.h"
#include "../modules/mod_rmDeletedWords/rmDeletedWords.h"
...
#include "../modules/mod_newModule/newModule.h"
```

Dále se do struktury `MM_MOD_FNC_API st_mod_fnc_parse` zapíší funkce, které se mají používat pro jednotlivé úkony. Struktura obsahuje ukazatele na funkce. Všechny funkce vracejí jako návratovou hodnotu int. Funkce pro inicializaci pluginu nemají parametry. Funkce pro parsování očekávají jako parametr ukazatel na strukturu `MYSQL_FTPARSER_PARAM`.

```
      {     0,                          //mod_plugin_init
            0,                          //mod_plugin_deinit
            modNewModuleInitParse,      //mod_init_parse
            modNewModuleDeinitParse,    //mod_deinit_parse
            modNewModuleParse,          //mod_parse
      },
```

Pokud by některé funkce měly prázdné tělo, je možné je v modulu vůbec nedefinovat a při registraci uvést na jejich místo nulu. Tím z části dojde i ke zrychlení zpracování (neprobíhají zbytečná volání). Funkce jsou volány v pořadí, které odpovídá umístení v této strukruře. Díky tomu lze snadno pořadí změnit, případně některý z modulů odstranit (stačí zakomentovat část kódu odpovídající registraci funkcí ve struktuře).

Soubory modulu je nutné přidat do souboru `Makefile`, aby se zajistilo jejich zařazení do výsledného zkompilovaného souboru a kompilace proběhla korektně.

2.2.4. Debugování a logování

mmanager umožňuje pohodlné debugování nově vytvářených modulů a jejich testování samostatně i s ostatními moduly. Pro testování modulu není typicky nutné vytvářet jednoúčelové struktury, které jsou dostupné na serveru. mmanger zajišťuje zpřístupnění struktury `MYSQL_FTPARSER_PARAM` a ve funkci `main()` je prováděno její naplnění, takže s ní moduly mohou pracovat ekvivalentně jako s instancí na serveru.

Debug mód se aktivuje definováním příslušných direktiv. Direktiva `DEBUG_MODE` zajišťuje tisk výstupů ladících funkcí. Direktiva `DEB_BUILD_MMANAGER` zajistí v souboru `mmanger.cpp` ponechání funkce `main()`, takže je možné modul přeložit jako spustitelný. Definice direktiv v souboru `MODShareHeader.h` je zapsána typicky takto (pro překlad na ostrý provoz bez debugovacích výstupů by měly být obě direktivy odstraněny):

```
#define DEBUGING_MODE
#define DEB_BUILD_MMANAGER
```

Debugování lze v zásadě provádět dvěma způsoby. První variantou je naplnění struktury pro registraci modulů a spuštění funkcí v pořadí, v jakém je bude pouštět server. Funkce `main()` pro tuto variantu vypadá takto:

```
// vytvoření instance struktury MYSQL_FTPARSER_PARAM
char * text_for_index = 0; //load file to memory
int text_length = 0; //length of text
if ((text_length =
     fetchFileToMem("~/text.txt", &text_for_index)) < 0)
  return 2;
MYSQL_FTPARSER_PARAM parser_param;
parser_param.length = text_length;
parser_param.doc = text_for_index;
modCreateBaseXMLParse(&parser_param); //parse column's data

//run standard procedure and print result
runModulesStandardWay(&parser_param);
```

V první části se plní struktura `MYSQL_FTPARSER_PARAM`, aby bylo vytvořeno adekvátní prostředí ke spuštění modulů. Následně proběhne už jen zavolání funkce `runModulesStandardWay()`, která zajistí zavolání všech pěti funkcí pluginu, které se volají během jeho životního cyklu. To vyvolá zaregistrované funkce modulů. Pro tento účel není pluginem volána funkce serveru pro předání tokenů `mysql_add_word`.

Druhý způsob umožňuje širší využití debugovacích nástrojů a snazší spuštění funkcí v atypickém pořadí. Pro tuto variantu je zamýšlen i ladící tisk, který zobrazuje aktuální stav struktury `tagStore` na standardní výstup ve XML formátu. Díky tomu je možné snadno sledovat, jak se ve struktuře projevují úpravy jednotlivých funkcí. Následující kód je možné použít po korektním naplnění `parser_param`.

```
Global::allMyWords.printContent(); //ladící tisk
modNewModuleInitParse(&parser_param);
Global::allMyWords.printContent();
modNewModuleParse(&parser_param);
Global::allMyWords.printContent();
modNewModuleDeinitParse(&parser_param);
```

mmanager obsahuje v namespace `NSBEAM` funkci pro logování zpráv do souboru. Tato funkcionalita najde využití v okamžiku, kdy je nutné ladit či zaznamenávat informace při běhu pluginu na serveru. Pokud chce modul používat logování, musí před zavoláním příkazu `NSBEAM::log()` provést inicializaci funkcí `NSBEAM::logInit()`. Logovací funkce umožňuje zapsat řetězec znaků, ale i číslo, či případně zpracovat složitější formátování. V následující části kódu je ukázán možný způsob použití.

```
int cislo=5;
char znaky[] = "kratky text";

NSBEAM::logInit(); //inicializace před prvním použitím
NSBEAM::log("--- logging session start ---\n\n");

NSBEAM::log("cislo: ");
NSBEAM::log(5);
NSBEAM::log("\n");
NSBEAM::log(5, "cislo %i je integer \n\n");

NSBEAM::log(znaky, "zobrazit %s je snadne\n");

NSBEAM::log("\n--- logging session end ---\n");
NSBEAM::logDeinit(); //ukončení logování
```

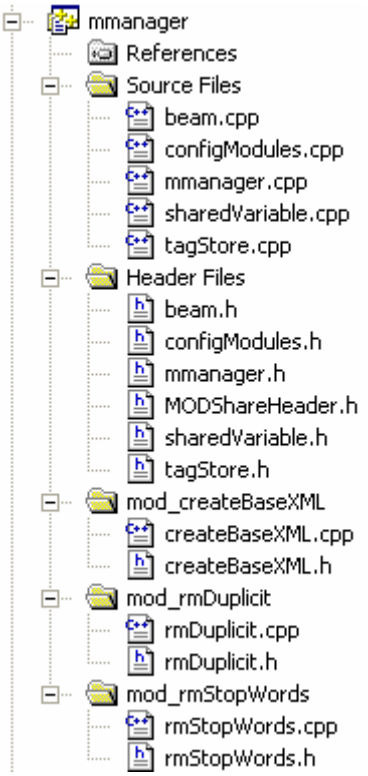
Výstup tohoto kódu by vytvořil soubor s následujícím obsahem.

```
--- logging session start ---

cislo 5
cislo 5 je integer

zobrazit kratky text je snadne

--- logging session end ---
```

2.3. Instalace mmangeru

2.3.1. Kompilace mmangeru

K přeložení mmanageru do tvaru sdílené knihovny je nutné mít v souboru `Makefile` zahrnutý soubor obsahující samotné tělo MySQL pluginu (`mmanger.cpp`), dále strukturu definující funkce pluginů (`configModules.h, configModules.cpp`). Navíc do překladu vstupují soubory modulů, kde má každý zpravidla jeden hlavičkový a jeden zdrojový soubor a další pomocné soubory jak je vidět na Obr.2.



Obr.2 - Soubory vstupující do překladu mmanageru

Pro překlad na UNIX-like systémech se používá program `make`. Samotné přeložení se provede následující sekvencí příkazů:

```
make clean
make
```

První příkaz odstraňuje soubory z předchozího překladu, proto jej není nutné používat v případě, že se jedná o první překlad. Druhý příkaz provede kompilaci a jeho výsledkem je soubor `mmanager.so` (plugin serveru MySQL).

2.3.2. Instalace a práce s pluginem v MySQL

Modul `mmanger.so` je serverem očekáván v adresáři `cesta-k-mysql-serveru/lib/mysql`. Typicky se kopírování provádí příkazem:

```
cp mmanger.so /usr/local/mysql/lib/mysql
```

Ověření nastavení cesty lze provést v souboru `my.cnf` typicky umístěném v `/etc/my.cnf`:

```
[mysqld]
plugin_dir=/home/mbaros/mmanager/mysql
```

Alternativou k ověření cesty je následující SQL příkaz

```
mysql> SHOW VARIABLES LIKE 'basedir';
+---------------+-----------------------------+
| Variable_name | Value                       |
+---------------+-----------------------------+
| basedir       | /home/mbaros/mmanager/mysql/ |
+---------------+-----------------------------+
1 row in set (0.00 sec)
```

Nyní je vše připraveno k tomu, aby byl plugin připojen do serveru. Provedeme připojení pomocí konzole.

```
#příkaz má tvar mysql –u jmenoUzivatele
#v případě účtu bez hesla
mysql –u mbaros

#v případě účtu s heslem – klient nabídne dialog
#pro vložení hesla
mysql –u root -p
```

Plugin se do serveru zavede příkazem `INSTALL PLUGIN`, ve kterém se zadává jméno pluginu a knihovna pluginu - v našem případě `mmanger.so`.

```
mysql> INSTALL PLUGIN mmanager_parser SONAME 'mmanager.so';
Query OK, 0 rows affected (0.01 sec)
```

Nainstalovaný plugin by se měl zobrazit v přehledu pluginů se statusem `ACTIVE`. Tím se ověří, že instalace proběhla úspěšně.

```
mysql> show plugins;
+-----------------+--------+----------------+-------------+
| Name            | Status | Type           | Library     |
+-----------------+--------+----------------+-------------+
| binlog          | ACTIVE | STORAGE ENGINE | NULL        |
...
| mmanager_parser | ACTIVE | FTPARSER       | mmanager.so |
+-----------------+--------+----------------+-------------+
8 rows in set (0.00 sec)
```

Parser plugin se přiřazuje ke každému fulltextovému indexu individuálně, což umožňuje široké možnosti přizpůsobení. Fulltext index lze definovat přímo při vytvoření tabulky, nebo po jejím vytvoření příkazem ALTER. Je možné zavést index i pro naplněnou tabulku.

V `ALTER` příkazu je nutné specifikovat sloupečky, pro které se bude index vytvářet. V následujícím příkladě je vytvořena tabulka v připravené databázi `texty` ukládající novinové články. Každý článek obsahuje nadpis, anotaci a samotný text článku (`telo`).

```
mysql> USE texty;
Database changed

mysql> DROP TABLE IF EXISTS clanky;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE clanky (
    ->  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->  nadpis VARCHAR(200),
    ->  anotace TEXT,
    ->  telo TEXT
    -> ) ENGINE=MyISAM DEFAULT CHARSET=utf8
    ->   COLLATE=utf8_czech_ci;
```

Po vytvoření tabulky vytvoříme fulltextový index nad sloupečky `nadpis` a `telo` s použitím parseru `mmanger_parser`.

```
mysql> ALTER TABLE clanky ADD FULLTEXT INDEX (nadpis, telo) WITH
PARSER mmanager_parser;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Manipulace s daty v tabulce se nijak neodlišuje od tabulky, ve které není plugin použit. Standardním použitím příkazu INSERT vložíme do tabulky testovací data pro ověření funkčnosti pluginu.

```
mysql> INSERT INTO clanky (nadpis, telo) VALUES
    ->  ('MYSQL instalace','Na stránkách mysql.com ...'),
    ->  ('Optimalizace MySql','Pro optimální chod ...'),
    ->  ('Tipy a triky','Představíme si 1000 triků pro...');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Instalací fulltext indexu se rozšířila klíčová slova pro použití v příkazu SELECT. Nejčastěji používaným dotazem je vyhledávání pomocí MATCH a AGAINST, které k nalezení hledaného slova využívá právě fulltext index. S výsledky dotazu lze opět pracovat běžným způsobem. Kromě toho lze výstup rozšířit o informaci o relevanci atd. Následuje několik typických dotazů využívajících fulltext index jako ukázka, jak jej lze využít.

```
mysql> SELECT * FROM clanky WHERE MATCH(nadpis, telo) AGAINST
('instalace');
+----+-----------------+---------+--------------------------+
| id | nadpis          | anotace | telo                     |
+----+-----------------+---------+--------------------------+
|  1 | MYSQL instalace | NULL    | Na stránkách mysql.com ..|
+----+-----------------+---------+--------------------------+
1 row in set (0.00 sec)
mysql> SELECT * FROM clanky WHERE MATCH(telo, nadpis) AGAINST
('instalace');
+----+-----------------+---------+---------------------------+
| id | nadpis          | anotace | telo                      |
+----+-----------------+---------+---------------------------+
|  1 | MYSQL instalace | NULL    | Na stránkách mysql.com ... |
+----+-----------------+---------+---------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT MATCH(nadpis, telo) AGAINST ('triky'), nadpis FROM
clanky;
+--------------------------------------+--------------------+
| MATCH(nadpis, telo) AGAINST ('triky') | nadpis             |
+--------------------------------------+--------------------+
|                                    0 | MYSQL instalace    |
|                                    0 | Optimalizace MySql |
|                  0.65545833110809 | Tipy a triky       |
+--------------------------------------+--------------------+
```

```
mysql> SELECT MATCH(telo) AGAINST ('1000'), nadpis FROM clanky;
ERROR 1191: Can't find FULLTEXT index matching the column list

mysql> ALTER TABLE clanky ADD FULLTEXT INDEX (telo) WITH PARSER
'mmanager';
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT MATCH(telo) AGAINST ('1000'), nadpis FROM clanky;
+-----------------------------+--------------------+
| MATCH(telo) AGAINST ('1000') | nadpis             |
+-----------------------------+--------------------+
|                           0 | MYSQL instalace    |
|                           0 | Optimalizace MySql |
|          0.67003107070923 | Tipy a triky       |
+-----------------------------+--------------------+
3 rows in set (0.00 sec)
```

Na prvním a posledním příkladu je navíc ukázáno, jak MySQL pracuje s indexy. Index je identifikován neuspořádanou *n*-ticí sloupečků (první ukázka dokazuje, že nezáleží na pořadí). Ale pokud je vytvořen index pro sloupečky `nadpis` a `telo` neznamená to, že je lze fulltextově vyhledávat ve sloupečku `telo`. Pro takové vyhledávání je nutné vytvořit vlastní index (ukázáno ve třetím příkladu). Ve světle úvodu do indexace v kapitole 1.3.2 Indexace je toto chování naprosto logické – MySQL si pro každou *n*-tici vytváří jeden index. Nelze tedy technicky vyhledávat s omezením na jeden z *n* sloupečků, protože informace o tom, z jakého konkrétního sloupečku slovo pochází, by byla nadbytečná a neukládá se.

3. Programátorská dokumentace

Tato kapitola popisuje funkce a datové struktury, které jsou používány pro komunikaci s MySQL serverem, moduly mmangeru a významné prvky mmangeru, které ovlivňují interface, uspořádány podle umístění do souborů, které kopíruje logicky související celky..

3.1. MySQL plugin

Funkce odpovídající interface fulltext pluginu zprostředkovávající spojení mezi serverem a moduly.

3.1.1. mmanager_parser_plugin_init

Inicializuje potřebné struktury po načtení fulltext pluginu serverem MySQL.

Deklarace

```
static int mmanager_parser_plugin_init(void*)
```

Parametry

Parametr není využíván.

Poznámka

V debug režimu je možné funkci zavolat přímo případně přes volání `runModulesStandardWay()`.Je využívána k dopředné inicializaci objektů k urychlení procesu zpracování textu.

3.1.2. mmanager_parser_plugin_deinit

Deinicializuje použité strukruty při odinstalaci fulltext pluginu ze serveru.

Deklarace

```
static int mmanager_parser_plugin_deinit(void*)
```

Parametry

Parametr není využíván.

Poznámka

Pro volání funkce platí stejná pravidla popsána v poznámce v čáti 3.1.1 mmanager_parser_plugin_init.

3.1.3. mmanger_parser_init

Inicializace pro aktuálně připravenou query.

Deklarace

```
static int mmanger_parser_init(
      MYSQL_FTPARSER_PARAM *param
      )
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Funkce je volána před každým parsováním query.

3.1.4. mmanger_parser_deinit

Inicializace pro aktuálně připravenou query.

Deklarace

```
static int mmanger_parser_deinit(
      MYSQL_FTPARSER_PARAM *param
      )
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Funkce je volána před každým parsováním query.

3.1.5. mmanger_parser_parse

Parsování query předané v parametru.

Deklarace

```
static int mmanger_parser_parse(
      MYSQL_FTPARSER_PARAM *param
      )
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Funkce je volána pro každou query. každým parsováním query. Tokeny jsou serveru předávány do funkce `add_word` k dalšímu zpracování – bližší popis této funkce je uveden v části 3.1.6 add_word.

Jsou rozlišovány podle parametru `param->type` módy parseru a v závisloti na nich jsou upraveny data posílaná k zaindexování.

3.1.6. add_word

Realizuje předání tokenů MySQL serveru.

Deklarace

```
static void add_word(
      MYSQL_FTPARSER_PARAM *param,
      char *word,
      size_t len
      )
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

`word` – ukazatel na začátek řetezce tokenu určeného ke zpracování.

`len` – platná délka řetězce tokenu.

Poznámka

Tokeny se předávají zavoláním funkce param->mysql_add_word() s parametry ukazatel na řetezec, délka řetězce a odkazem na strukturu využívanou v módu . V debug režimu je tato funkce nahrazena funkcí s prázdným tělem..

3.1.7. main

Entry point pluginu v případě, že je přeložen v debug režimu jako spustitelný soubor.

Deklarace

```
int main(int argc, char * argv[])
```

Parametry

`argc` – počet parametrů z příkazového řádku.

`argv` – pole ukazatelů na řetězce s parametry.příkazové řádky.

Parametr není využíván.

Poznámka

Pro přeložení pluginu jako spustitelného souboru je nutné kromě debug režimu definovat i direktivu `DEB_BUILD_MMANAGER`.

3.1.8. runModulesStandardWay

Spustí funkce modulů v pořadí ve kterém jsou zaregistrovány v poli `st_mod_fnc_parse`. Určeno pro konečnou fázi tvorby modulů – testování před nasazením.

Deklarace

```
int runModulesStandardWay(MYSQL_FTPARSER_PARAM * param)
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Je určeno pro volání v debug režimu v kódu funkce main. Pro spuštění je nutné kromě debug režimu definovat i direktivu `DEB_BUILD_MMANAGER`. Jsou volány za sebou všechny funkce fulltext pluginu ve stejném pořadí jako na serveru.

3.2. Beam

Funkce určeny pro informování o chybách, tisk ladících výpisů a logování informací do souboru příp. další běžně používané operace.

3.2.1. pDeb

Standardizovaný tisk debudovacích výstupů.

Deklarace

```
void pDeb(char * Message, char * Style)
```

Parametry

`Message` - ukazatel na řetězec obsahující ladící hlášku.

`Style` - nepovinný parametr umožňující definovat první parametr printf a tím přesněji ovlivnit .tvar výstupu. Příklad použití formátování je uveden v kapitole 2.2.4 Debugování a logování.

Poznámka

Výstupy se tisknou jen v případě, že je nastavena direktiva `DEBUGING_MODE`.

3.2.2. pErr

Obdoba příkazu assert zobrazující chybovou hlášku v případě, že je druhý parametr nastaven na true..

Deklarace

```
int pErr(char * Message, bool isErr)
```

Parametry

`Message` - ukazatel na řetězec obsahující ladící hlášku.

`isErr` - příznak chyby.

Poznámka

Pokud je nastaven `isErr` na `true` je považována hláška za chybu a je zobrazena chybová hláška. Funkce končí návratovou hodnotou `1`. V opračném případě je hláška potlačena a funkce vrací hodnotu `0`.

3.2.3. fetchFileToMem

Načtení celého souboru z disku do paměti.

Deklarace

```
int fetchFileToMem(const char* filename, char **result)
```

Parametry

`filename` - cesta k souboru, který se má načíst.

`result` – adresa ukazatele, který má po načtení souboru ukazovat na jeho začátek v paměti.

Poznámka

Funkce vrací hodnotu -1 v případě neúspěchu otevření souboru a -2 v případě, že selže na začátku nebo během procesu čtení. Při úspěšném ukončení vrací délku načteného souboru.

3.2.4. logInit

Inicializace funkčnosti logování.

Deklarace

```
int NSBEAM::logInit()
```

Parametry

Funkce nemá parametry.

Poznámka

Otevře soubor pro zápis logovacích informací, který je definován v souboru beam.cpp ukazatelem na řetězec `NSBEAM::logFilename`. V selhání je vypsána chybová hláška a vrácena hodnota `1`. Jinak je vracena hodnota `0`-

3.2.5. log

Ukládá číslo (resp. řetězce) příp. data předaná jako formátovací informace do souboru..

Deklarace

```
int NSBEAM::log(int m, char * style)
int NSBEAM::log(char * m, char * style)
```

Parametry

`int m` – message – číslo k uložení.

`char * m` – message – řetězec k uložení.

`style` - nepovinný parametr umožňující definovat první parametr printf a tím přesněji ovlivnit .tvar výstupu. Příklad použití formátování je uveden v kapitole 2.2.4 Debugování a logování.

Poznámka

Logování není závislé na nastavení debug módu, provádí se vždy pokud je voláno.

3.2.6. logDeinit

Ukončení logování.

Deklarace

```
int NSBEAM::logDeinit()
```

Parametry

Funkce nemá parametry.

Poznámka

Uzavření souboru (streamu) pro logování

3.3. modul CreateBaseXML

První modul. Zpracováván query a plní struktru `tagStore` pro další produkty. Jeho úkolem je prověst parsování textu na tokeny.

3.3.1. modCreateBaseXMLParse

Zpracová postupně query (param->doc). Určuje hranice tokenů.

Deklarace

```
int modCreateBaseXMLParse(MYSQL_FTPARSER_PARAM * param)
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Pro rozpoznání konce tokenu je volána funkce `is_space()`.Vzniklý řetězec je dále podstoupen zpracování do funkce `add_word_to_ts()`.

3.3.2. is_space

Vrací pro vložený znak zda je považován za oddělovač nebo ne.

Deklarace

```
int is_space(char * c)
```

Parametry

`c` – znak k vyhodnocení.

Poznámka

Za oddělovače jsou považovány znaky, které funkce `isalnum` vyhodnotí jako `false`.

3.3.3. add_word_to_ts

Předaný řetězec vloží jako nové slovo do struktury `tagStore`.

Deklarace

```
int modCreateBaseXMLParse(MYSQL_FTPARSER_PARAM * param)
```

Parametry

`param` – instance.

Poznámka

Zajišťuje vytvoření základní struktury tagů a jejich registraci voláním funkce `Word::addTag()`.

3.4. modul rmDuplicit

Nastavení příznaku smazáno TAG_DELTED pro druhé a další výskyty stejného slova. Modul porovnává hodnoty v tagu TAG_CURRENT a je tedy možné zařazením za modul provádějící převod na základní tvar dosáhnout dstranění i všech tvarů daného slova. Kromě odstraňování slov je modul schopen upravit relevanci slov tak, aby nebyla porušena rovnováha odstraněním duplicitních výskytů.

3.4.1. modRmDuplicit

Označuje znovu se vyskytující slova jako smazaná.

Deklarace

```
int modRmDuplicit(MYSQL_FTPARSER_PARAM * param)
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Rozpoznání opakujících je prováděno hashováním, kde klíčem jsou slova a je vracen objekt `ReadedItem`, který uchovává informaci o počtu výskytů a identifikačním čísle slova v `tagStore`.

Pokud je nastaven příznak v proměnné `NSModRD::recountRel` je po průchodu tagStore upravena relevance jednotlivým slovům.

3.5. modul rmStopwords

Načte soubor stopwords jehož cesta je uložena v proměnné `NSRSW::stopWordsFile` a následně podle tohoto seznamu označuje slova, která mu odpovídají jako smazaná nastavením příznaku 1 v TAG_DELETED.

3.5.1. modRmStopWordsParse

Nastavuje příznak 1 v TAG_DELETED slovům ze seznamu stopwords.

Deklarace

```
int modRmStopWordsParse(MYSQL_FTPARSER_PARAM * param)
```

Parametry

`param` – instance struktury MySQL serveru obsahující text k parsování s dalšími rozšiřujícími informacemi. Detailnější popis je uveden v části 2.1 Struktura MYSQL_FTPARSER_PARAM.

Poznámka

Seznam stopwords je textový soubor ve kterém je na každém řádku přávě jedno slovo.

3.6. nastavení direktiv

Globální direktivy se nachází v souboru src/modules/MODShareHeader.h. Tento soubor je prostřednictvím include vkládán do všech modulů, hlavního souboru pluginu příp. pomocných souborů. Direktivy určují uzpůsobení kódu (sdílená knihovna, spustitelný soubor) a definují jména pro tagy.

3.6.1. DEBUGING_MODE

Nastavuje debug mód. Některé funkce jsou dostupné jen v tomto režimu, který je určen pro ladění nových modulů a součinosti s ostatními.

3.6.2. DEB_BUILD_MMANAGER

V rámci debug režimu figuruje tato direktiva jako rozšířené nastavení, které upraví kód pluginu (zejména přidá entry point main) což umožní přeložit kód jako aplikaci a testovat plugin krokovat a používat další standardní nástroje k ladění.

Kromě toho je možné využít speciálního ladícího tisku volaného funkcí `Global::allMyWords.printContent()`., který na stdout vypíše obsah struktury `tagStore` ve formátu XML.

3.6.3. definice TAG...

Direktivy s prefixem TAG_ jsou určeny pro definování názvů tagů pod kterými budou ukládány do `tagStore`. Proto je vhodné mít tento seznam na jednom místě, aby se předešlo kolizím jmen. V programu se nepoužívají názvy přímo, ale vždy přes definované TAG_... aliasy. Díky tomu je možné v případě problémů se skutečným jménem v `tagStore` jen upravit na jednom místě název tagu. Aktuálně nastavené jména tagů jsou tyto:

```
#define TAG_WORD "word"
#define TAG_INDEX "index"
#define TAG_BASE "base"
#define TAG_CURRENT "current"
#define TAG_DELETED "deleted"
#define TAG_RELEVANCE "relevance"
```

Hodnoty, které se používají pro práci s tagy by měly být také definované v zmíněném hlavičkovém souboru, aby bylo možné snadno provést případnou změnu. Aktuálně nastavené hodnoty jsou tyto:

```
#define NOT_ENOUGH_PARAMETERS "Enter name of file to create xml."
#define TAGNAME_DELETED "deleted"
#define TAGNAME_NTF "ntf"
#define TAGNAME_TF "tf"
#define DELETED_TRUE "1"
#define DELETED_FALSE "0"
#define TAG_RELEVANCE_BASIC_VAL "0.2"
#define TAG_DELETED_BASIC_VAL DELETED_FALSE
```

3.6.4. DEBUGING_MODE

Nastavuje debug mód. Některé funkce jsou dostupné jen v tomto režimu, který je určen pro ladění nových modulů a součinosti s ostatními.

3.6.5. DEBUGING_MODE

Nastavuje debug mód. Některé funkce jsou dostupné jen v tomto režimu, který je určen pro ladění nových modulů a součinosti s ostatními.

4. Závěr

Cílem bakalářské práce bylo vytvořit modulární fulltextový vyhledávač. Po analýze možností, které by umožnily tuto funkcionalitu, jsem zvolil implementaci modularity pomocí MySQL pluginu zejména kvůli dopředné kompatibilitě. Pro tento plugin bylo vytvořeno rozhraní, které umožňuje snadno přidávat, odstraňovat a měnit pořadí modulů. Podle mého názoru byl cíl bakalářské práce splněn. Plugin poskytuje kromě možnosti zapojení modulu možnost jeho ladění ve dvou variantách a v případě potřeby provádět logování činností při běhu na serveru.

Pro usnadnění tvorby vlastního modulu byly představeny kroky, které je nutné učinit pro jeho implementaci. Věřím, že Module manager najde uplatnění jako nástroj pro tvorbu nových pluginů umožňujících řešit i složitější lingvistické otázky související s tvorbou tokenů a prácí s nimi.

.

5. Literatura

[1] MySQL – http://www.mysql.com
[2] Freeman Eric, Freeman Elizabeth (2004): Head First Design Patterns
[3] Dubois Paul (2005): MySQL, Third Edition, Que
[4] Galamboš Leo (2004): přednáška Dobývání informací z Webu (SWI107)
[5] Hajič Jan (2005): Disambiguation of Rich Inflection (Computational Morphology of Czech). Karolinum, Praha