Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Michal Král

Dotazování databází a webu

Katedra softwarového inženýrství

Vedoucí diplomové práce: Prof. RNDr. Jaroslav Pokorný, CSc.

Studijní program: Informatika (datové inženýrství)

Za pomoc při vytváření této práce bych chtěl především poděkovat svému vedoucímu Prof. RNDr. Jaroslavu Pokornému, CSc., který mi po celou dobu od výběru tématu až po výslednou verzi poskytoval mnoho cenných rad.

Dále pak mé poděkování patří také majitelům serveru Jyxo.cz, že mi dovolil zatěžovat jejich vyhledávač.

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

V Praze dne Michal Král

Obsah

Název práce: Dotazování databází a webu

Autor: Michal Král

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: Prof. RNDr. Jaroslav Pokorný, CSc.

e-mail vedoucího: pokorny@ksi.ms.mff.cuni.cz

Abstrakt: Databáze a web jsou dvě prostředí, kde se nejvíce používá vyhledávání. Navíc v řešení mnoha problémů čerpá jedna skupina vývojářů od druhé. Ve většině projektů ale oba zdroje informací zůstávají stále odděleny a současné čerpání dat z obou je spíše vyjímkou. Tato práce se věnuje zapojení webových vyhledávačů přímo do SQL dotazů. Kromě částečného zmapování prací spojujících obě tématiky se zde věnujeme i technickým aspektům propojení včetně metody asynchronní iterace. Jádro práce však spočívá v realizaci tří funkcí volajících vyhledávač a zároveň vhodných k využití v dotazech. Vybranými funkcemi jsou www_rank, která k výrazu vrací jeho důležitost, dále pak www_near, která ke dvěma výrazům vrací hodnotu značící míru vzájemné souvislosti, a konečně agregační funkce www_best_address, která k množině výrazů vrátí jejich společný nejlepší odkaz. K tomuto účelu byl vybrán databázový server Oracle 10g a internetový vyhledávač Jyxo.cz.

Klíčová slova: WSQ, databáze, web

Title: Query databases and web

Author: Michal Král

Department: Department of software engineering

Supervisor: Prof. RNDr. Jaroslav Pokorný, CSc.

Supervisor's e-mail address: pokorny@ksi.ms.mff.cuni.cz

Abstract: Databases and web are two environments, where searching is used the most. In addition one group of developers draw on the experience of the other one in solving a lot of problems. In majority of projects, both sources of information stay isolated and synchronous convey from both is rather exception. This dissertation follow integration of internet searchers directly into SQL queries. Besides partly analysing papers connecting both subjects, it describes technical aspects of this connection, including method of asynchronous iteration. The gist of this paper lay in realization of three functions, which call web searcher and at the same are suited for use in DB queries. Selected functions are www_rank, which returns relevancy of phrase, then it is www_near, which returns relative relationship of two phrases and the last one is agregation function www_best_address, which returns the best common link for group of phrases. For this purpose we chose database server Oracle 10g and Jyxo.cz web searcher.

Keywords: WSQ, database, web

Kapitola 1

Úvod

Většina dat je v současné době uložena buď na internetu ve formě webových stránek nebo v relačních databázových systémech. Z hlediska kvality a uložení informací však jde o naprosto opačné přístupy. Na jedné straně internet jako zdroj informaci téměř bez jakýchkoliv omezujících podmínek a na druhé databáze, velmi rychlá ve vyhledávání dat. Jako již velké množství jiných se i tato práce pokusí oba rozdílné systémy propojit, zaměří se hlavně na využití informací získaných na internetu pro rozšíření možností dotazování nad DB. Zároveň však musí řešit problém že data na webu nejsou nijak strukturovaná, a proto je nutné, pro udržení kvality informací požadované po databázi, brát v potaz údaje pouze v nějakém hromadném tvaru, který zvětší jejich věrohodnost.

Cílem této práce je právě tuto bariéru alespoň částečně prolomit a navrhnout a implementovat několik funkcí, které budou schopné dodat doplňující informace k údajům z databáze tak, že k jejich získání použijí webový vyhledávač a zároveň udrží důvěryhodnost dat. Nejdříve se ale ve druhé kapitole pokusíme zmapovat práce spojující prostor databází a prostor vyhledávání na webu, ukážeme jak starší práce, které se úzce dotýkají této, jako i velmi aktuální trendy. Následující kapitola pojednává jednak o technice zvané asynchronní iterace, která může vhodně doplnit i výsledky této práce, tak o technických možnostech propojení mezi databází a internetovým vyhledávačem a prochází možnosti vybraných DB serverů i několika vyhledávačů. V poslední kapitole před závěrem se pak už věnujeme výše zmíněným funkcím. Jde o následující funkce:

www_rank - vstupním parametrem je vyhledávaný textový řetězec a výstupem číslo značící používanost řetězce na internetu (viz. kapitola 4.2 na straně 36)

www_near - vstupem jsou dva textové parametry a výstupem číslo ur-

čující, jak moc si jsou tyto dva řetězce blízké (viz. kapitola 4.3 na straně 44)

www_best_address - vstupním parametrem je opět vyhledávaný řetězec a výstupním je podle vyhledávače nejlepší adresa k danému řetězci. Použití i jako agregační funkce a výsledkem nemusí být přímo adresa, ale může to být i jen doména (viz. kapitola 4.4 na straně 51)

Ke každé z těchto funkcí si nejdříve projdeme její využití, potom různé způsoby její realizace a jejich výhody či nevýhody, pak vlastní způsob implementace a případná problémová místa a nakonec provedeme rozbor výsledků nad testovacími daty. Samotná implementace pak je provedena nad databázovým serverem od společnosti Oracle verze 10g a s využitím internetového vyhledávače Jyxo.cz.

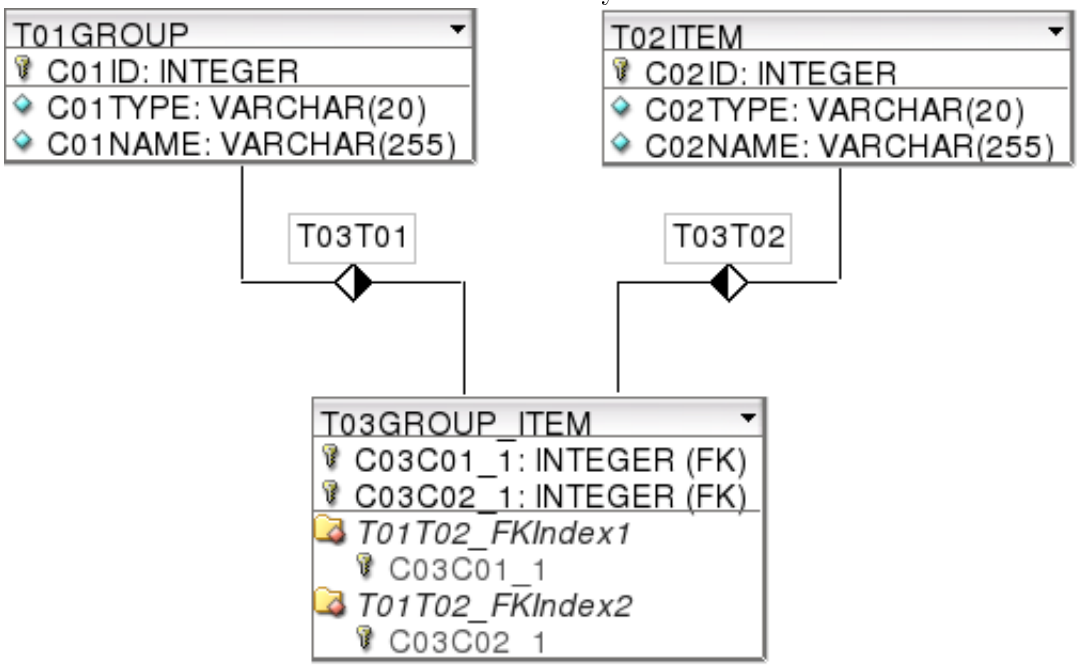
1.1 Datový model

Pro lepší vysvětlení příkladů a pro vlastní ověření kvality funkcí budeme používat jednoduchý datový model, který obsahuje pouze tři tabulky. První obsahuje seznam číselníků, nazveme ji **T01GROUP**, druhá seznam k nim odpovídajících položek, tuto pojmenujeme **T02ITEM**, třetí bude vazební tabulka mezi první a druhou, přisoudíme jí jméno **T03GROUP_ITEM**. Tabulky budeme plnit daty z několika skupin a budeme pozorovat chování funkcí v jednotlivých případech, proto přidáme do tabulek s daty ještě položku TYPE, která bude rozlišovat jednotlivé skupiny. Datový model graficky znázorňuje obrázek 1.1

Takto můžeme do tabulky T01GROUP vložit záznamy představující třeba kontinenty (všechny záznamy budou mít ve sloupečku C01TYPE hodnotu "STATY_CS"), do tabulky T02ITEM vybrané státy (opět bude jako typ ve sloupci C02TYPE uvedeno "STATY_CS") a ve vazební tabulce uvedeme, které město patří do kterého státu. Právě příklad se státy a městy použijeme pro ukázkové dotazy v úvodních kapitolách i jako základní příklad pro zkoumání chování hledaných funkcí.

Datový model by šlo ještě vylepšit tak, že by vazba mezi položkou a číselníkem byla realizována přímo jako odkaz z tabulky T02ITEM, ale pro náš účel bude lepší držet se tohoto obecnějšího modelu, který umožňuje jedné položce patřit do více číselníků.

Obrázek 1.1: Datový model



Kapitola 2

Propojení DB a internetu

Vzájemné ovlivňování se je na problémech dotazování nad DB a dotazování nad internetem patrné již dlouhou dobu. Oblasti prolínání jsou velmi různorodé a v této kapitole zmíníme jenom vzorek vybraný s ohledem na téma práce. Jako zástupce přímo konkurenčního projektu vybereme WSQ/DSQ popsaný v [1A], jako ukázku poznatků načerpaných u DB využitých u webu pak techniky uvedené v [2A] a velmi aktuální projekt Yahoo!Pipes popsaný například v [3A], v poslední podkapitole ještě krátce představíme opačným směrem vedené úvahy reprezentované článkem [4A].

2.1 WSQ/DSQ

WSQ/DSQ [:wisk-disk:] je zkratka užívaná pro dva způsoby propojení webových vyhledávačů a jazyka SQL. Zkratka WSQ znamená "web-supported (database) queries" a značí použití informací nalezených na internetu. Druhá vznikla z "database-supported (web) queries" (DSQ) a používá se pro opačný přístup, tedy použití DB při dotazech nad internetem.

2.1.1 DSQ

Při použití nejrozšířenějších internetových vyhledávačů už je každý dotaz vyhodnocován jako DSQ, ale pro uživatele je toto rozšíření ukryto. Jako příklad použijeme český vyhledávač JYXO.

Základním kamenem každého vyhledávače je seznam stránek, které prohledává, přesněji řečeno těch, které už prohledal a poznamenal si jejich obsah (takzvaná indexace). Ke stránce je možné udržovat i další údaje ulehčující pozdější prohledávání a ideálním nástrojem pro to je databáze. Dalším užívaným využitím DB je seznam slov, jejichž vyhledávání nemá smysl, v češtině je

to třeba spojka "a" nebo sloveso "je", obecně všechna velmi často se vyskytující slova bez vlastního vztahu k tématu (spojky, citoslovce, některá slovesa). Poslední, pro uživatele nepříliš užitečnou a oblíbenou, ale pro provozovatele vyhledávače výhodnou vlastností spojenou s DB, je uveřejňování placených odkazů.

Jak lze vidět, prohledávat internet úplně bez využití DB vlastně nelze a vždy se bude jednat o DSQ.

2.1.2 WSQ

Naopak využití vyhledávačů není při psaní dotazů nad DB příliš časté. Hlavní cesta dosavadního výzkumu směřovala přes namapování vyhledávače na virtuální stránky v DB a jejich následné využití v dotazech.

Konkrétně se používají dvě tabulky: **WebPage** a **WebCount** (v případě využití více vyhledávačů, rozlišují se koncovkou _GO pro Google, _JX pro Jyxo a podobně). Struktura tabulky WebPage vypadá následovně:

SearchExp - textový sloupeček značící hledaný řetězec, ve formátu jako printf na UNIXu. Tady například "%1 NEAR %2"

T1 - text dosazovaný za %1 v SearchExp

T2 ,T3 , ... - obdobně jako T1

URL - adresa nalezené stránky

Rank - pořadí nalezené stránky v seznamu vráceném vyhledávačem

Date - datum poslední aktualizace stránky

První sloupečky, jejichž počet závisí na tvaru SeachExp, slouží jako vstup do vyhledávače a po nahrazení %x řetězcem ve sloupečku Tx tvoří dohromady text vložený do vyhledávače. Poslední tři sloupečky vyhledávač vrací. Tabulka je nekonečná, protože pro každý nový hledaný řetězec obsahuje nové záznamy. Druhou tabulkou je WebCount, která je vlastně agregací první. Krom stejných sloupečků tvořících dotaz obsahuje už jen sloupeček **Count**, obsahující počet vrácených odkazů. WebCount lze realizovat i jako pohled vytvořený nad WebPage, ale tím ztratíme výhodu, že informace o počtu vrácených stránek vrací vyhledávač téměř okamžitě, kdežto vlastní odkazy jsou opožděny.

Pokud bychom chtěli například pomocí těchto tabulek simulovat naši první funkci, mohlo by to vypadat třeba takto:

Zdrojový kód 1 Dotazy k ilustraci WWW_rank

```
SELECT C01NAME,
       WWW_rank(C01NAME)
FROM T01GROUP;

~

SELECT C01NAME,
        WebCount.Count
FROM T01GROUP INNER JOIN
     WebCount ON C01NAME = WebCount.T1
              AND WebCount.SearchExp = ''%1'';
```

V obou případech nám DB ke každému číselníku vrátí ještě jeho důležitost podle internetu. Zde pomocí funkce získáme hlavně přehlednost a možná lepší chování, ale brát důležitost podle počtu vrácených odkazů je první a vcelku rozumný návrh na implementace naší funkce.

Druhá funkce je opět číselná, takže půjde modelovat relativně jednoduše. Můžeme to zkusit třeba takto:

Zdrojový kód 2 Dotazy k ilustraci WWW_near

```
SELECT C02NAME,
       C01NAME,
       WWW_near(C01NAME, C02NAME)
FROM T01GROUP,
     T02ITEM
ORDER BY C02NAME,
        WWW_near(C01NAME, C02NAME);

~

SELECT C01NAME,
       C02NAME,
       WebCountBoth.Count/
        (WebCountGroup.Count+WebCountItem.Count)
FROM T01GROUP INNER JOIN
     WebCount WebCountGroup ON WebCountGroup.T1 = C01NAME
                            AND WebCountGroup.SearchExp = ''%1'',
     T02ITEM INNER JOIN
     WebCount WebCountItem ON WebCountItem.T1 = C02NAME
                           AND WebCountItem.SearchExp = ''%1'',
     WebCount WebCountBoth
WHERE WebCountBoth.T1 = C01NAME
  AND WebCountBoth.T2 = C02NAME
  AND WebCountBoth.SearchExp = ''+%1 +%2''[1]
ORDER BY C01NAME,
  WebCountBoth.Count/(WebCountGroup.Count+WebCountItem.Count);
```

Rozdíly mezi oběma zápisy jsou v podstatě stejné jako v předchozím případě. S funkcí získáme jednodušší zápis, ale hlavně můžeme k získání jejího výsledku použít složitější výpočty.

Napodobit poslední funkci by bylo o poznání náročnější. Nejdříve si pro zjednodušení definujme následující pohled, který nám pro každý číselník a pro každý odkaz vrácený alespoň pro jednu položku tohoto číselníku vrátí počet, kolikrát byl tento odkaz vrácen dohromady pro všechny položky tohoto číselníku:

Zdrojový kód 3 Vytvoření pohledu V04URLCount

```
CREATE VIEW V04URLCount AS
SELECT C03C01_1 C04C01_1,
       WebPage.URL C04URL,
       COUNT(*) C04Count
FROM T03GROUP_ITEM INNER JOIN
     T02ITEM ON C03C02_1 = C02ID INNER JOIN
     WebPage ON C02NAME = WebPage.T1
            AND WebPage.SearchExp = ''%1''
GROUP BY C03C01_1,
         WebPage.URL;
```

Potom můžeme funkci WWW_best_address nepřesně přiblížit takto:

Zdrojový kód 4 Dotazy k ilustraci WWW_best_address

```
SELECT C01NAME,
       WWW_best_address(C02NAME)
FROM T01GROUP INNER JOIN
     T03GROUP_ITEM ON C01ID = C0301_1 INNER JOIN
     T02ITEM ON C0302_1 = C02ID
GROUP BY C01ID;

~

SELECT C01NAME,
       URL
FROM T01GROUP INNER JOIN
     V04URLCount ON C01ID = C04C01_1
WHERE C04Count = (SELECT MAX(COUNT)
                  FROM V04URLCount V04
                 WHERE C01ID = V04.C03C01_1);
```

Tentokrát už bude rozdíl v chování obou variant výraznější. V případě simulace pomocí WebPage pravděpodobně bude každý odkaz vrácen právě jednou a ve výsledku nezískáme žádnou použitelnou informaci. Navíc se pomocí SQL velmi těžko realizuje omezení na rovnosti odkazů na doménu (odkazy jsou různé, ale mají stejnou doménu až do 4. řádu). Pokud by druhý dotaz vrátil smysluplný odkaz, který by se pro celý číselník vyskytl v odpovědích od vyhledávače vícekrát, pak by tento měl být i výsledkem hledané funkce.

Shrneme-li naše závěry, tak při použití funkcí získáváme klasické výhody zapouzdření: jednodušší zápis, funkčnost zapsánu na jednom místě a v neposlední řadě ušetříme pisatele dotazu od zjišťování struktury dvou nových tabulek. Navíc my v této práci jednou zjistíme ideální vzorečky pro funkce a uživatel je později může používat jako černé skříňky. Všechny výhody se ještě znásobí, pokud budeme chtít místo jednoho vyhledávače použít více různých vyhledávačů a výsledky spojit dohromady.

2.2 Jazyky pro dotazování nad webem

Mnoho práce bylo provedeno na poli aplikace konceptů vytvořených databázovou komunitou pro dotazování nad webem. Viditelným výsledkem pro běžného uživatele internetu je využití vzájemných odkazů mezi stránkami pro ohodnocení jednotlivých stránek u výsledků internetových vyhledávačů. Dnes už si vyhledávání pouze na základě obsahů stránek bez jejich vztahů téměř ani nedovedeme představit, ale stále je běžně dostupná jen malá možnost zapojit do dotazu i právě grafovou strukturu webu. Do výbavy vyhledávačů zatím patří pouze možnost omezit dotaz na stránky odkazující na určitou adresu. Složitější podmínky však zadávat nelze.

V [2A] lze ovšem nalézt ukázky několika jazyků určených k dotazování nad orientovaným grafem, jakým je například web, a zároveň inspirovaných jazyky určenými k dotazování nad databázemi. Například jazyk WebSQL pracuje s celým webem jako s částí relačního modelu obsahujícího dvě virtuální tabulky, v jedné existuje záznam pro každou internetovou stránku a v druhé záznam pro každý odkaz. Samotná struktura dotazů pak vychází, jak název jazyka napovídá, z jazyka SQL, určeného pro dotazování nad relačními databázemi. Obdobně existují i jazyky vycházející z Datalogu nebo OQL, stejně jako jazyky, které místo z konceptu relačních databází vycházejí ze stromové struktury nebo jiných grafových struktur.

Všechny tyto projekty se od tématu zpracovávaného v této práci liší tím, že sice využívají znalostí získaných při vývoji a dotazování nad databázemi, samotné jazyky však slouží pouze k dotazům na webem nebo jinou obdobnou grafovou strukturou, ale neumožňují zároveň využít údajů uložených přímo v DB. Naopak tyto jazyky umožňují složitější konstrukce s ohledem na vzájemné vztahy jednotlivých stránek.

2.3 Yahoo! Pipes

Zástupcem nejmodernějších trendů spojujících databáze a internet je projekt Yahoo! Pipes popsaný v článku [3A]. Tato služba umožňuje přistupovat k některým službám, především ke kanálům RSS, a v podstatě k webu obecně, jako k databázi. Rozhraní je grafické a tudíž přístupnější i neprogramátorské veřejnosti, ale podobnost s jazyky pracujícími s DB je zřetelná. Srovnání s SQL se přímo nabízí. Nejdříve se vyberou zdroje dat, typicky RSS kanály, ty lze poté pomocí podmínky spojovat k sobě a nakonec se nad daty provede filtrace. Odpovídá to tedy struktuře dotazů obsahujících klauzule FROM, INNER JOIN a WHERE.

Na tomto příkladu je vidět jeden ze základních rozdílů ve vyhledávání nad DB proti aktuálním směrům vývoje prohledávání webu. Při práci nad databází se očekává, že se jednou provede dotaz a jeho výsledek se víceméně nemění, zatímco velká část práce nad webem se věnuje zpracovávání nikoliv starých informací, ale těch nově vzniklých. Yahoo! Pipes jsou typickým příkladem, kdy hlavním cílem je filtrovat přidaná data, ale výsledek v nějakém konkrétním čase mě příliš nezajímá. Z role zjištění aktuálního stavu se dotazy přesouvají do role komorníka, který k nám pustí pouze ty zprávy, o které máme zájem a chrání nás od zahlcení.

2.4 Dotazy na k nejlepších výsledků

Základním principem dotazů na internetové vyhledávače je výsledek složený pouze z k nejlepších výsledků a navíc seřazených podle důležitosti. Využití stejného principu při vyhledávání v relačních databázích lze najít v [4A]. Toto je ukázka, kdy technika prozkoumaná a používaná v internetových vyhledávačích může najít uplatnění i v aplikacích postavených nad relační databází.

Podobné dotazy lze očekávat například u systému obsahujících seznam objektů v prostoru, kdy uživatel hledá nejbližší objekt ke zvolené souřadnici nebo obecně u systémů založených na fuzzy logice. Důležitým se tato technika stane především pokud budeme údaje získávat z nějakého zdroje s pomalou odezvou, tedy například z webu, kdy počet vyhodnocovaných záznamů velmi ovlivňuje časovou náročnost celého dotazu.

Kapitola 3

Technické aspekty propojení

V této kapitole se budeme zabývat využitelností vybraných databázových strojů pro zapojení internetového vyhledávače do SQL a v první podkapitole i optimalizací vyhledávání nad zdrojem s dlouhou odezvou, jakým je například internetový vyhledávač. Především nás tedy bude zajímat možnost vytváření virtuálních tabulek, také cesty k propojení přímo s webovým vyhledávačem a v neposlední řadě i varianty vytváření nových SQL funkcí. Jako DB servery jsme vybrali Oracle, MS SQL a MySQL, jelikož patří v ČR mezi nejrozšířenější a MySQL je světově nejpoužívanější DB stroj s otevřeným kódem.

3.1 Asynchronní iterace

Pokud chceme vysvětlit pojem WSQ/DSQ je vhodné ještě vysvětlit druhý, velmi často zároveň se vyskytující, tedy pojem asynchronní iterace. Jde o způsob, jak vyřešit problém pomalých dotazů WSQ, kdy se sériově volá velké množství dotazů na webové vyhledávače a zbytečně se nečinně čeká na odpověď, přestože vyhledávač umí zpracovat velké množství dotazů najednou. Základní myšlenka spočívá v tom, že virtuální tabulky místo hodnoty vrací pouze odkaz na ni, co nejdelším počítáním pouze s tímto odkazem a nahrazení odkazu hodnotou až později.

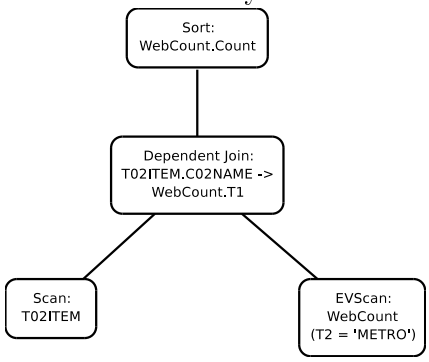
Základní cíl je jasný, minimalizovat čas stání databázového stroje a tím urychlit výpočet celého dotazu. Druhým a neméně podstatným cílem je realizovatelnost ve stávajících DB strojích. Nejlépe se celý postup ukáže na jednoduchém příkladu uvedeném v dotazu 5.

Tímto dotazem získáme ke každému státu počet odkazů, které obsahují jednak název státu jednak také slovo "METRO", výsledek je pak seřazen právě podle počtu odkazů.

Zdrojový kód 5 Dotaz pro ukázku asynchronní iterace

```
SELECT C01NAME, WebCount.Count
FROM T02ITEM,
    WebCount
WHERE C02NAME = WebCount.T1
  AND WebCount.T2 = 'METRO'
  AND WebCount.SearchExp = '+T1 +T2'
ORDER BY WebCount.Count;
```

Obrázek 3.1: Základní vyhodnocovací strom

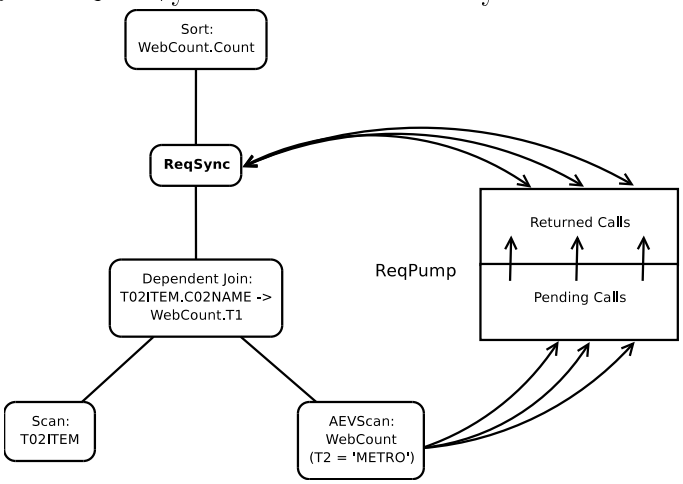


Obrázek 3.1 zobrazuje jeden z možných vyhodnocovacích plánů ukázkového dotazu. Pro tento i všechny následující plány budeme uvažovat vyhodnocovací model založený na iterátoru, který podporuje operace *Open*, *GetNext* a *Close*. Operace *Dependent join* potom vyžaduje pro každé volání *GetNext* na svého pravého syna, vyhodnocení vazby na svého levého syna. *EVScan* je operátor průchodu externí virtuální tabulky (pro nás tabulek WebPage a WebCount). Pokud bychom nevyužili paralelismu, *EVScan* by v průběhu tohoto plánu prováděl sekvenci volání vyhledávače (jedno pro každé volání *GetNext*) a procesor na DB stroji by vždy nečinně stál. Přirozeně by bylo nejlepší vyvolat více prohledávání internetu bez zatěžování procesoru paralelizací. Pro dosažení takového chování použijeme asynchronní iteraci, techniku zahrnující tři komponenty:

1. Pozměněnou verzi *EVScan* nazývanou *AEVScan*.
2. Nový fyzický operátor nazývaný *ReqSync* (z anglického Request Synchronizer, v doslovném překladu dotazový synchronizátor).
3. Modul ReqPump, které řídí vnější asynchronní volání.

Základní myšlenkou je změna vyhodnocovacího plánu nahrazením *EVScans* pomocí *AEVScans* a vložením jednoho nebo více *ReqSync* operátorů do plánu.

Obrázek 3.2: Vyhodnocovací strom asynchronní iterace



AEVScans a *ReqSync* operátory společně komunikují s globálním modulem *ReqPump*. Další změny plánu ani jiné nové operátory už pak nejsou k podpoře asynchronní iterace potřeba.

Nyní si na našem příkladu ukážeme chování asynchronní iterace. Obrázek 3.2 v porovnání s obrázkem 3.1 obsahuje *AEVScan* místo *EVScan*, byl přidán operátor ReqSync a je začleněn ReqPump. V průběhu vytváření řádků ve vyhodnocovacím procesu umožňujeme, aby libovolná hodnota sloupce byla označena pomocí speciálního řetězce, který slouží ke dvěma účelům:

1. Označuje, že hodnota atributu je neúplná.

2. Jednoznačně určuje volání ReqPump s touto neúplnou hodnotou, která čeká na správné vyplnění po skončení tohoto volání.

Na operátory *Scan* a *Sort* nemá asynchronní iterace žádný vliv, ani *Dependent Join* jí není nijak ovlivněn. Nyní přistoupíme k *AEVScan*. Když *Dependent Join* dostane nový záznam z T02ITEM (jeho levý syn), zavolá *Open* na *AEVScan* a poté *GetNext* s T02ITEM.C02NAME. *AEVScan* v tu chvíli pošle zprávu *ReqPump*, zaregistruje volání V s T1 = T02ITEM.C02NAME a T2 = 'METRO', kde V je jednoznačný identifikátor volání. *ReqPump* je modul obstarávající asynchronní volání sítě a ukládání výsledků. V případě volání V je výsledkem hodnota WebCount.Count. *ReqPump* tedy uloží tuto hodnotu do hash tabulky *ReqPumpHash*, kde klíčem záznamu bude hodnota V. Poté co *AEVScan* zaregistruje svoje volání u *ReqPump*, vrátí do *Dependent Join* (jako výsledek *GetNext*) jeden řádek T z WebCount, kde sloupeček Count obsahuje speciální řetězec, který identifikuje volání V. *Dependent Join* poté spojí T s T02ITEM.C02NAME a vrátí nový řádek svému předku (*ReqSync*).

Zdrojový kód 6 Dotaz pro asynchronní iteraci s různým počtem řádků

```
SELECT C01NAME, WebPage.URL
FROM T01GROUP,
    WebPage
WHERE C01NAME = WebPage.T1
  AND WebPage.Rank <= 3
  AND WebPage.SearchExp = 'T1'
ORDER BY C01NAME, WebPage.Rank;
```

Teď je na řadě *ReqSync*. Když je zavolána metoda *Open* pomocí Sort, *ReqSync* zavolá *Open* na *Dependet Join* níže a pak volá dále *GetNext* až do získání všech výsledků, přičemž si ukládá všechny vrácené řádky do paměti (varianta s úplným ukládáním do paměti byla zvolena pro svou jednoduchost). *ReqSync* musí spolupracovat s *ReqPump* tak, aby mohl nahrazovat skutečné hodnoty ve výsledku před tím, než je vrátí svému předku. Jejich vzájemná komunikace je variantou na standardní problém producent/zákazník, proto ho zde nebudeme více rozebírat. Za zmínku stojí především poznatek, že řádky které nijak nezávisí na *ReqPump* mohou přes *ReqSync* projít rovnou a bez čekání.

Doposud jsme se vyhnuli jednomu problému, který v úvodním jednoduchém příkladu nenastával. V něm jsme se mohli spolehnout, že tabulka WebCount vrátí vždy jeden záznam při každém spojení s ní, ale u tabulky WebPage už toto neplatí. Ta může k některým pro některé hodnoty sloupečků *SearchExp* a *Tx* obsahovat více než jeden řádek a pro některé naopak nemusí obsahovat žádný. Znovu použijeme k osvětlení příklad, tentokrát uvedený jako dotaz 6.

Ukázkový dotaz nám ke každému státu v tabulce T01GROUP vrací maximálně tři nejdůležitější dotazy, které obsahují název státu. Vyhodnocovací plán k tomuto příkladu je uveden na obrázku 3.3, kde *ReqPump* může na jeden dotaz vrátit 0, 1, 2 nebo 3 záznamy.

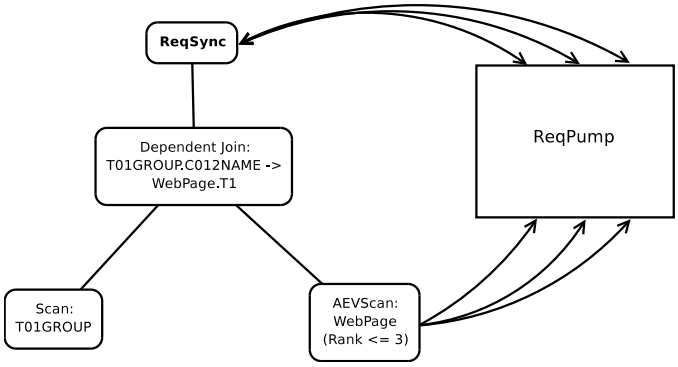
Problém se řeší na úrovni *ReqSync* následovně:

1. Pokud *ReqPump* k dotazu C nevrátí žádný záznam, pak *ReqSync* si řádek T, který vyžaduje výsledek tohoto dotazu vymaže.

2. Pokud *ReqPump* k dotazu C vrátí právě jeden záznam, pak *ReqSync* podle něj doplní T.

3. Pokud *ReqPump* k dotazu C vrátí 2 nebo 3 záznamy, potom *ReqSync* dynamicky zkopíruje T a jednotlivé verze doplní.

Obrázek 3.3: Vyhodnocovací plán k dotazu s různým počtem řádků



Pro bližší informace, výsledky výkonostních testů a zdroj této kapitoly odkazuji na článek [1A].

3.2 Dynamické pohledy

Nejdříve se podíváme na způsoby vytváření dynamických pohledů WebCount a WebPage. V teoretické rovině mají tyto pohledy nekonečné množství sloupců, ale v praxi tuto vlastnost nepodporují ani samotné vyhledávače, například české Jyxo.cz bere v potaz maximálně 15 hledaných slov a nejpoužívanější Google sice 32, ale stále jde o konstantní počet. Navíc, dotazy obsahující více než 5 slov už nemívají moc dobrý smysl a ani výsledky. Můžeme se proto rozhodnout pro omezení i našich pohledů na nejvýše 5 sloupců Tx (v této práci vystačíme dokonce jen se dvěma). Pokud využijeme jenom první dva, tak ostatní pak v dotazu můžeme nechat libovolné, jelikož stejně nejsou obsaženy v SearchExp, ale nejlepší je nastavit je na prázdný řetězec nebo NULL. Výsledná definice tabulky WebPage by vypadala jako v dotazu 7 a jednoduchý dotaz třeba jako v dotazu 8.

3.3 Propojení SQL a internetových vyhledávačů

Druhý problém nastává s plněním těchto pohledů. Pokud se chceme držet způsobu, kdy nezasahujeme do vlastního databázového stroje, pomocí SQL přístup k informacím z vyhledávače také nemáme a pomocí PL/SQL (resp. TSQL nebo jiného obdobného jazyka) také ne, zbývá nám už jen propojení na nějaký jiný programovací jazyk, který bude dělat prostředníka mezi DB a internetovým vyhledávačem. Všechny tři námi vybrané stroje tuto možnost podporují.

Zdrojový kód 7 Definice tabulky WebPage

```
CREATE TABLE WebPage(
  SearchExp VARCHAR(100) NOT NULL,
  T1 VARCHAR(30) NOT NULL,
  T2 VARCHAR(30) NULL,
  T3 VARCHAR(30) NULL,
  T4 VARCHAR(30) NULL,
  T5 VARCHAR(30) NULL,
  URL VARCHAR(500) NOT NULL
  Rank NUMBER NOT NULL,
  Date DATE NOT NULL
);
```

Zdrojový kód 8 Jednoduchý dotaz nad tabulkou WebPage s omezeným počtem sloupců

```
SELECT C01NAME,
       WebPage.URL
FROM T01GROUP INNER JOIN
     WebPage ON C01NAME = WebPage.T1
            AND WebPage.SearchExp = ''%T1''
            AND WebPage.T2 = NULL
            AND WebPage.T3 = NULL
            AND WebPage.T4 = NULL
            AND WebPage.T5 = NULL;
```

3.3.1 UDF v MySQL

Nejdříve si projdeme možnosti, které nabízí MySQL ve verzi 5.1. Pro podrobnější informace je dostupná dokumentace [3W], především kapitola 26.3, zde uvedeme pouze základy. Uživatelsky přidávané funkce se zde jmenují UDF (z anlického User-Defined Function) a mají následující možnosti:

- mohou vracet textové řetězce nebo čísla (přirozená i reálná)
- lze definovat funkce pracující s jednou hodnotou i agregační funkce
- mohou kontrolovat počet a typy parametrů, které dostanou na vstupu
- může říci, jestli může vrátit NULL nebo vždy konkrétní hodnotu a NULL znamená chybu

Syntaxe vytvoření UDF vypadá následovně:

```
CREATE [AGGREGATE] FUNCTION název_funkce
RETURNS{STRING|INTEGER|REAL|DECIMAL}
SONAME název_sdílené_knihovny;
```

kde AGGREGATE značí, jestli jde o agregační funkci, za RETURNS následuje návratový typ a za SONAME název knihovny, ve které je funkce uložena. Pro zrušení UDF je syntaxe obdobná:

```
DROP FUNCTION název funkce;
```

Funkce samotné pak musí být psány v C nebo C++ a pokud chceme používat UDF, pak musíme mít server MySQL zkompilován dynamicky. Pokud tedy chceme do MySQL přidat funkci XXX(), pak musíme v C napsat následující funkce:

- XXX() - povinná
  hlavní funkce, obsahující vlastní počítání hodnoty funkce. Mapování jednotlivých datových typů SQL a odpovídajících v C/C++ je popsáno v tabulce 3.1.

  Tabulka 3.1: Odpovídající datové typy SQL a C/C++ pro UDF

SQL	C/C++
STRING	char *
INTEGER	long long
REAL	double

- XXX_init() - volitelná
  využívá se pro kontrolu počtu argumentů, kontrolu datových typů argumentů, alokaci paměti, specifikaci maximální délky výsledné hodnoty funkce nebo specifikaci, jestli může být návratová hodnota funkce rovna NULL.

- XXX_deinit() - volitelná
  slouží k dealokaci paměti, kterou funkce XXX_init() alokovala.

Pokud chceme vytvářet agregační funkci, přibudou ještě dvě další:

- XXX_clear() - povinná ve verzi 5.1
  vymaže hodnotu agregační funkce, aniž by měla další hodnotu jako parametr.

- XXX_add() - povinná
  přidá argument k hodnotě agregační funkce.

MySQL pak volá jednotlivé C/C++ funkce u agregační funkce následovně:

1. Zavolá XXX_init() pro připravení paměti pro ukládání výsledků.
2. Srovná si tabulky podle GROUP BY výrazu.
3. Zavolá XXX_clear() pro první hodnotu v každé skupině.
4. Zavolá XXX_add() pro každý záznam patřící k téže skupině.
5. Zavolá XXX() pro získání hodnoty skupiny pokud se mění skupina záznamů nebo byl zpracován poslední záznam.
6. Opakuje kroky 3, 4 a 5 dokud existují řádky k zpracování.
7. Zavolá XXX_deinit() pro uvolnění paměti.

3.3.2 CLR funkce v MS SQL

MS SQL server poskytuje velmi podobnou funkčnost, ovšem pod názvem CLR. Opět budeme čerpat z oficiální dokumentace, tentokrát tedy z [4W] z kapitoly "CLR User-Defined Functions". Hlavní rozdíl je, že CLR musí být naprogramovány v některém z .NET jazyků (C#, Visual Basic). V celém konceptu je znát, že DB server i prostředí pro CLR funkce patří pod jednoho výrobce, takže jsou k sobě více srostlá. Stejně jako v minulém případě existuje mapování mezi jednotlivými datovými typy a opět existují funkce skalární i agregační, ale ještě navíc i funkce vracející tabulku.

Pokud budeme vytvářet skalární funkci, která vrací jednu hodnotu, máme práci proti MySQL trochu jednodušší v tom, že nám stačí pouze jedna funkce. Pokud si proces chceme osvětlit příkladem, pak část napsaná v C# je uvedena ve zdrovojém kódu 9 a odpovídající deklarace funkce v TSQL už je záležitostí několika řádků kódu:

```
CREATE ASSEMBLY FirstUdf FROM 'FirstUdf.dll'[1]
CREATE FUNCTION CountSalesOrderHeader() RETURNS INT
AS EXTERNAL NAME FirstUdf.T.ReturnOrderCount
```

[1] Za předpokladu, že funkce ReturnOrderCount byla zkompilována do knihovny FirstUdf.dll.

Zdrojový kód 9 Jednoduchá CLR v C#

```
using Microsoft.SqlServer.Server;
using System.Data.SqlClient;
public class T
{
    [SqlFunction(DataAccess = DataAccessKind.Read)]
    public static int ReturnOrderCount()
    {
        using (SqlConnection conn
          = new SqlConnection("context connection=true"))
       {
            conn.Open();
            SqlCommand cmd = new SqlCommand(
"SELECT COUNT(*) AS 'Order Count'
 FROM SalesOrderHeader", conn);
            return (int)cmd.ExecuteScalar();
        }
    }
}
```

Oproti MySQL zde máme výhodu, že můžeme definovat i funkce, které vrací výsledkovou sadu a nám by se tedy velmi hodily právě místo dynamických view WebPage a WebCount. Opět si pouze ukážeme jednoduchý příklad, který zprostředkovává čtení z EventLogu. Část napsaná v C# viz. zdrojový kód 10, zaregistrování odpovídající funkce v TSQL se pak příliš neliší od skalární funkce

```
CREATE ASSEMBLY tvfEventLog
FROM'D:\assemblies\tvfEventLog\tvfeventlog.dll'
WITH PERMISSION_SET = SAFE
GO
CREATE FUNCTION ReadEventLog(@logname nvarchar(100))
RETURNS TABLE
(logTime datetime, Message nvarchar(4000),
  Category nvarchar(4000), InstanceId bigint)
AS
EXTERNAL NAME tvfEventLog.TabularEventLog.InitMethod
GO
```

a její použití v TSQL na získání posledních 10 záznamů pak vypadá následovně

```
SELECT TOP 10 *
FROM dbo.ReadEventLog(N'Security') as T
```

.

Zdrojový kód 10 Tabulková CLR funkce v C#

```
using System;
using System.Data.Sql;
using Microsoft.SqlServer.Server;
using System.Collections;
using System.Data.SqlTypes;
using System.Diagnostics;
public class TabularEventLog
{
    [SqlFunction(FillRowMethodName = "FillRow")]
    public static IEnumerable InitMethod(String logname)
    {
        return new EventLog(logname, Environment.MachineName).Entries;
    }
    public static void FillRow(Object obj, out SqlDateTime timeWritten,
out SqlChars message, out SqlChars category, out long instanceId)
    {
        EventLogEntry eventLogEntry = (EventLogEntry)obj;
        timeWritten = new SqlDateTime(eventLogEntry.TimeWritten);
        message = new SqlChars(eventLogEntry.Message);
        category = new SqlChars(eventLogEntry.Category);
        instanceId = eventLogEntry.InstanceId;
    }
}
```

Pokud dáme SearchExp a T1 až T5 jako vstupní parametry funkce, tak můžeme vytvořit funkci, která nám bude vracet výsledkovou sadu, kterou očekáváme od WebPage, resp. WebCount, dokonce už přímo s omezením hodnot, takže ušetříme několik WHERE-podmínek.

Poslední variantou CLR funkcí jsou funkce agregační. Stejně jako u MySQL zde je potřeba napsat více funkcí než ve skalárním případě, ale jejich složení a účel je malinko jiný. Třída, která reprezentuje agregační funkci musí obsahovat následující metody:

public void Init() volá se na začátku každé skupiny v GROUP BY, připravuje výpočet agregace
odpovídá funkci XXX_clear() pro MySQL

public void Accumulate(input-type value) provádí přidání nové hodnoty do agregace, většinou obsahuje vlastní logiku agregace
odpovídá funkci XXX_add() pro MySQL

public void Merge(function_class other) slouží k sloučení dvou podvýsledků agregace v případě, že byly počítány nezávisle
pro MySQL nemá odpovídající funkci

public return_type Terminate() dokončuje agregaci a vrací výsledek, volá se na konci každé skupiny v GROUP BY
odpovídá funkci XXX() pro MySQL

Pro názornost další jednoduchý příklad, tentokrát bude funkce jako vstupní parametry dostávat znakové řetězce a výstupem bude jejich spojení s ", " jako oddělovačem. Zdrojový kód třídy v C# lze najít v ukázce kódu s číslem 11 na straně 30.

3.3.3 Java funkce v ORACLE

I DB server od firmy Oracle nabízí podobné možnosti jako MS SQL. Z hlediska množství podporovaných jazyků, jde v podstatě o doplněk. Podporované jsou C i C++, ale nejblíže je mu java. Od verze 8i dokonce Oracle DB server obsahuje vlastní Java Virtual Machine. Druhy funkcí, které můžeme vytvářet pak jsou stejné. Podporované jsou skalární, tabulkové i agregační funkce. Společné omezení je, že všechny metody, které chceme používat v DB, musí být napsány jako **public** a **static**. Při dodržení této podmínky už lze v podstatě libovolnou metodu naprogramovanou v jazyce java zaregistrovat v Oracle serveru a volat ji pak přímo z SQL nebo PL/SQL. Pro tabulkové a agregační funkce pak je stejně jako jinde potřeba metod více a i jejich názvy nejsou libovolné.

Postup pro propojení databázového stroje Oracle s javou lze nalézt v dokumentaci na stránkách [2W] a případně i v příloze této práce, kde jsou uloženy všechny zdrojové kódy vedoucí k výsledkům. Zde si ukážeme pouze jednoduchý příklad se skalární funkcí.

Prvním krokem je napsání java třídy obsahující public static proceduru, kterou chceme volat z DB. Třídu nazveme *simple*, napíšeme do ní krátkou proceduru *getText*, která bude vracet neměnný text a uložíme do souboru *simple.java*.

Zdrojový kód 12 Třída simple

```
public class Simple {
  // return some text
  public static String getText(){
    return "Berte život s úsměvem, stejně z něho nevyjdete živi.";
  }
}
```

Druhým krokem je přeložení této třídy pomocí java překladače. Podporovaná verze je 1.4, s verzí 1.5 a vyšší bývají problémy a pokud to jde, je lépe se jim vyhnout. Na překlad stačí jednoduchý příkaz

```
javac simple.java
```

Tím získáme soubor *simple.class*, který musíme načíst do DB, aby s ním šlo dále pracovat. Na to již potřebujeme DB server a jeho utilitu loadjava. Pokud chceme volat tuto proceduru ze schématu *www_search*, které má heslo třeba *password*, pak použijeme příkaz

```
loadjava -user www_search/password simple.class
```

Další fáze už probíhá v PL/SQL pod zvoleným uživatelem www_search, který už má nyní vytvořenou třídu k dispozici. My budeme chtít zpřístupnit proceduru getText jako funkci tohoto uživatele, což provedeme příkazem

```
CREATE FUNCTION someText
RETURN VARCHAR2
AS
LANGUAGE JAVA
NAME 'simple.getText() return java.lang.String';
```

Funkci someText() nyní můžeme volat v rámci SQL dotazů a pokud vše proběhlo správně, bude nám vracet očekávaný text.

3.4 Internetové vyhledávače

Posledním velkým problémem, který nám zůstává je, jak předat informace z internetového vyhledávače našim procedurám napsaným v C, C++, C#, javě nebo podobném jazyce, který je následně předá DB serveru.

V součastnosti nejznámější Google (www.google.com) poskytoval až do 5.12.2006 java třídu, která umožňovala přímo přístup k jeho vyhledávacímu

stroji pouze za registraci a dodržení několika omezujících podmínek ohledně zatěžování serveru (maximální počet dotazů denně a minimální časová pauza mezi dvěma dotazy). Nyní už podobnou možnost má pouze pro JavaScript, což je vhodné pro internetové stránky, ale už ne pro naše účely. Obdobně další světový vyhledávač Yahoo (www.yahoo.com) aktuálně žádný přímý vhodný nástroj neposkytuje. Stejně je na tom i dva z českých strojů Jyxo.cz a Morfeo.

V této situaci máme několik možností. Z hlediska programování je nejjednodušší se přímo domluvit s majitelem serveru, jelikož pravděpodobně všichni program na přímé dotazování bez využití webových stránek mají, ale neposkytují ho veřejně. Pro akademický projekt by se určitě rozumné podmínky domluvit daly. Druhou možností je načítat data přímo z www stránek daného serveru. U všech tří zmiňovaných lze napsat dotaz pomocí změny adresy (vyhledávaná slova jsou součástí adresy). Toto řešení není z hlediska programátorské etiky úplně čisté a opět by bylo vhodné kontaktovat majitele serveru.

Pokud chceme výsledky komerčně využít nebo dál rozšiřovat, pak nás nic jednoduchého nečeká. Buď můžeme vytvořit vlastní internetový vyhledávač (třeba bez www rozhraní) a nebo počkat, až se podobný program třeba v javě objeví, ať už pod komerční nebo jinou licencí. V ideálním případě se domluví mezi sebou některý výrobce DB serverů a majitelé vyhledávacích strojů a možnosti zkoumané v této práci budou dostupné už přímo v některé z dalších verzí DB serverů.

Obecně není příliš důležité, který vyhledávač bude zvolen. Lze i získávat výsledky od několika různých a tyto hodnoty potom vzájemně kombinovat. Množství poskytovaných funkcí je v současné době velmi podobné a i množství indexovaných stránek (ty, které vyhledávač při dotazu prohledává) není nijak drasticky různé. Záleží zde tedy více na osobních preferencích než na funkčnosti.

Zdrojový kód 11 Třída agregační funkce v C#

```
using System;
using System.Data;
using Microsoft.SqlServer.Server;
using System.Data.SqlTypes;
using System.IO;
using System.Text;
[Serializable]
[SqlUserDefinedAggregate(
    Format.UserDefined,
    IsInvariantToNulls = true,
    IsInvariantToDuplicates = false,
    IsInvariantToOrder = false,
    //maximalni velikost v bytech v persistentnim stavu
    MaxByteSize = 8000)
]
public class Concatenate : IBinarySerialize
{
    /// Proměnná udržující aktuální výsledek kontaktenace
    private StringBuilder intermediateResult;

    /// Úvodní nastavení vnitřní proměnné
    public void Init() {
        this.intermediateResult = new StringBuilder();
    }

    /// Přidání další hodnoty, pokud je NULL, pak nic nepřidávám
    public void Accumulate(SqlString value) {
        if (value.IsNull) { return;}
        this.intermediateResult.Append(value.Value).Append(',');
    }

   /// Připojení částečně dopočítané agregace k této
    public void Merge(Concatenate other) {
        this.intermediateResult.Append(other.intermediateResult);
    }

    /// Dopočítání konečného výsledku a jeho vrácení
    public SqlString Terminate() {
        string output = string.Empty;
        // Případné vymazání koncové čárky
        if (this.intermediateResult != null
            && this.intermediateResult.Length > 0)
        { output = this.intermediateResult.ToString(0,
this.intermediateResult.Length - 1);}
        return new SqlString(output);
    }
}
```

Kapitola 4

Zapouzdření vyhledávačů pro SQL

4.1 Obecné vlastnosti

4.1.1 Architektura

V této kapitole si ukážeme různé možnosti, jak počítat výsledky našich funkcí. Pro technickou realizaci byl vybrán DB server od firmy Oracle (Oracle USA, Inc.), přesněji Enterprise Edition verze 10g2. Důvodů je několik. Proti MySQL serveru poskytuje lepší propojenost s jazykem java a proti MS SQL serveru disponuje větší přenositelností mezi systémy a opět lepší přístup k programům napsaným v javě. Bohužel Express Edition stejného DB stroje neumožňuje používání javy vůbec, takže ji také nelze použít.

Pří výběru programovacího jazyka bylo potřeba brát ohledy na dvě vlastnosti, které ve výsledku rozhodly pro využití jazyku java. První a důležitější požadavek byl přístup k výsledkům internetového vyhledávače. Ve prospěch javy svědčila například již výše uvedená možnost využití API funkcí od serveru google.com, která byla bohužel později znemožněna, ale také jednoduchá práce s internetovými stránkami, která umožňuje získat výsledky vyhledávání přes www rozhraní vyhledávače. Pro ostatní jazyky lze využít například knihovny CURL. Pokud by ovšem některý renomovaný vyhledávač poskytl v jazyce, který lze volat z DB serveru, rozhraní pro vyhledávání, bylo by vhodné výběr jazyka znovu zvážit. My jsme se vydali cestou www rozhraní a využití knihovny java.net.URL, která nám poskytla jednoduchý přístup k výsledkům. To do značné míry ovlivnilo i výběr DB stroje, jejichž snadné propojení byl druhý požadavek na programovací jazyk. Ovšem byla by možná kombinace MS SQL a třeba jazyka C#.

Při výběru vhodného vyhledávače hrála svou roli především schopnost

nastavení výsledků vyhledávání pomocí změny adresy stránky. Vzhledem k využití i dat v českém jazyce, je velkou výhodou, pokud prohledávač sám nabízí skloňování a doplňování diakritiky. Nakonec nejlépe vyhověl těmto požadavků server Jyxo.cz, který všem požadavkům vyhovuje velmi dobře. Umí skloňovat i časovat české výrazy i automaticky doplňovat diakritiku, zároveň lze pomocí adresy stránky s výsledky ovlivnit jak počet zobrazených výsledků, tak různě parametrizovat dotaz. Lze určit, jestli se hledané slovo má vyskytovat v obsahu stránky, v jejím titulku nebo dokonce v adrese nalezené stránky. Lze nastavit, aby se ve výsledku nezobrazovaly stejné nebo podobné odkazy a odkazy ze stejné domény. Všechny tyto vlastnosti se nám budou v dalším průběhu velmi hodit. Skladbu odkazu, pomocí které dosáhneme parametrizace dotazu, si ukážeme podrobněji:

`http://jyxo.cz/s?q=ochrana+title%3Ajelen+url%3Abeloocasy&cnt=21&o=nostem,noacc,noduprem,nocls`[1]

http://jyxo.cz/s? toto je úvodní část dotazu, která je stejná pro všechny dotazy

q=..& tato část obsahuje vlastní dotaz

ochrana požadujeme, aby se v obsahu stránky vyskytlo slovo "ochrana"

+title%3Ajelen dále chceme, aby titulek stránky obsahoval slovo "jelen"

+url%3Abeloocasy adresa hledané stránky pak má obsahovat "beloocasy"

cnt=21 maximální počet zobrazených odkazů na první stránce s výsledky nastavíme na 21

o=... nastavení vlastností ohledně českého jazyka

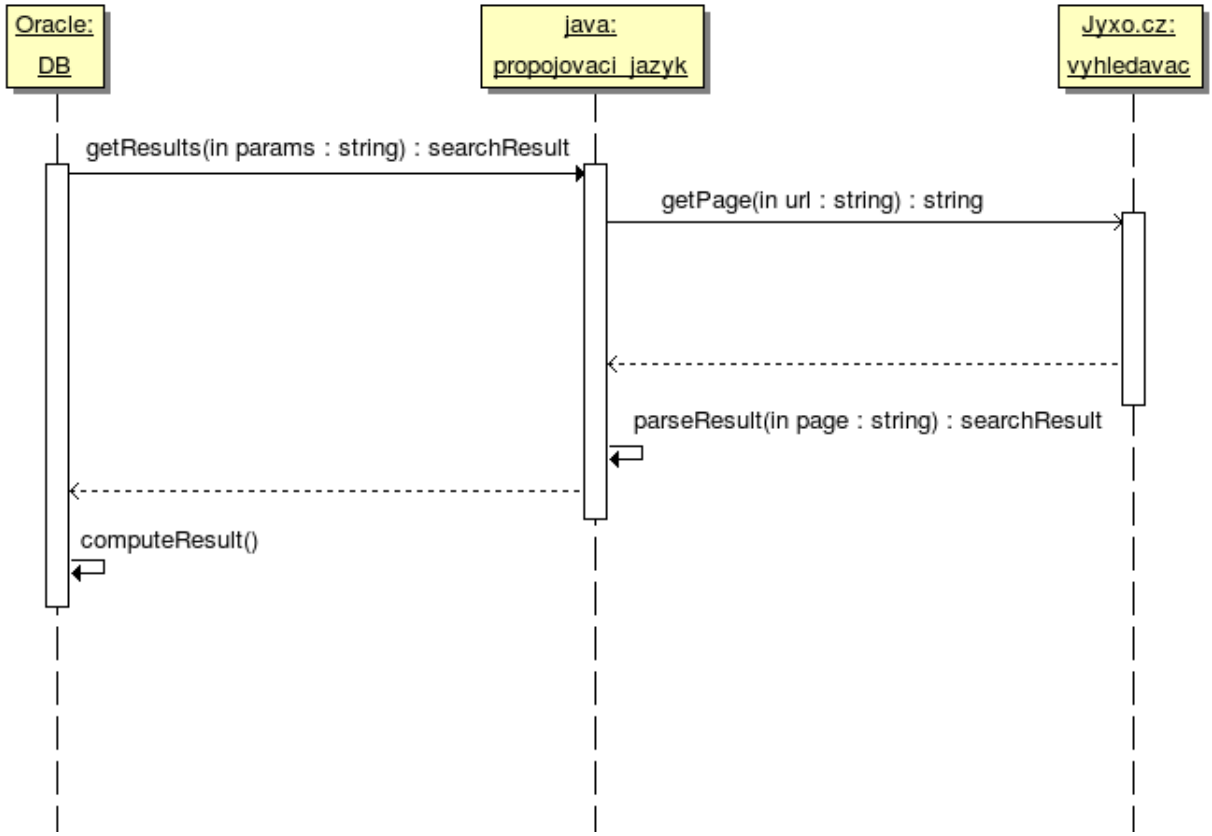
stem, nostem zapnutí/vypnutí skloňování a časování slov

acc, noacc zapnutí/vypnutí doplňování diakritiky

duprem, noduprem zapnutí/vypnutí odstraňování totožných a velmi podobných stránek

cls, nocls zapnutí/vypnutí zobrazování maximálně 2 stránek z každé domény

[1] Odkaz je rozdělen na dva řádky pouze z důvodů lepšího zobrazení, jedná se o souvislý text odkazu.



Ani tento výčet vlastností není úplný a o všech schopnostech a dovednostech tohoto vyhledávacího portálu se nejlépe dozvíte z nápovědy (http://jyxo.cz/d/help).

Pro naše účely jsou na výsledkové stránce užitečné jenom některé údaje. V mnohých situacích si vystačíme s odhadovaným počtem výsledných odkazů k dotazu, ale někdy budeme navíc ještě potřebovat i vlastní seznam odkazů, samozřejmě ve správném pořadí.

Z hlediska volání jednotlivých prostředí probíhají výpočty všech funkcí velmi podobně. Z databáze je volána funkce v javě, která jako parametr dostane dotaz, skládající se z vyhledávaného výrazu a dalších parametrů pro vyhledávání. Tato funkce složí adresu s výsledky, načte tuto stránku, získá z ní důležité údaje a ty poté vrátí zpět do DB, která z nich dopočítá celkový výsledek funkce. Pro lepší názornost zobrazuje toto volání obrázek 4.1.

Pokud v rámci jedné DB funkce bude potřeba získat výsledky na několik různých dotazů, může se v ní volání java funkcí s různými parametry opakovat a celkový výsledek DB funkce se pak počítá z jednotlivých podvýsledků získaných pomocí javy.

4.1.2 Data

Testovací data, která budeme používat pro ověření výsledků funkcí, jsou uložena v datovém modelu znázorněném na obrázku 1.1. Všechna data jsou uvedena v prvním pádě a bez diakritických znamének. Využíváme možnosti, že diakritiku doplní vyhledávač sám a my obejdeme problémy s kódováním

českých znaků. Zároveň z toho však plyne, že nebudeme výrazy hledat jako fráze, ale podle jednotlivých slov. U víceslovných výrazů tedy nebudeme hledat pouze stránky, kde leží toto sousloví pohromadě a v přesném tvaru, ale i jednotlivá slova nezávisle na sobě v různých částech stránky.

Jako základ nám poslouží 4 skupiny údajů, které pokrývají různé vlastnosti dat:

živočišné druhy, CZ jako skupiny slouží jednotlivé třídy podkmene obratlovců (ryby, savci, ptáci, hmyz a obojživelníci) a jako položky pak zástupci těchto tříd. Pro každou skupinu bylo vybráno 5 zástupců, kdy při výběru byla snaha vybrat obecně používaná slova (kočka, jelen) a byla použita jejich rodová i druhová jména v českém jazyce v 1. pádě. Cílem této skupiny je zjistit chování funkcí při víceslovných výrazech, které se vyskytují i na stránkách, které nejsou přímo o nich. V DB označeno kódem *Obratlovci_CS*.

živočišné druhy, LA jde o stejné skupiny i položky údajů jako v předchozím případě, pouze jsou všechny názvy uvedeny v latině, opět jsou použita rodová i druhová jména. U této skupiny by nemělo docházet k výskytům na stránkách, které nejsou primárně o nich, jelikož se jedná o odborné výrazy. Cílem je tedy ověřit vlastnosti funkcí při specializovaných výrazech. V tabulkách rozlišeno typem *Obratlovci_LA*.

státy, CZ zde nám jako skupiny poslouží světadíly (Evropa, Amerika, Asie, Afrika, Austrálie a Antarktida) a jako položky vybrané státy ležící na území těchto světadílů. Opět bylo pro většinu světadílů vybráno 5 prvků, pouze pro Austrálii jenom 4 a pro Antarktidu žádný (tato skupina tedy slouží pouze v rámci skupin a nemá žádnou vazbu k jednotlivým zástupcům). Pro kontinenty i pro státy jsou použity jednoslovné běžné české názvy v 1. pádě. Budeme tedy zkoumat chování při velmi obecných jednoslovných výrazech, které se jak v případě skupin tak i záznamů vyskytují na mnoha stránkách pojednávajících o jiném tématu. Tato data jsou odlišena kódem *Staty_CS*.

státy, EN varianta předchozího, ale s údaji v angličtině. Díky použití na internetu nejrozšířenějšího jazyka budou tato častá slova obsažena ve velkém množství stránek a ve velké většině stránky nebudou souviset s naším hledaným tématem. Z hlediska množství výskytu výrazu by mělo jít o extrém a opak proti prvkům ze skupiny latinských živočišných druhů. Poznávacím znakem těchto dat je kód *Staty_EN*.

Všechna použitá data jsou uvedena v příloze, kde jsou i skripty k jejich naplnění do tabulek, v tabulce 4.1 si však uvedeme seznam skupin výrazů.

Tabulka 4.1: Seznam skupin výrazů

C01ID	C01TYPE	C01NAME
1	Obratlovci_CS	Ryby
2	Obratlovci_CS	Ptaci
3	Obratlovci_CS	Obojzivelnici
4	Obratlovci_CS	Hmyz
5	Obratlovci_CS	Savci
6	Obratlovci_LA	Osteichthyes
7	Obratlovci_LA	Aves
8	Obratlovci_LA	Amphibia
9	Obratlovci_LA	Insecta
10	Obratlovci_LA	Mammalia
11	Staty_CS	Evropa
12	Staty_CS	Asie
13	Staty_CS	Afrika
14	Staty_CS	Australie
15	Staty_CS	Amerika
16	Staty_CS	Antarktida
17	Staty_EN	Europe
18	Staty_EN	Asia
19	Staty_EN	Africa
20	Staty_EN	Australia
21	Staty_EN	America
22	Staty_EN	Antarctica

Zde si dovolíme malou odbočku. Jistě by bylo velmi zajímavé zkoumat výsledky pro data týkající se osob, ale v takovém případě narážíme na několik problémů. Když vynecháme složité shromažďování i uskladňování dat a použijeme pouze volné údaje (chráněné údaje se stejně nesmí nikde volně na internetu vyskytovat, takže se nejedná o velký ústupek), narazíme na to, že jméno jako hlavní identifikátor je velmi nejednoznačný a pravděpodobně narazíme na stránky, které se týkají jiného člověka. Z tohoto pohledu je možné uvažovat pouze o lidech, kteří jsou natolik známí, že se budou vyskytovat na dostatečném počtu stránek. Využití si lze představit například pokud by prvky byly tvořeny jmény členů vlády nebo některé sportovní reprezentace. Na zjišťování dalších údajů třeba o klientech firmy to v obecném případě pravděpodobně fungovat nebude.

4.1.3 Požadavky

Očekávané funkční výsledky budou uvedeny u popisu jednotlivých funkcí. Zde si sepíšeme obecná očekávání a požadavky, která budou omezovat naše možnosti.

Všechny funkce musí být napsány tak, aby je šlo volat přímo v SQL dotazech. Jinými slovy to znamená, že půjde o PL/SQL funkce a nikoliv procedury, které sice nabízejí větší možnosti co se týče především manipulace s daty v tabulkách, ale nelze je volat v rámci dotazu.

Z hlediska optimalizace budeme řešit časovou náročnost pouze z pohledu počtu volání internetového vyhledávače a java procedur, ale nikoliv na úrovni technické (načítání webové stránky, její procházení a podobně). Na takováto "slabá" místa pouze upozorníme v kódu, ale nebudeme hledat ideální variantu. Přesto je cílem, aby funkce doběhly v reálném čase, ale nelze počítat s tím, že tento čas bude odpovídat standardní době trvání vyhodnocení SQL dotazu. Obecně je jejich použití mířeno k analýze údajů, nikoliv k aplikacím a dotazům, kdy se očekává rychlá odpověď.

4.2 Funkce WWW_rank

4.2.1 Popis funkce a příklady

Cílem této funkce je přiřadit k výrazu jeho důležitost podle internetu tak, že nižší číslo značí méně důležitý výraz a vyšší číslo důležitější výraz. Už z tohoto zadání lze poznat, že je velmi obecné a v podstatě nelze určit, které výsledky jsou správné a které jsou špatné. Podstatnou vlastností této funkce je to, že nás nezajímá přímo hodnota výsledku, ale pouze porovnání výsledku pro jeden výraz s výsledkem pro druhý výraz, tedy uspořádání jednotlivých hodnot. Z tohoto plyne první funkční požadavek, tedy že budeme požadovat, aby výsledek pro všechny výrazy patřil do intervalu (0,1). Tím se nám sice zhorší přehlednost, ale zase máme zaručenu omezenost výsledku a jednoduchost případného dalšího zpracování.

Její využití je pak především mířeno do klauzule ORDER BY, kde bude sloužit k tomu, aby důležité údaje byly zobrazovány v první části výsledkové sady našeho dotazu. U našich dat ji tedy můžeme využít k tomu, že budeme potřebovat zjistit, který ze savců je nejznámější a nejtypičtější zástupce této třídy. Dotaz 13 naznačuje využití funkce www_rank pro získání nejdůležitějšího zástupce ke každé skupině. Obdobně by šlo získat i nejpodstatnějšího zástupce celkem nebo si prostě jenom záznamy podle výsledků www_rank uspořádat.

Zdrojový kód 13 Nejdůležitější zástupce ke každé skupině

```
SELECT C01NAME, C02NAME
FROM T01GROUP INNER JOIN
     T03GROUP_ITEM ON C01ID = C03C01_1 INNER JOIN
     T02NAME G ON C03C02_1 = G.C02ID
WHERE NOT EXISTS (SELECT 'x'
                  FROM T02NAME L
                  WHERE www_rank(G.C02NAME) <= www_rank(L.C02NAME)
                    AND G.C02ID <> L.C02ID
                    AND EXISTS (SELECT 'x'
                                FROM T03GROUP_ITEM
                                WHERE C03C02_1 = L.C02ID
                                  AND C03C01_1 = C01ID));
```

Proti dalším dvěma funkcím zde budeme dávat menším důraz na přesnost výsledků, ale větší na rychlost zpracování.

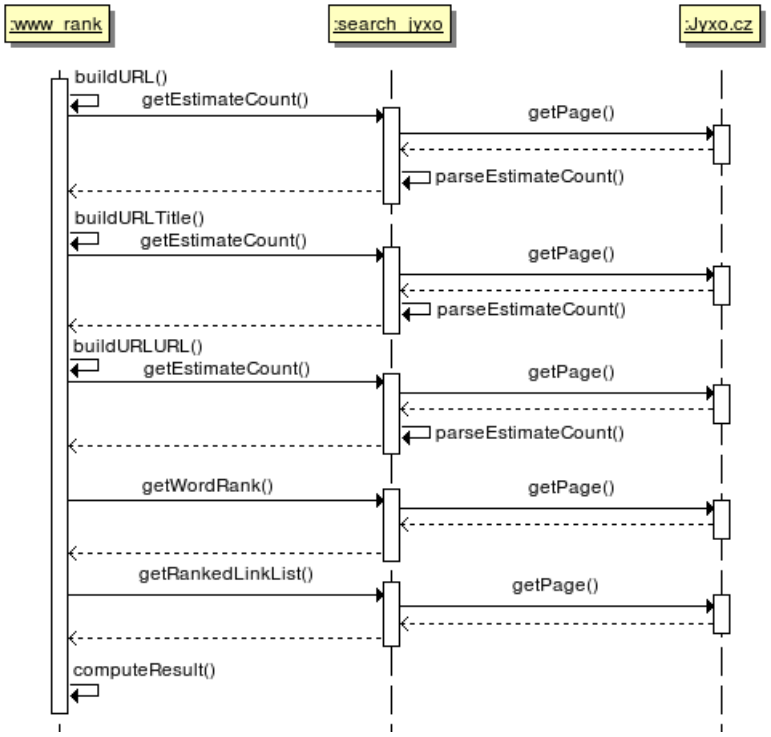
4.2.2 Návrh řešení

Základem pro získání výsledku nám bude počet stránek, které vyhledávač vrátí po dotazu na náš výraz. Tuto hodnotu lze přesně získat pouze skutečným nalezením všech odkazů, ale už na první stránce s výsledky je zobrazen odhadovaný počet výsledků, který je pro náš účel dostatečný.

Další možností jsou různé varianty dotazu. Můžeme výraz hledat v obsahu stránky, v jejím titulu nebo jenom v její adrese. Každý z těchto dotazů zkoumá jiné vlastnosti výrazů a v ideálním případě by měl být celkový výsledek složen ze všech dílčích výsledků, ale už na první pohled je vidět, že počet stránek obsahujících hledaný výraz v adrese by neměl mít velkou váhu, jelikož url většinou vzniká buď nezávisle na obsahu nebo jde o zkratky z názvů. V naprosté většině případů však adresa nebude obsahovat přímo hledaný výraz, především pak u delších slov. V případě titulků a obsahu stránek už vyhledávání smysl dává.

Jiná úprava by mohla spočívat v tom, že nebudeme počítat pouze počet získaných odkazů, ale každý odkaz bude ohodnocen svojí relevancí, což typicky vyhledávač provádí a podle této relevance odkazy třídí na výsledkové stránce. Do celkového výsledku by se započítal například součet všech ohodnocení nebo součet ohodnocení prvních 100 výrazů. Spočívá zde riziko v tom, že pro některé způsoby ohodnocování stránek vyhledávačem existuje způsob, jak zvýšit ohodnocení některé stránky, ale takové situace bývají spíše

Obrázek 4.2: Extrémní varianta funkce www_rank



vyjímečné a ve většině případů by výsledky neovlivnily.

Zajímavým číslem by mohlo být i jak často je výraz zadáván do vyhledávače, tedy kolik uživatelů zvolené slovo nebo sousloví hledá. Tím bychom zkoumali hlavně aktuální zajímavost tématu a jeho popularitu. Vedlejší efektem by pak byla větší variabilita výsledků v čase, protože internetové vyhledávače jsou používány výrazně častěji, než jsou vytvářeny nové stránky.

Pokud nakonec vezmeme všechna tato čísla, přidělíme jim různé váhy, tak můžeme spočítat vážený průměr a ten vzít za důležitost výrazu. Sekvenční diagram takovéto časově náročné varianty znázorňuje obrázek 4.2. Je zde extrémně mnoho volání internetového vyhledávače (každá funkce kromě závěrečného počítání výsledku ho v sobě obsahuje). Otázkou zůstává, jestli výsledek získaný z maxima možných informací vyváží časovou náročnost.

4.2.3 Technická realizace

Extrémní varianta uvedená v minulé kapitole je sice ideální z hlediska množství informací započítaných do výsledné hodnoty, ale je z mnoha důvodů nereálná. Nepraktičnost vyhledávání výrazu v adrese stránky už jsme osvětlili a především z důvodu, že pro většinu výrazů vrací nulu, ho vynecháme.

Jiné problémy nastávají, pokud bychom chtěli použít ohodnocení jednotlivých vrácených odkazů. Především tuto hodnotu náš vybraný vyhledávač Jyxo.cz na stránce s výsledky neuvádí. Pokud bychom chtěli tuto hodnotu využívat, bylo by vhodnější využít vyhledávač Morfeo, který uvádí kvalitu odkazu u každého výsledku. Navíc odkrývá i postup, jakým ji počítá, takže ji jde lépe využít. Ani v případě, kdy by byla tato informace přístupná, by ovšem nebylo její využití příliš vhodné. Uvedeme si příklad. Pokud do zmíněného vyhledávače Morfeo vložíme názvy světadílů z našich testovacích dat (skupina Staty_CS), mají všechny první odkazy velmi podobné hodnoty relevance dokumentu. Druhou nevýhodou je, že pokud chceme získat nějaké relevantní údaje a tedy zjišťovat důležitost u více odkazů, bude mít stahovaná stránka větší velikost a odezva bude znatelně větší. To je při podobných hodnotách prvních odkazů zbytečné a i od této varianty prozatím upustíme a relevanci odkazů nebudeme brát v potaz.

Třetí možností, která se ukáže jako nevhodná k použití, je získávání častosti použití slova ve vyhledávači. Zde leží kámen úrazu v získávání dat. Částečně by šlo využít například seznam uveřejněný na Jyxo.cz na stránce http://jyxo.cz/top/ , kde je uveřejněno 5000 nejčastějších dotazů za rok 2006, nebo serveru Yahoo! Buzz, který uveřejňuje informace průběžně, ale pouze 20 nejhledanějších výrazů. Údaje získané z Jyxo.cz se dají využít staticky, tedy načteny do tabulky a jejich využití by nezabralo téměř žádný čas. Ale ztratí se tím proměnlivost v čase, kterou všechny ostatní hodnoty získané z vyhledávače mají a je to jeden z rozdílů jeho využití proti DB. Naopak údaje z Yahoo! Buzz jsou aktuální, ale je jich jen velmi málo a navíc se většinou jedná o jména osob, tedy informace o našich výrazech se z tohoto zdroje nedozvíme. Zde je opět velký problém v přístupu k vyhledávači přes jeho www rozhraní, při lepší integraci do DB nebo do javy (respektive jiného jazyka přístupného z databáze), by údaje o častosti vyhledávání zvoleného výrazu byli dobrým kandidátem do zapracování do funkce www_rank, jeho statickou podobu tam však zapracovávat nebudeme.

Nakonec nám tedy zbyly pouze údaje o množství stránek, kde se hledaný výraz vyskytuje v obsahu stránky, a o množství stránek, kde je výraz obsažen v titulku. Tyto údaje budeme zjišťovat pomocí jednoduchého dotazu, kdy využijeme toho, že lze pomocí adresy nastavit i počet výsledků zobrazených na jedné stránce a místo standardního počtu odkazů upravíme adresu tak, že stránka bude obsahovat pouze jeden odkaz, což je minimum. Stránku bez jediného odkazu nelze získat, takže takto maximálně ušetříme přenášená data a přitom nepřijdeme o žádné informace, jelikož celkový odhadovaný počet odkazů je uveden vždy. Dohromady složená adresa pro zjištění počtu stránek, které v obsahu mají slova "vlk" a "obecny" vypadá následovně

```
http://jyxo.cz/s?q=+vlk+obecny&cnt=1
```

Každé slovo v sousloví je uvozeno znakem plus, který značí, že výskyt slova na stránce je povinný a nikoliv jen volitelný. Na závěr adresy je právě podmínka, že na stránce má být zobrazen maximálně jeden výsledek. Obdobně vypadá i adresa pokud stejný výraz hledáme v titulku stránky

```
http://jyxo.cz/s?q=title%3Avlk+title%3Aobecny
```

Na oba tyto údaje nám stačí jedna java procedura, která se jmenuje *getEstimateCount* a je obsažená v třídě *search_jyxo.* Tato procedura zajistí načtení stránky a z ní poté vyčte údaj o počtu stránek. Jako parametr jí slouží upřesnění dotazu, tedy část ve které se liší jednotlivé adresy a která následuje za otazníkem, tedy v našich ukázkách buď "q=+vlk+obecny&cnt=1" nebo "q=title%3Avlk+title%3Aobecny". Tady porušujeme naši snahu o minimalizaci počtu volání javy z DB, protože pokud budeme chtít využít obě čísla, musíme proceduru volat dvakrát, ale výsledkem je přístupnost obou variant přímo z DB a díky tomu můžeme tyto dva údaje porovnat. Tabulka 4.2 porovnává množství stránek, které obsahují položky v tabulce T01GROUP. V prvním sloupci po identifikaci údajů je uvedeno množství stránek, které obsahují název skupiny v obsahu a ve druhém počet pro stejný výraz a titulek. Pak následují dva sloupce pro průměr počtu stránek všech položek dané skupiny. Poslední čtyři svým obsahem kopírují předchozí, pouze místo hodnoty je v nich uvedeno pořadí skupiny v rámci typu záznamů.

Z tabulky je vidět, že výsledky pro obsah i titulek jsou velmi podobné. Přestože mají všechny typy záznamů minimálně pět skupin, rozdíl v pořadí pro obsah a pro titulek není ani v případě přímého pořadí ani u pořadí průměru záznamů nikdy větší než dva. Větší rozdíly jsou mezi pořadím pro vlastní název skupiny proti pořadí podle průměrů položek, ale i zde dopadá hledání v obsahu i v titulku velmi podobně. Mnohem více záleží na výběru výrazů, než způsobu položení dotazu. Na základě tohoto závěru vytvoříme naši verzi funkce www_rank tak, že její výsledek budeme usuzovat pouze z počtu stránek obsahujících hledané sousloví nebo z počtu stránek majících ho v nadpise, ale nikoliv z obou čísel zároveň. Každou z možností má svoje výhody i nevýhody. Prohledávání titulků dosahuje znatelně menších čísel a proto zde může rozhodovat každá stránka, kdežto u obsahu se jedná o hodnoty řádově vyšší a tedy podložené větším množstvím stránek. Naopak výhodou titulků je, že se výrazně snižuje šance, že stránka svým obsahem příliš nesouvisí s hledaným výrazem a ten se zde objevuje jen jaksi mimochodem. Jak jsme se přesvědčili, tak výsledky jsou stejně velmi podobné, takže na konečném rozhodnutí příliš nezáleží, ale my si vybereme variantu, kde se dotaz pokládá pouze nad titulky stránek.

Tabulka 4.2: Porovnání počtu stránek skupiny

C01ID	C01TYPE	C01NAME	Počet stránek - obsah	Počet stránek - titulek	Průměr počtu stránek - obsah	Průměr počtu stránek - titulek	Pořadí - obsah	Pořadí - titulek	Pořadí - obsah, průměr	Pořadí - titulek, průměr
1	Obratlovci_CS	Ryby	673303	18008	3810	26	1	1	1	2
2	Obratlovci_CS	Ptaci	462464	12939	784	22	2	2	4	4
3	Obratlovci_CS	Obojzivelnici	37901	457	945	25	5	5	3	3
4	Obratlovci_CS	Hmyz	181222	2798	482	12	3	3	5	5
5	Obratlovci_CS	Savci	105698	1062	2158	60	4	4	2	1
6	Obratlovci_LA	Osteichthyes	843	14	422	9	5	4	2	4
7	Obratlovci_LA	Aves	7131	277	253	10	2	1	4	3
8	Obratlovci_LA	Amphibia	2461	36	297	107	4	2	3	1
9	Obratlovci_LA	Insecta	17508	19	73	3	1	3	5	5
10	Obratlovci_LA	Mammalia	2476	10	711	102	3	5	1	2
11	Staty_CS	Evropa	1349857	45356	485110	7496	1	1	1	1
12	Staty_CS	Asie	473906	5581	240302	3209	5	5	4	5
13	Staty_CS	Afrika	508529	13798	197115	4202	4	3	5	3
14	Staty_CS	Australie	672803	8889	319503	3239	2	4	3	4
15	Staty_CS	Amerika	556900	15064	368585	4233	3	2	2	2
16	Staty_CS	Antarktida	105725	492	0	0	6	6	6	6
17	Staty_EN	Europe	680033	16147	204671	1660	1	1	1	3
18	Staty_EN	Asia	268721	2225	130569	1677	3	5	4	2
19	Staty_EN	Africa	191645	2466	55445	187	5	4	5	5
20	Staty_EN	Australia	241745	4826	163724	1638	4	3	3	4
21	Staty_EN	America	314255	7194	194636	2381	2	2	2	1
22	Staty_EN	Antarctica	27999	92	0	0	6	6	6	6

Posledním úkolem, který nás čeká je normalizace hodnoty výsledku do intervalu <0,1). Funkce, kterou hledáme, musí být deterministická, rostoucí, pro hodnotu 0 musí vždy vrátit 0 a musí rozlišovat přirozená čísla z intervalu <1,100000>. Pro vlastní převod do intervalu mezi nulou a jedničkou použijeme k podobnému účelu velmi často užívanou funkci

$$f(x) = 0, pro\, x = 0$$

$$f(x) = 1 - \exp\left(-\frac{x}{k}\right), pro\, x \geq 1$$

Ještě zbývá najít vhodný koeficient k, abychom dostávali rozdílné výsledky na našem oboru hodnot. Jako vhodný se ukazuje koeficient 10000, jehož vliv demonstruje následující tabulka

x\k	1	10000
1	0.632120559	0.000099995
2	0.864664717	0.000199980
90000	1.000000000	0.999876590
99999	1.000000000	0.999954596
100000	1.000000000	0.999954600

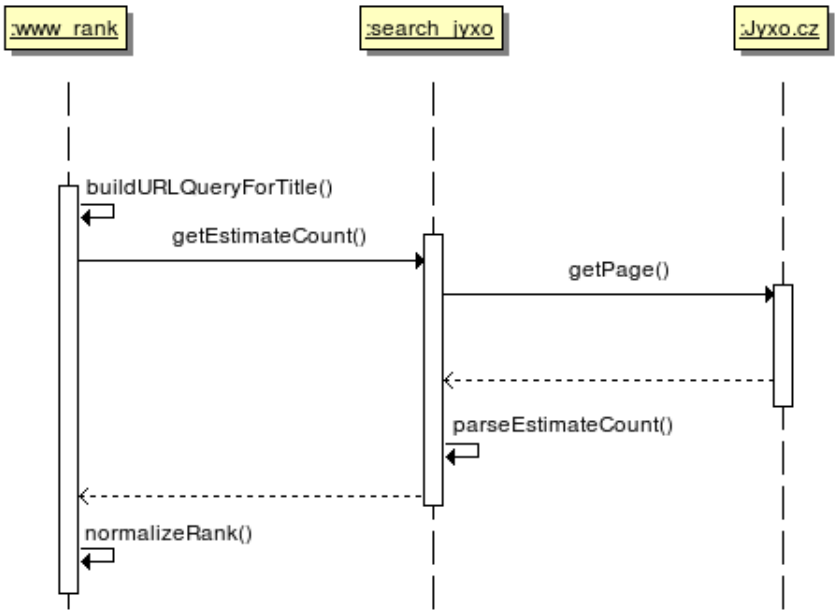
V prvním sloupci jsou uvedeny vstupní hodnoty funkce, v druhém sloupci jsou výsledky pro k=1 a v druhém pro k=10000. Dosáhli jsme tedy toho, že $f(2) - f(1) \approx f(100000) - f(90000) \approx 0.0001$, což nám umožní velmi dobře rozlišit výsledky na obou stranách oboru hodnot.

4.2.4 Výsledky

Pro realizaci funkce *www_rank* vznikla javovská třída *search_jyxo*. Obsahuje procedury *getResultPage*, která vrací stránku, jejíž adresu získá jako vstupní parametr, *parseEstimateCount*, která ze stránky s výsledky načte odhadovaný počet celkově nalezených výsledků, a *getEstimateCount*, která vrací odhadovaný počet nalezených odkazů pro dotaz za pomoci předchozí procedury. Na straně databáze máme také 3 různé podprogramy. První se jmenuje *get_est_count* a pouze zapouzdřuje volání *search_jyxo.getEstimateCount* pro volání z DB. Druhá je *getTitleEstimateCount* a jejím cílem je složit parametry dotazu tak, aby byl výraz, který dostane jako parametr, vyhledáván v titulku stránek. Třetí a poslední je *www_rank*, která k volání *getTitleEstimateCount* přidává ještě normalizaci hodnoty výsledku. Vzájemné volání jednotlivých procedur je zobrazeno na sekvenčním diagramu 4.3.

Z hlediska časové optimalizace se nám povedlo minimalizovat počet volání mezi DB a javou na jedno a stejně tak počet volání internetového vyhledávače

Obrázek 4.3: Sekvenční diagram www_rank



Jyxo.cz z javy na jedno volání. Při testech na osobním počítači a pevném, nikoliv vytáčeném, připojení k internetu byla doba odpovědi ve vteřinách přibližně rovna počtu záznamů vrácených dotazem. Tedy vteřina na každé volání funkce *www_rank*, což se dá považovat za pozitivní překvapení, jelikož toto volání v sobě ukrývá načtení celé internetové stránky. Zde je potřeba zdůraznit, že znatelného zrychlení v práci funkce *www_rank* bylo dosáhnuto použitím klauzule "PARALLEL_ENABLE" v definici všech použitých DB funkcí.

Výsledky k jednotlivým záznamům lze nalézt na přiloženém CD v adresáři RESULTS v souboru www_rank. Zde pouze uvedeme, že celkovým vítězem (výrazem s největší hodnotou www_rank) se stal v kategorii skupin výraz "Evropa", která je uvedena v titulku 45 356 stránek a tím pádem dosáhla hodnoty www_rank rovné 0.98928, a v položkách výraz "Nemecko", 13 488 stránek a www_rank 0.74045. Nejznámějším živočichem je, bráno dohromady v českých i latinských názvech, "Canis lupus" neboli vlk obecný, který je ve své latinské podobě uveden v titulku 493 stránek a dosáhl tedy hodnoty 0.04810.

4.3 Funkce WWW_near

4.3.1 Popis funkce a příklad

Cílem funkce www_near je rozhodnout, jak mnoho jsou si dva výrazy podobné z hlediska míst, kde se vyskytují. Hodnota 0 by měla nastávat pro slova, která spolu nijak nesouvisí, což by mělo znamenat, že neexistuje stránka, kde se vyskytují naráz obě slova. Naopak vysoký výsledek nastane, pokud jde o slova vyskytující se vždy spolu, případně, pokud dostane funkce jako oba své parametry stejný výraz.

Nejde tedy o vyhledávání synonym, ale o nacházení souvislostí. V našem případě budeme zjišťovat, do které třídy patří který živočich nebo na kterém kontinentu leží který stát. Ovšem pro synonyma nebo překlady by funkce ve výsledku mohla fungovat též, ale primárním cílem zůstává korektní naplnění tabulky T03GROUP_ITEM pomocí dotazu popsaného v dotazu 14.

Zdrojový kód 14 Použití funkce www_near

```
SELECT C01NAME, C02NAME
FROM T02ITEM INNER JOIN
     T01GROUP ON C02TYPE = C01TYPE
WHERE www_near(C01NAME, C02NAME) =
      (SELECT MAX(www_near(C01NAME, C02NAME))
       FROM T01GROUP gr
       WHERE gr.C01TYPE = C02TYPE)
ORDER BY C02ID;
```

Časová náročnost nás u této funkce příliš trápit nebude, mnohem důležitější je správná funkčnost. Pokud se povede, aby dotaz 14 vracel ve všech testovacích případech správný výsledek, může běžet i řádově desítky minut, ale samozřejmě je hned druhým cílem, aby volání www_near bylo co nejrychlejší.

Funkce, která by přesněji odpovídala funkčnosti, kterou hledáme, by vypadala malinko jinak. Měla by dva parametry, kdy první by byl stejný jako doposud, tedy libovolný textový výraz. Druhým parametrem by byl seznam výrazů (speciální typ, tabulka, kurzor) a výsledkem funkce by byl nejpříbuznější výraz z tohoto seznamu vzhledem k prvnímu parametru. Takovéto řešení by mělo jednu podstatnou výhodu, mělo by zároveň přístup ke všem porovnávaným výrazům. Tím by šlo docílit lepší výsledků, než za našeho stavu, kdy počítáme vztah pro každou dvojici výrazů zvlášť. Navíc by se tím daly dále zjednodušit SQL dotazy, jak ukazuje dotaz 15. Ztrátou by pak

byla nemožnost jiného použití funkce, například by se velmi těžko zařazovala do ORDER BY klauzule nebo hledání přes více hodnot pro první parametr by také bylo velmi komplikované. Právě pro tuto přehnanou specializaci se nebudeme variantě pracovně nazvané *www_nearest* věnovat a zůstaneme u původní varianty *www_near*.

Zdrojový kód 15 Funkce www_nearest, alternativní funkce k funkci www_near

```
SELECT C02NAME,
       www_nearest(C02NAME, (SELECT C01NAME
                             FROM T01GROUP
                             WHERE C01TYPE = C02TYPE))
FROM T02ITEM;
```

4.3.2 Návrh řešení

Teoreticky existuje mnoho možností, jak takovouto funkci počítat. Základní otázka zní, jestli má být symetrická nebo nikoliv. Pokud bude našim cílem pouze spojování dvojic číselník-položka, je tato vazba asymetrická a tedy i chování funkce k parametrům by mohlo býti různé. Pokud ovšem přidáme zmiňované hledání spojení mezi českým a latinským názvem živočichů, ztrácí nestejný přístup smysl. S ohledem na obecnost a také uživatele, který se díky tomu nemusí starat o pořadí atributů, se nejdříve pokusíme vytvořit funkci symetrickou a pokud tato cesta selže, budeme pokračovat asymetrickou. Případně lze definovat při nalezení asymetrické varianty symetrickou takto:

$$www\_near_{sym}(a,b) = \frac{www\_near_{asym}(a,b) + www\_near_{asym}(b,a)}{2}$$

nebo

$$www\_near_{sym}(a,b) = \max\left(www\_near_{asym}(a,b), www\_near_{asym}(b,a)\right)$$

Nejjednodušší možností realizace www_near je její vypočítávání z počtu stránek, které zároveň obsahují oba výrazy. Řídili bychom se v tomto případě heslem "v jednoduchosti je síla" a šlo by o velmi rychlé řešení a navíc ho prohození výrazů nijak neovlivní. Otázkou zůstává efektivita, ale tu lze ověřit až praktickými pokusy. Každopádně je toto číslo vážným kandidátem, aby se v nějaké formě v celkovém výsledku projevilo.

Obměnou je varianta s použitím vyhledávání v titulku stránky. Zde se nezdá příliš smysluplné vyhledávat zároveň v nadpisu oba výrazy. Lepších

výsledků pravděpodobně dosáhneme, pokud budeme jeden výraz hledat v titulku, druhý v obsahu a podruhé obráceně. Zajímavou otázkou je, jestli se větší hodnoty dosáhne, když budeme v nadpise hledat číselník a v obsahu položku nebo naopak.

Prohledávání adresy stránky na obsah hledaného výrazu nebývá příliš efektivní, jak bylo objasněno již u funkce www_rank. Zde však je šance, že když se podaří jeden z výrazů v adrese najít, tak druhý bude obsažen buď v titulku nebo v obsahu stránky a takto nalezený odkaz bude mít velkou váhu. Jelikož nám v tomto případě výrazně méně záleží na čase, je vhodné tuto cestu zachovat.

Další variantou by mohlo být použití pozičních operátorů, lépe řečeno vzdálenostních. Například určení, že mezi prvním a druhým výrazem může být na stránce maximálně 10 slov. Pravděpodobně bychom takto získali lepší výsledky pro hledání vztahů číselník-položka než pro synonyma. Je zde velké riziko neúměrného časového prodloužení při malé přidané hodnotě proti obyčejnému prohledávání.

Jistou obměnou všech dosavadních postupů by bylo, nepočítat přímo s množstvím stránek obsahujících oba výrazy, ale toto číslo nejdříve vydělit důležitostí jednotlivých výrazů. Obzvláště, pokud by bylo našim cílem měřit provázanost dvojice výrazů jako veličinu (například porovnávat, jestli je více příbuzná dvojice "vlk obecný"-"ryby" nebo dvojice "Nemecko"-"Asie"), nikoliv pouhé hledání nejpříbuznějšího slova ke zvolenému. V našem případě, kdy v podstatě pevně zvolíme položku a k ní hledáme správný číselník, ztrácí oddělené četnosti na významu, jelikož hodnota pro položku je konstantní (stále stejná položka) a pro číselníky nebudou příliš velké rozdíly, jelikož se jedná o výrazy ze stejné kategorie. Použití této modifikace by přivedlo do úvahy dalších 6 čísel (dvě hodnoty pro vyhledávání v adrese, dvě pro titulek a dvě pro obsah), ale prozatím ji odložíme a použijeme, až pokud ostatní způsoby nepovedou ke správným výsledkům.

Nejpravděpodobnější cestou k dobrému výsledku je kombinování všech předchozích variant, například jeden výraz vyhledávat v adrese stránky a druhý v titulku nebo v obsahu. Těmito kombinacemi získáme v základní variantě 9 čísel (obsah-obsah, titulek-titulek, adresa-adresa, dvakrát obsah-titulek, dvakrát obsah-adresa a dvakrát titulek-adresa, tedy bez použití hodnot pro samostatné výrazy a hodnot při určení vzdálenosti mezi slovy). Jednotlivým takovýmto mezivýsledkům přiřadíme různé koeficienty a sečteme. Při správném nastavení koeficientů by nám vážený součet měl postačit ke správnému uspořádání souvislostí mezi výrazy.

4.3.3 Technická realizace

Prvním technickým problémem, který v tomto případě potkáme je skutečnost, že vyhledávače nepodporují vzdálenostní operátory. Tedy nelze jim říci, že hledaná slova mají být od sebe maximálně vzdálena x slov. Bohužel se tuto vlastnost nepovedlo najít u žádného vyhledávače. Přijdeme tím o možnost regulovat vzdálenost výrazů, ale zároveň ušetříme problémy s hledáním optimální hodnoty.

Zůstává nám tedy devět možností, jak položit dotaz. První je nejjednodušší a budeme v ní hledat stránky, ve kterých budou oba výrazy ležet v obsahu. Další je hledání obou výrazů v titulku stránky, což vede k dalším dvěma, tedy první výraz hledaný v titulku a druhý v obsahu a naopak. Zbylých pět je hledání obou v adrese stránky a zbylé kombinace, dvakrát obsah s adresou a dvakrát titulek s adresou.

Dopředu nelze příliš říci, které z těchto hodnot budou potřeba pro získání správných výsledků a které budou mít jak vysoký koeficient. Lze pouze odhadnout, že největší váhu by měly mít hodnoty spojené s adresou stránky, především pokud se povede nalézt stránku, která bude mít v adrese oba dva hledané výrazy, svědčí to o velmi těsném vztahu. Další v důležitosti budou pravděpodobně ostatní dotazy obsahující v sobě hledání v adrese, tedy kombinace adresy s obsahem a adresy s titulkem, a dotaz na oba výrazy obsažené v titulku. Nejnižší váha tím pádem zůstává spojení titulku s obsahem a úplně nejnižší pro dotaz ohledně pouze obsahu. Tímto dotazem budeme zachraňovat situace, kdy všechny ostatní dotazy vrátí nula záznamů.

Prvním krokem, který provedeme, bude, že si pro všechny dvojice číselník-položka, které patří do stejného typu zjistíme všechny nastíněné hodnoty. Dvojice položka a číselník, kdy každý náleží do jiného typu vynecháme čistě pro ušetření, jelikož lze důvodně očekávat, že pokud budou hodnoty odpovídat v rámci typů, nemělo by se nám to s výrazně tématicky odlišnými výrazy pokazit. Jelikož je každé vyhledávání časově náročné, je efektivní si tyto hodnoty uložit do speciální tabulky a prozatím využít je, než se neustále ptát vyhledávače na stejné dotazy. My k tomuto účelu využijeme tabulku T04NEAR, která bude obsahovat pouze odkazy do tabulek T01GROUP a T02ITEM a poté 9 sloupců, do kterých jsme si uložili všechny počty stránek nalezených ke každé dvojici. Poslední sloupec této tabulky nazveme C04NEAR a budeme si do něho průběžně ukládat hodnoty funkce www_near pro každou dvojici, tím si zjednoduší psaní dotazů při testování různých koeficientů. Nyní můžeme testovat koeficienty a další způsoby kombinování bez opětovného volání internetového vyhledávače.

Využijeme toho, že máme korektně naplněnou tabulku T03GROUP_ITEM a můžeme lehce porovnávat získané výsledky s těmi očekávanými. Měřítkem

naší úspěšnosti nám bude dotaz popsaný v 16, tedy seznam položek, ke kterým funkce www_rank nevybrala správně nejvíce odpovídající číselník. Ideálního stavu dosáhneme, až nám tento dotaz nevrátí žádný záznam.

Zdrojový kód 16 Dotaz pro měření úspěšnosti www_near

```
SELECT C02NAME,
       C01NAME
FROM T04NEAR INNER JOIN
     T01GROUP ON C04C01_1 = C01ID INNER JOIN
      T02ITEM ON C04C02_1 = C02ID LEFT JOIN
      T03GROUP_ITEM ON C03C01_1 = C01ID
                   AND C03C02_1 = C02ID
WHERE NOT EXISTS (SELECT 'X'
                  FROM T04NEAR S
                  WHERE S.C04C02_1 = T04NEAR.C04C02_1
                    AND S.C04NEAR > T04NEAR.C04NEAR)
   AND C03C01_1 IS NULL
ORDER BY T04NEAR.C04C02_1;
```

První, vyzkoušíme koeficienty podle úvahy nastíněné o dva odstavce dříve. V řeči čísel to odpovídá následujícímu schématickému vzorci

$$\begin{aligned} www\_near\,(v1,v2) \;=\; & \frac{1}{1000000}CC\,(v1,v2)+\frac{1}{1000}CT\,(v1,v2)+\frac{1}{1000}CT\,(v2,v1)+\\ & \frac{1}{1000}TT\,(v1,v2)+\frac{1}{1000}UT\,(v1,v2)+\frac{1}{1000}UT\,(v2,v1)+\\ & \frac{1}{1000}CU\,(v1,v2)+\frac{1}{1000}CU\,(v2,v1)+UU\,(v1,v2) \end{aligned}$$

, kde názvy funkcí označují prostor hledání prvního nebo druhého parametru. C je pro hledání v textu, T pro titulek a U pro adresu. CC je tedy funkce, která vrací počet stránek, kde první i druhý argument jsou obsaženy v textu. Podobně UT znamená, že první parametr se hledá v adrese a druhý v titulku.

Pro tyto hodnoty koeficientů vrací testovací dotaz 4 záznamy, což na první pokus jistě není špatné. Jmenovitě se nepovede správně přiřadit položky "Cervotoc spizni", "Nicrophorus vespillo", "Kocka domaci" a "Kazakhstan", tedy výrazy tří různých typů. Při bližším pohledu na zmíněné čtyři případy nás jistě napadne vyzkoušet zjednodušenou variantu

$$\begin{aligned} www\_near\,(v1,v2) \;=\; & \frac{1}{1000000}CC\,(v1,v2)+\frac{1}{1000}CU\,(v1,v2)+\\ & \frac{1}{1000}CU\,(v2,v1)+UU\,(v1,v2) \end{aligned}$$

Změna spočívá v tom, že vypustíme všechny hodnoty, které v sobě zahrnují prohledávání titulku stránky. Ušetříme tedy dohromady 5 volání internetu na každé volání funkce www_rank a přitom získané výsledky jsou ještě lepší. Ověřovací dotaz už vrací pouze 3 záznamy, když ubyla "Kocka domaci". Lepších výsledků pravděpodobně už dosáhnout nelze. V případě červotoče i latinského hrobaříka spočívá celý problém vždy v jedné stránce, která obsahuje špatnou kombinaci a jelikož neexistují "správné" stránky na stejné úrovni důležitosti, tato chyba se stane osudnou a převaha dobrých stránek typu CC je nedostatečná. Jde tedy o problém nedostatku dat, kdy špatná data příliš vyniknou. Lze říci, že jedna správně vytvořená stránka by mohla oba tyto případy zachránit, ale u podobně odborných výrazů jako v této práci používáme bude k podobných nepřesnostem docházet. Pokud zkusíme vyřešit tyto okrajové nedostatky tak, že pokud funkce CU a UC nabudou hodnoty 1, tak tuto jednu stránku budeme ignorovat, vyskytnou se problémy na jiném místě a bude špatně přiřazený číselník pro jiné dvě položky.

U Kazachstánu narážíme na odlišný typ problému. Zde výsledky ovlivnil jeden jediný film, který v názvu obsahuje slova "Kazachstán" a "Amerika", i když v jiném tvaru, a je ho na internetu tolik, že Asie jako skutečný kontinent k tomuto státu neměla šanci. Zde pravděpodobně stačí počkat, až přejde módní vlna, jelikož z výsledků je jasné, že pro dvojici "Kazakhstan"-"Asia" bylo dosaženo velmi dobrých výsledků.

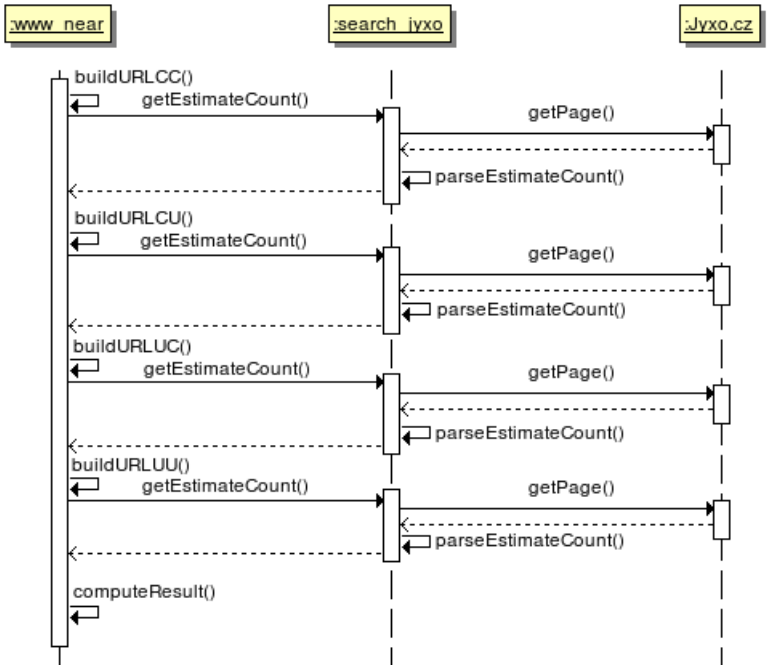
Z technického hlediska je řešení obdobné jako u funkce www_rank, pouze se vícekrát volá java funkce a následně internetový vyhledávač. Posloupnost volání je názorněji zobrazena na sekvenčním diagram 4.4.

4.3.4 Výsledky

Podíváme-li se na případ hledání číselníku k položce, určuje naše implementace funkce www_rank správný výraz pro 95 z 98 vzorků. Dosáhli jsme tedy úspěšnosti přibližně 97%, ale tato hodnota není příliš korektní, jelikož pomocí těchto dat jsme hledali správné koeficienty a funkci teprve utvářeli. Na správné ověření funkčnosti musíme vzít data jiná. K tomuto účelu použijeme druhý možný případ užití funkce www_rank. Našim cílem bude k sobě správně přiřadit české a latinské názvy živočichů.

Nejdříve se pokusíme najít správný latinský název pro každý český. Oborem hodnot, tedy množinou latinských výrazů, ze kterých budeme vybírat ten nejlepší, budou vždy všechny výrazy typu "Obratlovci_LA". Nebudeme se tedy omezovat jen na stejný číselník (živočišnou třídu), ale budeme hledat ve všech datech. Postup použijeme stejný jako v hledání vazeb položka-číselník. Všechna potřebná data si uložíme do speciální tabulky, k českému výrazu přiřadíme ten latinský, který spolu s ním dosáhne největší hodnoty

Obrázek 4.4: Sekvenční diagram funkce www_near



www_rank. Zajímat nás opět budou jenom špatně vyhodnocené výrazy. Překládáme 25 různých výrazů a odpovídající překlad hledáme také mezi 25 výrazy. Úspěšnost bude opět celkem dobrá, jelikož se správně určí 23 hodnot, jedna hodnota je určena nejednoznačně a jedna špatně.

Nejednoznačně určenou hodnotou je "Tamarin hnedy", který se spolu s "Saguinus inustus", toto je správná hodnota, i s "Canis lupus", toto je latinský název pro vlka obecného, vyskytuje na 18 stránkách a nelze tedy jednoznačně určit, který výraz je ten hledaný. Špatně určený je "Delfin obecny", který se spolu s "Canis lupus" nalézá na 13 stránkách, ale s "Sotalia fluviatilis", jako hledaným překladem, pouze na 10.

Druhý testovacím vzorkem bude opačné hledání. Jako základ si vezmeme latinské jméno obratlovce a hledaným výrazem bude jeho český překlad. Postup zvolíme úplně stejný. Diskuse nad výsledky bude tentokrát velmi krátká, protože jsme správně určili všechny hodnoty. Přeložili jsme tedy správně všech 25 latinských živočišných jmen.

Ve všech třech různých způsobech využití funkce www_rank jsme dosáhli úspěšnosti větší než 90%. Pokud bychom chtěli celkovou úspěšnost, hledali jsme dohromady 148 výrazů a z toho jsme korektně určili plných 143. Vyjádřeno v procentech, jde o přesnost 96,6%.

Při dopočítávání celkové hodnoty z jednotlivých mezivýsledků už nelze

jednoduchou cestou za pomoci koeficientů dosáhnout lepších výsledků. Ty by šlo možná získat buď za pomoci složitějšího algoritmu nebo pomocí neuronových sítí, pro něž jde o vhodnou úlohu, jelikož máme rozumně malou množinu vstupních parametrů a jako výsledek očekáváme jediné číslo. Navíc už nyní máme jistou učební množinu a její rozšiřování není složité. Naopak případné pokusy o vylepšení koeficientů by volaly po další moderní metodě, po využití genetických algoritmů. Zde by šlo využít jiné příhodné vlastnosti, snadného určení úspěšnosti jednotlivých sad koeficientů. Podobné cesty už ale nejsou předmětem této práce.

4.4 Funkce WWW_best_address

4.4.1 Popis funkce a příklady

Nyní postoupíme k poslední a k technicky nejnáročnější funkci popsané v této práci, k funkci *www_best_address*. Velkou změnou proti dosavadním dvěma funkcím bude už vlastní změna typu funkce, jelikož www_best_address bude funkcí agregační. Druhou její vlastností bude, že nevrací číslo, ale text. Jak již její název napovídá, tato funkce bude ke skupině výrazů hledat nejlepší společnou internetovou adresu.

Situace, kdy podobnou funkci potřebuji, je zřejmá. Máme spoustu výrazů a vyhledávač nám dokáže jednoduše vrátit ke každému z nich vhodné odkazy a dokonce uspořádané podle jejich předpokládaného užitku. Nás však nezajímá každý jednotlivý výraz, ale chceme se o nich něco dozvědět hromadně, případně najít stránku, kde se dozvíme i o dalších podobných výrazech ze stejné skupiny. Jednoduše na to lze použít www_best_address a spojíme-li ji s naším datovým modelem, můžeme dojít třeba k dotazu 17.

Zdrojový kód 17 Využití www_best_address

```
SELECT C01NAME,
       www_best_address(C02NAME)
FROM T02ITEM INNER JOIN
     T03GROUP_ITEM ON C02ID = C03C02_1 INNER JOIN
     T01GROUP ON C03C01_1 = C01ID
GROUP BY C01NAME,
         C01ID
ORDER BY C01ID;
```

Už na začátku si musíme uvědomit, že výsledkem nebude vždy adresa vedoucí přímo na portál. Může se třeba velmi lehce stát, že výsledkem bude

adresa domény i s několika stupni adresářové struktury, ale po vložení této adresy do vyhledávače nebude stránka nalezena. Podobná situace nastane, pokud například budeme hledat nejlepší adresu pro výrazy označující jednotlivé třídy obratlovců, jako máme v tabulce T01GROUP s C01TYPE = "Obratlovci_CS". Jako nejlepší výsledky nám vždy budou vráceny stránky ze stejné encyklopedie a budou mít následující tvar: "http://encyklopedie.cz/obratlovci/trida", kde jako třída budou jednotlivé názvy (ryby, savci, hmyz,...). Jako celkově nejlepší bude vybrána adresa "http://encyklopedie.cz/obratlovci", ale v tomto adresáři žádný soubor index.html nebo podobný být nemusí. Při troše hledání nás asi napadne, že když zkusíme adresu "http://encyklopedie.cz", tak se dozvíme vše podstatné o všech výrazech, které nás zajímají Je proto potřeba s tímto počítat a brát výsledek funkce www_best_address spíše jako ukazatel, kde informace leží, nikoliv jako přesnou adresu.

Další podstatnou změnou proti funkcím www_rank a www_near je, že www_best_address je výrazně více závislá na zvoleném internetovém vyhledávači. Dokud jsme využívali pouze odhadovaný počet nalezených odkazů, nebyly rozdíly mezi jednotlivými portály příliš patrné. Při současné velikosti internetu ho lze už považovat v podstatě za homogenní, tedy že poměr stránek o různých tématech zůstává stejný, ať prohledáváme libovolný dostatečně velký vzorek. V tomto případě však bereme do úvahy vlastní adresy stránek, což znamená, že vliv počtu oindexovaných stránek je výrazně větší. Ze stejného důvodu jsou důležitější i různé parametry vyhledávání jako třeba jestli vyhledáváme jenom v doméně .cz nebo ve všech doménách nebo jestli požadujeme, aby vyhledávač omezoval výsledky z podobných adres.

Z hlediska funkčnosti je pro nás mnohem důležitější, aby výsledkem byly relevantní adresy, než abychom tento výsledek dostali rychle. Je potřeba počítat s tím, že pro každý výraz nás zajímají přímo adresy odkazů, čímž časová náročnost znatelně vzroste. Časově bude podstatné, aby část po získání všech dat, tedy vlastní zpracování odkazů a jejich vyhodnocení proběhlo rychle a doba trvání celé funkce byla závislá pouze na rychlosti internetového připojení a odezvě vyhledávače.

4.4.2 Návrh řešení

Základní idea spočívá v tom, že si ke všem výrazům ve skupině zjistíme několik prvních odkazů dle vyhledávače a poté se z nich vezmeme nejčastěji se opakující doménu.

Při rozhodování, kolik odkazů nás bude od každého výrazu zajímat je potřeba skloubit několik protichůdných zájmů dohromady. Každý další odkaz v seznamu výsledků zvětšuje velikost načítané stránky. Rozdíl mezi zpracováním stránky s jedním nebo s 25 odkazy byl jedna vteřina proti šesti vteřinám.

Tabulka 4.3: Rozložení adresy na části

Část adresy	Pořadí
cz	1
cuni.cz	2
mff.cuni.cz	3
mff.cuni.cz/studim	4
mff.cuni.cz/studim/obecne	5
mff.cuni.cz/studim/obecne/dplayout.htm	6

Navíc se můžeme dostat k v podstatě nepodstatným stránkám. Už od druhé strany výsledků většinou nebývají příliš relevantní odkazy. Při opačném extrému a zvolení velmi malého počtu vyhledaných výsledků se nám může velmi lehce stát, že se domény získaných odkazů nebudou nikde prolínat a nám se bude těžko vyhodnocovat nejobsažnější adresa. Tento problém obzvláště vynikne pokud bude agregováno málo výrazů. Z toho plyne myšlenka, že počet odkazů hledaných pro každý výraz by se mohl odvozovat až podle počtu výrazů v agregaci. Obecně by se množství požadovaných odkazů mělo pohyboval mezi deseti a dvacetipěti, protože zde někde je ze zkušenosti hranice subjektivně užitečných odkazů, což typicky odpovídá dva a půl násobku počtu odkazů na jedné stránce s výsledky. Z tohoto pohledu nemá složitější výpočet smysl, protože rozdíl není příliš velký. Při větším počtu výrazů budou výsledky velmi podobné a při malém (1-4) je zase velmi relativní, která adresa je nejlepší.

Druhým závažným otazníkem je získání výsledné adresy ze záplavy odkazů k jednotlivým výrazům. Zřejmé je, že je potřeba rozdělit každou adresu po doménách i po stupních adresářové struktury. Proto například adresu „*http://www.mff.cuni.cz/studium/obecne/dplayout.htm*" rozložíme na části podle tabulky 4.3. Z této tabulky je také vidět, že jsme nepoužili protokol, protože pro vlastní ohodnocování nás příliš nezajímá a pokud bychom ho chtěli uvádět ve výsledku, je lepší ho zpětně dohledat z adres. Navíc ať už přistupujeme k souborům přes jakýkoliv protokol, jedná se stále o stejnou doménu. Ze stejného důvodu vynecháváme i část „www.". Z našeho pohledu nepřináší žádnou novou informaci, naopak by nám mohla zbytečně rozdělovat adresy na dvě různé skupiny, přestože by šlo o stejnou doménu.

Po rozdělení všech adres všech výrazů v agregaci získáme složitější tabulku, kde bude navíc ke každé části adresy ještě údaj, kolikátá v pořadí byla její původní adresa v seznamu adres, a také údaj, ke kterému výrazu v agregaci náleží. Toto jsou všechny dostupné a potřebné informace k získání ohodnocení pro všechny domény. Pro každou doménu by její ohodnocení mělo

být přímo úměrné k počtu jejích výskytů v seznamu všech domén, protože nás častěji se vyskytující domény zajímají více než ty málo se vyskytující. Mělo by být přímo úměrné stupni domény, protože „mff.cuni.cz/studium" je pro nás výrazně přesnější informace než „cuni.cz". A mělo by být nepřímo úměrné pořadí odkazů, které v ní leží v seznamu výsledků, protože věříme vyhledávači, že jím určené důležité odkazy jsou pro nás skutečně nejdůležitější. Pokud si představíme seznam částí domén jako tabulku můžeme k popisu výpočtu ohodnocení používat SQL. Tabulka bude mít pět sloupců:

phase výraz, pro který byla tato adresa nalezena

link_part část adresy

link_order pořadí odkazu ve výsledcích pro výraz

part_order stupeň domény

Nejjednodušší vzorec pro výpočet ohodnocení popisuje dotaz 18. Případně

Zdrojový kód 18 Jednoduchý vzorec pro výpočet ohodnocení částí domén

```
SELECT link_part,
       SUM(part_order/link_order)
FROM LINK_PARTS
GROUP BY link_part;
```

lze použít i různé varianty tohoto vzorce s důrazem na ten parametr, který by nám přišel důležitější. Další možností je použití klauzule

```
WHERE part_order > 1
```

pro zabezpečení, že nám nebude vrácena pouze koncovka „cz", „com" nebo některá podobná, pokud jsme tomu nezabránili již při rozdělování adres na části.

Na závěr již stačí pouze vybrat tu část adresy, která má nejvyšší ohodnocení a máme výsledek. Případně můžeme ještě pro uživatele doplnit „www." v adrese a protokol, hlavně s ohledem na možnost výskytu „https".

4.4.3 Technická realizace

Systém vytváření agregačních funkcí je u Oracle velmi podobný jako u ostatních databázových strojů. Je potřeba vytvořit uživatelský typ, který bude

obsahovat jednak data potřebná pro ukládání informací v průběhu agregace, jednak také funkce, které vlastní agregaci počítají. Počet a požadavky na funkce jsou stejné jako jinde, první funkce se zavolá před každou agregovanou skupinou. Druhá při každém novém prvku skupiny. Třetí na konci skupiny, tato počítá a vrací výslednou hodnotu agregace. Čtvrtá funkce slouží k umožnění paralelního výpočtu a spojuje dva mezivýsledky do jednoho. Bližší informace a příklady lze získat buď z dokumentace [2W] nebo z přílohy k této práci, kde jsou uloženy i zdrojové kódy. Pro nás jsou v této chvíli podstatné hlavně názvy jednotlivých funkcí, tedy Initialize pro úvodní nastavení, Iterate pro krok, Merge pro spojení dvou a Terminate pro závěrečný výpočet hodnoty.

K dosažení našeho cíle musíme s každým výrazem provést několik věcí u nichž příliš nezáleží, ve které z částí budou provedeny. První možnost spočívá v tom, že můžeme v průběhu funkce Iterate pouze ukládat výrazy do seznamu a poté v Terminate ke všem zjistit vyhledané odkazy, ty rozdělit na části a vypočítat optimální doménu. Druhým extrémem je v Iterate pro výraz rovnou zjistit odkazy a ty rozdělit na části a v Terminate už pouze dopočítat výsledek. Výsledek bude vždy stejný. My zvolíme cestu přibližně uprostřed. K tomu potřebujeme definovat několik nových datových typů. Protože si budeme postupně při každé iteraci k výrazu ukládat získané odkazy k němu, vytvoříme typ jyxoSearchResult, který obsahuje výraz, adresu odkazu a pořadí odkazu pro výraz. Samozřejmě potřebujeme seznam odkazů, takže si vytvoříme související typ jyxoSearchResultSet, který bude tabulkou prvků typu jyxoSearchResult. Obdobně si založíme typy pro jednotlivé části odkazů. Základní linkPart, který obsahuje pořadí odkazu, ze kterého pochází část, pořadí části a také vlastní část. Příkazy pro založení těchto typů lze nalézt jednak ve zdrojovém kódu 19 jednak také v příloze k této práci.

Poslední typ, který potřebujeme, je již výše zmíněný typ reprezentující samotnou agregační funkci. Zde si k němu pod dotazem 20 ukážeme pouze hlavičku, tělo i s definicí funkcí je uloženo pouze v příloze.

Již z definice hlavičky je vidět, že jako informace, které bude postupně iterativně doplňovat, jsme si zvolili seznam odkazů. Z toho vyplývá, že v rámci funkce Initialize si pouze nastavíme lo_links na prázdný seznam. Ve funkci Iterate, tedy té, která se volá ke každému výrazu, si pomocí javy necháme zjistit vyhledané odkazy k výrazu a přidáme do seznamu lo_links. Obdobně jednoduchá je i funkce Merge, kde pouze spojíme dva seznamy odkazů do jednoho. Podstatná logika tedy zůstává do funkce Terminate, která ze sesbíraných informací dopočítá výsledek.

Z technického hlediska spočívají zajímavá místa v předání seznamu odkazů mezí javou a funkci Iterate a potom ve výběru nejlepší části odkazu, jelikož máme seznam odkazů a potřebujeme v ideálním případě tabulku částí

Zdrojový kód 19 Založení datových typů

```
create or replace TYPE jyxoSearchResult AS OBJECT (
  words VARCHAR2(255),
  link_order NUMBER,
  link_url VARCHAR2(255)
);
create or replace TYPE jyxoSearchResultSet
 AS TABLE OF jyxoSearchResult;

create or replace TYPE linkPart AS OBJECT (
  link_order NUMBER,
  part_order NUMBER,
  link_part VARCHAR2(255)
);
create or replace TYPE linkPartSet
 AS TABLE OF linkPart;
```

odkazů. První problém řešíme nepříliš elegantně, ale o to jednodušeji. Seznam si předáme jako jeden VARCHAR2, kde jednotlivé adresy jsou oddělené pomocí mezery. Sice se tím omezujeme na maximální délku 4000 znaků, ale toto řešení vyžaduje nejmenší znalosti práce v javě. Navíc pro naše potřeby počet míst uložitelných do typu VARCHAR2 postačuje. Z důvodů použitelnosti i k jiným účelům zapouzdříme tuto java funkci do speciální funkce v DB, která nám tento dlouhý text převede na tabulku, navíc přidá informaci o pořadí odkazů. Vytvoříme proto funkci getLinksTable, která má jediný parametr typu VARCHAR2, kterým je vyhledávaná fráze, a aby se k jejímu výsledku, který je typu jyxoSearchResultSet dalo chovat jako k tabulce, jedná se o PIPELINED funkci. Ve funkci Iterate si pak načteme do kurzoru prvky, které nám funkce getLinksTable k iterovanému výrazu vrátí a ty postupně přidáme do seznamu lo_links.

Velmi podobně je řešena i funkce Terminate. Ta má na začátku seznam všech odkazů ke všem výrazům a jejím cílem je určit celkový výsledek agregace. Jelikož chceme co nejvíce využít možností SQL a také úvahy z předcházející kapitoly, opět využijeme PIPELINED funkci. Tentokrát bude o trochu pracnější, jelikož na vstupu bude mít seznam odkazů a vracet bude záznamy typu linkPart. V této funkci se tedy budeme starat o rozdělení odkazů na části, a proto ji nazveme getLinksPartTable.

Zbytek práce funkce Terminate už spočívá pouze v dotazu obdobném jako je uveden v kódu 18, kde je ovšem v klauzuli FROM použita místo tabulky

Zdrojový kód 20 Typ reprezentující funkci www_best_address

```
create or replace TYPE www_best_addressRoutines AS OBJECT (
  lo_links jyxoSearchResultSet,

  STATIC FUNCTION ODCIAggregateInitialize(
    best_address IN OUT NOCOPY www_best_addressRoutines
  ) RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateIterate(
    self IN OUT NOCOPY www_best_addressRoutines,
    value IN VARCHAR2
  ) RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateMerge(
    self IN OUT NOCOPY www_best_addressRoutines,
    second IN www_best_addressRoutines
  ) RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateTerminate(
    self IN www_best_addressRoutines,
    returnValue OUT NOCOPY VARCHAR2,
    flags IN number
  ) RETURN NUMBER
);
```

právě funkce getLinksPartTable s lo_links jako vstupním parametrem.

Pokud se rozhodneme pro uživatele doplňovat i protokol a případně i doménu „www", otevře se nám otázka, kterou možnost v případě více vybrat. Jelikož stačí libovolná funkční možnost, tak stačí kterákoliv z těch, které se nám v nalezených odkazech skutečně vyskytnou. Obecně nelze dopředu určit, který protokol uživatel upřednostňuje. Máme v podstatě pouze dvě možnosti, buď bude funkce getLinksPartTable vracet ke každé části odkazu i její tvar s použitím protokolu a domény „www" a my pak v dotazu použít třeba funkci MIN na její získání místo samotné části. Druhou možností je zpětné vyhledání těchto informací v seznamu odkazů až po vybrání části.

4.4.4 Výsledky

Jako hlavní kritérium funkčnosti funkce www_best_address použijeme dotaz 17, který vrací ke každému číselníku nejlepší adresu dopočítanou podle všech položek do tohoto číselníku patřících. Zajímat nás bude především použitelnost výsledných adres a jejich vztah k číselníku. Důležitá také bude závislost získaných odkazů na počtu odkazů vrácených k jednotlivým výrazům vyhledávačem, stejně jako závislost na vzorci pro výpočet ohodnocení části adresy. Ve všech případech však budeme používat dotaz s podmínkou na úroveň části větší než 1, jelikož samotné domény „.cz" nebo „.com" nám žádnou informaci nepřinesou. Jako vždy si zde pouze provedeme shrnutí výsledků a samotné hodnoty uvedeme pouze v příloze.

Nejdříve vyzkoušíme různé varianty výpočtu ohodnocení části adresy. Budeme testovat pouze základní varianty. Jmenovitě tedy již zmíněné:

$$value_{part} = \sum_{\forall vyskyty} (partOrder \div linkOrder)$$

dále zjednodušené a na pořadí odkazů nehledící

$$value_{part} = \sum_{\forall vyskyty} partOrder$$

a konečně nejjednodušší

$$value_{part} = \sum_{\forall vyskyty} 1$$

Jak lze jistě uhodnout a koneckonců i z výsledných hodnot jasně vidět, poslední varianta má výraznou tendenci brát časté, ale zbytečně nepřesné výsledky. Navíc zde nastává situace, že i v jinak ideálním případě, kdy nám vyhledávač vrátí ke každému výrazů na prvním místě stejný odkaz, určený třeba až do 4. domény, zde nemáme určeno, kterou z částí tohoto odkazu máme preferovat. Přesnější části adres nemají žádnou možnost mít větší ohodnocení, než méně přesné. Je vhodné zde minimálně doplnit podmínku, že pokud se nalezne více částí se stejným ohodnocením, vybere se ta nejpřesnější. V testovacím dotazu je typickým příkladem špatného výsledku odkaz „wz.cz" pro skupinu „Obojzivelnici", zbytek nevychází nijak špatně. Klademe zde velký důraz na množství výskytů adresy, ale menší už na její přesnost a důležitost.

Druhý způsob výpočtu má problémy u agregací tvořených z malého počtu prvků. Zde dochází k tomu, že jeden extrémně dlouhý odkaz má i přes pouze jeden výskyt a navíc na pozdějším místě relativně vysoké ohodnocení.

Dáváme tedy větší důraz na přesnost odkazu, naopak potlačen je počet opakování i důležitost podle vyhledávače. Víme však, že důležitost odkazů máme částečně ošetřenou omezením množství vrácených odkazů, takže nedochází k velkým rozdílům v jejich kvalitě. Jako příklad ideálního výsledku zde můžeme uvést adresu „www.mrk.cz/r/atlasy/atlas_ryb/maloostni/kaproviti" pro českou i latinskou variantu ryb. Opačným případem, tedy skupinou s nevhodným výsledkem je zde číselník států třeba pro Afriku a získaný výsledek „www.offshorenews.cz/Odkazy/odkazy", který se vyskytuje zadních částech v pořadí odkazů, ale má dostatek domén.

Větší vliv pořadí odkazů dává první uvedená varianta. Ještě více výsledků zde zastává server navajo.cz, který lze téměř považovat za univerzální odpověď na všechny dotazy. U této a jejích dalších variant, které více berou v potaz počet opakování, už jsou rozdíly mezi jednotlivými výsledky minimální a ideál se vybírá těžko. Ze subjektivního hlediska lze pro tato data preferovat výpočet

$$value_{part} = \sum_{\forall vyskyty} \{2 + (partOrder \div linkOrder)\}$$

V obecném případě vybrat nejlepší pravděpodobně nelze, ale přičítání konstanty se ukazuje jako velmi užitečné.

V předešlých zkoumáních jsme všude používali omezení, že počítáme pouze prvních deset odkazů ke každému dotazu. Při pohledu na testy s posledním vybraným vzorcem (s konstantním připočítáním dvojky ke každému výskytu) a použitím omezením na 25, 10, 5 a 2 odkazy na výraz zjistíme, že výsledky jsou velmi podobné a pouze u dvojky mají tendenci směřovat k příliš specializovaným odkazům. Vzhledem k tomu, že počet získávaných odkazů hraje velkou roli v časové náročnosti, je lepší zvolit co nejmenší hodnotu. Při velmi pomalém připojení a spěchu po výsledcích 5 a při rychlejším internetu si můžeme dovolit až 10 odkazů, ale víc nemá smysl. Poměr odezvy přibližně odpovídá počtu odkazů, tedy 1:2.

Na výsledcích je vidět relativně velká závislost na zvoleném vyhledávači, proto se zde často vyskytuje navajo.cz a blog.cz, které Jyxo.cz dává na začátek většiny dotazů. Celkově je to parametr, který výsledky ovlivňuje ze všech nejvíce. U všech vyhledávačů budou výsledky „použitelné", pouze jiné.

Kapitola 5

Závěr

Hlavní úkol této práce, tedy navrhnout a implementovat funkce využívající internetový vyhledávač a které jsou zároveň vhodné k použití v rámci SQL dotazů, se povedlo úspěšně splnit ve všech třech případech až s nadočekáváním dobrými výsledky. Funkci www_near lze využít jak na nalézání vazby typu číselník-položka, tak na dohledávání překladů odborných názvů. Obdobně funkce www_best_address vrací při použití relevantní hodnoty. Zároveň se podařilo dosáhnout rozumné časové odezvy všech funkcí.

Pokračování této práce lze vidět v několika směrech. Jednak čistě technickém, kdyby se šlo cestou zapracování těchto a případně dalších funkcí do některého DB serveru, pravděpodobně tedy MySQL. S tím úzce souvisí zapojení konceptu asynchronní iterace, který by dále snížil časové nároky dotazů. Podobně by šlo pracovat i opačným směrem a zjednodušit cesty k internetovému vyhledávači tak, aby ho šlo volat přímo a nikoliv pouze přes http rozhraní. Z hlediska rozvoje samotných funkcí by bylo zajímavé volání více různých vyhledávačů v rámci jednoho dotazu a následné spojení dílčích výsledků. Tímto způsobem by se dalo zamezit tendenci funkce www_best_address vracet odkaz na malý okruh portálů pro valnou většinu dotazů. Pokud bychom hledali složitější koncept, nabízí se varianta přímého zapojení výsledků do dotazů. To by znamenalo k pohledu obsahujícímu výsledky vyhledávače vytvořit další pohled, který by obsahoval vzájemné odkazy mezi stránkami. Tím by se k programátorům SQL dostal opravdu mocný nástroj na práci s webem.

Literatura

[1A] Roy Goldman, Jennifer Widom: „WSQ/DSQ: A Practial Approuach for Combined Querying of Databases and the Web" (2000), http://citeseer.ist.psu.edu/widom00wsqdsq.html

[2A] Daniela Florescu, Alon Levy, Alberto Mendelzon: „Database Techniques for the World-Wide Web: A Survey" (1998), http://citeseer.ist.psu.edu/florescu98database.html

[3A] Alex Iskold: Yahoo! Pipes and The Web As Database, server Read/WriteWeb (2007), http://www.readwriteweb.com/archives/yahoo_pipes_web_database.php

[4A] Nicolas Bruno, Luis Gravano, Amélie Marian: „Evaluating Top-k Queries over Web-Accessible Databases" (2004), http://citeseer.ist.psu.edu/704857.html

[1W] Yahoo! Pipes, http://pipes.yahoo.com

[2W] ORACLE Database Documentation Library: "PL/SQL User's Guide and Reference", http://www.oracle.com/pls/db102/homepage

[3W] MySQL 5.1 Reference Manual, http://ftp.linux.cz/pub/mysql/doc/refman/5.1/en/index.html

[4W] MS SQL Documentation, http://msdn2.microsoft.com/en-us/library/ms131077.aspx

[5W] CURL, http://curl.haxx.se/

[6W] Jyxo.cz, http://jyxo.cz

[7W] Morfeo, http://morfeo.centrum.cz/

[8W] Yahoo! Buzz Index, http://buzz.yahoo.com/

Seznam obrázků

Seznam zdrojových kódů

Seznam tabulek