Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Martin Čermák

Index pro textové vyhledávání nad relačními daty

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: Informatika
Studijní obor: softwarové systémy

Rád bych poděkoval vedoucímu mé diplomové práce RNDr. Michalovi Kopeckému, Ph.D. za cenné rady a odbornou pomoc.

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

V Praze dne 5. 8. 2008

Martin Čermák

Obsah

Abstrakt

Název práce: Index pro textové vyhledávání nad relačními daty
Autor: Martin Čermák
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Michal Kopecký, Ph.D.
e-mail vedoucího: Michal.Kopecky@mff.cuni.cz
Abstrakt:
Pro textové vyhledávání v relačních databázích byl navržen systém Précis, který umožňuje vyhledávat požadovaná data v celé databázi. Odpovědí na takovýto dotaz bude sjednocení výsledků, které bude obsahovat nejen data přímo související s dotazem, ale také informace, které s nimi souvisejí jen nepřímo. Protože tento systém je založen na vyhledávání textových výrazů nad všemi sloupci všech tabulek najednou, nejsou standardní indexy založené na B-stromech ani textové indexy pro tento účel příliš vhodné.
V práci byla navržena a implementována indexová struktura umožňující vyhledávání v libovolném množství tabulek a sloupců. Tato struktura je založena na invertovaném souboru. Implementovaný index umožňuje zadávání boolovských dotazů. Nalezené dokumenty jsou ohodnoceny a setříděny podle tohoto ohodnocení. Uživatelské rozhraní Précis indexu umožňuje používání běžných SQL dotazů pro vyhledávání požadovaných dokumentů. Implementace indexu je vytvořena pro databázový server Oracle.

Klíčová slova: textové vyhledávání, Précis, index, relační data, Oracle Database

Abstract

Title: Index for free form querying over relational data
Author: Martin Čermák
Department: Department of Software Engineering
Supervisor: RNDr. Michal Kopecký, Ph.D.
Supervisor's e-mail address: Michal.Kopecky@mff.cuni.cz
Abstract:
Précis system has been designed for text based searching over relational database. This system enables user to search requested data over whole database. Answer to these free-form queries is a synthesis of results containing not only information directly related to the query selections but also information implicitly related to them. Neither standard B-tree based indices nor text based indices are suitable for this purpose because we need to search requested data in all columns of all tables within the database.
The goal of this thesis is to design and implement index structure, which will contain data from any number of columns and tables. This structure is based on inverted file. Implemented index supports boolean queries. Result documents are weighted and ordered by this weight. User interface of Précis index uses standard SQL queries to search for desired documents. Implementation of the index is created for Oracle Database server.

Keywords: information retrieval, Précis, index, relational data, Oracle Database

1. Úvod

Pro vyhledávání v relačních databázích se většinou používají indexy, vytvořené nad jednotlivými sloupci tabulek, případně nad skupinami sloupců jedné tabulky. Tyto indexy jsou zpravidla vytvořeny pomocí B-stromu. Takto vytvořený index lze však použít pouze pro hledání ve strukturovaných datech. Pokud chceme vyhledávat nějaká slova v textu, například článku nebo knize, budeme potřebovat fulltextové vyhledávání. Většina databázových serverů dnes nabízí možnost vytvoření indexu pro takovýto způsob vyhledávání. Vždy se však jedná o index, vytvořený pouze nad jednou tabulkou stejně, jako v případě indexů klasických.

Vyhledávání v databázi s použitím klasických indexů vyžaduje, aby uživatel rozuměl struktuře databáze. Jedním ze způsobů, jak uživateli poskytnout co nejrozsáhlejší odpověď na jeho dotaz je použití dotazů Précis [4, 8]. Uživatel zadá pouze seznam klíčových slov a systém vyhledá všechny relevantní záznamy ve všech tabulkách v databázi. Navíc systém vyhledá informace, související s již nalezenými záznamy, například přes vazby realizované pomocí cizích klíčů. Výsledek pak může být zobrazován v podobě hierarchie výsledků, nebo formátován pomocí šablon do vět.

Aby však takovýto systém pro dolování dat mohl v relační databázi efektivně vyhledávat, je potřeba vytvořit index ze všech v ní uložených dat. Podstatným rozdílem oproti běžným – klasickým i fulltextovým – indexům je tedy schopnost indexovat více tabulek zároveň.

1.1. Cíl práce

Cílem mé práce je návrh a implementace fulltextového indexu, který bude umět efektivně vyhodnocovat Précis dotazy, využívající termy a boolovské operátory. Index by měl zahrnovat libovolné množství sloupců z různých tabulek, případně schémat. Implementace indexu bude realizována pro Oracle Database 10g. Výsledná implementace bude pracovat nad takovými datovými strukturami, ve kterých bude relativně snadné vkládání nových, mazání a hlavně vyhledávání dokumentů.

Tato práce si tedy klade za cíl navázat na [9], kde však pro vyhledávání v databázi nejsou použity žádné indexy, ale pouze dotazy typu:

```
SELECT sloupec11, sloupec12
FROM tabulka1
WHERE
   (sloupec11 LIKE %term1% OR sloupec11 LIKE %term2%) OR
   (sloupec12 LIKE %term1% OR sloupec12 LIKE %term2%)
UNION ALL
…
```

Dotazy tohoto typu budou fungovat relativně rychle pouze na velmi malé databázi. Pokud by mělo být prohledáváno velké množství dat tímto způsobem – a pro tyto případy je Précis dotazování určeno především – tak by čas na získání odpovědi byl příliš dlouhý. Navíc zde nelze nalezené záznamy uspořádat podle jejich relevance.

1.2. Databázový server

Implementace je vytvořena pro databázový systém Oracle Database 10g. Tento sever je nabízen také ve verzi Express Edition, která je volně dostupná i pro komerční použití. Důvody, proč jsem se rozhodl pro tento databázový server, jsou jeho velké rozšíření a rozsáhlá dokumentace umožňující relativně snadné rozšíření databáze o další funkcionalitu.

Ve verzi Express Edition je také dostupné fulltextové vyhledávání Oracle Text, jehož část je v implementaci využita.

2. Précis

Dotazování formou Précis [4, 8] vyžaduje od uživatele pouze zadání klíčových slov. Pro ně pak systém sám vyhledá relevantní informace. Například pro dotaz „Woody Allen“ by mohla odpověď vypadat následovně:

“Woody Allen se narodil 1. prosince 1935 v Brooklynu, New York, USA. Jako režisér pracoval na filmech Match Point (2005), Melinda and Melinda (2004), Anything Else (2003). Hrál ve filmech Hollywood Ending (2002), The Curse of the Jade Scorpion (2001).”

Bez převodu na textovou reprezentaci by odpověď mohla vypadat:

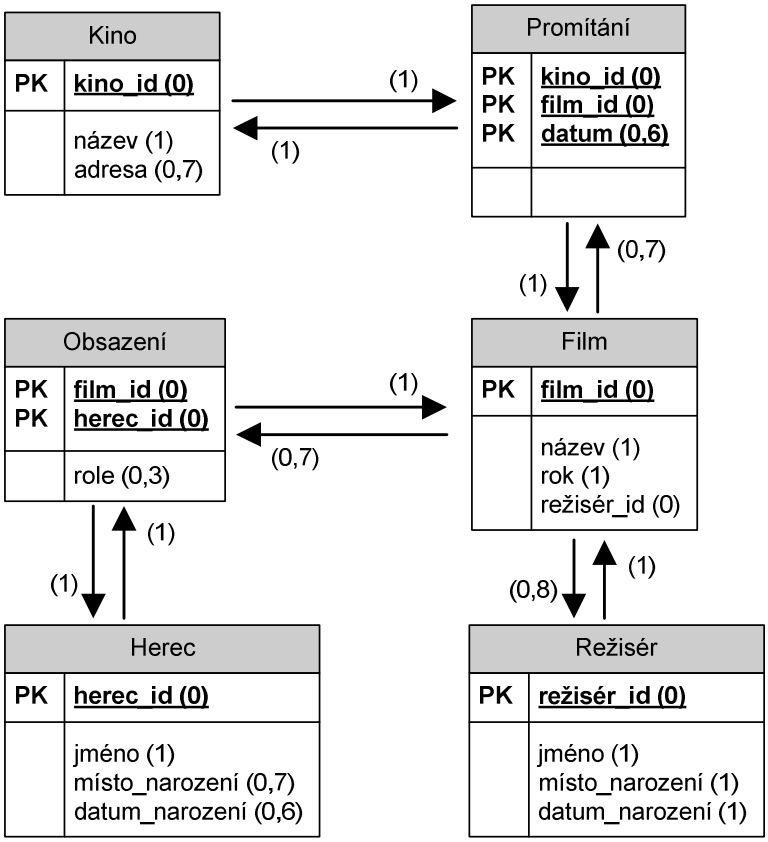
Režisér: Woody Allen | 1. prosinec 1935 | Brooklyn | New York | USA
Film: Match Point | 2005
Film: Melinda and Melinda | 2004
Film: Anything Else | 2003
Obsazení: Hollywood Ending | 2002
Obsazení: The Curse of the Jade Scorpion | 2001

Na rozdíl od běžného textového vyhledávání jsou v tomto případě informace uloženy v klasické relační databázi. Abychom mohli získat takovouto textovou odpověď, je potřeba, aby mezi jednotlivými tabulkami byly vytvořeny závislosti. Tyto závislosti budou ohodnoceny a v relační databázi budou reprezentovány cizími klíči. Stejně tak budou ohodnoceny i sloupce v každé z tabulek. Ohodnocením závislostí a sloupců je reálné číslo $w \in <0, 1>$, které může být závislé na uživatelských preferencích. Ohodnocení rovné 1 znamená, že se jedná o významný vztah. Pokud bude v odpovědi jeden objekt z takovéhoto vztahu, pak by se tam měl objevit i ten druhý.

Ohodnocení závislé na uživatelských preferencích pak mohou způsobit, že pro stejný dotaz získá každý uživatel jinou odpověď co do obsahu či uspořádání faktů. Ohodnocení orientované cesty délky k se vypočítá podle vzorce $w = w_1.w_2 \ldots w_k$

Pro příklad uveďme následující schéma [4] a jeho graf (obr. 2-1).

Kino (kino_id, název, adresa)
Promítání (kino_id, film_id, datum)
Film (film_id, název, rok, režisér_id)
Obsazení (film_id, herec_id, role)
Herec (herec_id, jméno, místo_narození, datum_narození)
Režisér (režisér_id, jméno, místo_narození, datum_narození)



Obrázek 2-1 Graf ukázkové databáze. Čísla v závorkách označují ohodnocení dané vazby.

Podívejme se například na ohodnocení vztahů mezi tabulkami Film a Režisér. Údaje z obou tabulek jsou na sobě závislé. Ohodnocení obou závislostí je však rozdílné, což znamená, že pokud v odpovědi budou informace o nějakém režisérovi, tak by se tam měly vždy objevit i filmy, které režíroval. Ohodnocení tohoto vztahu je maximální, tedy 1. Naopak pokud budou v odpovědi filmy, tak jména režisérů v odpovědi být nemusí – ohodnocení tohoto vztahu je již menší.

2.1. Vyhodnocení Précis dotazu

Vyhodnocení dotazu se skládá z několika po sobě jdoucích kroků, které jsou zobrazeny na obrázku 2-2. V popisu návrhu je zde předpokládáno, že dotaz je

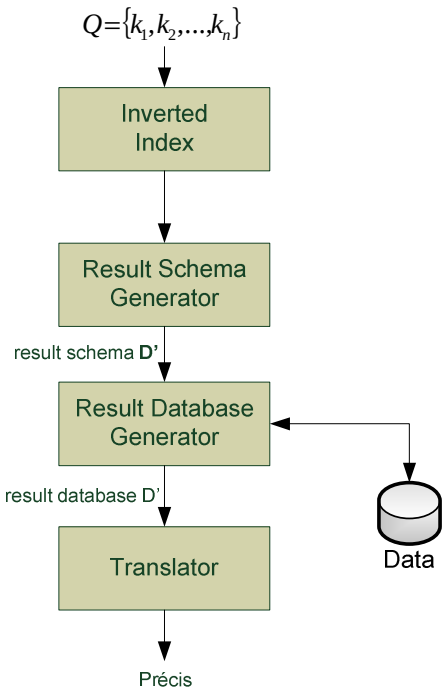
tvořen seznamem termů. Termy nejsou navzájem vázány žádnými logickými spojkami.

Inverted Index – pro všechny termy z dotazu Q vytvoří seznam výskytů v databázi D

Result Schema Generator – zjistí, která část databáze může obsahovat informace relevantní k dotazu Q. Výstupem pak bude schéma **D'**, které vznikne z **D** použitím pouze těch relací, do kterých vede orientovaná cesta z výskytů termů.

Result Database Generator – tato část vytvoří výslednou databázi D' odpovídající schématu **D'**.

Translator – provede závěrečný krok, kde výslednou databázi D' převede do textové podoby tak, jak bylo ukázáno v úvodu této kapitoly.



Obrázek 2-2 Schéma vyhodnocení Précis dotazu [4]

3. Dokumentografické informační systémy

Dokumentografické informační systémy se používají pro vyhledávání textu v nestrukturovaných datech. Navíc zpravidla umožňují ohodnocení nalezených dokumentů, aby uživatel nemusel procházet všechny, které systém nalezl, ale nalezl ty nejvíce relevantní dokumenty na začátku seznamu.

3.1. Efektivita DIS

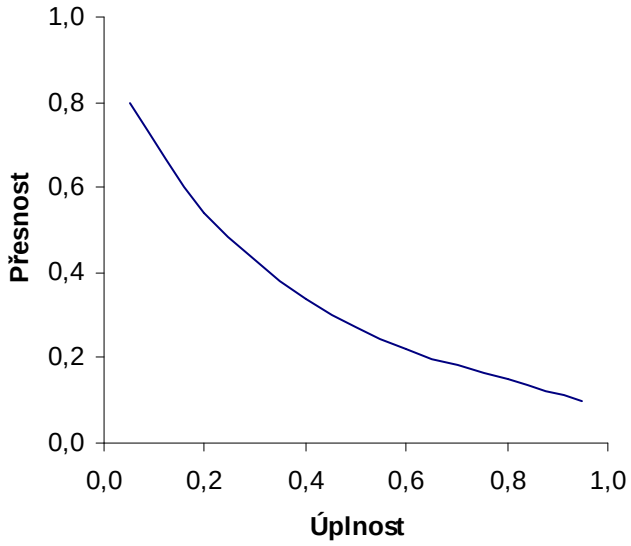
Při hodnocení efektivity DIS nás bude zajímat rychlost zpracování požadavku, způsob kladení dotazů, forma odpovědi, ale především schopnost systému vrátit v odpovědi relevantní dokumenty. Za relevantní dokumenty jsou považovány ty dokumenty, které poskytnou tazateli požadovanou informaci. Míra schopnosti systému uspokojit tazatelovy požadavky se často vyjadřuje pomocí dvou ukazatelů: *koeficientu přesnosti* a *koeficientu úplnosti*. Tyto koeficienty jsou dány vztahy:

$$Přesnost = \frac{počet\ vybraných\ relevantních\ dokumentů}{počet\ všech\ relevantních\ dokumentů\ v\ kolekci}$$

$$Úplnost = \frac{počet\ vybraných\ relevantních\ dokumentů}{počet\ všech\ vybraných\ dokumentů}$$

Přesnost odpovídá pravděpodobnosti, že dokument ve výsledku bude pro tazatele relevantní, úplnost pak odpovídá pravděpodobnosti, že relevantní dokument bude uveden ve výsledku. V ideálním případě by oba tyto koeficienty byly rovné jedné. V takovém případě by DIS dal do odpovědi všechny relevantní dokumenty a zároveň by odpověď neobsahovala žádný nerelevantní dokument. V praxi však takového výsledku zpravidla není možné dosáhnout pomocí nabízených možností formulace dotazu. Ukazuje se, že oba tyto koeficienty jsou na sobě nepřímo závislé. Jejich závislost je zobrazena v grafu 3-1.

Koeficienty přesnosti a úplnosti nejsou závislé pouze na použitém DIS, ale také na položeném dotazu a na konkrétním tazateli. Obecně platí, že pokud je dotaz velmi úzce specifikován, tak je vráceno jen malé množství nerelevantních dokumentů, ale také velmi málo relevantních dokumentů. Tím dosáhneme vysoké přesnosti, ale za cenu nízké úplnosti. Naopak, pokud je dotaz obecnější, pak systém vrátí více relevantních dokumentů, ale odpověď bude obsahovat i velké množství dokumentů nerelevantních. Zvýší se tedy koeficient úplnosti, ale dojde k poklesu přesnosti.



Graf 3-1 Vztah přesnosti a úplnosti

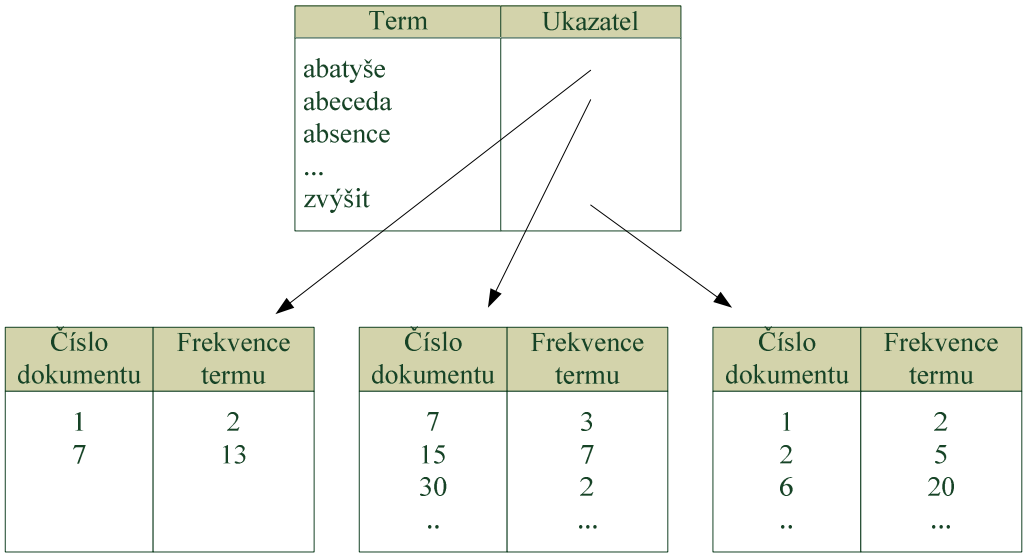
Pokud bychom chtěli zároveň zvýšit přesnost i úplnost odpovědi, pak bychom museli použít mnohem složitější přístup ke kolekci dokumentů, založený například na znalosti sémantiky textů. Takovýto přístup je obvykle neúměrně výpočtově náročnější.

Vzhledem k definici relevantního dokumentu je hodnota koeficientů navíc závislá na subjektivním názoru tazatele. Dva tazatelé, kteří zformulovali shodný dotaz v jazyce, který DIS poskytuje, se nemusí shodovat v názoru na to, zda konkrétní dokument je nebo není relevantní, tedy zda obsah dokumentu uspokojuje jejich požadavky.

3.2. Invertovaný soubor

Invertovaný soubor je nejběžnějším typem datové struktury používané v DIS. Každý dokument je reprezentován seznamem termů. Invertovaný soubor obsahuje seznam dvojic <*id_dokumentu*;*id_termu*> setříděný primárně podle termů. Nad tímto seznamem je vytvořen seznam termů, kde každá položka tohoto seznamu odkazuje na začátek odpovídajícího úseku dvojic. Pro každý term je tak možné efektivně vyhledat seznam dokumentů, ve kterých se vyskytuje. V souboru mohou být navíc uložena ohodnocení, jak moc je daný term pro daný dokument významný. Toto ohodnocení zpravidla vychází z frekvence termu v dokumentu [3]. Pomocí ohodnocení termů pak lze spočítat ohodnocení dokumentů, které vyhovují dotazu a podle něj určit pořadí dokumentů v odpovědi. Dále pak mohou být u záznamů další doplňující informace, jako například pozice v textu, případně může být text rozdělen na odstavce nebo věty. Množství a typ doplňujících informací souvisí s konkrétním modelem implementovaného DIS.

Struktura invertovaného souboru se tedy skládá ze dvou částí: ze seznamu termů a ze seznamu výskytů daného termu v dokumentech. Struktura souboru je zobrazena na obrázku 3-1.



Obrázek 3-1 Struktura invertovaného souboru

3.3. *Konstrukce invertovaného souboru*

Při konstrukci invertovaného souboru je potřeba dokumenty převést na jednotlivé termy. U každého termu si musíme pamatovat číslo dokumentu, frekvenci termu v dokumentu, případně nějaké bližší určení výskytu termu v dokumentu. Každý term je porovnán se slovníkem nevýznamných slov, tzv. stop list. Tento slovník obsahuje slova, která nejsou vhodná pro textové vyhledávání, jako například spojky, předložky a slova, která se v kolekci dokumentů vyskytují velmi často. Pokud bychom do invertovaného souboru přidali i tato slova, pak by invertovaný soubor byl podstatně větší, hledání v něm by bylo pomalejší a výsledky hledání by se příliš nezlepšily.

Při odhadu velikosti invertovaného souboru lze využít empirický Zipfův zákon, který byl popsán již v roce 1949. Zákon říká, že pokud seřadíme sestupně všechna slova z dostatečně velké kolekce dokumentů sestupně podle četnosti jejich výskytu, pak součin pořadí a frekvence výskytů bude přibližně konstantní. Frekvenci vypočítáme jako podíl počtu výskytů hledaného slova a počtu všech slov v kolekci dokumentů. Formálně lze Zipfův zákon napsat:

$$f_t * r_t \approx k ,$$

kde f_t je frekvence výskytů termu t v kolekci dokumentů, r_t je pořadí termu a k je konstanta závislá na dané kolekci dokumentů. Vlastnost Zipfova zákona je vidět v tabulce 3-1 [2], která obsahuje prvních 10 slov anglického frekvenčního slovníku, vzniklého na základě kolekce obsahující přibližně 1.000.000 výskytů slov.

pořadí	slovo	frekvence	pořadí*frekvence
1	the	0,069971	0,069971
2	of	0,036411	0,072822
3	and	0,028852	0,086556
4	to	0,026149	0,104596
5	a	0,023237	0,116185
6	in	0,021341	0,128046
7	that	0,010595	0,074165
8	is	0,010099	0,080792
9	was	0,009816	0,088344
10	he	0,009543	0,095430

Tabulka 3-1 Prvních 10 slov anglického frekvenčního slovníku

3.4. Modely DIS

Model DIS určuje, jaké informace budeme potřebovat z dokumentů získat a jakým způsobem se v kolekci dokumentů bude vyhledávat. Každý model používaný v DIS má své výhody i nevýhody. Obecně lze říci, že čím lepší specifikaci dotazu bude model umožňovat, tím kvalitnější bude odpověď a tím složitější bude zároveň její nelezení. V relačních databázích se nejčastěji používá boolovský model anebo některé jeho rozšíření. Na boolovském modelu je například založen fulltextový index Oracle Text.

3.4.1. Boolovský model

Boolovský model se používá již od padesátých let minulého století. V tomto modelu jsou dokumenty reprezentovány množinami termů. Dotaz se formuluje ve formě boolovské formule, sestávající se z atomických formulí – termů, spojených operátory AND, OR a NOT. Dále mohou jednotlivé implementace povolovat používání závorek, případně dalších operátorů. Mezi operátory, často implementované v současných relačních databázích, patří například proximitní operátory, určené pro výběr dokumentů, ve kterých se termy vyskytují současně ve stejné větě, respektive ve stejném odstavci, případně jsou od sebe vzdáleny nejvýše *n* slov. Dále pak dotazovací jazyky mohou povolovat využívání regulárních výrazů pro definice disjunkcí všech termů, které vyhovují dané masce. Dalším způsobem, jak lze zvýšit úspěšnost vyhledávání, je použití tezauru. Ten nabízí možnost nalezení dokumentů, které sice neobsahují přímo hledaný term, ale přesto mohou být pro uživatele relevantní.

Přestože boolovský model má mnoho vylepšení, má stále podstatné nevýhody. Mezi hlavní zápory modelu patří nemožnost uspořádání výsledků podle relevance. Standardně model totiž neuvažuje ohodnocení dokumentů a všechny vybrané dokumenty jsou stejně významné. Z tohoto důvodu lze jen obtížně

regulovat počet nalezených dokumentů na dotaz a snadno se může stát, že systém jich buď vrátí velké množství, nebo naopak žádný. Navíc při konjunkci více termů stačí, aby dokument neobsahoval jediný z nich, a dokument nebude vybrán, přestože by pro tazatele mohl být zajímavý. Naopak při disjunkci více termů budou dokumenty obsahující pouze jeden z nich i dokumenty obsahující všechny hledané termy, považovány za stejně významné.

3.4.2. Vektorový model

Ve vektorovém modelu je každému dokumentu přiřazen vektor hodnot, kde každá dimenze odpovídá jednomu termu. Vektor může obecně obsahovat libovolné nezáporné číslo. Obvykle se však pro reprezentaci používají hodnoty z intervalu <0; 1>. Hodnota vyjadřuje stupeň relevance daného termu k dokumentu. Stejně jako pro dokumenty v kolekci je vytvořen vektor i pro každý dotaz. Vzájemná podobnost dokumentů a dotazu je počítána na základě podobnosti dvojice vektorů. Pro výpočet podobnosti dokumentu a dotazu lze použít různé míry. Předpokládejme, že v kolekci dokumentů **D** bylo použito *n* různých termů $t_1,...,t_n$. Potom každý dokument D_i je reprezentován vektorem

$$D_i = (w_{i1}, w_{i2}, ..., w_{in}), \text{ kde } w_{ij} \in <0,1>$$

Dotaz *Q* bude reprezentován obdobným vektorem

$$Q = (q_1, q_2, ..., q_n), \text{ kde } q_j \in <0,1>$$

Podobnost dotazu s dokumentem bude vyjadřovat funkce $Sim(Q, D_i) \in <0,1>$. Pro výpočet podobnosti může být použit například:

- Skalární součin:

$$Sim(Q, D_i) = \sum_{j=1}^{n} q_j * w_{ij}$$

- Kosinova míra:

$$Sim(Q, D_i) = \frac{\sum_{j=1}^{n} q_j * w_{ij}}{\sqrt{\sum_{j=1}^{n} q_j^2} * \sqrt{\sum_{j=1}^{n} w_{ij}^2}}$$

- Jaccardova míra:

$$Sim(Q, D_i) = \frac{\sum_{j=1}^{n} q_j * w_{ij}}{\sum_{j=1}^{n} q_j + \sum_{j=1}^{n} w_{ij} - \sum_{j=1}^{n} q_j * w_{ij}}$$

Pro výpočet vah jednotlivých termů v dokumentu se nejčastěji používají vztahy vycházející z frekvence termu v dokumentu. Důvodem je předpoklad, že čím se term v dokumentu objevuje častěji, tím je pro jeho identifikaci důležitější. Protože však slova vyskytující se v jazyce, a tedy i v každém z dokumentů, nejčastěji budou zpravidla nevýznamná, je vhodné tato slova zařadit do stop listu. Frekvenci termu budeme označovat *TF* a její hodnota se vypočítá pomocí vztahu

$$TF_{ij} = \frac{\textit{počet výskytů termu v dokumentu}}{\textit{počet výskytů všech termů v dokumentu}}$$

Protože hodnota *TF* je velice nízká i pro časté termy, provádí se obvykle její normalizace tak, aby hodnoty byly v rozsahu <0, 1>. Tato normalizace zároveň zajistí, že termy, které se v dokumentu objevují jen s minimální frekvencí, nebudou vůbec uvažovány. Hodnota normalizované frekvence termů se vypočítá

$$NTF_{ij} = 0 \qquad \text{pro } TF_{ij} < \varepsilon$$

$$NTF_{ij} = \frac{1}{2} + \frac{1}{2} \cdot \frac{TF_{ij}}{\max_k(TF_{ik})} \qquad \text{jinak}$$

Ohodnocení termů v dokumentu můžeme dále zlepšit tím, že budeme uvažovat i výskyt termu v rámci celé kolekce dokumentů. Tato myšlenka vychází z toho, že pokud se nějaký term vyskytuje ve většině dokumentů z kolekce, pak bude méně významný pro vyhledávání než takový, který se vyskytuje jen v menší části dokumentů. Termy, které jsou obsaženy ve většině dokumentů, změní výsledek vyhledávání je zanedbatelně, a proto nejsou pro vyhledávání důležité. Jednou z možností, jak četnost výskytu termu v kolekci dokumentů zohlednit je využití tzv. *inverzní frekvence termu* v dokumentech (IDF). Tato hodnota klesá s počtem výskytů termu v různých dokumentech a je dána vztahem

$$IDF_j = 1 + \log\left(\frac{\textit{počet všech dokumentů v kolekci}}{\textit{počet dokumentů obsahující term } j}\right)$$

což formálně můžeme zapsat jako

$$IDF_j = 1 + \log\frac{n}{\left|\{D_i : t_j \in D_i\}\right|} \qquad \text{pro } i = 1 \ldots n,$$

kde *n* je počet dokumentů v kolekci. Ohodnocení termu je pak dáno vztahem

$$w_{ij} = NTF_{ij} * IDF_j$$

Výsledné váhy se často již během indexace normalizují tak, aby měl výsledný vektor, reprezentující dokument, jednotkovou velikost. V takovém případě se ohodnocení termů vypočítá pomocí vztahů

$$v_{ij} = NTF_{ij} * IDF_j$$

$$w_{ij} = \frac{v_{ij}}{\sqrt{\sum_k v_{ik}^2}}$$

3.4.3. Rozšířený boolovský model

Rozšířený boolovský model vychází z klasického boolovského modelu, ale navíc řeší některé problémy, které jsou spojené s vyhodnocováním boolovských dotazů. Při vyhodnocování konjunkce termů se tak například mohou dokumenty, obsahující jen některé z požadovaných termů, také dostat do výsledku, ale s nižším ohodnocením než ty, které obsahují všechny hledané termy. Obdobně jsou i při vyhodnocování disjunkce termů dokumenty obsahující více hledaných termů lépe ohodnoceny a ve výsledku jsou upřednostňovány. Obecně lze říci, že konjunktivní dotazy se chovají částečně jako dotazy disjunktivní a naopak.

Výhodou modelu zůstává snadnost implementace podobně jako u klasického boolovského modelu. Přitom výpočetní náročnost není o mnoho vyšší.

Fuzzy množiny

Použitím fuzzy množin lze spojit boolovský model s požadavkem na ohodnocování nalezených dokumentů [1]. S každým prvkem množiny je v tomto případě asociován stupeň příslušnosti k množině.

Konjunkce a disjunkce termů je vyhodnocována obdobně, jako u boolovského modelu, tedy průnikem resp. sjednocením množin dokumentů obsahujících dané termy. Rozdíl je pouze v tom, jak jsou tyto operace pro fuzzy množiny definovány.

Označme $Deg_A(u) \in <0,1>$ stupněm příslušnosti prvku *u* do množiny *A*. Pokud prvek *u* patří do množiny *A*, pak $Deg_A(u) = 1$, pokud do ní nepatří, pak $Deg_A(u) = 0$. Pokud prvek *u* patří do množiny *A* jen částečně, pak platí $Deg_A(u) \in <0,1>$. Operace sjednocení a průniku fuzzy množin *A* a *B* mohou být definovány následovně:

$$Deg_{A \cap B}(u) = \min(Deg_A(u),\ Deg_B(u))$$

$$Deg_{A \cup B}(u) = \max(Deg_A(u),\ Deg_B(u))$$

nebo

$$Deg_{A \cap B}(u) = Deg_A(u) * Deg_B(u)$$

$$Deg_{A \cup B}(u) = Deg_A(u) + Deg_B(u) - Deg_A(u) * Deg_B(u)$$

MMM model

V MMM modelu (Mixed Min and Max) se podobnost dotazu a dokumentu počítá jako lineární kombinace minimální a maximální váhy dokumentu. Mějme dokument *D* s vahami $w_1, w_2, \ldots, w_m$ pro termy $t_1, t_2, \ldots, t_m$ a dotazy

$$Q_{\mathrm{OR}} = t_1 \,\mathrm{OR}\, t_2 \,\mathrm{OR} \ldots \mathrm{OR}\, t_m$$

$$Q_{\mathrm{AND}} = t_1 \,\mathrm{AND}\, t_2 \,\mathrm{AND} \ldots \mathrm{AND}\, t_m$$

pak koeficient podobnosti je vypočítán následovně

$$Sim(Q_{\mathrm{OR}}, D) = C_{\mathrm{OR1}} * \max(w_1, w_2, \ldots, w_m) + C_{\mathrm{OR2}} * \min(w_1, w_2, \ldots, w_m)$$

$$Sim(Q_{\mathrm{AND}}, D) = C_{\mathrm{AND1}} * \min(w_1, w_2, \ldots, w_m) + C_{\mathrm{AND2}} * \max(w_1, w_2, \ldots, w_m)$$

kde *C* jsou koeficienty pro konjunkci resp. disjunkci termů. Přesné hodnoty koeficientů nejsou dány, ale musí platit

$$C_{\mathrm{OR1}} < C_{\mathrm{OR2}} \text{ a } C_{\mathrm{AND1}} < C_{\mathrm{AND2}}$$

Podle [5] se experimenty podařilo ukázat, že nejlepších výsledků bylo dosaženo s koeficienty C_{AND1} v rozmezí 0,5 a 0,8. Hodnota C_{OR1} by měla být vyšší než 0,2. Zároveň model poskytoval mnohem lepší výsledky, než klasický boolovský model.

Paiceův model

Rozšířením MMM modelu vzniknul Paiceův model. Na rozdíl od MMM, tento model zahrnuje do výpočtu váhy všech termů, které se účastní konjunkce nebo disjunkce. Podobnost dotazu $Q = t_1, t_2, \ldots, t_m$ s dokumentem *D* je dána vzorcem

$$Sim(Q, D) = \frac{\sum_{i=1}^{m} r^{i-1} w_i}{\sum_{i=1}^{m} r^{i-1}},$$

kde t_i jsou termy, w_i jsou váhy daného termu pro dokument *D* a *r∈<0;1>* je koeficient. Pro konjunkci termů jsou váhy w_i setříděny ve vzestupném pořadí, pro disjunkci jsou setříděny opačně. Hodnota koeficientu *r* může být různá pro konjunkci a disjunkci termů. Podle [7] lze dosáhnout dobré výsledky stanovením hodnoty *r* na 1 pro konjunkci a 0,7 pro disjunkci.

Pro konjunkci nebo disjunkci dvou termů se tento model chová stejně jako MMM. Výpočetní složitost bude oproti MMM vyšší, protože ohodnocení termů musí být setříděno.

4. Oracle data cartridge

Databázový server Oracle umožňuje implementaci tzv. data cartridge, pomocí kterých lze rozšířit schopnosti databáze o novou funkcionalitu [10]. Lze tak například vytvořit nový datový typ a definovat způsob, jak mají být data interpretována a ukládána. Data cartridge obsahuje balík funkcí a procedur, které definují požadované chování datového typu. Tento balík se instaluje do databázového serveru, kde bude uložen. Všechny jeho funkce a procedury budou spouštěny na serveru.

V databázovém serveru Oracle je rovněž definováno rozhraní, pomocí kterého lze implementovat funkce, potřebné pro indexaci nového datového typu. Implementací tohoto rozhraní se vytvoří objekt, který se nazývá *indextype*. Rozhraní indexu může být implementováno pomocí jazyka PL/SQL, Java, C nebo C++. Oracle nazývá indexy, vytvořené s pomocí tohoto rozhraní, indexy doménovými. Tyto indexy se používají i vytvářejí, podobně jako běžné indexy, prostřednictvím SQL příkazů.

Další částí data cartridge může být tzv. *statictics type*, což je rozšíření databázového optimalizátoru. Pomocí této součásti lze například definovat funkce pro výpočet statistiky, selektivity a časové náročnosti vyhledávání v indexu. Optimalizátor pak používá tyto funkce při rozhodování, zda má být použit doménový index, nebo zda raději použít index jiný.

4.1. Oracle Text

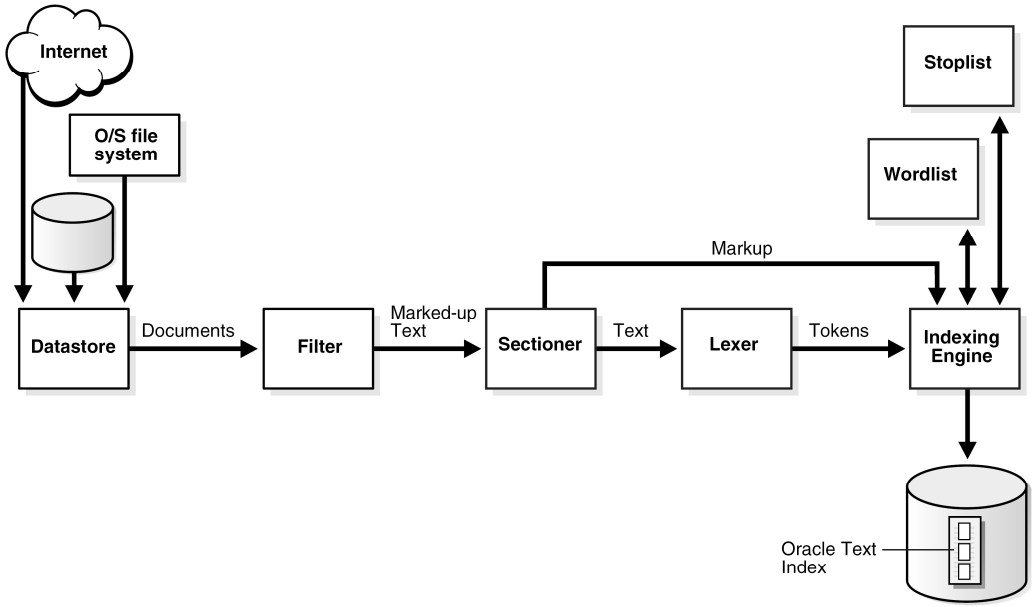
Oracle Text [11] je fulltextový index, který se do databázového serveru Oracle instaluje jako data cartridge. Umožňuje vytvoření invertovaného souboru pro kolekci dokumentů. Tyto dokumenty mohou být načteny z jednoho nebo více sloupců tabulky, ze souboru uloženého na serveru, nebo z internetu pomocí URL adresy.

4.1.1. Vytvoření indexu

Fulltextový index lze vytvořit pomocí SQL příkazu ve tvaru:

```
CREATE INDEX idx_name ON doc_table(text)
  INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS (...);
```

Proces vytvoření indexu, zobrazený na obrázku 4-1, se skládá z několika kroků. Jednotlivé kroky lze nastavit pomocí klauzule parameters při vytváření indexu.



Obrázek 4-1 Oracle Text – indexace [11]

Filter

Pomocí filtru lze načítat formátovaná data a převést je do textu. Oracle Text podporuje velké množství formátů, jako například Microsoft Word, Excel, nebo PDF. Dále je možné nastavit převod znakové sady indexovaných dokumentů do sady, kterou používá databázový server. V případě, že budou indexována pouze textová data, lze filtr vypnout.

Sectioner

Tento modul se používá pro rozdělení dokumentů do sekcí podle struktury dokumentu. Modul lze využít pro strukturované dokumenty jako například HTML nebo XML. Při vyhledávání v indexu pak uživatel může, pomocí operátoru WITHIN, zadat požadavek na vyhledávání pouze v určité sekci.

Lexer

Lexer převádí prostý text na jednotlivé termy. Pomocí parametrů indexu lze nastavit použitý jazyk, jaké mají být použity bílé znaky, případně zda mají být termy převedeny na velká písmena.

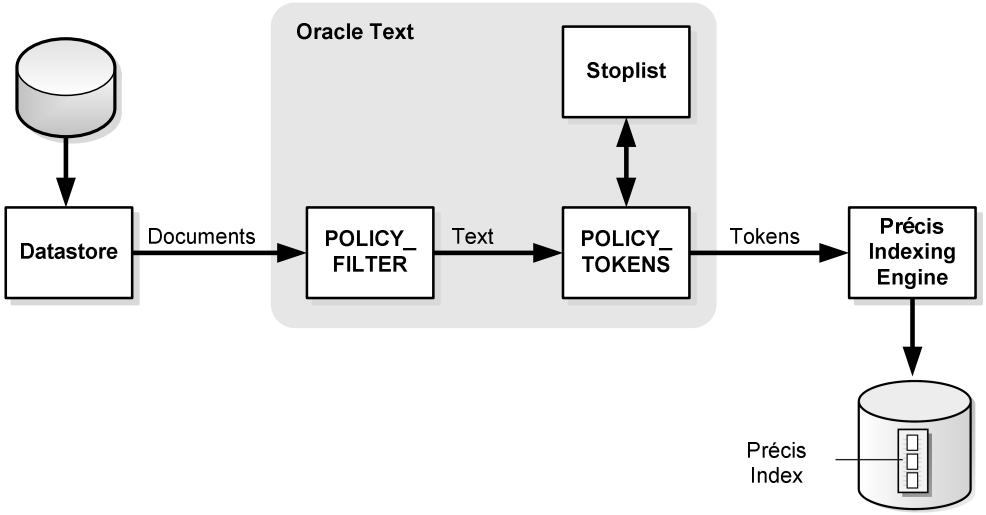
Indexing Engine

Z termů vytvoří index v podobě invertovaného soboru. Zároveň kontroluje, zda indexované termy nejsou na stoplistu.

4.2. Précis index

Proces vytvoření Précis indexu je zobrazen na obrázku 4-2. Nejprve je načten dokument z databáze. Pokud jde o binární data, tedy formátovaný text, použije se filtr pro převod do prostého textu. Poté je dokument převeden na seznam termů a jejich pozice v dokumentu. Protože pozice termů v indexu nejsou potřeba, je tento seznam převeden na množinu termů s počtem výskytů daného termu v dokumentu. Pro převod dokumentu do prostého textu a vytvoření seznamu termů se využívají funkce policy_filter a policy_tokens [12] z Oracle Textu.

Z počtu výskytů termu v dokumentu je spočítána frekvence termu. Celkové ohodnocení termu je dáno součinem frekvence termu (TF) a inverzní frekvence termu (IDF). Tyto veličiny jsou definovány v kapitole 3.4. Výpočet ohodnocení termu však probíhá až při vyhledávání v indexu, protože IDF je závislé na celkovém počtu dokumentů v indexu a počtu dokumentů, ve kterých se daný term vyskytuje. Tyto hodnoty se budou měnit v závislosti na indexovaných dokumentech. Hodnota IDF je tedy vypočítána až při vyhodnocování dotazu.



Obrázek 4-2 Précis index – indexace

5. Implementace

Při implementaci jsem se zaměřil na základní krok vyhodnocení Précis dotazu – tedy na vytvoření textového indexu a nalezení vyhovujících záznamů na základě dotazu.

Index vytvořený pomocí Oracle Text není vhodný pro Précis dotazy, protože tento index umožňuje vyhledávat pouze v datech jedné tabulky. Naproti tomu Précis dotazy potřebují vyhledávat data z různých tabulek. V implementaci Précis indexu může být proto indexováno libovolné množství sloupců tabulek z různých schémat.

5.1. Programovací jazyk

Indexační modul je napsán v jazyce PL/SQL a C++. PL/SQL je překládaný jazyk, vykonávaný přímo jádrem databázového serveru Oracle, ve kterém lze snadno manipulovat s databázovými objekty. V tomto jazyce jsou napsány procedury, které hlídají změny v indexovaných tabulkách a uživatelské rozhraní. V jazyce C++ je naprogramováno vytváření, úprava a vyhledávání v datových strukturách, ve kterých je index uložen. Propojení aplikace v jazyce C++ s databázovým serverem je zprostředkováno pomocí OCI rozhraní.

Nejdříve jsem se pokoušel index implementovat výhradně s použitím jazyka PL/SQL. Hlavní výhodou tohoto řešení by byla přenositelnost. Všechny funkce a procedury by bylo možné spustit na libovolné platformě, pro kterou je Oracle server dostupný. Je pravdou, že kód napsaný v jazyce C++ by měl jít přeložit i na jiných operačních systémech, než pro který byl původně určen, ale přesto zde mohou nastat některé komplikace při překladu.

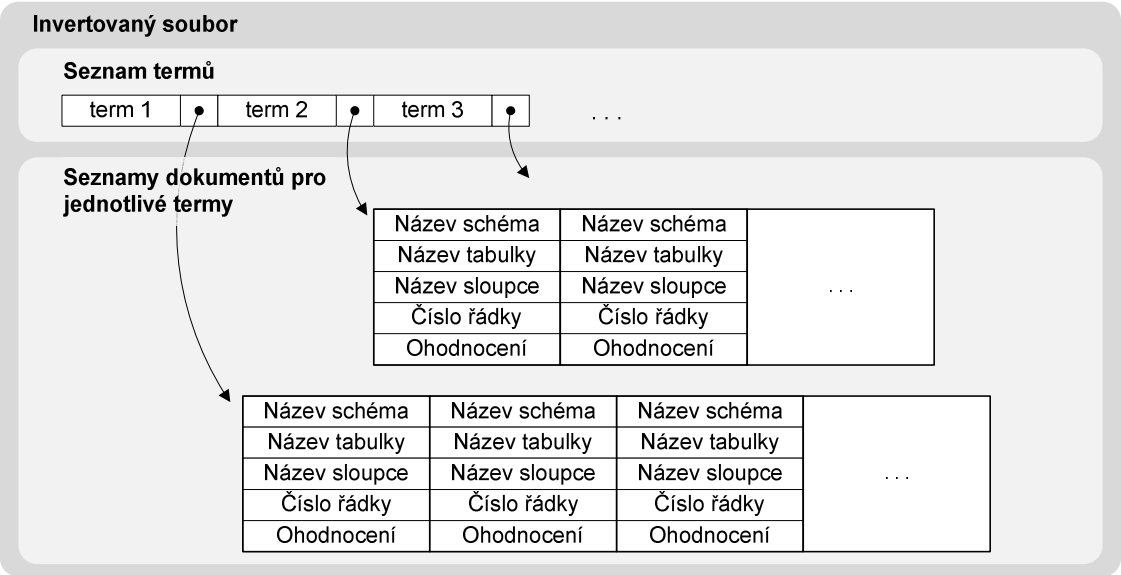
Přestože je jazyk PL/SQL velice mocný nástroj a lze v něm napsat téměř jakoukoli proceduru, potřebnou pro manipulaci s daty v rámci databáze Oracle, ukázalo se, že výkonnostně je mnohem horší, než ekvivalentní algoritmus napsaný v jazyce C++. Na druhou stranu je časově poměrně drahé spouštět externí procedury napsané v C++ z Oracle. Proto jsem se v rámci implementace snažil počet volání externích procedur z PL/SQL minimalizovat.

5.2. Struktura indexu

Index je implementován pomocí invertovaného souboru, který byl popsán v kapitole 3.2. Vzhledem k tomu, že v tomto indexu mohou být dokumenty z různých schémat a tabulek, budeme dokument identifikovat pomocí čtveřice <schéma, tabulka, sloupec, řádek>. Navíc budeme mít spolu s informací o přítom-

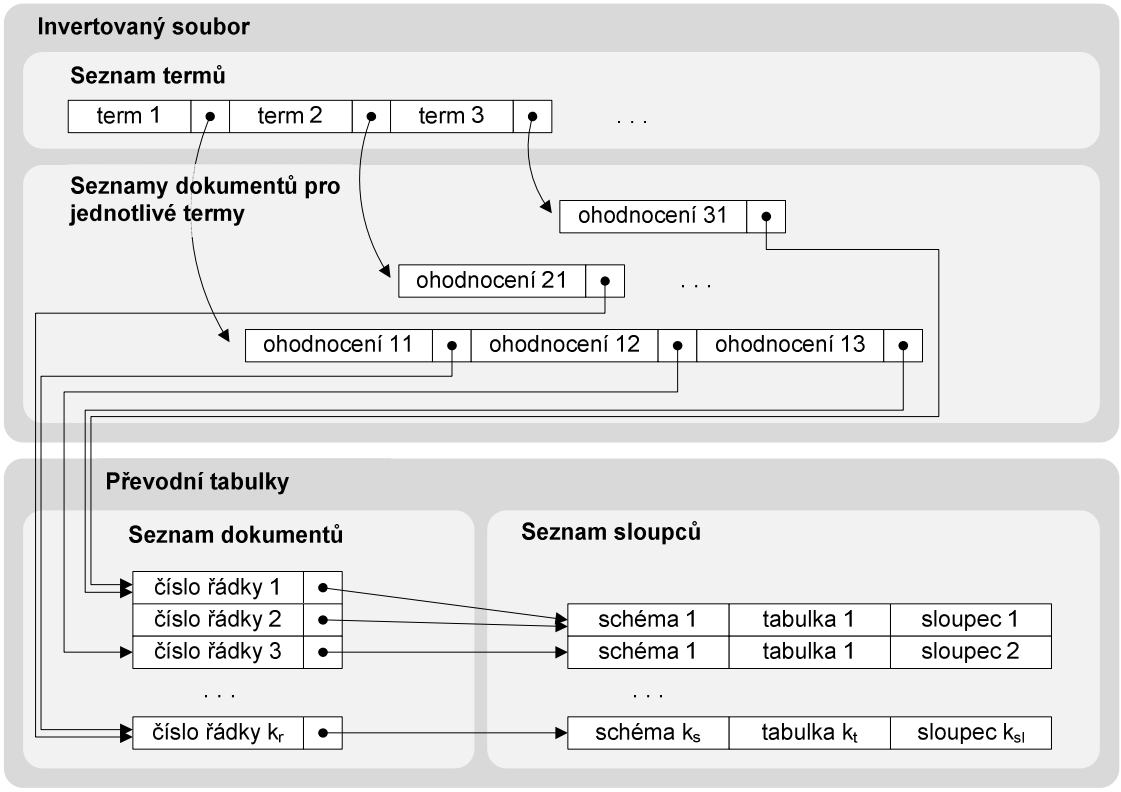
nosti každého termu v dokumentu uloženu hodnotu TF, tedy frekvenci termu. Skutečné ohodnocení dokumentu bude spočítáno jako součin TF a IDF. Hodnota IDF bude spočítána až při vyhodnocování dotazu. Pro zjednodušení budeme hodnotu TF, která je uložena v invertovaném souboru, označovat jako ohodnocení.

Seznamy dokumentů, ve kterých se termy vyskytují, tedy budou obsahovat identifikaci dokumentu a jeho ohodnocení.



Obrázek 5-1 Obecná struktura indexu

Název schématu, tabulky a sloupce mohou mít v databázi Oracle až 30 znaků. Řádek bude identifikován pomocí RowID, což je jednoznačný identifikátor přiřazený každému řádku v databázi. Vzhledem k tomu, že dokumenty se v seznamech výskytů budou opakovat a identifikace pomocí čtveřice <schéma, tabulka, sloupec, řádek> by proto zabrala hodně místa, budeme do invertovaného souboru ukládat pouze ukazatel do převodní tabulky, obsahující seznam dokumentů. Tento ukazatel bude implementován 32 bitovým identifikátorem unikátním v rámci indexu. Protože počet indexovaných sloupců, tedy trojic <schéma, tabulka, sloupec>, bude podstatně menší než počet dokumentů, rozdělíme převodní tabulku na seznam dokumentů a seznam sloupců. Seznam sloupců bude závislý pouze na struktuře databáze. Naopak seznam dokumentů se bude měnit podle toho, jak budeme do databáze přidávat nebo z ní mazat data. Získáme tak strukturu zobrazenou na obrázku 5-2.



Obrázek 5-2 Struktura indexu s převodními tabulkami

5.2.1. Příklad struktury indexu

Strukturu indexu si ukážeme na příkladu databáze CD disků. Předpokládejme, že máme dvě tabulky:

Disk (ID, Interpret, Album, Žánr)

Skladba (ID, Disk_ID, Číslo stopy, Název skladby)

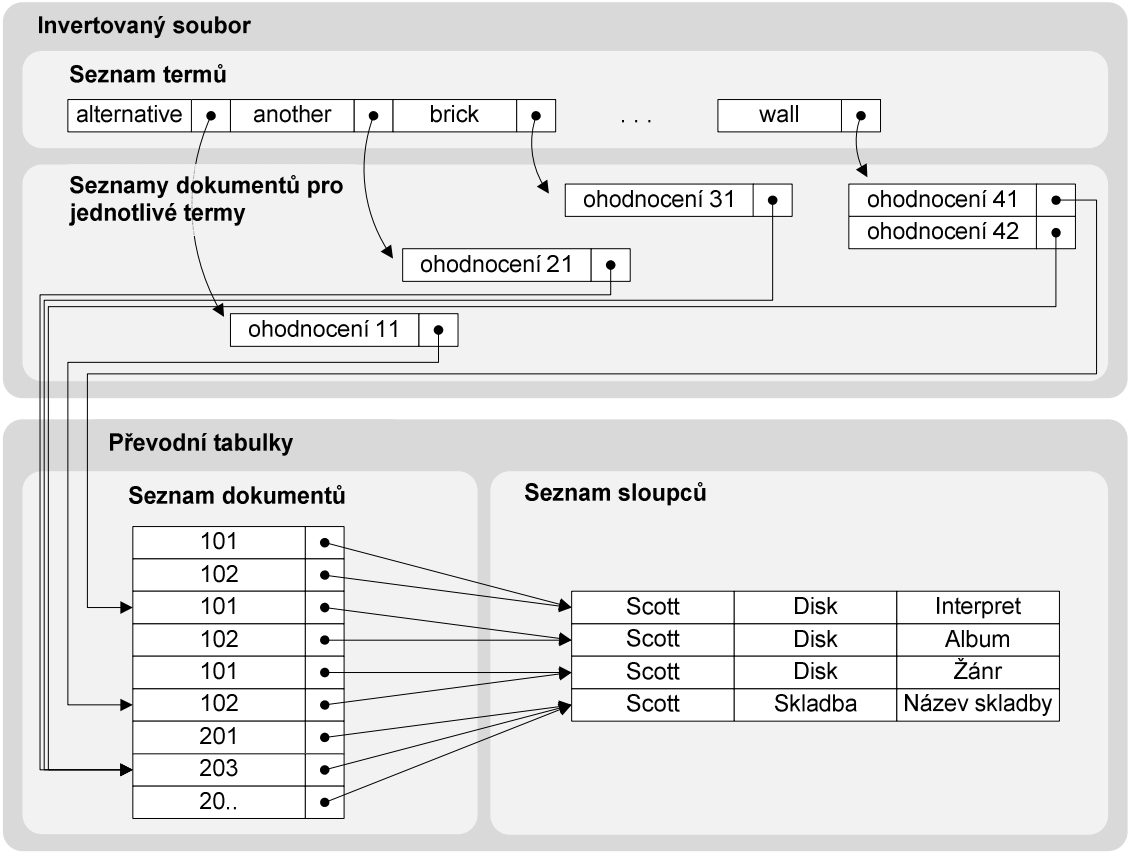
V tabulkách budou tato data:

ID	Interpret	Album	Žánr
101	Pink Floyd	The Wall	Rock
102	Nirvana	Nevermind	Alternative rock

ID	Disk_ID	Číslo stopy	Název skladby
201	101	1	In the Flash?
202	101	2	The Thin Ice
203	101	3	Another Brick in the Wall
204	102	1	Smells Like Teen Spirit
205	102	3	Come As You Are

Tabulka 5-1 Ukázková databáze

Předpokládejme, že jsou tabulky uloženy ve schématu Scott a že jsou indexovány všechny textové údaje. Vytvořený index je zobrazen na obrázku 5-3. Struktura indexu i převodních tabulek bude vždy stejná bez ohledu na to, zda budou indexovány jen data z vlastního schématu, nebo i z cizích.



Obrázek 5-3 Příklad invertovaného souboru

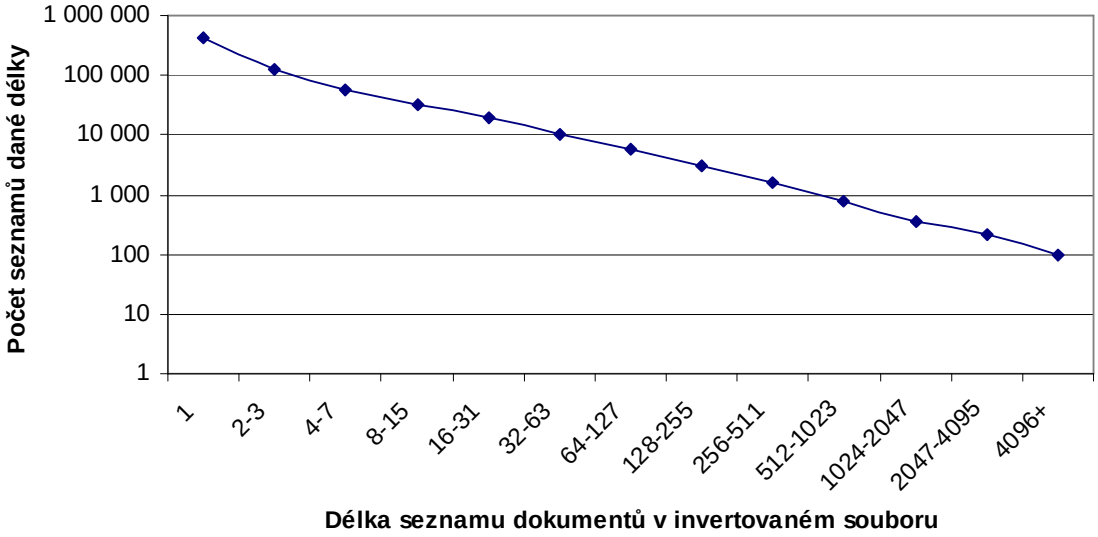
5.3. *Datové struktury použité v indexu*

Základní datovou strukturou použitou v implementaci jsou neredundantní B-stromy s pevnou délkou klíče. Záznamy jsou tedy uloženy nejen v listech stromu, ale také v jeho vnitřních uzlech. Všechny uzly stromu tak mohou mít stejnou strukturu. V indexu jsem použil tyto stromy pro jejich snazší implementaci oproti B^+ nebo B* stromům.

Výhodou pevné délky klíče je snazší vyhledávání záznamu v uzlu stromu. Délka termů, které budou uloženy v stromě, se však může pohybovat v rozmezí 1 až 64 bajtů. Pokud by byly všechny termy uloženy v jednom stromě, tak by zde docházelo k zbytečnému plýtvání pamětí. Proto jsou pro seznam termů vytvořeny 4 B-stromy s délkami klíčů 6, 12, 24 a 64 bajtů. Převážná většina termů bude uložena ve stromech s délkou klíče 6 nebo 12 bajtů a ztráty v důsledku pevné délky klíčů tak budou minimální.

V případě seznamů obsahující výskyty termu v dokumentech již pevná délka klíče nevadí, protože tyto seznamy budou obsahovat jako klíč číslo dokumentu, což je 4 bajtový identifikátor. Délka těchto seznamů může být v řádu tisíců. Proto bude potřeba nastavit velikost uzlu B-stromu tak, abychom umožnili efektivní práci i s takto dlouhými seznamy. Při vyhledávání v indexu nás budou zajímat dokumenty, které mají nejlepší ohodnocení. Aby B-stromy nemusely být procházeny celé, budou pro každý seznam výskytů vytvořeny dva stromy. V prvním budou dokumenty setříděny podle váhy tak, aby bylo snadné nalézt nejlépe ohodnocené dokumenty. Záznamy ve druhém stromě budou setříděny podle čísla dokumentu, aby bylo snadné dohledat ohodnocení pro daný term a libovolný dokument. Tento způsob sice zabírá dvakrát více paměti, než je nezbytně nutné, ale umožňuje nižší počet načtených bloků při vyhledávání v indexu.

V grafu 5-1 jsou zobrazeny počty termů v závislosti na množství jejich výskytů v kolekci dokumentů. Graf je vytvořen z testovací kolekce 32 000 článků české wikipedie. Tato kolekce obsahuje více než 685 000 různých termů. Z nich se 612 000 termů, což je téměř 90%, v kolekci dokumentů vyskytuje méně než 8 krát.



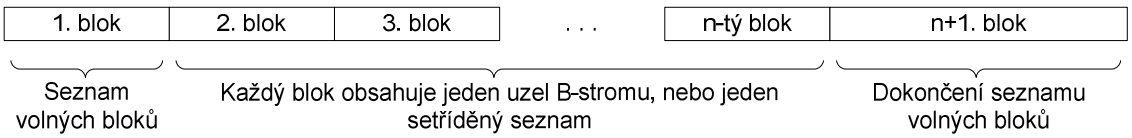
Graf 5-1 Počet seznamů dokumentů v invertovaném souboru v závislosti na jejich délce

Délka seznamu výskytů v invertovaném souboru u většiny termů bude obsahovat pouze několik záznamů. Vytvořené B-stromy by však zabíraly místo na disku pro dva uzly. Z tohoto důvodu se v indexu pro krátké seznamy výskytů používají setříděné seznamy. V tomto seznamu budou záznamy setříděny pouze podle čísla dokumentu. Výhodou setříděných seznamů je jejich velice snadná implementace a zároveň práce s nimi bude rychlejší než s B-stromy. U těchto seznamů se používá blokový přístup, podobně jako u B-stromů, takže seznam bude

možné snadno upravovat. Seznam tedy bude zabírat na disku vždy celý blok. Index umožňuje použití setříděných seznamů s různou délkou. Seznam tak zabere blok, který je nejmenší větší, než je délka seznamu.

5.4. Implementace datových struktur

Všechna data indexu jsou uložena v databázi v záznamech typu BLOB. Počet těchto záznamů je dán parametry při vytváření indexu. Každý takovýto záznam je rozdělen na bloky, jejichž velikost je také dána parametry při vytváření indexu. Tyto parametry po vytvoření indexu již není možné měnit. Bloky každého záznamu jsou očíslovány a první blok má číslo 1. V BLOBu může být až 2^{32} - 1 bloků. Pokud je velikost jednoho bloku 8kB, pak lze teoreticky uložit až 32TB dat do každého BLOBu.



Obrázek 5-4 Obecná struktura BLOBu

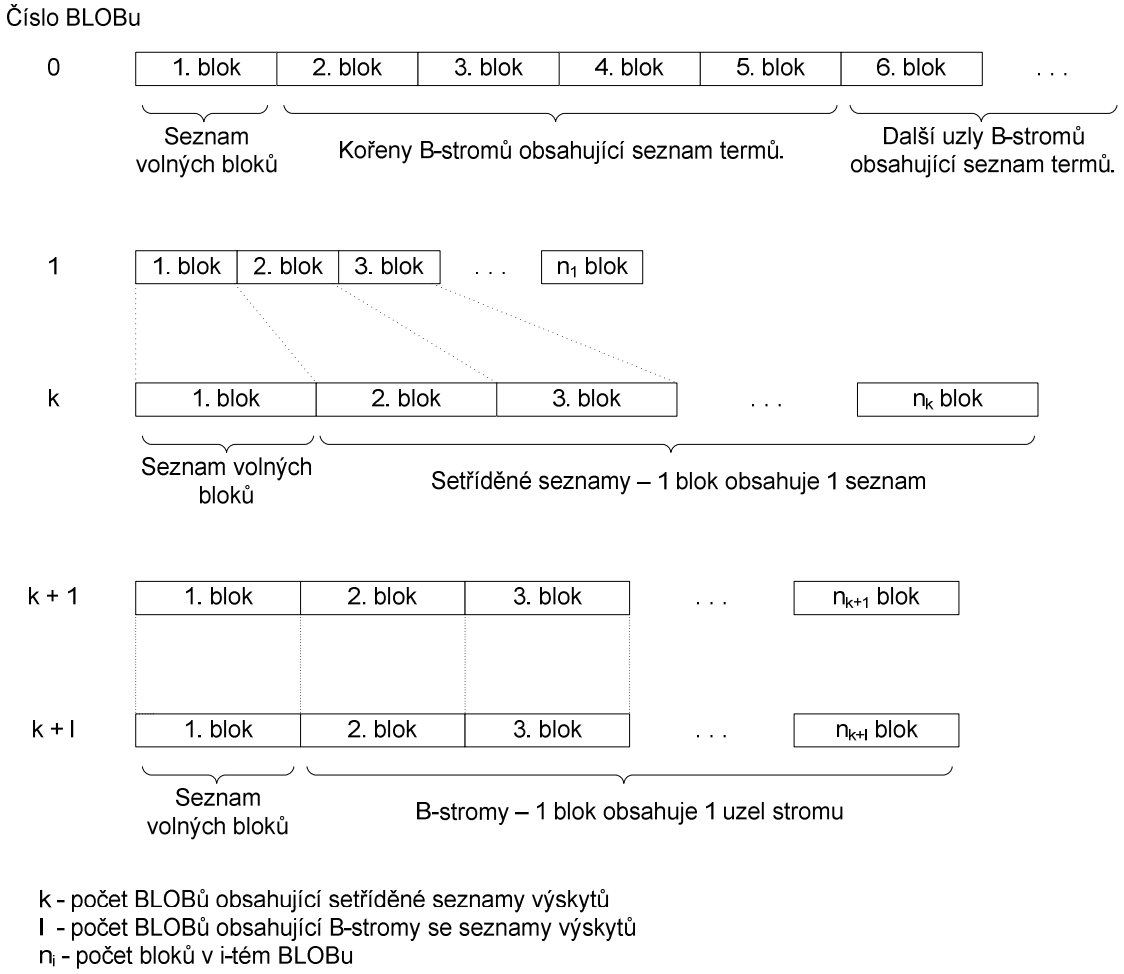
V indexu jsou použity dvě datové struktury: B-stromy a setříděné seznamy. V každém bloku je buď jeden uzel stromu, nebo jeden seznam. Jeden BLOB obsahuje všechny bloky stejného typu. Setříděné seznamy tedy nemohou být uloženy ve stejném BLOBu jako stromy. Setříděné seznamy se používají pro výskyty dokumentů, pokud jejich počet je dostatečně malý, aby se vešel do jednoho bloku.

Protože je seznam termů rozdělen podle délek termů, bude index obsahovat celkem 4 B-stromy se seznamem termů. Tyto stromy budou vždy uloženy v nultém BLOBu a jejich kořeny budou uloženy ve 2. až 5. bloku.

Dalších *k* BLOBů bude obsahovat setříděné seznamy. V indexu lze použít více BLOBů se setříděnými seznamy, ale v každém z nich musí být uloženy bloky s jinou délkou. Z implementačních důvodů nemá význam, aby index obsahoval dva BLOBy pro setříděné seznamy se sejnou délkou bloku, protože ten druhý BLOB by zůstal prázdný. Počet BLOBů se seznamy a délky jednotlivých bloků lze definovat při vytváření indexu. Pro vyšší rychlost synchronizace a vyhledávání v indexu je vhodné použít více setříděných seznamů délky od jednotek až po stovky záznamů.

Index dále obsahuje *l* BLOBů, ve kterých budou uloženy B-stromy. Jejich počet, stejně jako velikost bloků v nich uložených, je dán parametrem při vytváření indexu. Počet těchto BLOBů není omezen. Všechny bloky s B-stromy ve všech BLOBech však musejí mít stejnou velikost.

Defaultní hodnoty pro výše zmíněné parametry jsou popsány v kapitole 5.5.3. Struktura indexu je zobrazena na obrázku 5-5.



Obrázek 5-5 Struktura indexu

5.4.1. Volné bloky

První blok v každém BLOBu obsahuje seznam volných bloků. Pakliže se do prvního bloku nevejde celý seznam, pak tento seznam pokračuje za posledním blokem použitým pro uložení B-stromů, respektive setříděných seznamů výskytů. Velikost seznamu uloženého za posledním blokem není vázána na velikost bloků – jeho velikost není omezena. Důvodem uložení seznamu na konec BLOBu, místo do volných bloků, je, že pokud jsou použity bloky malé velikosti, pak by tento seznam mohl obsadit velké množství bloků. Pak by načtení a uložení seznamu vyžadovalo více přístupů na disk. Způsobem, který jsem použil v implementaci, je možné celý seznam přečíst nebo zapsat pomocí jednoho nebo dvou přístupů na disk – v závislosti na tom, zda se celý seznam vejde do prvního bloku, nebo bude část seznamu uložena na konci BLOBu.

Před zahájením úprav indexu je celý seznam volných bloků načten do paměti a z BLOBu je smazán. Tento seznam tvoří haldu, na jejímž vrcholu je blok s nejnižším číslem. Při zaplňování prázdných bloků jsou tedy nejprve použity ty, co jsou blíže k začátku BLOBu. Po dokončení úprav v indexu je opět seznam volných bloků zapsán do BLOBu.

Tabulka 5-3 popisuje formát dat v prvním bloku. Dokončení seznamu volných bloků na konci BLOBu již obsahuje pouze čísla volných bloků.

Pozice od začátku	**Velikost**	**Popis**
0	2B	Počet použitých záznamů (volných bloků) v prvním bloku
2B	4B	Číslo bloku, kde začíná seznam volných bloků, který se již nevešel do prvního bloku. Pokud se seznam volných bloků vejde celý do prvního bloku, pak je tato hodnota 0.
6B	4B	Počet použitých záznamů (volných bloků) v seznamu na konci BLOBu. Pokud se seznam volných bloků vejde celý do prvního bloku, pak je tato hodnota 0.
10 + 4(k-1)	4B	Číslo k-tého volného bloku

Tabulka 5-2 Formát dat v prvním bloku každého BLOBu

5.4.2. Setříděné seznamy

Setříděné seznamy se používají pouze pro malý počet výskytů dokumentů. Pokud se v průběhu úprav indexu zvětší délka tohoto seznamu, může být seznam přesunut do BLOBu s většími bloky. Pokud je počet dokumentů tak velký, že by se nevešel do jednoho bloku, pak je použit B-strom. Výhoda setříděného seznamu je v tom, že lze celý přečíst nebo zapsat jedním přístupen na disk. Navíc nemusí obsahovat ukazatele, které jsou použity v B-stromech, takže se do stejného bloku vejde více záznamů.

Na začátku seznamu je uložen počet záznamů v tomto seznamu, který zabírá 2B. Za ním pak následují dvojice <*číslo dokumentu, TF*>, které zabírají 4+2B. Velikost bloku pro setříděné seznamy musí být dělitelná číslem 2+6k, kde k je maximální počet záznamů v seznamu.

5.4.3. B-stromy

B-stromy se používají pro seznam termů a výskyty termu v dokumentech. Klíč B-stromu má pevnou délku a proto každý uzel má daný minimální a maximální počet záznamů, který je dán velikostí bloku. Při práci se stromem se používá vyrovnávací paměť, kam jsou ukládány načtené uzly. Tato paměť se používá nejen

při úpravách stromu, ale také při čtení. Detailní funkce vyrovnávací paměti je popsána v sekci 5.4.4.

Při vytváření nového stromu se použije metoda *MakeTree*, která setřídí záznamy v paměti a pak z nich vytvoří strom. Tento způsob vytvoření B-stromu je podstatně rychlejší, než postupné vkládání jednotlivých záznamů do stromu, protože každý uzel je zapsán na disk pouze jednou a minimalizuje se přesouvání dat uzlů v paměti.

Formát dat

Všechny uzly v B-stromě mají stejný formát. V každém uzlu, kromě kořene, musí být alespoň $\lfloor b/2 \rfloor$ záznamů, kde b je maximální počet záznamů v uzlu. Tím je zajištěno, že celkové využití paměti bude alespoň 50%.

B-strom obsahuje	Část záznamu	Celková velikost	Velikost	Popis
Termy	Klíč	6 - 64	6, 12, 24, 64	term (pokud je kratší, než je velikost klíče, pak je term doplněn nulami); jsou používány 4 stromy s různou délkou termu
	Metadata	12	2	počet dokumentů, ve kterých se term vyskytuje
			2	číslo LOB objektu, ve kterém jsou uloženy výskyty termu v dokumentech
			4	číslo bloku v LOB objektu obsahující buď setříděný seznam, nebo kořen B-stromu dokumentů setříděných podle čísla dokumentu
			4	číslo bloku v LOB objektu obsahující kořen B-stromu dokumentů setříděných podle váhy. Pokud jsou dokumenty uloženy v setříděném seznamu, pak je tato hodnota nula.
Dokumenty (setříděné podle čísla dokumentu)	Klíč	4	4	číslo dokumentu
	Metadata	2	2	frekvence termu (TF)
Dokumenty (setříděné podle váhy)	Klíč	6	2	frekvence termu (TF)
			4	číslo dokumentu
	Metadata	-	-	-

Tabulka 5-3 Popis záznamů uložených v B-stromech

Na začátku každého uzlu je uložen počet záznamů v uzlu. Tato hodnota zabírá 2 bajty. Pak následuje ukazatel na další uzel a záznam tvořený klíčem a metadaty. Mezi každýma dvěma záznamy je uložen ukazatel a za posledním záznamem je poslední ukazatel. Velikost ukazatele je 4 bajty. Jeho hodnota je číslo bloku, ve kterém je uložen příslušný uzel stromu. Ve stromě se vyhledává pouze pomocí klíče. Metadata jsou jen kopírována a B-strom je nijak neinterpretuje. Velikosti klíče a metadat závisí na použití stromu a jsou popsány v tabulce 5-3.

5.4.4. Vyrovnávací paměť

Nejpomalejší činností jak při vyhledávání v indexu, tak při jeho úpravě jsou přístupy na disk. Rychlost disku je přibližně 1 000 až 1 000 000 krát pomalejší, než přístup do operační paměti. Proto je výhodné během úprav indexu co nejvíce dat ukládat pouze do paměti a na disk je zapsat až ve chvíli, kdy množství obsazené paměti překročí určitou mez. Z toho důvodu je v algoritmech využit odkládací prostor, kam jsou ukládána data, která by měla být zapsána na disk. Až při zaplnění určeného prostoru jsou data na disk skutečně zapsána.

Během aktualizace indexu se vyrovnávací paměť používá u B-stromů, lob objektů, seznamu termů a seznamu výskytů dokumentů. Při vyhledávání v indexu se používá pouze pro načtené uzly B-stromu.

B-stromy

Uzly všech B-stromů, se kterými pracuje, jsou ukládány do paměti. Počet uzlů, které mohou být naráz uloženy v paměti, je předem dán a je pro všechny stromy stejný. Paměť je však alokována až ve chvíli, kdy je požadována. U každého načteného uzlu je příznak, kdy byl naposledy použit a zda byl změněn. Pokud je paměť plná a je potřeba načíst další uzel, pak je z paměti odstraněn ten uzel, který se nejdéle nepoužíval. Uzel je zapsán na disk pouze v případě, že byl změněn, jinak může být z paměti smazán ihned. Obdobně při ukončení práce se stromem budou zapsány na disk pouze ty uzly, které byly změněny.

LOB objekty

Pro každý LOB objekt je alokována paměť pevné délky. Velikost této paměti je standardně 256kB. Tato paměť se používá pouze při úpravách indexu. Je zde využito toho, že pokud jsou do indexu přidávána nová data, pak jsou ve většině případů přidávána na konec LOBu. Uzly, které by měly být zapsány na konec LOBu, jsou zapsány pouze do vyrovnávací paměti. Obsah paměti je pak nejednou zapsán do databáze až ve chvíli, kdy je celá paměť obsazena. Zápis se provede jedním voláním Oraclu a navíc data mohou být zapsána sekvenčně. To výrazně zrychlí přidávání nových dat.

5.5. Databázové objekty

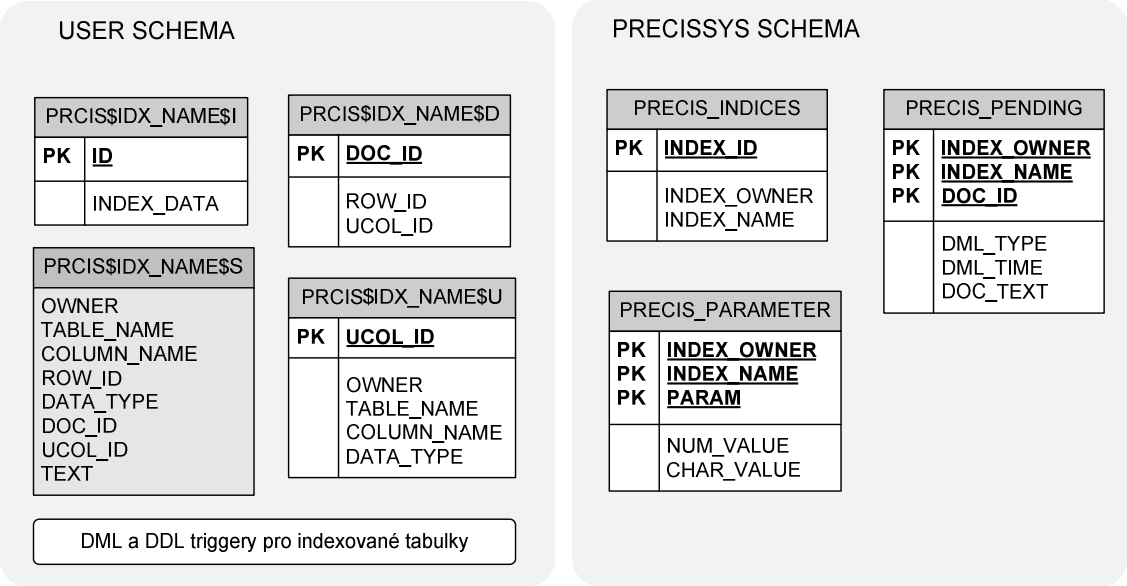
Každý vytvořený Précis index musí být pojmenován. Jméno může obsahovat nejvýše 15 znaků, protože v databázi budou vytvořeny další objekty, jako například tabulky a sekvence, které budou ve svém názvu obsahovat také název indexu. Aby se předešlo případným problémům, je název indexu převeden na velká písmena. Indexy s názvy „precis_idx“ a „PRECIS_IDX“ budou považovány za totožné. Název smí obsahovat pouze písmena, číslice a znak „_“. Index nesplňující tyto požadavky nebude vytvořen.

Fulltextové indexy je výhodné upravovat v dávkách a ne při každé změně libovolného dokumentu. Proto se při změně dokumentu uloží do databáze pouze informace o změně a úprava indexu proběhne až později. Hlavní nevýhoda tohoto způsobu je, že při vyhledávání v indexu se může stát, že hledaný dokument není nalezen, protože dosud nebyl indexován. Na druhou stranu však je tento způsob správy indexu podstatně rychlejší a nejsou brzděny úpravy jednotlivých dokumentů. Případ, že by byl vrácen dokument, který byl již z databáze odstraněn, nastat nemůže, protože u všech nalezených dokumentů se kontroluje, zda jsou stále přítomny v databázi.

Při vytvoření indexu je pro každou indexovanou tabulku vytvořen trigger, který je spouštěn při úpravě tabulky. Informace o změnách trigger ukládá do tabulky PRECIS_PENDING. Při synchronizaci indexu se postupně projde tato tabulka a změny v dokumentech se zapíší do indexu.

V indexu jsou všechny dokumenty označeny identifikátorem. Tento identifikátor je implementován 32 bitovým číslem a je jednoznačný v rámci daného indexu. Pro převod z čísla dokumentu na konkrétní záznam v databázi se používají převodní tabulky PRECIS$*IDX_NAME*$U a PRECIS$*IDX_NAME*$D, kde *IDX_NAME* je název indexu. V tabulce PRECIS$*IDX_NAME*$I je uložen vlastní index. Tyto 3 tabulky jsou určeny pouze pro jeden index.

Další používanou tabulkou je PRECIS_PARAMETER, kde jsou uloženy parametry indexu. Většina těchto parametrů je definována při vytvoření indexu a jejich změna by způsobila nefunkčnost indexu. Tabulka PRECIS_INDICES obsahuje seznam vytvořených Précis indexů.



Obrázek 5-6 Schéma databázových objektů

5.5.1. Triggery

V aplikaci se používají dva typy triggerů – pro DML příkazy nad indexovanými tabulkami a triggery hlídající DDL příkazy TRUNCATE TABLE a DROP TABLE.

Přidáním sloupce do indexu, je spuštěna procedura, která vytvoří trigger pro danou tabulku. Trigger je typu „AFTER INSERT OR UPDATE OR DELETE FOR EACH ROW“. Pro každou tabulku, která obsahuje alespoň jeden indexovaný sloupec, bude vytvořen jeden trigger.

Trigger bude mít název ve tvaru PRECIS_*INDEX-ID*_*TABLE-ID*_TRG, kde *TABLE-ID* je číslo identifikující tabulku načtené z pohledu ALL_OBJECTS. *INDEX-ID* je číslo Précis indexu. Tím je zajištěno, že nemůže dojít k vytvoření triggerů se stejným názvem i v případě, že pro jednu tabulku bude vytvořeno více indexů. Trigger spustí odpovídající proceduru z balíčku PRECIS_DML pro každý indexovaný sloupec. Tato procedura zapíše provedenou operaci do tabulky PRECIS_PENDING.

Pokud je dokument dostatečně krátký, bude jeho text uložen přímo v tabulce PRECIS_PENDING. Při synchronizaci indexu pak není potřeba načítat krátké dokumenty z tabulek a synchronizace může proběhnout rychleji. V případě potřeby, například z důvodu bezpečnosti, je možné tuto funkci deaktivovat nastavením maximální velikosti takto ukládaných dokumentů na nulu.

		Typ DML operace v tabulce PRECIS_PENDING	
		před spuštěním triggeru	po spuštění triggeru
Úprava dokumentu	Vytvoření	-	Insert
	Smazání	- Insert Update	Delete - Delete
	Změna	- Insert Update	Update Insert Update

Tabulka 5-4 Změna v tabulce PRECIS_PENDING v závislosti na předchozí úpravě dokumentu

Každý dokument může mít v tabulce PRECIS_PENDING nejvýše jeden záznam o provedené DML operaci. Vzhledem k tomu, že jeden dokument může být upraven vícekrát mezi synchronizacemi indexu, je PRECIS_PENDING upraveno podle tabulky 5-4. Znamená to například, že pokud byl dokument přidán do databáze, tak bude do tabulky PRECIS_PENDING přidána informace o vloženém dokumentu. Pokud bude dokument odstraněn dříve, než proběhne synchronizace indexu, tak bude smazán záznam z tabulky PRECIS_PENDING o jeho vytvoření. Při synchronizaci indexu již nebude nikde informace, že daný dokument někdy existoval. Pokud však před smazáním dokumentu proběhne synchronizace indexu, tak bude do tabulky PRECIS_PENDING zapsána informace, že daný dokument byl smazán a má být z indexu odstraněn.

Trigger pro DDL příkazy bude existovat pouze jeden v uživatelském schématu bez ohledu na počet vytvořených indexů. Název triggeru bude PRECIS_TBLDDL_TRG. Při spuštění tohoto triggeru se nejprve zjistí, zda je daná tabulka indexována pomocí Précis indexu. V případě že ano, bude zapsán do tabulky PRECIS_PENDING seznam všech řádek, které byly v tabulce před jejím smazáním. Při zápisu se postupuje stejně, jako kdyby byl obsah tabulky odstraněn příkazem „DELETE FROM table“.

Příklad DML triggeru

```
CREATE OR REPLACE TRIGGER TEST."PRECIS_214_44878_TRG"
  AFTER INSERT OR UPDATE OR DELETE ON "TEST"."FREEDB_TRACK"
  FOR EACH ROW
BEGIN
  IF DELETING THEN
    precis_dml.dml_delete(
      'FREEDB_IDX', 'TEST', 'FREEDB_TRACK', :old.ROWID);
  ELSIF INSERTING THEN
    IF :new."TTITLE" IS NULL OR lengthb(:new."TTITLE") < 256
    THEN
      precis_dml.dml_insert(
        'FREEDB_IDX', 2, :new.ROWID, :new."TTITLE");
    ELSE
      precis_dml.dml_insert('FREEDB_IDX', 2, :new.ROWID);
    END IF;
  ELSE
    IF :new."TTITLE" <> :old."TTITLE" THEN
      IF lengthb(:new."TTITLE") < 256 THEN
        precis_dml.dml_update(
           'FREEDB_IDX', 2, :new.ROWID, :new."TTITLE");
      ELSE
        precis_dml.dml_update('FREEDB_IDX', 2, :new.ROWID);
      END IF;
    END IF;
  END IF;
END;
/
```

Příklad DDL triggeru

```
CREATE OR REPLACE TRIGGER TEST.precis_tblddl_trg
  BEFORE DROP OR TRUNCATE
  ON SCHEMA
DECLARE
  idx_names precis_idx.t_index_names;
BEGIN
  precis_idx.get_index_names(user, idx_names);
  FOR i IN 1..idx_names.COUNT() LOOP
    precis_dml.dml_truncate(
        idx_names(i),
        ora_dict_obj_owner,
        ora_dict_obj_name);
  END LOOP;
END;
/
```

5.5.2. Tabulka PRECIS_PENDING

Tabulka PRECIS_PENDING obsahuje informace o změnách v indexovaných tabulkách od předchozí synchronizace indexu. Tabulka je společná pro všechny indexy a je uložena v schématu PRECISSYS. Uživatel má k tabulce přístup pouze prostřednictvím balíčku PRECIS_PEND.

Záznamy v tabulce vytvářejí triggery při úpravách v indexovaných tabulkách. Tyto údaje jsou pak použity při synchronizaci indexu a následně smazány během synchronizace.

5.5.3. Tabulka PRECIS_PARAMETER

Tato tabulka obsahuje hodnoty parametrů všech indexů. Stejně jako předchozí tabulka je i tato společná pro všechny indexy. Přístup k ní je řešen pomocí balíčku PRECIS_PARAM. Popis všech parametrů je v tabulce 5-5.

Název parametru	**Výchozí hodnota**	**Popis**
token_node_size	8132	velikost bloku dat B-stromu, ve kterém jsou uloženy termy
index_node_size	8132	velikost bloku dat B-stromu, ve kterém jsou uloženy výskyty termu
index_blob_count	3	počet BLOBů, ve kterých jsou uloženy B-stromy (BLOB obsahující seznamy termů nepočítá)
small_index_blob_count	6	počet BLOBů, ve kterých jsou uloženy seznamy výskytů termu
small_index_node_size*N*	38, 98, 428, 1706, 4262, 8132	velikost bloku dat seznamu výskytů termu musí být definováno přesně small_index_blob_count hodnot a musí platit: node_size*N* < node_size*N+1* Velikost bloku musí být dělitelná číslem 2+6*k*, kde *k* je maximální počet dokumentů, které mohou být v seznamu uloženy. Pro blok velikosti 38 to znamená, že může obsahovat nejvýše 6 dokumentů.
document_count	0	celkový počet indexovaných dokumentů (tato hodnota se automaticky aktualizuje při synchronizaci indexu). Hodnota se používá pro výpočet IDF a následně ohodnocení dokumentů při vyhledávání v indexu.
paice_coef_and	900	Paiceův koeficient pro výpočet ohodnocení konjunkce termů (hodnota je uváděna v tisícinách a může být kdykoli změněna)

Název parametru	Výchozí hodnota	Popis
paice_coef_or	700	Paiceův koeficient pro výpočet ohodnocení disjunkce termů (hodnota je uváděna v tisícinách a může být kdykoli změněna)
multi_schema_index	0	Označuje, zda v indexu mohou být tabulky z různých schémat. Změna se provádí pomocí funkcí set_as_single_schema resp. set_as_mutli_schema
suffix_search_min_token_len	4	Pokud délka hledaného termu bude větší nebo rovna hodnotě tohoto parametru, pak budou pro vyhledávání dokumentů použity i termy s libovolnou příponou

Tabulka 5-5 Parametry indexu

5.5.4. Tabulka PRECIS_INDEX

V této tabulce je uložen seznam všech Précis indexů všech uživatelů. Každému indexu je přiřazen jednoznačný identifikátor, který se používá například v názvu triggeru. Tato tabulka také slouží pro kontrolu, zda je zadaný index v databázi definován.

Uživatel k tabulce nemá přímý přístup, ale muže k ní přistupovat pomocí funkcí z balíčku PRECIS_IDX.

5.5.5. Tabulky PRECISIDX_NAMEU a PRECISIDX_NAMED

Každý indexovaný sloupec z databáze má v tabulce $U jeden řádek. V této tabulce je uložen název indexované tabulky, sloupce a tzv. „universal column id“, což je číselný identifikátor sloupce, který je jednoznačný v rámci daného indexu.

Identifikátor sloupce se pak používá v tabulce $D, která přiřazuje každému dokumentu identifikátor. Tato tabulka bude obsahovat tolik řádek, kolik je indexovaných dokumentů. Číslo dokumentu je opět jednoznačné pouze v rámci indexu. Pro nalezení dokumentu v tabulce se používá RowID. RowID je interní identifikátor databáze Oracle, který odkazuje na fyzické umístění záznamu v datovém úložišti. Použití tohoto identifikátoru pro přístup k datům tabulky je rychlejší, než použití primárního klíče.

5.5.6. Tabulka PRECISIDX_NAMEI

Tato tabulka obsahuje vlastní index. Počet záznamů v této tabulce je konstantní a je dán parametry při vytváření indexu. V první řádce jsou B-stromy s termy a v ostatních jsou seznamy nebo B-stromy s výskyty jednotlivých termů v dokumentech.

Použité B-stromy předpokládají pevnou délku klíče. To má výhodu v rychlejším vyhledávání záznamu v uzlu stromu – je zde použito binární vyhledávání. Na druhou stranu je problém s indexováním termů, které mají různou délku. Při tokenizaci textu se vytvářejí termy o maximální délce 64 bajtů. V indexu se používá znaková sada databáze, což v případě výchozího nastavení v Oracle Database 10g XE bude UTF-8. V případě použití tohoto, nebo jiného více bajtového, kódování může být maximální délka termu ve znacích kratší.

5.5.7. Pohled PRECISIDX_NAMES

Pohled je vytvořen spojením tabulek PRECISIDX_NAMEU a PRECISIDX_NAMED a umožňuje snazší vyhledávání v indexu. Seznam sloupců pohledu a jeho využití je popsáno v uživatelské příručce.

5.6. Aktualizace indexu

Úpravy indexu probíhají spouštěním synchronizace indexu. Při úpravě v indexovaných tabulkách se pouze zapisují změny do tabulky PRECIS_PENDING. Díky tomu jsou změny v tabulkách jen minimálně časově ovlivněny. Vlastní úprava indexu, která je poměrně dosti časově náročná, tak může proběhnout později, například když databázový server není tolik vytížen. Navíc indexace většího množství dokumentů je výrazně rychlejší, než kdyby byl každý dokument indexován zvlášť. Při indexaci více dokumentů jsou změny v seznamech výskytů dokumentů ukládány do paměti a teprve když množství takto obsazené paměti překročí určitou hranici, je část zapsána do databáze. Vzhledem k tomu, že přístupy na disk jsou řádově pomalejší než práce s daty v paměti, lze tímto způsobem výrazně zkrátit čas potřebný pro úpravu indexu.

Nejprve je načten seznam všech termů z indexu. Jako další se projde tabulka PRECIS_PENDING a načtou se čísla dokumentů, které byly smazány nebo upraveny. Všechny výskyty těchto dokumentů musejí být z indexu odstraněny. Následně jsou z tabulky PRECIS_PENDING odstraněny záznamy o smazaných dokumentech. Dokumenty, které byly označeny jako upravené, se označí jako nové.

Poté, co jsou všechny výskyty upravených nebo smazaných dokumentů odstraněny z indexu, projde se znovu tabulka PRECIS_PENDING a načtou se

všechny nové nebo aktualizované dokumenty. Tyto dokumenty jsou tokenizovány a vloženy do indexu.

Při úpravě indexu je celý seznam termů načten do paměti a z databáze jsou všechny termy smazány. Během aktualizace indexu je celý seznam uložen pouze v paměti. Při přidávání nových termů v průběhu úprav tedy nemusíme změny ukládat na disk. Po dokončení úprav indexu je pak celý seznam zapsán pomocí funkce *MakeTree*, která vytvoří B-strom sekvenčně ze setříděné posloupnosti.

Protože externí procedura, upravující index, běží s právy uživatele, který ji spustil, musejí být všechny úpravy a přístupy k tabulce PRECIS_PENDING řešeny prostřednictvím volání funkcí z balíčku PRECIS_PEND. Obdobně pro načítání a úpravu parametrů z tabulky PRECIS_PARAMETER jsou použity funkce z balíčku PRECIS_PARAM. Tyto funkce jsou spouštěny pomocí tzv. callback volání. U tohoto typu volání není potřeba navazovat z externí procedury nové spojení s databází, ale využije se již vytvořené spojení. Navíc volání probíhá v rámci stejné transakce, ve které je spuštěna externí procedura. Více informací a omezení při používání callback volání lze nalézt v [10].

Celá procedura aktualizace indexu probíhá v rámci jedné transakce. V případě, že by aktualizace nebyla dokončena například z důvodu chyby hardware, tak se po restartu serveru index vrátí do stavu před spuštěním aktualizace.

5.6.1. Mazání dokumentů z indexu

Mazání dokumentů je rozděleno na dvě části. Nejprve se smažou dokumenty ze setříděných seznamů, pak jsou odstraněny z B-stromů.

Mazání dokumentů ze seznamů je urychleno tím, že bloky se seznamy nejsou načítány jednotlivě, ale po větších množstvích. Tím se sníží počet přístupů na disk. Všechny seznamy se projdou, a pokud obsahují číslo smazaného nebo upraveného dokumentu, tak se záznam ze seznamu odstraní. V případě, že byl alespoň jeden dokument odstraněn, musí se ještě poznamenat nový počet dokumentů v seznamu. Po projití všech seznamů musí být opraveny počty dokumentů, které jsou poznamenány u termů. Pokud počet dokumentů v seznamu klesne na nulu, bude seznam i term z indexu odstraněn.

Poté, co jsou odstraněny dokumenty ze setříděných seznamů je potřeba ještě projít B-stromy. Protože dokumenty jsou uloženy ve dvou stromech, je potřeba je odstranit z obou dvou. Nejprve se pokusíme smazat dokument ze stromu, kde jsou záznamy tříděné podle čísla dokumentu. V případě, že je dokument nalezen a odstraněn, smažeme dokument ještě z druhého stromu, kde jsou záznamy tříděny podle váhy dokumentu. Abychom záznam ve druhém stromě

mohli najít, potřebujeme hodnotu váhy, kterou jsme získali při mazání dokumentu z prvního stromu. Při tomto mazání se může stát, že počet dokumentů klesne na nulu. V tom případě smažeme oba B-stromy i term.

5.6.2. Přidání nových dokumentů do indexu

Pro přidání nového dokumentu je nejprve potřeba převést text na jednotlivé termy. Tím získáme pro každý nový dokument seznam termů. V druhé části pak vytvoříme seznamy dokumentů k jednotlivým termům. Během tohoto převodu je potřeba co nejvíce dat držet v paměti, aby bylo vytvoření indexu co nejrychlejší.

Vytváření indexu je prováděno v externí proceduře a tokenizace dokumentů probíhá pomocí callback volání. Protože tato zpětná volání jsou poměrně dost časově náročná, je během jednoho callbacku tokenizováno více dokumentů najednou. Výsledek tokenizace je uložen do jednoho dočasného BLOBu, který je předán zpět do externí procedury.

Výsledek tokenizace, tedy termy a jejich výskyty v dokumentech, je uložen do vyrovnávací paměti. Pro práci s touto pamětí se používá objekt TokenDocCache. Množství takto uložených dat je dáno parametrem programu. Do paměti se ukládají termy i seznamy dokumentů pro daný term. Termy jsou v paměti vždy všechny a jsou zapsány na disk až při dokončení synchronizace indexu. Velikost paměti obsazené seznamem termů je závislá pouze na počtu termů v indexu a jejich průměrné délce. I v případě, že bude v indexu 1 000 000 termů, velikost obsazené paměti bude přibližně 20 až 25 MB.

Velikost obsazené paměti obsahující seznamy dokumentů pro dané termy bude růst s počtem nově indexovaných dokumentů. Délky seznamů budou značně rozdílné a mohou se pohybovat od jednoho záznamu až po počet nově indexovaných dokumentů. Nároky na paměť tedy mohou být velmi vysoké. Z tohoto důvodu je maximální množství takto zabrané paměti omezeno a při jejím překročení musí být některé seznamy zapsány na disk. Jsou zde uplatňovány dvě strategie. Jednak jsou zapisovány ty seznamy, do kterých nejdéle nepřibyl žádný dokument, a také jsou zapisovány seznamy, jejichž délka překročila určitý limit. Druhá strategie je použita proto, že pokud bychom dlouhé seznamy v paměti nechávali, tak by tyto seznamy rostly a postupně by vytlačovaly stále více krátkých seznamů. V tomto případě je výhodnější dlouhý seznam přesunout na disk a ponechat si v paměti větší množství kratších seznamů.

Po dokončení aktualizace indexu musejí být obsahy všech vyrovnávacích pamětí zapsány do databáze – tedy na disk. Časová náročnost této operace bude značně časově náročná, obzvlášť při přidávání dokumentů do indexu, kdy bude

potřeba nejprve z disku načíst původní seznam výskytů, upravit jej a znovu zapsat zpět.

5.7. Vyhledávání v indexu

Indexy používáme, abychom mohli v datech rychle hledat. Navíc pomocí indexu můžeme nalezené dokumenty setřídit podle ohodnocení – tedy podle toho, jak moc dokument odpovídá položenému dotazu. V tomto případě dotaz může být libovolný boolovský výraz obsahující spojky AND, OR a negaci.

5.7.1. Parsování dotazu

Protože dotaz zadává uživatel, je pravděpodobné, že v dotazu mohou být chyby. Proto je vlastní dotaz nejprve kontrolován na správnost uzávorkování a použití logických spojek konjunkce, disjunkce a negace. Pro konjunkci je, v souladu s konvencemi Oracle Text, použit znak „&", pro disjunkci je použit znak „|". Pro negaci se používá znak „-" (mínus). Pokud není použita žádná spojka, pak se automaticky předpokládá, že se jedná o konjunkci dvou termů.

Protože práce s obecným boolovským dotazem je poměrně komplikovaná, je dotaz převeden do disjunktivně normální formy. Po převodu tedy získáme disjunkci konjunkcí termů a negací termů. Vyhodnocení takovéhoto výrazu je již mnohem jednodušší, protože neobsahuje vnořené podvýrazy. Pro převod obecného boolovského výrazu na DNF je použita komponenta BoolStuff[1]. Dotaz však může obsahovat písmena s diakritikou, případně jiné speciální znaky, které by mohly dělat problémy v knihovně BoolStuff. Proto jsou termy z dotazu nejprve očíslovány a při převodu výrazu do DNF jsou již místo termů použity pouze číselné hodnoty.

5.7.2. Vyhodnocení dotazu

Nejprve je potřeba vyhledat zadané termy v indexu. Pokud je zadaný term dostatečně dlouhý, jsou hledány i termy, které mají libovolnou příponu. Minimální délka termu, pro který budou v indexu hledány i přípony lze definovat parametrem „suffix_search_min_token_len". Výchozí hodnota tohoto parametru jsou 4 znaky. Každý index v databázi může mít tento parametr nastaven na jinou hodnotu. U kratších termů je požadována přesná shoda. Čím je nalezený term delší, tím je považován za méně podobný zadanému termu. Váha dokumentů, ve kterých se

[1] BoolStuff je knihovna napsaná v jazyce C++ a umožňuje převod boolovského výrazu do DNF. Knihovnu napsal Pierre Sarrazin, více informací viz http://sarrazip.com/dev/boolstuff.html

tento term vyskytuje, je pak násobena koeficientem podobnosti, který se vypočítá vztahem

$$koef_{hledaný\_term,nalezený\_term} = \frac{2|heladný\_term|}{|hedaný\_term| + |nalezený\_term|},$$

kde $|term|$ označuje délku termu ve znacích. Pokud je tento koeficient menší než 0,5, pak nalezený term vůbec není uvažován. Podobné termy s předponou hledány nejsou, protože použitý způsob uložení termů neumožňuje snadné hledání termů s různou předponou. Jedním ze způsobů, jak umožnit vyhledávání v indexu slova s předponami, by bylo uložení termů se všemi rotacemi [6]. To by však vyžadovalo podstatně více místa na disku pro uložení seznamu termů.

Tato metoda má samozřejmě tu nevýhodu, že přidáním přípony k nějakému slovu může vzniknout jiné slovo, které nemá s hledaným nic společného. Lepší variantou by bylo vyhledávání pomocí kořene slova, ale to by vyžadovalo použití tokenizátoru s vestavěným stemmerem. Další alternativou by pak bylo umožnit zadávat do dotazu zástupné znaky, ale to tato implementace indexu neumožňuje.

Takto získáme pro každý term zadaný v dotaze seznam termů z indexu. Pokud je seznam prázdný, znamená to, že požadovaný term nebyl v kolekci dokumentů nalezen. Pokud seznam obsahuje více termů, pak to znamená, že relevantní jsou dokumenty obsahující libovolný term ze seznamu. Míra relevance pak bude záviset na ohodnocení daného dokumentu pro daný term a také na koeficientu podobnosti s termem z dotazu.

Pro vyhodnocení boolovského dotazu je použit paiceův model, který je založený na fuzzy množinách. Pří výpočtu ohodnocení pro konjunkci nebo disjunkci termů jsou do výpočtu zahrnuty váhy všech termů z dotazu. V našem případě lze koeficienty modelu nastavit v tabulce PRECIS_PARAMETER. Koeficienty jsou dány parametry paice_coef_and a paice_coef_or pro konjunkci resp. disjunkci termů. Popis paiceova modelu a význam koeficientů je uveden v kapitole 3.4.3 Rozšířený boolovský model.

Při vyhledávání dokumentů pro zadané termy použijeme B-strom, kde jsou dokumenty setříděny podle váhy. Ze stromu od každého termu vybereme ty dokumenty, jejichž váha je nejvyšší. Pokud dokumenty nejsou uloženy v B-stromě, ale v seznamu, setřídíme tento seznam podle váhy dokumentů. Tímto způsobem postupně načteme požadované množství dokumentů. Vzhledem k tomu, že ohodnocení dokumentu ještě není v tuto chvíli konečné, počet načtených dokumentů bude záviset na tom, kolik jich uživatel chce nalézt a také na počtu pozitivních termů v konjunkci.

Díky tomu, že máme dokumenty setříděné podle váhy, nemusíme procházet celé B-stromy u všech termů. Stačí pouze projít dokumenty s nejvyšším ohodnocením. Tím získáme seznam dokumentů, které mají dobré ohodnocení alespoň u jednoho zadaného termu v dotazu. Při zjišťování váhy dokumentu vůči ostatním termům pak použijeme B-stromy setříděné podle čísla dokumentu.

V případě disjunkce poddotazů již nezjišťujeme ohodnocení všech dokumentů ve všech poddotazech, protože by to bylo zbytečně časově náročné a již by to příliš neovlivnilo celkový výsledek. Použijeme pouze ta ohodnocení, která byla spočítána při hledání nejlepších dokumentů poddotazu. Pro výpočty ohodnocení všech konjunkcí i disjunkcí je použit paiceův model.

5.7.3. Pseudo algoritmus pro vyhledávání v indexu

Následující algoritmy popisují základní části vyhledávání v indexu. Algoritmus předpokládá existenci funkcí pracujícími s daty uloženými v indexu. Jednou z nich je GetNextDocument, která umožňuje postupné načítání dokumentů setříděných podle ohodnocení.

Algoritmus: QueryIndex

```
Vstup: Query Q, MinWeight w_min
Výstup: WeightList W
Proměnné: ParsedQuery Q', TokenRootsList T, SubqueryRoots S,
SubqueryDocumentWeightList SD, Weight w

Q' := ParseQuery(Q)
Q' := GetDisjunctiveNormalForm(Q')

for all Token t in Q do
  T_i := GetTokenIndexTrees(t)
end for

S := GetDnfTermRoots (Q')
for all Subquery s in S do
  SubqueryFindDocs(T, s, SD)
end for

for all Document d in SD do
  w := ExtendedBoolOr(SD_d)
  if w > w_min then
    W_d := w
  end if
end for
```

```
Algoritmus: GetTokenIndexTrees
  Vstup: Token t
  Výstup: TokenRoots T
  Proměnné:

  GetParameter(suffix_search_min_token_len)

  if strlen(t) < suffix_search_min_token_len then
    FindToken(t)
    if token found then
      T0 := t
    else
      T0 := empty
    end if
  else
    T := FindTokensWithSuffix(t) do
    // Načte z indexu seznam termů, které mají jako prefix term zadaný v dotazu.
    // Ke každému nalezenému termu přidá koeficient podobnosti, který je počítán
    // na základě rozdílu délek zadaného a nalezeného termu.
  end if
```

```
Algoritmus: SubqueryFindDocs
  Vstup: MaxDocumentCount num_docs, MinWeight wmin
  Výstup: DocWeights W
  Proměnné: PositiveTokens Tpos, NegativeTokens Tneg, Weight w

  read_count := num_docs*(1 + log(count(Tpos)))

  while count(W) < read_count and Document d := GetNextDocument(Tpos, Tneg)
  do
    if d not in W then
      w := GetWeight(Tpos, Tneg, d)
      if w > wmin then
        Wd := w
      end if
    end if
  end while
```

5.7.4. Příklad vyhodnocení dotazu

Postup algoritmu pro vyhodnocení dotazu si ukážeme na příkladu. Předpokládejme, že uživatel zadal dotaz „(osobní počítač | notebook) -mac“. Pokud mezi termy není žádná logická spojka, pak se předpokládá, že se jedná o konjunkci. Uživatel tedy hledá dokumenty, které obsahují slova *osobní počítač* nebo *notebook*, ale neobsahují *mac*.

Dotaz je nejprve převeden do disjunktivně normální formy. Po převodu bude dotaz vypadat následovně:

(osobní & počítač & -mac) | (notebook & -mac)

Nyní se každý term z dotazu vyhledá v indexu. Standardně se pro termy délky alespoň 4 znaky hledají také termy s příponou. Ke každému termu se vytvoří pole odkazující na kořeny stromů, které obsahují čísla dokumentů, ve kterých se daný term vyskytuje. Zároveň je pro termy s příponou spočítána podobnost se zadaným termem. Podobnost se počítá pouze na základě délky nalezeného termu. V ukázkovém příkladě tak získáme tyto termy:

osobní, *osobních*, *osobního*, *osobním*, *osobnímu*, ...
počítač, *počítače*, *počítačem*, *počítačový*, *počítačový*, ...
notebook, *notebooku*, *notebooky*, ...
mac

Jako další krok bude nalezení dokumentů vyhovujícím jednotlivým konjunkcím. V konjunkci mohou být buď termy a negace termů, nebo pouze pozitivní termy. Pokud by konjunkce obsahovala jen negace termů, pak do výsledku nepřidá žádné dokumenty.

Ze všech pozitivních termů z konjunkce jsou postupně brány dokumenty s nejlepším ohodnocením. V tomto kroku se využívá toho, že seznam výskytů dokumentů je uložen rovněž jako setříděný podle jejich ohodnocení. Toto ohodnocení však platí pouze pro jeden z termů v konjunkci a proto musí být přepočítáno ohodnocení dokumentu pro celou konjunkci. Po přepočtu se může ohodnocení zvýšit i snížit v závislosti na tom, zda daný dokument obsahuje také některé z ostatních termů z konjunkce.

Protože konjunkce může obsahovat i negativní termy, musí se navíc u každého dokumentu kontrolovat, zda v něm není obsažen některý z těchto termů. V případě že ano, pak se dokument nedostane mezi vyhledané dokumenty bez ohledu na ohodnocení pozitivními termy.

Tento postup se opakuje pro všechny konjunkce z upraveného dotazu.

1. konjunkce: (osobní & počítač & -mac) bude po doplnění termů s příponami upravena na:
(osobní | osobních | osobního | ...) & (počítač | počítače | počítačem | ...) & -mac

2. konjunkce: (notebook & -mac) bude po doplnění termů s příponami upravena na:
(notebook | notebooku | notebooky | ...) & -mac

Pro obě konjunkce postupně získáme seznam dokumentů, které budou setříděny od nejlepšího ohodnocení jednoho z termů. Toto ohodnocení pak bude přepočítáno pro celý poddotaz. Z obou poddotazů tak získáme seznam dvojic: dokument a ohodnocení.

Poté, co vyhodnotíme všechny poddotazy, spočítáme ohodnocení dokumentů v celém dotazu. Pro některé dokumenty budeme mít ohodnocení z obou poddotazů, ostatní pak budou ohodnoceny jen v jednom z nich. V případě, že ohodnocení dokumentu pro jeden z poddotazů nebylo spočítáno, tak jej považujeme za nulové. Toto zjednodušení sice může mírně ovlivnit výsledné uspořádání dokumentů, ale výpočet všech ohodnocení by byl dosti časově náročný a již by výsledek příliš neovlivnil.

6. Měření výkonnosti

Úkolem indexu je umožnit co nejrychlejší vyhledávání v datech. K tomuto účelu byly navrženy jeho datové struktury. Samotné vytvoření datových struktur však může být dosti časově náročné. Proto nás při měření výkonnosti bude zajímat nejen vyhledávání v indexu, ale také vytvoření a úprava indexu po změnách dat v databázi.

Pro měření výkonnosti již během vývoje byl zajímavý nejen celkový čas běhu algoritmů, ale také jejich jednotlivých částí. Pro tento účel byl do kódu přidán vlastní profiler, který měří časy potřebné pro určité části aplikace. Získané informace byly použity pro odhalení úzkých hrdel aplikace. Čas je měřen pomocí Windows funkce *QueryPerformanceCounter*. Tato funkce vrací přesný časový údaj pomocí 64 bitového čísla a lze tak měřit časové intervaly v řádu nanosekund. Profiler naměřená data ukládá do tabulek PRECIS_PROFILER_RUN a PRECIS_PROFILER_DATA. Profiler sleduje, kolik času zabralo vykonávání jednotlivých procedur. Pokud procedura volá další procedury, tak je spočítán jednak celkový čas a navíc čas bez volaných vnořených procedur.

Ukázka výstupu z profileru v tabulce 6-1 zobrazuje synchronizaci indexu po přidání prvních 1000 dokumentů z české wikipedie do prázdné indexované tabulky. Celkový čas běhu aktualizace indexu byl 28s. Z tabulky lze rovněž vyčíst, že téměř 24s trvala tokenizace dokumentů.

6.1. Software a hardware

Všechny testy byly prováděny na databázovém serveru Oracle Database 10g Release 2 Express Edition. Sever běžel na notebooku HP Compaq nc6220 s operačním systémem Windows XP. Testovací notebook měl následující konfiguraci:

Procesor:	Intel Pentium M 1,86GHz (Centrino)
RAM	2*1GB DDR II 667MHz, Kingston CL5
HDD	Hitachi Travelstar 7K60, 60GB 7200 rpm
OS	Windows XP SP2 Home Edition Eng.

Název procedury	Počet volání	Celkový čas		Čas bez volání podprocedur		Čas jednoho volání		
						min.	průměr	max.
Btree::Btree	4	0,0	0,0%	0,0	0,0%	0,005	0,007	0,01
Btree::CreateNode	4	0,0	0,0%	0,0	0,0%	0,006	0,01	0,012
Btree::CreateNode - no cache	339	0,6	0,0%	0,6	0,0%	0,002	0,002	0,002
Btree::FlushCache	4	15,2	0,1%	0,1	0,0%	3,148	3,79	5,058
Btree::MakeTree	4	228,5	0,8%	4,5	0,0%	0,173	57,122	126,839
Btree::ReadNode	4	0,0	0,0%	0,0	0,0%	0,002	0,002	0,002
Btree::WriteNode	4	0,0	0,0%	0,0	0,0%	0,002	0,002	0,002
Btree::~Btree	4	15,2	0,1%	0,1	0,0%	3,159	3,802	5,071
IndexTree::AddDocuments	96724	2411,4	8,5%	430,4	1,5%	0,016	0,025	86,652
IndexTree::AddDocuments - 1b	96724	1981,0	7,0%	431,6	1,5%	0,013	0,02	86,647
IndexTree::IndexTree	96724	337,5	1,2%	337,5	1,2%	0,003	0,003	4,259
IndexTree::~IndexTree	96724	204,4	0,7%	204,4	0,7%	0,001	0,002	4,316
LobData::CreateNode	97060	1772,8	6,2%	409,9	1,4%	0,009	0,018	86,641
LobData::FlushCache	39	1007,2	3,5%	1007,2	3,5%	0,011	25,827	86,622
LobData::LoadFreeNodes	10	64,2	0,2%	0,1	0,0%	0,865	6,424	27,558
LobData::ReadNode	14	71,7	0,3%	71,7	0,3%	0,86	5,121	27,545
LobData::SaveFreeNodes	10	174,6	0,6%	0,1	0,0%	3,787	17,463	66,595
LobData::WriteNode	97074	1552,6	5,5%	404,8	1,4%	0,005	0,016	86,636
LobData::WriteNode - FlushCache	29	751,9	2,6%	0,2	0,0%	18,755	25,927	86,627
LobData::WriteNode - to cache	97060	206,3	0,7%	206,3	0,7%	0,001	0,002	4,995
LobData::WriteNode - to lob	14	189,6	0,7%	189,6	0,7%	3,133	13,54	66,578
PrecisIndex::Delete	1	3,7	0,0%	3,7	0,0%	3,693	3,693	3,693
PrecisIndex::Insert - oracle	8	23259,3	81,7%	23259,3	81,7%	2,331	2907,41	4108,97
PrecisIndex::Insert - tokenize	8	23945,8	84,1%	20,2	0,1%	4,677	2993,22	4193,84
PrecisIndex::UpdateIndexCacheToken	8	666,3	2,3%	666,3	2,3%	0,139	83,292	106,935
PrecisIndex::UpdateIndexFlushCache	1	3738,4	13,1%	109,0	0,4%	9	3738,41	3738,41
TokenTree::AddTokens	6	3629,4	12,7%	676,2	2,4%	16,784	604,905	1875,35
TokenTree::ReadTokensToCache	4	8,1	0,0%	0,5	0,0%	1,377	2,014	3,893
TokenTree::TokenTree	4	8,1	0,0%	0,0	0,0%	1,381	2,018	3,897
TokenTree::~TokenTree	4	228,7	0,8%	0,2	0,0%	0,187	57,18	126,95

Tabulka 6-1 Výstup z profileru při aktualizaci indexu

6.2. Testovací databáze – články z české wikipedie

Pro testovací účely jsem použil dvě databáze. První z nich je databáze článků české wikipedie uvedená ke stažení dne 7. 3. 2008. Z této databáze jsem použil pouze texty článků. Databáze ke stažení obsahuje články v XML formátu, který lze pomocí nástroje xml2sql[2] převést do textového formátu. Data z tohoto formátu lze pomocí příkazu LOAD DATA importovat do databáze MySQL. Celá databáze obsahuje více než 184.000 článků a zabírá přibližně 373 MB. Pro testování nebylo tolik dat potřeba, a proto byla do databáze Oracle překopírována pouze část této databáze. Pro import dat z MySQL do databáze Oracle jsem použil PHP skript, který je společně s celou databází uložen na přiloženém CD.

Pro zjištění chování indexu na různě velkých datech byly vytvořeny kolekce dokumentů obsahující 1000 až 32000 článků. Aby byly výsledky snáze porovnatelné mezi sebou, omezil jsem velikost článků od 512 B do 64 kB. Průměrná velikost článků se v různých kolekcích pohybovala v rozmezí od 2700 B do 3066 B. Parametry jednotlivých kolekcí jsou uvedeny v tabulce 6-2.

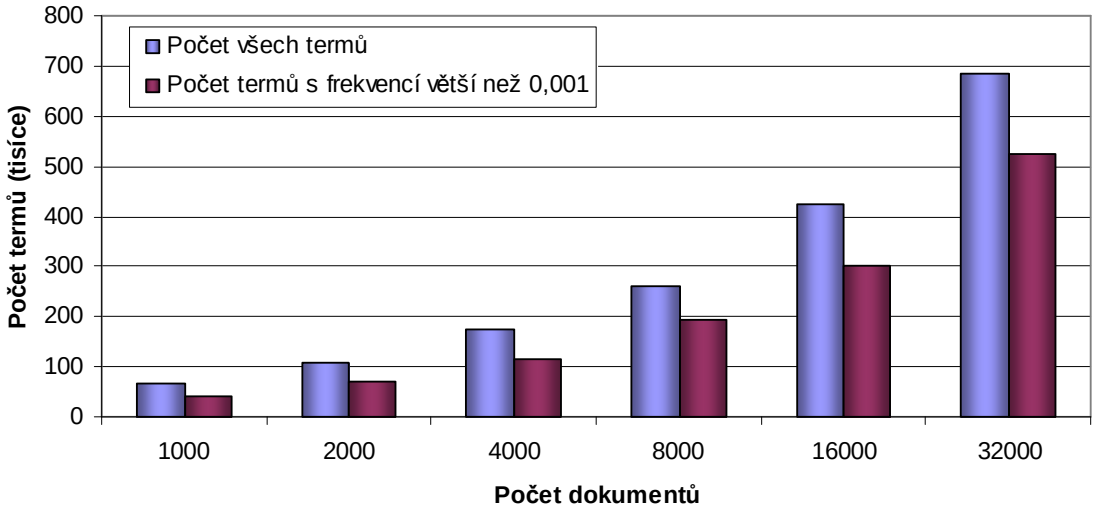
Počet dokumentů	**Počet různých termů**	**Průměrná velikost dokumentu [B]**	**Celková velikost [MB]**
1000	67167	3066	2,9
2000	107198	2849	5,4
4000	174368	3009	11,5
8000	259343	2692	20,5
16000	424489	2725	41,6
32000	685656	2669	81,5

Tabulka 6-2 Parametry testovaných kolekcí dokumentů

Jak je vidět z tabulky 6-2, dokumenty obsahují poměrně velké množství termů. Je to dáno tím, že v indexu jsou všechna slova včetně překlepů, která se v dané kolekci dokumentů objevila. Z tohoto důvodu zabírají vytvořené indexy dost místa a jejich vytvoření je poměrně časově náročné. Nejjednodušším způsobem snížení počtu termů, je použití omezení na minimální frekvenci termu v dokumentu. Takto však můžeme omezit počet indexovaných termů jen u dostatečně dlouhých dokumentů. Počty všech termů v indexu a termů s frekvencí vyšší než 1/1000 je vidět v grafu 6-1, kde jsou zobrazeny počty termů pro různé

[2] xml2sql je nástroj, který umožňuje konverzi wikipedie z formátu XML, ve kterém lze volně stáhnout, do SQL příkazů, nebo do textového formátu, který lze použít při importu dat do MySQL databáze. Xml2sql lze stáhnout z http://meta.wikimedia.org/wiki/Xml2sql

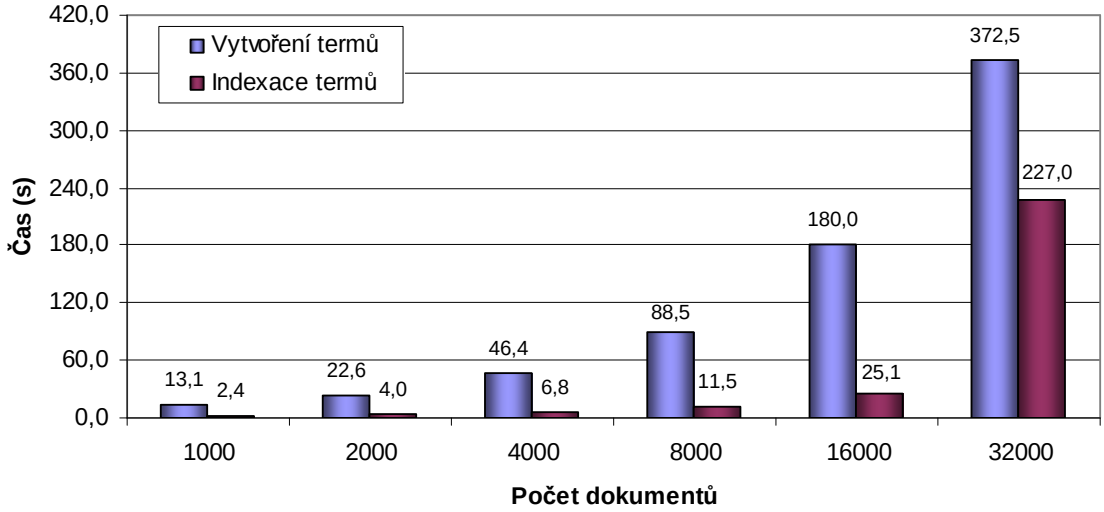
kolekce dokumentů. Vzhledem k tomu, že termy které se v dokumentech vyskytují s nižší frekvencí než 1/1000, budou zpravidla nevýznamné, budeme v následujících testech uvažovat vytvořený index s tímto omezením.



Graf 6-1 Počet termů v dokumentech

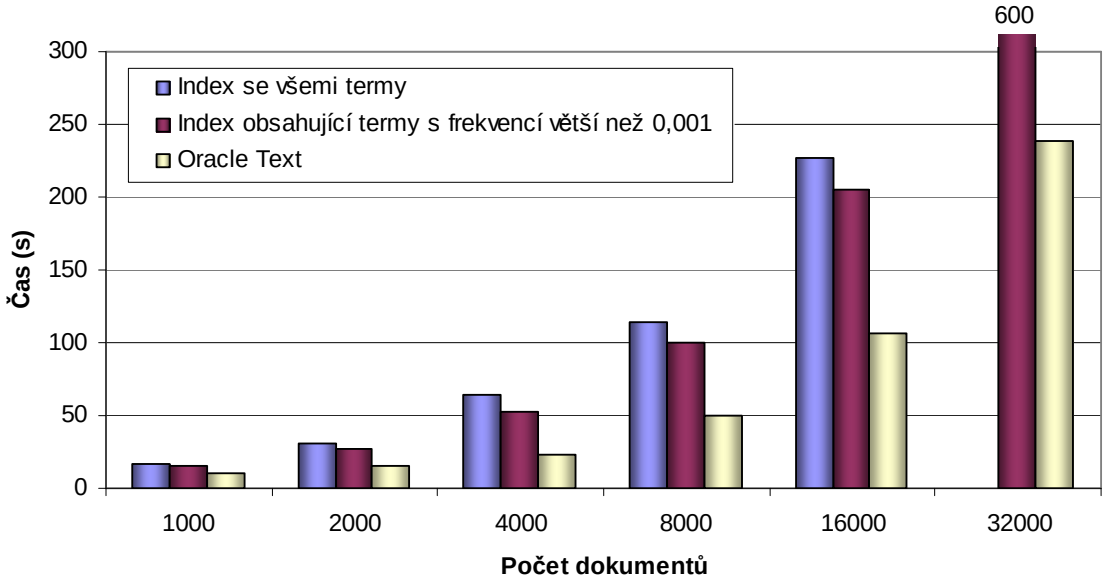
6.2.1. Vytvoření indexu

V prvním testu byla měřena rychlost vytvoření indexu pro různě velké kolekce dokumentů. Výsledky měření jsou zobrazeny v grafu 6-2. Z grafu je patné, že značná část času při vytváření indexu připadá na tokenizaci dokumentů. To je dáno tím, že musejí být načteny jednotlivé dokumenty z databáze a také tím, že tokenizace je prováděna voláním funkcí Oracle Textu z PL/SQL pro každý dokument zvlášť.



Graf 6-2 Vytvoření indexu - česká wikipedie

V případě, kdy bylo indexováno 32000 dokumentů, což odpovídá přibližně 81 MB dat, dochází k velkému prodloužení času indexace termů. Tento časový nárůst je dán tím, že v tomto případě se již celý index nevejde do přidělené operační paměti a data musejí být ukládána do databáze již v průběhu indexace. Celkově se tak výrazně zvyšuje počet přístupů na disk.



Graf 6-3 Čas potřebný pro vytvoření indexu – porovnání s Oracle Text

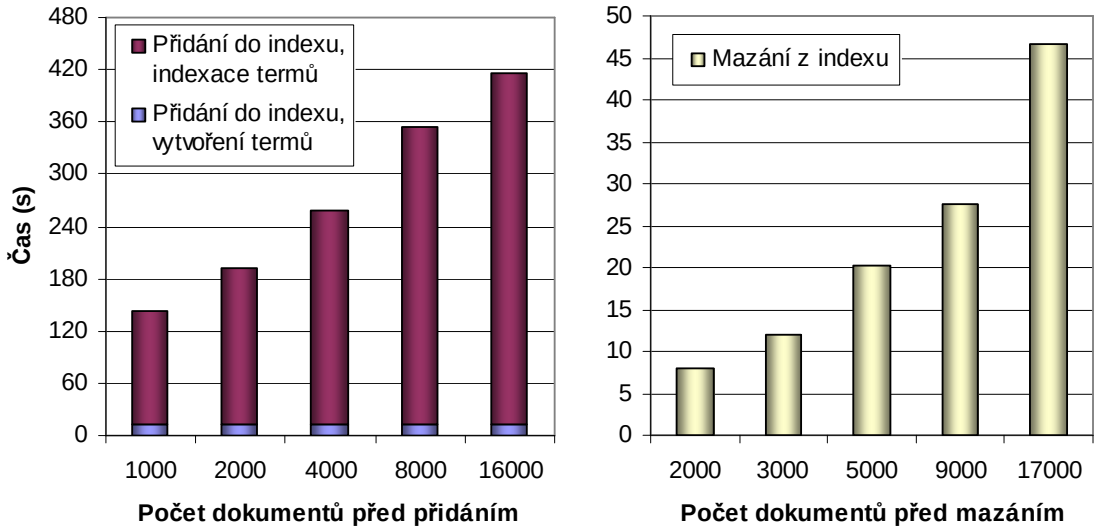
V grafu 6-3 je zobrazeno porovnání rychlosti vytvoření fulltextového indexu pomocí Précis indexu a Oracle Text. V databázi článků byl indexován pouze jeden sloupec z jedné tabulky, takže lze snadno provést srovnání s klasickým textovým indexem. V grafu jsou uvedeny časy vytvoření indexu při použití všech termů a termů, jejichž frekvence v dokumentu je vyšší než 1/1000. Oracle text ukládal do indexu všechny termy. Oracle text navíc ukládá do indexu pozici termu v dokumentu.

Z grafu 6-3 je patrné, že implementace Précis indexu vytvoří index za přibližně dvojnásobek času oproti Oracle Text. To však platí pouze pro kolekce dokumentů, jejichž celý index se vejde do paměti. Vzhledem k tomu, že Oracle Text má za sebou podstatně delší vývoj, tak je to celkem uspokojivý výsledek. Navíc, pokud by se podařilo zrychlit tokenizaci dokumentů, mohly by být časy srovnatelné.

6.2.2. Aktualizace indexu

Při úpravě indexovaných dat se zapisuje pouze informace o provedené změně do tabulky PRECIS_PENDING. Změny v indexu se provedou až při zavolání funkce sync_index, která pomocí záznamů z tabulky PRECIS_PENDING projde změněné, přidané, nebo smazané dokumenty. V následujících testech jsem měřil

rychlost aktualizace indexů, které již obsahovaly různě velké kolekce dokumentů. Do kolekce dokumentů bylo nejprve přidáno 1000 nových dokumentů a index byl aktualizován. Poté bylo těchto 1000 dokumentů opět odebráno a index byl znovu aktualizován. Do všech kolekcí dokumentů byla přidána stejná sada nových dokumentů. Výsledky testu jsou zobrazeny v grafu 6-4.



Graf 6-4 Aktualizace indexu – přidávání a mazání 1000 dokumentů

Tak dlouhé časy aktualizace indexu po přidání 1000 dokumentů jsou způsobeny velkým počtem termů, které tyto dokumenty obsahují. Čím více dokumentů již bylo indexováno, tím vyšší je pravděpodobnost, že term z nové kolekce již v indexu byl. V takovém případě musí být z databáze načten původní seznam výskytů tohoto termu, aktualizován a znovu zapsán na disk. Navržený způsob bohužel neumožňuje připsání nových výskytů na konec seznamu, protože seznamy jsou setříděné a není zde možnost seznamy fragmentovat.

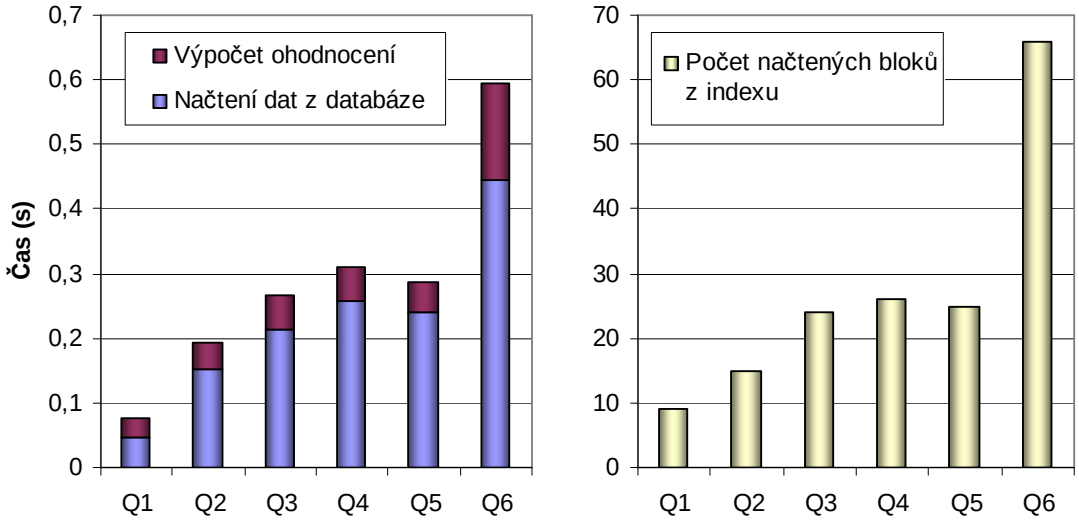
Při mazání dokumentů z indexu se musí projít celý index a vyhledat v něm výskyty mazaného dokumentu. Pokud index obsahuje malé množství dokumentů, pak většina seznamů výskytů je uložena v setříděných seznamech a ne v B-stromech. Mazání z těchto seznamů je výrazně rychlejší, než mazání z B-stromů, protože tyto seznamy mohou být načítány a zapisovány po větších množstvích. Tím je podstatně snížen počet přístupů na disk.

6.2.3. Vyhledávání v indexu

Během testů rychlosti vyhledávání byla použita největší kolekce dokumentů obsahující 32000 článků. Rychlost byla měřena pro šest různých dotazů. Byl sledován nejen celkový čas pro vyhodnocení dotazu, ale také počet načtených

bloků z databáze a čas potřebný pro načtení těchto bloků. Výsledky měření jsou zobrazeny v grafu 6-5 pro tyto dotazy:

Q1 Astronomie
Q2 Informační systémy
Q3 Pulp Fiction
Q4 Vyhledávání databáze Oracle –MySQL
Q5 (Fulltextové vyhledávání) | (Dokumentografické informační systémy)
Q6 Index relační databáze



Graf 6-5 Vyhledávání v indexu

6.3. *Testovací databáze – FreeDB*

Jako druhou testovací databázi jsem použil volně dostupnou databázi FreeDB obsahující informace o audio CD. Jedná se o textovou databázi, kde pro každé CD je vytvořen jeden soubor, který obsahuje název alba, interpreta, seznam skladeb, žánr a případné další doplňující informace. Protože celá databáze obsahuje velké množství dat, použil jsem aktualizaci za měsíc vydanou dne 8. 7. 2008. Tato aktualizace obsahuje informace o 36 116 CD nosičích. Celková velikost souborů je 35,6MB.

Pro konverzi dat do databáze Oracle jsem použil PHP skript, který postupně načítá všechny soubory ze zadaného adresáře. Pro každé načtené CD vytvoří záznam v tabulce FREEDB_DISC a informace o skladbách uloží do tabulky FREEDB_TRACK. Struktura obou tabulek, textová databáze a PHP skript je uložen na přiloženém CD.

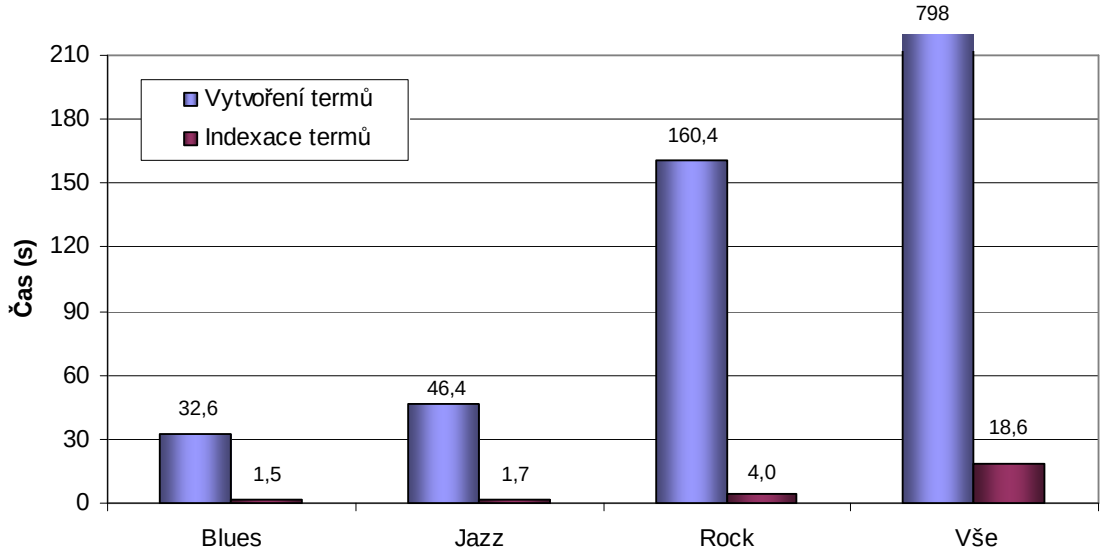
Soubory v databázi FreeDB jsou rozděleny do adresářů podle žánrů. Toho jsem využil a vytvořil jsem 4 testovací sady obsahující různý počet záznamů. Množství záznamů v jednotlivých kolekcích je uveden v tabulce 6-3.

Žánr (kolekce)	Počet CD disků	Počet skladeb	Velikost textové databáze (MB)
Blues	1 788	22 727	1,6
Jazz	2 567	29 393	2,1
Rock	8 905	116 142	8,3
Vše (celkem 11 žánrů)	36 116	488 145	35,6

Tabulka 6-3 Počet záznamů v testovacích kolekcích

6.3.1. Vytvoření indexu

V grafu 6-6 je zobrazen čas potřebný pro vytvoření indexu pro jednotlivé testovací kolekce. Indexovány byly jen sloupce FREEDB_DISC.TITLE a FREEDB_TRACK.TTITLE, tedy pouze názvy CD a názvy skladeb. Pro největší kolekci, obsahující všechna CD to představuje celkem více než 524 000 dokumentů. V dokumentech bylo uloženo 12,6MB dat, což není tak mnoho. Vzhledem k velkému množství dokumentů bylo při vytváření indexu spotřebováno značné množství času tokenizací dokumentů. Důvodem je pravděpodobně neefektivní volání tokenizace z PL/SQL.

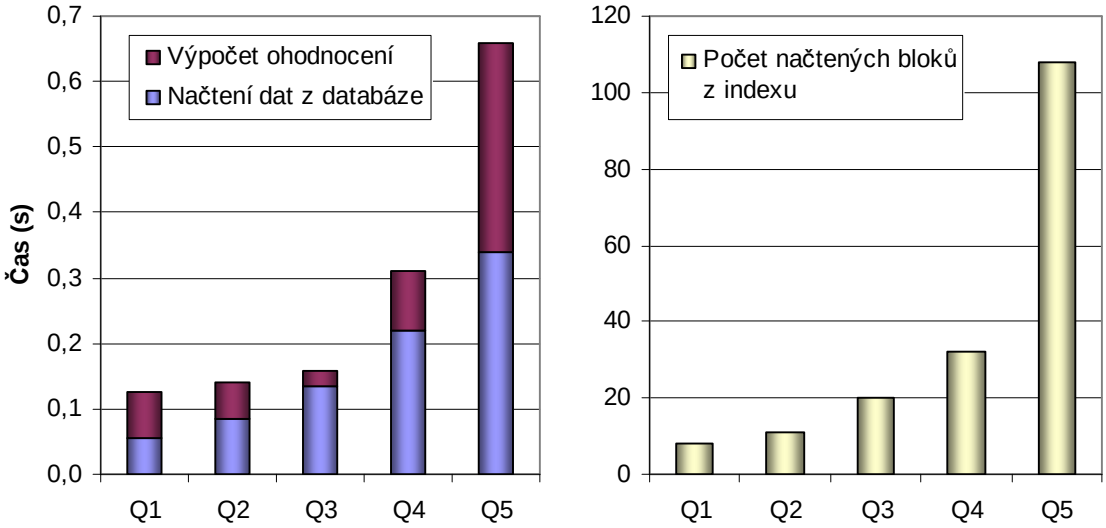


Graf 6-6 Vytvoření indexu

6.3.2. Vyhledávání v indexu

Během testů rychlosti vyhledávání byla použita databáze obsahující všechna data. Rychlost byla měřena pro pět různých dotazů. Dotazy se lišily hladně v počtu zadaných termů. Stejně jako u předchozí testovací databáze, byl sledován nejen celkový čas pro vyhodnocení dotazu, ale také počet načtených bloků z databáze a čas potřebný pro načtení těchto bloků. Výsledky měření jsou zobrazeny v grafu 6-7 pro tyto dotazy:

Q1 Metallica
Q2 Nirvana | Metallica
Q3 Pulp Fiction
Q4 Nirvana (Nevermind | Unplugged in New York)
Q5 Iron Maiden Can I Play With Madness



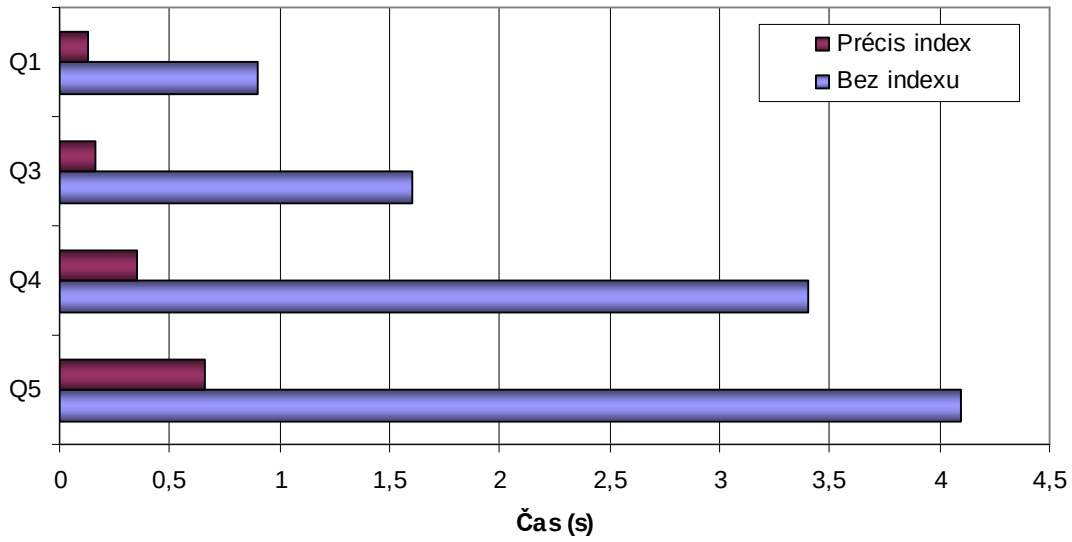
Graf 6-7 Vyhledávání v indexu

6.4. *Vyhledávání v databázi bez indexu*

V následujícím testu byla porovnána rychlost vyhledávání v databázi bez indexu a s použitím Précis indexu. Rychlost vyhledávání byla měřena v obou testovacích databázích. První testovanou databázi byla FreeDB. Pro vyhledávání bez indexu jsem použil obdobnou formu dotazů jako [9], tedy

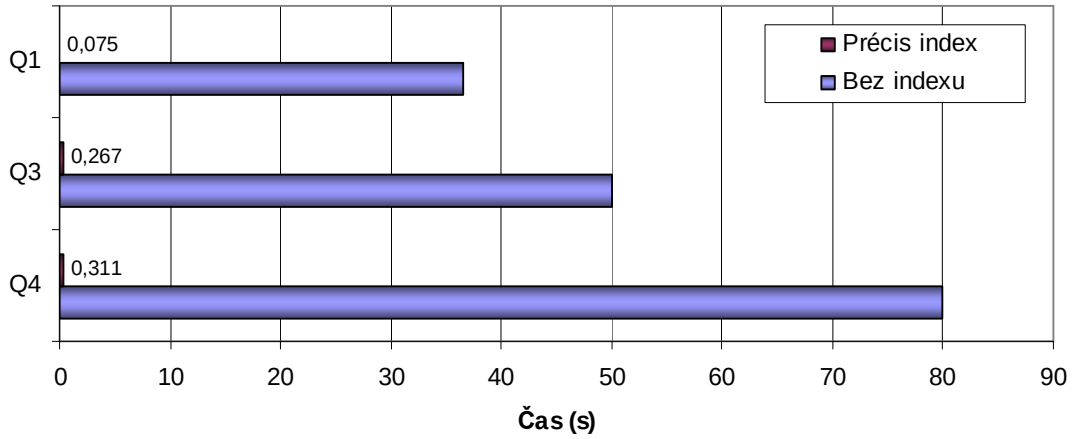
```
SELECT title FROM freedb_disc
 WHERE upper(title) LIKE '%PULP%' OR upper(title) LIKE '%FICTION%'
UNION ALL
SELECT ttitle FROM freedb_track
 WHERE upper(ttitle) LIKE '%PULP%' OR upper(ttitle) LIKE '%FICTION%'
```

Pro porovnání rychlosti byly použity stejné dotazy, jako v předchozích testech. Výsledky jsou zobrazeny v grafu 6-8. Vyhledávání bez indexu bylo podle očekávání výrazně pomalejší, než vyhledávání pomocí indexu. Přesto však čas na vyhledávání dat bez indexu byl relativně přijatelný. To je dáno tím, že Oracle celou databázi měl během vyhledávání načtenou v operační paměti a nemusel data načítat z disku.



Graf 6-8 Porovnání rychlosti vyhledávání v databázi bez indexu a s Précis indexem – databáze FreeDB

Obdobný test jsem provedl i s databází článků wikipedie. V tomto případě musela být všechna data během vyhledávání načítána z disku, což se na času vyhledávání projevilo ještě výrazněji než v předchozím případě. Výsledky měření jsou zobrazeny v grafu 6-9. Vyhledávání v takto velké databázi bez indexu je téměř nepoužitelné. Použité dotazy jsou stejné jako v kapitole 6.2.3.



Graf 6-9 Porovnání rychlosti vyhledávání v databázi bez indexu a s Précis indexem – databáze článků české wikipedie

7. Závěr

Cílem této práce bylo vytvoření indexu, který by byl vhodný pro Précis dotazy. Tyto dotazy prohledávají celou databázi a proto je potřeba do indexu vložit data z více tabulek. To běžné indexy neumožňují.

Podařilo se mi navrhnout takové datové struktury, které umožňují rychlé vyhledávání dokumentů podle zadaného dotazu. Dotazem může být libovolná kombinace konjunkcí, disjunkcí a negací termů. Navíc mohou dotazy obsahovat uzávorkování. Datové struktury indexu umožňují aktualizaci v případě, že dojde ke změnám v indexovaných dokumentech.

Index byl navržen tak, aby umožňoval co nejjednodušší používání. Pro vyhledávání v něm se používá běžných SQL dotazů, které jsou velmi podobné fulltextovému indexu Oracle Text. Protože index obsahuje data z více tabulek, je potřeba dodržet určitou syntaxi, aby databázový server použil Précis index.

Práce navazuje na [9], kde pro vyhledávání v databázi nejsou použity žádné indexy. Místo toho jsou tabulky procházeny sekvenčně a jsou hledány shody s hledanými termy z dotazu. V kapitole 6.4 byl proveden test rychlosti vyhledávání v datech bez indexu a s použitím Précis indexu. Z testu vyplývá, že pokud jsou prohledávaná data uložena ve vyrovnávací paměti databázového serveru, pak vyhledávání v nich je časově přijatelné. Pokud však data musejí být načtena z disku, pak jejich prohledávání bez indexu je z časových důvodů nepoužitelné. V obou případech však bylo vyhledávání pomocí Précis indexu výrazně rychlejší než bez něj. Navíc index umožňuje setřídit nalezené dokumenty podle ohodnocení, což bez něj není možné.

Přestože index splnil požadavky, které byly kladeny na začátku práce, z výsledů testů vyplývají některé možné změny ve struktuře indexu. Současná implementace indexu používá pro seznam výskytů termu dva B-stromy. V prvním jsou dokumenty setříděny podle čísla a ve druhém podle ohodnocení. Tento způsob byl zvolen, aby v případě dlouhých seznamů nemusel být procházen celý strom pro nalezení dokumentu s nejlepším ohodnocením. V praxi však většina těchto seznamů není tak dlouhá, aby tento přístup výrazně zkrátil čas potřebný pro vyhledávání v indexu. Naopak to ve většině případů znamená, že jsou načítána data z obou stromů a zvyšuje se tak počet přístupů na disk. Navíc index zabírá více místa na disku a jeho vytvoření je více časově náročné. Výhoda tohoto řešení by se projevila až pro větší počet indexovaných dokumentů.

Rychlost aktualizace indexu by bylo možné zvýšit tak, že by byla umožněna fragmentace seznamu výskytů. V současné implementaci toto možné není, a proto po přidání nových dokumentů do indexu musí být nejprve načten původní seznam

výskytů, upraven a znovu uložen na disk. Fragmentace seznamu výskytů by umožňovala výrazně rychlejší přidávání dokumentů do indexu za cenu zpomalení vyhledávání v indexu.

Značnou část času pří vytváření indexu zabrala tokenizace dokumentů, která byla prováděna prostřednictvím Oracle Textu. To se projevilo obzvláště v případě indexování strukturovaných dat, kde bylo indexováno velké množství krátkých dokumentů – jako v případě databáze FreeDB. Vytváření indexu by se dalo výrazně urychlit použitím jednoduššího tokenizátoru pro strukturovaná data.

8. Literatura

[1] Grossman D., Frieder O.: Information Retrieval: Algorithms and Heuristics. Kluwer Academic Publishers, 1998.

[2] Kopecký M.: Dokumentografické Informační Systémy - Slidy k přednášce DBI010, Praha, 2007.

[3] Korfhage R.: Information Storage and Retrieval. John Wiley & Sons, Inc., 1997.

[4] Koutrika G. a kol.: Précis: The Essence of a Query Answer. ICDE 2006, 69-78.

[5] Kowalski G.: Information Retrieval Systems: Theory and Implementation. Kluwer Academic Publishers, 1997.

[6] Manning Ch. a kol.: Introduction to Information Retrieval. Cambridge University Press, 2008.

[7] Pokorný J. a kol.: Dokumentografické informační systémy, Praha, 1998.

[8] Simitsis A. a kol.: Précis: from unstructured keywords as queries to structured databases as answers. International Journal on Very Large Data Bases (VLDB Journal), Volume 17, Number 1, 2008, 117-149.

[9] Štuller J.: Master Thesis: Précis querying over relational data, Prague, 2007.

[10] Oracle Database: Data Cartridge Developer's Guide, 10g Release 2, 2005.

[11] Oracle Text: Application Developer's Guide, 10g Release 2, 2005.

[12] Oracle Text Reference, 10g Release 2, 2005.

Příloha A. Uživatelská příručka

A.1. Instalace Précis indexu

Précis Index je určen pro databázový server Oracle Database 10g. V implementaci se používá Oracle Text a proto je potřeba, aby tato komponenta byla na serveru nainstalována před instalací Précis indexu.

Před spuštěním instalace je dále potřeba okopírovat knihovnu precis.dll do příslušného adresáře na serveru. Standardně se knihovny ukládají do adresáře ORACLE_HOME\bin. Zvolený adresář je pak potřeba doplnit do instalačního skriptu install.sql. Instalační skript je vytvořen pro SQL*Plus. V průběhu instalace budou vytvořeny následující databázové objekty:

- Uživatelské schéma, kde budou uloženy procedury. Výchozí název pro toto schéma je PRECISSYS, ale při instalaci může být název změněn. Příklady v této příručce předpokládají název schématu PRECISSYS.
- Role PRECISAPP. Uživatelé, kteří budou členy této role, budou mít právo používat Précis index.
- C++ knihovna obsahující funkce pro aktualizaci a vyhledávání v indexu.
- Ostatní databázové objekty, jako například balíčky, funkce, tabulky a veřejná synonyma.

A.2. Uživatelská práva

Aby uživatel mohl používat Précis indexy, musí mu být přiřazena role PRECISAPP. Tato role obsahuje práva na spouštění procedur uživatelského rozhraní indexu. Dále obsahuje roli CTXAPP, opravňující uživatele používat fulltextové vyhledávání Oracle Text. Tato role je nezbytná, protože pro tokenizaci dokumentů je použit Oracle Text.

Většina procedur Précis indexu je spouštěna s právy volajícího uživatele. Tím je zaručeno, že aplikace má přístup ke všem indexovaným tabulkám.

Pokud bude chtít uživatel indexovat i tabulky z jiných schémat, než je jeho vlastní, bude dále potřebovat práva CREATE ANY TRIGGER a ADMINISTER DATABASE TRIGGER. Tato práva jsou potřeba, protože aktuálnost indexu je udržována pomocí triggerů. První jmenované právo je vyžadováno pro trigger, který hlídá změny v indexovaných tabulkách provedené pomocí DML dotazů. Druhé zmíněné právo je potřeba pro vytvoření triggeru, který je spouštěn při provedení TRUNCATE TABLE nebo DROP TABLE.

Seznam všech práv potřebných pro používání Précis indexu je popsán v tabulce A-1.

Právo nebo role	Poznámka
PRECISAPP CREATE SEQUENCE CREATE TABLE CREATE TRIGGER CREATE VIEW	
CREATE ANY TRIGGER ADMINISTER DATABASE TRIGGER SELECT ON table	Tato práva jsou potřeba pouze v případě, že budou indexovány i tabulky z jiných schémat

Tabulka A-1 Seznam práv potřebných pro vytvoření indexu

A.3. Vytvoření a správa indexu

Vzhledem k tomu, že indexovány mohou být sloupce z různých tabulek, nelze Précis index vytvářet a spravovat pomocí standardních SQL příkazů. Pro správu indexu slouží balíček PRECIS. Všechny funkce z tohoto balíčku mají jako parametr název indexu. Jejich seznam je v tabulce A-2, podrobný popis pak dále v této kapitole.

Název funkce	Popis
create_index	Vytvoří nový index, tabulky a triggery potřebné pro index
drop_index	Odstraní index včetně všech uložených dat
drop_index_force	Stejné jako drop_index, pouze se nekontroluje, zda index již existuje
add_column	Přidá do indexu sloupec tabulky
drop_column	Odebere z indexu sloupec tabulky
sync_index	Provede aktualizaci indexu
set_as_multi_schema, set_as_single_schema	Nastaví, zda bude možné do indexu přidat pouze tabulky z vlastního schématu, nebo i z ostatních.
set_param	Nastaví parametr indexu
get_param	Načte parametr indexu

Tabulka A-2 Seznam funkci z balíčku PRECIS

Příklad

```
BEGIN
    --vytvoříme index s názvem PRECIS_IDX
  PRECIS.CREATE_INDEX('precis_idx');

    -- nastavíme některé parametry indexu
  PRECIS.SET_PARAM('precis_idx', 'index_node_size', 4066);
  PRECIS.SET_PARAM('precis_idx', 'index_blob_count', 16);

    -- přidáme do indexu sloupce
  PRECIS.ADD_COLUMN('precis_idx', 'DISC', 'TITLE');
  PRECIS.ADD_COLUMN('precis_idx', 'DISC', 'GENRE');
  PRECIS.ADD_COLUMN('precis_idx', 'TRACK', 'TTITLE');

    -- provede prvni synchronizaci indexu
  PRECIS.SYNC_INDEX('precis_idx');

    -- po synchronizaci indexu již můžeme měnit jen některé
    -- parametry
  PRECIS.SET_PARAM('precis_idx', 'paice_coef_and', 600);
  PRECIS.SET_PARAM('precis_idx', 'paice_coef_or', 500);
END;

-- provedeme dotazy z indexu
SELECT s.*, precissys.score(1)
  FROM precis$precis_idx$s s
  WHERE precissys.contains(doc_id, 'Metallica', 10, 1) > 0;

-- smažeme z indexu jeden sloupec
call PRECIS.DROP_COLUMN('precis_idx', 'DISC', 'TITLE');

-- smažeme celý index
call PRECIS.DROP_INDEX('precis_idx');
```

A.4. Dotazování v indexu

Aby bylo vyhledávání v indexovaných datech co nejjednodušší, je k tomuto účelu vytvořen doménový index. Tento index je formálně vytvořen nad tabulkou obsahující seznamy všech indexovaných dokumentů. Díky tomu lze v indexu vyhledávat pomocí standardního SQL příkazu. Přesto je však potřeba dodržet syntaxi dotazu, aby Oracle použil vytvořený doménový index.

Doménový index je formálně vytvořen na sloupcem DOC_ID tabulky PRECIS$*IDX_NAME*$D. Dotaz tedy musí být buď nad touto tabulkou, nebo nad pohledem PRECIS$*IDX_NAME*$S. Dotaz bude ve tvaru

```
SELECT s.*, precissys.score(k)
  FROM precis$index_name$s s
  WHERE precissys.contains(doc_id, dotaz, limit, k) > ohodnocení;
```

kde *dotaz* bude textový řetězec obsahující boolovský dotaz v indexu, *limit* maximální počet a *ohodnocení* minimální ohodnocení vrácených dokumentů. Score

je pomocný operátor, který je vázaný na operátor contains. Jeho přidání do dotazu neznamená žádné další výpočty – slouží pouze pro možnost vypsání ohodnocení.

Popsaný pohled navíc obsahuje volání funkce, která zobrazí začátek vyhledaného dokumentu. Při praktickém použití indexu by tento sloupec pravděpodobně nebyl potřeba a zbytečně by zdržoval vyhledávání. Všechny sloupce pohledu $S jsou popsány v tabulce A-3.

Sloupec	**Datový typ**	**Popis**
Owner	Varchar2(30)	Vlastník tabulky (schéma)
Table_name	Varchar2(30)	Název tabulky
Column_name	Varchar2(30)	Název sloupce
Row_ID	Rowid	Identifikátor řádky v tabulce
Data_type	Varchar2(106)	Datový typ sloupce tabulky
Doc_ID	Number	Číslo dokumentu – unikátní identifikátor v rámci daného indexu
Ucol_ID	Number	Číslo sloupce – unikátní identifikátor v rámci daného indexu
Text	Varchar2(4000)	Začátek vyhledaného dokumentu

Tabulka A-3 Sloupce pohledu PRECIS$*IDX_NAME*$S

Příklad

Následující dotaz vypíše 10 nejlépe ohodnocených dokumentů obsahující slovo „Metallica“ v databázi CD disků.

```
SELECT s.*, precissys.score(1)
  FROM precis$freedb_idx$s s
  WHERE precissys.contains(doc_id, ‘Metallica’, 100, 1) > 0;
```

A.5. Přehled funkcí uživatelského rozhraní

A.5.1. CREATE_INDEX

Vytvoří nový index, tabulky a triggery potřebné pro index. Název indexu může obsahovat pouze písmena bez diakritiky, číslice a znak „_“ (podtržítko). Název indexu bude převeden na velká písmena. Na velikosti písmen v názvu indexu tedy nezáleží. Maximální délka názvu indexu je 15 znaků. Parametry indexu je možné zadat po jeho vytvoření pomocí funkce SET_PARAM.

Syntaxe

```
PRECIS.CREATE_INDEX(index_name IN VARCHAR2);
```

A.5.2. DROP_INDEX

Odstraní index včetně všech uložených dat. Před odstraněním se kontroluje, zda je zadaný index definován. V případě, že je index poškozen a nejde touto procedurou odstranit, použijte proceduru DROP_INDEX_FORCE, která existenci indexu nekontroluje.

Syntaxe

```
PRECIS.DROP_INDEX(index_name IN VARCHAR2);
```

A.5.3. DROP_INDEX_FORCE

Stejné jako DROP_INDEX, pouze se nekontroluje, zda index již existuje.

Syntaxe

```
PRECIS.DROP_INDEX_FORCE(index_name IN VARCHAR2);
```

A.5.4. ADD_COLUMN

Přidá do indexu sloupec tabulky. Po přidání se neprovádí synchronizace indexu, takže nová data jsou do indexu přidána až po zavolání funkce SYNC_INDEX. Na rozdíl od názvu indexu je název uživatelského schématu, tabulky a sloupce citlivý na velikost písmen. Standardně je tedy potřeba uvádět tyto názvy velkými písmeny. Pokud budete chtít přidat do indexu i tabulky z cizích schémat, je potřeba nejprve spustit proceduru SET_AS_MULTI_SCHEMA.

Syntaxe

```
PRECIS.ADD_COLUMN(
  index_name   IN VARCHAR2,
  [table_owner IN VARCHAR2,]
  table_name   IN VARCHAR2,
  column_name  IN VARCHAR2
);
```

Příklad – tabulky i index jsou v jednom schéma

```
PRECIS.CREATE_INDEX(‘precis_idx’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘DISC’, ‘TITLE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘DISC’, ‘GENRE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘TRACK’, ‘TTITLE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘TRACK’, ‘EXTT’);
PRECIS.SYNC_INDEX(‘precis_idx’);
```

Příklad – tabulky jsou uloženy v jiném schématu než index

```
PRECIS.CREATE_INDEX(‘precis_idx’);
PRECIS.SET_AS_MULTI_SCHEMA(‘precis_idx’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘FREEDB’, ‘DISC’, ‘TITLE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘FREEDB’, ‘DISC’, ‘GENRE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘FREEDB’, ‘TRACK’, ‘TTITLE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘FREEDB’, ‘TRACK’, ‘EXTT’);
PRECIS.SYNC_INDEX(‘precis_idx’);
```

A.5.5. DROP_COLUMN

Odebere z indexu sloupec tabulky.

Syntaxe

```
PRECIS.DROP_COLUMN(
  index_name   IN VARCHAR2,
  [table_owner IN VARCHAR2,]
  table_name   IN VARCHAR2,
  column_name  IN VARCHAR2
);
```

A.5.6. SYNC_INDEX

Provede indexaci nových dat, která byla do indexovaných tabulek přidána od minulé synchronizace.

Syntaxe

```
PRECIS.SYNC_INDEX(index_name IN VARCHAR2);
```

A.5.7. SET_AS_MULTI_SCHEMA, SET_AS_SINGLE_SCHEMA

Nastaví, zda bude možné do indexu přidat pouze tabulky z vlastního schématu, nebo i z ostatních. Při vytvoření nového indexu lze přidávat pouze tabulky z vlastního schématu. Pro přidávání tabulek z ostatních schémat jsou potřeba speciální oprávnění – viz kapitola A.2. Procedury mohou být spuštěny kdykoli po vytvoření indexu i v případě, že v indexu jsou již nějaká data. Pokud jsou v indexu tabulky z cizích schémat, tak uživatel nesmí změnit typ indexu na „single schema“.

Syntaxe

```
PRECIS.SET_AS_MULTI_SCHEMA (index_name IN VARCHAR2);
PRECIS.SET_AS_SINGLE_SCHEMA (index_name IN VARCHAR2);
```

A.5.8. SET_PARAM

Tato funkce slouží pro nastavení parametru indexu. Většina parametrů však může být nastavena pouze po vytvoření indexu, před první synchronizací. Změna parametrů, které nastavují vlastnosti fyzického uložení indexu, po první synchronizaci indexu by způsobila jeho nefunkčnost. Seznam parametrů a jejich popis je uveden v kapitole 5.5.3. Index nekontroluje, zda zadaný parametr může být změněn.

Syntaxe

```
PRECIS.SET_PARAMETER(
  index_name  IN VARCHAR2,
  parameter   IN VARCHAR2,
  value       IN NUMBER
);
```

Příklad

```
PRECIS.CREATE_INDEX(‘precis_idx’);
PRECIS.SET_PARAM(‘precis_idx’, ‘index_node_size’, 4066);

PRECIS.ADD_COLUMN(‘precis_idx’, ‘DISC’, ‘TITLE’);
PRECIS.ADD_COLUMN(‘precis_idx’, ‘DISC’, ‘GENRE’);
PRECIS.SYNC_INDEX(‘precis_idx’);

PRECIS.SET_PARAM(‘precis_idx’, ‘paice_coef_and’, 600);
PRECIS.SET_PARAM(‘precis_idx’, ‘paice_coef_or’, 500);
```

A.5.9. GET_PARAM

Umožňuje načtení číselného parametru indexu. Seznam parametrů je uveden v kapitole 5.5.3.

Syntaxe

```
PRECIS.GET_PARAM(
  index_name     IN VARCHAR2,
  parameter_name IN VARCHAR2
)
RETURN NUMBER;
```

Příklad

```
BEGIN
  dbms_output.put_line(
    'Velikost bloku: ' ||
    PRECIS.GET_PARAM('FREEDB_IDX','index_node_size')
  );
END;

SELECT PRECIS.GET_PARAM('FREEDB_IDX','index_node_size') FROM DUAL;
```

A.5.10. GET_INDEX_NAMES

Vrátí seznam vytvořených Précis indexů pro zadaného uživatele. Seznam bude vrácen pomocí druhého argumentu procedury, který musí být typu „TABLE OF VARCHAR2(30)“.

Syntaxe

```
PRECIS_IDX.GET_INDEX_NAMES(
  user_name   IN  VARCHAR2,
  index_list  OUT t_index_names
);
```

Příklad

```
DECLARE
  idx_list PRECIS_IDX.t_index_names := PRECIS_IDX.t_index_names();
BEGIN
  PRECIS_IDX.GET_INDEX_NAMES(user, idx_list);

  FOR i IN 1..idx_list.COUNT LOOP
    dbms_output.put_line(idx_list(i));
  END LOOP;
END;
```

A.5.11. Operátor CONTAINS

Operátor CONTAINS se uvádí v klauzuli WHERE. Jako první parametr musí být zadán název sloupce, nad kterým je formálně vytvořen Précis index. To bude vždy DOC_ID. Poslední parametr je volitelný a používá se v případě použití operátoru SCORE v dotaze. Jeho hodnota musí být stejná, jako první parametr odpovídajícího operátoru SCORE z dotazu.

Syntaxe

```
SELECT *
  FROM PRECIS$INDEX_NAME$D
  WHERE PRECISSYS.CONTAINS(doc_id, dotaz, limit[, k]) > ohodnocení
```

A.5.12. Operátor SCORE

Operátor SCORE může být přidán do klauzule SELECT. Funguje jako pomocný operátor pro vypsání ohodnocení nalezeného dokumentu do odpovědi.

Syntaxe

```
SELECT d.*, PRECISSYS.SCORE(k)
  FROM PRECIS$INDEX_NAME$D d
  WHERE PRECISSYS.CONTAINS(doc_id, dotaz, limit, k) > ohodnocení
```

Příloha B. Přiložené CD

/oracle	Instalace databázového serveru Oracle Database 10g Express Edition
/precis_idx	Instalační skript pro Précis index
/source/precis_lib	Zdrojový kód knihovny v C++
/source/precis_ora	Zdrojové kódy databázových objektů
/test_db/cswiki	Testovací databáze – články české wikipedie
/test_db/freedb	Testovací databáze – databáze CD disků
/text	Text práce