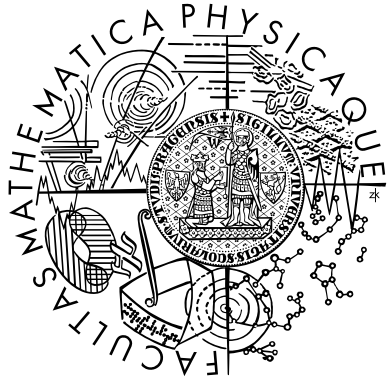
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jiří Dokulil

Dotazování nad RDF daty

Katedra softwarového inženýrství

Vedoucí diplomové práce: Prof. RNDr. Jaroslav Pokorný, CSc.
Studijní program: Informatika

Rád bych na tomto místě poděkoval všem, kteří mi byli nápomocni při psaní této práce. Zejména děkuji Prof. RNDr. Jaroslavu Pokornému, CSc. za cenné rady a připomínky. Děkuji také rodičům za jejich trpělivost a podporu při psaní práce.

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

V Praze dne 19.4.2006 Jiří Dokulil

Obsah

1 Úvod **7**

2 Resource Description Framework **8**

3 Dotazovací jazyk SPARQL **9**
3.1 Základní dotaz ve SPARQL 9
3.2 Dotazy kombinující více základních dotazů 10
3.2.1 Volitelný poddotaz 10
3.2.2 Sjednocení 11
3.3 Literály 11
3.4 Filtrující podmínky 12
3.5 Práce s více grafy 13
3.6 Další varianty zápisu dotazu 13
3.7 Vyjadřovací síla jazyka SPARQL 13
3.7.1 Jednoduchá cesta (Path Expression) 13
3.7.2 Nepovinná cesta (Optional Expression) 14
3.7.3 Sjednocení (Union) 14
3.7.4 Rozdíl (Difference) 14
3.7.5 Kvantifikace (Quantification) 15
3.7.6 Agregace (Aggregation) 15
3.7.7 Rekurze (Recursion) 15
3.7.8 Reifikace (Reification) 15
3.7.9 Kolekce a kontejnery (Collections and Containers) . . . 16
3.7.10 Jmenné prostory (Namespace) 16
3.7.11 Jazykové značky (Language) 16
3.7.12 Lexikální prostor (Lexical Space) 17
3.7.13 Prostor hodnot (Value Space) 17
3.7.14 Odvození (Entailment) 17
3.8 Vyjadřovací síla jazyka SPARQL - vyhodnocení 17

4 Reprezentace dat **19**
4.1 Uložení trojic 19
4.2 Uložení literálů 19
4.2.1 Problém typovaných literálů 20
4.2.2 Řešení problému typovaných literálů 21
4.3 Indexace 22

4.3.1 Literály . . . 22
4.3.2 Trojice . . . 22

5 Vyhodnocení SPARQL dotazů 24
5.1 Myšlenka překladu dotazů . . . 24
5.2 Terminologie . . . 24
5.3 Překlad základního grafového dotazu . . . 28
5.4 Kombinování poddotazů . . . 30
5.4.1 Kombinace pomocí OPTIONAL . . . 30
5.4.2 Kombinace pomocí UNION . . . 31
5.5 Filtrování výsledků poddotazu . . . 32

6 Implementace 35

7 Data 37
7.1 Číselníky . . . 37
7.2 Identifikace . . . 38
7.3 Kontakty . . . 38
7.4 Velikosti a četnosti . . . 39
7.4.1 Predikáty . . . 39
7.4.2 Subjekty . . . 39
7.4.3 Objekty . . . 40

8 Dotazy a měření 42
8.1 Testovací prostředí . . . 42
8.2 Měření dotazů . . . 42
8.3 Dotazy s URI v predikátu a bez literálů . . . 43
8.3.1 Seznam osob . . . 43
8.3.2 Seznam číselníků a jejich hodnot . . . 45
8.4 Dotazy s literály . . . 46
8.4.1 Jméno osoby se zadaným příjmením . . . 46
8.4.2 Jedna hodnota jednoho číselníku . . . 46
8.5 Dotazy na metadata . . . 47
8.5.1 Dotazy na třídy a vlastnosti . . . 48
8.5.2 Dotazy kombinující data a metadata . . . 49
8.6 Dotazy využívající číselníky . . . 49
8.6.1 Dotaz na jeden údaj o osobě na základě znalosti jiného 50
8.6.2 Seznam osobních údajů . . . 51
8.7 Dotaz obsahující nepovinný podgraf . . . 52
8.7.1 Seznam osob s nepovinným jménem . . . 52
8.7.2 Vnořené nepovinné bloky . . . 52
8.8 Dotazy obsahující číselné literály . . . 53

8.8.1 Seznam osob a titulů 53
8.9 Dotaz na objekt s velkou četností 55
8.10 Dotazy obsahující sjednocení 56
8.10.1 Příjmení a rodná příjmení 56
8.11 Dotazy s filtrující podmínkou 57
8.11.1 Seznam trojic s objektem typu decimal 57
8.11.2 Číselníky s alespoň 100 položkami 58
8.11.3 Osoby bez křestního jména 59
8.11.4 Vnořený nepovinný blok s filtrací 59
8.12 Test na přítomnost hodnoty v databázi 60
8.12.1 Existence identifikace 61
8.12.2 Existence osoby 61
8.13 Vyhodnocení měření 62
8.13.1 Uložení dat v databázi 62
8.13.2 Celková rychlost systému 63
8.13.3 Příčiny nedostatečného výkonu 63

9 Indexy **64**
9.1 Udržování jednoduchého indexu 64
9.2 Udržování složitého indexu 65
9.3 Detekce použitelnosti indexů 65
9.3.1 Jeden jednoduchý index 66
9.3.2 Více jednoduchých indexů 66
9.4 Použití indexů nad testovacími daty 66
9.4.1 Použití indexu na seznam osob 67
9.4.2 Použití indexu pro zjištění jména z příjmení 69
9.4.3 Index pro rodná čísla 69
9.4.4 Složitější dotaz využívající index nad rodnými čísly . . 70
9.4.5 Dotaz kombinující indexy 71
9.5 Přínos 72

10 Statistiky **73**
10.1 Statistiky o predikátech 73
10.2 Statistiky o subjektech 73
10.3 Statistiky o objektech 73
10.4 Konstrukce plánu pro vyhodnocení SQL dotazu 74

11 Závěr **75**

Literatura **76**

Název práce: *Dotazování nad RDF daty*
Autor: *Jiří Dokulil*
Katedra (ústav): *Katedra softwarového inženýrství*
Vedoucí diplomové práce: *Prof. RNDr. Jaroslav Pokorný, CSc.*
e-mail vedoucího: *Jaroslav.Pokorny@mff.cuni.cz*

Abstrakt: Jazyk RDF je základním stavebním kamenem Sémantického webu. Je to nástroj pro popis zdrojů na webu. Definuje však pouze formát a sémantiku dat, ne však způsob, jak se na tato data dotazovat.

Cílem této práce bylo vytvořit systém, který by umožnil dotazování nad RDF daty. Byla vytvořena částečná implementace jazyka SPARQL. Standard SPARQL je v současné době vyvíjen konzorciem W3C.

Pro uložení RDF dat byla vybrána relační databáze Oracle a bylo navrženo relační schéma, které umožňuje přeložit SPARQL dotazy na SQL dotazy.

Navržené postupy byly otestovány nad rozsáhlými RDF daty. Pro ukázkové SPARQL dotazy byla provedena měření rychlosti vyhodnocení převedených dotazů. Analýza faktorů negativně ovlivňujících rychlost vedla k návrhu dvou možných vylepšení. Jedno z nich bylo implementováno a jeho přínos byl otestován nad daty.

Klíčová slova: *RDF, dotazovací jazyky, sémantický web*

Title: *RDF querying*
Author: *Jiří Dokulil*
Department: *Department of Software Engineering*
Supervisor: *Prof. RNDr. Jaroslav Pokorný, CSc.*
Supervisor's e-mail address: *Jaroslav.Pokorny@mff.cuni.cz*

Abstract: The RDF is one of the basic technologies of the Semantic Web. It is a language describing resources on the web. It defines the format and semantics of such data but does not provide query capabilities.

The aim of this thesis is to create system capable of querying RDF data. We have created a partial implementation of the SPARQL query language. The SPARQL standard is currently being developed by the W3C Consorcium.

We have chosen the Oracle relational database to store the RDF data. The proposed database schema allows us to evaluate SPARQL queries by translating them into SQL queries.

The proposed methods have been tested on a large set of RDF data. We have created several examples of SPARQL queries, translated them to SQL, and measured the evaluation time of the translated queries. Afterwards, factors with negative impact on the evaluation speed have been analyzed. We have proposed two ways of improving the evaluation times based on this analysis, implemented one of them, and measured its effects.

Keywords: *RDF, query languages, semantic web*

1 Úvod

Jazyk RDF [7] je základním stavebním kamenem Sémantického webu [3]. Je to nástroj pro tvorbu popisu zdrojů, zvláště pak zdrojů na Internetu, jako například nadpis, autor a datum poslední změny webové stránky, autorská a licenční práva k dokumentu vystavenému na Internetu nebo třeba informace o časovém rozvrhu dostupnosti zdroje.

Zobecněním konceptu zdroje na Internetu je možné použít RDF pro popis objektů, které lze na Internetu pouze identifikovat, nikoliv však získat. Příkladem jsou třeba podrobnosti (velikost, cena, barva, ...) o zboží v elektronickém obchodě.

Jazyk RDF popisuje zdroje pomocí trojic, které je možné interpretovat jako orientovaný ohodnocený graf.

Samotné RDF je však pouze nástroj pro uložení dat. Definuje formát a sémantiku, ne však již způsob, jak se nad těmito daty dotazovat. V současné době existuje několik jazyků, které byly buď přímo vytvořeny pro dotazování nad RDF daty, nebo je možné je za tímto účelem upravit. Například RDQL [12], SeRQL [6] nebo RQL [1].

Cílem této práce je navrhnout systém, který by umožnil nad RDF daty pokládat dotazy. Jako dotazovací jazyk jsme zvolili jazyk SPARQL (SPARQL Query Language for RDF) [11]. Jde o nově vznikající jazyk, proto jej napřed podrobněji představíme v kapitole 3, kde jej také zařadíme do klasifikace RDF dotazovacích jazyků definované v [5]. V kapitole 4 vysvětlíme zvolený způsob uložení RDF dat a v kapitole 5 postup, kterým lze SPARQL dotazy nad takto uloženými daty vyhodnocovat.

Kapitola 6 popisuje vytvořenou prototypovou implementaci navržených postupů. S pomocí této implementace jsme provedli sadu výkonnostních testů. Daty, nad kterými testy probíhaly, a testy samotnými se zabývají kapitoly 7 a 8.

Kapitola 9 ukazuje jednu možnost, jak zlepšit výkon systému. I tuto funkci jsme implementovali a na základě měření zhodnotili její přínos pro výkon systému.

2 Resource Description Framework

RDF používá k popisu zdrojů trojice. Trojice se skládá ze subjektu, predikátu a objektu. Tvrzení odpovídající této trojici je takové, že subjekt má vlastnost určenou predikátem a hodnotou této vlastnosti je objekt.

V následujícím příkladu používáme jmenné prostory `ex` a `dc`. Jmenné prostory jsou zkratky pro zápis URI. Pokud jmenný prostor `ex` definujeme jako zkratku za http://expamle.org/, pak ex:book1 je zkratka, která je vyhodnocena jako http://example.org/book1.

```
<ex:book1> <dc:creator> <ex:person192>
```

Tato trojice zastupuje tvrzení, že autorem knihy, které jsme přiřadili URI ex:book1, je osoba, které náleží URI ex:person192. Další tvrzení pak můžou přidat informace o osobě ex:person192

Jiná možnost, jak vyjádřit předchozí tvrzení, je třeba jako

```
<ex:book1> <dc:creator> "John Smith"
```

Tvrzení říká, že autor ex:book1 je určen literálem "John Smith". Literál je konstantní výraz, který může být typovaný nebo netypovaný. Slouží k vyjádření čísel, časových údajů apod. pomocí jejich lexikální reprezentace. Vždy je možné nahradit literál vhodným URI. Pokud tak učiníme, tak bude navíc možné tvořit další tvrzení o tomto URI. To není možné v případě literálu, protože na rozdíl od URI může literál v trojici vystupovat pouze jako objekt.

RDF tvrzení lze reprezentovat jako orientovaný graf s ohodnocenými hranami, kde zdroje a literály tvoří uzly grafu a tvrzení odpovídají hranám grafu. Hrana je orientována od subjektu k objektu a ohodnocena predikátem.

Aplikace nemusí vyžadovat přímý přístup ke všem uzlům grafu. Některé uzly mohou sloužit jen jako propojení jiných uzlů a přístup k nim probíhá vždy přes hrany grafu z ostatních uzlů. Takové uzly je možné vytvořit jako anonymní, tj. uzly, které nemají přiřazené ani URI ani lexikální hodnotu.

Více o jazyku RDF se lze dočíst ve standardu [7].

3 Dotazovací jazyk SPARQL

Jazyk SPARQL vzniká pod záštitou konzorcia W3C, přesněji RDF Data Access Working Group. V současné době standard prochází fází připomínek a je velmi pravděpodobné, že bude pracovní skupině vrácen k přepracování s tím, že celé připomínkové řízení bude třeba zopakovat. Převážná část připomínek se však týká formálních definic jazyka, které jsou často nejasné a nejsou sladěny s formální definicí RDF. Lze tedy očekávat, že v samotném jazyce už nebude docházet k zásadním změnám, ale spíše jen k menším úpravám a hlavně zpřesňování jeho definice.

3.1 Základní dotaz ve SPARQL

Základní dotaz ve SPARQL vypadá například takto:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE
{
  <http://example.org/book1> dc:title ?title
}
```

Výsledkem tohoto dotazu jsou objekty všech trojic, jejichž subjektem je http://example.org/book1 a jako predikát v nich vystupuje dc:title.

V dotaze je možné použít více proměnných a trojic:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  ?book dc:title ?title .
  ?book dc:creator ?author
}
```

Obě trojice sdílejí jednu proměnnou, v obou případech na místě subjektu. Proto dvojice autor a titul ve výsledku musí vždy náležet jedné knize. Pokud má kniha více autorů nebo titulů, pak se do výsledku dostane kartézský součin titulů a autorů knihy.

Přesněji řečeno, výsledkem dotazu jsou všechna ohodnocení proměnných, která splňují podmínku za WHERE. V případě výše uvedených jednoduchých dotazů je splnění podmínky definováno tak, že trojice, které vzniknou z dotazu dosazením za proměnné, musí tvořit podgraf (až na isomorfismus anonymních uzlů) RDF grafu, nad kterým se dotazujeme.

3.2 Dotazy kombinující více základních dotazů

Základní dotazy je možné kombinovat do složitějších a tak vytvořit dotazy, které pracují s nepravidelnou strukturou RDF dat, například je možné ošetřit situaci, kdy je některý z údajů nepovinný nebo nedosažitelný a odpovídající trojice se nemusí v datech vyskytovat.

3.2.1 Volitelný poddotaz

Pomocí klíčového slova OPTIONAL můžeme část dotazu označit za volitelnou. Proměnné, které se vyskytují pouze v této části, mohou zůstat neohodnocené a v tom případě nemusí (dokonce nesmí) volitelná část dotazu existovat jako podgraf dotazovaného RDF grafu.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  ?book dc:title ?title .
  OPTIONAL { ?book dc:creator ?author }
}
```

Tento dotaz vrátí název všech knih, které název mají, a k těmto knihám i autora, pokud je v datech uložen. Za proměnné ?book a ?title musí být dosazeno. Proměnná ?author bude ohodnocena právě tehdy, když v grafu existuje trojice se subjektem ?book a predikátem dc:creator. Pokud taková trojice existuje, pak její objekt musí být přiřazen do proměnné ?author. Pokud jich je takových trojic více, ve výsledku se objeví všechna přípustná dosazení. Pokud taková trojice neexistuje, zůstane proměnná ?author neohodnocená.

3.2.2 Sjednocení

Pokud dvě části dotazu spojíme klíčovým slovem UNION, pak je ohodnocení proměnných platným výsledkem dotazu, pokud odpovídá alespoň jedné z částí.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  { ?book dc:title ?title }
  UNION
  { ?book dc:creator ?author }
}
```

Výsledek tohoto dotazu bude obsahovat dvě části. V jedné bude seznam titulů a k nim prázdné ?author, ve druhé seznam autorů a prázdné ?title.

3.3 Literály

Při zápisu dotazu lze na místě objektu uvést literál.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?author
WHERE
{
  ?book dc:title "Bylo nás pět" .
  ?book dc:creator ?author
}
```

Význam dotazu se nemění. Pořád platí, že ohodnocení proměnných se dostane do výsledku, pokud po dosazení do trojic dotazu vznikne podgraf grafu, nad kterým se dotazujeme. Tedy dotaz má vrátit autora knihy Bylo nás pět.

Kromě čistého literálu je možné určit požadovaný datový typ nebo jazykovou značku. Pak musí tento typ nebo značka být stejné u literálu v dotazovaném grafu.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?author
WHERE
{
  ?book dc:title "Bylo nás pět"@cz .
  ?book dc:creator ?author^^xsd:string
}
```

Tento dotaz vrací autora knihy, jejíž český název je "Bylo nás pět", pokud je jméno autora uloženo jako řetězec.

3.4 Filtrující podmínky

K dotazu je možné přidat i jiná omezení než jen existenci/neexistenci trojic. Za klíčové slovo FILTER lze uvést podmínku, která je sestavena z konstant, aritmetických operací, porovnání, volání funkcí a proměnných.

Každé přípustné ohodnocení proměnných, které splňuje poddotaz omezený podmínkou, musí zároveň splňovat i tuto podmínku.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title
WHERE
{
  ?book dc:title ?title .
  ?book ns:price ?price .
  FILTER ( ?price < 100 )
}
```

Tento dotaz vrátí názvy knih, které mají uvedenou cenu a tato cena je menší než 100. Pokud by kniha měla uvedeno více cen, pak musí alespoň jedna z nich být menší než 100.

Literál 100 v podmínce je zkratka za "100"^^xsd:integer, tedy literál s hodnotou 100 a typem integer.

SPARQL definuje sadu vestavěných funkcí, které umožňují například extrahovat z hodnoty její datový typ nebo jazykovou značku.

3.5 Práce s více grafy

SPARQL umožňuje v jednom dotaze přistupovat k více RDF grafům. To zhruba odpovídá práci s více databázovými schématy. Oproti databázovým schématům je zde jeden zásadní rozdíl. Jako jméno grafu, ke kterému dotaz přistupuje, je možné použít proměnnou, která je vázána i v jiné části dotazu.

3.6 Další varianty zápisu dotazu

Standard definuje několik zkratek, které lze použít pro zápis dotazu. Ty ukážeme na příkladech v 3.7.

Dále umožňuje kromě SELECT použít i SELECT DISTINCT, které způsobí, že z výsledku jsou odstraněny duplicity, stejně jako u SQL.

Je možné použít i formu SELECT *, což vybere všechny proměnné.

3.7 Vyjadřovací síla jazyka SPARQL

V následující části popíšeme možnosti jazyka SPARQL pomocí příkladů užití definovaných v [5]. Tak získáme i možnost srovnat jeho sílu s ostatními jazyky.

Pro zjednodušení zápisu dotazů byla ze všech dotazů vynechána úvodní část s definicí prefixů. Jde o tyto řádky:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX s: <http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-
           query/sample.rdf#>
```

3.7.1 Jednoduchá cesta (Path Expression)

Dotaz má vrátit jméno autora publikace X, přičemž X je URI.

```
SELECT ?name
WHERE { <X> s:author ?x . ?x ?p ?z . ?z s:name ?name }
```

Jedná se o základní grafový dotaz. Prostřední trojice je v dotazu uvedena, protože v dotazovaných datech jsou autoři uloženi v kolekci. Tato trojice tedy přistupuje k jednotlivým členům kolekce.

3.7.2 Nepovinná cesta (Optional Expression)

Dotaz má vrátit jméno autora publikace X a jeho email, pokud je zadán.

```
SELECT ?name ?email
WHERE { <X> s:author ?x . ?x ?p ?z . ?z s:name ?name .
  OPTIONAL { ?z s:email ?email } }
```

Tento dotaz je přímo ukázkové použití bloku OPTIONAL.

3.7.3 Sjednocení (Union)

Dotaz má vrátit názvy všech témat a všech publikací.

```
SELECT ?title
WHERE { { ?x a s:Topic ; rdfs:label ?title}
  UNION { ?x a s:Publication ; s:title ?title } }
```

Tento dotaz je zapsán jedním z možných zkrácených zápisů. Pokud je k oddělení trojic použit středník místo tečky, pak následující trojice je uvedena jen jako dvojice s tím, že subjekt je shodný s předchozí trojicí.

Dotaz také používá klíčové slovo a, což je zkratka za rdf:type.

3.7.4 Rozdíl (Difference)

Dotaz má vrátit názvy všech témat, které nejsou názvy publikací.

```
SELECT ?title
WHERE { ?x a s:Topic ; rdfs:label ?title .
  OPTIONAL { ?y a s:Publication ; s:title ?title } .
  FILTER(!bound(?y)) }
```

Rozdíl je řešen pomocí OPTIONAL bloku a testování v něm přidané proměnné y na to, že do ní nebylo v bloku OPTIONAL dosazeno. Vykřičník značí negaci a funkce bound je vestavěná funkce jazyka SPARQL, která a testuje, jestli bylo do proměnné dosazeno. Jde o tzv. negaci selháním (Negation as Failure) známou z logického programování.

3.7.5 Kvantifikace (Quantification)

Dotaz má vrátit osoby, které jsou autory všech publikací.

```
SELECT DISTINCT ?person
WHERE { ?person a s:Person .
  OPTIONAL { ?pub a s:Publication .
    OPTIONAL { ?pub s:author ?x . ?x ?y ?person } .
    FILTER (!bound(?x)) } .
  FILTER(!bound(?pub)) }
```

Dotaz zapíšeme pomocí převodu obecného kvantifikátoru na negaci existenčního kvantifikátoru, která je realizována pomocí negace selháním představené u předchozího dotazu.

Poddotaz začínající prvním OPTIONAL vrací publikace, jejichž autorem není ?person. Vnější část vybírá pouze ty osoby, pro které taková publikace neexistuje.

3.7.6 Agregace (Aggregation)

Tento příklad je definován jako dotaz, který má vrátit počet autorů publikace.

SPARQL neposkytuje agregační funkce. Navíc jako výsledek dotazu v SPARQL vystupují pouze proměnné a do těch mohou být dosazeny pouze uzly RDF grafu. Proto není ani možné agregace simulovat.

3.7.7 Rekurze (Recursion)

Tento příklad je definován jako dotaz, který má vrátit všechna témata patřící (rekurzivně) pod téma "Information Systems". SPARQL neumožňuje pokládat rekurzivní dotazy.

3.7.8 Reifikace (Reification)

Dotaz má vrátit osobu, která klasifikovala publikaci X.

```
SELECT ?person
WHERE { ?x rdf:subject <X> .
  ?x rdf:predicate s:isAbout .
  ?x dc:creator ?person }
```

Stejně jako samotné RDF, ani SPARQL nemá přímou podporu pro práci s reifikací. Protože reifikace v RDF je řešena pouze přidáním několika standardizovaných predikátů pro trojici, subjekt, predikát ap., není problém k těmto trojicím pomocí SPARQL přistupovat.

3.7.9 Kolekce a kontejnery (Collections and Containers)

Dotaz má vrátit prvního autora publikace X.

```
SELECT ?name
WHERE { <X> s:author ?x . ?x rdf:_1 ?z .
  ?z s:name ?name }
```

Podpora pro kolekce je pouze omezená. Je sice možné zjistit prvního autora, ale již nelze zjistit například autora posledního, protože SPARQL neobsahuje podporu pro práci s kolekcemi. Pouze umožňuje položit dotaz na trojice, pomocí kterých je kolekce uložena.

3.7.10 Jmenné prostory (Namespace)

Dotaz má vrátit všechny zdroje, jejichž URI pochází ze jmenného prostoru, jehož adresa začíná "http://www.aifb.uni-karlsruhe.de/".

SPARQL pracuje jen s celými URI. Jmenné prostory lze použít pouze jako zkratky dotazů (pomocí klíčového slova PREFIX).

3.7.11 Jazykové značky (Language)

Dotaz má vrátit německý název publikace, jejíž anglický název je "Database Management".

```
SELECT ?name
WHERE { ?x rdfs:label "Database Management"@en , ?name .
  FILTER (lang(?name)="de") }
```

Tento dotaz používá další zkrácenou formu zápisu. Čárka oddělující trojice se chová obdobně jako středník, s tím rozdílem, že místo trojice se zapíše jen objekt. Subjekt i predikát jsou převzaty z předchozí trojice. Dotaz také používá funkci lang, která vrací jazykovou značku literálu jako řetězec.

3.7.12 Lexikální prostor (Lexical Space)

Dotaz má vrátit všechny publikace, které mají počet stránek “08”.

```
SELECT ?name
WHERE { ?x a s:Publication ; s:pages "08" }
```

Hodnota v datech musí přesně odpovídat literálu z dotazu. Pokud je v datech uvedeno “8”, pak údaj neodpovídá dotazu.

3.7.13 Prostor hodnot (Value Space)

Dotaz má vrátit publikace, jejichž počet stránek je číslo 8.

```
SELECT ?name
WHERE { ?x a s:Publication ; s:pages 8 }
```

Číslo 8 v dotazu je zkratka za “8”^^xsd:integer, tj. typovaný literál typu integer s hodnotou 8. Oproti předchozímu dotazu zde nezáleží na zápisu. Dotazu odpovídá jak “8”^^xsd:integer, tak “08”^^xsd:integer.

3.7.14 Odvození (Entailment)

Dotaz má vrátit všechny instance třídy Publication a jejích potomků. Tento dotaz není možné ve SPARQL zapsat. Protože se RDFS zapisuje jako RDF trojice, je samozřejmě možné je do databáze uložit, ale standard SPARQL je neumožňuje interpretovat jinak než jako obyčejné RDF trojice. Protože SPARQL nemá podporu pro rekurzivní dotazy, není ani možné provést interpretaci RDFS ručně jako součást dotazu.

3.8 Vyjadřovací síla jazyka SPARQL - vyhodnocení

Hlavní slabina jazyka SPARQL při porovnání s ostatními jazyky spočívá v chybějící podpoře pro RDFS odvozování. Kvůli absenci rekurzivních dotazů není možné toto odvození ani simulovat, protože si neporadí správně se subClassOf. Z osmi jazyků, které již byly zařazeny do použité klasifikace pouze dva neumožňují odvozovaní, tři umožňují jeho simulaci a tři mají přímou podporu.

Na druhou stranu pouze tři z osmi jazyků mají podporu pro práci s jazykovou značkou literálů. Pouze polovina ze srovnávaných jazyků obsahuje přímou podporu pro práci prostorem hodnot.

Celkově nevychází SPARQL mezi srovnávanými jazyky jako nejsilnější. Naopak ve srovnání s populárním SeRQL [6] vychází jako jednoznačně slabší. SeRQL dovoluje více než SPARQL, ale SPARQL neobsahuje nic navíc. Stejně dopadlo i srovnání s RDFQL [13]. Při srovnání s ostatními jazyky již není výsledek jednoznačný, vždy existují funkce, které má navíc SPARQL, ale také funkce, které ma navíc druhý srovnávaný jazyk.

4 Reprezentace dat

V této části se budeme zabývat tím, jak uložit RDF data do relační databáze tak, aby se nad nimi bylo možné efektivně dotazovat pomocí SPARQL. Pro uložení používáme databázi Oracle Database 10g, přesto jsou základní principy s menšími úpravami použitelné i v dalších SŘBD. V počáteční fázi vývoje jsme úspěšně otestovali varianty systému pro Microsoft SQL Server a IBM DB2. Konečná volba padla na Oracle, protože byl dostupný dostatečně výkonný server s tímto SŘBD.

Napřed se zaměříme na uložení RDF trojic a literálů do tabulek. V další části popíšeme, jak zajistit efektivitu přístupu k uloženým údajům.

4.1 Uložení trojic

Zcela přirozený způsob, jak reprezentovat RDF trojice, je tabulka se třemi sloupci, což je vlastně množina trojic. Proto jsme také zvolili relační databázi jako prostředek pro uložení RDF dat. Myšlenka tabulky se třemi sloupci je dobrý základ, ale některé vlastnosti RDF vyžadují složitější reprezentaci. Paleta toho, co se může vyskytnout na jednotlivých pozicích v trojici, je totiž složitější, což vyžaduje i složitější reprezentaci.

Protože se bude tabulka trojic při vyhodnocování převedených SPARQL dotazů často účastnit spojení, je vhodné udržet ji co nejjednodušší. Toho lze dosáhnout tím, že v tabulce trojic budou pouze umělé identifikátory, které budou sloužit jako cizí klíče do tabulky literálů.

Tuto tabulku jsme pojmenovali `TRIPLES`. Její schéma vypadá takto:

TRIPLES		
id	number(18)	primarní klíč
subject	number(18)	
predicate	number(18)	
object	number(18)	

4.2 Uložení literálů

Literály jsou uloženy v tabulce `LITERALS`. Ta obsahuje

- sekvenčně generovaný primární klíč
- textovou hodnotu literálu

- volitelně jazykovou značku, která určuje jazyk literálu
- volitelně určení datového typu literálu

Aby správně fungovalo vyhodnocení dotazů, musí být jeden literál uložen v této tabulce nejvýše jednou. Při detekci duplicit jsou dva literály považovány za identické, pokud se shodují ve všech čtyřech složkách. Duplicity nejsou povoleny, ale také nejsou potřeba. Všechny stejné literály jsou v trojicích reprezentovány jedním identifikátorem.

Pokud bychom připustili více výskytů jednoho literálu, pak by bylo nutné pro porovnání rovnosti jednoho členu z trojice s členem jiné trojice spojit nejen dvakrát tabulku `TRIPLES`, ale i dvakrát tabulku `LITERALS`. Toto porovnání je velmi časté, protože odpovídá použití jedné proměnné ve více trojicích. Bez unikátnosti literálu by tak došlo k výraznému zpomalení jedné ze základních operací potřebných k vyhodnocení SPARQL dotazů.

Datový typ literálu je určen pomocí URI, proto je jejich množství prakticky neomezené. Proto je nutné datový typ ukládat jako řetězec s touto URI.

Identifikátory jazyka se řídí standardem RFC 3066 [2]. Ten dovoluje tvorbu víceúrovňových identifikátorů, přičemž pouze obsah prvních dvou úrovní je standardizován. Pro další úrovně definuje pouze omezení na jejich délku a přípustné znaky. Proto je nutné i tyto identifikátory uchovávat jako řetězce.

4.2.1 Problém typovaných literálů

Při vyhodnocení SPARQL dotazů je nutné literály porovnávat. To lze provést buď porovnáním řetězců, kterými byl literál zadán, nebo porovnáním literálů se znalostí sémantiky použitého datového typu. Příkladem je porovnání 2 a 2.0. Řetězcově jde o dva různé literály. Pokud je však interpretujeme jako desetinná čísla, jde o stejné literály. SPARQL toto chování u některých datových typů za určitých okolností vyžaduje.

Pokud dotaz obsahuje literál, jehož typ je například integer, pak je nutné vyhledávat jeho hodnotu i mezi hodnotami typu decimal. Na druhou stranu, pokud hledáme ohodnocení proměnných a pak jej aplikujeme na trojice dotazu, není možné do proměnné na jednom místě dosadit hodnotu typu integer a na jiném místě stejnou hodnotu, ale typu decimal.

Aby bylo možné tuto funkčnost rozumně používat, potřebujeme prostředek, který dovolí rychle najít v tabulce literálů ty záznamy, jejichž hod-

nota se při interpretaci uvažující datové typy rovná jedné zvolené hodnotě. Porovnání beroucí do úvahy datový typ však není triviální, a pokud by se provádělo jako podmínka při selekci z tabulky `LITERALS`, bylo by nutné tabulku projít celou a porovnání provést pro každý řádek. Při vyhodnocení jednoho dotazu však může být nutné vyhledávat takové záznamy i vícekrát. Proto není přijatelné, aby každé vyhledání znamenalo prohledat sekvenčně celou tabulku `LITERALS`. Tím by příliš utrpěl výkon celého systému.

4.2.2 Řešení problému typovaných literálů

Možné řešení je uložit ke každému literálu zároveň jeho reprezentaci v příslušném datovém typu. Přímé uložení by vyžadovalo přidat k tabulce `LITERALS` tolik sloupců, kolik datových typů by měl systém podporovat. Tím by tabulka příliš narostla a navíc by řešení nebylo snadno rozšiřitelné o nové typy.

Námi navržené řešení rozšiřuje tabulku `LITERALS` pouze o jeden sloupec, který obsahuje identifikátor, jímž je reprezentována typovaná hodnota literálu. Identifikátor slouží jako cizí klíč do jedné z množiny tabulek, z nichž každá odpovídá jednomu datovému typu. Tyto tabulky budeme v dalším textu značit pomocí `TYPED(datový typ)`, kde datový typ je vždy datový typ definovaný v XSD [4], proto budeme vynechávat z názvu typu jmenný prostor XSD. Příkladem mohou být tabulky `TYPED(integer)` a `TYPED(decimal)`.

Primární klíče tabulek `TYPED(*)` jsou generovány z jedné společné sekvence. Pokud jeden typovaný literál lze interpretovat kvůli konverzím jako literál více typů, pak je odpovídající hodnota uložena do tabulek pro všechny typy, na které ji lze zkonvertovat, a ve všech je použit stejný primární klíč. Aby tento systém mohl fungovat, je nutné při zakládání jednoho typovaného literálu založit záznamy pro všechny možné konverze.

Jako příklad uvažme hodnotu 2 typu integer. Pokud je tato hodnota poprvé vložena do databáze, pak je nutné založit v tabulce `TYPED(integer)` záznam s hodnotou 2, ale zároveň také do tabulky `TYPED(decimal)` uložit hodnotu 2 se stejným primárním klíčem.

Pokud však bude třeba uložit hodnotu 2.5, pak bude založen jen záznam v tabulce `TYPED(decimal)`.

Tabulka `LITERALS` vypadá celkově takto:

LITERALS		
id	number(18)	primarní klíč
value	nvarchar(2000)	
type	nvarchar(2000)	
lang	varchar(100)	
typed	number(18)	

Jako příklad tabulek uchovávajících typované hodnoty uvádíme schéma tabulky `TYPED(decimal)`.

TYPED(decimal)		
id	number(18)	primarní klíč
value	number	

4.3 Indexace

V této části se budeme zabývat metodami, jak zajistit efektivní přístup k datům.

4.3.1 Literály

U tabulky `LITERALS` budeme určitě potřebovat index nad hodnotou literálu (sloupec `value`), protože vyhledávání jedné konkrétní hodnoty literálu je při vyhodnocování SPARQL dotazů běžné.

Není potřeba vytvářet indexy nad sloupci `type` a `lang`. Jazyk SPARQL neumožňuje položit dotaz, který by vyhledával literály pouze pomocí datového typu nebo jazykové značky. Na druhou stranu je nutné vytvořit index nad sloupcem `typed`, aby se zabránilo sekvenčnímu průchodu při dotaze na typovaný literál. Pro tento typ dotazů je vhodné založit také indexy nad sloupcem `value` tabulek `TYPED(*)`.

4.3.2 Trojice

U tabulky `TRIPLES` se výrazně liší obsah a využití sloupců `subject` a `object` od sloupce `predicate`. Kardinalita `subject` a `object` bude typicky velká, protože v databázi budou tvrzení o mnoha různých zdrojích a jejich hodnoty budou různé. Na druhou stranu predikáty, které spíše odpovídají schématu databáze, obvykle pochází z velmi malé množiny.

V dotazech se bude běžně objevovat omezení na jednu konkrétní hodnotu objektu. To odpovídá jednoduché selekci v relační databázi. Této podmínce

bude vyhovovat spíše menší množství trojic. Přesto můžou existovat hodnoty objektů, pro které bude selekci vyhovovat velké množství trojic.

Uvažme například databázi osob a literál "muž", který se vyskytuje jako objekt v polovině trojic určujících pohlaví. Další příklad je literál "1", který se může v různých rolích (příznak, známka, počet) vyskytovat ve velkém množství trojic.

Časté může být i omezení na jednu konkrétní hodnotu subjektu. Paralela tohoto omezení v SQL je omezení na rovnost primárního klíče a konstanty. Přestože tomuto omezení v RDF bude obvykle vyhovovat více než jedna trojice, jejich množství bude spíše omezené, protože množství tvrzení o jednom zdroji nebývá příliš velké.

Na druhou stranu v případě predikátů bude velmi častý dotaz, který má vracet pouze trojice s jedním konkrétním predikátem, a výsledkem bude velké množství trojic.

Proto má smysl v případě sloupce `predicate` uvážit použití bitmapového indexu místo obvyklého B-stromu.

Výhodné by mohlo být zajistit, aby se trojice se stejným predikátem na disku nacházely blízko sebe. Sekvenční zpracování všech trojic s jedním predikátem by se zrychlilo. Toto uspořádání dat na disku lze dosáhnout tím, že tabulku `TRIPLES` vytvoříme jako tabulku organizovanou jako index (Index Organized Table – tabulka je uložena v B-stromu) s `predicate` jako prvním sloupcem indexu.

V části zabývající se testy výkonu dotazů porovnáme verzi používající běžný index založený na B-stromu s verzí založenou na bimtapovém indexu a verzí využívající tabulku organizovanou jako index. U poslední možnosti navíc vyzkoušíme i případ, kdy tabulku označíme jako paralelně zpracovatelnou.

Napřed však musíme objasnit postup, kterým jsou SPARQL dotazy vyhodnocovány nad reprezentací představenou v této kapitole. Postup překladu SPARQL dotazů do SQL je popsán v následující kapitole.

5 Vyhodnocení SPARQL dotazů

RDF data ukládáme do relační databáze, proto vyhodnocení SPARQL dotazů provádíme jejich překladem do jazyka SQL. Výsledkem SQL dotazu jsou přípustná ohodnocení proměnných SPARQL dotazu, ze kterého SQL dotaz vznikl.

5.1 Myšlenka překladu dotazů

SPARQL dotaz rozdělíme na poddotazy, z nichž některé jsou základní a je možné je přímo přeložit do SQL, zatímco některé vzniknou vhodnou kombinací jednodušších SQL dotazů, které odpovídají menším částem dotazu.

5.2 Terminologie

SPARQL dotaz je možné nahlížet jako derivační strom, kterým byl zkonstruován podle gramatiky jazyka (ta je součástí standardu), kde uzel ohodnotíme neterminálem, jehož derivací vznikl, případně hodnotou terminálu, pokud uzel vznikl z terminálu. Částí SPARQL dotazu (neboli poddotaz) je pak část tohoto stromu začínající jedním uzlem a obsahující všechny jeho potomky.

Uzly rozdělíme do několika kategorií:

1. Uzly, které pouze pomáhají konstrukci gramatiky, například neterminály, jež pouze slučují několik dalších, aby byly snadněji použitelné v jiné části gramatiky.

2. Uzly, které pomáhají tvořit syntax jazyka, ale pro vyhodnocení nemají význam. Jde například o uzel, jenž říká, že kolem grafového výrazu za WHERE musí být závorky.

3. Uzly ohodnocené neterminálem Triples. Tyto uzly a jejich potomci definují základní grafový dotaz (seznam trojic).

4. Potomci uzlů předchozího typu. Jejich převodem na seznam trojic se podrobněji zabývat nebudeme, protože jde jen o nezajímavý technický problém.

5. Uzly, které kombinují dva poddotazy do jednoho většího poddotazu. Jde o uzel GraphPattern, jeho pravého potomka (ohodnoceného GraphPatternNotTriples) a potomka tohoto potomka, který je ohodnocen GroupOrUnionGraphPattern (odpovídá spojení více poddotazů pomocí UNION) nebo OptionalGraphPattern (odpovídající spojení základního dotazu a základního dotazu uzavřeného do bloku OPTIONAL).

6. Uzly typu Constraint, které definují podmínku, jíž jsou filtrovány výsledky grafového dotazu.

7. Potomci uzlu Constraint. Ty definují podmínku filtrace.

Standard definuje i další typy uzlů, ale pro naši omezenou implementaci vystačíme s těmito. Příklad derivačního stromu ukazuje obrázek 1. U neterminálů je uveden jejich název a kategorie, u terminálů pouze hodnota. Jde o strom následujícího dotazu.

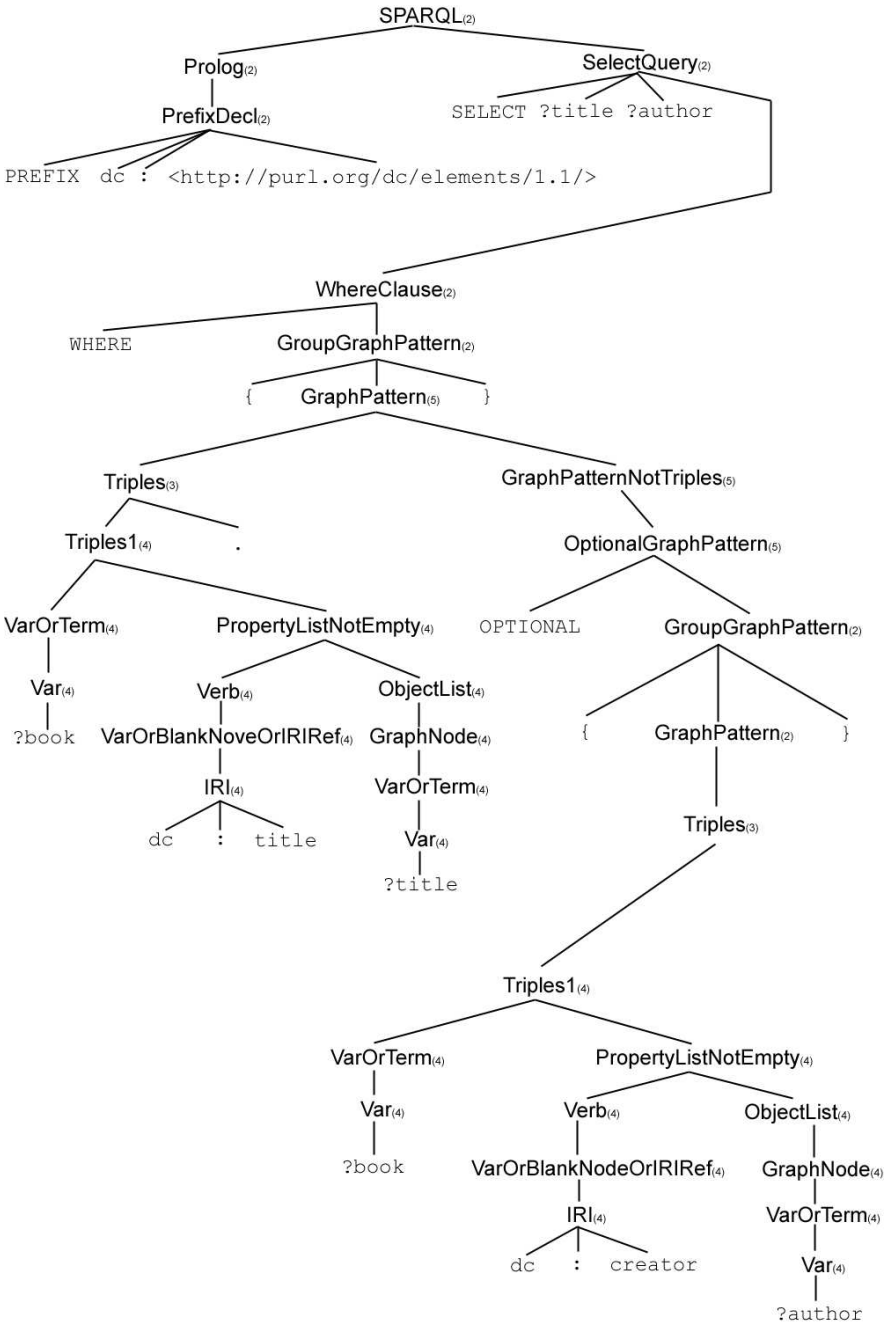
```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  ?book dc:title ?title .
  OPTIONAL { ?book dc:creator ?author }
}
```

Pro potřeby překladu dotazů ze SPARQL do SQL definujeme nový strom na základě původního.

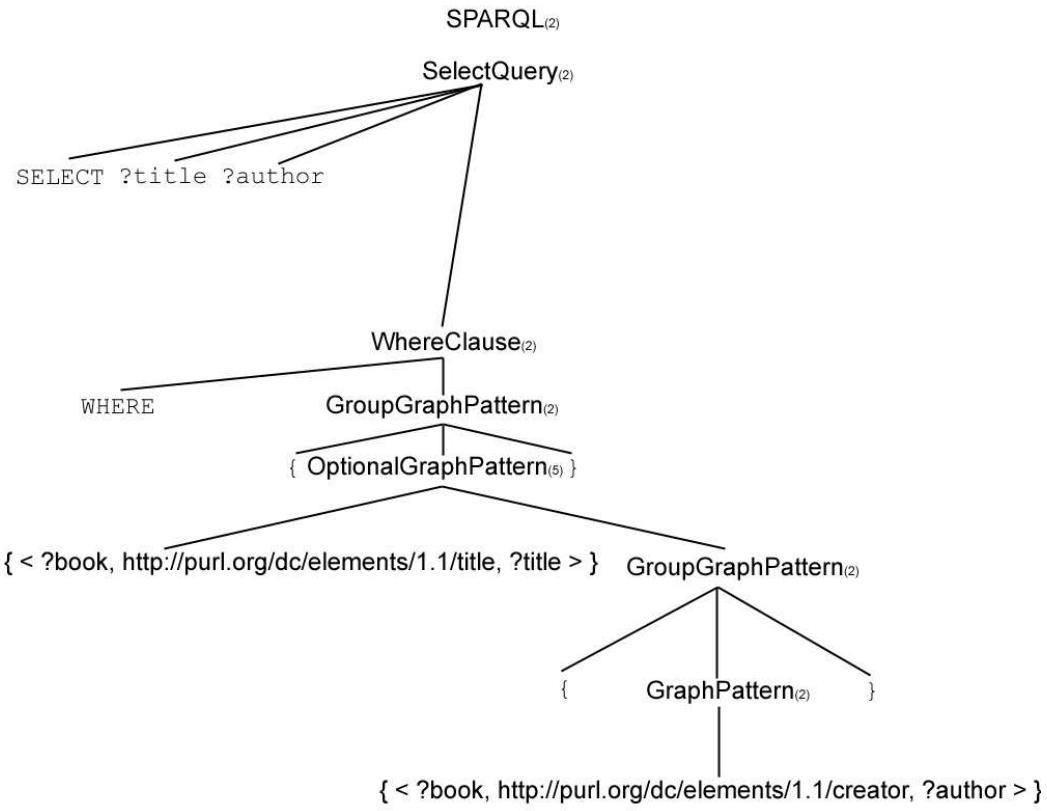
- Odstraníme definice jmenných prostorů (klíčové slovo PREFIX) a nahradíme jejich použití plným zněním.

- Uzly Triples a jeho potomky (typ 3 a 4) sloučíme do jednoho uzlu, který ohodnotíme seznamem trojic definovaných tímto podstromem.

- Sloučíme trojice typu 5 (tedy uzel GraphPattern, GraphPatternNotTriples a GroupOrUnionGraphPattern nebo OptionalGraphPattern). Nově vzniklý uzel ohodnotíme stejně, jako byl původně ohodnocen potomek GraphPatternNotTriples, tj. GroupOrUnionGraphPattern nebo OptionalGraphPattern.



Obrázek 1: Příklad derivačního stromu



Obrázek 2: Příklad upraveného derivačního stromu

Upravený strom z obrázku 1 ukazuje obrázek 2. V dalším textu budeme vždy pracovat s upraveným stromem dotazu.

Pro uzel n, který je buď Triples nebo leží na cestě od Triples ke kořeni, definujeme $\hat{n}$ jako SPARQL poddotaz, který odpovídá podstromu začínajícímu n. V případě uzlu typu Triples jde například o seznam trojic.

Dále označme $var(n)$ množinu proměnných, které jsou použity v dotaze $\hat{n}$.

Pak $F(n)$ je SQL dotaz, jehož výsledkem je množina přípustných (vůči $\hat{n}$) ohodnocení proměnných z $var(n)$.

Pro n uzel typu 1 a 2 platí, že má jednoho potomka m a $F(n) = F(m)$.

Pro popis algoritmů budeme také používat lehce upravenou relační algebru. Úpravu zavádíme, protože budeme spojovat převážně více výskytů tabulek `LITERALS` a `TRIPLES`. Běžný postup s přejmenováním jednotlivých sloupců by příliš komplikoval zápis.

Proto každou použitou relaci rovnou přejmenujeme a k jejím sloupcům budeme přistupovat pomocí tečkové notace. Následující tabulka ukazuje operace, které budeme používat.

$\text{TRIPLES}_{\text{T}_1}$	`TRIPLES` přejmenovaná na T_1
$\text{TRIPLES}_{\text{T}_1} \times \text{TRIPLES}_{\text{T}_2}$	kartézský součin dvou `TRIPLES`
$\text{T}(\varphi)$	selekce relace T na podmínku φ
$\text{T}_1.subject = \text{T}_2.predicate$	podmínka na rovnost hodnot sloupců v tabulkách přejmenovaných na T_1 a T_2
$\text{T}_1.subject =$ "val"	podmínka na rovnost hodnoty ve sloupci a konstanty
$\varphi_1 \wedge \varphi_2$	konjunkce podmínek
$T(old \rightarrow new)$	přejmenování sloupce old na new
$T[col1, col2]$	projekce na sloupce col1 a col2

SQL dotazy budeme často zapisovat pomocí této varianty relační algebry. Jejich překlad do skutečného SQL je zcela přímočarý.

5.3 Překlad základního grafového dotazu

Uvažme dotaz

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
```

```
  ?book dc:title ?title .
  ?book dc:creator ?author
}
```

Dotaz chceme přeložit tak, aby odpovídal výrazu N:

$$
\begin{aligned}
N_1 &:= \text{TRIPLES}_{\text{T}_1} \times \text{TRIPLES}_{\text{T}_2} \times \text{LITERALS}_{\text{L}_1} \times \text{LITERALS}_{\text{L}_1} \\
N_2 &:= N_1(\text{T}_1.\mathit{subject} = \text{T}_2.\mathit{subject}) \\
N_3 &:= N_2(\text{T}_1.\mathit{predicate} = \text{L}_1.\mathit{id} \wedge \text{T}_2.\mathit{predicate} = \text{L}_2.\mathit{id}) \\
N_4 &:= N_3(\text{L}_2.\mathit{value} = \text{“http://purl.org/dc/elements/1.1/title”}) \\
N_4 &:= N_3(\text{L}_2.\mathit{value} = \text{“http://purl.org/dc/elements/1.1/title”}) \\
N &:= N_4(\text{T}_1.\mathit{object} \rightarrow \mathit{title}, \text{T}_2.\mathit{object} \rightarrow \mathit{author})[\mathit{title}, \mathit{author}]
\end{aligned}
$$

Po převedení do SQL vypadá dotaz následujícím způsobem:

```
SELECT t1.object AS title, t3.object AS author
FROM       triples t1
INNER JOIN literals t2 ON t2.id = t1.predicate
INNER JOIN triples t3 ON t1.subject = t3.subject
INNER JOIN literals t4 ON t4.id = t3.predicate
WHERE t2.value = 'http://purl.org/dc/elements/1.1/title'
AND   t4.value = 'http://purl.org/dc/elements/1.1/creator'
```

V tomto případě jsme zvolili jiná jména pro tabulky, protože naše implementace popisovaného převodu je generuje tímto způsobem.

Analogicky budeme postupovat v případě dotazu obsahujícího více trojic.

Formálně budeme F stavět průchodem modifikovaného stromu dotazu zdola nahoru. V listech tohoto stromu jsou uzly Triples nebo potomci Constraint. Napřed se zaměříme pouze na první typ a uzly, které se nachází na cestě od nich ke kořeni.

Vezměme n uzel odpovídající základnímu grafovému dotazu $\hat{n}$. Pak definujeme $F(n)$ takto:

$$
\begin{aligned}
F(n) = \; & \mathit{SQL}((\text{TRIPLES}_{\text{T}_1} \times \ldots * \text{TRIPLES}_{\text{T}_k} \times \\
& \times \text{LITERALS}_{\text{L}_1} \times \ldots \times \text{LITERALS}_{\text{L}_l})(\varphi_1 \wedge \ldots \wedge \varphi_m) \\
& (\mathit{varcol}_1 -> \mathit{var}_1, \ldots, \mathit{varcol}_v -> \mathit{var}_v)[\mathit{var}_1, \ldots, \mathit{var}_v])
\end{aligned}
$$

kde k je počet trojic dotazu $\hat{n}$, l počet literálů a anonymních uzlů dotazu $\hat{n}$ a v počet proměnných $\hat{n}$.

Podmínky φ_i jsou prvky množiny $Cond$, pro jejíž tranzitivní uzávěr $Cond^*$ platí, že $(T_i.coll_i = T_j.colr_j) \in Cond^*$ právě tehdy, když trojice odpovídající T_i obsahuje na místě $coll_i$ stejnou proměnnou jako trojice odpovídající T_j na místě $colr_j$. Přičemž $coll_i$ a $colr_j$ jsou `subject`, `predicate` nebo `object`.

Pro $varcol_i$ platí, že jde o název jednoho ze sloupců, který odpovídá proměnné var_i z dotazu $\hat{n}$. Takových sloupců může být více, ale vlastnosti $Cond$ zaručují, že nezáleží na výběru reprezentanta.

Funkce SQL převádí výraz relační algebry do SQL.

5.4 Kombinování poddotazů

V této části se zaměříme na uzly typu GroupOrUnionGraphPattern a OptionalGraphPattern. V nich dochází ke kombinaci dvou SQL dotazů do jednoho většího.

Označme m takový uzel. Uzel m má dva potomky m_1 a m_2, jejichž pořadí je určeno pořadím v zápisu dotazu. Platí $F(m) = \varphi(F(m_1), F(m_2))$, kde φ je funkce odpovídající typu uzlu.

5.4.1 Kombinace pomocí OPTIONAL

Jako příklad použijeme modifikaci předchozího příkladu.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  ?book dc:title ?title .
  OPTIONAL { ?book dc:creator ?author }
}
```

Při překladu přeložíme volitelný poddotaz do SQL tak, jako by se jednalo o samostatný dotaz, a výsledek připojíme pomocí levého vnějšího spojení k SQL vzniklému převedením zbytku dotazu.

```
SELECT opt1.author AS author,
COALESCE (t1.subject, opt1.book) AS book,
t1.object AS title
FROM triples t1
INNER JOIN literals t2 ON t2.ID = t1.predicate
```

```
LEFT JOIN
(SELECT t3.object AS author, t3.subject AS v_book
 FROM triples t3
 INNER JOIN literals t4 ON t4.id = t3.predicate
 WHERE t4.value = 'http://purl.org/dc/elements/1.1/creator'
) opt1 ON (opt1.book = t1.subject OR t1.subject IS NULL)
WHERE t2.VALUE = 'http://purl.org/dc/elements/1.1/title'
```

Funkce $\varphi(F(m_1), F(m_2))$ vypadá v případě poddotazu OptionalGraphPattern tak, že k tabulkám z $F(m_1)$ je připojeno pomocí levého vnějšího spojení celé $F(m_2)$ a toto spojení je omezeno na rovnost proměnných obdobně jako v případě spojování trojic v základním bloku.

Z takto vytvořeného dotazu jsou projekcí vybrány sloupce, které odpovídají proměnným z $var(m_1) \cup var(m_2)$.

Výsledek dotazu je však potřeba ještě připravit na případ, že $F(m_1)$ ponechá některou z proměnných neohodnocenou, zatímco $F(m_2)$ již ne. Tedy je potřeba upravit výsledek dotazu, aby vracel ohodnocení z $F(m_1)$, pokud existuje, jinak musí vrátit ohodnocení z $F(m_2)$.

5.4.2 Kombinace pomocí UNION

Opět modifikujeme předchozí příklad.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
{
  { ?book dc:title ?title }
  UNION
  { ?book dc:creator ?author }
}
```

Tento dotaz přeložíme i do SQL jako sjednocení. Protože oba poddotazy obsahují různé proměnné, je potřeba přidat do SQL, která z nich vzniknou, sloupce s hodnotou NULL.

```
SELECT NULL AS author, t1.subject AS book, t1.OBJECT AS title
  FROM triples t1
 INNER JOIN literals t2 ON t2.ID = t1.predicate
```

```
 WHERE t2.value = 'http://purl.org/dc/elements/1.1/title'
UNION
SELECT t3.OBJECT AS author, t3.subject AS book, NULL AS title
  FROM triples t3
 INNER JOIN literals t4 ON t4.ID = t3.predicate
 WHERE t4.value = 'http://purl.org/dc/elements/1.1/creator'
```

Funkce $\varphi(F(m_1), F(m_2))$ pro sjednocení vypadá tak, že dotazy $F(m_1)$ a $F(m_2)$ napřed rozšíří tak, aby jejich výsledkem byly sloupce odpovídající $var(m_1) \cup var(m_2)$ a to ve stejném pořadí. Označíme-li tato rozšíření jako $F'(m_1)$ a $F'(m_2)$, pak platí $\varphi(F(m_1), F(m_2)) = F'(m_1)$ UNION $F'(m_2)$.

Pro spojení SQL poddotazů je použito pouze UNION. Při použití UNION ALL by se do výsledku dostalo dvakrát ohodnocení proměnných, které vyhovuje oběma větvím.

Je třeba si uvědomit, že tato eliminace duplicit se zásadně liší od eliminace duplicit, kterou provádí SPARQL dotaz začínající SELECT DISTINCT. Pomocí UNION zajistíme, že výsledkem dotazu je skutečně množina (přípustných ohodnocení). SELECT DISTINCT eliminuje duplicity, které vzniknou projekcí všech proměnných použitých v dotazu jen na vybrané proměnné.

5.5 Filtrování výsledků poddotazu

Podmínky FILTER není možné zcela přímočaře implementovat jejich přidáním do SQL dotazu kvůli způsobu, jakým je v nich nakládáno s typovým systémem RDF. Je nutné vytvořit v SQL složitější funkce, které řeší například porovnání dvou RDF literálů operátorem $<$. To se negativně projeví na schopnosti SQL stroje optimalizovat vyhodnocení výsledného SQL dotazu.

Pro implementaci jsme v databázi definovali strukturovaný datový typ RDF_LITERAL a funkce, které s ním pracují. Funkce odpovídají SPARQL funkcím a operátorům. Příklad několika funkcí uvádí následující tabulka:

jméno funkce	**popis**
rdf_integer	vytvoří RDF_LITERAL z číselné konstanty
rdf_var	vytvoří RDF_LITERAL z identifikátoru literálu přistupuje do tabulky `LITERALS`
rdf_plus	sečte dvě hodnoty
rdf_eq	porovná dvě hodnoty na rovnost
rdf_not	logická negace
rdf_bound	implementace SPARQL funkce bound

Na volání těchto funkcí je třeba podmínku přeložit, jak ukáže následující příklad.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title
WHERE
{
  ?book dc:title ?title .
  ?book ns:price ?price .
  FILTER ( ?price < 100 )
}
```

Do SQL je dotaz přeložen takto:

```
SELECT t1.subject AS book,
       t3.object AS price,
       t1.object AS title
FROM triples t1
INNER JOIN literals t2 ON t2.ID = t1.predicate
INNER JOIN triples t3 ON t1.subject = t3.subject
INNER JOIN literals t4 ON t4.ID = t3.predicate
WHERE t2.value = 'http://purl.org/dc/elements/1.1/title'
AND t4.value = 'http://example.org/ns#price'
AND rdf_to_bool (rdf_literal_from_id (t3.object)) = 1
```

Překlad filtrující podmínky do SQL probíhá pomocí funkce $C(F(m), c)$, kde $\hat{m}$ je poddotaz, který podmínka omezuje, a c kořen podstromu reprezentující podmínku.

Pokud je jako parametr použita proměnná, pak je na odpovídající místo v SQL vloženo volání funkce rdf_var, která převádí identifikátor literálu na hodnotu typu RDF_LITERAL. Parametrem volání je sloupec $F(m)$ odpovídající použité proměnné.

Pokud je c konstanta typu integer a $val(c)$ hodnota této konstanty, pak platí $C(F(m), c) = \text{rdf\_integer}(val(c))$.

Jako poslední příklad vezměme podmínku $?x+1 = 5$. Předpokládejme, že proměnné x odpovídá sloupec jménem v_x. Dále definujme $l(c)$ jako prvního potomka uzlu c a $r(c)$ jako druhého potomka. Pak platí

$$C(F(m), +) = \text{rdf\_plus}(C(F(m), l(+)), C(F(m), r(+)))$$

kde + značí uzel odpovídající + v příkladu. Obdobně označíme uzel reprezentující = jako =.

Dosazením za $l(+)$ a $r(l)$ získáme

$$C(F(m), +) = \text{rdf\_plus}(\text{rdf\_var}(v\_x), \text{rdf\_integer}(1)).$$

Obdobně použitím vztahu pro překlad porovnání na rovnost

$$C(F(n), =) = \text{rdf\_eq}(C(F(m), l(=)), C(F(m), r(=)))$$

získáme výsledný výraz

$$C(F(n), =) = \text{rdf\_eq}(\text{rdf\_plus}(\text{rdf\_var}(v\_x), \text{rdf\_integer}(1)), \text{rdf\_integer}(5)).$$

Protože návratová hodnota funkcí realizující vestavěné funkce i relační operátory je RDF_LITERAL, je nutné ještě výsledek převést na porovnání platné v SQL. Proto je výsledek převodu definován jako

$$F(n) = \text{rdf\_to\_bool}(C(F(m), c))$$

Analogicky je možné přeložit libovolnou podmínku, například u negace selháním z 3.7.5 lze vnitřní filtrující podmínku !bound(?x) přepsat jako

$$C(F(n), !bound(?x)) = \text{rdf\_not}(\text{rdf\_bound}(\text{rdf\_val}(v\_x)))$$

6 Implementace

Aby bylo možné navržené postupy odzkoušet nad daty, vytvořili jsme prototypovou aplikaci, která převod dotazů provádí. Pro implementaci jsme zvolili jazyk C# a prostředí .NET.

Pro syntaktickou a sémantickou analýzu SPARQL dotazů využíváme generátor kompilátorů Coco/R [9]. Coco/R využívá pro sémantickou analýzu rekurzivní sestup, je tedy schopné generovat překladače pro LL(1) jazyky. Standard SPARQL uvádí, že gramatika jazyka je LL(1), ale po přepsání gramatiky do formátu přijímaného Coco/R se ukázalo, že obsahuje LL(1) konflikty. Ty se však podařilo snadno vyřešit prostředky poskytovanými Coco/R dočasným zvětšením výhledu na 2.

Větší problém se ukázal v lexikální analýze. Analyzátor generovaný Coco/R pracuje pouze s výhledem na jeden znak dopředu, a proto není schopen rozlišit operátor < od zápisu URI, což je adresa uzavřená mezi < a >. Protože se jedná o experimentální implementaci, vyřešili jsme problém nahrazením operátoru < operátorem << a analogicky operátoru <= operátorem <<=.

Naším cílem bylo pouze prozkoumat chování systému pomocí měření a nikoliv implementovat systém pro nasazení do provozu, proto jsme vytvořili pouze jednoduchou aplikaci, která převádí SPARQL dotazy do SQL. Samotné spuštění dotazu a zpracování výsledku aplikace neprovádí.

Výsledný dotaz vrací tři sloupce pro každou z proměnných, která má být součástí výsledku dotazu. Sloupce obsahují hodnotu, typ a jazykovou značku literálu.

Testovací prostředí a aplikace, jejichž pomocí jsme měření prováděli popisuje kapitola 8. Než se však dostaneme k samotným testům, představíme v kapitole 7 testovací data.

Implementace nepokrývá celý standard SPARQL a místy se od něj odchyluje.

- Chybí podpora pro práci s více grafy. Možnost určit název grafu pomocí proměnné vázané i v jiné části dotazu by zkomplikovala reprezentaci dat. Navíc přínos takto obecně pojaté práce s více grafy je nejasný.

- Při použití bloku OPTIONAL nastane problém v okamžiku, kdy se na jednom místě vyskytne více OPTIONAL bloků, které sdílí některé proměnné. Současný návrh standardu sice v textu připouští vícenásobné OPTIONAL bloky, ale formální definice existuje pouze pro případ,

kdy je tento blok jen jeden. V jednom ze starších návrhů [10] byl tento případ samostatně ošetřen, ale příslušný odstavec byl v pozdějších verzích odstraněn. Omezení spočívalo v tom, že pokud se proměnná vyskytuje ve více OPTIONAL blocích, pak do ní musí být hodnota přiřazena v bloku mimo optional. Naše implementace vyžaduje toto omezení.

- Naše implementace umí vyhodnotit pouze typ dotazu SELECT. Nepodporuje CONSTRUCT, DESCRIBE a ASK.
- Chybí podpora pro třídění výsledku a pro omezení výsledku jen na jeho část pomocí OFFSET a LIMIT.

7 Data

V této části představíme data, která byla použita při testování implementace jazyka SPARQL. Podrobný popis postupu, kterým byla data získána obsahuje [8], zde se zaměříme pouze na jejich obsah. V relační databázi by šlo o databázové schéma, v terminologii RDF se hovoří o třídách a vlastnostech těchto tříd. Všechny třídy a vlastnosti patří do jmenného prostoru http://www.is.cuni.cz/stoh/metadata/, který však u nich pro zpřehlednění zápisu v této kapitole neuvádíme nebo jej zkracujeme do jmenného prostoru `stoh`.

Struktura testovacích dat je relativně složitá, ale protože pochází ze systémů, které zpracovávají informace o lidech, jsou v jejím centru právě informace o osobách, tedy jméno, příjmení, datum narození atd. V RDF jde o třídu ot_osoba.

Zajímavé predikáty, které mají jako definiční obor třídu ot_osoba, ukazuje následující tabulka.

ot_osoba		
jmeno	řetězec	jméno
prijmeni	řetězec	příjmení
datum_narozeni	datum a čas	datum narození

7.1 Číselníky

Data obsahují i jednotně řešené číselníky, které jsou uloženy ve třídách cst_ciselnik a cht_ciselnik_plochy. Třída cst_ciselnik reprezentuje jeden celý číselník, zatímco jedna instance třídy cht_ciselnik_plochy obsahuje jednu položku jednoho číselníku.

cst_ciselnik		
nazev_tabulky	řetězec	jméno číselníku pro strojové zpracování
uzivatelsky_nazev	řetězec	popisek číselníku

cht_ciselnik_plochy		
kod	řetězec	kód pro strojové zpracování
hodnota	řetězec	popisek položky
id_ciselnik	cizí klíč	odkaz na číselník
poradi	číslo	pořadí položky v číselníku

Tyto číselníky (instance třídy cht_ciselnik_plochy) jsou používány jako hodnoty vlastností v instancích ostatních tříd. Přestože v systému, kterým byla data získána, není vyžadováno, aby jako hodnoty jedné vlastnosti byly používány pouze položky jednoho konkrétního číselníku, je toto omezení v datech dodržováno.

7.2 Identifikace

Pro uložení identifikačních údajů osob — jako jsou třeba rodné číslo nebo číslo pasu — byla založena třída ot_identifikace. Tato třída obsahuje cizí klíč na osobu (atribut id_osoba), druh identifikace (atribut druh, cizí klíč na cht_ciselnik_plochy) a samotnou hodnotu identifikace (atribut identifikace).

Tento způsob uložení identifikací byl zvolen proto, aby bylo možné přidat nový druh identifikace pouhou změnou číselníku druhů identifikací.

ot_identifikace		
druh	číselník	druh identifikace (RČ, číslo OP, ...)
identifikace	řetězec	hodnota identifikace
id_osoba	cizí klíč	odkaz na osobu vlastnící identifikaci

7.3 Kontakty

Instance třídy ot_kontakt slouží k uchování adres, telefonů, emailů a adres www stránek. Ve vlastnostech druh_kontaktu a poddruh_kontaktu je určeno, o který typ kontaktu se jedná. Hodnoty těchto vlastností jsou položky číselníku PER_KONTAKT, respektive PER_KONTAKT_PODDRUH. Samotné údaje o kontaktu jsou uloženy v dalších atributech této třídy, jako je třeba ulice, email, ap. Kontakty jsou vázány na osobu, organizaci nebo pracovní úvazek (adresa do zaměstnání).

ot_kontakt		
druh_kontaktu	číselník	druh kontaktu (adresa, telefon, ...)
poddruh_kontaktu	číselník	upřesnění druhu (trvalá, kontaktní adresa,pevná linka, mobilní telefon, ...)
id_osoba	cizí klíč	odkaz na osobu vlastnící kontakt
id_org	cizí klíč	odkaz na organizaci vlastnící kontakt
id_vztah	cizí klíč	odkaz na úvazek vlastnící kontakt
telefon	řetězec	telefonní číslo
email	řetězec	e-mailová adresa

7.4 Velikosti a četnosti

V této části se zaměříme na velikost dat a další souhrnné údaje.

7.4.1 Predikáty

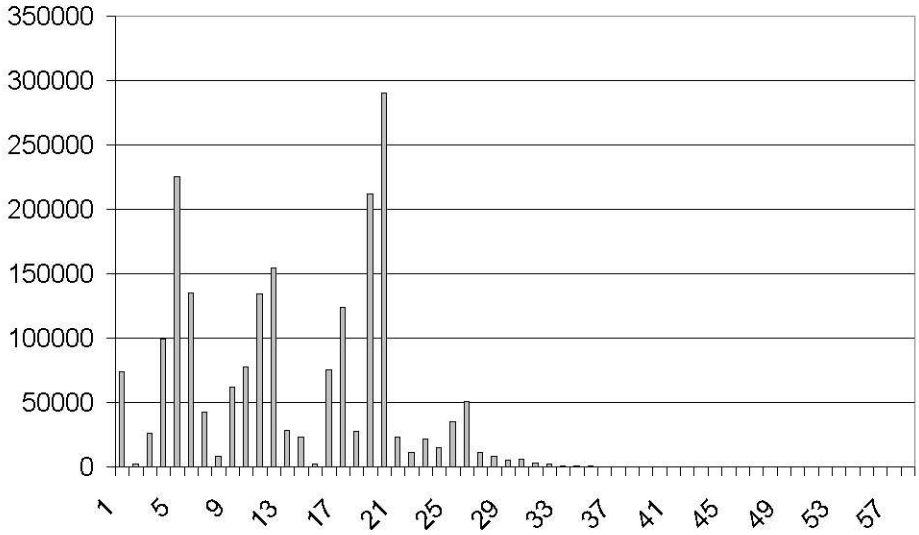
V následující tabulce jsou zobrazeny četnosti vybraných predikátů ve sloupce `predicates` tabulky `TRIPLES`.

predikát	**četnost**
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	2020198
http://www.w3.org/2000/01/rdf-schema#domain	1889
http://www.w3.org/2000/01/rdf-schema#range	1889
stoh:ot_osoba__datum_narozeni	89247
stoh:ot_osoba__jmeno	91166
stoh:ot_osoba__pohlavi	89571
stoh:ot_osoba__prijmeni	91528
stoh:cht_ciselnik_plochy__hodnota	43707
stoh:cht_ciselnik_plochy__id_ciselnik	43707
stoh:cht_ciselnik_plochy__kod	43707
stoh:cht_ciselnik_plochy__poradi	43706
stoh:cst_ciselnik__nazev_tabulky	117
stoh:cst_ciselnik__popis	115
stoh:ot_identifikace__druh	186888
stoh:ot_identifikace__identifikace	186888
stoh:ot_identifikace__id_osoba	186888
stoh:ot_kontakt__druh_kontaktu	189029
stoh:ot_kontakt__email	19883
stoh:ot_kontakt__id_osoba	188845

7.4.2 Subjekty

Četnosti jednotlivých subjektů v tabulce `TRIPLES` se pohybují od 1 do 58. Histogram těchto četností ukazuje graf 3. Na ose X jsou vyneseny četnosti subjektů v tabulce `TRIPLES` a na ose Y počet různých subjektů s danou četností.

Pro četnosti od 38 dále nepřesahuje počet subjektů s danou četností 20. Z obrázku je vidět, že typická četnost subjektu je asi 5 až 20. Reálná data



Graf 3: Histogram četností hodnot sloupce subject

tedy potvrzují předpoklady, které jsme použili v části 4.3.2.

7.4.3 Objekty

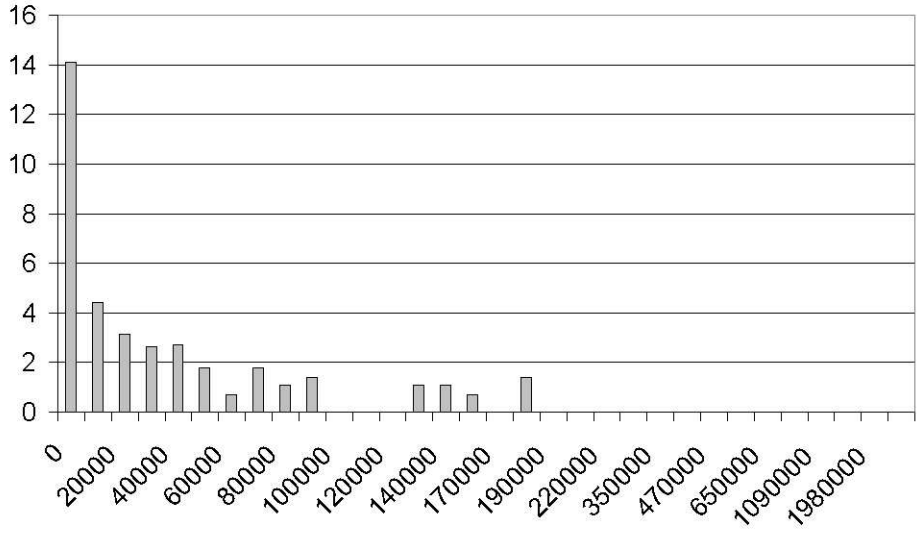
Četnosti hodnot v sloupci `object` tabulky `TRIPLES` se liší od rozložení hodnot ve sloupci `subject`.

Graf 4 ukazuje rozložení hodnot objektů v tabulce `TRIPLES`. Na ose X jsou vyneseny četnosti objektů v `TRIPLES` sloučené po desítkách tisíc a na ose Y je logaritmus počtu různých objektů, které mají četnost z daného rozsahu (hodnota na ose X až hodnota na ose X + 9999).

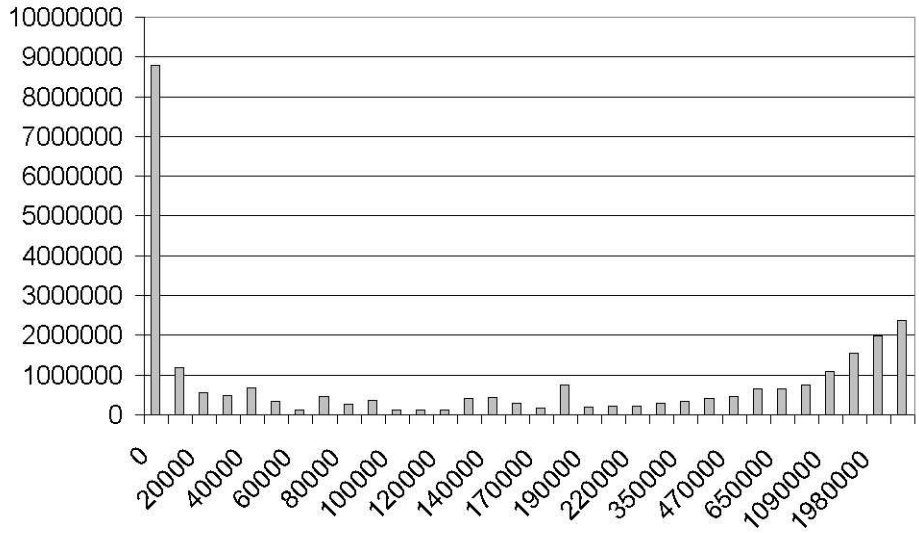
Rozdílná volba měřítka je vynucena tím, že největší četnost jedné hodnoty dosahuje 2.37 milionu, zatímco u subjektů bylo maximum 58.

Protože jsou četnosti velké, tvoří řádky obsahující objekty s velkou četností nezanedbatelnou část všech řádků tabulky `TRIPLES`. To je vidět v grafu 5, který se od předchozího liší tím, že na osu Y vynášíme počet všech řádků tabulky `TRIPLES`, jež obsahují hodnoty s četností odpovídající rozsahu na ose X (pokud by nebyly sloučeny četnosti po 10000, jednalo by se o četnost vynásobenou počtem různých objektů s touto četností).

Z grafů je vidět, že je platné varování z kapitoly 4, které říkalo, že selektivita selekce na sloupec `object` sice může, ale rozhodně nemusí, být velká.



Graf 4: Histogram četností sloupce object



Graf 5: Počty řádků podle četnosti sloupce object

8 Dotazy a měření

V této kapitole ukážeme dotazy, které je možné pokládat nad testovacími daty. Uvedeme jejich znění v jazyku SPARQL, výsledné rychlosti dosahované při různých reprezentacích dat a pokusíme se zanalyzovat faktory, které rychlost negativně ovlivňují.

Pro zjednodušení zápisu byl u dotazů vynechán seznam použitých prefixů. Před všechny dotazy je třeba pro jejich správné vyhodnocení přidat:

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix mt : <http://www.is.cuni.cz/stoh/metadata/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

8.1 Testovací prostředí

Databáze běžela na stroji se dvěma XEON 3.06GHz, 6GB RAM a SCSI diskovým polem se čtrnácti 144GB 10k RPM disky pro uložení dat a indexů. Pro RDF databázi bylo vyhrazeno 600MB RAM.

Jako aplikační server sloužil stroj se čtyřmi XEON 2.50GHz a 16GB RAM. I ten byl vybaven SCSI diskovým polem, ale jeho rychlost neměla na výsledky vliv, protože aplikace běžící na tomto stroji nebyly náročné na práci s diskem.

8.2 Měření dotazů

K dotazům uvádíme i tabulky s výsledky experimentálního měření rychlosti. Při měření jsme postupovali následujícím způsobem.

1. SPARQL dotaz převedeme do SQL.
2. SQL dotaz vložíme do měřicího systému. Ten automaticky provede kroky 3 až 8.
3. Proběhne restart databáze, aby se vyprázdnily cache.
4. Pauza 10 vteřin, aby databáze dokončila start.
5. Spuštění SQL dotazu.

6. Změření času do vrácení první řádky.

7. Změření celkového času dotazu a celkového počtu vrácených řádek.

8. Druhé měření bez restartu databáze (body 4 až 7).

Dále uvedené výsledky ukážou, že vyprázdnění cache v databázi a opakování měření má smysl, protože časy prvního a druhého pokusu se výrazně liší. Další iterace však již zrychlení nepřinášejí.

8.3 Dotazy s URI v predikátu a bez literálů

Napřed ukážeme několik dotazů, které používají v trojicích pouze proměnné a URI.

8.3.1 Seznam osob

Následující dotaz vrací seznam jmen a příjmení všech osob, které je mají.

```
select ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni }
```

U tohoto dotazu si ukážeme i výsledek převodu do SQL.

```
SELECT (SELECT value
          FROM literals
         WHERE ID = v_jmeno) AS v_jmeno,
       (SELECT type
          FROM literals
         WHERE ID = v_jmeno) AS tv_jmeno,
       (SELECT lang
          FROM literals
         WHERE ID = v_jmeno) AS lv_jmeno,
       (SELECT value
          FROM literals
         WHERE ID = v_prijmeni) AS v_prijmeni,
       (SELECT type
          FROM literals
```

```
       WHERE ID = v_prijmeni) AS tv_prijmeni,
     (SELECT lang
        FROM literals
       WHERE ID = v_prijmeni) AS lv_prijmeni
FROM (SELECT t1.OBJECT AS v_jmeno,
             t1.subject AS v_osoba,
             t3.OBJECT AS v_prijmeni
        FROM triples t1
             INNER JOIN literals t2 ON t2.ID = t1.predicate
             INNER JOIN triples t3 ON t1.subject = t3.subject
             INNER JOIN literals t4 ON t4.ID = t3.predicate
       WHERE t2.value =
'http://www.is.cuni.cz/stoh/metadata/ot_osoba__jmeno'
         AND t4.value =
'http://www.is.cuni.cz/stoh/metadata/ot_osoba__prijmeni')
```

Problémem tohoto dotazu je rychlost. Zatímco v relační databázi by postačoval sekvenční průchod tabulkou osob, v naší databázi je nutné několik spojení a selekcí.

Optimalizátor v Oracle 10g zvolil (v případě základní reprezentace) přístup, kdy napřed pomocí indexu najde záznamy v tabulce literálů, které obsahují mt:osoba__prijmeni, a pomocí hašovaného spojení je spojí s tabulkou trojic. Tak získá seznam osob a jejich příjmení. Dalším hašovaným spojením tento seznam znovu spojí s tabulkou trojic a získá navíc všechna tvrzení o osobách. Posledním hašovaným spojením na tabulku literálů, která je napřed indexem omezena na záznamy s mt:osoba__jmeno, získá i jména osob.

Nakonec ještě vyhodnotí vnořené dotazy ze začátku dotazu, které převádí identifikátory literálů na jejich hodnotu, typ a jazykovou značku. Převod je vyhodnocen přístupem do tabulky literálů přes primární klíč.

Následující tabulka ukazuje čas potřebý k vyhodnocení dotazu, čas potřebný na vrácení první řádky výsledku a oba údaje při opakovaném spuštění dotazu. Údaj o opakovaném spuštění uvádíme proto, aby se ukázal vliv vyrovnávacích pamětí v databázi.

V řádcích uvádíme časy pro jednotlivé varianty, které se liší uložením tabulky trojic.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	20s	2920ms	11s	914ms
B-strom	14s	878ms	13s	292ms
paralelní	13s	369ms	12s	315ms
bitmapa	19s	4634ms	11s	902ms
	Počet řádků: 91166			

V tomto případě je rychlost běžného a bitmapového indexu na stejné úrovni, stejně tak jako neparalelní i paralelní varianta tabulky trojic organizované jako index. Indexová organizace se v tomto případě ukazuje jako výrazně lepší při prvním spuštění. Ovšem se zapojením cache se výsledek otočí, ale ne tak výrazně.

8.3.2 Seznam číselníků a jejich hodnot

Významnou součástí dat jsou číselníky. V této a následující sekci uvedeme mimo jiné i několik příkladů pro práci se samotnými číselníky, které využijeme jako součást dalších dotazů.

Následující dotaz vypíše seznam číselníků, jejich názvů, hodnot a kódů hodnot.

```
select ?ciselnik ?nazev ?hodnota ?kod
where { ?ciselnik mt:cst_ciselnik__nazev_tabulky ?nazev .
  ?hodnota mt:cht_ciselnik_plochy__id_ciselnik ?ciselnik .
  ?hodnota mt:cht_ciselnik_plochy__kod ?kod }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	21s	1219ms	9541ms	569ms
B-strom	15s	325ms	12s	183ms
paralelní	13s	284ms	10s	207ms
bitmapa	14s	1065ms	9451ms	551ms
	Počet řádků: 43707			

Vyhodnocení i rychlost jsou obdobné jako v předchozím případě, ale ukazuje se, že při větším počtu trojic se zadaným predikátem dává bitmapový index lepší výsledky než běžný index.

8.4 Dotazy s literály

V této části se budeme zabývat dotazy, které obsahují literál na místě objektu. Chování tohoto typu dotazů se výrazně liší od dotazů uvedených v předchozí části.

8.4.1 Jméno osoby se zadaným příjmením

Příkladem je dotaz, který má zjistit jméno osoby, u níž známe příjmení a datum narození. Přesněji řečeno, dotaz zjistí všechna jména všech osob, které mají zadané příjmení a datum narození.

```
select ?jmeno
where { ?osoba mt:ot_osoba__prijmeni 'Smith' .
  ?osoba mt:ot_osoba__datum_narozeni '1950-01-01T00:00:00' .
  ?osoba mt:ot_osoba__jmeno ?jmeno }
```

Vyhodnocení dotazu je rychlé. Důvodem je, že nedochází k žádným velkým spojením. V databázi existuje malé množství trojic, které odpovídají první a druhé trojici dotazu. Díky velké kardinalitě sloupce TRIPLES.object se databáze rozhodne při vyhodnocení dotazu začít právě touto selekcí, která je díky indexu nad TRIPLES.object vyhodnocena rychle. Dále přes rovnost hodnoty v subjektu připojí ostatní trojice, což jde opět rychle a to díky indexu nad TRIPLES.subject a malému počtu trojic se stejným subjektem.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	241ms	241ms	37ms	37ms
B-strom	103ms	103ms	9ms	9ms
paralelní	111ms	111ms	8ms	8ms
bitmapa	100ms	100ms	9ms	9ms
	Počet řádků: 1			

8.4.2 Jedna hodnota jednoho číselníku

Nahrazením názvu a kódu v dotaze 8.3.2 konkrétními hodnotami získáme v proměnné ?hodnota anonymní uzel, který reprezentuje námi zvolenou hodnotu číselníku.

```
select ?hodnota
where {
  ?ciselnik mt:cst_ciselnik__nazev_tabulky "PER_KONTAKT" .
  ?hodnota mt:cht_ciselnik_plochy__id_ciselnik ?ciselnik .
  ?hodnota mt:cht_ciselnik_plochy__kod "3" }
```

Místo kódu lze použít i textovou hodnotu.

```
select ?hodnota
where {
  ?ciselnik mt:cst_ciselnik__nazev_tabulky "PER_KONTAKT" .
  ?hodnota mt:cht_ciselnik_plochy__id_ciselnik ?ciselnik .
  ?hodnota mt:cht_ciselnik_plochy__hodnota "trvalé bydliště"
}
```

Tento typ dotazu bude velmi častý, protože u několika běžně používaných tříd je přítomná vlastnost druh, která určuje, o jaký podtyp dané třídy se ve skutečnosti jedná (tím jsou simulovány podtřídy). Například místo rozdělení kontaktů na podtřídy adresa, telefon, email ap., jsou atributy všech podtříd sloučeny do ot_kontakt a typ kontaktu se pozná z atributu druh_kontaktu, což je právě číselníkovaný atribut.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	2921ms	2921ms	1090ms	1090ms
B-strom	772ms	772ms	115ms	115ms
paralelní	7021ms	7021ms	113ms	113ms
bitmapa	2836ms	2836ms	863ms	863ms
	Počet řádků: 1			

Ukázalo se, že paralelní přístup k tabulce trojic organizované jako index u některých dotazů vykazuje neočekávané zhoršení rychlosti.

8.5 Dotazy na metadata

V RDF jsou data i metadata uložena stejným způsobem a na stejném místě. Tedy je možné se nad metadaty dotazovat stejně jako nad daty, případně pokládat kombinované dotazy nad daty a metadaty zároveň.

Typově jde o dotazy, které jsme již ukázali v předchozích dvou sekcích.

8.5.1 Dotazy na třídy a vlastnosti

Tento typ dotazů se vyznačuje tím, že pracuje pouze s daty tvořícími pouhý zlomek velikosti databáze.

Následující dotaz vrací seznam všech tříd v databázi.

```
select ?trida
where { ?trida a rdfs:Class }
```

V dotaze je použito klíčové slovo a, což je zkratka za rdf:type.

Vyhodnocení v databázovém stroji vhodně využívá toho, že predikáty mají malou kardinalitu a že je lepší v případě, kdy je znám predikát i objekt, vybírat trojice pomocí indexu nad objekty. Díky tomu je dotaz vyhodnocen během několika milisekund.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	91ms	90ms	20ms	19ms
B-strom	63ms	63ms	19ms	19ms
paralelní	38ms	37ms	21ms	20ms
bitmapa	73ms	72ms	21ms	20ms
	Počet řádků: 226			

Dotaz na seznam všech vlastností vypadá a chová se téměř identicky.

```
select ?predicate
where { ?predicate a rdf:Property }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	176ms	132ms	88ms	56ms
B-strom	152ms	110ms	87ms	53ms
paralelní	111ms	77ms	82ms	50ms
bitmapa	148ms	113ms	91ms	58ms
	Počet řádků: 1889			

Seznam tříd a jejich vlastností získáme takto:

```
select ?trida ?vlastnost
where { ?trida a rdfs:Class .
  OPTIONAL { ?vlastnost rdfs:domain ?trida } }
```

Použití OPTIONAL zajistí, že se do seznamu dostanou i třídy bez vlastností.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	3992ms	2782ms	1409ms	938ms
B-strom	199ms	142ms	164ms	117ms
paralelní	194ms	147ms	150ms	105ms
bitmapa	3870ms	2600ms	1601ms	1073ms
	Počet řádků: 1890			

Indexová organizace tabulky `TRIPLES` se ukazuje jako řádově lepší. Rozdíl mezi paralelní a sériovou verzí je však minimální.

8.5.2 Dotazy kombinující data a metadata

Tímto typem dotazu můžeme například zjišťovat, instance kterých tříd odkazují na jednu konkrétní instanci.

```
select distinct ?trida
where {
  ?osoba mt:ot_osoba__datum_narozeni "1990-05-11T00:00:00" .
  ?x ?atribut ?osoba . ?x a ?trida }
```

Omezení na jednu instanci ot_osoba je řešeno přes omezení data narození, protože osoby jsou anonymní uzly. To by nebyla v návrhu reálné databáze dobrá volba, ale pro testovací účely jsme zvolili tuto variantu.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	428ms	428ms	17ms	16ms
B-strom	233ms	233ms	12ms	12ms
paralelní	493ms	493ms	12ms	12ms
bitmapa	305ms	305ms	18ms	18ms
	Počet řádků: 6			

8.6 Dotazy využívající číselníky

V této části se budeme zabývat dotazy, které jako svoji součást budou obsahovat dotaz na číselník. To přispívá k nárůstu složitosti dotazů, ale pouze v počtu trojic tvořících dotaz. Typově jde pořád o stejné dotazy jako doposud.

8.6.1 Dotaz na jeden údaj o osobě na základě znalosti jiného

Následujícím dotazem zjistíme email osoby, u níž známe rodné číslo.

```
select ?email
where { ?ident mt:ot_identifikace__identifikace "550130123" .
  ?ident mt:ot_identifikace__druh ?rc .
  ?rc mt:cht_ciselnik_plochy__hodnota "rodné číslo" .
  ?ident mt:ot_identifikace__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__druh_kontaktu ?druh .
  ?druh mt:cht_ciselnik_plochy__hodnota "Email" .
  ?kontakt mt:ot_kontakt__email ?email }
```

Zásadní význam na vyhodnocení dotazu má trojice dotazu `?ident mt:ot_identifikace__identifikace "550130123"`, protože obsahuje jednoznačné omezení hodnoty objektu. Tato část je vyhodnocena jako první a postupně jsou připojovány další trojice, které jsou na ni vázány přes proměnnou. Díky tomu nikde nedochází k spojování velkých tabulek a vyhodnocení dotazu je rychlé.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	1063ms	1063ms	9ms	9ms
B-strom	527ms	527ms	12ms	12ms
paralelní	551ms	551ms	9ms	9ms
bitmapa	557ms	557ms	8ms	8ms
	Počet řádků: 1			

Podobně se bude chovat i dotaz, který vrací více než jednu hodnotu, pokud bude na začátku také stát omezení na konkrétní hodnotu. Následující dotaz vrací seznam druhů kontaktů dostupných k osobě se zadaným rodným číslem.

```
select distinct ?druh
where { ?ident mt:ot_identifikace__identifikace "550130123" .
  ?ident mt:ot_identifikace__druh ?rc .
  ?rc mt:cht_ciselnik_plochy__hodnota "rodné číslo" .
  ?ident mt:ot_identifikace__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__id_osoba ?osoba.
```

```
?kontakt mt:ot_kontakt__druh_kontaktu ?edruh .
?edruh mt:cht_ciselnik_plochy__hodnota ?druh }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	470ms	470ms	10ms	10ms
B-strom	162ms	162ms	11ms	11ms
paralelní	151ms	150ms	10ms	10ms
bitmapa	439ms	439ms	9ms	9ms
	Počet řádků: 3			

V obou případech se potvrdila převaha indexové organizace a drobný nárůst rychlosti bitmapového indexu oproti B-stromu.

8.6.2 Seznam osobních údajů

Následující dotaz vypíše seznam jmen a příjmení lidí a jejich rodných čísel.

```
select ?jmeno ?prijmeni ?rc
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni .
  ?ident mt:ot_identifikace__druh ?drc .
  ?drc mt:cht_ciselnik_plochy__hodnota "rodné číslo" .
  ?ident mt:ot_identifikace__id_osoba ?osoba .
  ?ident mt:ot_identifikace__identifikace ?rc }
```

Podobně jako u dotazů na jednu hodnotu v tomto případě začne vyhodnocení na místě, kde je jasně omezen objekt. Tedy se najde hodnota číselníku pro rodné číslo. Pak se připojí na trojici identifikace, čímž vznikne velká kolekce dat. A tu je nutné spojit na další velká data.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	106s	24s	70s	8352ms
B-strom	50s	19s	35s	5600ms
paralelní	74s	44s	36s	5601ms
bitmapa	77s	10s	71s	9096ms
	Počet řádků: 86883			

Paralelizace přístupu k indexově organizované tabulce `TRIPLES` se opět ukázala problematická, ale sériová verze potvrdila nejlepší dosahované rychlosti z předchozích dotazů. Celková rychlost je však neuspokojivá.

8.7 Dotaz obsahující nepovinný podgraf

V této části provedeme měření dotazů obsahujících blok OPTIONAL.

8.7.1 Seznam osob s nepovinným jménem

S využitím OPTIONAL bloku se můžeme zeptat na příjmení a — pokud existuje — i jméno.

```
select ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__prijmeni ?prijmeni .
  OPTIONAL { ?osoba mt:ot_osoba__jmeno ?jmeno } }
```

Rychlost dotazu ukazuje tabulka.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	22s	865ms	10s	113ms
B-strom	7982ms	298ms	6476ms	108ms
paralelní	24s	3211ms	6399ms	90ms
bitmapa	15s	410ms	10s	111ms
	Počet řádků: 91528			

I tento výsledek potvrzuje závěry z předchozích dotazů, tj. že indexová organizace je nejvýhodnější, ale s její paralelní verzí mohou být problémy.

8.7.2 Vnořené nepovinné bloky

```
select ?jmeno ?prijmeni ?mesto ?psc
where
{
  ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni .
  OPTIONAL {
    ?kontakt mt:ot_kontakt__id_osoba ?osoba .
    ?kontakt mt:ot_kontakt__druh_kontaktu ?druh .
    ?druh mt:cht_ciselnik_plochy__hodnota "Adresa" .
    ?kontakt mt:ot_kontakt__mesto ?mesto .
    OPTIONAL {
      ?kontakt mt:ot_kontakt__pcs ?psc
```

```
    }
  }
}
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	201s	6670ms	171s	3834ms
B-strom	166s	5612ms	133s	1083ms
paralelní	153s	7536ms	125s	1301ms
bitmapa	199s	7222ms	168s	2088ms
	Počet řádků: 106939			

Výsledky všech verzí jsou neuspokojivé, ale ukázalo se, že paralelizace někdy přináší i nezanedbatelné zlepšení rychlosti.

8.8 Dotazy obsahující číselné literály

V této části se budeme zabývat dotazy, které obsahují číselné konstanty. Ty je potřeba vyhodnotit s použitím tabulky `TYPED(decimal)`.

8.8.1 Seznam osob a titulů

Napřed srovnáme dvě verze dotazu, který vrací seznam osob a ke každé osobě uvádí její první titul. Pokud takový titul nemá, není osoba v seznamu zobrazena.

Při zápisu tak, jak jsme jej používali doposud, by dotaz vypadal takto:

```
select ?jmeno ?prijmeni ?titulvalue
where {
  ?titul mt:ot_osoba_titul__poradi "1" .
  ?titul mt:ot_osoba_titul__id_osoba ?osoba .
  ?titul mt:ot_osoba_titul__id_titul ?ctitul .
  ?ctitul mt:cht_ciselnik_plochy__hodnota ?titulvalue .
  ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni
}
```

Pokud však využijeme možností jazyka SPARQL pro zápis číselných konstant, můžeme jej zapsat i následujícím způsobem:

```
select ?jmeno ?prijmeni ?titulvalue
where {
  ?titul mt:ot_osoba_titul__poradi 1 .
  ?titul mt:ot_osoba_titul__id_osoba ?osoba .
  ?titul mt:ot_osoba_titul__id_titul ?ctitul .
  ?ctitul mt:cht_ciselnik_plochy__hodnota ?titulvalue .
  ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni
}
```

Rozdíl mezi dotazy je nejen v zápisu. V prvním dotaze se ptáme na ty tituly, jejichž pořadí je textově zapsáno jako "1". Ve druhém případě na ty tituly, jejichž pořadí je číselně 1. Avšak kvůli způsobu, jakým testovací data vznikla, jde v našem případě o ty samé záznamy.

Rozdíl mezi dotazy po převedení do SQL je v tom, že u druhého dotazu je navíc připojena tabulka `TYPED(decimal)`, která je omezena tak, že vrací pouze jeden řádek. Výsledek je spojen s tabulkou `LITERALS`, přičemž tomuto omezení vyhovuje právě jeden řádek, a to ten samý řádek, který vyhovuje podmínce na textovou hodnotu "1" z prvního dotazu. Zbytek dotazu je stejný.

Rychlost prvního dotazu ukazuje tato tabulka:

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	29s	21s	15s	9807ms
B-strom	11s	8765ms	4628ms	2246ms
paralelní	72s	66s	4573ms	2235ms
bitmapa	28s	18s	38s	11s
	Počet řádků: 5849			

Rychlost druhého dotazu vypadá následovně:

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	45s	14s	41s	10s
B-strom	5622ms	1983ms	4744ms	1520ms
paralelní	6285ms	2344ms	4716ms	1530ms
bitmapa	43s	12s	41s	11s
	Počet řádků: 5849			

Opět se potvrdila převaha indexové organizace u `TRIPLES`. Navíc první dotaz ukazuje další velkou odchylku v rychlosti paralelní verze.

8.9 Dotaz na objekt s velkou četností

Dotaz na první titul osoby je zajímavý tím, že obsahuje literál "1", což je literál s druhou největší četností výskytů ve sloupci `object` tabulky `TRIPLES`. Přesto je dotaz vyhodnocen rychle.

Zkusíme srovnat vliv selektivity literálu na rychlost dotazu v případě jednoduššího dotazu.

```
select ?s ?p
where { ?s ?p "1" }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	158s	210ms	153s	138ms
B-strom	123s	147ms	122s	84ms
paralelní	124s	144ms	122s	91ms
bitmapa	145s	150ms	143s	96ms
	Počet řádků: 3531814			

```
select ?s ?p
where { ?s ?p 1 }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	101s	274ms	83s	134ms
B-strom	87s	159ms	85s	196ms
paralelní	87s	177ms	84s	181ms
bitmapa	81s	234ms	80s	160ms
	Počet řádků: 1987905			

Rozdíl v počtu trojic odpovídajících prvnímu a druhému dotazu je způsoben tím, že existují literály s hodnotou "1" typu string a decimal. Prvnímu dotazu odpovídají oba, druhému pouze číselné.

```
select ?s ?p
where { ?s ?p "-1" }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	796ms	793ms	57ms	55ms
B-strom	159ms	157ms	60ms	58ms
paralelní	104ms	103ms	65ms	63ms
bitmapa	199ms	197ms	56ms	53ms
	Počet řádků: 265			

Jak je vidět, platí očekávaná úměra mezi počtem vyhovujících řádek a časem potřebným na zpracování dotazu.

Co je však zajímavé, je fakt, že celý dotaz 8.8.1, který obsahoval speciální případ trojice tvořící druhý dotaz v této části, seběhnul rychleji. Podle plánů vyhodnocení dotazů to vypadá, že zásadní rozdíl nespočívá v tom, že v případě jednoduššího dotazu je na místě predikátu použita proměnná. Problém pravděpodobně spočívá v převodu identifikátorů literálů na textové hodnoty. Počet těchto převodů se liší o několik řádů.

8.10 Dotazy obsahující sjednocení

Sjednocení ve SPARQL vyhodnocujeme pomocí sjednocení i v SQL. Tato část by proto neměla přinést nějaká zásadní překvapení.

8.10.1 Příjmení a rodná příjmení

Následující dotaz vrací v jednom sloupci příjmení a rodná příjmení osob.

```
select ?prijmeni
where {
  { ?osoba mt:ot_osoba__prijmeni ?prijmeni }
  UNION
  { ?osoba mt:ot_osoba__rodne_prijmeni ?prijmeni }
}
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	4262ms	961ms	3425ms	476ms
B-strom	3704ms	541ms	3229ms	273ms
paralelní	3544ms	331ms	3245ms	271ms
bitmapa	4009ms	839ms	3332ms	430ms
	Počet řádků: 95932			

Rychlost dokonce přesahuje rychlost dotazu 8.3.1 (seznam osob), který vrací výsledek menší o počet osob se zadaným rodným příjmením. Původ tohoto rozdílu spočívá v tom, že tento dotaz neobsahuje křestní jména, čímž ušetří čas strávený v případě dotazu 8.3.1 převodem identifikátorů jmen na textové hodnoty.

8.11 Dotazy s filtrující podmínkou

V této části vyzkoušíme několik SPARQL dotazů obsahujících klauzuli FILTER. Protože jsou filtrovací podmínky realizovány pomocí netriviálních databázových PL/SQL funkcí, nelze očekávat, že by výkon dotazů byl velký. Příčina leží jak ve složitosti některých funkcí, tak v tom, že optimalizátor nemůže využít podmínky klauzule FILTER například k filtraci relací pomocí indexu, protože nerozumí sémantice našich filtrovacích funkcí.

8.11.1 Seznam trojic s objektem typu decimal

Začneme dotazem, který vrací seznam všech trojic, jejichž objekt je typu decimal.

```
select ?s ?p ?o
where { ?s ?p ?o . filter ( datatype(?o) = xsd:decimal ) }
```

Po převodu do SQL vypadá dotaz takto:

```
SELECT ...
  FROM (SELECT t1.OBJECT AS v_o,
               t1.predicate AS v_p,
               t1.subject AS v_s
          FROM triples t1
         WHERE rdf_to_bool(
                 rdf_eq(
                   rdf_type(rdf_literal_from_id (t1.OBJECT)),
                   rdf_literal_from_iri(
                     'http://www.w3.org/2001/XMLSchema#decimal'
                   )
                 )
               ) = 1)
```

Vyhodnocení tohoto dotazu probíhá tak, že server projde celou tabulku `TRIPLES` a pro každý řádek vyhodnotí uvedené funkce. Při vyhodnocení funkce rdf_literal_from_id databáze přistupuje do tabulky `LITERALS`, kde zjistí skutečnou hodnotu literálu.

To je příčina extrémní délky trvaní dotazu. Měření základní verze ukázalo, že potřebný čas dosahuje téměř 5200 vteřin. Kdyby nebyla logika filtrovací podmínky schována ve volání PL/SQL funkcí, mohla by databáze použít efektivnější metodu vyhodnocení dotazu.

8.11.2 Číselníky s alespoň 100 položkami

Jako další zkusíme dotaz, který filtrovací podmínku kombinuje se složitějším grafovým dotazem. Dosažené výsledky by měly být příznivější než v případě dotazu 8.11.1, což byl nejhorší možný případ dosažitelný s jednou trojicí.

```
select distinct ?cis_nazev
where {
  ?polozka mt:cht_ciselnik_plochy__id_ciselnik ?cis .
  ?cis mt:cst_ciselnik__nazev_tabulky ?cis_nazev .
  ?polozka mt:cht_ciselnik_plochy__poradi ?poradi .
  filter ( ?poradi >= 100 )
}
```

Výsledkem grafové části dotazu je velké množství trojic, které je pak nutné omezit filtrovací podmínkou. V tabulce uvádíme dosažené časy.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	14s	14s	12s	12s
B-strom	12s	12s	11s	11s
paralelní	12s	12s	12s	12s
bitmapa	13s	13s	12s	12s
	Počet řádků: 15			

Pro srovnání uvádíme ještě tabulku dotazu s vynecháním podmínky.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	18s	18s	10s	10s
B-strom	13s	13s	8133ms	8133ms
paralelní	17s	17s	7836ms	7836ms
bitmapa	12s	12s	6532ms	6532ms
	Počet řádků: 117			

Ztráta rychlosti způsobená filtrující podmínkou není v tomto případě velká. Navíc se opět se ukázalo, že bitmapový index je vhodný při kombinování několika trojic bez omezení na hodnotu objektu.

8.11.3 Osoby bez křestního jména

Zkusíme na reálných datech ukázat obrat použitý v 3.7.4 (negace pomocí selhání). Následující dotaz vrací osoby bez křestního jména.

```
select ?prijmeni
where {
  ?osoba mt:ot_osoba__prijmeni ?prijmeni .
  OPTIONAL { ?osoba mt:ot_osoba__jmeno ?jmeno } .
  FILTER (!bound(?jmeno))
}
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	32s	32s	24s	24s
B-strom	26s	26s	21s	21s
paralelní	56s	56s	22s	22s
bitmapa	32s	32s	23s	23s
	Počet řádků: 362			

Dotaz je pomalý, protože databáze je nucena napřed sestavit seznam všech příjmení a k nim jmen (pokud existují). A pro každou položku tohoto seznamu vyhodnotí funkci, která zjistí, že do jména nebyla přiřazena hodnota.

8.11.4 Vnořený nepovinný blok s filtrací

Upravíme dotaz 8.7.2 tak, aby vracel jména, příjmení osob a k nim města, pokud není v kontaktu zadáno PSČ.

```
select ?jmeno ?prijmeni ?mesto ?psc
where
{
  ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni .
  optional {
    ?kontakt mt:ot_kontakt__id_osoba ?osoba .
    ?kontakt mt:ot_kontakt__druh_kontaktu ?druh .
    ?druh mt:cht_ciselnik_plochy__hodnota "Adresa" .
    ?kontakt mt:ot_kontakt__mesto ?mesto .
    optional {
      ?kontakt mt:ot_kontakt__pcs ?psc
    } .
    filter (!bound(?psc))
  }
}
```

Jde opět o variantu negace selháním.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	247s	10s	217s	3044ms
B-strom	190s	3261ms	175s	1865ms
paralelní	206s	9573ms	174s	2205ms
bitmapa	255s	10s	217s	3076ms
	Počet řádků: 91201			

Doba potřebná na vyhodnocení dotazu je velmi dlouhá, je však třeba uvážit, že na poměry jazyka SPARQL jde o dosti komplikovaný dotaz.

8.12 Test na přítomnost hodnoty v databázi

Zatím jsme pokládali dotazy, které byly omezeny hlavně tvarem grafu nebo tvarem grafu a několika málo hodnotami literálů.

Avšak v systému, ze kterého data pocházejí, tvořily významnou část zátěže databáze dotazy, ve kterých byly zadány hodnoty všech vlastností kromě odkazů (cizích klíčů) a úkolem systému bylo ověřit, jestli takový záznam v databázi existuje nebo ne.

8.12.1 Existence identifikace

Napřed vyzkoušíme dotaz obsahující několik málo trojic.

```
select ?identifikace
where {
  ?identifikace mt:ot_identifikace__identifikace '8006205431' .
  ?identifikace mt:ot_identifikace__druh ?druh .
  ?druh mt:cht_ciselnik_plochy__kod '2'
}
```

Spustili jsme dvě varianty dotazu. Jedna obsahovala takové literály, aby dotaz vrátil právě jeden řádek, v případě druhé byl výsledek prázdný.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	130ms	130ms	8ms	8ms
B-strom	128ms	128ms	9ms	9ms
paralelní	94ms	93ms	8ms	8ms
bitmapa	84ms	83ms	7ms	7ms
	Počet řádků: 1			

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	77ms		5ms	
B-strom	82ms		7ms	
paralelní	66ms		7ms	
bitmapa	58ms		5ms	
	Počet řádků: 0			

Rychlost vyhodnocení tohoto dotazu je dobrá.

8.12.2 Existence osoby

Jako další variantu vyzkoušíme dotaz na existenci osoby. Ten obsahuje více trojic než dotaz předchozí.

```
select ?osoba
where {
  ?osoba mt:ot_osoba__jmeno 'Josef' .
  ?osoba mt:ot_osoba__prijmeni 'Dvořák' .
```

```
  ?osoba mt:ot_osoba__datum_narozeni '1968-04-06T00:00:00' .
  ?osoba mt:ot_osoba__rok_maturity '1987' .
  ?osoba mt:ot_osoba__trvaly_pobyt_v_cr 'A' .
  ?osoba mt:ot_osoba__pohlavi ?pohlavi .
  ?pohlavi mt:cht_ciselnik_plochy__kod '1'
}
```

Opět srovnáme verze, kdy je vyhledávání úspěšné a neúspěšné.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	3935ms	3935ms	700ms	700ms
B-strom	649ms	649ms	79ms	79ms
paralelní	2701ms	2701ms	77ms	77ms
bitmapa	2847ms	2847ms	698ms	698ms
	Počet řádků: 1			

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	8129ms		1154ms	
B-strom	826ms		117ms	
paralelní	4805ms		118ms	
bitmapa	5225ms		1147ms	
	Počet řádků: 0			

Na rozdíl od předchozího případu jsou časy, které dosahuje první spuštění dotazu, značně neuspokojivé.

8.13 Vyhodnocení měření

Měření přinesla výsledky užitečné v několika směrech.

8.13.1 Uložení dat v databázi

Jako nejrychlejší varianta pro uložení tabulky `TRIPLES` se ukázala indexová organizace. Paralelní přístup k této tabulce se však neukázal jako dobrá volba. Paralelizace sice v některých případech vede ke zlepšení, ale v mnoha případech také k výraznému zhoršení.

Pro indexaci sloupce `predicate` tabulky `TRIPLES` organizované běžným způsobem se ukázal bitmapový index jako lepší než B-strom.

Pokud jde o prostorové nároky, tak nejméně místa zabírá (při započtení velikosti všech indexů) tradiční organizace s bitmapovým indexem nad sloupcem `predicate`, nejvíce místa zabíra indexová organizace. Nejhorší výsledek indexové organizace nespočívá v tom, že by byla větší samotná tabulka, ale v tom, že indexy nad takovouto tabulkou jsou asi dvakrát větší než při tradiční organizaci. To převáží i fakt, že při indexové organizaci již nemusíme zakládat index nad sloupcem `predicate`.

8.13.2 Celková rychlost systému

Celkově se však ukázalo, že ve většině případů není rychlost vyhodnocení dostatečná pro systém s interakcí uživatele. Na druhou stranu v mnoha případech byla doba potřebná pro vrácení prvního záznamu výrazně kratší než vyhodnocení celého dotazu. Doba potřebná pro zobrazení části záznamů, které by byl uživatel ještě schopen ručně zpracovat, proto může být dostatečná. Ale i v mnoha případech dotazů s malým výsledkem byla rychlost nedostatečná.

8.13.3 Příčiny nedostatečného výkonu

Při pohledu na plány vyhodnocení dotazů se ukazuje, že optimalizátor dotazů v Oracle špatně odhaduje počty řádek, které vzniknou spojením tabulek při vyhodnocování SQL dotazu. Proto často volí neoptimální pořadí a algoritmy spojení. Obvyklý odhad pro velikost výsledku je jeden řádek, proto zvolí spojení pomocí hnízděných cyklů. Ve skutečnosti bývají počty často kolem několika tisíc a vhodnější by bylo například hašované spojení.

V kapitolách 9 a 10 se budeme některými z výkonnostních problémů zabývat a ukážeme i jedno možné zlepšení.

9 Indexy

Už u jednoduchých seznamů se ukazuje slabina celého systému. Pro jejich vytvoření je nutné provést velká spojení, což sráží jejich výkon. Bylo by vhodné předpřipravit si již spojená data. Pokud by dotaz vypadal stejně, jako dotaz na seznam osob na začátku této kapitoly, pak by zjevně bylo přínosné vytvořit tabulku, která by obsahovala právě výsledek následujícího dotazu:

```
select ?osoba ?jmeno ?prijmeni
where { ?osoba mt:osoba__jmeno ?jmeno .
  ?osoba mt:osoba__prijmeni ?prijmeni }
```

Tím vznikne v databázi struktura naplněná daty, která může urychlit přístup k RDF datům. Proto ji budeme v dalším textu nazývat RDF index.

Pro uložení této tabulky by bylo pravděpodobně výhodné použít indexovou organizaci, která se již osvědčila pro uložení tabulky `TRIPLES`. Pořadí sloupců v tabulce ponecháme stejné, jako bylo pořadí v dotaze. Tím by se RDF index ještě více přiblížil běžnému indexu, včetně toho, že volbu optimálního pořadí sloupců ponechává na uživateli databáze.

9.1 Udržování jednoduchého indexu

Problematické je udržování indexu při změně dat v databázi. Jedna možnost by byla pro každou přidanou trojici $< s', p', o' >$ zjistit, jaké řádky je potřeba přidat do indexu a skutečně je přidat.

Napřed se zaměřme na takový dotaz D, ve kterém není nikde použita proměnná jako predikát a všechny predikáty jsou různé. Navíc D nesmí obsahovat anonymní uzly, OPTIONAL, UNION ani FILTER a všechny zkratky pro zápis dotazu musí být rozloženy na základní tvar. Dotaz může obsahovat literály.

Označme G všechny grafy, které odpovídají dotazu definujícímu index, po přidání nové trojice. Přesněji G je množina RDF grafů, které dostaneme, pokud do trojic tvořících dotaz D dosadíme postupně všechna přípustná ohodnocení proměnných, která jsou řešením dotazu D.

G lze rozdělit na disjunktní množiny G_1 a G_2, kde grafy v G_1 neobsahují nově přidanou trojici a grafy v G_2 ano. Přitom platí, že G_1 jsou právě výsledky dotazu D před přidáním nové trojice.

Protože všechny predikáty v dotaze jsou různé, je místo, kde se nová trojice vyskytuje v grafech z množiny G_2, jednoznačně určené. Díky tomu můžeme snadno sestavit dotaz D', jehož výsledkem jsou právě grafy z G_2.

Označme s subjekt trojice z dotazu D, jejíž predikát je shodný s predikátem nové trojice. Obdobně označme o objekt této trojice.

Vytvoříme dotaz D_s. Může nastat jedna z následujících možností

- s je URI. Pak pokud $s = s'$ bude $D_s = D$, jinak platí $G_2 = \emptyset$.
- s je proměnná. Pak D_s získáme nahrazením s' (s' je URI nebo anonymní uzel) za všechny výskyty s v dotaze D.

Z dotazu D_s pak vytvoříme dotaz D' takto

- o je URI nebo literál. Pak pokud $o = o'$ bude $D' = D_s$, jinak platí $G_2 = \emptyset$.
- o je proměnná. Pak D' získáme nahrazením o' (o' je literál, URI nebo anonymní uzel) za všechny výskyty o v dotaze D.

Výsledkem dotazu D' jsou právě záznamy, které je nutné přidat do indexu, aby odpovídal RDF grafu po přidání $< s', p', o' >$.

9.2 Udržování složitého indexu

Dotaz, na jehož základě jsme index vytvořili, může být výrazně složitější, než dovolují omezení v části 9.1. Také v případě, kdy nově přidaných trojic je velké množství, nemusí být uvedený postup, který pro každou trojici vyhodnotí dotaz D', optimální.

Alternativou, kterou lze použít i u složitých indexů, je celý index smazat a znovu vytvořit. To se může ukázat jako vhodnější i pro velké změny v jednoduchém indexu.

9.3 Detekce použitelnosti indexů

Obzvláště u složitějších indexů nebude snadné zjistit, kdy lze index použít pro vyhodnocení dotazu. Další zajímavý problém je automatické vytváření indexu podle statistik používaných dotazů.

Obecné řešení těchto problémů je však nad rámec této práce. Proto se zaměříme pouze na použití jednoduchých indexů v dotazech.

9.3.1 Jeden jednoduchý index

V této sekci budeme používat funkci $graph(D)$, která z derivačního stromu základního grafového dotazu D vytvoří pseudo-RDF graf obsahující právě trojice tvořící dotaz D. Protože D může obsahovat proměnné, není $graph(D)$ obecně RDF graf. Funkce $var(D)$ vrací množinu proměnných, které jsou použity v dotaze D.

Dále označme RDF-L množinu všech literálů.

Jednoduchý index definovaný dotazem D je *použitelný* pro vyhodnocení dotazu Q, pokud existuje funkce $I : var(D) \rightarrow \text{RDF-L} \cup var(Q)$ taková, že $graph(I(D)) \subseteq graph(Q)$. $I(D)$ značí aplikaci funkce I na všechny proměnné v dotaze D. Navíc musí platit, že $I(x) \in$ RDF-L pouze v případě, že se proměnná x v dotaze D vyskytuje nejvýše jednou.

Použitelný index D ovlivní výsledek převodu dotazu Q ze SPARQL do SQL. K SQL dotazu $F(Q)$ vytvořenému podle postupu z kapitoly 5 definujeme SQL dotaz využívající index $F_I(Q, D)$. $F_I(Q, D)$ vznikne tak, že z $F(Q)$ odstraníme výskyty tabulky `TRIPLES` odpovídající trojicím z $I(D)$ a přidáme spojení na tabulku reprezentující index D v databázi. Podmínky spojení, které se vázaly na odstraněné tabulky `TRIPLES`, je třeba nahradit spojením na tabulku indexu, čehož lze dosáhnout například přejmenováním sloupců tabulky indexu pomocí funkce I.

9.3.2 Více jednoduchých indexů

Naše implementace používá jednoduchý hladový algoritmus, který postupně prochází dostupné indexy, a pokud je některý použitelný, tak jej použije. Tento postup se opakuje, pokud se v poslední iteraci vyskytnul použitelný index.

Protože použitím každého indexu je z dotazu odstraněn alespoň jeden výskyt tabulky TRIPLES, algoritmus vždy končí.

Jednoduše lze nalézt příklad, který ukazuje, že algoritmus obecně nenalezne optimální řešení, pokud vezmeme jako metriku počet odstraněných výskytů tabulky TRIPLES.

9.4 Použití indexů nad testovacími daty

V následujícím textu se budeme zabývat použitím indexů pro zrychlení dotazů nad našimi testovacími daty. Aby bylo možné indexy pohodlně tvořit,

rozšířili jsme syntax jazyka SPARQL o dotazy na tvorbu indexů. Dotaz vypadá takto:

```
CREATE INDEX jméno_indexu
AS ?proměnná1 ?proměnná2 ...
WHERE jednoduchý_grafový_dotaz
```

Pořadí proměnných za AS určuje pořadí proměnných v indexu. Grafový dotaz za WHERE musí vyhovovat podmínce na dotaz definující jednoduchý index. Na začátek dotazu je možné přidat deklarace prefixů jako v běžném dotaze SELECT.

U obou typů dotazů v této kapitole vynecháváme prefixy stejně jako v kapitole 8.

9.4.1 Použití indexu na seznam osob

Uvažujme první dotaz kapitoly 8, tj.

```
select ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni }
```

Pro jeho vyhodnocení můžeme použít index:

```
create index osobajmena as
?osoba ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni }
```

Pro založení indexu v databázi vygeneruje systém následující SQL dotazy:

```
CREATE TABLE osobajmena(
  osoba NUMBER(18),
  jmeno NUMBER(18),
  prijmeni NUMBER(18),
  PRIMARY KEY (osoba, jmeno, prijmeni)
)
ORGANIZATION INDEX;
```

```
INSERT INTO osobajmena
            (osoba, jmeno, prijmeni)
SELECT v_osoba AS osoba,
       v_jmeno AS jmeno,
       v_prijmeni AS prijmeni
  FROM (SELECT t1.OBJECT AS v_jmeno, t1.subject AS v_osoba,
               t3.OBJECT AS v_prijmeni
          FROM triples t1
    INNER JOIN literals t2 ON t2.ID = t1.predicate
               INNER JOIN triples t3 ON t1.subject = t3.subject
               INNER JOIN literals t4 ON t4.ID = t3.predicate
         WHERE t2.VALUE =
   'http://www.is.cuni.cz/stoh/metadata/ot_osoba__jmeno'
           AND t4.VALUE =
   'http://www.is.cuni.cz/stoh/metadata/ot_osoba__prijmeni')
```

SQL dotaz, kterým bude dotaz vyhodnocen, vypadá následujícím způsobem. Pro zkrácení zápisu vynecháváme seznam část za prvním select, která je shodná se začátkem neindexované verze.

```
SELECT ...
  FROM (SELECT t1.v_jmeno AS v_jmeno,
               t1.v_osoba AS v_osoba,
               t1.v_prijmeni AS v_prijmeni
          FROM osobajmena t1)
```

Následující tabulka ukazuje rychlost dotazu při použití indexu:

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	5736ms	204ms	5998ms	150ms
B-strom	5548ms	199ms	5308ms	141ms
paralelní	6069ms	170ms	5200ms	121ms
bitmapa	5713ms	290ms	5329ms	190ms
	Počet řádků: 91166			

Připomeneme ještě rychlost bez indexu:

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	20s	2920ms	11s	914ms
B-strom	14s	878ms	13s	292ms
paralelní	13s	369ms	12s	315ms
bitmapa	19s	4634ms	11s	902ms
	Počet řádků: 91166			

Přínos indexu je zcela zjevný.

9.4.2 Použití indexu pro zjištění jména z příjmení

Protože se proměnné ?jmeno a ?prijmeni v indexu vyskytují pouze jednou, můžeme index použít i pro vyhledání konkrétních hodnot.

Protože na prvním místě je proměnná osoba, není možné vyhledat přímo záznam s konkrétním jménem průchodem jediné větve B-stromu, ve kterém je index uložen. Databázový systém je naopak nucen projít všechny listy. Ale jejich počet je ve srovnání s velikostí tabulky trojic malý, proto může být vhodné uvážit použití indexu i přes nutnost jej zpracovat celý.

```
select ?jmeno
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni "Smith" }
```

Na tento dotaz je náš index použitelný, pokud uvážíme $I(prijmeni) =$ "*Smith*". Vyhodnocení dotazu zabere 58/26ms (první/opakované spuštění) v indexované a 36/7ms v neindexované verzi. Obě tedy podávají dobré výkony, ale neindexovaná je rychlejší.

V tomto jednoduchém případě se použití indexu neukazuje jako výhodné.

9.4.3 Index pro rodná čísla

V této části ukážeme složitější index. Jde o index, který spojuje osoby a rodná čísla. Oproti indexu nad jmény osob obsahuje literál na místě jednoho z objektů.

```
create index rodnacisla as ?osoba ?rc ?ident ?rckod
where { ?ident mt:ot_identifikace__id_osoba ?osoba .
?ident mt:ot_identifikace__druh ?rckod .
?rckod mt:cht_ciselnik_plochy__kod "2" .
?ident mt:ot_identifikace__identifikace ?rc }
```

Napřed zkusíme dotaz, který přesně odpovídá indexu.

```
select ?osoba ?rc
where {
?ident mt:ot_identifikace__id_osoba ?osoba .
?ident mt:ot_identifikace__druh ?rckod .
?rckod mt:cht_ciselnik_plochy__kod "2" .
?ident mt:ot_identifikace__identifikace ?rc
}
```

V tomto případě se indexovaná verze ukazuje jako výrazně lepší než neindexovaná. V případě indexované dotaz potřebuje 18 vteřin při prvním spuštění a 5 vteřin pro opakované, zatímco neindexovaná verze trvá 92 respektive 42 vteřin.

9.4.4 Složitější dotaz využívající index nad rodnými čísly

Následující dotaz kombinuje složitou část pokrytou indexem a složitou část mimo index. Jeho výsledkem jsou dvojice rodné číslo a email pro osoby, které mají oba údaje zadány.

```
select ?rc ?email
where {
  ?kontakt mt:ot_kontakt__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__druh_kontaktu ?dkon .
  ?kontakt mt:ot_kontakt__email ?email .
  ?dkon mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__id_osoba ?osoba .
  ?ident mt:ot_identifikace__druh ?rckod .
  ?rckod mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__identifikace ?rc
}
```

Při vyhodnocení tohoto dotazu se ukázal problém s převodem do SQL a vyhodnocení v databázovém stroji. Výhoda tohoto přístupu, který přenechává optimalizaci plně v režii databázového stroje, se zároveň ukazuje jako nevýhoda.

Tímto konkrétním dotazem se nám zjevně podařilo strefit se do hodnot, u nichž jsou malé rozdíly v odhadu cen jednotlivých plánů vykonání dotazu.

To by vysvětlovalo původ velkých rozdílů mezi jednotlivými verzemi neindexovaných dotazů, protože i drobná změna v pořadí spojení tabulek `TRIPLES` nebo použití různých přístupových metod můžou vést k zásadní změně velikosti a rychlosti spojení tabulek.

V následující tabulce je rychlost při použití indexu.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	13s	4072ms	12s	4248ms
B-strom	1656ms	1149ms	1143ms	708ms
paralelní	1751ms	1259ms	1149ms	717ms
bitmapa	8903ms	3640ms	8122ms	3554ms
	Počet řádků: 7861			

Tato tabulka ukazuje rychlost bez indexu.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	54s	44s	54s	44s
B-strom	717s	716s	745s	743s
paralelní	16s	11s	14s	10s
bitmapa	275s	101s	286s	105s
	Počet řádků: 7861			

Přestože výsledky měření vykazují značné anomálie, vliv indexu je zjevně pozitivní. K odstranění anomálního chování by mohl vést postup, který nastíníme v kapitole 10.

9.4.5 Dotaz kombinující indexy

V této části budeme zkoumat dotaz, který kombinuje jak index nad jmény osob, tak index nad rodnými čísly. Jde o rozšíření předchozího dotazu o zjištění jména a příjmení.

```
select ?rc ?email ?jmeno ?prijmeni
where {
  ?kontakt mt:ot_kontakt__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__druh_kontaktu ?dkon .
  ?kontakt mt:ot_kontakt__email ?email .
  ?dkon mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__id_osoba ?osoba .
```

```
  ?ident mt:ot_identifikace__druh ?rckod .
  ?rckod mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__identifikace ?rc .
  ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni
}
```

Změřili jsme všechny čtyři kombinace přítomnosti a nepřítomnosti indexů, vždy se základní verzí reprezentace tabulky trojic. Pořadí jednotlivých řešení vypadá podle očekávání.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
oba indexy	14s	4551ms	12s	3521ms
rodná čísla	18s	5090ms	17s	4420ms
osoby	114s	11s	101s	11s
bez indexu	204s	194s	71s	61s
	Počet řádků: 7859			

Nejrychlejší bylo použití obou indexů, druhé pak použití indexu na rodná čísla. Výrazně horší čas byl dosažen použitím indexu nad jmény osob. Nejpomalejší při prvním spuštění byl běh bez indexů. Jeho druhé spuštění však seběhlo za 71 vteřin a tím překonalo variantu s indexem nad jmény osob.

9.5 Přínos

Experimenty ukazují, že použití jednoduchých indexů skutečně pozitivně ovlivňuje rychlost dotazů.

Bude však ještě potřeba zdokonalit metodu volby optimálních indexů. Další výzkum by se mohl zaměřit i na složitější indexy nebo na možnost automaticky navrhovat nové indexy na základě statistiky vykonávaných dotazů a jejich rychlostí.

10 Statistiky

Jak ukazuje pohled na plány vyhodnocení SQL dotazů v databázi, při velkém počtu spojených tabulek `TRIPLES` optimalizátor v Oracle předpokládá, že výsledek spojení bude obsahovat jen jednu řádku. To obvykle není pravda.

Na základě tohoto odhadu pak provádí spojení pomocí hnízděných cyklů, což není vhodné, pokud jsou spojovány velké relace.

Oracle umožňuje ovlivnit plán vyhodnocení konkrétního dotazu pomocí speciálně formátovaného komentáře, který se pak nazývá *hint*.

10.1 Statistiky o predikátech

S ohledem na jejich malý počet může mít systém realizující RDF dotazy neustále k dispozici statistiky o četnosti všech predikátů. Z toho systém může usuzovat na velikost spojení a podle toho zvolit vhodnou metodu realizace spojení.

10.2 Statistiky o subjektech

Jak se ukázalo v kapitole 7, četnosti subjektů se pohybují v malém rozsahu. Proto by uchovávaní podrobných statistik nejspíše nemělo smysl.

10.3 Statistiky o objektech

Počty výskytu jednotlivých literálů ve sloupce `object` se mohou výrazně lišit. To má zásadní vliv na výsledný výkon dotazu, protože optimalizátor v Oracle obvykle používá selekci na sloupec `object` na nejhlubší listové úrovni stromu vyhodnocení dotazu. A pokud tato selekce vrátí velký počet řádků, může dojít k výraznému zpomalení celého dotazu.

Bylo by možné k tabulce `LITERALS` přidat další sloupec, ve kterém by byla právě četnost literálu v `TRIPLES.object`.

Systém převádějící SPARQL dotazy do SQL by pak potřeboval přístup do databáze, ze které by si přečetl údaje potřebné k optimalizaci aktuálního dotazu. Tento dotaz jde vyřídit velmi rychle a výsledky by bylo možné ukládat do cache, případně spoléhat na cache v databázi.

Změřili jsme rychlost obdobného dotazu a při prvním spuštění dosáhla 153ms, při druhém 6ms. Rychlost by tedy měla být dostatečná, protože pro

každý dotaz bude potřeba převést pouze literály v něm obsažené. Těch bude obvykle velmi málo (jednotky).

Zároveň by bylo možné spojení na tabulku `LITERALS`, na kterou je provedena selekce podle hodnoty literálu, nahradit v SQL přímo selekcí na identifikátor literálu, který si můžeme nechat vrátit z databáze zároveň se statistikou četnosti.

Navíc, pokud se literál v databázi nenachází, můžeme celý základní graf v SPARQL dotaze převést do SQL jako dotaz, který vrátí prázdnou relaci. To může ušetřit čas potřebný na vyhodnocení dotazu.

10.4 Konstrukce plánu pro vyhodnocení SQL dotazu

Pomocí hintů můžeme upravit plán vyhodnocení SQL dotazu tak, aby lépe odpovídal datům v databázi, protože o nich máme k dispozici podrobnější statistiky než optimalizátor v databázovém stroji.

Konstrukce tohoto plánu je však složitá a mimo rozsah tohoto textu, ale chceme se na ni zaměřit v dalším výzkumu.

11 Závěr

Cílem práce bylo vytvořit implementaci systému, který by umožnil dotazování nad RDF daty. Implementovali jsme významnou část připravovaného W3C standardu SPARQL. I přes nedostatky v jeho formální definici jde o zajímavý jazyk s dobrou vyjadřovací schopností.

Napřed jsme navrhli formát uložení RDF dat v relační databázi. Ten vychází z myšlenky uložení RDF trojic do tabulky se třemi sloupci. Bylo však nutné vyřešit různé technické problémy, což si vynutilo o něco složitější reprezentaci RDF trojic a literálů.

Nad takto uloženými daty jsme navrhli metodu převodu SPARQL dotazů na dotazy jazyka SQL. Ta vychází z přímého překladu základních grafových dotazů, ze kterých je SPARQL dotaz sestaven, na SQL a jejich zkombinování způsobem, který odpovídá kombinaci základních grafových dotazů v překládaném SPARQL dotazu.

Výkon přeložených dotazů jsme otestovali nad rozsáhlými daty a pokusili se zanalyzovat, kde leží výkonnostní problémy. Rychlost dotazů se v mnoha případech ukázala jako neuspokojivá. Zároveň jsme během měření srovnali různé datové struktury pro uložení a indexaci trojic v rekační databázi. Ukázalo se, že mezi testovanými strukturami jsou v mnoha případech výrazné rozdíly. Jako nejvhodnější ve většině případů vycházela indexově organizovaná tabulka, tj. uložení trojic v B-stromu, jako by se nejednalo o tabulku, ale o index. Tato reprezentace je však náročnější na prostor na disku. Z hlediska nároků na místo na disku vychází nejlépe bitmapový index.

Jedním z hlavních problémů, které brzdí vyhodnocení dotazů, je velké množství spojení tabulky obsahující trojice. Proto jsme navrhli a implementovali tzv. jednoduché RDF indexy, které uchovávají v databázi již spojené trojice určitého typu. O jaké trojice se jedná, definuje uživatel databáze při zakládání indexu. Tento index lze použít ke zrychlení vyhodnocení dotazu, jehož částí je dotaz, kterým byl index definován. Provedli jsme experimenty, které potvrdily pozitivní přínos indexů při vyhodnocení dotazů.

Náš experimentální systém je pouze základ, v jehož vývoji bude nutné i nadále pokračovat. Rádi bychom v budoucnu přidali například další konstrukce jazyka SPARQL nebo podporu složitějších indexů. Zajímavé by se mohlo ukázat i využití hlubších znalostí o struktuře a četnostech uložených RDF dat ke konstrukci lepších plánů vyhodnocení pro SQL dotazy vzniklé ze SPARQL dotazů. Univerzálně zaměřený optimalizátor relační databáze velmi často tvoří výrazně suboptimální plány.

Literatura

[1] Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Schol D.: *RQL: A Declarative Query Language for RDF*, in Proceedings of the Eleventh International World Wide Web Conference, USA, 2002

[2] Alvestrand H. (2001): *Tags for the Identification of Languages* http://www.ietf.org/rfc/rfc3066.txt

[3] Berners-Lee T., Hendler J., Lassila O. (2001): *The Semantic Web*, Scientific American, May 2001

[4] Biron P. V., Malhotra A. (2004): *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation, 28 October 2004 http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[5] Broekstra J., Haase P. (2004): *A comparison of RDF query languages*, in Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004, 502-517

[6] Broekstra J., Kampman A.: *SeRQL: A Second Generation RDF Query Language*, SWAD-Europe, Workshop on Semantic Web Storage and Retrieval, Netherlands, 2003

[7] Carroll J. J., Klyne G. (2004): *Resource Description Framework: Concepts and Abstract Syntax*, W3C Recommendation, 10 February 2004 http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[8] Dokulil J. (2006): *Transforming Data from DataPile Structure into RDF*, in Proceedings of the Dateso 2006 Workshop, Desna, Czech Republic, 2006, 54-62 http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-176/paper8.pdf

[9] Mössenböck, H. : Coco/R for various languages - Online Documentation http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/

[10] Prud'hommeaux E., Seaborne A. (2004): SPARQL Query Language for RDF, W3C Working Draft, 12 October 2004 http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/

[11] Prud'hommeaux E., Seaborne A. (2005): SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005
http://www.w3.org/TR/2006/WD-rdf-sparql-query-20060220/

[12] Seaborne A.: *RDQL - A Query Language for RDF*, W3C Member Submission, 9 January 2004
http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[13] *RDFQL Scripting Reference*
RDFQL http://www.intellidimension.com/default.rsp?
topic=/pages/rdfgateway/reference/script/default.rsp