SPARQL jako objektově-orientovaný dotazovací jazyk

Tomáš Burger, FIT VUT Brno, burger@fit.vutbr.cz

SPARQL je dotazovací jazyk pro RDF. Jakub Gütner ukázal ve své disertaci souvislost objektově orientované databáze a RDF. Podívejme se teď, lze-li tuto souvislost rozšířit i na dotazovací jazyky – tj. objektově orientované OQL versus SPARQL určený pro RDF.

Antipasta (úvod)

Relační databáze ukládají data jako relace hodnot stále stejné struktury (tedy jako řádky v tabulce). Objektově orientované databáze ukládají data objekt po objektu do svých extentů (a podobně to dělají i XML databáze – dokument po dokumentu). RDF se vydává opačným směrem a místo aby hledalo vyšší logické celky, zredukovalo celý databázový model na jednu jedinou strukturu, zvanou RDF Statement. RDF Statement je trojice: subjekt, predikát a objekt. RDF data jsou pak jen množina takovýchto trojic, která mohou pouze být obohacena o RDF schéma, díky kterému lze interpretovat trojice a jejich vztahy a zkoumat sémantiku RDF dat až na samu hranici umělé inteligence.

RDF statement se tedy skládá ze subjektu, predikátu a objektu. Subjekt a predikát musí být vždy URI (resource identifier), který musí být sémanticky jednoznačný (tj. musí pokaždé označovat totéž), aniž by byl na URI kladen jakýkoliv další požadavek nad rámec specifikace RFC.

Objekt může být buď URI nebo literál určitého typu podle specifikace XML Schématu. Pokud je objekt URI, může sloužit jako subjekt v dalším statementu a tak lze z RDF dat budovat orientované grafy, kde URI a literály slouží jako uzly grafu a predikáty jako orientované hrany (od subjektu k objektu).

Příklad RDF dat pak může vypadat třeba takto (používáme tzv. N3 notaci a RDF schéma zvané FOAF – „friend of a friend“, určené k „social networkingu“):

```
@prefix foaf: <http://xmlns.com/foaf/0.1> .
@prefix vutbr: <http://xml.vutbr.cz/> .
vutbr:xburge05 foaf:mbox <mailto:xburge05@fit.vutbr.cz> .
vutbr:xburge05 foaf:name "Tomas Burger" .
```

Specifický typ subjektu nebo objektu je tzv. blank node, což je vlastně dočasné, transientní URI, platné jen v rámci jedněch RDF dat a určené ke spojování RDF statementů v případě, že stálé, persistentní URI pro subjekt nebo objekt není k dispozici. Blank nodes se zapisují s podtržítkem jako prvním znakem.

Můžeme tedy k našemu příkladu přidat:

```
vutbr:xburge05 foaf:knows _a .
_a foaf:mbox <hruska@fit.vutbr.cz> .
_a foaf:homepage <http://www.fit.vutbr.cz/~hruska> .
```

Takto zapsané RDF nám umožňuje specifikovat, že osoba vutbr:xburge05 zná kohosi s emailem hruska@fit.vutbr.cz, aniž bychom museli znát jeho URI.

Podle předpokladů, které stály u vzniku tzv. sémantického webu a z něhož je RDF odvozeno, by RDF statementy měly být rozesety po celém Internetu, uloženy ve standardních HTML stránkách, kde by specifikovaly obsah stránky formou srozumitelnou strojům. Pomocí RDF schémat a webových ontologií (což jsou schémata, která mohou definovat pokročilejší vztahy mezi RDF statementy a třeba i mezi různými schématy) by inteligentní stroje, zvané „inteligentní agenti“, surfovaly po internetu, porozuměly obsahu HTML stránek a dohledávaly svým uživatelům hotová, předzpracovaná data.

Jsme daleko od tohoto cíle. RDF dnes slouží spíš jako maximálně flexibilní datový formát, když ani relační tabulky ani objekty či XML dokumenty nevyhovují.

Pokud tedy máme RDF databázi, což je vlastně jedna „tabulka“ statementů, chtěly bychom v ní vyhledávat. K tomu slouží standardní dotazovací jazyk SPARQL (je to zatím jen návrh standardu, ale některé existující dotazovací jazyky – jako RDQL – jsou jazyku SPARQL velmi podobné).

SPARQL-dotaz do RDF dat má dvě hlavní části: vyhledávací vzory (pattern) a formát výsledku. Vyhledávací vzor je „jakoby“ RDF statement, kde některé pozice jsou nahrazené proměnnými. Výsledkem hledání v RDF datech jsou série hodnot proměnných, které vyhovují všem vyhledávacím vzorům v dotazu. Z těchto hodnot lze pak zformátovat výsledek dotazu – buď pouze vypsat do tabulky (SELECT) nebo zkonstruovat nové RDF statementy (CONSTRUCT).
Přidejme si ještě nějaká „demo“ data:

```
vutbr:xburge05 foaf:knows vutbr:masarik .
vutbr:masarik foaf:knows _b .
_b foaf:homepage <http://www.fit.vutbr.cz/~hruska> .
vutbr:masarik foaf:homepage <http://www.fit.vutbr.cz/~masarik> .
```

Sestavme SPARQL dotaz, který vypíše URL domácích stránek lidí, které zná xburge05.

```
SELECT ?homepage
WHERE { vutbr:xburge05 foaf:knows ?someone .
  ?someone foaf:homepage ?homepage }
```

Výsledkem dotazu budou dvě hodnoty proměnné „homepage“:

- <http://www.fit.vutbr.cz/~hruska>
- <http://www.fit.vutbr.cz/~masarik>

Dotaz

```
SELECT ?mbox ?homepage
WHERE { vutbr:xburge05 foaf:knows ?someone .
  ?someone foaf:homepage ?homepage .
  ?someone foaf:mbox ?mbox . }
```

vrátí jediný výsledek [<hruska@fit.vutbr.cz>, <http://www.fit.vutbr.cz/~hruska>], protože naše demo data neobsahují email Karla Masaříka. To se ale dá spravit pomocí nepovinných (optional) vzorů:

```
SELECT ?mbox ?homepage
WHERE { vutbr:xburge05 foaf:knows ?someone .
  ?someone foaf:homepage ?homepage .
  OPTIONAL { ?someone foaf:mbox ?mbox } }
```

kde i případ s neznámým emailem bude shledán platným a dotaz vrátí dva výsledky:

mbox	homepage
<hruska@fit.vutbr.cz>	<http://www.fit.vutbr.cz/~hruska>
	<http://www.fit.vutbr.cz/~masarik>

Dotazy lze doplnit i logickými podmínkami, takže zatímco

```
SELECT ?mbox
WHERE { ?anyone foaf: mbox ?mbox }
```

vrátí dva emaily, dotaz

```
SELECT ?mbox
WHERE { ?anyone foaf:mbox ?mbox . FILTER !isblank(?anyone) }
```

vrátí email pouze jeden (isblank() je standardni SPARQL operátor, vedle toho je ale k dispozici celá škála aritmetických, logických i řetězcových operátorů, převzatá najmě ze specifikace standardu XPath).
Místo klauzule SELECT je možné použít klauzuli CONSTRUCT a sestavit tak z hodnot proměnných nové RDF statementy:

```
@prefix dc: <http://purlo.org/dc/elements/1.1/>
CONSTRUCT { ?homepage dc:creator ?mbox}
WHERE { ?someone foaf:mbox ?mbox .
  ?someone foaf:homepage ?homepage }
```

„DC“ je obvyklý prefix standardního RDF schématu, zvaného Dublin-core, které obsahuje elementy pro anotaci textů, jako datum publikace, identifikaci autora a editora atd. Uvedený dotaz vyprodukuje jeden nový RDF statement, protože pouze kombinace [<hruska@fit.vutbr.cz>, <http://www.fit.vutbr.cz/~hruska>] splňuje oba vzory.
SPARQL také disponuje konstrukty obvyklými v SQL jako DISTINCT, ORDER BY, LIMIT a OFFSET, a to s obvyklým významem.

Dále v textu se pokusíme o následující: transformujeme nejprve relační (toť jest pokus o vlastní „objev“) a pak objektově orientovanou databázi (podle specifikace J. Güttnera) na RDF data a pokusíme se pomocí SPARQL simulovat standardní vyhledávací jazyky – totiž relační algebru (a SQL) pro relační data a OQL podle ODMG standardu pro objektově-orientovaná data.

Primi piati (první kolo)

Coddova relační algebra je postavena na pojmu relace a na definici několika základních operací. Doména je množina možných hodnot a pokud máme dáno několik takovýchto domén, podmnožina jejich kartézského součinu je relace. Řečeno databázově: doména je políčko v definici tabulky, množina hodnot je dána typem políčka (doménou) a kartézský součin jsou všechny možné kombinace hodnot, které může obsahovat jedna věta v tabulce. – Tabulku pak lze samozřejmě chápat jako podmnožinu takovéhoto kartézského součinu.
Relační algebra pak definuje osm základních operací nad relacemi, z nichž nejpodstatnější jsou tři: SELECT, PROJECT a JOIN. Každá operace má na vstupu jednu nebo dvě (pro JOIN) relace a na výstupu zase relaci.
Vezměme opět podobná data jako příklad:

table FIT		
nickname	email	homepage
xburge05	burger@fit.vutbr.cz	http://www.burger.cz/
hruska	hruska@fit.vutbr.cz	http://www.fit.vutbr.cz/~hruska
masarik	masarik@fit.vutbr.cz	http://www.fit.vutbr.cz/~masarik

SELECT vybere z relace podmnožinu podle specifikované podmínky:

```
SELECT FIT WHERE left(nickname,1) = „X“ GIVING STUDENTS
```

Výsledek bude jedna řádka:

table STUDENTS		
nickname	email	homepage
xburge05	burger@fit.vutbr.cz	http://www.burger.cz/

Operace PROJECT provede výběr sloupců:

```
PROJECT FIT OVER [nickname, email] GIVING EMAILS
```

Výsledek pak bude následující:

table EMAILS	
nickname	email
xburge05	burger@fit.vutbr.cz
hruska	hruska@fit.vutbr.cz
masarik	masarik@fit.vutbr.cz

Operace JOIN spojí dvě relace tak, že sloučí množinu jejich domén a zahrne do výsledku ty kombinace, které se shodují na společných doménách (standardní databázový „inner join"). Pro demonstraci si přidejme ještě jednu tabulku:

table PHD	
nickname	tutor
xburge05	hruska
masarik	hruska
lukas	meduna

Operace

```
JOIN EMAILS WITH PHD GIVING PHDTUTORS
```

pak vrátí ve výsledku následující relaci:

table PHDTUTORS		
nickname	email	tutor
xburge05	burger@fit.vutbr.cz	hruska
masarik	masarik@fit.vutbr.cz	hruska

Následně se pokusme najít obecný postup, jak zakódovat relaci do RDF takovým způsobem, aby bylo možné operace jako SELECT, PROJECT a JOIN vyjádřit ve SPARQL.
Postupovat budeme takto:

- definujme si speciální jmenný prostor CODD.
- zaveďme jeden speciální predikát pro označení relace – CODD:RELATION.
- dále zaveďme predikáty pro domény v tabulkách – jako například VUTBR:NICKNAME (domény nejsou specifické pro relační algebru, ale pro datový model, proto nepatří do jmenného prostoru relační algebry, ale „aplikace“).
- ujasněme si RDF typ pro každou doménu v tabulkách – jestli je to literál (a jakého typu) anebo je to URI (obecně je výhodnější co možná nejvíc dat transformovat na URI, ačkoliv to nebude vždycky úplně možné).
- zaveďme také URI pro každou relaci/tabulku v jmenném prostoru aplikace – jako například VUTBR:FIT.
- následně můžeme pro každý prvek v relaci a pro každou doménu generovat RDF statementy.

Protože relace v relační algebře nemá identitu – je to jen vektor hodnot – bude pro modelování subjektu nejvhodnější zvolit blank node. Vezměme tedy jednu kombinaci z relace (například

xburge05	burger@fit.vutbr.cz	http://www.burger.cz/

z relace FIT) a zvolme volný blank node – třeba _r1. Nejprve přiřadíme zvolenou kombinaci do relace:

```
    _r1 CODD:RELATION VUTBR:FIT .
```

a pak vypíšeme doménu po doméně všechny hodnoty v kombinaci:

```
    _r1 VUTBR:NICKNAME "xburge05" .
    _r1 VUTBR:EMAIL <mailto:burger@fit.vutbr.cz> .
    _r1 VUTBR:HOMEPAGE <http://www.burger.cz> .
```

Pokud tedy relace má 3 domény/sloupce, vygenerujeme pro každý záznam 4 RDF statementy.
Takto převedeme všechny relace na RDF statementy, které dohromady zformují naše RDF data.
Protože operace v relační algebře produkují z relací zase relace, budeme muset pro naše SPARQL dotazy používat formát výstupu definovaný pomoci CONSTRUCT, které vyrobí další RDF statementy, odpovídající nové relaci.
SELECT se vyrobí snadno: sestavíme vyhledávací vzor z celé relace, aplikujeme filtr na patřičná pole a výsledek uložíme do nových statementů svázaných s novou relací. Tedy náš příklad převedeme na následující SPARQL dotaz:

```
    CONSTRUCT {
      _x CODD:RELATION VUTBR:STUDENTS .
      _x VUTBR:NICKNAME ?nickname .
      _x VUTBR:EMAIL ?email .
      _x VUTBR:HOMEPAGE ?homepage }
    WHERE {
      ?y CODD:RELATION VUTBR:FIT .
      ?y VUTBR:NICKNAME ?nickname .
           FILTER regex(str(?nickname), "^x") .
      ?y VUTBR:EMAIL ?email .
      ?y VUTBR:HOMEPAGE ?homepage }
```

PROJECT je ještě poněkud jednodušší, protože můžeme ignorovat všechny ostatní položky kromě těch, které nás zajímají:

```
    CONSTRUCT {
      _x CODD:RELATION VUTBR:EMAILS .
      _x VUTBR:NICKNAME ?nickname .
      _x VUTBR:EMAIL ?email }
    WHERE {
      ?y CODD:RELATION VUTBR:FIT .
      ?y VUTBR:NICKNAME ?nickname .
      ?y VUTBR:EMAIL ?email }
```

Nejzajímavější bude konstrukce SPARQL dotazu pro JOIN – jedna sada vyhledávacích vzorů bude prohledávat první relaci, druhá sada druhou relace a obě sady budou sdílet proměnné, na kterých se mají shodnout:

```
CONSTRUCT {
  _x CODD:RELATION VUTBR:PHDTUTORS .
  _x VUTBR:NICKNAME ?nickname .
  _x VUTBR:EMAIL ?email .
  _x VUTBR:TUTOR ?tutor }
WHERE {
  ?y CODD:RELATION VUTBR:EMAILS .
  ?y VUTBR:NICKNAME ?nickname .
  ?y VUTBR:EMAIL ?email .
  ?z CODD:RELATION VUTBR:PHD .
  ?z VUTBR:NICKNAME ?nickame .
  ?Z VUTBR:TUTOR ?tutor }
```

Všimněme si, že přestože se jedná o tři elementární – a tudíž ortogonální – operace relační algebry, jejich protějšky ve SPARQL jsou si velmi podobné a vlastně se jedná pořád o tutéž relaci. Je to způsobeno tím, že RDF je mnohem elementárnější datové úložiště než relační algebra a elementární datový fragment (RDF statement) je mnohem „menší“ než v relační algebře (jedna řádka v relaci/tabulce).

Nejvážnější omezení ve SPARQL proti tradičnímu SQL je absence agregačních funkcí – možnost zjistit počet nebo součet hodnot v určité proměnné přes všechny RDF statementy, které vyhovují vyhledávacímu vzoru.

Secondi piati (druhé kolo)

Objektově orientovaný model se od relačního liší v mnoha zásadních ohledech. Zde jsou aspoň ty klíčové:

- Základní jednotkou uloženou v databázi je objekt.
- Objekt má vlastní identitu, atributy a relace.
- Atributy mohou mít jednu hodnotu nebo kolekci hodnot – hodnotou atributu je literál v RDF terminologii.
- Relace jsou vztahy mezi objekty v databázi, jsou obousměrné a mohou mít kardinalitu 1:1, 1:N nebo M:N.
- Objekt je nezávislý na hodnotách svých atributů i na relacích a je určen výhradně svou identitou.
- Objekty jsou organizovány do tříd.
- Každý objekt ví, do kterých tříd patří – ve standardním objektově orientovaném modelu s jednoduchou dědičností existuje vždy jedna třída, do které objekt patří a která je nejmenší taková, je to tzv. vlastní třída objektu.
- Některé třídy představují pojmenované kolekce objektů – tzv. extenty. Extent obsahuje všechny objekty dané třídy a jejích podtříd. Extent v objektově orientovaných dotazovacích jazycích nahrazuje tabulku.

Na dotazování do objektově-orientovaných databází se používá dotazovací jazyk OQL. Jedná se o standard definovaný normou ODMG. V porovnání s SQL, tak jak ho prakticky známe, jazyk OQL je plně rekurzivní, tj. má omezenou množinu konstrukcí, které na vstupu mají kolekci objektů a na výstupu opět kolekci objektů, kterou lze opět použít jako vstup pro další „vrstvu“ v dotazu.

Popišme si nejdříve zase nějaký příklad, který budeme používat. Příklad je napsán v jazyce ODL („Object Definition Language“), který je ve standardu ODMG určen pro popis tříd.

```
class Person (extent persóna) {
  attribute string name;
};

class Student extends Person {
  attribute string email;
  relationship Teacher tutor inverse Teacher::teaches;
```

```
    }

    class Teacher extends Person {
      relationship set<Student> teaches inverse Student::tutor;
    }
```

Teď tuto definici převedeme na SODA RDF schéma podle Jakuba Güttnera – už to je první významný rozdíl proti relačním databázím: pro relační databáze jsme žádné RDF schéma neměli a vlastně všechny statementy byly stejně platné. Pro ODMG sestavíme RDF schéma, takže RDF statementy o instancích tříd (tj. o objektech) už budou nějakému schématu podřízené.
Zavedeme si tři nové prefixy, jeden pro samotné Güttnerovo schéma, jeden pro naše vlastní třídy a vlastní schéma a jeden pro vlastní data.

```
    @prefix soda: http://www.fit.vutbr.cz/~guttner/soda .
    @prefix vb: http://www.vutbr.cz/#classes .
    @prefix vd: http://www.vutbr.cz/#data .
```

Pro jednotlivé třídy a jejich definice pak dostaneme následující RDF schéma:

```
    vb:Person rdf:type soda:Class .
    vb:Student soda:subTypeOf vb:Person .
    vb:Teacher soda:subTypeOf vb:Person .

    vb:name    rdf:subclassOf soda:Attribute;
               rdf:domain vb:Person;
               rdf:range xsd:String .
    vb:email   rdf:subclassOf soda:Attribute;
               rdf:domain vb:Student;
               rdf:range xsd:String .
    vb:tutor   rdf:subclassOf soda:Attribute;
               rdf:domain vb:Student;
               rdf:range vb:Teacher .

    vb:Students rdf:type soda:Collection;
               soda:collectionOf vb:Student .
    vb:teaches rdf:subclassOf soda:Atribute;
               vb:teaches rdf:domain vb:Teacher;
               rdf:range vb:Students .
```

Vzorek dat si můžeme naformulovat v dalším formátu, který je součástí ODMG, a to OIF („object interchange format“), což je vlastně popis, jak objekty, jejichž třídy jsou popsané pomocí ODL, exportovat do textových souborů a stěhovat je z jedné databáze do druhé. Poznatek, že export dat a dotazování nelze obsloužit jedním jazykem, jako je to v případě SQL, je významný a zajímavý rozdíl.
OIF definice dat, která jsou podobná našemu SQL příkladu, budiž následující:

```
    xburge05       Student{ name "Tomas Burger", email "burger@fit.vutbr.cz",
tutor hruska }
    hruska         Teacher{ name "Tomas Hruska", teaches { xburge05, masarik }}
    masarik        Student{ name "Karel Masarik", email "masarik@fit.vutbr.cz",
tutor hruska }
    lukas          Student{ name "Roman Lukas", email "lukas@fit.vutbr.cz",
tutor meduna }
    meduna         Teacher{ name "Alexandr Meduna", teaches { lukas }}
```

Tato data pak podle receptu Jakuba Güttnera převedeme na RDF statementy, přičemž využijeme jak schéma SODA, tak schéma VB. Prefix VD pak využijeme pro identity objektů. OIF sdílí s RDF jednu pozitivní vlastnost: umožňuje, aby identifikátory objektů měly jakýsi sémantický význam. Pokud by OIF soubor byl export z nějaké objektově-orientované databáze, pravděpodobně by místo srozumitelných identifikátorů objektů obsahoval celá čísla. Ale pro RDF by to nebyl problém, URI nemá žádný předepsaný sémantický vztah ke skutečnosti, URI musí pouze být jednoznačné.

```
    vd:xburge05    soda:type vb:Student;
                   vb:name "Tomas Burger";
                   vb:email "burger@fit.vutbr.cz";
                   vb:tutor vd:hruska .
```

```
    vd:hruska      soda:type vb:Teacher;
                   vb:name "Tomas Hruska";
                   vb:teaches [ vd:xburge05; vd:masarik ] .
    vd:masarik     soda:type vb:Student;
                   vb:name "Karel Masarik";
                   vb:email "masarik@fit.vutbr.cz";
                   vb:tutor vd:hruska .
    vd:lukas       soda:type vb:Student;
                   vb:name "Roman Lukas";
                   vb:email "lukas@fit.vutbr.cz";
                   vb:tutor vd:meduna .
    vd:meduna      soda:type vb:Teacher;
                   vb:name "Alexandr Meduna";
                   vb:teaches [ vd:lukas ] .
```

Jenom si připomeňme, že zápis:

```
    vd:hruska vb:teaches [ vd:xburge05; vd:masarik ] .
```

je jenom syntaktickou zkratkou pro zápis kolekce, propojené přes blank node. Plný zápis by tedy vypadal takto:

```
    vd:hruska vb:teaches _hruska_students .
    _hruska_students rdf:_1 vd:xburge05 .
    _hruska_students rdf:_2 vd:masarik .
```

Tak teď máme připravena veškerá data, schémata a modely a můžeme začít studovat dotazování. Základní myšlenkou v OQL je fakt, že jazyk je uzavřený a plně rekurzivní: na vstupu je kolekce objektů a na výstupu je opěk kolekce objektů, případně objekt jediný, který lze ve vhodný okamžik nazřít jako kolekci o jednom prvku. V tomto je OQL podobné relační algebře a nepodobné SQL. Takže pokud objektově-orientované schéma již obsahuje nějakou kolekci přímo, můžeme ji přímo zadat do dotazu, takže například dotazy

```
    persons
```

nebo

```
    hruska.teaches
```

jsou platné OQL dotazy, které vrátí kolekci objektů. Nás ale budou více zajímat tradiční dotazy tvaru

```
    select <selektory> from <kolekce> where <podmínka>
```

i když OQL je jazyk poměrně bohatý na různé konstrukce a tato klasická je jen „jedna z mnoha“. Tady například dotaz:

```
    select (Student)x.email from persons
```

vrátí emaily studentů v databázi (casting má „filtrační“ účin), zatímco

```
    select distinct p.name from (select (Student)x from persons) s, s.tutor p
```

vrátí jména jejich učitelů. Samozřejmě lze omezit výstup i nějakou dodatečnou podmínkou, třeba dotaz

```
    select s.email from persons p, (Teacher)p.teaches s where p.name="Tomas
Hruska"
```

vrátí emaily studentů profesora Hrušky. Přestože OQL postrádá “background” relační algebry, který by vypíchl z praktického nástroje klíčové atomární operace (hle, toť výborný podnět pro primární výzkum), lze některé základní kroky ve zpracování dotazu vypozorovat – všechno to jsou vlastně operace nad kolekcí:

A. vytvoření kolekce z extentu nebo z relace pojmenovaného nebo zpracovávaného objektu
B. filtrování kolekce podle hodnoty nějakého atributu
C. filtrování kolekce přetypováním na předepsanou třídu
D. převod kolekce objektů na kolekci hodnot některého jeho atributu

Dotaz

```
    select s.email from persons p, (Teacher)p.teaches s where p.name="Tomas
Hruska"
```

by se pak dal rozložit na následující kroky:

1. vytvoř kolekci z extentu persons (A)

2. filtruj kolekci podle hodnoty atributu name (B)
3. filtruj kolekci přetypováním na třídu Teacher (C)
4. vytvoř kolekci z relace teaches zpracovávaných objektů (A)
5. převeď kolekci studentů na kolekci jejich emailů (D)

Jsme tyto elementární operace schopni převést na SPARQL?

Předně SPARQL neví nic o nadtřídách, podtřídách či dědičnosti, a to přesto, že RDF koncept podtřídy zná a naše SODA schéma říká, že soda:subtypeOf je vztah, z něhož plyne automaticky i vztah podtřídy. To je však informace určená inteligentním agentům a ne dotazovacímu jazyku. Abychom ale s prostým dotazovacím jazykem dosáhli požadovaných výsledků, musíme doplnit do našich dat minimálně informaci o dědičnosti, kterou dotazovací jazyk není schopen dovodit sám. Neznamená to, že by OQL, který dědičnosti rozumí, byl nějak inteligentnější, protože ten rozumí jen tomuto jedinému vztahu, zatímco SPARQL nedělá rozdíl mezi dědičností a jinými relacemi mezi třídami a objekty. Nejprve je třeba tedy data obohatit o nadtřídový uzávěr. Což v našem případě znamená přidat pro každý objekt RDF statement, který z něj učiní objekt z třídy Person:

```
vd:xburge05    soda:type vb:Person .
vd:hruska      soda:type vb:Person .
vd:masarik     soda:type vb:Person .
vd:lukas       soda:type vb:Person .
vd:meduna      soda:type vb:Person .
```

Další krok je identifikovat v RDF datech kolekce: na rozdíl od analogie k SQL, kde relace tvořila vždy specifický RDF graf, v případě OQL můžeme za kolekci považovat sekvenci URI, která ukazují na nějaký objekt. Finální kolekce, ze které už nelze pokračovat, může obsahovat nejen URI, ale i literály. Zkusme tedy zrekonstruovat pět kroků našeho příkladu jako SPARQL dotazy. Nejprve sestavíme z extentu persons kolekci identifikátorů všech objektů, které patří do třídy Person:

```
SELECT ?person WHERE { ?person soda:type vb:Person . }
```

Snadné. I filtr přes hodnotu name je snadný:

```
SELECT ?person
WHERE {
  ?person soda:type vb:Person .
  ?person vb:name ?name .
  FILTER ?name = "Tomas Hruska" . }
```

V dalším kroku přidáme filtr přetypováním. RDF není přísné v typech, místo přetypování můžeme jen dále filtrovat a udělat vzor pro požadovaný typ:

```
SELECT ?person
WHERE {
  ?person soda:type vb:Person .
  ?person vb:name ?name .
  FILTER ?name = "Tomas Hruska" .
  ?person soda:type vb:Teacher . }
```

Teď dokonce můžeme vynechat dotaz na typ vb:Person, protože víme, že typ Teacher je podtypem typu Person a tudíž můžeme garantovat, že objekt, který patří do třídy Teacher, patří i do třídy Person, má atribut name atd. To je informace, kterou víme jako myslící lidé, kterou by měl být schopen vyprodukovat inteligentní agent, ale která zůstane utajena SPARQL procesoru. Ale pro přehlednost si dotaz zredukujme a kontrolu na typ Person vynechme:

```
SELECT ?person
WHERE {
  ?person soda:type vb:Teacher .
  ?person vb:name ?name .
  FILTER ?name = "Tomas Hruska" . }
```

V dalším kroku se přesuneme z kolekce učitelů do kolekce jejich studentů:

```
SELECT ?student
WHERE {
  ?person soda:type vb:Teacher .
  ?person vb:name ?name .
  FILTER ?name = "Tomas Hruska" .
  ?person vb:teaches ?student . }
```

Všimněme si, že SPARQL je „beze směru“, že stejně dobře by fungoval vyhledávací vzor „?student vb:tutor ?person“, ale pravděpodobně by tato varianta byla asi méně efektivní pro případnou optimalizaci. Pokud by ovšem RDF schéma neobsahovalo informaci o inverzním vztahu predikátů „tutor“ a „teaches“ a optimalizátor by tuto informaci nevyužil. – V tom případě by dotazy byly zcela ekvivalentní za předpokladu, že predikáty by v datech byly skutečně konzistentně inverzní, což je zaručeno u objektově-orientované databáze, nikoliv však u obecných RDF dat.
V posledním kroku převedeme kolekci studentů na kolekci jejich emailů:

```
SELECT ?email
WHERE {
  ?person soda:type vb:Teacher .
  ?person vb:name ?name .
  FILTER ?name = "Tomas Hruska" .
  ?person vb:teaches ?student .
  ?student vb:email ?email . }
```

A je to! Zdá se tedy, že s troškou úsilí lze převést standardní OQL dotaz na SPARQL dotaz.
Jaké je poučení: stejně jako u relační algebry se různé jevy v OQL redukovaly na stejný jev ve SPARQL – vyhledávací vzor. Na rozdíl od relační algebry jsme si v případě OQL vystačili s konstruktem SELECT, který vždycky obsahoval jednu proměnnou, obsahující kolekci po posledním kroku. A jednotlivé kroky provádění dotazu se hromadily ve WHERE sekci dotazu.
Podotkněme, že OQL umí produkovat jako výsledek i struktury, což by v případě SPARQL vedlo na více než jednu proměnnou v sekci SELECT.
Hlavní nevýhoda SPARQL tak opět spočívá především v absenci agregačních funkcí, a to nejen v elementární podobě jako součet nebo počet všech prvků, ale také v sofistikovanější podobě s využitím GROUP BY a HAVING, tak jak to umí jak SQL tak OQL.
Kromě toho uzavření predikátu soda:type na nadtřídy představuje problém pro velké systémy a pro praktické použití by bylo lepší udělat doplnění „virtuálně“, nebo-li ve skutečnosti naučit SPARQL-procesor skutečně provést úvahu o nadtřídách a podtřídách, tak jak ji samo provádí OQL.

Dolce (závěrem)

Závěrem dvě úvahy.
Zaprvé: jak OQL tak SQL obsahuje některé složité konstrukce, jak pracovat s relativně složitými objekty (alespoň v porovnání s RDF). Pokud se tyto složité objekty podaří převést (rozuměj rozložit) na RDF statementy, tak pak i OQL a SQL dotazy lze převést na SPARQL dotazy - rozuměj opět rozložit. Výsledkem převodu libovolné konstrukce je totiž zase a pouze jen sekvence vyhledávacích vzorů. Dokonce ani nebylo potřeba užít některých složitějších konstrukcí, které SPARQL nabízí. Všechny složité konstrukce OQL a SQL jsme převedli na velmi základní a elementární operace ve SPARQL.
Dává to jistou naději, že vyjadřovací možnosti při použití sofistikovanějších SPARQL konstrukcí se mohou dostat hluboko za možnosti OQL či SQL.
Je to pravděpodobně způsobeno tím, že SPARQL se může odpoutat od přirozených vazeb v elementárních datech, jako je relace (tj. sounáležitost políček v jedné řádce relace), nebo jako je objekt (tj. sounáležitost vazeb, atributů a objektové identity). Konec konců blank nody (tj. jevy bez identity) nebo třídy definované množinovými operacemi nebo pomocí výskytu atributu (jak je to povoleno specifikací RDF schématu) jdou přesně za tyto hranice. Na druhou stranu, pokud se soustředíme na konzistenci dat a na objektově-orientované principy zapouzdření, nejsou tyto možnosti něco, co by nás mělo těšit, ale spíš něco, před čím bychom měli být na pozoru.
Za druhé: právě pokud hovoříme o zapouzdřenosti, SPARQL obsahuje jeden inspirativní moment (nebudu zastírat, že tento odstavec byl motivací k sepsání celého předešlého textu) – konstrukt DESCRIBE. Vedle konstruktů SELECT a CONSTRUCT konstrukt DESCRIBE přijme URI a vrátí RDF graf, který nám cosi o tomto URI říká – a je věcí databáze, aplikační logiky, objektu samého, našeho kontextu a naší autorizace, co vlastně DESCRIBE vrátí. Například můžeme zavolat:
DESCRIBE vd:xburge05 .
a SPARQL procesor může odpovědět

```
    vd:xburge05    soda:type vb:Student;
                   vb:name "Tomas Burger";
                   vb:email "burger@fit.vutbr.cz" .
```

pokud jsme "vítaný host" a máme dost autorizací a SPARQL procesor nazná, že můžeme vědět víc, může dokonce odpověď vypadat až takto:

```
    vd:xburge05    soda:type vb:Student;
                   vb:name "Tomas Burger";
                   vb:email "burger@fit.vutbr.cz";
                   vb:tutor vd:hruska .
    vd:hruska      soda:type vb:Teacher;
                   vb:name "Tomas Hruska" .
```

což je vlastně zapouzdření a objektově-orientovaná individualita par excelence. Představme si, že extent vrátí jen kolekci URI (což je vlastně pravda i dnes) a my tuto kolekci obohatíme o DESCRIBE pro každé takové URI, sjednocením vytvoříme RDF graf a na něm provedeme dotaz – výsledkem bude kolekce URI, kterou opět rozšíříme pomocí DESCRIBE a získané stamenty opět přidáme mezi RDF data. Nakonec tak pravděpodobně získáme mnohem víc dat než na počátku, na druhou stranu, pokud se kvůli scházející autorizaci nebo z jiných důvodů nemáme dozvědět, že určitý objekt není studentem, ačkoliv jím ve skutečnosti je, výsledek DESCRIBE pro tento objekt prostě tuto informaci neprozradí.

Reference

Následuje seznam odkazů pro další čtení:

URI	http://www.isi.edu/in-notes/rfc2396.txt
ODMG	http://www.odmg.org/
RDF	http://www.w3.org/TR/rdf-primer/
N3	http://www.w3.org/2000/10/swap/Primer.html
RDFS	http://www.w3.org/TR/rdf-schema/
Dublin core	http://dublincore.org/documents/dces/
Güttner	http://www.fit.vutbr.cz/research/view_pub.php?id=7609
FOAF	http://www.foaf-project.org/
XML Schema	http://www.w3.org/TR/xmlschema-2/
SPARQL	http://www.w3.org/TR/rdf-sparql-query/
Jena - RDQL	http://jena.sourceforge.net/tutorial/RDQL/index.html
Relační algebra	http://www.rsu.edu/faculty/johnnycarroll/cs3223/fa2002/Lectures/special_lecture_index.htm