Processamento de Consultas SPARQL em uma Base Relacional de Entidades

João G. Pauluk, Mariana M. Garcez Duarte, Rafael L. Prado e Carmem S. Hara

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
{jgpauluk, marianamgd, rafaellimaprado426}@gmail.com, carmem@inf.ufpr.br

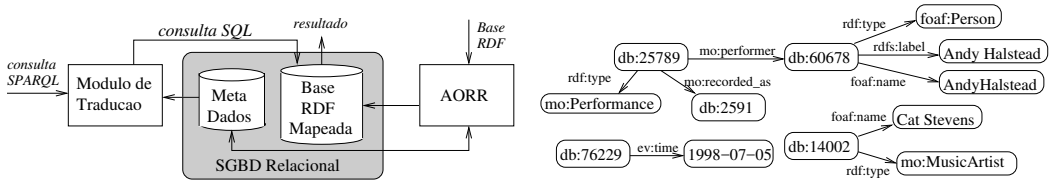
Abstract. *The huge volume of existing RDF datasets requires SPARQL queries to be efficiently processed. One approach to achieve this goal is to store RDF on a group-by-entity relational database, which explores structural similarity to group sets of triples in a single line of a relation. In this paper, we propose a method for translating SPARQL queries to SQL to be processed on such a database. Our experiments showed that the execution time of the translated queries are in average 250% lower, compared to queries on a triples relation.*

Resumo. *A grande quantidade de dados de RDF existente nos dias de hoje requer que as consultas SPARQL sejam processadas de forma eficiente. Uma possível abordagem para atingir tal objetivo é o armazenamento dos dados em uma base relacional de entidades, que explora a similaridade das estruturas para a horizontalização das triplas. Neste artigo, é proposto um método para a tradução de consultas SPARQL para SQL para ser processada sobre uma base relacional de entidades. Os experimentos realizados mostram que as consultas traduzidas e executadas sobre esta base obtiveram um ganho de desempenho de aproximadamente 250% em relação à consulta sobre uma tabela de triplas.*

1. Introdução

A *Web* semântica é uma extensão da *World Wide Web*, que promove a visão de dados interconectados e interpretáveis pelas máquinas. O W3C definiu o RDF como seu modelo de dados padrão e SPARQL como sua linguagem de consulta. Uma base RDF é composta por triplas SPO (sujeito, predicado e objeto), que pode ser representada na forma de um grafo, uma vez que o objeto de uma tripla pode ser o sujeito de outra. Uma consulta SPARQL é composta por padrões de triplas, que definem padrões de subgrafos a serem pesquisados na base. A grande quantidade de dados RDF existente requer que as consultas SPARQL sejam processadas de forma eficiente. Existem vários métodos para atingir tal objetivo [Aluç et al. 2014], sendo um deles o mapeamento dos dados RDF para o modelo relacional e a conversão das consultas SPARQL para SQL. Assim, é possível tirar proveito das otimizações que um SGBDR oferece ao utilizar a linguagem de consulta SQL. Tal método foi utilizado pelo Sistema de Armazenamento Otimizado de Dados RDF em SGBDR (AORR) [Prado et al. 2018].

O AORR explora a similaridade de estrutura dos dados para a horizontalização das triplas que compõem a base RDF. Esta estratégia de armazenamento é denominada neste trabalho de base relacional de entidades e visa melhorar o desempenho de consultas no formato estrela e flocos de neve [Aluç et al. 2014] através da diminuição do número de auto-junções necessárias para processá-las. Em [Prado et al. 2018] é proposto o algoritmo de extração de estrutura da base RDF, juntamente com os dados e metadados gerados. No



(a) Arquitetura do sistema (b) Grafo RDF

Figura 1. Arquitetura do sistema e Grafo RDF

entanto, o processo de tradução de consultas não é detalhado. Este artigo preenche esta lacuna, apresentando um algoritmo de tradução de consultas. A Figura 1a apresenta os componentes do sistema. O módulo de tradução de consultas é responsável por receber uma consulta SPARQL e traduzi-la para SQL baseado nos metadados de mapeamento, a fim de ser processada sobre a base relacional de entidades gerada pelo sistema AORR.

O restante do artigo está estruturado da seguinte forma. A Seção 2 apresenta trabalhos relacionados. O AORR é apresentado na Seção 3 e a Seção 4 detalha o algoritmo de tradução. A Seção 5 descreve a implementação, bem como uma análise experimental. A Seção 6 conclui o artigo enumerando alguns trabalhos futuros.

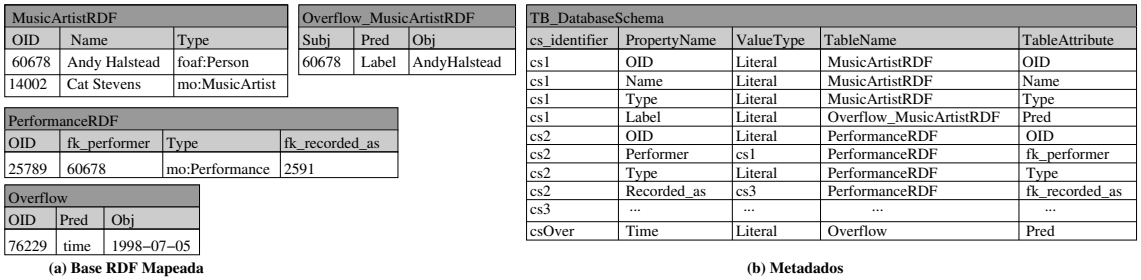
2. Trabalhos Relacionados

Existem diversas propostas para a tradução de consultas SPARQL para SQL. A proposta de [Chebotko et al. 2009], propõe um algoritmo genérico de tradução, na qual é criada uma subconsulta SQL para cada padrão de tripla da consulta SPARQL. Esta subconsulta é gerada a partir das funções α e β, que associam um padrão de tripla à relação na qual ela está armazenada no SGBD relacional e ao atributo, respectivamente. Estas subconsultas formam uma única consulta SQL com a utilização de operações como *inner join* para uma sequência de padrões ou *left outer join* para padrões opcionais. Outros trabalhos, tais como [Michel et al. 2016] e [Rodriguez-Muro et al. 2012] consideram que a base RDF foi gerada a partir de uma especificação R2RML [Das et al. 2012], que determina como uma base relacional foi mapeada para RDF. O processo de tradução utiliza esta informação para traduzir consultas SPARQL para SQL sobre a base relacional original. A proposta de tradução de consultas em [Chaloupka and Necasky 2016] é realizada a partir de um mapeamento entre os modelos definido pelo usuário.

O processo de tradução apresentado neste artigo é inspirado na proposta de [Chebotko et al. 2009]. Como [Chebotko et al. 2009], o artigo possui a ideia de subconsultas aninhadas de acordo com os tipos de junção entre cada padrão de tripla. No entanto, ao contrário das funções genéricas α e β, as funções que associam os padrões de triplas a tabelas e atributos são definidas a partir das tabelas de metadados gerados a partir do processo de extração de esquema do sistema AORR [Prado et al. 2018]. O AORR gera uma base relacional baseada em entidades, além de tabelas de overflow que armazenam triplas que não se adequam às estruturas das entidades.

3. Base Relacional de Entidades Gerada pelo AORR

O Sistema AORR [Prado et al. 2018] possui um módulo de extração de estrutura de uma base RDF, que tem por objetivo identificar tipos de entidades, ou seja, conjuntos de sujeitos que possuem estruturas similares. Para cada entidade é criada uma tabela com seus



MusicArtistRDF

OID	Name	Type
60678	Andy Halstead	foaf:Person
14002	Cat Stevens	mo:MusicArtist

Overflow_MusicArtistRDF

Subj	Pred	Obj
60678	Label	AndyHalstead

PerformanceRDF

OID	fk_performer	Type	fk_recorded_as
25789	60678	mo:Performance	2591

Overflow

OID	Pred	Obj
76229	time	1998–07–05

(a) Base RDF Mapeada

TB_DatabaseSchema

cs_identifier	PropertyName	ValueType	TableName	TableAttribute
cs1	OID	Literal	MusicArtistRDF	OID
cs1	Name	Literal	MusicArtistRDF	Name
cs1	Type	Literal	MusicArtistRDF	Type
cs1	Label	Literal	Overflow_MusicArtistRDF	Pred
cs2	OID	Literal	PerformanceRDF	OID
cs2	Performer	cs1	PerformanceRDF	fk_performer
cs2	Type	Literal	PerformanceRDF	Type
cs2	Recorded_as	cs3	PerformanceRDF	fk_recorded_as
cs3	...	...	...	...
csOver	Time	Literal	Overflow	Pred

(b) Metadados

Figura 2. Base relacional gerada pelo AORR

predicados. Dada a natureza não estruturada das bases RDF, algumas triplas não se adequam ao esquema relacional extraído. Assim, o AORR cria um conjunto de tabelas de triplas (SPO) para armazenar tais informações. Estas tabelas são chamadas de *overflow*. As tabelas de overflow de entidades armazenam predicados infrequentes de sujeitos armazenados na tabela de entidade correspondente, como por exemplo, `MusicArtistRDF` e `Overflow_MusicArtistRDF`. Além disso, há sujeitos que não pertencem a nenhum tipo de entidade. Eles são armazenados na tabela *Overflow*. A Figura 2a apresenta a base relacional gerada pelo AORR a partir da base RDF ilustrada na Figura 1b.

Para permitir a tradução de consultas SPARQL para SQL sobre a base gerada, o AORR armazena informações sobre o mapeamento em tabelas de metadados. As principais são : `TB_Subj_OID`, `TB_FullPredicate` e `TB_DatabaseSchema`. A tabela `TB_Subj_OID` associa IRIs a identificadores (OID) numéricos. A tabela `TB_FullPredicate` associa IRIs de predicados ao nome de uma coluna em uma tabela de entidade ou a um label utilizado como valor da coluna predicado nas tabelas de overflow. Por exemplo, a IRI `http://purl.org/ontology/mo/recorded_as` é associada ao label `fk_recorded_as` da tabela `PerfomanceRDF`. Já a tabela `TB_DatabaseSchema` traz informações sobre o mapeamento, como mostra a Figura 2b. Cada tipo de entidade é identificado por um `cs_identifier`, que possui um predicado `PropertyName` com um valor do tipo `ValueType`, que pode ser um literal ou uma IRI que pertence a uma outra entidade. O predicado é armazenado na tabela `TableName` na coluna `TableAttribute`. Observe que uma entidade pode ter predicados armazenados na tabela de entidade como no seu overflow (como é o caso de `cs1` no Exemplo da Figura 2b). Neste artigo considera-se que cada predicado encontra-se ou na tabela de entidade ou no seu overflow, mas não em ambos. Além disso, todos os predicados em sujeitos na tabela `Overflow` possuem o mesmo valor `csOver` para o atributo `cs_identifier` da tabela `TB_DatabaseSchema`.

4. Tradução de Consultas SPARQL para SQL

Este artigo considera consultas formadas por padrões de grafos básicos (BGP - *Basic Graph Patterns*), nas quais os padrões de triplas são da forma *(variável, IRI, variável)*. O Algoritmo 1 apresenta como é realizada a tradução de uma consulta SPARQL Q para SQL. Primeiro, cada variável no resultado é associada a um rótulo (Linha 2) e é realizada uma busca dos identificadores de cada IRI de predicado na tabela *TB_FullPredicate* (Linha 3). Os demais passos são detalhados a seguir.

Procura por padrões estrela (Linhas 5-8). Neste passo os padrões de tripla em Q que possuem o mesmo sujeito são agrupados, ou seja, são identificados os padrões estrela na

Algoritmo 1: Algoritmo de Tradução

Entrada: Consulta SPARQL Q = select $?x_1...?x_n$ where PT
Saída: Consulta SQL Q'

```
1  início
2      Associa cada variável x_1, ..., x_n a rótulos name(x_1), ..., name(x_n);
3      Busca em TB_FullPredicate os IDs das IRIs de predicados em PT;
4      Seja V = {v_1, ...v_p} o conjunto de sujeitos (variáveis) em PT;
5      para cada variável v ∈ V faça
6          entidades(v) := busca em TB_DatabaseSchema entidades cs que
7                          possuem todos os predicados de v em PT;
8      fim
9      para cada variável v ∈ V faça
10         filtra( entidades(v) );
11     fim
12     condJuncao := extraiCondJuncao(entidades(v_1), ..., entidades(v_p));
13     para cada variável v faça
14         para cada entidade cs em entidade(v) faça
15             sql(v, cs) := geraSubConsulta(v, cs);
16         fim
17         sql(v) := sql(v, cs_1) UNION ...UNION sql(v, cs_j);
18     fim
19     Q':= SELECT name(x_1), ... name(x_n)
20           FROM sql(v_1), ...sql(v_p) WHERE condJuncao;
21 fim
```

consulta. A partir deles, é realizada uma busca na tabela `TB_DatabaseSchema` para determinar quais entidades possuem todos os predicados existentes em cada padrão estrela. Considere por exemplo, a consulta SPARQL na Figura 3a. O sujeito $?a$ é associado à entidade com identificador $cs1$ e o sujeito $?b$ à entidade $cs2$. Além de restringir o processo de tradução às entidades $cs1$ e $cs2$, a tabela `TB_DatabaseSchema`, possui ainda as informações da tabela, atributo e tipo associado a cada predicado.

Filtragem por ligações (Linhas 9-12). Os tipos dos predicados são utilizados para filtrar ligações sujeito-objeto entre diferentes padrões estrela. Para ilustrar, considere o padrão de tripla *?b performer ?a* da Figura 3a . Como $?a$ é sujeito de outras triplas, há uma ligação sujeito-objeto. Assim, $cs1$, entidade relacionada à $?a$ deve corresponder ao tipo do predicado *fk_performer* de $cs2$. Com a verificação dos tipos dos objetos, é possível filtrar as entidades associadas a cada sujeito. O mesmo tipo de filtragem é utilizado para ligações objeto-objeto. Estas ligações dão origem às condições de junção entre padrões estrela (Linha 24 da Figura 3b).

Montagem do comando SQL (Linhas 13-20). Para cada variável sujeito é gerada uma subconsulta SQL. Essas subconsultas são inseridas na cláusula FROM da consulta final (Linha 20). Quando houver mais de uma entidade associada a uma mesma variável, as subconsultas geradas para cada variável são colocada em uma única expressão com a operação de UNION (Linha 17). A geração da subconsulta para cada entidade relacio-

```
SELECT ?n ?t ?l ?b
WHERE {
      ?a name ?n .
      ?a type ?t .
      ?a label ?l .
      ?b performer ?a .
}
```

(a) Consulta SPARQL

```
1 SELECT a.name248558 AS n,
2          a.type8HG5ET AS t,
3          a.labelYGC7VX AS l,
4          b.Subj AS b
5 FROM
6 (SELECT a1.OID,
7          a1.Name AS name248558,
8          a1.Type AS type8HG5ET,
9          o1.Obj AS labelYGC7VX
10 FROM MusicArtistRDF a1,
11      Overflow_MusicArtistRDF o1
12 WHERE a1.Name IS NOT NULL
13       AND a1.Type IS NOT NULLL
14       AND o1.Pred = 'label'
15       AND o1.Obj IS NOT NULL
16       AND a1.OID = o1.OID) a,
17 (SELECT b1.OID,
18          s.Subj,
19          b1.fk_performer AS fk_performer0SZFR6,
20 FROM PerformanceRDF b1,
21      TB_Subj_OID s,
22 WHERE b1.fk_performer IS NOT NULL
23       AND b1.OID = s.OID) b
24 WHERE a.OID = b.fk_performer0SZFR6
```

(b) Consulta SQL

Figura 3. Consulta SPARQL traduzida em consulta SQL

nada a uma variável tem as seguintes linhas gerais: **(a)** para cada entidade: a consulta utiliza a tabela de entidade correspondente e possivelmente múltiplas vezes sua tabela de overflow específico, que são combinadas pela operação de junção sobre o atributo OID. Por exemplo, para a variável $?a$, é gerada a junção das tabelas `MusicArtistRDF` e `Overflow_MusicArtistRDF` (Linhas 6-16 da Figura 3b); **(b)** para o Overflow: são realizadas múltiplas auto-junções da tabela `Overflow` sobre o atributo OID, uma vez para cada predicado da entidade *CSover* na tabela `TB_DatabaseSchema`; **(c)** caso a própria variável sujeito estiver no resultado é necessário fazer uma junção com a tabela `TB_Subj_OID` para obter a IRI relacionada ao OID (Linhas 17-23 da Figura 3b).

5. Análise Experimental

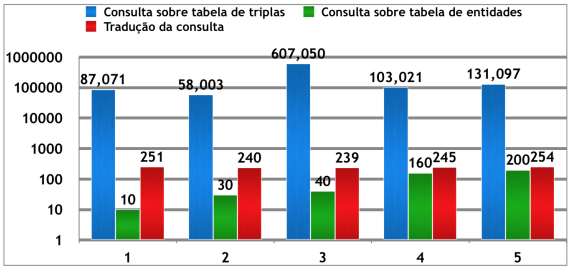
O algoritmo de tradução foi implementado utilizando a linguagem Python, a ferramenta de *parsing* ANTLR[1] e o SGBD MySQL. Nesta seção são relatados experimentos realizados para determinar o tempo de tradução das consultas e comparar o tempo de execução das consultas traduzidas com o tempo de execução sobre uma tabela de triplas SPO. O computador utilizado para executar os experimentos foi um MacOSX Intel Core m3 1.1 GHz com 8 GB de memória RAM. O banco de dados utilizado possui 26 tabelas, que foram geradas com o sistema AORR, a partir da base RDF Peel[2].

Os testes executados consistiram de 5 consultas. Os testes realizados foram baseados em três padrões estrelas presentes na base Peel, como detalhado na Figura 4b. As consultas contém os seguintes padrões: Consulta 1: PE1; Consulta 2: PE2; Consulta 3: PE3; Consulta 4: PE1 e PE2; Consulta 5: PE1, PE2 e PE3[3]. A Base SPO tem tamanho de 44.58 MB, enquanto a base estruturada pelo AORR tem 21.84 MB. Foram criados índices na tabela SPO sobre o predicado e sujeito, enquanto as tabelas de entidades tem índices apenas sobre o atributo OID. O impacto da criação dos índices foi apresentada anteriormente em [Duarte and Hara 2018], que também explorou uma forma alternativa de tradução de consultas baseada em visões. O tempo de processamento das consultas, bem como o tempo de tradução, reportados em milisegundos, estão ilustrados na Figura 4a. Devido a diferença nos valores, foi utilizada uma escala logarítmica. É possível perceber que o tempo de tradução das consultas permanece praticamente constante e que seu custo é alto, comparado ao tempo de processamento da consulta. Isso se deve às diversas junções sobre a tabela `TB_DatabaseSchema` executadas pelo algoritmo de

[1] `http://www.antlr.org/`

[2] `http://dbtune.org/bbc/peel/`

[3] As consultas estão disponíveis em `http://www.inf.ufpr.br/mmgduarte/SBBD18/`



(a) Tempo de Execução das consultas

Padão Estrela	Padrões Utilizados
PE 1	?a ev:place ?p ?a mo:produced_signal ?s ?a rdf:type ?t
PE 2	?s mo:published_as ?x ?s rdf:type ?y ?s rdfs:label ?l
PE 3	?x elem:title ?b ?x rdf:type ?c ?x rdfs:label ?d

(b) Caracterização das consultas

Figura 4. Resultado dos Experimentos

tradução. No futuro, é planejado explorar a utilização de índices sobre esta tabela e uma representação em memória. O tempo de execução das consultas sobre as tabelas de entidades apresenta um ganho significativo em relação às consultas sobre a base no formato SPO. Isso se deve ao volume da tabela de triplas, que utiliza IRIs completas, além da quantidade de auto-junções no processamento de consultas. Contabilizando o tempo da tradução e processamento, as tabelas de entidades apresentam uma redução no tempo de processamento de aproximadamente 250%. A consulta C3 apresenta um comportamento diferenciado, com um ganho de 2175% devido a quantidade de dados envolvidos.

6. Conclusão

Este artigo apresentou um algoritmo de tradução de consultas SPARQL para SQL sobre uma base relacional de entidades geradas com o sistema AORR. O algoritmo foi implementado e os resultado experimentais mostraram que o tempo de processamento das consultas SPARQL teve uma redução de aproximadamente 250% quando comparado a uma base relacional de triplas. Como trabalho futuro é planejado estender o algoritmo para consultas SPARQL mais expressivas e a otimização do processo de tradução.

Referências

Aluç, G., Ozsu, M. T., and Daudjee, K. (2014). Workload matters: Why rdf databases need a new design. *Proc. of the Int. Conf. on Very Large Data Bases*, 7(10):837–840.

Chaloupka, M. and Necasky, M. (2016). Efficient sparql to sql translation with user defined mapping. In *Proc. of the Knowledge Engineering and Semantic Web Conference*.

Chebotko, A., Lu, S., and Fotouhi, F. (2009). Semantics preserving sparql-to-sql translation. In *Data Knowledge Engineering*, pages 973–1000. Volume 68 Issue 10.

Das, S., Sundara, S., and Cyganiak, R. (2012). R2rml: Rdb to rdf mapping. `http://www.w3.org/TR/r2rml/`.

Duarte, M. M. G. and Hara, C. S. (2018). Otimização do mapeamento de consultas SPARQL para SQL. In *Escola Regional de Banco de Dados*.

Michel, F., Zucker, C. F., and Montagnat, J. (2016). A generic mapping-based query translation from sparql to various target database query languages. In *Proc. of the 12th International Conference on Web Information Systems and Technologies*.

Prado, R. L., Schroeder, R., and Hara, C. S. (2018). Armazenamento otimizado de dados RDF em um SGBD relacional. In *Proc. of the Brazilian Symposium on Databases*.

Rodriguez-Muro, M., Hardu, J., and Calvanese, D. (2012). Quest: Effcient sparql-to-sql for rdf and owl. In *Proc. of the ISWC 2012 Posters Demonstrations Track (ISWC-PD)*.