JOÃO GUSTAVO PAULUK

PROCESSAMENTO DE CONSULTAS SPARQL EM UM SGBD RELACIONAL

Monografia apresentada à disciplina de Trabalho de Graduação em Banco de Dados como requisito à conclusão de curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Profª. Drª. Carmem Satie Hara

CURITIBA

2016

Resumo

É notório que a acessibilização da computação está influenciando as formas e os tipos de como são armazenados os dados. Com isto, verificou-se que o volume de informações armazenadas tem crescido de uma forma que não se imaginava. No entanto, as tecnologias atualmente utilizadas não possuem uma proposta de reaproveitamento do conhecimento já disponível. Com a ciência deste problema, foi proposto o modelo RDF. A modelagem do conhecimento em uma forma de triplas, aliada a padrões para identificar unicamente um recurso, permite a realização de inferências lógicas sobre os dados. Além disso, também oferece uma maior facilidade no compartilhamento de informações de uma aplicação para outra, sem que haja a necessidade de haver um entendimento prévido da forma de organização das informações contidas naquela base de dados.

SPARQL é o framework proposto pela W3C para realizar consultas sobre esta base de dados. Sua estrutura permite que padrões de triplas sejam combinados para a geração de um resultado final. Porém, em grandes volumes de dados, torna-se necessário a otimização destas consultas. Esta monografia apresenta um sistema que permite a realização de consultas SPARQL em uma base de dados RDF, utilizando um SGBD relacional como *backend*. O objetivo deste trabalho é fornecer uma metodologia de mapeamento de uma consulta SPARQL para SQL que seja compatível com o esquema relacional proposto. A localização das triplas na base de dados é realizada através de uma tabela que possui informações sobre a coluna e a tabela em que está armazenada. Após consulta nesta tabela, o método proposto propõe uma filtragem destas informações obtidas para possibilitar a montagem de uma consulta em SQL. Foi possível verificar durante os experimentos desenvolvidos que as consultas traduzidas para tabelas de dados estruturados tiveram

um desempenho superior de mais de 700% em relação ao armazenamento de triplas em três colunas. Porém, a maneira de implementação desta tradução fez com que muito dos ganhos obtidos fossem minimizados. Um desempenho ainda melhor poderia ser obtido utilizando uma outra linguagem de programação e/ou realizar testes em bases de dados maiores.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo

1

Introdução

Há uma crescente necessidade de agregar, coletar, compartilhar e reutilizar as informações em um mundo em que o conhecimento está cada vez mais descentralizado (Walji, 2010). Por isso, a procura por tecnologias que consigam compartilhar e interligar informações aumenta.

Atualmente, o principal motivo para armazenar os dados é prover facilidades para uma rápida recuperação das informações pelo ser humano. Isto requer também o desenvolvimento de técnicas para o processamento de conhecimento automático através de aplicações. A Web semântica surge com este propósito. Ela fornece uma estrutura para que informações descentralizadas sejam organizadas de tal forma que um processamento computacional semântico seja possível (Berners-Lee et al., 2001).

Para possibilitar o desenvolvimento da Web semântica, uma das importantes tecnologias é o *Resource Description Framework* (RDF) (Berners-Lee et al., 2001). O RDF é estruturado através de triplas (i.e., sujeito, predicado, objeto) e permite o armazenamento de "recursos", isto é, qualquer tipo de documento, conceito abstrato, objetos físicos, etc. (Manola et al., 2014). Sua estrutura foi criada para permitir que um computador entenda seu vocabulário e sua relação entre cada dado armazenado, possibilitando realizar inferências lógicas sobre eles e facilitando o uso destes dados por outras aplicações (Passin, 2005).

Maria moraEm Rua 1 .
Maria estuda Computação .
José moraEm Rua 1 .
Maria estuda Matemática .

(a) Representação de uma base em formato de triplas.

sujeito	predicado	objeto
Maria	moraEm	Rua 1
Maria	curso	Computação
José	moraEm	Rua1
José	curso	Matemática

(b) Representação em uma tabela de três colunas.

nome	moraEm	curso
Maria	Rua 1	Computação
José	Rua 1	Matemática

(c) Representacao das triplas em um esquema estruturado.

Figura 1.1: Formas de armazenamento do RDF

A base de conhecimento RDF pode ser armazenada em um SGBD relacional em uma única tabela com três colunas, uma para cada componente da tripla, ou de uma forma mais estruturada. Por exemplo, as triplas presentes na Figura 1.1(a) pode ser armazenada em uma tabela de três colunas (Figura 1.1(b)) ou em uma versão mais estruturada, Figura 1.1(c).

Para realizar consultas neste banco de conhecimento, a W3C[1] recomenda a utilização do *SPARQL Protocol and RDF Query Language* (SPARQL) (Manola et al., 2014). A figura 1.2 apresenta um exemplo de consulta para obter o nome das pessoas em que moram na "Rua 1"e estudam "Computação". Observe, porém, que as URIs que deveriam estar contidas no predicado e objeto foram substituidas por strings para facilitar o entendimento.

```
SELECT ?n WHERE (?n moraEm "Rua 1") AND (?n estuda "Computação")
```

Figura 1.2: Exemplo de consulta SPARQL.

Dado que estes dados são armazenados em um esquema de triplas, a consulta deverá realizar ao menos uma auto-junção para obter as pessoas que moram na "Rua 1"e estudam "Computação"ao utilizar a tabela de três colunas. Em geral, a quantidade de auto-junções

[1]https://www.w3.org/. Acesso em: 21 de dez. 2016.

em uma consulta com n padrões de triplas é igual a $n - 1$, resultando, na maioria das vezes, em sérios problemas de desempenho.

Não obstante, existe uma crescente adoção e criação de padrões para o RDF por parte de governos, comunidades científica (Shadbolt et al., 2006) e indústrias (Cardoso, 2007; Huang et al., 2011). Há portanto, a necessidade de tornar consultas SPARQL mais eficientes. Uma das maneiras é obter um esquema relacional que capture a estrutura da base de conhecimento. Com isto, é possível aproveitar as otimizações já realizadas pelo SGBD relacional (Pham et al., 2015).

Por isso, Pham et al., 2015 propõe uma técnica para obter um esquema relacional a partir de uma base de conhecimento em RDF. Este modelo analisa a frequência dos recursos, as ontologias do documento e as anotações para criar um esquema relacional. Esta proposta foi adaptada por Lima, 2016, considerando que a base relacional não será consultada diretamente, mas utilizada apenas como um *backend* de armazenamento eficiente para RDF.

O objetivo desta monografia é desenvolver um metodologia de mapeamento de uma consulta SPARQL para SQL, considerando uma base relacional estruturada da forma proposta por Lima, 2016.

O restante deste trabalho é organizado da seguinte maneira: o Capítulo 2 apresenta a definição e os conceitos pertinentes ao modelo RDF e ao framework SPARQL. No Capítulo 3, é apresentado propostas já existentes para a criação de um esquema relacional estruturado de uma base RDF e para o mapeamento de uma consulta SPARQL para SQL. O Capítulo 4 define o sistema AORR e detalha o funcionamento da metodologia de tradução das consultas SPARQL para SQL proposta nesta monografia. Os experimentos realizados e os resultados obtidos foram descritos no Capítulo 5. Finalmente, o Capítulo 6 apresenta as considerações finais e prospostas para trabalhos futuros.

Capítulo

2

RDF

Este capítulo apresenta o modelo RDF e a forma em que é estrurado. Além disso, define e explica o funcionamento do *framework* SPARQL. O capítulo é organizado da seguinte maneira: a Seção 2.1 apresenta o modelo RDF e suas características principais. O framework recomendado pela W3C para realizar consulta numa base de dados RDF, o SPARQL, é apresentado na Seção 2.2.

2.1 Resource Description Framework

Resource Description Framework (RDF) é o *framework* sugerido pela W3C para expressar informações sobre "recursos"na Web semântica (Graham Klyne, 2014). Neste contexto, um recurso simboliza qualquer conceito, como documentos, pessoas, conceitos abstratos, sequências de caracteres e até metadados.

Ao contrário da forma de armazenamento de dados atualmente, em que não existe um consenso na forma em que os recursos são referenciados, o objetivo do RDF é fornecer um modelo para que informações possam ser processadas por aplicações sem a necessidade de intervenção humana. Através deste framework, as bases de conhecimentos não ficam atrelados especificamente às aplicações ao qual foram projetadas (Passin, 2005), permitindo que se publique e interligue dados na Web. Desta maneira, sistemas podem utilizar

a disponibilidade de ferramentas comuns de análise e processamento de dados em seus aplicativos (Manola et al., 2014).

Uma característica importante do RDF é o relacionamento entre dois recursos. Esta relação é realizada de maneira análoga ao que é possível realizar em português, isto é, a utilização de um verbo para conectar dois substantivos. Por exemplo, seja dois substantivos, João e sanduíche. Consideremos que o João está preparando o sanchuíche, a sentença "João faz o sanduíche" é verdadeira. Em RDF, um proposição é sempre composta por três elementos, por isso denominado **tripla**. Na analogia, os recursos 'João' e 'sanduíche' compõem, respectivamente o **sujeito** e o **objeto** da tripla. Por fim, o elemento 'faz' é chamado de **predicado**.

Se a tripla acima fosse compartilhada com outras aplicações, é provável que existam diversas pessoas com o nome 'João', além de haver diferentes tipos de 'sanduíches', não sendo possivel, portanto, se referir a estes elementos unicamente. Dado que um dos objetivos do RDF é o de interligar bases de dados de diferentes origens (Manola et al., 2014), os recursos em RDF são representados de três maneiras distintas:

1. *Internationalized Resource Identifier* (IRI): Um *Uniform Resource Identifier* é um conjunto de caracteres utilizados para identificar um recurso. Ele permite que aplicações analisem partes comum da URI sem a necessidade de conhecer os esquemas de todos os identificadores (Berners-Lee et al., 2005). Um IRI pode ser definido de forma similar a URI, mas estende seu conjunto de caracteres ao UCS (Duerst e Suignard, 2003). Em RDF, um IRI é um recurso global que poderá ser visto por outros documentos e deverá ser a única referência para identificar um dado recurso.

2. **Literal**: Os literais são utilizados para representar valores, como número, datas e textos. Segundo (Graham Klyne, 2014), um literal em um grafo RDF pode conter dois ou três dos seguintes elementos:

   (a) Forma léxica: Um conjunto de caracteres no formato UNICODE.

   (b) Tipo do dado: Um IRI identificando o tipo do dado representado.

(c) Tag de linguagem: Uma tag de linguagem deve ser utilizada se e somente se o tipo do dado é um http://www.w3.org/1999/02/22-rdf-syntax-ns#langString.

3. Blank nodes: *Blank nodes* são utilizados para descrever uma relação entre determinado recurso, mas não denota, necessariamente, algo em específico. O RDF não define uma estrutura interna para este tipo de representação de recurso (Graham Klyne, 2014). Por ser visível apenas do documento, são considerados como uma variável local (Passin, 2005).

Os diferentes tipos de dados em RDF são exemplificados na Tabela 2.1.

tipo de dado	exemplo
IRI	<http://dbtune.org/bbc/peel/perf_ins/2084792e4ec60da59557d59f45203534>
literal	"1993-01-09"
literal com tipo	"200.0"^^xsd:integer
literal com linguagem	"Album de John Peel"^^xsd:string@"pt"

Tabela 2.1: Exemplos dos tipos de dados em RDF.

Por fim, dada sua estrutura de triplas, um banco de conhecimento RDF pode ser representado por um grafo direcionado rotulado, onde:

- Vértice de saída representa o sujeito.
- Vértice de chegada indica o objeto
- Arco rotulado simboliza o predicado

De uma forma genérica, uma tripla em RDF pode ser representada pelo grafo da Figura 2.1.

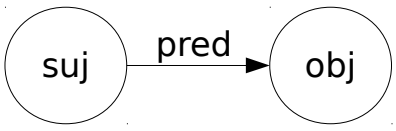

Figura 2.1: Representação de uma tripla em um grafo.

Um exemplo de base de dados em que triplas são armazenadas em uma tabela relacional de colunas (sujeito, predicado, objeto) pode ser observada no Tabela 2.2. Esta monografia, para simplificação, denotará uma URI ao inserir uma barra (/) no início de

sua palavra e literais entre aspas duplas. Ela será utilizada como base para os exemplos das figuras desta monografia.

sujeito	predicado	objeto
/pessoa1	/nome	"Alice"
/pessoa1	/conta	/conta1
/pessoa1	/conjuge	"Fernando"
/pessoa2	/nome	"Joaquim"
/pessoa2	/conta	/conta2
/pessoa3	/nome	"Joao"
/pessoa3	/conta	"1234-5"
/pessoa3	/type	/pessoa
/pessoa4	/nome	"Maria"
/pessoa4	/conta	"2589-9"
/pessoa4	/type	/pessoa
/pessoa3	/dependente	/pessoa4
/conta1	/saldo	"100.00"
/conta1	/type	/conta
/conta1	/lanctCta	"52.00"
/conta2	/saldo	"100.00"
/conta2	/type	/poupança
/transacao1	/type	/transacao
/transacao1	/descricao	"Compra no cartão de crédito"

Tabela 2.2: Exemplo de base RDF representada em esquema relacional de três colunas.

2.2 SPARQL Protocol And RDF Query Language

SPARQL Protocol And RDF Query Language (SPARQL) é o padrão definido pela W3C para consultar e manipular dados armazenados em RDF na Web Semântica (Harris e Seaborne, 2013).

SPARQL é a definição do seguinte conjunto de especificações para este fim:

- Linguagem de consulta em uma base de dados RDF.
- Apresentação dos resultados da consulta SPARQL em quatro formatos: XML, JSON, CSV, TSV.
- Extensão da linguagem de consulta SPARQL para delegar subconsultas a outros pontos SPARQL.
- Linguagem para atualização de grafos RDF.
- Protocolo para realizar consultas e atualizações em serviços SPARQL.

- Método para descobrir e um vocabulário para descrever serviços SPARQL.
- Gerenciamento de grafos RDF através de operações HTTP.

Como o objetivo deste trabalho é possibilitar que consultas SPARQL sejam feitas em um modelo relacional de uma base RDF, esta monografia irá descrever somente a linguagem de consulta SPARQL.

2.2.1 SPARQL Query Language

A linguagem recomendada para a realização de consultas em uma base de conhecimento RDF é a *SPARQL Query Language*. Segundo Pérez et al., 2009, todo o processamento de uma consulta SPARQL é composta por três etapas: o reconhecimento de padrões, aliado a seus filtros e uniões, os modificadores de resultados e a apresentação do resultado.

A primeira etapa é composta, na maioria das vezes, por diversos padrões de triplas. Um padrão de tripla é semelhante a estrutura de uma tripla, mas seu sujeito, predicado e objeto podem ser variáves, IRIs ou literais (Harris e Seaborne, 2013). Entre cada padrão de tripla é possivel aplicar as operações de conjunção ou concatenação (`.`) , disjunção(`OPTIONAL`) ou união (`UNION`). Por exemplo, observe a consulta da Figura 2.2(a) com as operação de conjunção e disjunção. Já a Figura 2.2(b), ilustra o resultado desta consulta, se executada na base de dados da Figura 2.2.

```
SELECT
  ?n ?c ?s
WHERE {
  ?a nome ?n .
  ?a conta ?c
  OPTIONAL ?c saldo ?s
}
```

(a) Exemplo de consulta SPARQL.

n	c	s
Alice	/conta1	"100.00"
Joaquim	/conta2	"200.0"
João	"1234-5"	
Maria	"2589-9"	

(b) Resultado da consulta SPARQL.

Figura 2.2: Exemplo de consulta para a base do apêndice.

Além disso, ainda é possível projetar somente as triplas que satisfaçam uma determinada expressão booleana ao utilizar a operação `FILTER` e/ou somente campos que possuem

dados não vazios (`BOUND`). Ainda assim, é permitido utilizar outras funções, como a verificação de expressões regulares e operadores lógicos entre literais de tipos numéricos.

Em uma consulta SPARQL, os literais dentro de um padrão de triplas deverão estar contidas entre aspas simples ou duplas e podem apresentar seu tipo de dado ou uma anotação de linguagem. Nestes casos, somente triplas que possuem estas anotações de dados é que serão exibidas no resultado final.

A segunda parte é a aplicação de modificadores de resultado, em que operadores como projeções e a limitação de quantidade de resultados podem ser aplicadas. A última é a apresentação de saída que pode ser de diferentes tipos, como consultas sim/não ou seleção de valores que combinem com o padrão de triplas (Pérez et al., 2009).

Capítulo

3

Mapeamento RDF para Relacional

Este capítulo apresenta uma proposta já existente para o armazenamento estruturada de uma base de conhecimento RDF em um esquema relacional. Também apresenta um modelo de mapeamento SPARQL - SQL de um outro autor.

O capítulo é organizado da seguinte forma: A Seção 3.1 apresenta uma forma de mapeamento de dados RDF para um esquema relacional. Na Seção 3.2, um processo de tradução de uma consulta SPARQL para SQL é exposto.

3.1 Emergent Relational Schema

Emergent Relational Schema é o resultado do algoritmo proposto por Pham et al., 2015 para extrair um esquema relacional de uma base de conhecimento RDF. Esta forma de armazenamento permite que consultas sejam mais eficientes que as realizadas em um esquema de triplas (Pham et al., 2015).

Este trabalho utiliza o conceito de ***characteristic set*** para denominar um conjunto de predicados de um certo sujeito em uma base RDF. De acordo com Neumann e Moerkotte, 2011, um recurso pode ser unicamente identificado através de seus arcos de saída. Deste modo, apesar do modelo RDF não possuir um esquema fixo, esta definição utiliza a idéia de que é possível caracterizar um recurso através de um subconjunto de suas arestas de

saída. Por exemplo, para a base do Apêndice 2.2, alguns dos *characteristic sets* seriam (nome, conta, cônjuge), que representa /pessoa1, e (nome, conta, type), para /pessoa4.

O algoritmo para obter o *Emergent Relational Schema* é baseado nas seguintes etapas:

1. Descoberta dos *characteristics sets* (CSs): Nesta etapa, é verificado toda a base de triplas para encontrar os predicados que ocorrem em cada sujeito. Se esta tripla possuir um literal como seu objeto, o algoritmo armazenará este tipo em uma *hash map*. Propriedades não literais são analisadas para procurar relacionamentos entre os *characteristic sets*. Estas relações são armazenadas em uma estrutura de dados específica para este objetivo.

2. Rotulamento dos CSs: Este trabalho também possui como meta fornecer um esquema relacional de uma base RDF enxuta e com nomes de tabelas e atributos amigáveis. Por isso, o algoritmo utiliza o rotulamento das *characteristic set* e seus relacionamentos como uma técnica auxiliar para nomear os atributos dos esquemas relacionais e diminuir a quantidade de tabelas a serem criadas.

   Esta etapa é baseada na exploração das informações semânticas da base, isto é, as ontologias. A técnica consiste em verificar se existe algum rótulo na ontologia. Caso não seja possível nomear o atributo desta maneira, é verificada a hierarquia da entidade nesta ontologia. Se ainda assim uma descrição satisfatória não for encontrada, o relacionamento entre os *characteristic set* é levado em conta. Por fim, se nenhum dos métodos mencionados funcionarem, é utilizado somente o prefixo da ontologia (e.g., http://purl.org/goodrelations/v1#offers torna-se offers).

3. Junção dos CSs: Esta etapa realiza junções dos *characteristic sets* que são semanticamente ou estruturalmente semelhantes. Na verificação a partir da semântica dos dados, é considerado que rótulos gerados a partir de uma mesma ontologia são iguais e, por isso, são unidos. Em situações que as classes são distintas, verifica-se as hierarquias destes *characteristics sets* e une-os se possuirem um ancestral em comum (conceito de generalização). Ao analisar a estrutura dos registros, são combinados os *caracteristcs sets* que identificam propriedades semelhantes. Finalmente,

a quantidade de referências a um mesmo *characteristic set* também pode acarretar uma junção.

4. Filtro de esquemas: Esta etapa realiza a filtragem de *characteristic sets*. O primeiro critério é retirar os *characteristic set* poucos frequentes. Em seguida, são eliminadas as propriedades poucos comuns de *characteristic sets* ainda não filtrados. Esta fase considera e mantém tabelas que, embora pequenas, são muito referenciadas por outras.

5. Filtragem de instâncias: A última etapa é destinada a manter uma homogeneidade das instâncias. Três tipos de filtro são realizados nesta etapa:

   (a) Entidades de tipos diferentes: Como SGBDs relacionais fixam um tipo de dado para uma certa entidade, há a necessidade de avaliar se multiplas colunas de tipo distintos deverão ser mantidos. Caso não haja uma frequência mínima, os dados são movidos para o *overflow*.

   (b) Tabelas de relacionamentos infrequentes: As tabelas de relacionamentos que não são muito referenciadas são enviadas ao *overflow*.

   (c) Tabelas multivaloradas pequenas: Tabelas multivaloradas que não possuam uma certa quantidade mínima de valores também são retiradas do esquema e passam a existir no *overflow*.

pessoa1		
OID	nome	conta
/pessoa1	Alice	/conta1
/pessoa2	Joaquim	/conta2

pessoa2			
OID	nome	conta	type
/pessoa3	João	"1234-5"	/pessoa
/pessoa4	Maria	"2589-9"	/pessoa

conta		
OID	saldo	type
/conta1	100.00	/conta
/conta2	200.00	/poupança

overflow		
subj	pred	obj
/pessoa1	/cônjuge	"Fernando"
/pessoa5	/nome	"Maria"
/pessoa3	/dependente	pessoa4
/conta1	/lanctoCta	52.00
/transacao1	/type	/transacao
/transacao1	/descricao	"Pagamento em cartão de crédito"

Figura 3.1: Exemplo de um Emergent Relational Schema.

Para cada *characteristic set* resultante deste algoritmo, é criada uma classe. Cada classe, torna-se uma tabela e suas propriedades, colunas. De acordo com (Pham et al., 2015), o algoritmo gera um esquema relacional que identifica uma regularidade dos dados em mais de 90%. A Figura 3.1 ilustra uma possível estruturação dos dados da Figura 2.2 gerado a partir deste algoritmo.

3.2 Tradução SPARQL para SQL

Esta seção apresenta as características do mapeamento de consulta proposta por Chebotko et al., 2009. Eles consideram que a conversão de uma base de conhecimento RDF para uma banco de dados relacional envolve três tipos de mapeamento. O primeiro deles é o **mapeamento de esquema**, que consiste na elaboração de tabelas relacionais que permitam que as triplas sejam armazenadas. O outro é o **mapeamento dos dados**, destinado a mapear as triplas para estas novas tabelas. Finalmente, o último é o **mapeamento de consulta**, responsável por traduzir uma consulta SPARQL para SQL.

Os dois primeiros mapeamentos podem ser realizados conforme descrito na seção 3.1. O último é o proposto por Chebotko et al., 2009 e apresenta as seguintes características:

- Corretude: a consulta SQL não deve ser somente semanticamente equivalente à original, mas seus resultados também devem ser semelhantes.
- Independência do esquema: o mapeamento entre as consultas não devem ser dependentes do esquema relacional.
- Eficiência: Tanto o processo tradução quanto a consulta SQL produzida devem ser eficientes.

Uma das propostas destes autores, como já mencionado, é fornecer um modelo de mapeamento de consultas que seja independentes do esquema relacional. Tal característica é alcançada ao utilizar as funções $\alpha(tp)$, que recebe como parâmetro um padrão de tripla e retorna a tabela em que a armazena, e $\beta(tp, POS)$, que retorna a coluna da localização

de *POS* na tabela que armazena o padrão de tripla *tp*, onde POS pode assumir o valor de sujeito, predicado ou objeto.

Por exemplo, assuma que a base de dados apresentada na Tabela 2.2 está armazenada na tabela `Tripla`. Ao procurar a localização do padrão de triplas `?a nome ?n` através da função alfa, o sistema retornará `Tripla`, isto é, $\alpha(?\texttt{a nome } ?\texttt{n}) = \texttt{Tripla}$. Ao tentarmos obter a localização do sujeito desta tripla na tabela retornada por alfa, o sistema retornará sua coluna, no exemplo, `sujeito`, ou seja, $\beta(?\texttt{a nome } ?\texttt{n}, sujeito) = \texttt{sujeito}$.

Além disso, os autores também apresentam a função `genCond-SQL`. Ela gera uma expressão booleana que é avaliada como verdadeira se e somente se combina com registros do esquema relacional e `genPR-SQL` é uma função que retorna somente a projeção da consulta aplicado aos atributos do esquema relacional.

Este algoritmo gera a consulta final, ao criar para cada padrão de tripla, uma subconsulta e aninhá-lo ao próximo padrão de triplas. Os relacionamentos entre as consultas SPARQL são definidas conforme a Tabela 3.1, em que `pt1` e `pt2` são padrões de triplas, `X` uma entidade e `expr` uma expressão booleana. Para cada padrão de tripla de uma consulta SPARQL, é gerado a seguinte subconsulta:

$$trans(tp, \alpha, \beta) = SELECT\ DISTINCT\ genPR - SQL$$
$$FROM\ \alpha(tp)\ WHERE\ genCond - SQL \quad (3.1)$$

SPARQL	SQL
pt1 AND pt2	pt1 INNER JOIN pt2
pt1 OPT pt2	pt1 LEFT OUTER JOIN pt2
pt1 UNION pt2	pt1 LEFT OUTER JOIN pt2 UNION pt2 LEFT OUTER JOIN pt1
FILTER expr	WHERE(expr)
BOUND(X)	X IS NOT NULL

Tabela 3.1: Relação dos operadores SPARQL para SQL.

Ao final do processo, a consulta SPARQL da Figura 1.1(a) resulta na consulta representada pela Figura 3.2.

```
SELECT
  q4.c AS c, q4.s AS s, r3.n
FROM
  (SELECT DISTINCT
     r2.c AS c, r1.s AS s
   FROM
     ((SELECT DISTINCT
        subj AS c, obj AS s
      FROM
        Tripla
      WHERE
        pred = '/saldo' ) r1
     LEFT OUTER JOIN (
      SELECT
        subj AS a, obj AS c
      FROM
        Tripla
      WHERE
        pred = '/conta' ) r2 ON
      r1.c = r2.c OR r1.c IS NULL OR r2.c IS NULL) q4
     INNER JOIN (
     SELECT DISTINCT
       subj AS a, obj AS n
     FROM
       pred = '/nome') r3 ON
      r3.a = q4.a
```

Figura 3.2: Consulta traduzida pelo algoritmo de Chebotko.

3.3 Discussões

Neste capítulo foi apresentado alguns dos modelos existentes para realizar o mapeamento de uma base de conhecimento RDF para o esquema relacional.

A geração de um esquema relacional, a partir de uma base de dados RDF, proposto por Pham et al., 2015 será utilizado como referência no próximo capítulo. Além disso, o mapeamento de uma consulta SPARQL para SQL proposta por esta monografia, terá algumas características das aqui apresentadas. Por exemplo, o conceito de α e β e a idéia de subconsultas aninhadas de acordo com seus tipos de junção entre cada padrão de tripla apresentadas por Chebotko et al., 2009 foram absorvidas e adaptadas para o modelo proposto.

Capítulo

4

AORR: Um Backend Relacional para RDF

Este capítulo apresenta o sistema AORR: Armazenamento Otimizado de dados RDF em SGBDR, que utiliza um SGBD relacional como *backend* de armazenamento RDF e processamento de consulta SPARQL. A Figura 4.1 ilustra os componentes do sistema.

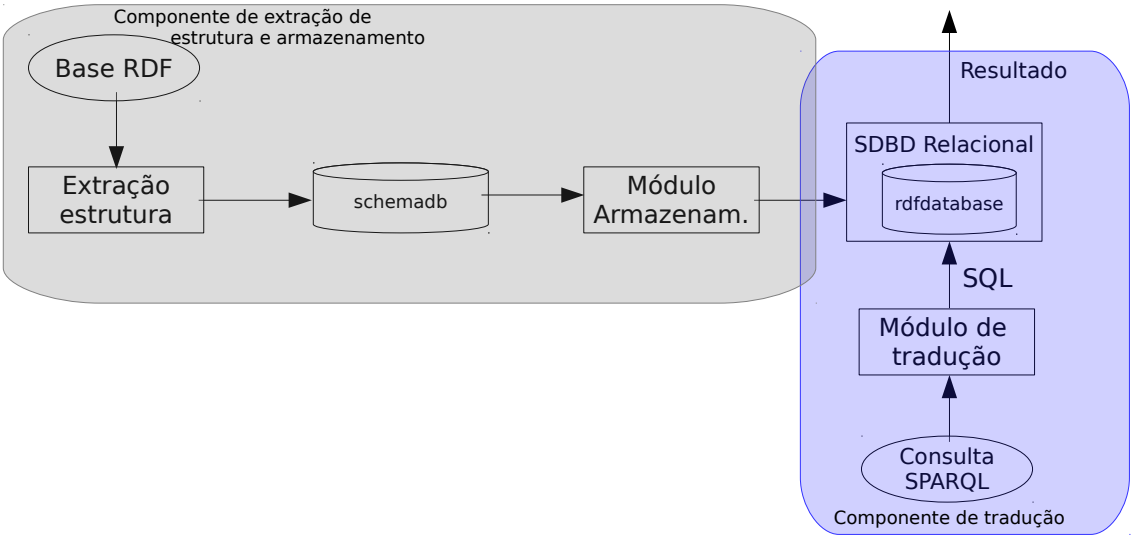

Figura 4.1: Componentes do sistema.

Em termos gerais, o componente de extração de estrutura e armazenamento é responsável por gerar um esquema relacional para armazenar uma base de dados em RDF.

O outro componente, Tradução de Consultas, é responsável por receber uma consulta em SPARQL e transformá-la em SQL para que possa assim, ser processada no sistema AORR.

O presente trabalho tem como objetivo desenvolver o módulo de tradução.

4.1 Módulo de extração de estrutura e armazenamento

O módulo de estrutura e armazenamento utiliza uma técnica similar a proposta por Pham et al., 2015 e foi desenvolvida por Lima, 2016.

O resultado destes módulos é composto por informações sobre o mapeamento da base RDF para um esquema relacional, armazenado em `dbschema`, e a base RDF `rdfdatabase` armazenada em um SGBD relacional `mappedb`.

O esquema da base `mappedb` é composto por um conjunto de esquemas de relações $S = \{R_1, ..., R_n, Ov_{R_1} ..., Ov_{R_n}, Ov\}$.

As relações $R_1, ..., R_n$ contém uma estrutura do tipo $\{s, p_1, ..., p_n\}$, no qual s é o sujeito correspondente aos predicados $p_1, ..., p_n$. Os registros destas colunas são os diferentes objetos $o_1, ..., o_n$ referentes aos predicados $p_1, ..., p_n$. Os valores destes objetos podem conter literais ou URIs. Nesta monografia, este conjunto de tabelas compõem a **parte estruturada** da base de dados. As demais tabelas, que serão descritas abaixo, são denominadas **parte não estruturada** do esquema.

Registros que possuem o mesmo tipo de sujeito que as tabelas $R_1, ..., R_n$, mas não possuem os requisitos necessários, apresentados em 3.1, para estarem presentes nestas tabelas são inseridas nas relações de *overflow* $Ov_{R_1}, ..., Ov_{R_n}$. Estas são tabelas que possuem três colunas para armazenar as triplas, isto é, para o sujeito, o predicado e o objeto.

Por fim, triplas que não se encaixam em nenhum dos casos anteriores são inseridas na tabela de *overflow* geral Ov. Sua estrutura é semelhante às tabelas $Ov_{R_1}, ..., Ov_{R_n}$, mas os sujeitos desta tabela não possuem relação com a parte estruturada da base.

Note que Lima, 2016 propõe que as triplas da parte não estruturada fiquem em uma tabela de *overflow* específicas de seu sujeito. Esta modelagem difere de Pham et al., 2015, devido a estes dados serem frequentemente utilizados. Com isto, seu tamanho reduzido tem potencialmente um desempenho superior durante o mapeamento da consulta. A Figura 4.2 apresenta estes conceitos aplicados ao exemplo da Figura 2.2. Observe que as tabelas $overflow_{pessoa1}$, $overflow_{pessoa2}$ e $overflow_{conta}$ foram criadas para armazenar as triplas "excedentes", cujos sujeitos se referem às tabelas *pessoa*1, *pessoa*2 e *conta*, respectivamente.

pessoa1		
OID	nome	conta
/pessoa1	Alice	/conta1
/pessoa2	Joaquim	/conta2

$overflow_{pessoa1}$		
subj	pred	obj
/pessoa1	/cônjuge	"Fernando"
/pessoa5	/nome	"Maria"

pessoa2			
OID	nome	conta	type
/pessoa3	João	"1234-5"	/pessoa
/pessoa4	Maria	"2589-9"	/pessoa

$overflow_{pessoa2}$		
subj	pred	obj
/pessoa3	/dependente	pessoa4

conta		
OID	saldo	type
/conta1	100.00	/conta
/conta2	200.00	/poupança

$overflow_{conta}$		
subj	pred	obj
/conta1	/lanctoCta	52.00

overflow		
subj	pred	obj
/transacao1	/type	/transacao
/transacao1	/descricao	"Pagamento em cartão de crédito"

Figura 4.2: Exemplo de um esquema gerado para AORR.

A informação sobre a localização de cada tripla de `rdfdatabase` em `mappedb` está armazenada na tabela `dbschema`. Esta tabela contém metadados dos predicados, dos tipos de sujeitos e tipos de objetos. Além disso, ela também armazena o nome da tabela e a coluna em que estão localizados. Para cada conjunto distinto de (tipo de sujeito, predicado, tipo de objeto, tabela), um registro é inserido neste esquema. A Figura 4.3 exibe como seria o `dbschema` da base de dados da Figura 4.2.

cs_identifier	property	valueType	tableAttribute	tableName
cs1	/nome	literal	nome	pessoa1
cs1	/conta	cs3	conta	pessoa1
cs1	/conjuge	literal	obj	overflow_pessoa1
cs1	/nome	literal	obj	overflow_pessoa1
cs2	/nome	literal	nome	pessoa2
cs2	/conta	literal	conta	pessoa2
cs2	/type	literal	type	pessoa2
cs2	/dependente	cs2	obj	overflow_pessoa2
cs3	/saldo	literal	saldo	conta
cs3	/type	literal	type	conta
cs3	/lanctoCta	literal	obj	overflow_conta
cs4	/type	liretal	obj	overflow
cs4	/descricao	literal	obj	overflow

Figura 4.3: Exemplo de dados da tabela dbschema.

4.2 Módulo de tradução

Nesta seção será apresentado o algoritmo de tradução de uma consulta SPARQL para SQL, considerando somente a parte estruturada da base `mappedb`. A extensão envolvendo a parte não estruturada deste esquema será tratada na seção 4.3.

Uma consulta SPARQL Q pode ser definida como um par (R, PT), onde R é o conjunto das variáveis a serem projetadas e PT é um conjunto de padrão de triplas (s, p, o), onde s é uma variável, p um literal e o pode ser um literal, URI ou variável e representam o sujeito, o predicado e o objeto do padrão de triplas, respectivamente.

```
SELECT
   ?n ?s ?t
WHERE {
   ?p /nome ?n .
   ?p /conta ?c .
   ?c /saldo ?s .
   ?c /type ?t
}
```

Figura 4.4: Exemplo de uma consulta SPARQL.

A Figura 4.4 apresenta um exemplo de consulta em SPARQL em que se deseja obter o nome das pessoas, suas contas, tipos das contas e os saldos. O processo de tradução desta consulta, e de qualquer outra, envolve as seguintes três etapas:

1. **Procura por padrões estrela** - É realizado uma tentativa de busca por padrões estrela, isto é, são separados todos os s distintos de PT em grupos. Ainda nesta

etapa, é realizada a busca das informações de localização de cada predicado em `dbschema`. A Figura 4.5 esquematiza esta primeira etapa considerando a consulta da Figura 4.4.

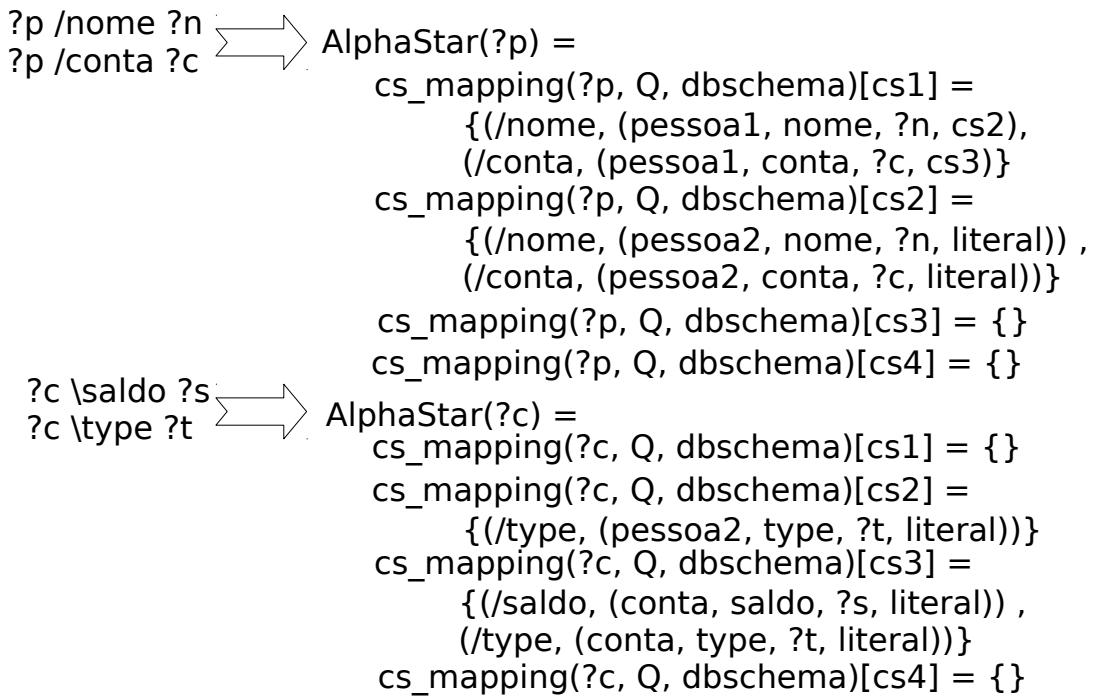

Figura 4.5: Exemplos de padrões estrela e estruturas alphaStar preenchidas.

2. **Filtragem por ligações** - A segunda parte deste processo realiza a filtragem e a montagem do relacionamento entre as variáveis. A Figura 4.6 explicita as ligações existentes entre as variáveis da consulta da Figura 4.4. Ligações sujeito com sujeito estão em vermelho. Em azul, ligações objeto - sujeto. A verificação da existência de conexões entre as variáveis possibilita a filtragem de subconsultas desnecessárias no resultado final. Por exemplo, observe que na figura 4.5 que o predicado `/type` com o tipo de sujeito `cs2` não conseguirá realizar a ligação sujeito - sujeito, pois não existe um predicado `/saldo` para este tipo de sujeito. Nestes casos, descartar-se-á esta informação. Se ocorrer de um predicado não possuir nenhuma informação de esquema, o processo retornará vazio.

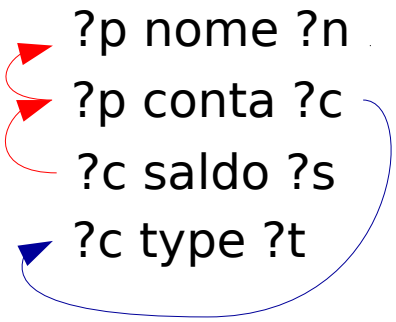

Figura 4.6: Ligações existentes entre as variáveis.

3. **Montagem do comando SQL** - Após confirmado a existência de ligações, a montagem da consulta em SQL é realizada ao organizar as projeções e selecionar e relacionar as tabelas obtidas nas etapas anterioes.

As próximas subseções explicam o funcionamento detalhado de cada um das etapas acima. Eles utilizam as seguintes estruturas de dados:

- A função $predicados(var, Q)$ é utilizada para retornar uma lista contendo todos os predicados presentes nos padrões de triplas que possuem a variável var na posição de sujeito da consulta Q. Isto é,

$$predicados(var, Q) = \{p | (var, p, \_) \in Q.tp\} \tag{4.1}$$

- $alphaStar_Q[var]$ é uma estrutura de dados utilizada para armazenar as informações dos padrões estrela que possuem como seu sujeito a variável var. Ela é composta de um vetor de $cs\_mappings$. Sua definição formal é

$$alphaStar_Q[var] = \text{vetor de } cs\_mapping \tag{4.2}$$

- A estrutura de dados $cs\_mapping_{[cs]}(var, Q, dbschema)$ armazena as informações da localização dos $predicados(var, Q)$ dos padrões de triplas de um padrão estrela,

em que o tipo de seu sujeito na base dados é cs. Isto é,

$$cs\_mapping_{[cs]}(var, Q, dbschema) = \{(p, (tab, atrib, varObj, tipoAtrib)) \text{ t.q. } p \in predicados(var, Q) \text{ e } (cs, p, tipoAtrib, tab, atrib) \in dbschema\} \quad (4.3)$$

4.2.1 Procura por padrões estrela

A primeira etapa do processo de tradução de uma consulta SPARQL $Q = (R, PT)$ é a busca de um padrão de estrela em PT. A importância desta etapa é que há uma maior probabilidade de que os predicados em um padrão de estrela estarem organizados em uma mesma tabela.

Inicialmente, para cada variável distinta v, é preenchido cada posição do vetor $alphaStar$[v] com vazio. O tamanho deste vetor é a quantidade de `cs` distintos em `dbschema` (linhas 1-3). Este algoritmo está representado no Algoritmo 1.

Algoritmo 1: Algoritmo para buscar padrões estrela

Função `alphaStarFind`

Entrada: $Q = (R, PT)$, `dbschema`
Saida: conjunto de `alphaStar`, um para cada sujeito distinto em $Q.PT$

1. para cada variavel v no papel de sujeito em Q
2. para cada `cs` existente em dbschema
3. $alphastar_Q$[v][cs] = {}
4. para cada variavel v, no papel de sujeito, e varObj, objeto do mesmo padrão de tripla, em PT
5. para cada registro r = (cs, pred, tipo, tabela, atributo) em `dbschema`
6. se pred ∈ predicados(v, Q) então
7. insere (pred, (tab, atrib, varObj, tipo)) em $alphaStar_Q$[cs]

Ao final desta etapa, as informações referentes ao tipo do objeto, tabela e coluna que armazenam cada predicado de PT encontrados em `dbschema` são armazenados em seus respectivos $alphaStar_Q[v][cs]$ (linhas 4-7). É importante notar que mesmo que um tipo de sujeito não possua todos os predicados que deveria possuir, ele é armazenado nesta etapa. A Figura 4.5 ilustra as estruturas de dados após o final desta etapa.

4.2.2 Filtragem por ligações

A próxima fase é a filtragem das informações carregadas no módulo durante a fase anterior. Este estágio é responsável por realizar a validação da existência tanto dos predicados presentes em PT, quanto do relacionamentos existentes entre eles. Além disso, também é realizada a ligação de todas as chaves estrangeiras às chaves primárias das tabelas utilizadas pela consulta SPARQL. Caso alguma destas junções não exista em `mappedb`, o processo é interrompido e sabe-se que o resultado é um conjunto vazio.

A primeira destas validações é a checagem das ligações entre os sujeitos de cada um dos padrões de triplas de PT. Este tipo de relacionamento está exemplificado em vermelho na Figura 4.6.

Algoritmo 2: Algoritmo para realizar ligações sujeito - sujeito

Função `subjSubjFilter`

Entrada: $alphaStar_Q$
Saida: `sim`, se processamento pode ser continuado. Caso contrário, `vazio`.

1. para cada $alphaStar_Q$[v]
2. para cada $alphaStar$[v][cs]
3. para cada padrão de triplas da forma $(p, (tab, atrib, var_{obj}, cs_{obj}))$
4. se var_{obj} <> null
5. se $alphaStar_Q[var_{obj}][cs_{obj}]$ é vazio
6. $alphaStar_Q[v][cs] = \{\}$
7. se todos $alphaStar$[v][cs] forem vazio
8. retorna `vazio`
9. retorna `sim`

Cada `cs_mapping` deve possuir uma informação sobre o atributo e tabela de cada padrão de tripla de um $alphaStar_Q$ (linhas 1-4). Se houver um `cs_mapping` que não possua estes dados (linha 5), ele é removido do vetor de $alphaStar_Q$ (linha 6). Ainda assim, caso ocorra que todos os $cs\_mappings$ de $alphaStar_Q$ estejam vazios, o processo retorna que a consulta é vazia (linhas 7-8). Este algoritmo é representado em Algoritmo 2. Por exemplo, veja na Figura 4.5 que o $cs\_mapping$ `cs2` de $alphaStar_Q[?c]$ possui somente o mapeamento para o predicado /type. Como não há informações a respeito do predicado /conta, as informações de esquema deste $cs\_mapping$ será descartado.

As outras junções que relacionam os objetos de $Q.PT$ com os seus sujeitos (representado pela cor azul Figura 4.6) e o que conecta os objetos de $Q.PT$ com os demais objetos possuem um comportamento similar.

A Figura 3 exibe o algoritmo proposto para realizar o processo de relacionamento dos sujeitos com os objetos de PT. Para cada predicado de um `cs_mapping csm` pertencente a $alphaStar_Q$, se o objeto for uma variável `v` do tipo `csx` (linhas 1-3), o programa procurará um `cs_mapping` do tipo `csx`, isto é, alphaStar[v][csx](linha 4). Caso não seja encontrado, este `csm` será removido do $alphaStar_Q$ (linha 5). Finalmente, se todos os `cs_mapping` de um alphaStar_Q forem vazios, o processo retorna que a consulta é vazia (linhas 6-7). Por exemplo, observe na Figura 4.5 que o *cs_mapping* cs2 do predicado /conta é do tipo literal. Como a base de dados não possui um predicado /conta, cujo sujeito é do tipo literal, esta informação será descartada.

Algoritmo 3: Algoritmo para realizar a filtragem objeto sujeito

Função `ObjSubjFilter`

Entrada: $alphaStar_Q$
Saida: Sim, se processamento pode ser continuado. Caso contrário, vazio.

```
1.  para cada alphaStar[v]
2.      para cada alphaStar[v][cs]
3.          para cada pt = (p, (tab, atrib, var_obj, cs_obj)), onde var_obj ≠ null
4.              se não existe alphaStar[x][cs_obj]
5.                  alphaStar[v][cs] = {}
6.      se todos os elementos de alphaStar[v] são vazio
7.          retorna vazio
8.  retorna sim
```

O processo de filtragem das ligações objeto com objeto segue a mesma lógica da ligação *objeto* → *sujeito* e seu algoritmo se encontra presente no Algoritmo 1.

4.2.3 Montagem SQL

A última etapa é a montagem de uma consulta em SQL. Esta fase utiliza as informações de atributos e tabelas e cada relacionamento que foram desenvolvidas nas etapa anteriores. Este processo segue a seguinte ordem:

1. Para cada `cs_mapping csi` de $alphaStar_Q[v]$ é criado uma subconsulta S_i, em que os predicados relacionados a v são projetados. Após esta etapa, cada S_i é unido,

Algoritmo 4: Algoritmo para realizar a filtragem objeto objeto

Função `objObjFilter`

Entrada: $alphaStar_Q$
Saida: Sim, se processamento pode ser continuado. Caso contrário, conjunto v

1. para cada $alphaStar$[v]
2. para cada $alphaStar$[v][cs]
3. para cada pt $(p, (tab, atrib, var_{obj}, cs_{obj}))$, onde $var_{obj<>null}$
4. para cada $S = alphaStar_Q[u][cx]$
5. se existe um conjunto $C = \{(r, (tabR, atribR, var_{obj}, csR_{obj})$
6. se não existe $(r, (tabR, atribR, var_{obj}, cs_{obj}))$ em C
7. $alphaStar_Q[v][cs] = \{\}$
8. se todos $alphaStar$[v][cs] são vazios
9. retorna `vazio`
10. retorna `sim`

formando a subconsulta $SQ(y)$ `AS` V_i. Esta etapa é equivalente a ligação sujeito - sujeito.

2. Para cada relação objeto - objeto e sujeito - sujeito encontrada na etapa anterior, uma condição $cond_k$ é criada para representar esta ligação no contexto deste esquema relacional e o relalacionamento existente entre os padrões de tripla.

3. Para cada variável v existente em $Q.R$, a coluna c_i, correspondente a v e presente na subconsulta $SQ(x)$ é criado uma projeção $Pr[l] = SQ(x).c_i$ `AS` $name(v)$, sendo $name(v)$ a função que retira os caracteres "?"e "$"do nome da variável.

4. Por fim, a montagem da consulta é realizada ao montar a consulta da seguinte forma:

$$\texttt{SELECT } Pr(1), \ldots, Pr(o) \texttt{ FROM } SQ(v_1), \ldots, \ SQ(v_n) \texttt{ WHERE } cond_1, \ldots, cond_m, \tag{4.4}$$

onde o é o número de elementos de $Q.PT$, n é o número de $alphaStar_Q$ e m é o número de ligações entre as variáveis.

A Figura 4.7 mostra o resultado da consulta final produzida para o exemplo da Figura 4.4.

```
SELECT
  p.nome AS n, c.saldo AS s, c.type AS  t
FROM
  (SELECT
     nome, conta
   FROM
     pessoa1) p,
  (SELECT
     OID, saldo, type
   FROM
     conta) c
WHERE
    p.conta = c.OID
AND p.nome IS NOT NULL
AND c.saldo IS NOT NULL
AND c.type IS NOT NTLL
```

Figura 4.7: Consulta traduzida pelo algoritmo apresentado.

4.3 Discussões

A tradução de uma consulta SPARQL de Chebotko et al., 2009, apresentada no seção 3.2, utiliza como base um esquema relacional em que todas as triplas são armazenados em uma única tabela (s, p, o). A conversão para uma consulta SQL apresentada neste trabalho, pressupõe que hajam tabelas estruturadas contendo estes dados. Deste modo, a utilização da tradução de Chebotko et al., 2009 na estrutura de Lima, 2016 não permite atributos que estão contidos em uma mesma tabela sejam selecionados em uma mesma seleção.

Além disso, a função `name` em Chebotko é apresentada como o retorno dos caracteres de uma variável, excluindo "?"e "$"por não serem permitidos como nome de coluna em SQL. Esta função é utilizada durante toda a tradução. Neste trabalho, no entanto, `name` é utilizado para o mesmo propósito, mas somente durante a montagem das projeções da consulta SQL final.

Para uma extensão deste projeto, pode-se prever a utilização da parte não estruturada da consulta. Pham et al., 2015 propõe que seja criada uma espécie de visão para os dados do *overflow* e unir à consulta produzida para a parte estruturada. Apesar desta monografia apresentar um esquema relacional um pouco diferente de Pham et al., 2015,

é possível utilizar esta mesma idéia. Para cada `cs_mapping` de $alphaStar_Q$ é realizada a união de seu overflow com sua parte estruturada.

Capítulo

5

Implementação

Este capítulo descreve detalhes da implementação dos algoritmos propostos e os experimentos realizados a partir dele.

O capítulo está organizado da seguinte maneira: a Seção 5.1 apresenta as tecnologias utilizadas para a implementação do mapeamento de consulta. O ambiente a execução dos testes propostos são descritos na Seção 5.2. Finalmente, na Seção 5.3, são apresentados as discussões a respeito dos resultados obtidos.

5.1 Tecnologias utilizadas

A implementação dos algoritmos apresentados no Capítulo 4, foi realizada utilizando a arquitetura da Figura 5.1.

Neste modelo, o programa recebe uma consulta em SPARQL e é encaminhado para um analisador léxico e sintático. Para este fim, foi utilizado a ferramenta de código aberto *ANother Tool for Language Recognition*[1]. Ela facilita a criação de uma árvore sintática a partir de uma gramática informada. ANTLR gera classes para que o programador possa percorrer a árvore sintática em diferentes linguagens de programação, dentre eles Python, Java e C++.

[1] http://www.antlr.org/. Acesso em: 21 de dez. 2016.

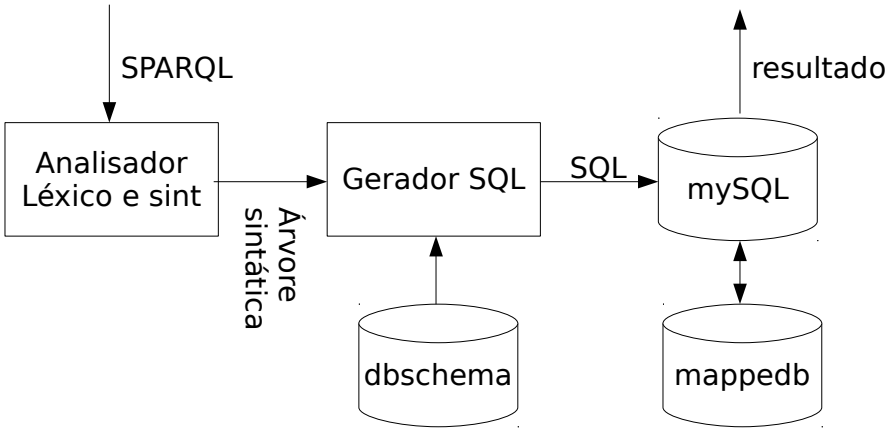

Figura 5.1: arquitetura de implementação.

Finalizado a interpretação da entrada fornecida, o gerador SQL é o resultado da implementação dos algoritmos apresentados na Capítulo 4. Ele foi implementado utilizando a linguagem python 2.7[2]. Recebe como entrada de seu módulo um consulta SPARQL já no formato $Q = (R, PT)$ e realiza a interface com a tabela `dbschema` para recuperar a localização de cada tripla.

Tanto a base de dados `mappedb`, quanto a tabela `dbschema`, foram armazenadas no SGBD relacional mySQL[3]. Na última etapa do processo, este banco de dados recebe a consulta SQL convertida e retorna o resultado da parte estruturada da base para a consulta Q.

5.2 Experimentos

Esta seção tem como objetivo avaliar as performances das consultas produzidas a partir dos algoritmos descritos em 4. Juntamente com esta análise, requisições a base `rdfdatabase` também serão realizadas para realizar comparações.

5.2.1 Ambiente de testes

Os experimentos aqui descritos foram realizados em uma máquina de CPU Intel Core i5-4200U 1.60Ghz com 4 *gigabytes* de memória RAM.

[2]http://www.antlr.org/. Acesso em: 21 de dez. 2016.
[3]https://www.mysql.com/. Acesso em: 21 de dez. 2016.

A base de dados RDF utilizada foi a Peel[4]. Ela possui registros contendo informações a respeito de músicas e performances do cantor John Peel. Além disso, esta base compõe um repositório ainda maior relacionada à música, o DBTune[5].

Esta base de dados foi carregada de duas formas distintas no banco de dados mySQL:

1. Tabela de triplas (TT): Todos os registros foram armazenados em um só tabela, denominada `RDF_Tripla`, com as colunas `subj`, `pred`, `obj` que armazenam o sujeito, predicado e objeto das triplas, respectivamente. Um índice nas colunas (pred, subj) foi criado para melhorar a performance das consultas realizadas nesta base de dados.

2. Tabelas estruturadas (TE): A base de dados RDF passou pela geração do esquema relacional proposta por Lima, 2016. De forma análoga a base de triplas, foi criado índices em todas as tabelas estruturadas, em sua coluna OID. A quantidade de registros presentes em cada tabela, pode ser visualizada na Tabela 5.1.

Tabela	Registros	Espaço(MB)
MusicArtistRDF	10427	2.5156
Overflow	0	0.0156
Overflow_MusicArtistRDF	2912	0.3906
Overflow_PerformanceRDF	13096	1.5156
Overflow_RecordingRDF	0	0.0156
Overflow_SignalRDF	0	0.0156
Overflow_SoundRDF	0	0.0156
Overflow_TrackRDF	0	0.0156
Overflow_TransmissionRDF	0	0.0156
Overflow_sameAsRDF	0	0.0156
Overflow_sub_eventRDF	0	0.0156
PerformanceRDF	28350	5.5156
RecordingRDF	3923	1.5156
SignalRDF	5534	1.5156
SoundRDF	3924	0.4375
TB_DatabaseSchema	49	0.0156
TB_FullPredicate	40	0.0156
TrackRDF	19110	3.5156
TransmissionRDF	3945	1.5156
chart_positionMultivalueRDF	1502	0.1875
createdMultivalueRDF	1522	0.1875
fk_engineerMultivalueRDF	3758	1.5156
fk_engineeredMultivalueRDF	3754	1.5156
fk_performedMultivalueRDF	11298	2.5156
fk_producedMultivalueRDF	3652	1.5156
fk_sameAsMultivalueRDF	860	0.1563
fk_sub_eventMultivalueRDF	30036	4.5156
instrumentMultivalueRDF	8742	1.5156
isrcMultivalueRDF	689	0.1094
labelMultivalueRDF	5746	1.5156
sameAsRDF	180	0.0469
sub_eventRDF	3	0.0156

Tabela 5.1: Quantidade de registros por tabela em TE.

[4]http://dbtune.org/bbc/peel/. Acesso em: 21 de dez. 2016.
[5]http://dbtune.org/. Acesso em: 21 de dez. 2016.

Ambas as bases foram carregadas com seus registros não normalizados. A informação referente ao espaço ocupado e a quantidade de registros total das tabelas pode ser visualizada na Tabela 5.2.

Base de dados	Registros(linhas)	Espaço(MB)
TT	267.613	45
TE	163.061	34

Tabela 5.2: Resumo de informações por base de dados.

5.2.2 Testes e resultados

Os testes realizados foram baseados em três casos de padrões estrelas presentes na base Peel. Cada um deles possuem três predicados distintos. A Tabela 5.3 ilustra estes casos.

Padrão estrela	Padrões utilizados
PE_1	?a <http://purl.org/NET/c4dm/event.owl#place> ?p ?a <http://purl.org/ontology/mo/produced_signal> ?s ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type ?t
PE_2	?s <http://purl.org/ontology/mo/published_as> ?x ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?y ?s <http://www.w3.org/2000/01/rdf-schema#label> ?l
PE_3	?x <http://purl.org/dc/elements/1.1/title> ?b ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c ?x <http://www.w3.org/2000/01/rdf-schema#label> ?d

Tabela 5.3: Padrões estrelas utilizados nos experimentos.

Todas as consultas foram produzidas a partir dos padrões de estrela da Figura 5.3. Cada uma das consultas produzidas foram executadas em ambas as bases, isto é, em TT e TE. A descrição das consultas segue o seguinte padrão:

1. C1: Consulta que utilizada somente PE1.
2. C2: Consulta que utiliza o PE2.
3. C3: consulta que utiliza o PE3.
4. C4: consulta em que é realizado uma junção entre PE1 e PE2.
5. C5: consulta em que ocorre junção entre todos padroes de triplas.

As consultas SPARQL e suas formas traduzidas para SQL, para os modelos TT e TE são apresentados no Apêndice A.

Os experimentos consistiram em executar dez vezes cada uma das consultas, sendo que as cinco primeiras como *warm up* e desconsideradas do resultado final. Além disso, todas as execuções foram realizadas com o parâmetro `SQL_NO_CACHE`[6], desta forma, garante-se que o banco de dados não utilizará o resultado que poderia estar *cache*.

Além de computar a duração de cada consulta, foi medido também o tempo de tradução de uma consulta SPARQL para SQL, utilizando a arquitetura de implementação exposta na Seção 5.1. A metodologia de contabilização do tempo foi feita de forma semelhante ao das consultas. A Tabela 5.2 ilustra o resumo dos tempos médios obtidos para a tradução, denotada $T(Trad)$ e para o processamento no banco de dados, $T(TT)$, para o tempo de duração nas tabelas de triplas e $T(TE)$, para tabelas estruturadas, e a quantidade de registros retornados em cada uma das consultas analisadas. A quantidade de registros retornados para a base TT e TE foram as mesmas.

Consulta	T(TT) (s)	T(TE) (s)	T(Trad) (s)	T(TE) + T(Trad)	Qtde. reg.
C1	0.045326541	0.006309875	0.05010290	0.056412784	3.976
C2	0.024669583	0.013354333	0.05702161	0.070375951	1.936
C3	0.229810875	0.031877833	0.03470019	0.066578028	19.214
C4	0.031623791	0.018156166	0.06561374	0.083769913	250
C5	0.033598291	0.022841541	0.05511248	0.077954022	250

Figura 5.2: Duração de execução médio de cada consulta.

5.3 Discussões

Observe na Tabela 5.2 que a base proposta por Lima, 2016, em termos de memória física, ocupa um espaço inferior a uma base de três colunas. Para a base Peel, esta diferença no tamanho ocupado foi de cerca de de 30% a menos na base TE. Além disso, a quantidade de linhas utilizadas na tabela TE foi consideravelmente menor, possuindo cerca de 60% do volume de TT.

Esta diferença na quantidade de linhas em tabelas, aliada a uma menor necessidade de junção entre as tabelas, fez com que consultas traduzidas para serem executadas em TE

[6]http://dev.mysql.com/doc/refman/5.7/en/query-cache-in-select.html. Acesso em: 21 de dez. 2016.

fossem consideravelmente mais rápidas. Os testes mostraram que o menor ganho foi de 1,5 vezes, caso de $C5$. Enquanto isso, obteve-se um ganho superior a 700% nas consultas $C1$ e $C3$, ilustrando a eficácia de armazenar as triplas de uma forma estruturada ao se utilizar um SGBD Relacional. A Figura 5.3 ilustra a comparação entre os desempenhos das consultas.

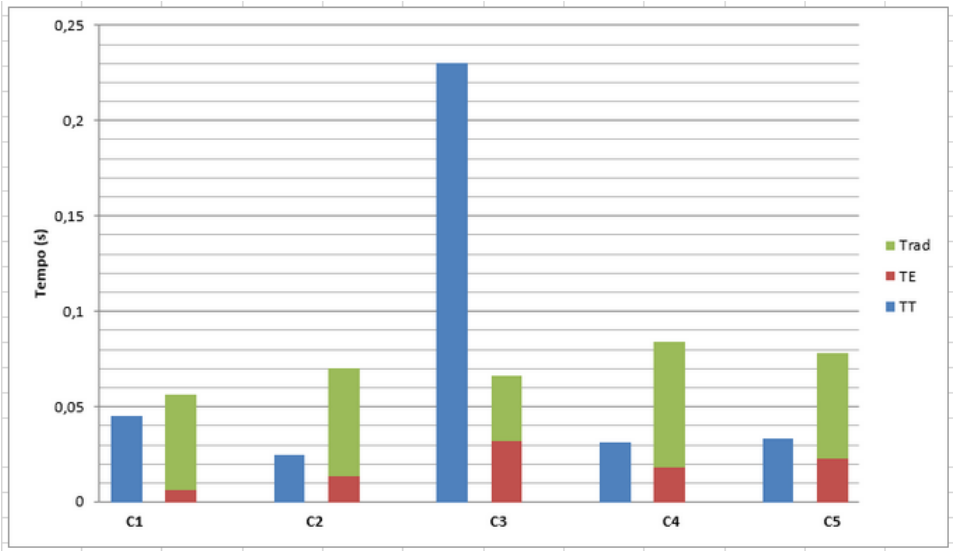

Figura 5.3: Gráfico de comparação de desempenho.

Além disso, as consultas $C1$ e $C3$ também evidenciam que o tamanho do resultado teve um impacto muito grande nos testes realizados na base TT. Observe a Tabela 5.2, em que, com exceção da consulta $C2$, o tamanho do *result set* determinou o desempenho das consultas para a base TT. Já para a base TE, a quantidade de junção entre as tabelas tiveram um relação maior, exceto para a consulta $C3$, cujo tamanho do resultado era muito superior que os demais.

Entretanto, ao levar em consideração o tempo de conversão de uma consuta SPARQL para SQL, a performance em TE só possuiu um desempenho superior em C3. No caso da consulta $C2$, a consulta chegou a ser até 350% mais lento. É importante notar, porém, que embora não termos computado o tempo de transformação de uma consulta SPARQL para SQL no modelo de esquema de triplas, em situações normais, haveria também um tempo para esta conversão. Além disso, outras formas de mapeamento de consulta poderão alterar o desempenho durante o processamento no SGBD.

Finalmente, é importante notar que em bases com uma quantidade maior de triplas, a diferença de duração entre uma consulta em uma base estruturada e uma tabela de três colunas pode ser alterada. Nestes casos, há uma maior possibilidade de consultas feitas na tabela de te três colunas apresente um desempenho ainda pior quando comparado ao realizado em base de tabelas estruturadas.

Por fim, a perda de desempenho decorrente do processo de tradução de uma consulta SPARQL pode ser atenuada ao implementar em uma outra linguagem de programação.

Capítulo

6

Conclusões

Esta monografia apresentou o sistema AORR, um modelo de armazenamento de uma base de dados RDF que utiliza um SGBD Relacional como *backend*. Ele é composto de dois módulos: o primeiro realiza a geração de um esquema relacional a partir de uma base RDF e o outro, converte de uma consulta SPARQL para SQL, compatível com a estrutura de dados criada anteriormente.

O objetivo deste trabalho era apresentar um modelo de tradução de consultas SPARQL e possibilitar a requisição de dados nesta nova organização de dados. O mapeamento proposto permitiu a checagem da existência dos padrões de triplas fornecidas em uma consulta SPARQL e retorná-las, quando existente.

A implementação deste trabalho permitiu realizar comparações entre as diferentes foras de armazenamento de dados. Consultas aplicados sobre uma estrutura de armazenamento em única tabela de três colunas e o esquema estruturado proposto por Lima, 2016 foram comparados em relação a seu desempenho.

Durante os experimentos, observou-se que as consultas traduzidas e executadas no esquema estruturado obteve um ganho de performance de mais de 700% para os casos de testes apresentados. Entretanto, ao combinar a duração do processo de tradução de uma consulta SPARQL para SQL, de uma forma geral, houve uma perda de desempenho quando levado em consideração a duração total. Os testes também permitiram verificar

que a utilização de uma outra base de dados RDF que possui uma maior quantidade de triplas, o tempo gasto durante o mapeamento da consulta poderá se tornar irrelevante.

6.1 Trabalhos Futuros

As seguintes extensões para o modelo de mapeamento de uma consulta SPARQL para SQL e sua implementação podem ser aplicadas:

- A implementação do mapeamento de consultas pode ser realizado em uma linguagem de programação que permite um melhor desempenho computacional.
- Incluir demais funcionalidades não propostas na consulta de seleção SPARQL. Por exemplo, traduzir consultas que possuem o relacionamento de disjunção (OPTIONAL) e união (UNION) entre os padrões de triplas.
- Estender o resultado das consultas traduzidas para incluir os registros presentes nas tabelas não estruturadas da base de dados.

Referências Bibliográficas

Berners-Lee, T.; Fielding, R.; Masinter, L. *Rfc3986*. Relatório Técnico, IETF, acesso em: 21 de dez. 2016, 2005.
Disponível em https://www.ietf.org/rfc/rfc3986.txt

Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Scientific American*, p. 29–37, 2001.

Cardoso, J. The semantic web vision: Where are we? *IEEE Intelligent Systems*, v. 22, n. 5, p. 84–88, 2007.

Chebotko, A.; Lu, S.; Fatouhi, F. Semantics preserving sparql-to-sql translation. *Data Knowledge Engineering*, v. 68, n. 10, p. 973–1000, 2009.

Duerst, M.; Suignard, M. *Internationalized resource identifies(iri)*. Relatório Técnico, IETF, 2003.

Graham Klyne, Jeremy J. Carroll, B. M. *Rdf 1.1 concepts and abstract syntax*. Relatório Técnico, World Wide Web Consortium, acesso em: 21 de dez. 2016, 2014.
Disponível em http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

Harris, S.; Seaborne, A. *Sparql 1.1 query language*. Relatório Técnico, World Wide Web Consortium, acesso em: 21 de dez. 2016, 2013.
Disponível em https://www.w3.org/TR/sparql11-query/

Huang, J.; Abadi, D. J.; Ren, K. Scalable sparql querying of large rdf graphs. In: *Proceedings of the VLDB Endowment*, 2011, p. 1123–1134.

LIMA, R. *Armazenamento otimizado de dados rdf em um sgbd relacional.* Dissertação de Mestrado, Universidade Federal do Paraná, 2016.

MANOLA, F.; MILLER, E.; MCBRIDE, B. *Rdf 1.1 primer.* Relatório Técnico, World Wide Web Consortium, acesso em: 21 de dez. 2016, 2014.
Disponível em http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/

NEUMANN, T.; MOERKOTTE, G. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In: *2011 IEEE 27th International Conference on Data Engineering*, 2011, p. 984–994.

PASSIN, T. B. *Explorer's guide to the semantic web.* Manning, 2005.

PHAM, M.-D.; PASSING, L.; ERLING, O.; BONCZ, P. Deriving an emergent relational schema from rdf data. In: *Proceedings of the 24th International Conference on World Wide Web*, 2015, p. 864–874.

PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. Semantics and complexity of sparql. *ACM Transactions on Database Systems*, v. 34, n. 3, 2009.

SHADBOLT, N.; HALL, W.; BERNERS-LEE, T. The semantic web revisited. *IEEE Inteligent Systems*, v. 21, p. 96–101, 2006.

WALJI, A. Knowledge in the era of decentralization. http://blogs.worldbank.org/dmblog/knowledge-in-the-era-of-decentralization, acesso em: 21 de dez. 2016, 2010.

Apêndice

A

Consultas utilizadas nos experimentos

Este apêndice apresenta as consultas utilizadas nos testes apresentados no Capítulo 5.

A.1 Consulta C1

```
SELECT
  ?a ?p ?s
WHERE {
  ?a <http://purl.org/NET/c4dm/event.owl#place> ?p .
  ?a <http://purl.org/ontology/mo/produced_signal> ?s .
  ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t
}
```

Figura A.1: Consulta C1 em SPARQL

```
SELECT
  a.fk_produced_signal2921H7 AS s,
  a.place6S18HL AS p,
  a.OID AS a
FROM
  (SELECT
     RecordingRDF.OID,
     RecordingRDF.place AS place6S18HL,
     RecordingRDF.fk_produced_signal AS fk_produced_signal2921H7,
     RecordingRDF.type AS type5QR6KC
   FROM
     RecordingRDF
   WHERE
        RecordingRDF.place IS NOT NULL
   AND RecordingRDF.fk_produced_signal IS NOT NULL
   AND RecordingRDF.type IS NOT NULL) a;
```

Figura A.2: Consulta C1 em SQL convertida pelo algoritmo proposto

```
SELECT
  place.subj,
  place.obj,
  sinal.obj
FROM
  RDF_Tripla AS place
  INNER JOIN RDF_Tripla AS sinal ON
       sinal.pred = '<http://purl.org/ontology/mo/produced_signal>'
  AND sinal.subj = place.subj
  INNER JOIN RDF_Tripla AS typeA ON
       typeA.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeA.subj = place.subj
WHERE
  place.pred = '<http://purl.org/NET/c4dm/event.owl#place>';
```

Figura A.3: Consulta C1 em SQL para a tabela de três colunas

A.2 Consulta C2

```
SELECT
  ?s ?x ?y ?l
WHERE {
  ?s <http://purl.org/ontology/mo/published_as> ?x .
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?y
  ?s <http://www.w3.org/2000/01/rdf-schema#label> ?l
}
```

Figura A.4: Consulta C2 em SPARQL

```
SELECT
   s.label935S6X AS l,
   s.typeRFH7DY AS y,
   s.fk_published_asOJMHLF AS x,
   s.OID AS s
FROM
   (SELECT
      SignalRDF.OID,
      SignalRDF.fk_published_as AS fk_published_asOJMHLF,
      SignalRDF.type AS typeRFH7DY,
      labelMultivalueRDF.label AS label935S6X
    FROM
      SignalRDF,
      labelMultivalueRDF
    WHERE
        SignalRDF.fk_published_as IS NOT NULL
    AND SignalRDF.type IS NOT NULL
    AND labelMultivalueRDF.OID = SignalRDF.OID
    AND labelMultivalueRDF.label IS NOT NULL) s;
```

Figura A.5: Consulta C2 em SQL convertida pelo algoritmo proposto

```
SELECT
   published.obj,
   typeP.obj,
   labelP.obj
FROM
   RDF_Tripla AS published
   INNER JOIN RDF_Tripla typeP ON
       typeP.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
   AND typeP.subj = published.subj
   INNER JOIN RDF_Tripla labelP ON
       labelP.pred = '<http://www.w3.org/2000/01/rdf-schema#label>'
   AND labelP.subj = published.subj
WHERE
   published.pred = '<http://purl.org/ontology/mo/published_as>';
```

Figura A.6: Consulta C2 em SQL para a tabela de três colunas

A.3 Consulta C3

```
SELECT
   ?x ?b ?c ?d
WHERE {
   ?x <http://purl.org/dc/elements/1.1/title> ?b .
   ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c
   ?x <http://www.w3.org/2000/01/rdf-schema#label> ?d

}
```

Figura A.7: Consulta C3 em SPARQL

```
SELECT
  x.labelEQQGKL AS d,
  x.typeSXE383 AS c,
  x.title78ECW9 AS b,
  x.OID AS x
FROM
  (SELECT
     rackRDF.OID,
     TrackRDF.title AS title78ECW9,
     TrackRDF.type AS typeSXE383,
     TrackRDF.label AS labelEQQGKL
   FROM
     TrackRDF
   WHERE
       TrackRDF.title IS NOT NULL
   AND TrackRDF.type IS NOT NULL
   AND TrackRDF.label IS NOT NULL) x;
```

Figura A.8: Consulta C3 em SQL convertida pelo algoritmo proposto

```
SELECT
  title.obj,
  typeX.obj,
  labelX.obj
FROM
  RDF_Tripla AS title
  INNER JOIN RDF_Tripla typeX ON
      typeX.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeX.subj = title.subj
  INNER JOIN RDF_Tripla labelX ON
      labelX.pred = '<http://www.w3.org/2000/01/rdf-schema#label>'
  AND labelX.subj = title.subj
WHERE
  title.pred = '<http://purl.org/dc/elements/1.1/title>';
```

Figura A.9: Consulta C3 em SQL para a tabela de três colunas

A.4 Consulta C4

```
SELECT
  ?a ?p ?s ?t ?x ?y ?l
WHERE {
  ?a <http://purl.org/NET/c4dm/event.owl#place> ?p .
  ?a <http://purl.org/ontology/mo/produced_signal> ?s .
  ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t
  ?s <http://purl.org/ontology/mo/published_as> ?x .
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?y
  ?s <http://www.w3.org/2000/01/rdf-schema#label> ?l .
}
```

Figura A.10: Consulta C4 em SPARQL

```
SELECT
  s.labelE2N7BO AS l,
  s.type96JINO AS y,
  s.fk_published_asIALM8L AS x,
  a.type8UREP3 AS t,
  a.fk_produced_signalXUD5UO AS s,
  a.placeBCUY6Q AS p,
  a.OID AS a
FROM
  (SELECT
     RecordingRDF.OID,
     RecordingRDF.place AS placeBCUY6Q,
     RecordingRDF.fk_produced_signal AS fk_produced_signalXUD5UO,
     RecordingRDF.type AS type8UREP3 FROM RecordingRDF
   WHERE
       RecordingRDF.place IS NOT NULL
   AND RecordingRDF.fk_produced_signal IS NOT NULL
   AND RecordingRDF.type IS NOT NULL) a,
   (SELECT
     SignalRDF.OID,
     SignalRDF.fk_published_as AS fk_published_asIALM8L,
     SignalRDF.type AS type96JINO,
     labelMultivalueRDF.label AS labelE2N7BO
   FROM
     SignalRDF,
     labelMultivalueRDF
   WHERE
       SignalRDF.fk_published_as IS NOT NULL
   AND SignalRDF.type IS NOT NULL
   AND labelMultivalueRDF.OID = SignalRDF.OID
   AND labelMultivalueRDF.label IS NOT NULL) s
WHERE
  s.OID = fk_produced_signalXUD5UO;
```

Figura A.11: Consulta C4 em SQL convertida pelo algoritmo proposto

```
SELECT
  place.subj, place.obj, sinal.obj, typeA.obj, published.obj, typeP.obj, labelP.obj
FROM
  RDF_Tripla AS place
  INNER JOIN RDF_Tripla AS sinal ON
      sinal.pred = '<http://purl.org/ontology/mo/produced_signal>'
  AND sinal.subj = place.subj
  INNER JOIN RDF_Tripla AS typeA ON
      typeA.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeA.subj = place.subj
  INNER JOIN RDF_Tripla published ON
      published.pred = '<http://purl.org/ontology/mo/published_as>'
  AND published.subj = sinal.obj
  INNER JOIN RDF_Tripla typeP ON
      typeP.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeP.subj = published.subj
  INNER JOIN RDF_Tripla labelP ON
      labelP.pred = '<http://www.w3.org/2000/01/rdf-schema#label>'
  AND labelP.subj = published.subj
WHERE
  place.pred = '<http://purl.org/NET/c4dm/event.owl#place>';
```

Figura A.12: Consulta C4 em SQL para a tabela de três colunas

A.5 Consulta C5

```
SELECT
  ?a ?p ?s ?t ?x ?y ?l ?b ?c ?d
WHERE {
  ?a <http://purl.org/NET/c4dm/event.owl#place> ?p .
  ?a <http://purl.org/ontology/mo/produced_signal> ?s .
  ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t
  ?s <http://purl.org/ontology/mo/published_as> ?x .
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?y
  ?s <http://www.w3.org/2000/01/rdf-schema#label> ?l .
  ?x <http://purl.org/dc/elements/1.1/title> ?b .
  ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c
  ?x <http://www.w3.org/2000/01/rdf-schema#label> ?d .
}
```

Figura A.13: Consulta C5 em SPARQL

```
SELECT
  x.label0Y2ZFL AS d,
  x.typeRM5OXD AS c,
  x.titleGAWQQX AS b,
  s.labelPK1P2P AS l,
  s.typeMLP9J3 AS y,
  s.fk_published_as3ZC5YY AS x,
  a.typeUWAFOS AS t,
  a.fk_produced_signal5RIHOT AS s,
  a.placeXY862H AS p,
  a.OID AS a
FROM
  (SELECT
     RecordingRDF.OID,
     RecordingRDF.place AS placeXY862H,
     RecordingRDF.fk_produced_signal AS fk_produced_signal5RIHOT,
     RecordingRDF.type AS typeUWAFOS
   FROM
     RecordingRDF
   WHERE
       RecordingRDF.place IS NOT NULL
   AND RecordingRDF.fk_produced_signal IS NOT NULL
   AND RecordingRDF.type IS NOT NULL) a,
  (SELECT SignalRDF.OID,
     SignalRDF.fk_published_as AS fk_published_as3ZC5YY,
     SignalRDF.type AS typeMLP9J3,
     labelMultivalueRDF.label AS labelPK1P2P
   FROM
     SignalRDF,
     labelMultivalueRDF
   WHERE
       SignalRDF.fk_published_as IS NOT NULL
   AND SignalRDF.type IS NOT NULL
   AND labelMultivalueRDF.OID = SignalRDF.OID
   AND labelMultivalueRDF.label IS NOT NULL) s,
  (SELECT
     TrackRDF.OID,
     TrackRDF.title AS titleGAWQQX,
     TrackRDF.type AS typeRM5OXD,
     TrackRDF.label AS label0Y2ZFL
   FROM
     TrackRDF
   WHERE
       TrackRDF.title IS NOT NULL
   AND TrackRDF.type IS NOT NULL
   AND TrackRDF.label IS NOT NULL) x
WHERE
    s.OID = fk_produced_signal5RIHOT
AND x.OID = fk_published_as3ZC5YY;
```

Figura A.14: Consulta C5 em SQL convertida pelo algoritmo proposto

```
SELECT
  place.subj,
  place.obj ,
  sinal.obj ,
  typeA.obj ,
  published.obj,
  typeP.obj ,
  labelP.obj,
  title.obj ,
  typeX.obj ,
  labelX.obj
FROM
  RDF_Tripla AS place
  INNER JOIN RDF_Tripla AS sinal ON
      sinal.pred = '<http://purl.org/ontology/mo/produced_signal>'
  AND sinal.subj = place.subj
  INNER JOIN RDF_Tripla AS typeA ON
      typeA.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeA.subj = place.subj
  INNER JOIN RDF_Tripla published ON
      published.pred = '<http://purl.org/ontology/mo/published_as>'
  AND published.subj = sinal.obj
  INNER JOIN RDF_Tripla typeP ON
      typeP.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeP.subj = published.subj
  INNER JOIN RDF_Tripla labelP ON
      labelP.pred = '<http://www.w3.org/2000/01/rdf-schema#label>'
  AND labelP.subj = published.subj
  INNER JOIN RDF_Tripla title ON
      title.pred = '<http://purl.org/dc/elements/1.1/title>'
  AND title.subj = published.obj
  INNER JOIN RDF_Tripla typeX ON
      typeX.pred = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
  AND typeX.subj = title.subj
  INNER JOIN RDF_Tripla labelX ON
      labelX.pred = '<http://www.w3.org/2000/01/rdf-schema#label>'
  AND labelX.subj = title.subj
WHERE
  place.pred = '<http://purl.org/NET/c4dm/event.owl#place>';
```

Figura A.15: Consulta C5 em SQL para a tabela de três colunas