;

UNIVERSIDADE FEDERAL DO PARANÁ

RAFAEL DE LIMA PRADO

ARMAZENAMENTO OTIMIZADO DE DADOS RDF EM UM SGBD RELACIONAL

CURITIBA PR
2017

RAFAEL DE LIMA PRADO

ARMAZENAMENTO OTIMIZADO DE DADOS RDF EM UM SGBD RELACIONAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação.*

Orientador: Prof.ª Dr.ª Carmem Satie Hara.

CURITIBA PR
2017

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

P896a Prado, Rafael de Lima
Armazenamento Otimizado de dados RDF em um SGBD Relacional [recurso eletrônico] / Rafael de Lima Prado. – Curitiba, 2017.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2017.

Orientador: Carmem Satie Hara .

1. Armazenamento de dados. 2. Banco de dados relacionais. 3. Banco de dados – Gerência. 4. SPARQL (Linguagem de programação de computador). 5. Metadados. 6. RDF (Resource Description Framework). I. Universidade Federal do Paraná. II. Hara, Carmem Satie. III. Título.

CDD: 005.74

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

MINISTÉRIO DA EDUCAÇÃO
SETOR CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **RAFAEL DE LIMA PRADO** intitulada: **Armazenamento Otimizado de Dados RDF em um SGBD Relacional**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 29 de Janeiro de 2018.

Carmem Satie Hara

CARMEM SATIE HARA
Presidente da Banca Examinadora (UFPR)

REBECA SCHROEDER FREITAS
Avaliador Externo (UDESC)

DANIEL KASTER
Avaliador Externo (UEL)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

RESUMO

A Web Semântica tem por objetivo tornar compreensíveis as informações disponíveis na Internet para as máquinas. O RDF é o modelo de dados padrão para a Web Semântica. Bases RDF são compostas por triplas `(sujeito, predicado, objeto)` e o SPARQL é a linguagem de consultas para RDF recomendado pelo W3C. Uma consulta define um padrão de triplas a ser encontrado na base de dados. Como o objeto de uma tripla pode ser o sujeito de outra, uma consulta SPARQL pode ser vista como a busca de um subgrafo no grafo que representa a base RDF. Dada a complexidade deste problema, a otimização do acesso às bases RDF é um desafio a ser enfrentado. Algumas abordagens têm surgido nesse contexto, representando os dados RDF em outros formatos, como, por exemplo, no modelo relacional. Nesse tipo de abordagem, os dados RDF são armazenados em um Sistema Gerenciador de Banco de Dados Relacional(SBGDR). A forma direta de mapeamento de dados RDF para uma base relacional se dá por meio de uma tabela SPO, ou seja, com três atributos - sujeito, predicado e objeto. Porém, esse método em geral não possui bom desempenho, uma vez que toda a base é armazenada em uma única tabela e consultas nessa tabela que envolvem mais de um padrão de triplas implicam na execução de auto-junções. O *Armazenamento Otimizado de Dados RDF em um SGBD Relacional* - AORR, proposto nesta dissertação, surge como uma forma alternativa de armazenamento de dados RDF em um SGBDR. O intuito é que consultas sobre o AORR possuam desempenho superior às realizadas sobre as tabelas SPO oriundas da abordagem direta. Tal ganho do AORR deve-se, principalmente, à identificação de entidades com estruturas similares na base RDF e a geração de uma tabela para cada entidade no esquema relacional. Dessa forma é minimizado a quantidade de junções a serem executadas para o processamento de consultas. Consultas SPARQL podem ser realizadas sobre a base relacional gerada devido às tabelas de metadados que o AORR cria no processo de extração de esquema. As informações de mapeamento armazenada nas tabelas de metadado possibilitam a tradução de consultas SPARQL para SQL. Elas também permitem que o AORR dê suporte a atualizações incrementais da base. Os resultados dos experimentos mostram que as consultas realizadas sobre a base gerada pelo AORR apresentam melhor desempenho que uma abordagem alternativa baseada em tabelas de entidade para o armazenamento de dados RDF.

Palavras-chave: RDF. SPARQL. Metadado. SGBDR.

ABSTRACT

The main goal of the Semantic Web is to make machines understand the information available on the Internet. RDF is the Semantic Web standard data model. RDF databases consist of triples *(subject, predicate, object)*. The W3C (*World Wide Web Consortium*) recommends SPARQL as the query language for RDF. A query in RDF involves searching for triple patterns in a database. Since the object of a triple can be the subject of another, SPARQL queries can interpreted as a problem of subgraph match on the graph representing an RDF database. Given the complexity of the problem, optimizing the access to RDF databases is a hard problem. One of the possible approaches to tackle this problem is to store RDF data in a different format, for example, the relational model. A direct mapping from RDF to the relational model stores the entire database in an SPO table *(subject, predicate, object)*. However this mapping does not present a good performance because queries involving more than one triple pattern require auto-joins on this table. This dissertation proposes AORR *(Armazenamento Otimizado de Dados RDF em um SGBD Relational)* as an alternative approach to store RDF data in a Relational Database Management System (RDBMS). AORR identifies entities with similar structure in the RDF database and creates a table in the relational database for each entity. This table stores several predicates associated with the same subject and thus the amount of auto-joins to process queries is minimized. SPARQL queries can be translated to SQL on the resulting relacional database due metadata tables that AORR generates during the schema extraction process. These metadata tables also enables incremental updates of the database. The experimental results show that queries executed on AORR have better performance than an alternative approach based on entity tables for storing RDF data.

Keywords: RDF. SPARQL. Metadata. RDBMS.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Web Semântica surgiu do desejo de tornar a máquina capaz de interpretar as informações da Web, exigindo cada vez menos interação humana. Para esse fim, é necessário que haja uma padronização de como essas informações são acessadas e na identificação de entidades do mundo real. Nesse contexto, o W3C, órgão padronizador da Web, criou e definiu o RDF(*Resource Description Framework*) como o modelo de dados padrão da Web Semântica, e o SPARQL como sua linguagem de consulta.

O RDF representa os dados como um conjunto de triplas (sujeito, predicado, objeto) sem que haja a necessidade de existir um esquema pré-definido[Pham et al., 2015].

A flexibilidade do RDF, somada com a crescente quantidade de dados disponível nesse formato representada, gerou a necessidade de tornar as consultas sobre dados RDF mais eficientes. Há diversas formas de se alcançar esse objetivo, sendo uma delas mapear os dados RDF para uma base de dados relacional, fazendo uso das otimizações que um SGBDR (Sistema Gerenciador de Banco de Dados Relacional) proporciona, tais como indexação, visões materializadas, particionamento horizontal, além das otimizações sobre a linguagem de consulta SQL. Embora existam diversas propostas para o armazenamento de RDF em sua forma nativa de grafo [Zeng et al., 2013, Penteado et al., 2015], o uso de um SGBDR não deve ser descartado para conjuntos de dados de até milhões de triplas [Zeng et al., 2013].

Essa dissertação tem por objetivo desenvolver um modelo de armazenamento eficiente para RDF em um SGBDR, chamado AORR. A escolha pelo SGBDR se deu por ser uma tecnologia bem estabelecida e explorada no mercado, sendo alvo de constantes aprimoramentos.

A abordagem direta de representar RDF em uma base relacional é a inserção das triplas em uma tabela SPO, ou seja, formada por três colunas - sujeito, predicado e objeto. Porém, consultas em uma tabela SPO são custosas devido a auto-junções. Tendo em vista que uma consulta básica SPARQL é definida por um conjunto de padrões de triplas, a execução da consulta sobre uma tabela SPO requer uma auto-junção para cada par de padrões. Dado que a tabela SPO contém todas as triplas da base RDF, uma auto-junção nessa tabela seria custoso.

Exemplo 1: Considere a tabela T_{SPO} da Figura 1.1, onde são utilizados s_i e p_j para representar IRIs de sujeitos e predicados, respectivamente. Para facilitar a compreensão, uma representação equivalente na forma de um grafo é apresentada ao lado. Por exemplo, p_1 representa a IRI `http://xmlns.com/foaf/0.1/name` e p_2 representa a IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. Uma consulta para obter os

sujeito	predicado	objeto	sujeito	predicado	objeto
s1	p1	'joao'	s2	p4	s4
s1	p2	Pessoa	s3	p5	'moto1'
s1	p3	'0000-0001'	s3	p2	Automotor
s1	p3	'0000-0002'	s4	p5	'moto2'
s1	p4	s3	s4	p2	Automotor
s2	p1	'fabiana'	s5	p5	'carro1'
s2	p2	Pessoa	s5	p2	Automotor
s2	p3	'0000-0003'	s5	p7	'32784738'
s2	p3	'0000-0004'	s6	p8	'Contato de Debora'

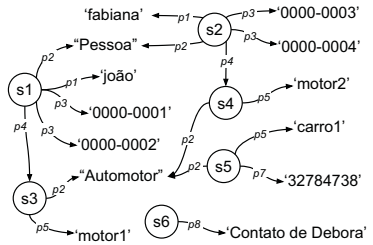


Figura 1.1: Base RDF em uma tabela SPO e sua representação em grafo

TB_FullPredicate

cs_identifier	tableName	overflowTable	columnName	predicate
CS1	PessoaRDF	<NULL>	nome	p1
CS1	PessoaRDF	<NULL>	type	p2
CS1	telefoneMultivalueRDF	<NULL>	telefone	p3
CS1	PessoaRDF	<NULL>	fk_veiculo	p4
CS2	AutomotorRDF	<NULL>	modelo	p5
CS2	AutomotorRDF	<NULL>	type	p2
CS2	<NULL>	Overflow_AutomotorRDF	cod_modelo	p7
Csover	<NULL>	Overflow	observacao	p8

TB_DatabaseSchema

cs_identifier	propertyName	valueType	tableName	tableAttribute
CS1	OID	literal	PessoaRDF	OID
CS1	nome	literal	PessoaRDF	nome
CS1	type	literal	PessoaRDF	type
CS1	veiculo	CS2	PessoaRDF	fk_veiculo
CS1	telefone	literal	telefoneMultivalueRDF	telefone
CS2	OID	literal	AutomotorRDF	OID
CS2	modelo	literal	AutomotorRDF	modelo
CS2	type	literal	AutomotorRDF	type
CS2	cod_modelo	literal	Overflow_AutomotorRDF	cod_modelo
Csover	observacao	literal	Overflow	pred

TB_Subj_OID

subj	OID	tableName
s1	1	PessoaRDF
s2	2	PessoaRDF
s3	3	AutomotorRDF
s4	4	AutomotorRDF
s5	5	AutomotorRDF
s6	6	Overflow

PessoaRDF

OID	nome	type	fk_veiculo
1	joao	Pessoa	3
2	fabiana	Pessoa	4

telefoneMultivalueRDF

OID	telefone
1	0000-0001
1	0000-0002
2	0000-0003
2	0000-0004

AutomotorRDF

OID	modelo	type
3	moto1	Automotor
4	moto2	Automotor
5	carro1	Automotor

Overflow

subj	pred	obj
6	'observacao	'Contato de Debora'

Overflow_PessoaRDF

subj	pred	obj

Overflow_AutomotorRDF

subj	pred	obj
5	'cod_modelo'	'32784738'

Figura 1.2: Exemplo de base relacional gerada pelo AORR

valores dos predicados p_1 e p_2 associados a um mesmo sujeito exigiria a execução de uma auto-junção de T_{SPO} sobre a coluna sujeito. Esta consulta pode ser expressa da seguinte forma em SPARQL: *select ?n ?t where {?s p1 ?n . ?s p2 ?t .}*, que resultaria em: *{('joao', Pessoa), ('fabiana', Pessoa)}*. □

Um dos grandes desafios enfrentados nesse trabalho é então mapear os dados de um modelo de dados que não exige esquema (RDF), para uma base de dados relacional, que exige a existência de um esquema. O método adotado pelo AORR é inspirado no sistema ERSR(*Emergent Relational Schema from RDF Data*)[Pham et al., 2015].

Exemplo 2: A base relacional gerada pelo AORR para o armazenamento das triplas na Figura 1.1 está ilustrada na Figura 1.2. Esta base contém 4 categorias de tabelas: tabelas de metadados (*Tb_FullPredicate*, *Tb_DatabaseSchema*, *Tb_Sbj_OID*), tabelas de entidades (*PessoaRDF, AutomotorRDF*), tabelas de predicados multivaloradas (*telefoneMultivalueRDF*) e tabelas de overflow. As tabelas de overflow são de dois tipos: overflow geral (*Overflow*) e overflow específico, uma para cada tabela de entidades (*Overflow_PessoaRDF, Overflow_AutomotorRDF*). O overflow geral é responsável por manter dados que não puderam ser identificados como entidades ou que gerariam tabelas muito pequenas. Em virtude da natureza semiestruturada do RDF, a tabela de overflow geral poderá conter muitos dados. Assim, as tabelas de overflow específico dividem esta carga ao manter atributos infrequentes que estão associados a uma entidade. As tabelas de metadados permitem que uma consulta SPARQL seja traduzida para SQL. Considere a consulta SPARQL do Exemplo 1. A tabela *Tb_FullPredicate* permite associar as IRIs dos

predicados $p1$ e $p2$ a uma única tabela *PessoaRDF* e às colunas *nome* e *type*, respectivamente. Assim, a consulta pode ser processada com apenas uma projeção sobre *PessoaRDF*. □

A grande diferença conceitual entre o processo de extração de esquema do AORR com relação ao ERSR consiste no propósito de utilização da base relacional gerada. Para o ERSR, a base é gerada para ser utilizada diretamente pelo usuário, e portanto a criação de nomes de tabelas e de atributos significativos é importante. No AORR, por outro lado, a finalidade é utilizar o SGBDR como *backend* de armazenamento, não havendo interação direta do usuário com a base relacional.

Além de dar suporte à tradução de consultas, as tabelas de metadados, juntamente com as tabelas de overflow, permitem que o AORR seja capaz de realizar atualizações incrementais na base.

O processo de geração da base relacional passa por três grandes etapas: extração de esquema, carga de dados e atualização da base. A etapa de extração de esquema da base relacional é baseada no processo ERSR [Pham et al., 2015]. Tanto o AORR como o ERSR são otimizados para processar consultas de busca por padrões básicos em grafos (consultas BGP, em particular consultas no formato estrela e flocos de neve [Aluç et al., 2014]). Contudo, os dois processos possuem algumas diferenças. O ERSR não considera a existência de tabelas de overflow específicas, mas apenas um overflow geral. Além disso, o ERSR não dá suporte a atualizações incrementais e não propõe o armazenamento de metadados, que dá suporte à tradução de consultas.

Nos resultados relatados pelo ERSR, o armazenamento de dados em tabelas de entidades apresentou desempenho superior no processamento de consultas de até 5 vezes, comparado a tabelas SPO [Pham et al., 2015]. Assim, na análise experimental realizada nesta dissertação foi comparado o desempenho de processamento de consultas entre o AORR e o ERSR. O AORR obteve melhor desempenho em todas as consultas consideradas, sendo o maior ganho de 198%, enquanto no pior caso houve ganho de 1,34%.

1.1 Objetivo Geral

O objetivo dessa dissertação é a utilização de técnicas para atualização de um SGBDR como *backend* de armazenamento de dados RDF e desenvolvimento de um sistema, chamado de AORR, que proporciona desempenho de consultas significativamente maior do que o da abordagem direta, e que dê suporte à atualização incremental da base.

1.2 Objetivos Específicos

Os objetivos específicos dessa dissertação são:

- desenvolver um processo de extração de esquema de um base RDF para a geração de uma base relacional;
- criar um modelo de Metadados que dê suporte à atualização incremental e suporte ao mapeamento de consultas SPARQL para consultas SQL na base relacional;
- obter melhor desempenho no processamento de consultas realizadas sobre a base relacional gerada pelo AORR, em comparação à executada em uma base gerada pela abordagem direta, ou seja, sobre uma tabela SPO.

1.3 Estrutura da Dissertação

A dissertação está organizada da seguinte forma: O Capítulo 2 apresenta o modelo RDF, a linguagem de consultas SPARQL, e a definição de metadados, sem os quais não seria possível a transformação de consultas SPARQL em consultas SQL no AORR. O Capítulo 3 apresenta alguns trabalhos relacionados, dentre os quais o ERSR; cujo método de extração de esquema relacional inspirou o processo adotado pelo AORR; por fim, discutem-se os trabalhos apresentados comparando-os com o AORR. O Capítulo 4 apresenta o AORR, apresentando inicialmente as diferenças entre o AORR e o ERSR, juntamente com o detalhamento do processo de extração de esquema, inserção de dados na base relacional gerada e atualização incremental. O Capítulo 5 apresenta a nomenclatura adotada para as tabelas e colunas geradas pelo AORR. O Capítulo 6 apresenta os testes realizados sobre a base relacional gerada pelo AORR. O Capítulo 7 apresenta a conclusão.

Capítulo 2

RDF

Esse capítulo começa definindo o que é Web Semântica. Logo em seguida é abordada a definição de RDF e como os dados RDF estão organizados. Na sequência, consultas SPARQL são descritas e exemplificadas. Por fim, apresenta-se o que são metadados e a importância deles para o AORR.

2.1 *O Framework RDF*

A Web Semântica tem por objetivo tornar os dados da Web interpretáveis para a máquina, ou seja, tornar o computador capaz de identificar dados na Web, e conseguir diferenciar conteúdos de domínios distintos como por exemplo, conteúdos esportivos e educacionais. É nesse contexto que surgiu o RDF[Dias, 2001].

O RDF(*Resource Description Framework*) foi criado e definido pela W3C, órgão padronizador da Web, como o modelo de dados padrão para representar informações na Web Semântica [Cyganiak et al., 2014]. Uma base RDF consiste em um conjunto de triplas <sujeito, predicado, objeto>, que podem ser representados na forma de um grafo, com dois nodos e uma aresta direcionada do nodo A para o nodo B, como exemplificado na Figura 2.1.

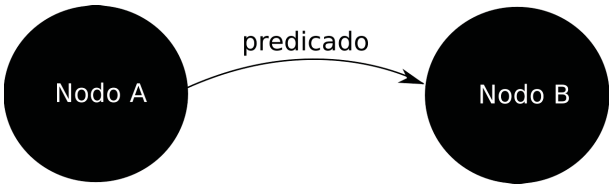


Figura 2.1: Tripla de um grafo RDF

Pela Figura 2.1, o nodo A é o 'sujeito', o nodo B é o 'objeto' e a aresta de A para B corresponde ao 'predicado'. O 'predicado' indica um relacionamento entre dois recursos, no caso, 'sujeito' e 'objeto', e é considerada uma IRI(*Internationalized Resource Identifier*). Já os nodos podem ser de três tipos: IRI, literal ou *blank node*.

Todo literal ou IRI é considerado um recurso na Web. Um recurso pode ser qualquer coisa, de conceitos abstratos a entidades físicas, como dinheiro, estabelecimento, 'José', protesto, '*WallStreet*', etc.

2.1.1 Nodos

Uma IRI é uma string UNICODE que é considerada única dentro de um grafo RDF [Cyganiak et al., 2014]. Sendo assim, a partir da IRI é possível identificar de forma única e universal um recurso na Web Semântica[Laufer, 2015].

Literais são constantes usados para valores de strings, datas e números. Os literais RDF possuem três componentes: o léxico, que é o único componente obrigatório, corresponde ao valor literal, representado por uma string UNICODE; o tipo de dado, componente opcional que representa o domínio do valor, que pode ser numérico, data, etc. Caso o literal seja uma string, ele pode ter um terceiro componente, que é uma *tag* especificando a linguagem.

Os *blank nodes* podem ser considerados como variáveis que denotam a existência de um recurso sem nome[Hogan et al., 2014]. Ou seja, não possuem uma IRI para identificar o recurso[Hayes e Patel-Schneider, 2014]. Não há referência no RDF para qualquer estrutura interna dos *blank nodes*[Cyganiak et al., 2014].

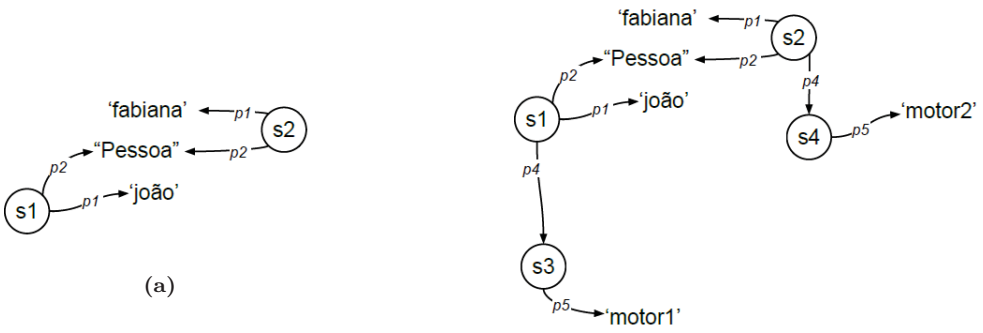
2.2 SPARQL

SPARQL é a linguagem de consulta padrão estipulada pela comunidade internacional W3C para RDF[Prud'hommeaux et al., 2008]. Basicamente, SPARQL é uma linguagem de casamento de subgrafos[Pérez et al., 2009]. Cada subgrafo da base RDF que casa com o padrão do subgrafo da consulta gera uma parte do resultado.

A maioria das consultas SPARQL apresenta um conjunto de padrões de triplas denominados *basic graph patterns* - padrões básicos de grafo. Os padrões de tripla são semelhantes às triplas do RDF, com a característica de que o sujeito, o predicado e o objeto podem ser variáveis[Prud'hommeaux et al., 2008]. Toda consulta SPARQL possui três partes:

- parte de padrão de consulta: inclui funcionalidades ao padrão de consulta como união entre padrões, partes opcionais, filtragem de possíveis combinações de valores e a possibilidade da escolha do prefixo do dado a ser combinado (Ex: PREFIX foo: <http://example.com/resources/>) .
- modificadores da solução: uma vez que obtido os resultados buscados, permite operadores como *distinct*, projeções e *limit*.
- saída: o resultado pode ser de vários tipos como descrição de recursos, construção de um novo RDF e seleção dos valores das variáveis com os padrões de tripla combinados[Pérez et al., 2009].

Dentre os padrões de grafos, o padrão estrela e o floco de neve merecem destaque, dado que o AORR é otimizado para esses padrões. A consulta SPARQL *select ?n ?t where {?s p1 ?n . ?s p2 ?t .}* apresenta um padrão estrela(a), e a consulta *select ?n ?t ?j where {?s p1 ?n . ?s p2 ?t . ?s p4 ?u . ?u p5 ?j .}* apresenta o padrão floco de neve(b), como pode ser visto no grafo abaixo:



(a) (b)

2.3 Armazenamento de dados RDF

O RDF é um modelo de dados abstrato podendo ser representado de diversas formas, desde que suas características continuem sendo representadas como descrito pelo modelo de [Laufer, 2015]. Alguns modelos de armazenamento de dados são: XML, Grafo, JSON e Tabelas. Para esse trabalho, o modelo de armazenamento de dados RDF adotado foi o de tabelas, por ser mapeado para uma base de dados relacional.

A escolha pelo SGBDR (Sistema Gerenciador de Banco de Dados Relacional) se deu por ser uma tecnologia bem estabelecida no mercado, facilitando a integração com outras bases, e pelas otimizações internas dos SGBDRs para as consultas SQL.

Um SGBDR necessita de um esquema pré-definido para dar suporte à consulta e armazenamento dos dados, ou seja, só é possível inserir ou consultar qualquer dado em um SGBDR se houver conhecimento prévio da tabela e seus atributos, '*schema first*'. Todavia, uma base RDF não necessariamente contém informações de esquema.

A abordagem direta de armazenamento do RDF em base relacional é feita a partir de uma tabela SPO, ou seja, uma tabela com três colunas - sujeito, predicado e objeto. Essa abordagem não foi a adotada, uma vez que consultas SPARQL sobre a tabela SPO tendem a ser custosas. Para cada par de padrão de consulta SPARQL, uma auto-junção precisa ser feita na tabela SPO.

Sendo assim, o primeiro desafio para o armazenamento de dados RDF em uma base relacional, é a geração de um esquema relacional. Para que o AORR seja capaz de transformar consultas SPARQL em consultas SQL, são necessárias tabelas de metadados sobre o esquema da base gerada.

2.3.1 Metadados

A definição mais comumente usada para metadados é a oriunda de sua própria etimologia, ou seja, metadados são 'dados sobre os dados'[Kowata, 2011].

Para um SGBDR, por exemplo, alguns metadados podem ser descritos pelo seu esquema, como nome e tipo dos atributos, nome da relação e chave estrangeira, [Kowata, 2011]. Como exemplo, considere a tabela *PessoaRDF* da Figura 1.2. Alguns metadados sobre a tabela *PessoaRDF* seriam:

- nome da tabela: 'PessoaRDF'
- nome e tipo dos atributos: 'OID' - inteiro, '*name*' - texto, '*type*' - texto, '*fk_veiculo*' - inteiro
- chaves estrangeiras: 'fk_veiculo'

O AORR cria um conjunto de tabelas de metadados de acordo com a extração de um esquema da base RDF. Essas tabelas são cruciais para que o AORR possa dar suporte à atualização incremental e à consultas sobre a base relacional.

Capítulo 3

Trabalhos Relacionados

Esse capítulo começa mencionando a crescente necessidade de investigar novas abordagens para obter um bom desempenho no processamento de consultas em bases RDF. Em seguida são apresentados alguns trabalhos de armazenamento de dados RDF em base relacional. A Seção 3.3 explica de forma detalhada o processo de extração de esquema proposto por [Pham et al., 2015]. Por fim, há uma discussão sobre as diferenças entre a abordagem AORR e os trabalhos mencionados.

3.1 Contexto

Com o crescimento da Web Semântica, novas abordagens de acesso otimizado aos dados RDF têm surgido. Algumas propostas adotam o mapeamento dos dados RDF para SGBDs relacionais(SGBDR). Há diversas razões para tal escolha, como o fato da tecnologia relacional ser bem estabelecida no mercado, e a maturidade dos otimizadores de consultas dos SGBDR [Bornea et al., 2013].

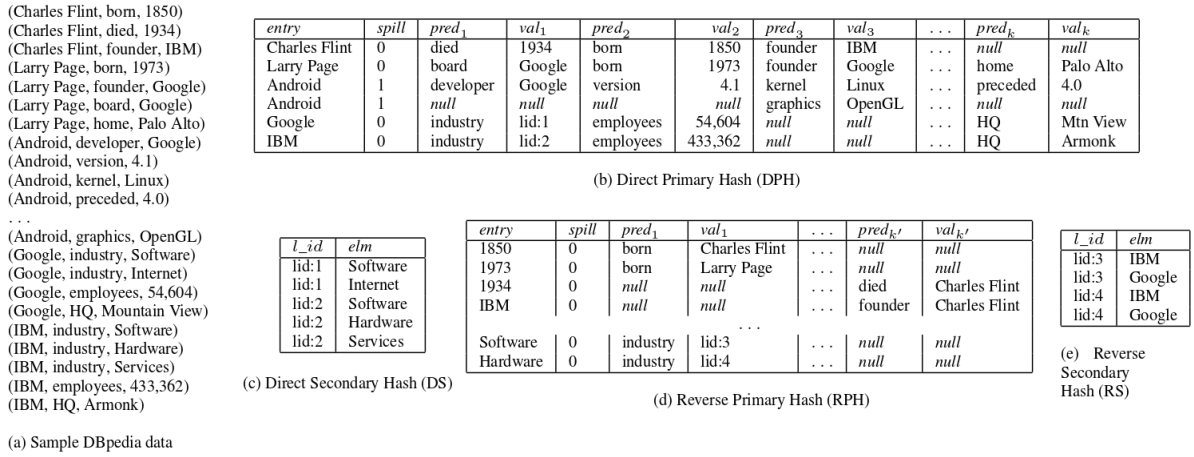
Existem abordagens que adotam métodos pautados unicamente nas propriedades, como [Abadi et al., 2007], que geram uma tabela por propriedade distinta da base. Outros possuem uma abordagem própria, como [Bornea et al., 2013], que a denominou como orientada a entidades. Semelhante a essa, [Scabora et al., 2017] também adota a ideia de um 'sujeito' para um conjunto de colunas 'k', com o intuito de diminuir o volume de dados da base, otimizando as consultas. Há também aquelas que realizam a extração de um esquema da base RDF, como [Pham et al., 2015] e [Ramunajam et al., 2009].

Aspectos interessantes a serem levados em conta para tais abordagens são, por exemplo, se a base relacional possui suporte à inserção de novos dados, e se suporta consultas SPARQL.

3.2 Abordagens

O trabalho de [Bornea et al., 2013] apresenta uma abordagem orientada a entidades (*entity-oriented*) de armazenamento de dados RDF em base relacional. O esquema gerado é chamado de DB2RDF *schema*. Essa abordagem considera a criação de quatro tabelas pré definidas de acordo com a quantidade de propriedades distintas de cada sujeito da base RDF - DPH, RPH, DS e RS. A Figura 3.1, extraída de [Bornea et al., 2013], exemplifica tais tabelas. Observa-se que as tabelas DS e RS são necessárias apenas para tratar atributos multivalorados, enquanto as tabelas DPH e RPH consideram os atribu-

tos monovalorados. A tabela DPH contém um registro para cada sujeito da base RDF, enquanto a RPH contém um registro para cada objeto. Percebe-se que tanto na DPH, quanto na RPH, há uma quantidade fixa de pares de atributos '$pred_k$', 'val_k'. O valor k para DPH e k' para RPH são definidos no artigo de acordo com verificações na base RDF. O grande problema dessa abordagem está nos *spill*. Um *spill* acontece quando, no caso da DPH, um sujeito possui mais do que k propriedades distintas, sendo então necessário inserir mais um registro com esse sujeito, como pode ser visto na Figura 3.1(b), para o sujeito *Android*.



entry	*spill*	$pred_1$	val_1	$pred_2$	val_2	$pred_3$	val_3	...	$pred_k$	val_k
Charles Flint	0	died	1934	born	1850	founder	IBM	...	*null*	*null*
Larry Page	0	board	Google	born	1973	founder	Google	...	home	Palo Alto
Android	1	developer	Google	version	4.1	kernel	Linux	...	preceded	4.0
Android	1	*null*	*null*	*null*	*null*	graphics	OpenGL	...	*null*	*null*
Google	0	industry	lid:1	employees	54,604	*null*	*null*	...	HQ	Mtn View
IBM	0	industry	lid:2	employees	433,362	*null*	*null*	...	HQ	Armonk

l_id	*elm*
lid:1	Software
lid:1	Internet
lid:2	Software
lid:2	Hardware
lid:2	Services

entry	*spill*	$pred_1$	val_1	...	$pred_{k'}$	$val_{k'}$
1850	0	born	Charles Flint	...	*null*	*null*
1973	0	born	Larry Page	...	*null*	*null*
1934	0	*null*	*null*	...	died	Charles Flint
IBM	0	*null*	*null*	...	founder	Charles Flint
			...			
Software	0	industry	lid:3	...	*null*	*null*
Hardware	0	industry	lid:4	...	*null*	*null*

l_id	*elm*
lid:3	IBM
lid:3	Google
lid:4	IBM
lid:4	Google

Figura 3.1: Exemplo de tabelas orientadas a entidades[Bornea et al., 2013]

Devido à flexibilidade das tabelas DPH e RPH, a inserção de novas triplas nessa abordagem tende a ser direta. Além disso, o trabalho possui um processo de otimização de consultas SPARQL, e de mapeamento delas para consultas SQL sobre essa base.

No trabalho de [Ramunajam et al., 2009] o principal objetivo é extrair um esquema relacional da base RDF. O principal critério para se criar uma tabela nesse modelo se baseia nos objetos das triplas com propriedade do tipo "rdfs::class". A ideia é que seja criado apenas um mapeamento de RDF para uma base relacional, e não a base propriamente dita. Sendo assim, consultas SQL realizadas sobre a base relacional virtual são mapeadas para SPARQL, e assim realizada sobre a base RDF.

[Abadi et al., 2007] apresentam uma abordagem orientada a propriedades, gerando uma tabela para cada propriedade distinta na base RDF. Essas tabelas contêm apenas dois atributos, onde um contém o sujeito, e o outro o objeto da tripla. Uma vantagem dessa abordagem é que as tabelas não contêm valores nulos pois, se um sujeito não possuir uma determinada propriedade, basta não inserir o sujeito na tabela. Inserções nessa base são diretas, ou seja, verifica-se a propriedade da tripla a ser inserida, busca-se a tabela respectiva à propriedade, e insere-se o sujeito e objeto na tabela. Todavia, a proposta não dá suporte ao mapeamento de consultas SPARQL em consultas SQL.

[Scabora et al., 2017] mapeiam as triplas RDF para uma única tabela na base relacional. Diferentemente da abordagem direta, o trabalho adota a ideia de múltiplas colunas. Cada linha da tabela corresponde a um vértice (sujeito), desde que a quantidade de arestas desse vértice seja inferior a 'k'. Caso a quantidade seja superior a 'k', um novo registro do mesmo vértice é inserido. A variável 'k' é a quantidade do par de colunas (vértice, peso_aresta), sendo esse definido de modo customizável e otimizado. Tal abordagem diminui o volume total da base, o que resultou em ganho no desempenho de consultas.

[Pham et al., 2015] propõem mapear os dados RDF para uma base relacional a partir da extração de um esquema, o que é chamado de *emergent schema*. Segundo o artigo, a abrangência do esquema gerado é superior a 90%. Porém, consultas nessa base são em SQL, não havendo um processo explícito de transformação de consulta SPARQL em SQL. Além disso, não há menção à atualização dessa base. No caso, a inserção de uma nova tripla sobre a base RDF implicaria na necessidade de gerar novamente a base relacional.

O AORR tem por base a extração de esquema de dados RDF proposto por [Pham et al., 2015], gerando assim tabelas de entidades. Por esse motivo o ERSR é apresentado em detalhe na próxima seção.

3.3 *Emergent Relational Schema from RDF*

A abordagem adotada no trabalho de [Pham et al., 2015], nesta dissertação chamada de ERSR, tem por objetivo mapear os dados RDF para uma base de dados relacional composta de tabelas que agrupem os predicados associados a uma mesma entidade. Tais tabelas são criadas a partir de um processo de extração de esquema da base RDF.

Busca-se com o esquema extraído diminuir a quantidade de auto-junções nas consultas quando a base é armazenada em uma tabela SPO e assim melhorar o desempenho no processamento de consultas. Com um esquema estabelecido é possível explorar também as otimizações de consultas SQL e auxiliares da base relacional como, por exemplo, indexação de colunas, visões materializadas e particionamento de tabelas.

O esquema é gerado baseado na identificação de similaridades estruturais dos dados RDF. Os dados que não se adéquam ao esquema extraído, ou seja, às tabelas de entidades geradas, são mapeados para uma tabela SPO na base relacional, denominada de Overflow. Assim, a base relacional resultante do processo é composta por um conjunto de tabelas de entidades, que agrupam em uma mesma tupla as propriedades comumente encontradas para os sujeitos que compõem a base. As demais propriedades são armazenadas na tabela de Overflow. A obtenção do esquema das tabelas de entidade é baseada no conceito de *characteristic sets* (CS). Um CS é caracterizado por um conjunto de propriedades. Assim, um sujeito pertence a um CS c se possui o conjunto de propriedades que caracteriza c.

Como exemplo, os sujeitos da Figura 3.2(a) pertentem a três CSs distintos, como consta na Figura 3.2(b). Nota-se que os sujeitos 's1' e 's2' possuem o mesmo conjunto de predicados 'comprou', 'nome', configurando então o 'cs1'; o sujeito 's3' possui o conjunto 'comprou', correspondendo ao 'cs2'; e os sujeitos 's3' e 's4', com o conjunto 'cor', 'type', formam o 'cs3'.

O processo de geração do esquema relacional proposto pelo ERSR, possui três parâmetros de configuração:

- Ub_{tbl}: quantidade máxima de tabelas geradas.
- min_t: quantidade mínima de registros em um tabela.
- T_{inf}: limite de infrequência, ou seja, o porcentual mínimo de sujeitos que devem possuir uma propriedade para que ela seja incluída em uma tabela de entidade.

Há cinco passos para a extração do esquema e carregamento dos dados para a base relacional: identificação dos CSs básicos, atribuição de rótulos aos CSs, junção, filtragem de esquema e filtragem de instância.

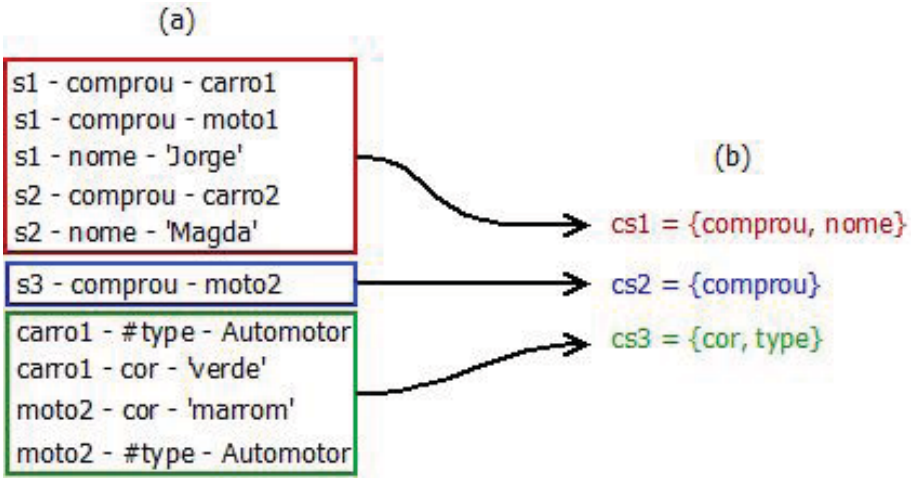


Figura 3.2: Exemplo de CS

3.3.1 Identificação dos CSs Básicos

O processo de extração do esquema relacional envolve operações sobre CSs. Assim, para não haver ambiguidade e facilitar a discussão, os CSs identificados diretamente da base RDF serão denominados de *CSs Básicos* (CSB).

Nesta primeira fase do processo, além da identificação dos CSBs, são extraídas outras informações que são utilizadas nas fases posteriores do processo de geração do esquema. A primeira estrutura associa cada CSB com a quantidade de sujeitos pertencentes a ele, como ilustrado na Figura 3.3(a). A segunda associa cada predicado literal ao seu tipo e à quantidade de predicados literais desse tipo, conforme exemplificado na Figura 3.3(b). A terceira associa triplas de predicados não literais, substituindo os sujeitos e objetos por seus respectivos CSBs, com a quantidade de triplas de igual configuração. Segue um exemplo dessa estrutura na Figura 3.3(c). A quarta estrutura, como pode ser visto na Figura 3.3(d), associa cada sujeito ao CSB em que se enquadra. A Figura 3.3 tem por base as triplas e os CSBs apresentados na Figura 3.2.

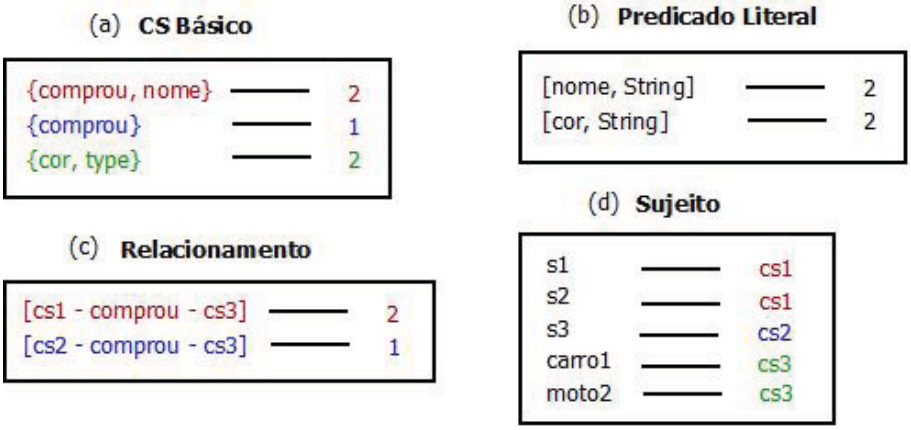


Figura 3.3: Tabelas geradas Identificação CS Básico

3.3.2 Atribuição de Rótulos aos CSs

Como o ERSR tem como objetivo gerar uma base relacional para ser consultada diretamente pelos usuários, busca-se atribuir rótulos curtos e representativos para os CSBs.

As atribuições podem ser definidas por características semânticas ou estruturais do CSB. Preferencialmente, atribuem-se rótulos de acordo com a classe ontológica associada ao CSB. Seguem os métodos utilizados para atribuição dos rótulos, em ordem de prioridade:

1. **Propriedade *Type***: A propriedade *Type* atribui uma classe ao sujeito. O objeto dessa propriedade é uma IRI que define uma classe ontológica. Logo, um rótulo com o nome da classe ontológica ao qual o CSB faz parte é de grande pertinência.

   Inicialmente, descartam-se os *Type* com frequência inferior a T_{inf}. Para os restantes é definida sua *incidência*, de acordo com a seguinte fórmula:

   $$incidencia(CSB, Type) = \frac{\%quantidadeSujeitosTypeCSB}{\%quantidadeSujeitosTypeBase} \tag{3.1}$$

   Aqui, *%quantidadeSujeitosTypeCSB* representa a porcentagem de sujeitos com a propriedade *Type* contidas no CSB, e *%quantidadeSujeitosTypeBase* denota a porcentagem de sujeitos com a propriedade *Type* em toda a base RDF.

   O CSB recebe o rótulo do *Type* de maior *incidencia*.

2. **Propriedade Discriminativa**: Existem propriedades que são características de uma classe ontológica, que são consideradas propriedades discriminativas. Busca-se então identificar uma classe ontológica cuja propriedade discriminativa pertence a um CSB. Por exemplo, dado um CSB com as propriedades *hasUnitOfMeasurement* e *valueAddedTaxIncluded*, é possível perceber que as mesmas propriedades constam na classe *PriceSpecification* da ontologia *GoodRelations*[Pham et al., 2015].

   Tendo encontrado essa propriedade, é calculada a similaridade entre o CSB e a classe ontológica referente. Se a similaridade for maior do que o T_{inf}, então o CSB recebe o rótulo dessa classe. Segue a função de similaridade adotada pelo ERSR:

   $$tfidf(p, cs) = \frac{1}{\|D_p(cs)\|} \; X \; log \frac{total\#CSs}{1 + \#containedCSs(p)} \tag{3.2}$$

   $$sim_{ij} = \frac{\sum_{p \in (cs_i \cap cs_j)} tfidf(p, cs_i) \; X \; tfidf(p, cs_j)}{\sqrt{\sum_{p_i \in cs_i} tfidf(p_i, cs_i)^2} \; X \; \sqrt{\sum_{p_j \in cs_j} tfidf(p_j, cs_j)^2}} \tag{3.3}$$

3. **Relacionamento entre CSs**: A atribuição de rótulo por relacionamento se dá de acordo com a propriedade que mais referencia o CSB em questão. Por exemplo, considere um CSB_y que não se enquadra nas duas atribuições de rótulo anteriores. Neste caso, buscam-se todos os CSB que possuem o CSB_y como objeto. O CSB_y terá como rótulo a propriedade que mais se referiu a ele. Segue um exemplo na Figura 3.4, o CSB_a é predominantemente referenciado pela propriedade 'Lugar', recebendo assim esse rótulo.

4. **Encurtar a IRI**: Caso não seja possível encontrar um rótulo com os três métodos anteriores, o CSB recebe o rótulo da propriedade mais fre-

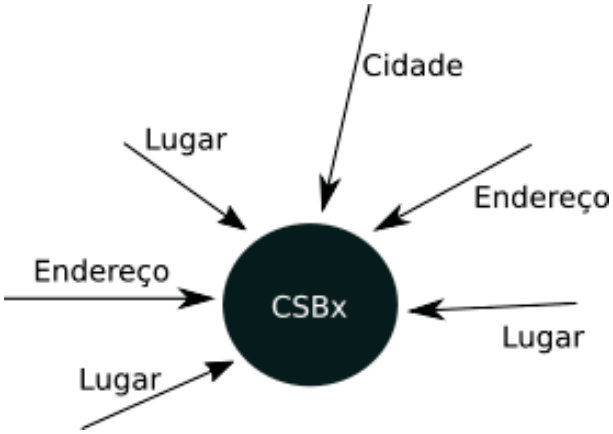


Figura 3.4: Exemplo de CSB sendo referenciado

quente, realizando apenas a remoção do prefixo ontológico, por exemplo `http://purl.org/ontology/mo/performed` é reduzido para `performed`.

O resultado da fase de Atribuição de Rótulos aos CSs é um conjunto de *CSs Table*(CST). Cada CST é potencialmente um candidato a gerar uma tabela no modelo relacional, cujo nome é o rótulo atribuído e o conjunto de atributos consiste dos predicados no CSB.

No artigo [Pham et al., 2015] não há a distinção de CSB e CST. A fim de facilitar a compreensão, foi criada a denominação de CS Tabela (CST) como a estrutura que possui rótulo e pode comportar vários CSBs, resultante da fase de junção, como detalhado na próxima seção.

3.3.3 Junção

Essa fase tem por propósito diminuir a quantidade total de CSTs, agrupando dados similares. As junções podem ser categorizadas como semânticas e estruturais, havendo duas regras para cada categoria.

As regras de junção semânticas são:

1. Juntam-se CSTs de mesmo rótulo se ambos os rótulos foram atribuídos pela regra de **Propriedade *Type*** ou pela regra de **Propriedade Discriminativa**.

2. Juntam-se CSTs de um mesmo ancestral ontológico. Se uma classe ontológica O_s é uma subclasse de uma outra classe ontológica O_c, então O_c é um ancestral ontológico de O_s. Sendo assim, se os dois CSTs têm seus rótulos atribuídos pelas regras de **Propriedade *Type*** ou pela de **Propriedade Discriminativa**, ou seja, por uma classe ontológica, se possuírem um mesmo ancestral ontológico e se o resultado da Equação 3.4 for superior a $\frac{1}{Ub_{tbl}}$, então é feita a junção dos dois CSTs, e o rótulo do CST formado muda para o do ancestral ontológico.

$$g_{score}(O_c) = \frac{\%instances_c overedby(O_s)}{\%instances_c overedby_o ntology} \tag{3.4}$$

As regras de junção estruturais são:

1. Juntam-se CSTs referenciados com a mesma propriedade a partir do mesmo CST origem. Dado um cst_x de propriedade p que referencia um cst_y, e a mesma propriedade, com origem também em cst_x, referencia cst_z, temos que se $\frac{ref(cst_x,p,cst_y)}{freq(cst_x)}$ e $\frac{ref(cst_x,p,cst_z)}{freq(cst_x)}$ forem maiores que Tinf, acontece a junção de cst_y e cst_z. Aqui, $ref(cst_x, p, cst_y)$ refere-se à quantidade de sujeitos de cst_x com predicado p que referenciam cst_y, e $freq(cst_x)$ denota a quantidade de sujeitos de cst_x.

2. Juntam-se CSTs se a função de similaridade entre eles resultar em um score superior a Tinf. A função de similaridade foi apresentada na Equação 3.3.

Por exemplo, Figura 3.5 apresenta as quatro possíveis junções, sendo representado em verde a primeira junção semântica, em azul a segunda junção semântica, em vermelho a primeira junção estrutural e em rosa a segunda junção estrutural.

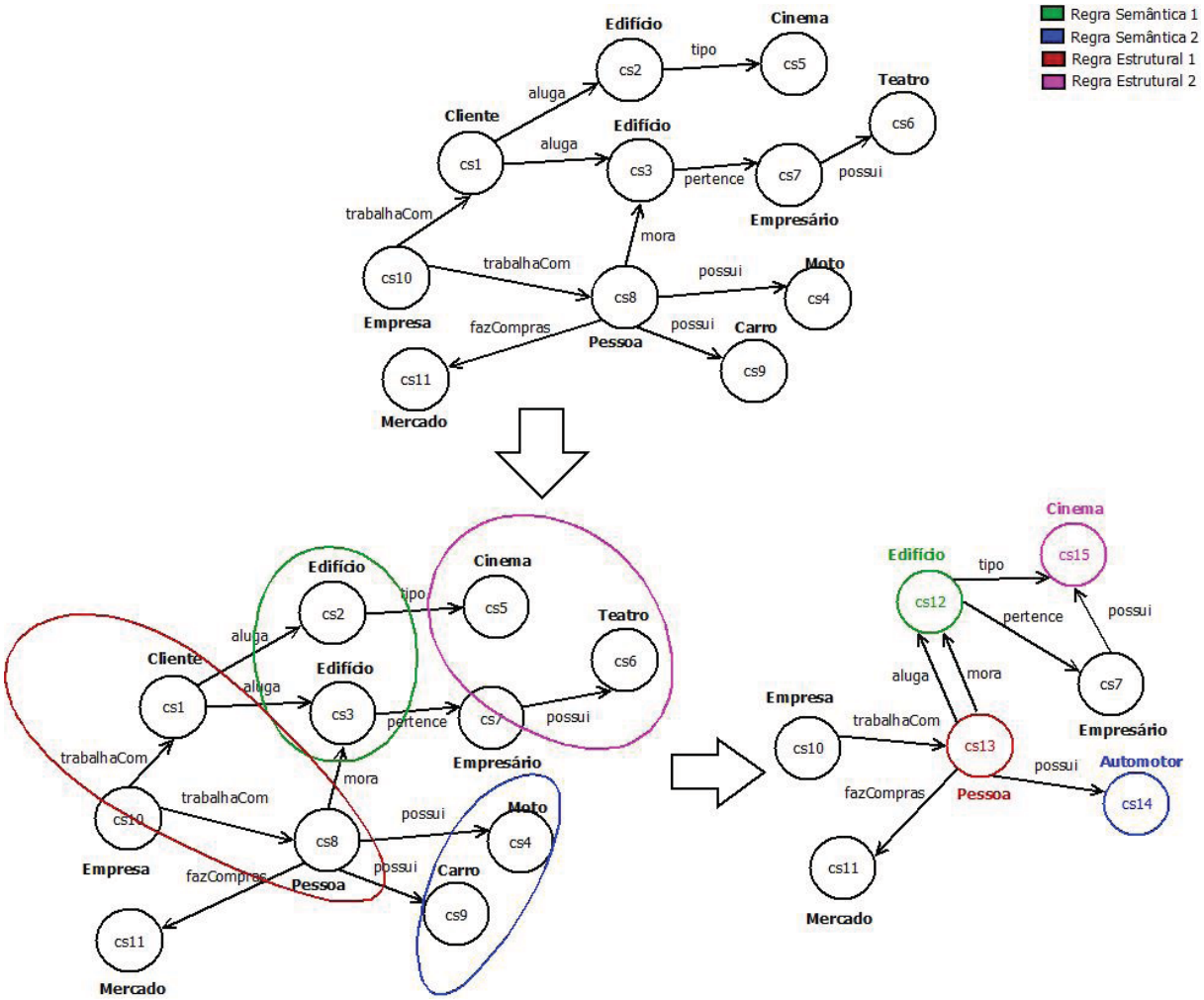


Figura 3.5: Exemplo de junção

3.3.4 Filtragem de Esquema

Após a junção dos CSTs, segue-se à etapa de filtragem de esquema, onde os CSTs menos significativos são migrados para a tabela Overflow. Inicialmente, verifica-se se cada CST possui a quantidade mínima de sujeitos (min_t), que é um parâmetro de configuração, sendo migrado para a tabela de Overflow os que não satisfizerem tal condição.

Verifica-se então se a quantidade de CSTs é maior do que o máximo de tabelas permitido (Ub_{tbl}). Caso seja maior, os CSTs com menor quantidade de sujeitos também são migrados para o Overflow.

Tabelas Dimensionais

As tabelas dimensionais constituem a única ressalva a respeito de CSTs que, mesmo não tendo muitos sujeitos, não são migrados para o Overflow.

Tabelas dimensionais são frequentemente referenciadas por outras tabelas. Sendo assim, é interessante que elas permaneçam como parte do esquema.

O algoritmo de identificação das tabelas dimensionais é o *PageRank* [Pham et al., 2015], que é aplicado sobre o grafo formado por todos os CST (vértices) e relacionamentos (arestas). Em cada iteração do algoritmo, a Equação 3.5 é aplicada. Na equação, $IR_k(cs_i)$ representa o score de referenciamento indireto de cs_i após k iterações, $ref(cs_j, cs_i)$ é o número de referências de cs_j para cs_i, $freq(cs_j)$ a frequência de cs_j, e $refsTo(cs_i)$ é a quantidade total de referências diretas ao cs_i.

$$IR_k(cs_i) = \sum_{cs_j \to cs_i} IR_{k-1}(cs_j) \times \frac{ref(cs_j, cs_i)}{refsTo(cs_i)} \times \frac{ref(cs_j, cs_i)}{freq(cs_j)} + refsTo(cs_i) \tag{3.5}$$

Os CSTs com $IR_k(cs_i) \geq Ub_{tbl}$ são considerados como tabelas dimensionais, portanto são mantidos no esquema gerado.

Minimização de Propriedades Infrequentes

Com as junções de CSTs, várias propriedades podem possuir baixa frequência, o que ocasionaria colunas com uma grande quantidades de valores nulos. Assim é interessante que essas propriedades sejam colocadas no Overflow.

Uma propriedade é considerada infrequente quando o $coverageRatio(p, cs)$ é inferior a T_{inf}. Na Equação 3.6, *freq(p,cs)* é a quantidade de sujeitos com predicado 'p' no CST, e *freq(cs)* é a quantidade de sujeitos total do CST.

$$coverageRatio(p, cs) = \frac{freq(p, cs)}{freq(cs)} \tag{3.6}$$

3.3.5 Filtragem de Instância

Finalizada a etapa de Filtragem de Esquema, pode-se considerar que o esquema da base relacional está definido. Normalmente, o esquema possui uma abrangência superior a 90% da base RDF, de acordo com experimentos relatados em [Pham et al., 2015]. O processo a ser realizado agora é o de inserção das instâncias na base relacional. Porém há algumas instâncias consideradas irregulares que precisam ser tratadas.

Maximização da Homogeneidade

Há casos em que uma mesma propriedade de objeto literal está definida com mais de um tipo. Um exemplo seria a propriedade nascimento com um conjunto de

objetos literais deste tipo "2015"^^#Integer - (inteiro), e outro conjunto de objetos do tipo "2015"^^#Date - (data).

A forma de lidar com esses casos irá depender da frequência de cada tipo no CST. Se a quantidade de propriedades de um tipo, calculada pela função $\sum_{p \in cs_i} \#ofTypes(p)$, for inferior a Tinf, então as triplas com esse tipo de objeto são consideradas infrequentes, e portanto migradas para o Overflow. Para cada tipo de objeto literal superior a Tinf, uma nova coluna desse tipo é adicionada à tabela.

Filtragem de Relacionamentos

São migrados para a tabela Overflow todos os relacionamentos entre dois CSTs, CST_x e CST_y, se eles não possuírem a frequência mínima(Tinf) de sujeitos com tal relacionamento. Se a maioria dos registros de CST_x tiverem pelo menos um relacionamento com CST_y, então apenas os que não possuem tal relacionamento são migrados para a tabela Overflow. Do contrário, apenas a coluna que representa o relacionamento entre os CSTs é migrada para o Overflow.

Atributos Multivalorados

Um atributo é chamado de multivalorado se, para um registro, o atributo referencia mais de um valor, resultando em uma cardinalidade superior a 1. Nesses casos, se a propriedade 'p' gerar um $meanp(p)$, Equação 3.8, inferior a $(1 + \frac{T_{inf}}{100})$, ela é considerada multivalorada. Na Equação 3.7, 'k' indica a cardinalidade, *'freq(p)'* a quantidade de sujeitos com a propriedade 'p' e '#times_p_has_k_object_values' a quantidade de vezes que a propriedade 'p' possui 'k' objetos.

Se o atributo for multivalorado, uma tabela é criada para comportar tais relações. Do contrário, o atributo é considerado monovalorado e os outros valores são migrados para a tabela Overflow.

$$p(k) = \frac{\#times\_p\_has\_k\_object\_values}{freq(p)} \tag{3.7}$$

$$meanp(p) = \sum p(k) \times k \tag{3.8}$$

3.4 Discussão

A proposta de [Bornea et al., 2013], ao contrário do AORR, não realiza a extração de um esquema da base RDF, mas adapta seu esquema de acordo com a quantidade de predicados distintos de cada sujeito. Apesar de dar suporte à atualização incremental, ela não é tratada diretamente, podendo comprometer o desempenho da proposta com muitas inserções de registros *spill*.

Semelhante à abordagem de [Bornea et al., 2013], [Scabora et al., 2017] também não realiza extração de esquema, mas cria um esquema baseado no peso atribuído a cada aresta. Diferente do AORR, o trabalho não trata atualização incremental.

Já a abordagem de [Abadi et al., 2007], por criar uma tabela para cada predicado distinto da base, facilmente atinge uma alta quantidade de tabelas. O AORR, a partir do processo de extração de esquema, busca gerar tabelas com base em uma quantidade mínima de registros, e realizar junções de tabelas quando é notado que possuem alta similaridade. Logo, o AORR tende a gerar menos tabelas do que a abordagem de [Abadi et al., 2007].

Apesar dessa proposta dar suporte à atualizações incrementais, ela não dá suporte ao mapeamento de consultas SPARQL para consultas SQL sobre a base relacional gerada, o que difere do AORR.

[Ramunajam et al., 2009] apresentam uma forma diferente de extração de esquema. Enquanto o [Ramunajam et al., 2009] possuem a preocupação de criar um mapeamento de RDF para relacional que comporte todos os dados, com base principalmente nas propriedades *rdfs::class* para a criação de tabelas, o AORR cria o seu esquema baseado em vários critérios, tais como: quantidade de registro mínima na tabela, quantidade máxima de tabelas e frequência mínima da propriedade do sujeito em todo CS. Sendo assim, o AORR tende a criar menos tabelas, e com mais registros. Todavia, a abordagem do AORR necessita de tabelas de overflow para comportar os dados que não se adequaram a tais critérios. Outra diferenciação se dá pelo fato do AORR criar a base relacional, enquanto o no R2D existe apenas o mapeamento relacional. Com isso, no R2D as consultas que seriam executadas na base relacional são transformadas de SQL para SPARQL, e assim executadas efetivamente na base RDF. Já no AORR o processo é inverso, uma vez que é de interesse que as consultas sejam realizadas sobre a base relacional gerada, fazendo uso das otimizações de um SGBDR.

Como o processo de extração de esquema do AORR foi baseado na proposta ERSR [Pham et al., 2015], apenas alguns pontos nesse processo os diferenciam. Enquanto no AORR há o interesse em utilizar a base relacional como *backend* de armazenamento, o ERSR tem por objetivo que a base seja utilizada diretamente por usuários. Além disso, o ERSR não dá suporte à atualização incremental, sendo necessário criar a base relacional novamente para comportar novos dados da base RDF. Outro ponto a destacar é que o AORR dá suporte à tradução de consultas SPARQL para SQL, uma vez que ele possui um conjunto de tabelas de metadados que são consultadas para a transformação. No ERSR, por outro lado, as consultas são formuladas diretamente em SQL, havendo a necessidade de conhecimento prévio do esquema da base relacional gerada. As diferenças entre o ERSR e o AORR são explicadas com mais detalhes no Capítulo 4.

Capítulo 4

Armazenamento Otimizado de dados RDF em SGBDR

Esse trabalho tem por objetivo armazenar de forma eficiente dados RDF em um SGBDR. A eficiência do sistema proposto, o AORR (Armazenamento Otimizado de dados RDF em SBGDR), deve-se ao armazenamento de grande parte dos dados RDF em tabelas de entidades. Estas tabelas são geradas a partir de um esquema extraído com base no processo de extração de esquema ERSR, detalhado na Seção 3.3.

Uma das diferenças do esquema gerado pelo AORR em relação ao ERSR é o propósito da base relacional gerada. No AORR a base gerada tem o propósito de ser utilizada como *backend* de armazenamento de aplicações. Assim, não são realizados alguns passos referentes à ontologia, uma vez que as classes ontológicas são utilizadas para criar rótulos e propriedades que possuam nomes de melhor assimilação humana. Além disso, no AORR, para permitir que a base seja consultada utilizando a linguagem SPARQL, é necessário que as informações sobre o mapeamento da base RDF para o esquema relacional sejam mantidas em tabelas de metadados.

Outro ponto que distingue o AORR do ERSR é o fato do AORR ser um sistema dinâmico, ou seja, ele dá suporte à atualização incremental de dados. Sendo assim, dada uma inserção de tripla na base RDF, essa tripla é inserida no AORR com base nos metadados. Os metadados são de crucial importância, uma vez que contêm todas as informações de mapeamento. A partir dos metadados, consultas SPARQL são mapeadas para consultas SQL, como pode ser visto, por exemplo, no trabalho de [Pauluk e Hara, 2016].

O sistema AORR não considera atualizações de esquema, mas apenas atualizações de instâncias de forma incremental. Em outras palavras, a proposta do AORR é estender o processo ERSR com uma atualização incremental. O ERSR foi proposto para gerar uma base RDF para que o usuário manipule a base relacional diretamente. O AORR, por outro lado, utiliza um SGBD relacional apenas como *backend* de armazenamento. Assim, a manipulação da base continua sendo através da linguagem SPARQL e atualizações envolvem triplas RDF. Dessa forma, é necessário haver um mapeamento das operações de atualização da base RDF para atualizações sobre a base relacional de forma incremental.

Na Figura 4.1 está ilustrado um esquema com as diferenças entre o AORR e o ERSR quanto à consulta e inserção de dados na base relacional gerada. A Figura 4.1(a) apresenta o processo ERSR. Nele, a inserção de triplas RDF acontece diretamente na base RDF, sendo necessário realizar uma nova geração da base relacional. Quanto às consultas, percebe-se a necessidade de conhecimento da estrutura da base relacional gerada, uma vez que elas são expressas em SQL e submetidas diretamente na base relacional. Já na Figura

4.1(b), que apresenta o processo AORR, nota-se que a inserção das triplas RDF passa por um módulo de 'Atualização Incremental', que é responsável por consultar as tabelas de metadados criadas durante o processo de extração de esquema, e realizar a inserção da tripla na base relacional, dispensando a necessidade de recriar a base. Nota-se que são consultas SPARQL passam por um módulo de tradução SPARQL-SQL. Esse módulo acessa as tabelas de metadados e então mapeia a consulta SPARQL para uma consulta SQL de acordo com o esquema da base relacional do AORR, não sendo necessário então nenhuma informação adicional em relação àquelas fornecidas pelas tabelas de metadados.

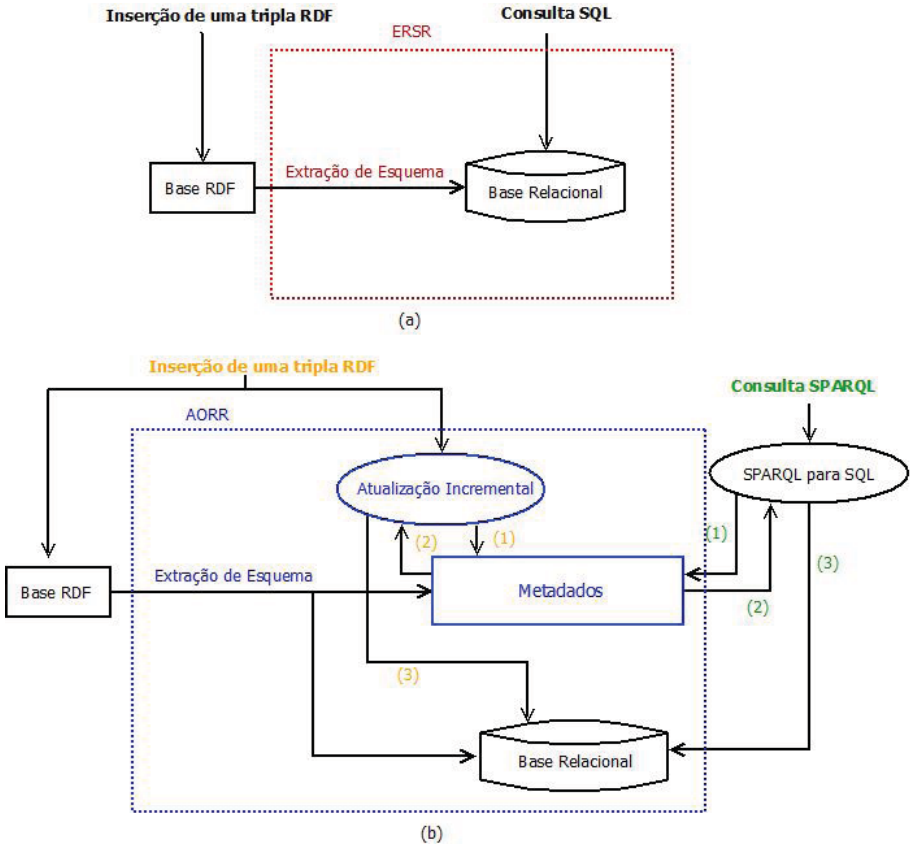


Figura 4.1: Inserção e consulta no ERSR(a) e AORR(b)

O AORR possui três etapas: extração de esquema, carregamento de dados e atualização da base.

4.1 Diferenças entre AORR e ERSR

O processo de extração de esquema do AORR foi baseado no processo ERSR, detalhado na Seção 3.3. No ERSR, dada uma base de dados RDF B, identificam-se os CS. Os dados que não se enquadram nessa estrutura são colocados em uma tabela no formato de tripla chamada Overflow.

No AORR, a tabela Overflow mencionada no ERSR foi renomeada para Overflow Geral (OverflowG). Tal distinção se fez necessária devido à adição de outras tabelas chamadas de Overflow Específico (OverflowE). Existe um OverflowE, que é também uma tabela SPO, para cada tabela de entidade resultante do processo de extração de esquema. A razão dessa distinção será explicada na Seção 4.4.

Na sequência é apresentado o processo de extração de esquema, ressaltando as diferenças entre o método proposto pelo AORR em relação ao realizado pelo ERSR.

4.1.1 Identificação dos CSs Básicos

Como descrito na Seção 3.3.1, um CS Básico (CSB) é caracterizado por um conjunto de predicados. Nesta etapa não há diferença entre o processo de identificação de CSBs do AORR em relação ao do ERSR.

4.1.2 Atribuição de Rótulos aos CSBs

Como exposto na Seção 3.3.2, ERSR propõe quatro métodos para a atribuição de rótulos aos CSs baseados nas seguintes características: propriedade *type*, propriedade discriminativa, relacionamento entre CSs e encurtamento de URI. Destas quatro, o AORR não considera o método baseado em propriedade discriminativa. A atribuição pela propriedade *type* foi mantida dado que a base RDF já define a classe ontológica a que o sujeito pertence. O resultado deste processo é um conjunto de CSs Tabela (CST), que correspondem a CSBs com rótulo.

4.1.3 Junção

Similarmente ao ERSR, na fase de junção, o AORR agrupa CSTs a fim de diminuir a quantidade de tabelas a serem geradas. Das regras propostas pelo ERSR, o AORR aplica a regra semântica que agrupa rótulos iguais atribuídos pela propriedade *type* e as duas regras estruturais descritas na Seção 3.3.3. A regra semântica de junção por ancestral ontológico não foi mantida no AORR, uma vez que o mesmo não considera hierarquias de classes ontológicas.

4.1.4 Filtragem de Esquema

A etapa de filtragem de esquema envolve um refinamento do esquema, no qual tabelas menos significativas são movidas para o Overflow. Uma diferença do AORR nesta etapa, com relação ao ERSR, é que além da tabela de Overflow Geral(OverflowG), o AORR também considera tabelas de Overflow Específico (OverflowE). É criado um OverflowE para cada tabela de entidade. Atributos que seriam migrados da tabela de entidade para o OverflowG pelo ERSR na filtragem de esquema, no AORR são migrados para o seu OverflowE. Dessa forma, as tabelas OverflowE contêm apenas registros cujo sujeito se encontra inserido na tabela de entidade referente àquele OverflowE.

O objetivo desta alteração é melhorar o desempenho de consultas que envolvam mais de um predicado de um mesmo sujeito, quando parte deles encontra-se em alguma tabela de entidade, e a outra parte em tabelas de overflow. Isso se dá pelo fato da tabela OverflowE conter apenas triplas com sujeitos inseridos na sua respectiva tabela de entidade. A tabela OverflowG no ERSR contém todos os dados que não se enquadraram às tabelas de entidade. Portanto, o volume de dados do OverflowG é maior, o que pode prejudicar o desempenho das consultas, em particular aquelas que envolvem auto-junções do OverflowG.

4.1.5 Filtragem de Instância

Há uma distinção no processo de filtragem de relacionamentos do AORR em relação ao do ERSR, descrito na Seção 3.3.5. No ERSR, quando constatado que uma propriedade será mantida no esquema, por possuir a quantidade mínima de sujeitos com tal propriedade, todos os registros que não a contêm são migrados para a tabela OverflowG. No AORR, para esses casos, tais propriedades permanecem na tabela. Isso se dá pelo fato de no ERSR não serem permitidos registros com atributos de relacionamento com valor nulo, enquanto o AORR permite.

Além disso, ainda no processo de filtragem de relacionamentos, quando uma propriedade é considerada irregular no ERSR, ela é colocada na tabela OverflowG. No AORR essa propriedade será colocada no OverflowE referente à CST a qual a propriedade pertencia.

Nas próximas seções são detalhados os algoritmos desenvolvidos para as três etapas que compõem o AORR: extração de esquema, que incluem as modificações realizadas no método ERSR, inserção dos dados e atualização da base.

4.2 Algoritmo de Extração e Geração da Base Relacional

Uma base RDF B é definida da seguinte forma:

Definição 1 *Uma base RDF B é definida como um conjunto de triplas (s,p,o), sendo pred(s) =* $\{\ p \mid (s,p,o) \in B\}$.

A Tabela 4.1 apresenta um exemplo de base RDF B, ou seja, conjunto de triplas (s,p,o).

O algoritmo de extração e geração da base relacional do AORR, Algoritmo 1, tem como entrada uma base RDF B, e um conjunto de parâmetros de configuração, e gera como saída uma base relacional R. Tais parâmetros afetam diretamente a geração da base relacional. Os parâmetros são: min_t, que define a quantidade mínima de registros que uma tabela deve conter; Ub_{tbl}, que corresponde a quantidade máxima de tabelas e T_{inf}, que define a frequência mínima que uma propriedade deve ter para ser mantida na tabela de entidade.

Algoritmo 1: Geração da base relacional

Entrada: base RDF B, parâmetros de configuração T_{inf}, min_t, Ub_{tbl}
Saída: uma base relacional R

```
1 início
2 |  (S, M) := criaBaseCategorizada( B );
3 |  label := atribuiRotulo(S, M, T_inf);
4 |  juntaCS( S, M, label, T_inf );
5 |  map := geraMapeamento( S, M, label, B );
6 |  filtragem( S, M, map, T_inf, min_t, Ub_tbl);
7 |  R := criaTabelas( map );
8 |  carregaDados( R, map, B, S, M );
9 fim
```

Tabela 4.1: Exemplo de triplas RDF

sujeito	predicado	objeto
s1	p1	'joao'
s1	p2	Pessoa
s1	p3	'0000-0001'
s1	p3	'0000-0002'
s1	p4	o1 (também é o s3)
s2	p1	'fabiana'
s2	p2	Pessoa
s2	p3	'0000-0003'
s2	p3	'0000-0004'
s2	p4	o2 (também é o s4)
s3	p5	'moto1'
s3	p6	Automotor
s4	p5	'moto2'
s4	p6	Automotor
s5	p5	'carro1'
s5	p6	Automotor
s5	p7	'32784738'
s6	p8	'Contato de Debora'

A sequência de chamadas de funções do Algoritmo 1 segue o processo detalhado na Seção 4.1. Inicialmente são identificados os CSs Básicos (CSBs), gerando uma base categorizada (Linha 2), como definido abaixo:

Definição 2 *Uma base RDF categorizada B_c de uma base RDF B é definida como um par (S, M), onde:*

- *S é um conjunto de CSBs,*
- *M é um mapeamento de S para um conjunto de sujeitos, tal que M(c) =* $\{ s \mid pred(s) = c,\ c \in S\}$.

Com base no exemplo da Tabela 4.1, tem-se CSB_1 = {p1,p2,p3,p4}, CSB_2 = {p5,p6}, CSB_3 = {p5,p6,p7} e CSB_4 = {p8}. Logo, o conjunto S = {CSB_1, CSB_2, CSB_3, CSB_4}, e M(CSB_1) = {s1,s2}, M(CSB_2) = {s3, s4}, M(CSB_3) = {s5} e M(CSB_4) = {s6}.

O passo seguinte consiste da atribuição de rótulos (Linha 3). O resultado do processo é uma função *label*, que mapeia cada CS c em S para um rótulo, que será utilizado como o nome da tabela de entidade que comportará os sujeitos em $M(c)$.

Seguindo o exemplo em questão, o rótulo atribuído ao CSB_1 é 'Pessoa' pela propriedade *type*, ao CSB_2 e CSB_3 é 'Automotor' também por *type*, e CSB_4 é 'observacao' por encurtamento de IRI. Sendo assim, após a atribuição de rótulos, tem-se estruturas formadas por pares *(label(CSB)*, CSB). No exemplo, essas estruturas são chamadas de CST_1, CST_2, CST_3 e CST_4, definidas respectivamente pelos pares (Pessoa, CSB_1), (Automotor, CSB_2), (Automotor, CSB_3) e (observacao, CSB_4).

O processo de junção de CSs (Linha 4) pode ser definido da seguinte forma:

Definição 3 *Dada uma base categorizada $B_c = (S, M)$ e dois CSBs, $c_1, c_2 \in S$, a junção de CSTs é definida como:*

- *merge_s(c_1, c_2): $S := S - c_1 - c_2 + (c_1 \cup c_2)$*

  merge_m(c_1, c_2): $M := M - M(c_1) - M(c_2) + M(c_1 \cup c_2), onde M(c_1 \cup c_2) = M(c_1) \cup M(c_2)$.

  merge_label(c_1, c_2): $label(c_1 \cup c_2) := label(c_1)$

A função '*merge_label(c_1, c_s)*' tem como resultado apenas um dos rótulos, escolhendo sempre o que vier primeiro, ou seja, c_1. Sendo assim, sempre que houver uma junção de CSs, se algum dos rótulos tiver sido atribuído pela propriedade *type*, esse deverá ser o rótulo resultante do '*merge_label(c_1, c_2)*'.

Considerando o exemplo adotado, a junção por rótulos entre o CST_2 e CST_3 resultará no CST_5, dado que CST_5 = (Automotor, CSB_5). Sendo assim, após a junção, S = $\{CSB_1, CSB_4, CSB_5\}$, sendo $CSB_5 = CSB_2 \cup CSB_3$, e M(CSB_5) = $M(CSB_2) \cup (CSB_3)$ = {s3, s4, s5}.

Os próximos passos do Algoritmo 1 são baseados na estrutura *map*, criada com a chamada da função *geraMapeamento* (Linha 5). A estrutura, bem como os detalhes da filtragem de esquema e instância (Linha 6), criação do esquema relacional (Linha 7) e inserção de dados na base relacional (Linha 8), são apresentados nas próximas seções.

4.2.1 Geração de mapeamento

O resultado final do processo de junção é armazenado em uma estrutura chamada *map*, criada com a chamada da função *geraMapeamento* (Linha 5). É a partir desta estrutura que os passos de filtragem e geração da base (Linhas 6-8) são executados.

A estrutura *map* mantém informações sobre o mapeamento dos CSs para o esquema relacional e pode ser definida da seguinte forma:

Definição 4 *Considere um conjunto de tipos literais L e uma base RDF categorizada $B_c = (S, M)$, com função de atribuição de rótulos label. map é um vetor multidimensional, tal que map[c][p][k][t], onde:*

- *c é um CS em S;*
- *p é um predicado que pertence a c;*
- *k contém a constante 'lit' ou a constante 'FK';*
- *se k = 'lit' então t é um tipo em L; caso k = 'FK', t contém label(c') para algum c' em S.*

O valor associado a map[c][p][k][t] é uma tupla (nomeTabela, nomeColuna, flg_multivalorado, flg_emOverflowE), onde:

- *nomeTabela contém o nome da relação na qual o predicado p de c é armazenado;*
- *nomeColuna é o atributo em nomeTabela que contém p;*
- *flg_multivalorado contém true se o predicado p é multivalorado e false, caso contrário;*

- *flg_emOverflowE contém true se o predicado p é armazenado na tabela de overflow específico de nomeTabela e false, caso contrário.*

Dando sequência ao exemplo adotado, se c = CSB_5, p = p1, k = 'lit', t = 'String', então *map['Pessoa'][p1]['lit']['String'] = ('PessoaRDF', 'nome', true, false)*, uma vez que *'nomeTabela' = 'PessoaRDF', 'nomeColuna' = 'nome', 'flg_multivalorado' = false, 'flg_emOverflowE' = false.*

O Algoritmo 2 apresenta o processo de criação da estrutura *map*.

Algoritmo 2: Geração do mapeamento - RDF para Relacional

```
   Entrada: S, M
   Saída: map
 1 início
 2   para cada c ∈ S faça
 3     para cada p ∈ c faça
 4       para cada (p,t) ∈ list_literal(c) faça
 5         se multivaloradoLiteral(c,p,t) então
 6           map[c][p]['lit'][t] := (label(c), attribute(p), true, false);
 7         senão
 8           map[c][p]['lit'][t] := (label(c), attribute(p), false, false);
 9       fim
10       para cada (p,c') ∈ relations(c) faça
11         se multivaloradoFK(c,p,c') então
12           map[c][p]['FK'][label(c')] :=
               (label(c), attribute(p), true, false);
13         senão
14           map[c][p]['FK'][label(c')] :=
               (label(c), attribute(p), false, false);
15       fim
16     fim
17   fim
18 fim
```

Para cada CS c são criadas 2 listas, *list_literal(c)* e *relations(c)*, que contêm os predicados com valores literais e que representam relacionamentos com outros CSs, respectivamente, bem como o tipo do objeto. Ou seja, *list_literal(c)* = $\{\ (p,t) \mid (s,p,o) \in B,\ s \in M(c),\ o$ *é um literal do tipo* $t\}$ e *relations(c)* = $\{\ (p,c') \mid (s,p,o) \in B,\ s \in M(c),\ o \in M(c')\}$.

No exemplo adotado, *list_literal(*CST_1*) = [(p1,String),(p2,String),(p3,String)]*, *list_literal(*CST_4*) = [(p8,String)]*, *list_literal(*CST_5*) = [(p5,String),(p6,String),(p7,Int)]*. Já a *relations(*CST_1*) = [(p4,*CST_5*)]*, *relations(*CST_4*) = []* e *relations(*CST_5*) = []*.

O Algoritmo 2 começa iterando sobre os CSs c em S, e sobre cada predicado p de c (Linhas 2 e 3). Dentro da iteração há dois 'laços' independentes a serem executados. Um deles é referente aos objetos literais (Linha 4), e o outro aos objetos de relacionamento (Linha 10).

Para os objetos literais, para cada par (p,t) em *list_literal(c)*, verifica-se se o predicado p, do tipo t é multivalorado. A função encarregada de tal verificação é a

multivaloradoLiteral(c,p,t), que determina se existem sujeitos s em $M(c)$ com mais de um objeto ligados por p com o mesmo tipo. Ou seja, são computados os conjuntos #(s,p,t) = $\{o|s, p, o \in B, o$ é do tipo $t\}$ e verificado se existem sujeitos com $|\#(s, p, t)|$ maior que 1. Em caso afirmativo, *multivaloradoLiteral(c,p,t)* retorna verdadeiro. Um processo similar é executado para os objetos de relacionamento nas Linhas 10 a 15.

Na criação da estrutura *map* o valor de *flg_emOverflowE* será sempre 'false', dado que a verificação se uma propriedade será armazenada em uma tabela de overflow é feita nos processos de filtragem de esquema e filtragem de instância.

Por exemplo, considere c = CST_1, p = p1, sendo $p1 \in c$. Como a função *list_literal(c)* retorna (p,t), sendo t o tipo do predicado literal, há apenas um elemento cujo p = p1, sendo tal elemento (p1, String). Na verificação da Linha 5 tem-se então que c = CST_1, p = p1, t = String. Observando a Tabela 4.1 nota-se que p1 não é uma propriedade multivalorada resultando na execução da Linha 8, deixando a estrutura *map* da seguinte forma: *map['Pessoa'][p1]['lit']['String'] = ('PessoaRDF', 'nome', false, false)*.

4.2.2 Filtragem de Esquema - AORR

Após a criação da estrutura *map*, o passo seguinte é a filtragem de esquema. Ele possui três etapas: filtragem por quantidade mínima de sujeitos e filtragem por quantidade máxima de tabelas, ambas definidas na Seção 7; e a filtragem de propriedades infrequentes, como apresentado na Seção 3.3.4. O Algoritmo 3 apresenta o processo geral de filtragem de esquema e os Algoritmos 4, 5 e 6 detalham as etapas que os compõem.

Algoritmo 3: Filtragem Esquema

```
   Entrada: S, M, map
1  início
2  |  filtragemEsquema_min_subj(S, M, map, min_t);
3  |  filtragemEsquema_max_tabela(S, M, map, Ub_tbl);
4  |  filtragemEsquema_pred_infrequentes(map, T_inf);
5  fim
```

O Algoritmo 4, que faz a filtragem por quantidade mínima de sujeitos, recebe como parâmetro de entrada o valor de min_t, que determina se um CS deve ser movido para a tabela OverflowG.

Algoritmo 4: Filtragem Esquema - Quantidade mínima de sujeitos

```
   Entrada: S, M, map, min_t
   Saída: map
1  início
2  |  para cada c ∈ map faça
3  |  |  se c não é uma tabela dimensional então
4  |  |  |  se |M(c)| < min_t então
5  |  |  |  |  migraCS_para_Overflow(c, map);
6  |  fim
7  fim
```

Sendo assim, se um dado CS c não possuir uma quantidade de sujeitos superior a min_t (Linha 3), e 'c' não configurar uma tabela dimensional, cuja equação consta na Seção , então a estrutura *map* é atualizada, mapeando tudo de 'c' para o OverflowG (Linha5). A função *migraCS_para_Overflow(c, map)* é responsável por organizar o *map* de forma que *c* não esteja mais mapeado como uma tabela de entidade, mas sim no OverflowG.

Baseado no exemplo adotado, considerando o valor de $min_t = 2$, então CST_4 tem o *map* alterado para que seja totalmente inserido no OverflowG, dado que $|M(CSB_4)| = 1$.

O Algoritmo 5 realiza o processo de filtragem por quantidade máxima de tabelas, e recebe como parâmetro de entrada o valor Ub_{tbl} que determina se um CS deve ser movido para o OverflowG.

Algoritmo 5: Filtragem Esquema - Quantidade máxima de tabelas

```
   Entrada: S, M, map, Ub_tbl
   Saída: map
1  início
2  |  S_o := ordenaCSs_crescente_qtd_suj(S);
3  |  enquanto |S_o| > Ub_tbl faça
4  |  |  seja c ∈ S_o o CS com menor qtd de sujeitos tal que c não é uma
   |  |   tabela dimensional
5  |  |  |  migraCS_para_Overflow(c, map);
6  |  |  |  S_o := S_o − c;
7  |  fim
8  fim
```

A função *ordernaCSs_crescente_qtd_suj*(S) (Linha 2) é responsável por ordenar o conjunto S de forma crescente, de acordo com $|M(c)|$. Itera-se então sobre os elementos (c) desse conjunto ordenado de S (S_o) enquanto a quantidade de CS for maior do que o limite, (Linha 3). Semelhante ao Algoritmo 4, se o CS *c* não for uma tabela dimensional, ele é mapeado na estrutura *map* para ser inserido no OverflowG.

No exemplo, considere que $Ub_{tbl} = 1$, $S_o = (CSB_1, CSB_4)$ de acordo com a função de ordenação crescente de CSs por quantidade de sujeitos associados. Uma vez que $|M(CSB_1)| = 2$, $|M(CSB_4)| = 3$ e CSB_1 não é uma tabela dimensional, ela não é migrada completamente para o OverflowG.

Por fim, o Algoritmo 6 realiza o processo de filtragem de propriedades infrequentes, recebendo como parâmetro de entrada o valor de T_{inf}, que determina se um predicado *p* do CS *c* deve ser transferido para o OverflowE.

A função *amount_subj_per_pred(c,p,t)* é responsável por contar a quantidade de sujeitos que possuem um determinado predicado, *amount_subj_per_pred(c,p,t)* = $|\{\ s \mid s \in M(c),\ (s,p,o) \in B,\ o$ *é do tipo* $t\}|$. A função *getNomeTabelaOverflowE(c)* retorna o nome da tabela OverflowE de c, $c \in S$.

Seguindo o exemplo em questão, considere c = CST_5, p = p7, k = 'lit', t = 'Int' e $T_{inf} = 40\%$. Nesse caso, como */M(*CST_5*)/* = 3, e *amount_subj_per_pred(*CST_5*, p7, 'Int')* = 1, o que corresponde a 33% aproximadamente. Tem-se que p7 é considerada uma propriedade infrequente de CST_5, e não deve ser inserida nas tabelas de entidade, mas no

Algoritmo 6: Filtragem Esquema - Propriedade Infrequente

Entrada: map, T_{inf}
Saída: map

```
1  início
2      para cada c ∈ map faça
3          para cada p ∈ map[c] faça
4              para cada k ∈ map[c][p] faça
5                  para cada t ∈ map[c][p][k] faça
6                      se amount_subj_per_pred(c, p, t) < T_inf então
7                          map[c][p][k][t].flg_emOveflowE := true;
8                          map[c][p][k][t].nomeTabela :=
                            getNomeTabelaOverflowE(c);
9                  fim
10             fim
11         fim
12     fim
13 fim
```

OverflowE de CST_5. Logo, a estrutura da *map* para essa propriedade desse CST mudaria de *map['Automotor'][p7]['lit']['Int'] = ('AutomotorRDF', 'cod_modelo', false, false)* para *map['Automotor'][p7]['lit']['Int'] = (", 'cod_modelo', false, true).*

4.2.3 Geração do Esquema Relacional

Por fim, baseado na estrutura de dados *map*, o esquema relacional é gerado, bem como as tabelas de metadados. O Algoritmo 7 descreve como a geração é feita:

Após a criação das tabelas de metadados, o algoritmo itera sobre todo c de *map*, dado que $c \in S$. Como todo CST contém um OverflowE associado, na Linha 4 ocorre a criação do OverflowE de c. Sendo assim, para cada $p \in map[c]$, $k \in map[c][p]$, $t \in map[c][p][k]$, verifica-se a tupla associada a cada *map[c][p][k][t]*. Se *flg_multivalorado = true*, então uma tabela para comportar a propriedade multivalorada p é criada, como pode ser visto na Linha 10. Caso contrário, a estrutura *predsTabela* é incrementada com o *nomeColuna* da tupla em questão, dado que *predsTabela* é definida como *predsTabela[c].listaNomeAtributos* = [], sendo $c \in S$.

Na Linha 16, após a iteração sobre todos os $p \in map[c]$, é criada a tabela referente ao CST iterado, com o nome das colunas contidas na estrutura *predsTabela[c]*.

Para o exemplo adotado, considere c = CST_1. Como cada $p \in CST_1$ contém apenas um tipo, então há uma tupla por propriedade:

map['Pessoa'][p1]['lit']['String'] = ('PessoaRDF', 'nome', false, false),
map['Pessoa'][p2]['lit']['String'] = ('PessoaRDF', 'type', false, false),
map['Pessoa'][p3]['lit']['String'] = ('telefoneMultivalueRDF', 'telefone', true, false),
map['Pessoa'][p4]['fk']['Automotor'] = ('PessoaRDF', 'fk_veiculo', false, false).

Sendo assim, na Linha 16 para c = CST_1, *predsTabela[CST_1].listaNomeAtributos = ['nome','type','fk_veiculo']*, criando assim a tabela *'PessoaRDF'* com tais colunas, além da coluna *OID* contida em toda tabela que não seja de metadados ou de overflow. Além

Algoritmo 7: Geração de Esquema

```
    Entrada: map
 1  início
 2      criaTabelasMetadados();
 3      para cada c ∈ map faça
 4          criaTabelaOverflowE(c);
 5          para cada p ∈ map[c] faça
 6              para cada k ∈ map[c][p] faça
 7                  para cada t ∈ map[c][p][k] faça
 8                      se map[c][p][k][t].flg_emOveflowE == false então
 9                          se map[c][p][k][t].flg_multivalorado == true então
10                              criaTabelaMultivalorada(
11                                  map[c][p][k][t].nomeTabela,
12                                  map[c][p][k][t].nomeColuna);
13                          senão
14                              predsTabela[c].listaNomeAtributos.add(
15                                  map[c][p][k][t].nomeColuna);
16                  fim
17              fim
18          fim
19          criaTabelaPrincipal(label(c), predsTabela[c].listaNomeAtributos);
20      fim
21  fim
```

disso, pelo fato da propriedade *'telefone'* ser multivalorada, na Linha 10 ocorrerá a criação da tabela *'telefoneMultivalueRDF'* com as colunas *(OID, telefone)*.

Observe que o nome da tabela é alterado quando o atributo é multivalorado, sendo definido pela concatenação do nome do atributo com *'MultivalueRDF'*. De forma similar, atributos que são *fk*, tem o seu nome acrescido do prefixo 'fk_'.

Tabelas de Metadados

Para que o AORR seja utilizado como *backend* de armazenamento, tabelas com informações do esquema da base foram criadas. Tais tabelas de metadados são necessárias tanto para a transformação de consultas SPARQL em consultas SQL, quanto para a atualização da base.

No AORR não é considerada a evolução de esquema, apenas movimentações de dados da parte de overflow para as tabelas de entidades, bem como inserção de novas triplas. Para tais ações, as tabelas de metadados são necessárias.

No AORR há três tabelas de metadados, sendo elas a *TB_Subj_OID, TB_DatabaseSchema e TB_FullPredicate*.

TB_Subj_OID

Contém a relação da IRI do sujeito (*'subj'*) para com o OID (*'OID'*) que o representa, além da tabela em que o sujeito se encontra inserido (*'tableName'*). Essa tabela

também é utilizada para se identificar se a IRI de um dado sujeito que está sendo inserido, ou de um objeto de um relacionamento, já existe na base.

Tabela 4.2: Exemplo de TB_Subj_OID

subj	OID	tableName
<http://dbtune.org/bbc/peel/artist/000449859d55f41aad74fb36f9fd7f46>	1	MusicArtistRDF
<http://dbtune.org/bbc/peel/perf_ins/000449859d55f41aad74fb36f9fd7f46>	2	PerformanceRDF
<http://dbtune.org/bbc/peel/artist/0004ca7431d195cd64459fc8e784daec>	3	MusicArtistRDF
.	.	.
.	.	.
.	.	.
<http://dbtune.org/bbc/peel/signal/119/e1b54f76aa6d53d03fd585de690bce5f>	69116	Overflow

TB_DatabaseSchema

Essa tabela armazena informações de todo o esquema da base, bem como o relacionamento entre as tabelas. Nessa tabela existe a coluna *'cs_identifier'*, que identifica um módulo na base. Esse módulo é formado pela tabela gerada a partir do CST, pela tabela *OverflowE* dessa CST e pelas tabelas multivaloradas referentes à essa CST. Como exemplo, considere um *'cs_identifier'* de valor 'CS1'. O 'CS1' identifica as tabelas *'PerformanceRDF'*, *'Overflow_PerformanceRDF'* e *'instrumentMultivalueRDF'*, como pode ser visto na Tabela 4.3

Vale salientar que, quando um relacionamento é descrito, sempre será entre 'cs_identifiers'. Sendo assim, cada módulo possui uma tabela central (tabela de entidade). Essa tabela central é a criada a partir do CST.

Além do *'cs_identifier'*, a *TB_DatabaseSchema* comporta o nome da propriedade na base (*'propertyName'*); o tipo dessa propriedade, ou seja, 'literal' se for um literal e o 'cs_identifier' da tabela referenciada, no caso de uma Chave Estrangeira (*'valueType'*); o nome da tabela em que a propriedade está inserida (*'tableName'*); e o atributo da tabela que comportará tal propriedade (*'tableAttribute'*).

Tabela 4.3: Exemplo de TB_DatabaseSchema

cs_identifier	propertyName	valueType	tableName	tableAttribute
CS1	OID	literal	PerformanceRDF	OID
CS1	place	literal	Overflow_PerformanceRDF	pred
CS1	fk_performance_of	CS5	PerformanceRDF	fk_performance_of
CS1	instrument	literal	instrumentMultivalueRDF	instrument
CS1	fk_performer	CS3	PerformanceRDF	fk_performer
CS5	OID	literal	TrackRDF	OID
CS5	title	literal	TrackRDF	title
CS2	OID	literal	SoundRDF	OID
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
CS3	OID	literal	MusicArtistRDF	OID
CSover	fk_sameAs	CS2	Overflow	pred
CSover	fk_sub_event	CS1	Overflow	pred

TB_FullPredicate

Esta tabela armazena o identificador do CS em que o predicado se encontra inserido (*'cs_identifier'*); tabela em que se encontra o atributo gerado (*'tableName'*); a tabela *OverflowE*, caso também esteja inserido nela (*'overflowTable'*); sua IRI completa do mesmo (*'predicate'*); e o nome do predicado na base, ou seja, o nome da coluna (*'columnName'*).

Tabela 4.4: Exemplo de TB_FullPredicate

cs_identifier	tableName	overflowTable	columnName	predicate
CS1	SoundRDF	<NULL>	type	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
CS2	chart_positionMultivalueRDF	<NULL>	chart_position	<http://purl.org/ontology/mo/chart_position>
CS2	fk_sameAsMultivalueRDF	<NULL>	fk_sameAs	<http://www.w3.org/2002/07/owl#sameAs>
CS3	MusicArtistRDF	Overflow_MusicArtistRDF	biography	<http://purl.org/ontology/mo/biography>

4.3 Carregamento de Dados

O Algoritmo 8 descreve o processo de carregamento dos dados de uma base RDF B em uma base relacional, dado que a estrutura de dados *map* especifica em qual tabela uma dada tripla $(s, p, o) \in B$ deve ser inserida. Leia-se no algoritmo de inserção que a função CSB(s) retorna um dado c, $c \in S$, sendo que $s \in M(c)$. A função *getCategoriaETipo(c, p, o)* retorna a estrutura de dados *catETipo*, que por sua vez, contém o atributo *categoria* e *tipo*, sendo $catETipo.categoria \in [' literal', ' FK']$ e $catETipo.tipo \in L$, contendo assim a informação da *categoria* de p, ou seja, *literal* ou *FK*, e *tipo* de o da tripla em questão. Já a função OID(s) retorna o OID associado a s. A estrutura de dados *col_e_nome* é uma lista $chave \rightarrow valor$, ou seja, $nomeDoAtributo \rightarrow valorDoAtributo$. Por fim, a função *csIdentifier(c)* é responsável por atribuir um identificador único a *c*

Ainda sobre o Algoritmo 8, segue uma breve descrição de seu funcionamento. A ideia é que cada tripla só pode ser inserida em uma de três tabelas distintas: na tabela principal, ou seja, *label(c)*, dado que $CSB(s) = c$; na tabela da propriedade multivalorada da tabela principal; ou ainda na tabela OverflowE da tabela principal. Sendo assim, a cada iteração, verifica-se se a tripla deverá ser inserida no OverflowE. Caso negativo, verifica-se então se deverá ser inserida em uma tabela de propriedade multivalorada. Caso negativo novamente, uma última verificação é realizada a fim de identificar se a tripla possui mais de um mesmo predicado com o mesmo nome, mesmo a estrutura *map* apresentando tal predicado como monovalorado. Tal fato poderia acontecer se, apesar de haver alguns sujeitos com essa propriedade como multivalorada, essa frequência é baixa, sendo assim considerada monovalorada. Para essa situação, as triplas dessa propriedade, com exceção de uma, são inseridas na tabela OverflowE.

Basicamente, itera-se até que o sujeito não seja mais o mesmo, daí a importância do arquivo estar agrupado por sujeitos. No conjunto de iterações de mesmo sujeito 's', a estrutura col_e_nome, sendo do tipo chave-valor, armazena como chave o atributo referente ao predicado 'p', e como valor o objeto 'o'. Quando identificado que o sujeito não é mais o mesmo da iteração anterior, a estrutura col_e_nome é utilizada para a inserção do registro de 's' na tabela de entidade em que o sujeito se enquadra, ou seja, CSB(s), dado que col_e_nome contém todos os atributos e respectivos valores do sujeito 's'. Após a inserção, a estrutura é zerada para comportar agora os atributos e valores do novo sujeito.

Quanto às tabelas de metadados, quase todas as informações necessários para populá-las está na *map*. As únicas informações faltantes são: OID, na tabela TB_Subj_OID; e preencher a coluna *overflowTable* com 'true', na tabela *TB_FullPredicate*,

Algoritmo 8: Inserção dados RDF em Base Relacional

```
Entrada: map, B
1  início
2      s' := 1;
3      para cada (s, p, o) ∈ B faça
4          se s ≠ s' então
5              se NÃO primeira iteracao e NÃO vazio col_e_nome então
6                  c' := CSB(s');
7                  insereEmTabelaPrincipal(label(c'), col_e_nome);
8                  col_e_nome := [ ];
9          c := CSB(s);
10         catETipo := getCategoriaETipo(c, p, o);
11         se
            map[c][p][catETipo.categoria][catETipo.tipo].flg_emOveflowE ==
            true então
12             insereEmOverflowE(OID(s),
13               map[c][p][catETipo.categoria][catETipo.tipo].nomeColuna, o);
14         senão
15             se map[c][p][l][t].flg_multivalorado == true então
16                 insereEmMultivalorada(map[c][p][l][t].nomeTabela, OID(s),
17                   map[c][p][catETipo.categoria]catETipo.tipo].nomeColuna, o);
18             senão
19                 se existe col_e_nome[map[c][p]
20                  [catETipo.categoria][catETipo.tipo].nomeColuna então
21                     insereEmOverflowE(OID(s), map[c][p][catETipo.categoria]
22                       [catETipo.tipo].nomeColuna, o);
23                     marcaOverflowTable(map[c][p][catETipo.categoria]
24                       [catETipo.tipo].nomeColuna, csIdentifier(c));
25                 senão
26                     se vazio col_e_nome então
27                         col_e_nome['OID'] := OID(s);
28                     col_e_nome[map[c][p][catETipo.categoria]
29                       [catETipo.tipo].nomeColuna] := o;
30         s' := s;
31     fim
32 fim
```

para os casos em que uma propriedade foi considerada monovalorada devido à infrequência dos casos apresentados como multivalorada. Nesse último caso, permanece apenas uma tripla dessa propriedade para cada sujeito, sendo as outras migradas para o *OverflowE*. A tabela *TB_Subj_OID* é preenchida no momento da geração do OID pela função *OID(s)*, enquanto a *TB_FullPredicate* é preenchida na execução da função 'marcaOverflowTable'.

Considerando ainda o exemplo empregado, na primeira iteração do algoritmo, temos que (s,p,o) = (s1, p1, 'joao'), $c = CSB_1$ (Linha 9), *catETipo.categoria* = 'lit' e *catETipo.tipo* = 'String' (Linha 10). Sendo assim, o *map['Pessoa'][p1]['lit']['String'].flg_emOverflowE* = false e *map['Pessoa'][p1]['lit']['String'].flg_multivalorado* = false. Verifica-se se o *nomeColuna* de *p1* já foi adicionado à estrutura *col_e_nome* (Linha 17). Constatado que não, verifica-se que *col_e_nome* está vazio adicionando assim o *nomeColuna* = 'OID' (Linha 22), bem como é adicionado a relação p1 => 'joao' a *col_e_nome* (Linha 23).

A estrutura *col_e_nome* segue sendo incrementada enquanto s = s1. Quando s = s2, ocorrerá uma inserção na tabela principal, no caso a *'PessoaRDF'*, com os valores referente a cada coluna contidos em *col_e_nome*. Logo, com base na Tabela 4.1, a estrutura *col_e_nome ficaria*:

col_e_nome['nome'] = ('joao'),
col_e_nome['type'] = ('Pessoa'),
col_e_nome['fk_veiculo'] = (OID(s3)).

A estrutura *col_e_nome* só não será incrementada nos casos em que a tripla é inserida em um *OverflowE*, ou em uma tabela multivalorada. Ainda no exemplo dado, nota-se que as triplas (s1,p3,'0000-0001') e (s1,p3,'0000-0002') foram inseridas diretamente na tabela *'telefoneMultivalueRDF'*, dado que a propriedade *p3* é uma propriedade multivalorada da tabela de entidade *'PessoaRDF'*.

Excepcionalmente, há um caso em que a estrutura *map* será atualizada durante o carregamento. Tal caso se dá quando apresenta-se mais de uma mesma propriedade para um dado sujeito, mesmo essa propriedade tendo sido categorizada como monovalorada (Linha 17). Para esse caso, apenas uma dessas propriedades e objetos são mapeados para a estrutura *col_e_nome*, sendo o restante delas mapeados para o *OverflowE*. Em seguida a estrutura *map* é atualizada com *flg_emOverflowE* = true para indicar que a propriedade em questão está tanto contida na tabela de entidade, quanto no seu *OverflowE*.

Ao final do processo de carregamento, a estrutura *map* é mapeada para que as tabelas de metadados sejam populadas.

4.4 Atualização da Base

É de interesse desse trabalho que o AORR se comporte como uma base dinâmica. Ou seja, uma vez criada a base relacional, ela não será criada novamente, mas atualizada. Tal suporte à atualização incremental da base gerada pelo AORR se faz possível através das tabelas de metadados.

4.4.1 Inserção de uma nova tripla

A inserção de novas triplas usa a seguinte estratégia geral: sempre que possível a inserção é realizada em uma tabela de entidade ou em seu overflow específico. Segue o algoritmo de atualização da base:

Algoritmo 9: Atualização incremental

```
1  início
2  |  se 'sujeito já se encontra inserido na base' então
3  |  |  se ('propriedade já consta para esse sujeito' E 'propriedade não
   |  |   multivalorada') OU 'propriedade não existe na tabela do sujeito'
   |  |   então
4  |  |  |  Executa 'inserção na tabela OverflowE da tabela'
5  |  |  senão
6  |  |  |  Executa 'inserção na tabela de entidade em que o sujeito se
   |  |  |   encontra'
7  |  senão
8  |  |  se 'propriedade Type' então
9  |  |  |  Executa 'inserção na tabela de entidade do Type'
10 |  |  senão
11 |  |  |  Executa 'inserção na tabela OverflowG'
12 fim
```

Segue a descrição do Algoritmo 9.

Após a inserção da tripla, a tabela TB_Subj_OID é atualizada. Caso seja constatado que o predicado da tripla inserida não existe no CS que o comporta, então as tabelas TB_DatabaseSchema e TB_FullPredicate são atualizadas com tal dado.

Considere a inserção de uma tripla (s, p, o). É possível determinar a tabela de entidade a qual s pertence em 2 casos. No primeiro, s já faz parte da base (Linha 6), o que pode ser determinado com uma busca na tabela de metadados *TB_Subj_OID*, e o valor do atributo *tableName* não é *Overflow*. Caso o sujeito não seja encontrado, um novo identificador OID é gerado e o sujeito é inserido na tabela *TB_Subj_OID*. Novos sujeitos só serão inseridos em uma tabela de entidade pelo segundo caso, que refere-se à inserção de triplas com o predicado *type*. É importante observar que as tabelas de entidades podem armazenar dados de um conjunto de tipos associados similares, mas cada *type* está associado a uma única tabela. Assim, uma tripla $(s, type, o)$ é armazenada em uma tabela de entidade T se T possuir uma coluna *type* (Linha 9) com pelo menos uma linha com valor igual a o. Caso esta tabela não exista, a tripla é armazenada no overflow geral (Linha 11).

A inserção de uma tripla (s, p, o) em uma tabela de entidade T, associada a um CS c, segue os seguintes passos. Primeiro, é verificado se c possui o predicado p. Para isso, a tabela de metadados *TB_FullPredicate* é pesquisada. Caso não exista, a tripla é inserida no overflow específico de T e as tabelas de metadados *TB_FullPredicate* e *TB_DatabaseSchema* são atualizadas para registrar a existência de um novo predicado no CS c. Se o predicado p já existir e for multivalorado, a tripla é inserida na tabela associada a este atributo, que está indicada pelo *tableName* da tabela *TB_FullPredicate* (Linha 6). Se p não for multivalorado, obtém-se (ou cria-se) a linha na tabela de entidade e atribui-se o valor o à coluna correspondente. Todavia, é possível que o atributo já possua um valor. Isso ocorre quando esperava-se que p fosse um atributo monovalorado. Neste caso, a tripla é inserida na tabela de overflow específico e as tabelas de metadados são

atualizadas para registrar a existência do predicado tanto na tabela de entidade como no seu overflow (Linha 4).

Por exemplo, considere a inserção da tripla $(s5, p5, '\mathit{carro\ popular}')$ na base ilustrada na Figura 1.2. Primeiro, é realizada uma busca na tabela de metadados *TB_Subj_OID* para verificar a existência do sujeito $s5$. Ele é encontrado com OID 5 e com o valor do atributo *tableName* igual a *AutomotorRDF*. Na sequência, é realizada uma busca na tabela *TB_FullPredicate* pelo predicado $p5$ com o CS associado à tabela AutomotorRDF. O predicado é encontrado, de onde obtém-se que o nome da coluna correspondente a $p5$ denomina-se *modelo*. A linha da tabela *AutomotorRDF* com OID 5 é então recuperada para preencher o valor do atributo *modelo*. No entanto, esta linha já possui este campo preenchido. Assim, a tripla *(5, modelo, carro popular)* é inserido na tabela *Overflow_AutomotorRDF*. Para registrar a existência de $p5$ tanto na tabela de entidade como no seu overflow, as tabelas *TB_FullPredicate* e *TB_DatabaseSchema* são atualizadas da seguinte forma. Na tabela *TB_FullPredicate*, a linha referente ao predicado $p5$ terá tanto o atributo *tableName* preenchido com *AutomotorRDF*, como o atributo *overflowTable* preenchido com *Overflow_AutomotorRDF*. Na tabela *TB_DatabaseSchema*, uma linha será inserida com os valores *(CS2, modelo, literal, Overflow_AutomotorRDF, modelo)*.

4.5 Processamento de Consultas

Esta seção exemplifica a utilização das tabelas de metadados na tradução de consultas SPARQL para consultas SQL. A tabela *TB_FullPredicate* é utilizada para encontrar o atributo referente ao predicado da consulta, enquanto a tabela *TB_DatabaseSchema* é utilizada para identificar relacionamentos entre as tabelas de entidades e seguir com a busca pelo padrão de triplas. Considere a consulta SPARQL da Figura 4.2(a) executada sobre a base ilustrada na Figura 1.2 após a inserção da tripla $(s5, p5, '\mathit{carro\ popular}')$. A consulta SQL resultante da tradução está ilustrada na Figura 4.2(b). Na consulta, a tabela *TB_DatabaseSchema* é utilizada para determinar quais CSs possuem os dois predicados, *modelo* e *cod_modelo* (referentes a $p5$ e $p7$), pois ambos estão associados ao mesmo sujeito $?v$ na consulta. Determina-se que apenas CS_2 satisfaz esta condição e que o o predicado *modelo* está presente tanto na tabela *AutomotorRDF*, como em seu overflow *Overflow_AutomotorRDF*, enquanto o predicado *cod_modelo* está presente apenas no *Overflow_AutomotorRDF*.

Sendo assim, duas subconsultas são geradas, uma para obter o atributo *modelo* da tabela *AutomotorRDF* e *cod_modelo* da tabela *Overflow_AutomotorRDF* e outra para obter os dois atributos de *Overflow_AutomotorRDF*. Elas são unidas com o operador UNION ALL para produzir o resultado final.

```
SELECT ?m ?c WHERE {
    ?v p5 ?m .
    ?v p7 ?c .
}
```

(a)

```
SELECT v.modeloXQ9G2 AS m,
    v.cod_modelo0VF65 AS c
FROM (SELECT a1.OID,
        a1.modelo AS modeloXQ9G2,
        o1.obj AS cod_modelo0VF65
    FROM AutomotorRDF a1,
        Overflow_AutomotorRDF o1
    WHERE a1.modelo IS NOT NULL
        AND o1.pred = 'cod_modelo'
        AND o1.obj IS NOT NULL
UNION ALL
SELECT o1.subj,
    o1.obj AS modeloXQ9G,
    o2.obj AS cod_modelo0VF65
FROM Overflow_AutomotorRDF o1,
    Overflow_AutomotorRDF o2
WHERE o1.subj = o2.subj
    AND o1.pred = 'modelo'
    AND o2.pred = 'cod_modelo'
    AND o1.obj IS NOT NULL
    AND o2.obj IS NOT NULL) v
```

(b)

Figura 4.2: Consulta SPARQL traduzida em consulta SQL

Esta tradução de consulta mostra que o volume de dados armazenados em tabelas de overflow podem ter um grande impacto no desempenho das consultas. Como no ERSR, o trabalho sobre o qual o processo de extração de esquema do AORR foi inspirado, não considera a existência de overflow específicos, mas apenas um único overflow geral, a consulta acima necessitaria fazer duas junções com tabelas potencialmente volumosas. O impacto da existência de tabelas de overflow específico sobre o desempenho das consultas é avaliado no experimento relatado na próxima seção.

Capítulo 5

Avaliação Experimental

Para avaliar o desempenho do esquema relacional gerado pelo AORR, foi desenvolvido um programa que faz a geração do esquema relacional e a inserção de dados RDF em um SGBD relacional.

O programa tem como entrada um arquivo com triplas RDF, a quantidade máxima de tabelas e a quantidade mínima de registros em uma tabela. A saída do programa é uma base relacional gerada no SGBDR MySQL, sendo o esquema criado a partir da execução de 'CREATE TABLE...', e os dados inseridos por meio de 'INSERT INTO...', como pode ser visto na Figura 5.1.

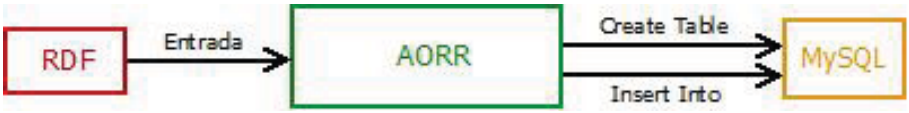


Figura 5.1: Entrada e Saída do AORR

O programa desenvolvido foi utilizado nos experimentos relatados neste capítulo, que têm por objetivo avaliar o desempenho de consultas na base relacional gerada pelo AORR, em relação à base gerada pelo ERSR. Sendo assim, o foco das consultas são as tabelas de OverflowE pelo fato dessas serem uma diferenciação estrutural entre os dois processos.

Todos os experimentos reportados neste capítulo foram executados em uma máquina com o SO Linux 3.13.0-32-generic #57 precise1-Ubuntu, processador Intel(R) Core(TM) i7-4710HQ CPU 2.50GHz e 2 GB de memória RAM. O MySQL Ver 14.14 Distrib 5.1.65 foi o SGBDR utilizado.

5.1 Base RDF e Relacional

A base RDF utilizada para os experimentos foi a Peel[1], que contém dados sobre apresentações musicais ao vivo. Os parâmetros de quantidade máxima de tabelas, quantidade mínima de registros e frequência mínima foi de 7, 1000 e 0.1 respectivamente. Tais valores foram colocados a fim de se simular um ambiente com uma tabela sendo completamente migrada para o OverflowG, dado que haveria 8 tabelas de entidade se não fosse pelo parâmetro passado.

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

A base relacional gerada pelo AORR possui 22 tabelas que podem ser visualizadas na Figura 5.1, bem como a quantidade de registros de cada uma, e o tamanho em MB.

As tabelas *MusicalWorkRDF*, *PerformanceRDF*, *PersonRDF*, *RecordingRDF*, *SignalRDF*, *SoundRDF* e *TransmissionRDF* são as tabelas de entidade; a *fk_engineerMultivalueRDF*, *fk_performerMultivalueRDF*, *fk_sub_eventMultivalueRDF* e *instrumentMultivalueRDF* são tabelas com propriedade multivalorada; a *Overflow_MusicalWorkRDF*, *Overflow_PerformanceRDF*, *Overflow_PersonRDF*, *Overflow_RecordingRDF*, *Overflow_SignalRDF*, *Overflow_SoundRDF* e *Overflow_TransmissionRDF* são as tabelas OverflowE; a tabela Overflow é a OverflowG e, por fim, a *TB_DatabaseSchema*, *TB_FullPredicate* e *TB_Subj_OID* tabelas de metadados.

O tempo médio de criação da base relacional foi aproximadamente de 4 minutos, para 276160 triplas.

Tabela 5.1: Base relacional gerada pelo AORR

Tabela	QtdRegistros	Tamanho MB
fk_engineerMultivalueRDF	3801	0.16
fk_performedMultivalueRDF	11869	0.49
fk_sub_eventMultivalueRDF	30062	1.55
instrumentMultivalueRDF	8926	0.43
MusicalWorkRDF	18273	2.93
PerformanceRDF	28631	4.13
PersonRDF	10611	1.76
RecordingRDF	3976	1.66
SignalRDF	5947	1.7
SoundRDF	3976	0.4
TransmissionRDF	3976	1.65
Overflow	7417	0.45
Overflow_MusicalWorkRDF	938	1.29
Overflow_PerformanceRDF	1330	0.14
Overflow_PersonRDF	7705	1.78
Overflow_RecordingRDF	501	0.08
Overflow_SignalRDF	782	0.1
Overflow_SoundRDF	0	0.03
Overflow_TransmissionRDF	1800	0.21
TB_DatabaseSchema	55	0.02
TB_FullPredicate	45	0.02
TB_Subj_OID	76894	7.7

As tabelas de entidade da base relacional foram indexadas na coluna *OID*, enquanto as tabelas de OverflowE e OverflowG na coluna *subj*. A estrutura de indexação utilizada é a árvore B.

5.2 Inserção de Dados

A fim de analisar o impacto do volume de dados armazenados no overflow, a partir da base relacional gerada pelo mapeamento, novas triplas foram criadas, inserindo novos registros na base. O incremento foi pensado baseado nos predicados utilizados

em cada consulta e que são armazenados no overflow. Por exemplo, se as consultas são realizadas sobre a tabela *SignalRDF* acessando as propriedades *'type'* e *'label'*, então o incremento de 5000 registros novos é realizado em cada uma destas propriedades. No caso do *Overflow_SignalRDF*, o incremento real será de 10000, dado que 5000 será para a propriedade *'type'* e outros 5000 para a propriedade *'label'*.

As inserções consideradas nos experimentos foram de 5000, 10000, 20000, 40000 e 80000 registros. Tais inserções ocorreram segundo ao processo de atualização incremental do AORR descrito na Seção 4.4.

As inserções foram direcionadas para os testes, ou seja, apenas adicionado o que é pertinente às consultas realizadas. Basicamente o processo de inserção foi de sujeitos não existentes. Sendo assim, para a inserção nas tabelas de entidade, primeiramente as triplas inseridas foram com a propriedade *'type'* da tabela em que se deseja inserir. Uma vez o sujeito inserido, é possível inserir as triplas com as propriedades pertinentes.

Dessa forma, as tabelas de entidade receberam um acréscimo de 5000, 10000, 20000, 40000 e 80000 registros. Já as tabelas de *OverflowE* e *Overflow* receberam o acréscimo proporcional a quantidade de propriedades pertinentes. No cenário de teste utilizado, os acréscimos respectivos a cada *OverflowE* e ao *OverflowG* é o apresentado na Tabela 5.2. O multiplicador é baseado na quantidade de propriedades utilizada nas consultas. Sendo assim, por exemplo, para o acréscimo de 40000 registros, a tabela *Overflow_SignalRDF* terá o incremento de 2*40000 = 80000 triplas, enquanto a tabela *Overflow_TransmissionRDF* terá de 1*40000 = 40000 triplas.

Tabela 5.2: Overflows e Multiplicador (qtd de propriedades relevantes)

Tabela	Multiplicador
Overflow	1
Overflow_MusicalWorkRDF	1
Overflow_PerformanceRDF	1
Overflow_PersonRDF	2
Overflow_RecordingRDF	1
Overflow_SignalRDF	2
Overflow_SoundRDF	1
Overflow_TransmissionRDF	1

Segue ainda na Tabela 5.3 a quantidade de triplas contidas no OverflowG, com e sem o OverflowE, e o tempo para se inserir tais quantidade de triplas.

Tabela 5.3: Volume OverflowG

Incrementos	ComOverE	SemOverE	Tempo/s
0	7417	20473	
5000	12417	70473	4
10000	17417	120473	9
20000	27417	220473	20
40000	47417	420473	35
80000	87417	820473	81

5.3 Consultas

Como a base Peel não disponibiliza um conjunto de consultas, foram geradas cinco consultas SPARQL sobre esta base, que foram definidas para explorar diferentes situações em que o OverflowE é utilizado. Segue a descrição e objetivo de cada uma delas.

Consulta 1: Buscar todos os `rótulos` de sinais musicais e códigos fonográficos (`isrc`). O objetivo desta consulta é explorar o desempenho de uma consulta realizada sobre uma tabela de entidade e seu OverflowE, dado que o `rótulo` e o `isrc` se encontram tanto na tabela `SignalRDF` bem como no tabela `Overflow_SignalRDF`.

Consulta 2: Buscar os títulos de trabalhos musicais e os rótulos dos seus sinais. O objetivo desta consulta é explorar o desempenho de uma consulta realizada sobre duas tabelas de entidade e seus respectivos OverflowE, dado que os atributos de título de trabalho musical e rótulo do sinal se encontram tanto nas tabelas de entidade como no OverflowE das tabelas `SignalRDF` e `MusicalWorkRDF`.

Consulta 3: Buscar as biografias das pessoas que fizeram fizeram apresentações musicais e o endereço da gravação. O objetivo desta consulta é analisar o impacto de se consultar apenas o OverflowE. O atributo de biografia é armazanenado em `Overflow_PersonRDF` e o endereço da gravadora em `Overflow_RecordingRDF`, mas não nas tabelas de entidade as quais estão associadas.

Consulta 4: Buscar todos os rótulos. O objetivo dessa consulta é analisar o impacto de consultas que, mesmo sendo armazenado no OverflowE, necessitam acessar o OverflowG. O atributo de rótulo está presente nas tabelas `PersonRDF`, `RecordingRDF`, `PerformanceRDF`, `MusicalWorkRDF`, `SignalRDF`, `TransmissionRDF` e `Overflow`.

Consulta 5: Busca o nome das pessoas que realizaram alguma gravação, o lugar e endereço em que foi gravado, o rótulo do sinal e rótulo do trabalho musical. O objetivo

desta consulta é semelhante ao objetivo da consulta 2, porém com maior diversidade na consulta, pois ela envolve as tabelas `PerformanceRDF`, `RecordingRDF`, `SignalRDF`, `PersonRDF` e `MusicalWorkRDF`.

O processo de tradução de SPARQL para SQL é baseado no programa desenvolvido por Pauluk [Pauluk e Hara, 2016]. Nele, as consultas SPARQL são traduzidas em uma consulta SQL sobre a base relacional gerada pelo AORR, que inclui tabelas de overflow específicas (OverflowE). A fim de comparar o AORR com o ERSR, que não considera a existência OverflowE, mas apenas uma única tabela de overflow geral (OverflowG), as mesmas consultas SPARQL são executadas sobre uma outra base de dados na qual todas as triplas dos OverflowE são migradas para o OverflowG. Para isso, as consultas SPARQL são traduzidas novamente para SQL, dado que a estrutura da base é diferente. Tal tradução é realizada de forma adequada pelo fato das tabelas de metadados terem sido alteradas de acordo, uma vez que não há mais as tabelas de OverflowE.

As consultas em SPARQL podem ser visualizadas no Apêndice A. Já as consultas traduzidas para SQL pelo processo do [Pauluk e Hara, 2016] estão no Apêndice B. Ainda as consultas sem o OverflowE, com todos os dados antes ali contidos, agora no OverflowG, estão contidas no Apêndice C.

5.4 Tempo de Execução das Consultas

Nesta seção são apresentados os tempos de execução das consultas detalhadas na Seção 5.3, considerando 6 bases de dados com volumes de dados nas tabelas de overflow. Estas bases são representadas no eixo horizontal dos gráficos. O valor 0 (zero) representa a base relacional gerada do mapeamento, ou seja, sem nenhum registro adicionado no overflow. As demais bases (5000, 10000, 20000, 40000, 80000) representam as bases resultantes dos incrementos no overflow, como detalhado na Seção 5.2.

A Figura 5.2 exibe os resultados da consulta 1. Percebe-se que quanto mais registros, maior é o ganho de tempo da consulta com o OverflowE em relação a sem o OverflowE. O tempo de consulta com OverflowE foi de 5198ms, enquanto sem foi de 7175ms, ou seja, um ganho de 27,55% para o teste de 80000 registros incrementados.

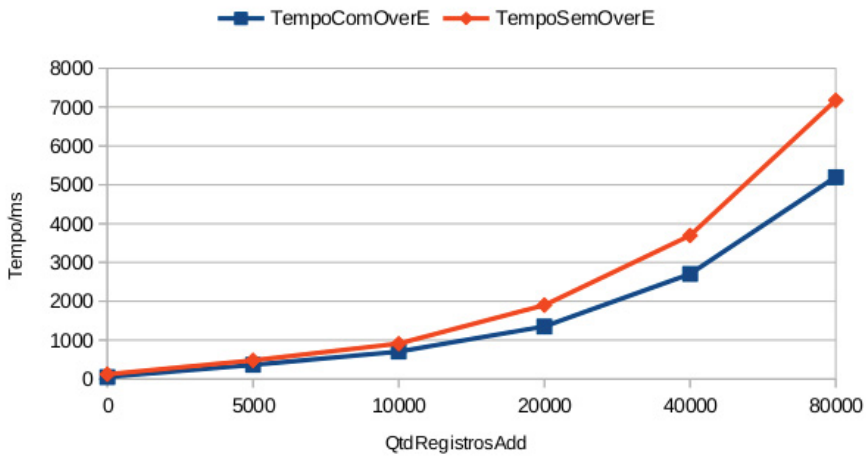


Figura 5.2: Consulta 1

Na consulta 2 houve uma grande diferença no tempo de execução em relação ao tempo da consulta 1, como pode ser visto na Figura 5.3. Pelo comando *explain* foi possível identificar que a causa dessa diferença se dá pelo uso de subconsultas, dado que depois de executadas, perde-se o ganho da indexação para a próxima subconsulta. Nota-se ainda na figura que não houve grande diferença entre as duas consultas, dado que com OverflowE o tempo foi de 1305180ms, e sem foi de 1276200ms para a base com incremento de 80000. Sendo assim, houve uma perda no tempo de consulta de 2,22%.

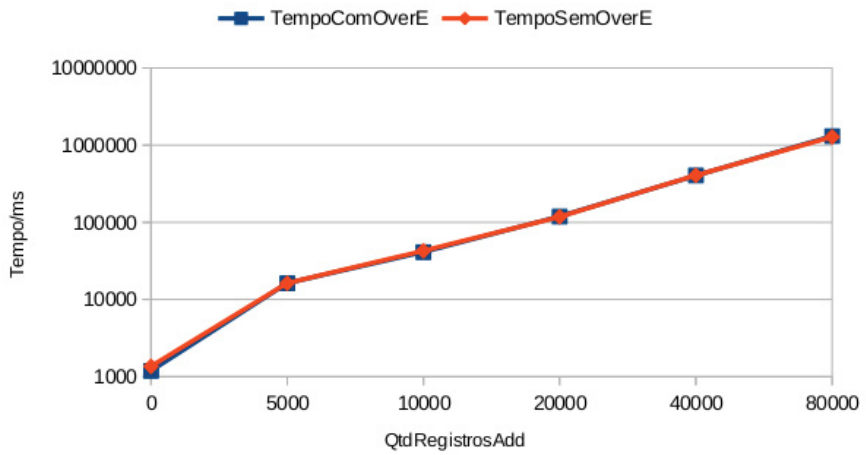


Figura 5.3: Consulta 2

Para um melhor comparativo, foi realizado uma adaptação das duas consultas SQL para que não houvesse mais subconsultas, como pode ser visto na Figura 5.4. O resultado é semelhante, porém houve um ganho de 1,34% da consulta com OverflowE, sobre sem o mesmo. As consultas SQL com OverflowE readaptadas com a remoção das subconsultas estão presentes no apêndice D. Já as readaptadas sem OverflowE constam no apêndice E.

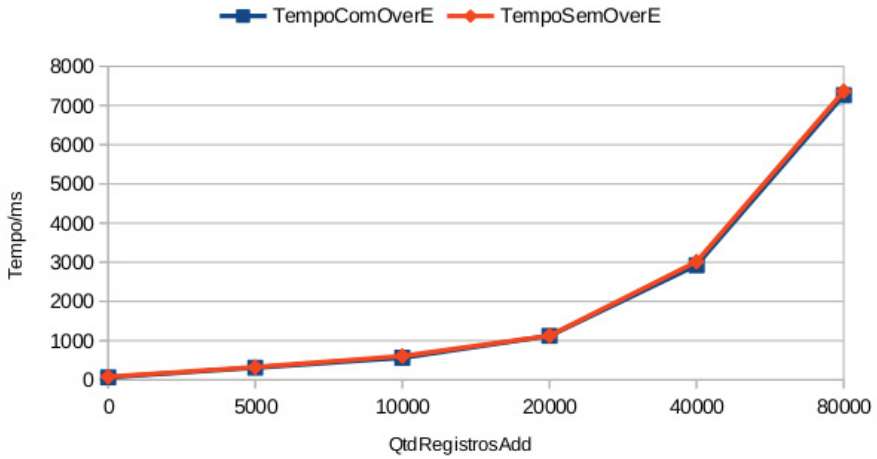


Figura 5.4: Consulta 2 - Adaptada

Já na consulta 3 é discrepante a perda de desempenho, como pode ser visto na Figura 5.5. Tal fato se dá pela consulta com OverflowE ser traduzida do SPARQL para

o SQL com subconsultas, enquanto a sem OverflowE não apresenta as mesmas. Sendo assim, nota-se a diferença de desempenho quando indexado, dado que com OverflowE o tempo foi de 544020ms, enquanto sem foi de 4751ms.

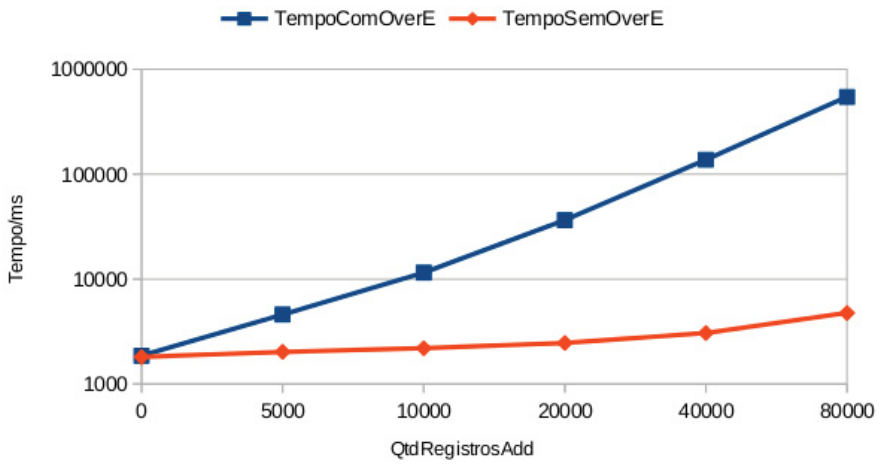


Figura 5.5: Consulta 3

A fim de se ter um melhor comparativo entre as consultas 3 com e sem o OverflowE, dado que a tradução da consulta impactou na diferença entre elas, foi adaptada a consulta com OverflowE para que não haja mais subconsultas. Nota-se na Figura 5.6, que para o incremento de 80000 registros, o ganho de 198,24% da consulta com OverflowE em relação a sem, dado que a com OverflowE foi executada em 1593ms, e sem foi em 4751ms.

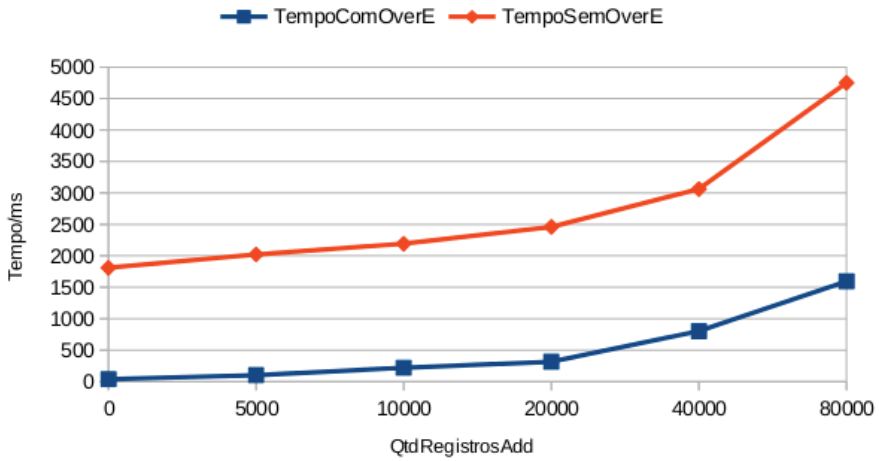


Figura 5.6: Consulta 3 - Adaptada

Na Figura 5.7 é possível observar uma sutil diferença de desempenho para a inserção de 80000 registros. Enquanto com OverflowE foi executado em 7238ms, sem o OverflowE foi em 7724ms, ou seja, um ganho de 6,71%.

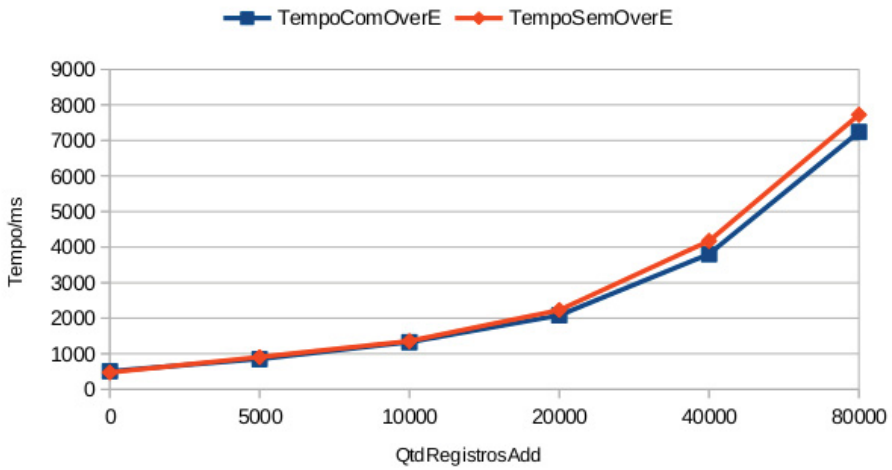


Figura 5.7: Consulta 4

Para a consulta 5 foi inviável a realização da mesma sem a adaptação da consulta com a remoção das subconsultas, dado que para o incremento de 5000, o tempo de execução foi superior a 2 horas. Sendo assim, a Figura 5.8 apresenta tempo de execução de 8650ms com OverflowE, e 9615ms sem o OverflowE, ou seja, um ganho de 11,15%.

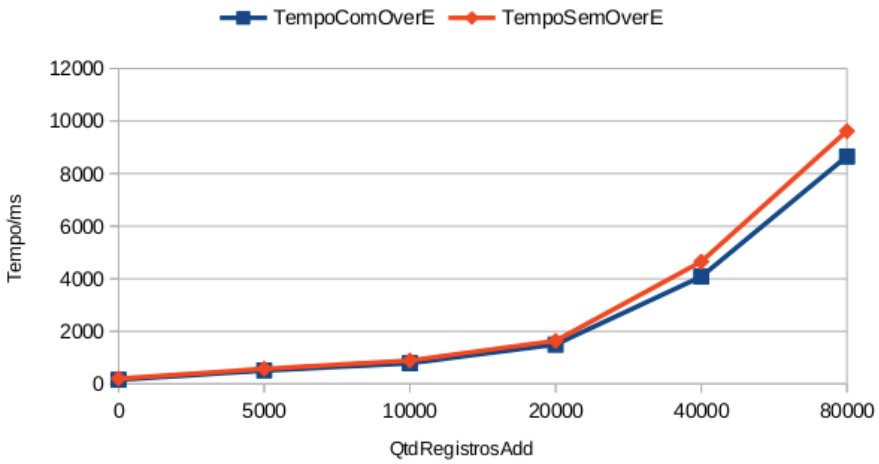


Figura 5.8: Consulta 5

5.5 Discussão

Os experimentos realizados mostram que no melhor caso houve um ganho de 198,24%, sendo essa a consulta 3. Tal ganho apresentado comprova que consultas realizadas apenas em OverflowE possuem um grande ganho de desempenho em relação às realizadas sobre apenas o OverflowG. No pior caso, desconsiderando a deficiência apresentada no processo de tradução, o tempo de execução foi semelhante, dado que a diferença foi de 1,34% na consulta 2.

Logo, é vantajoso do ponto de vista de desempenho de consulta o uso do OverflowE. Todavia, percebe-se o grande impacto que pode haver no desempenho a forma com que a consulta é traduzida.

Capítulo 6

Conclusão

O objetivo dessa dissertação é a utilização de técnicas para atualização de um SGBDR como *backend* de armazenamento de dados RDF e desenvolvimento de um sistema, chamado de AORR, que proporciona desempenho de consultas significativamente maior do que o da abordagem direta, e que dê suporte à atualização incremental da base. A dissertação descreve o processo realizado pelo AORR, baseado na extração de esquema do ERSR, para se alcançar o objetivo almejado.

A criação das tabelas de metadados foi apresentada na Seção 4.2.3, enfatizando a estrutura *map* como a base para a sua geração. Além disso, foram apresentadas as estruturas e a importância de cada tabela de metadados. Tal estrutura é gerada durante o processo de extração de esquema.

Após a geração do esquema relacional, o processo de inserção dos dados na base é iniciado. Esse processo é realizado baseado também na estrutura *map*, uma vez que ela contém informações sobre onde cada tripla deve ser inserida na base relacional, de acordo com o CS (*characteristic set*) a que pertence. Sendo assim, após o processo de inserção dos dados, considera-se que a geração da base relacional foi finalizada.

A fim de se ter maior desempenho nas consultas, o AORR possui uma tabela OverflowE para cada tabela de entidade criada. Sendo assim, menos registros são armazenados na tabela OverflowG, o que proporciona maior desempenho nas consultas que requerem auto-junções nesta tabela. Logo, as consultas realizadas sobre a base relacional gerada pelo AORR em relação às consultas realizadas sobre a abordagem direta apresentaram melhor desempenho com uma quantidade relevante de dados. Ou seja, quanto maior for a base, melhor tende a ser o ganho de desempenho do AORR.

Sendo assim, o Capítulo 5 apresenta o resultado das consultas realizadas sobre o AORR com OverflowE e sem OverflowE, sendo que esse último simula um cenário semelhante ao do ERSR. Observou-se um ganho de desempenho com o OverflowE nos melhores casos, e nos piores, a indiferença do método. Dado que tal desempenho do AORR foi superior ao do ERSR, e esse último apresentou ganho de até cinco vezes sobre a abordagem direta [Pham et al., 2015], tem-se que o AORR possui melhor desempenho do que a abordagem direta.

6.1 Publicações

As publicações resultantes da dissertação estão listados abaixo:

- Rafael L. Prado, Rebeca Schroeder, Carmem S. Hara, "Armazenamento Otimizado de dados RDF em um SGBD Relacional", Anais do Simpósio Brasileiro de Banco de

Dados 2018 (SBBD): Este artigo descreve o AORR, um sistema de geração de base relacional, a partir de uma base RDF, capaz de realizar atualizações incrementais na base relacional gerada, bem como dar suporte à tradução de consultas SPARQL em consultas SQL. Consultas realizadas sobre a base relacional gerada possui maior desempenho do que sobre uma base relacional gerada a partir da abordagem direta.

- Joao G. Pauluk, Mariana M. Garcez Duarte, Rafael L. Prado, Carmem S. Hara, "Processamento de Consultas SPARQL em uma Base Relacional de Entidades", Anais do Simpósio Brasileiro de Banco de Dados 2018 (SBBD): Este artigo apresenta uma técnica de tradução de consultas SPARQL em consultas SQL, com base em tabelas de metadados específicas contidas na base relacional de entidades.

6.2 Trabalhos Futuros

Dentre as possíveis extensões ao trabalho desenvolvido nesta dissertação, podem ser citados:

- Migração de dados: um processo de verificação periódico capaz de verificar se um conjunto de triplas na tabela OverflowG se enquadra em alguma tabela de entidade, realizando assim a migração dos dados das tabelas de overflow, para as tabelas de entidade. Tal migração requer também a atualização das tabelas de metadados.

- Evolução de esquema: um processo periódico de verificação capaz de analisar a estrutura da base, em relação aos dados contidos nas tabelas de overflow. Tais verificações resultariam em alterações no esquema da base, realizando a geração de novas tabelas, remoção de colunas, divisão de tabelas, dentre outras operações de reestruturação.

- Parametrização: criar um processo de geração automática dos parâmetros que são atualmente dados como entrada pelo usuário, tais como o número máximo de tabelas e a quantidade mínima de sujeitos, gerando assim um esquema relacional que proporcione melhor desempenho nas consultas.

- Experimentações: realizar novos experimentos, em grandes *benchmarks*, buscando explorar melhor a estrutura gerada.

Referências Bibliográficas

[Abadi et al., 2007] Abadi, D. J., Marcus, A., Madden, S. R. e Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. Em *VLDB*, páginas 411–422.

[Aluç et al., 2014] Aluç, G., Ozsu, M. T. e Daudjee, K. (2014). Workload matters: Why rdf databases need a new design. *Proceedings of the VLDB Endowment*, 7(10):837–840.

[Bornea et al., 2013] Bornea, M., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O. e Bhattacharjee, B. (2013). Building an efficient rdf store over a relational database. Em *ACM SIGMOD*, páginas 121–132.

[Cyganiak et al., 2014] Cyganiak, R., Wood, D. e Lanthaler, M. (2014). Rdf 1.1 concepts and abstract syntax. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[Dias, 2001] Dias, T. D. (2001). Web semântica: Conceitos básicos e tecnologias associadas. Trabalho de Conclusão de Curso (Graduação em Informática) - Universidade do Estado do Rio de Janeiro.

[Hayes e Patel-Schneider, 2014] Hayes, P. J. e Patel-Schneider, P. F. (2014). Rdf 1.1 semantics.

[Hogan et al., 2014] Hogan, A., Arenas, M., Mallea, A. e Polleres, A. (2014). Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27-28:42–69.

[Kowata, 2011] Kowata, E. T. (2011). Metadados de bancos de dados relacionais: Extração e exposição com o protocolo oai-pmh. *Dissertação de mestrado defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás.*

[Laufer, 2015] Laufer, C. (2015). Guia de web semântica. http://ceweb.br/guias/web-semantica//creditos/. Realização GOVERNO DO ESTADO DE SÃO PAULO.

[Pauluk e Hara, 2016] Pauluk, J. G. e Hara, C. S. (2016). Processamento de consultas sparql em um sgbd relacional. *Monografia apresentada à disciplina de Trabalho de Graduação em Banco de Dados, Universidade Federal do Paraná.*

[Penteado et al., 2015] Penteado, R. R. M., Schroeder, R. e Hara, C. S. (2015). Exploração de grafos RDF com distribuição controlada. Em *Anais do XXX SBBD - Short Papers*, páginas 69–74.

[Pham et al., 2015] Pham, M.-D., Passing, L., Erling, O. e Boncz, P. (2015). Deriving an emergent relational schema from rdf data. *Proceedings of the 24th International Conference on World Wide Web*, páginas 864–874.

[Prud'hommeaux et al., 2008] Prud'hommeaux, E., Seaborne, A., Laboratories, H.-P. e Bristol (2008). Sparql query language for rdf. `https://www.w3.org/TR/rdf-sparql-query/`.

[Pérez et al., 2009] Pérez, J., Arenas, M. e Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(10). Article 16.

[Ramunajam et al., 2009] Ramunajam, S., Gupta, A., Khan, L., Seida, S. e Thurasaisingham, B. (2009). R2d: Extracting relational structure from rdf stores. Em *Proc. of the IEEE/ACM WIC*, páginas 361–366.

[Scabora et al., 2017] Scabora, L. C., Oliveira, P. H., Kaster, D. S., Traina, A. J. M. e Traina-Jr, C. (2017). Relational graph data management on the edge: Grouping vertices' neighborhood with edge-k. Em *Anais do XXXII Simpósio Brasileiro de Banco de Dados*, páginas 124–135.

[Zeng et al., 2013] Zeng, K., Yang, J., Wang, H., Shao, B. e Wang, Z. (2013). A distributed graph engine for web scale rdf data. *Proc. of the VLDB Endowment*, 6(4):265–276.

Apêndice A

Consultas em SPARQL

Consulta 1

```
SELECT ?p ?j WHERE {
        ?a label ?p .
        ?a isrc ?j .
}
```

Consulta 2

```
SELECT ?p ?k WHERE {
        ?a label ?p .
        ?a fk_published_as_MusicalWork ?j .
        ?j title_xmlS_string ?k .
}
```

Consulta 3

```
SELECT ?p ?k WHERE {
        ?a biography ?p .
        ?a fk_produced ?j .
        ?j address ?k .
}
```

Consulta 4

```
SELECT ?b WHERE {
        ?a label ?b .
}
```

Consulta 5

```
SELECT ?l ?i ?m ?u WHERE {
        ?a fk_performer ?p .
        ?a fk_recorded_as ?j .
        ?a fk_performance_of ?k .
        ?p name ?l .
        ?p fk_produced ?b .
        ?b address ?i .
        ?j label ?u .
        ?k label ?m .
}
```

Apêndice B

Consulta Traduzidas para SQL

Consulta 1

```
SELECT
    a.isrcGXQ9G2 AS j,
    a.labelY0VF65 AS p
FROM
    (SELECT
        SignalRDF.OID,
        SignalRDF.label AS labelY0VF65,
        SignalRDF.isrc AS isrcGXQ9G2
    FROM
        SignalRDF
    WHERE
        SignalRDF.label IS NOT NULL
        AND SignalRDF.isrc IS NOT NULL

    UNION ALL

    SELECT
        o1.subj,
        o1.obj AS labelY0VF65,
        o2.obj AS isrcGXQ9G2
    FROM
        Overflow_SignalRDF o1,
        Overflow_SignalRDF o2
    WHERE
        o1.subj = o2.subj
        AND o1.pred = 'label'
        AND o2.pred = 'isrc'
        AND o1.obj IS NOT NULL
        AND o2.obj IS NOT NULL

    UNION ALL
```

```
    SELECT
       SignalRDF.OID,
       SignalRDF.label AS labelY0VF65,
       Overflow_SignalRDF.obj AS isrcGXQ9G2
    FROM
       SignalRDF,
       Overflow_SignalRDF
    WHERE
       SignalRDF.label IS NOT NULL
       AND Overflow_SignalRDF.subj = SignalRDF.OID
       AND Overflow_SignalRDF.pred = 'isrc'
       AND Overflow_SignalRDF.obj IS NOT NULL

    UNION ALL

    SELECT
       SignalRDF.OID,
       Overflow_SignalRDF.obj AS labelY0VF65,
       SignalRDF.isrc AS isrcGXQ9G2
    FROM
       SignalRDF,
       Overflow_SignalRDF
    WHERE
       SignalRDF.isrc IS NOT NULL
       AND Overflow_SignalRDF.subj = SignalRDF.OID
       AND Overflow_SignalRDF.pred = 'label'
       AND Overflow_SignalRDF.obj IS NOT NULL) a
```

Consulta 2

```
SELECT
    a.labelY0VF65 AS p,
    b.title_xmlS_stringL4MN74 AS k
FROM
    (SELECT
       SignalRDF.OID,
       SignalRDF.label AS labelY0VF65,
       SignalRDF.fk_published_as_MusicalWork AS fk_published_as_MusicalWorkJ1UF37
    FROM
       SignalRDF
    WHERE
       SignalRDF.label IS NOT NULL
       AND SignalRDF.fk_published_as_MusicalWork IS NOT NULL

    UNION ALL

    SELECT
```

```
        SignalRDF.OID,
        Overflow_SignalRDF.obj AS labelY0VF65,
        SignalRDF.fk_published_as_MusicalWork AS fk_published_as_MusicalWorkJ1UF37
    FROM
        SignalRDF, Overflow_SignalRDF
    WHERE
        SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND Overflow_SignalRDF.subj = SignalRDF.OID
        AND Overflow_SignalRDF.pred = 'label'
        AND Overflow_SignalRDF.obj IS NOT NULL) a,

    (SELECT
        MusicalWorkRDF.OID,
        MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
        MusicalWorkRDF
    WHERE
        MusicalWorkRDF.title_xmlS_string IS NOT NULL

    UNION ALL

    SELECT
        Overflow_MusicalWorkRDF.subj,
        Overflow_MusicalWorkRDF.obj AS title_xmlS_stringL4MN74
    FROM
        Overflow_MusicalWorkRDF
    WHERE
        Overflow_MusicalWorkRDF.pred = 'title_xmlS_string'
        AND Overflow_MusicalWorkRDF.obj IS NOT NULL) b
WHERE
    a.fk_published_as_MusicalWorkJ1UF37 = b.OID;
```

Consulta 3

```
SELECT
    j.addressWK0MNU AS k,
    a.biography1F77HU AS p
FROM
    (SELECT
        o1.subj as OID,
        o1.obj AS fk_producedLIDKQA,
        o2.obj AS biography1F77HU
    FROM
        Overflow_PersonRDF o1, Overflow_PersonRDF o2
    WHERE
        o1.subj = o2.subj
        AND o1.pred = 'fk_produced'
```

```
        AND o2.pred = 'biography'
        AND o1.obj IS NOT NULL
        AND o2.obj IS NOT NULL) a,

    (SELECT
        Overflow_RecordingRDF.subj as OID,
        Overflow_RecordingRDF.obj AS addressWK0MNU
    FROM
        Overflow_RecordingRDF
    WHERE
        Overflow_RecordingRDF.pred = 'address'
        AND Overflow_RecordingRDF.obj IS NOT NULL) j
WHERE
    j.OID = fk_producedLIDKQA;
```

Consulta 4

```
SELECT
    a.label7NB6ES AS b
FROM
    (SELECT
        TransmissionRDF.OID,
        TransmissionRDF.label AS label7NB6ES
    FROM
        TransmissionRDF
    WHERE
        TransmissionRDF.label IS NOT NULL

    UNION ALL

    SELECT
        Overflow_TransmissionRDF.subj,
        Overflow_TransmissionRDF.obj AS label7NB6ES
    FROM
        Overflow_TransmissionRDF
    WHERE
        Overflow_TransmissionRDF.pred = 'label'
        AND Overflow_TransmissionRDF.obj IS NOT NULL

    UNION ALL

    SELECT
        SignalRDF.OID,
        SignalRDF.label AS label7NB6ES
    FROM
        SignalRDF
    WHERE
```

```
   SignalRDF.label IS NOT NULL

UNION ALL

SELECT
   Overflow_SignalRDF.subj,
   Overflow_SignalRDF.obj AS label7NB6ES
FROM
   Overflow_SignalRDF
WHERE
   Overflow_SignalRDF.pred = 'label'
   AND Overflow_SignalRDF.obj IS NOT NULL

UNION ALL

SELECT
   RecordingRDF.OID,
   RecordingRDF.label AS label7NB6ES
FROM
   RecordingRDF
WHERE
   RecordingRDF.label IS NOT NULL

UNION ALL

SELECT
   PersonRDF.OID,
   PersonRDF.label AS label7NB6ES
FROM
   PersonRDF
WHERE
   PersonRDF.label IS NOT NULL

UNION ALL

SELECT
   PerformanceRDF.OID,
   PerformanceRDF.label AS label7NB6ES
FROM
   PerformanceRDF
WHERE
   PerformanceRDF.label IS NOT NULL

UNION ALL

SELECT
   Overflow.subj,
   Overflow.obj AS label7NB6ES
```

```
    FROM
      Overflow
    WHERE
      Overflow.pred = 'label'
      AND Overflow.obj IS NOT NULL

    UNION ALL

    SELECT
      MusicalWorkRDF.OID,
      MusicalWorkRDF.label AS label7NB6ES
    FROM
      MusicalWorkRDF
    WHERE
      MusicalWorkRDF.label IS NOT NULL) a
```

Consulta 5

```
SELECT
    j.label4ZO3BW AS u,
    k.labelK4VHFL AS m,
    b.addressOB8ZVD AS i,
    p.nameJECP52 AS l
FROM
    (SELECT
      PerformanceRDF.OID,
      PerformanceRDF.fk_performance_of AS fk_performance_ofJ6TLM9,
      Overflow_PerformanceRDF.obj AS fk_recorded_asYXPD8Z,
      PerformanceRDF.fk_performer AS fk_performerVV1YZG
    FROM
      PerformanceRDF,
      Overflow_PerformanceRDF
    WHERE
      PerformanceRDF.fk_performance_of IS NOT NULL
      AND Overflow_PerformanceRDF.subj = PerformanceRDF.OID
      AND Overflow_PerformanceRDF.pred = 'fk_recorded_as'
      AND Overflow_PerformanceRDF.obj IS NOT NULL
      AND PerformanceRDF.fk_performer IS NOT NULL) a,

    (SELECT
      PersonRDF.OID,
      Overflow_PersonRDF.obj AS fk_producedST4Y9Q,
      PersonRDF.name AS nameJECP52
    FROM
      PersonRDF,
      Overflow_PersonRDF
    WHERE
```

```
        Overflow_PersonRDF.obj IS NOT NULL
        AND Overflow_PersonRDF.pred = 'fk_produced'
        AND PersonRDF.OID = Overflow_PersonRDF.subj
        AND PersonRDF.name IS NOT NULL) p,

    (SELECT
        Overflow_RecordingRDF.subj as OID,
        Overflow_RecordingRDF.obj AS addressOB8ZVD
    FROM
        Overflow_RecordingRDF
    WHERE
        Overflow_RecordingRDF.pred IS NOT NULL) b,

    (SELECT
        SignalRDF.OID,
        SignalRDF.label AS label4ZO3BW
    FROM
        SignalRDF
    WHERE
        SignalRDF.label IS NOT NULL

    UNION ALL

    SELECT
        Overflow_SignalRDF.subj as OID,
        Overflow_SignalRDF.obj AS label4ZO3BW
    FROM
        Overflow_SignalRDF
    WHERE
        Overflow_SignalRDF.pred = 'label'
        AND Overflow_SignalRDF.obj IS NOT NULL) j,

    (SELECT
        MusicalWorkRDF.OID,
        MusicalWorkRDF.label AS labelK4VHFL
    FROM
        MusicalWorkRDF
    WHERE
        MusicalWorkRDF.label IS NOT NULL) k
WHERE
    k.OID = fk_performance_ofJ6TLM9
    AND j.OID = fk_recorded_asYXPD8Z
    AND p.OID = fk_performerVV1YZG
    AND b.OID = fk_producedST4Y9Q
```

Apêndice C

Consultas Traduzidas para SQL sem Overflow Específico

Consulta 1

```
SELECT
    a.isrcGXQ9G2 AS j,
    a.labelY0VF65 AS p
FROM
    (SELECT
        SignalRDF.OID,
        SignalRDF.label AS labelY0VF65,
        SignalRDF.isrc AS isrcGXQ9G2
    FROM
        SignalRDF
    WHERE
        SignalRDF.label IS NOT NULL
        AND SignalRDF.isrc IS NOT NULL

    UNION ALL

    SELECT
        o1.subj,
        o1.obj AS labelY0VF65,
        o2.obj AS isrcGXQ9G2
    FROM
        Overflow o1,
        Overflow o2
    WHERE
        o1.subj = o2.subj
        AND o1.pred = 'label'
        AND o2.pred = 'isrc'
        AND o1.obj IS NOT NULL
        AND o2.obj IS NOT NULL
```

```
    UNION ALL

    SELECT
       SignalRDF.OID,
       SignalRDF.label AS labelY0VF65,
       Overflow.obj AS isrcGXQ9G2
    FROM
       SignalRDF,
       Overflow
    WHERE
       SignalRDF.label IS NOT NULL
       AND Overflow.subj = SignalRDF.OID
       AND Overflow.pred = 'isrc'
       AND Overflow.obj IS NOT NULL

    UNION ALL

    SELECT
       SignalRDF.OID,
       Overflow.obj AS labelY0VF65,
       SignalRDF.isrc AS isrcGXQ9G2
    FROM
       SignalRDF,
       Overflow
    WHERE
       SignalRDF.isrc IS NOT NULL
       AND Overflow.subj = SignalRDF.OID
       AND Overflow.pred = 'label'
       AND Overflow.obj IS NOT NULL) a
```

Consulta 2

```
SELECT
    a.labelY0VF65 AS p,
    b.title_xmlS_stringL4MN74 AS k
FROM
    (SELECT
       SignalRDF.OID,
       SignalRDF.label AS labelY0VF65,
       SignalRDF.fk_published_as_MusicalWork AS fk_published_as_MusicalWorkJ1UF37
    FROM
       SignalRDF
    WHERE
       SignalRDF.label IS NOT NULL
       AND SignalRDF.fk_published_as_MusicalWork IS NOT NULL

    UNION ALL
```

```
    SELECT
       SignalRDF.OID,
       Overflow.obj AS labelY0VF65,
       SignalRDF.fk_published_as_MusicalWork AS fk_published_as_MusicalWorkJ1UF37
    FROM
       SignalRDF, Overflow
    WHERE
       SignalRDF.fk_published_as_MusicalWork IS NOT NULL
       AND Overflow.subj = SignalRDF.OID
       AND Overflow.pred = 'label'
       AND Overflow.obj IS NOT NULL) a,

    (SELECT
       MusicalWorkRDF.OID,
       MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
       MusicalWorkRDF
    WHERE
       MusicalWorkRDF.title_xmlS_string IS NOT NULL

    UNION ALL

    SELECT
       Overflow.subj,
       Overflow.obj AS title_xmlS_stringL4MN74
    FROM
       Overflow
    WHERE
       Overflow.pred = 'title_xmlS_string'
       AND Overflow.obj IS NOT NULL) b
WHERE
    a.fk_published_as_MusicalWorkJ1UF37 = b.OID;
```

Consulta 3

```
SELECT
    a.addressWK0MNU AS k,
    a.biography1F77HU AS p
FROM
    (SELECT
       o1.subj,
       o1.obj AS fk_producedLIDKQA,
       o2.obj AS biography1F77HU,
       o3.obj AS addressWK0MNU
    FROM
       Overflow o1, Overflow o2, Overflow o3
```

```
    WHERE
      o1.subj = o2.subj
      AND o1.obj = o3.subj
      AND o1.pred = 'fk_produced'
      AND o2.pred = 'biography'
      AND o3.pred = 'address'
      AND o1.obj IS NOT NULL
      AND o2.obj IS NOT NULL
      AND o3.obj IS NOT NULL) a
```

Consulta 4

```
SELECT
    a.label7NB6ES AS b
FROM
    (SELECT
       TransmissionRDF.OID,
       TransmissionRDF.label AS label7NB6ES
    FROM
       TransmissionRDF
    WHERE
       TransmissionRDF.label IS NOT NULL

    UNION ALL

    SELECT
       SignalRDF.OID,
       SignalRDF.label AS label7NB6ES
    FROM
       SignalRDF
    WHERE
       SignalRDF.label IS NOT NULL

    UNION ALL

    SELECT
       RecordingRDF.OID,
       RecordingRDF.label AS label7NB6ES
    FROM
       RecordingRDF
    WHERE
       RecordingRDF.label IS NOT NULL

    UNION ALL

    SELECT
       PersonRDF.OID,
```

```
    PersonRDF.label AS label7NB6ES
  FROM
    PersonRDF
  WHERE
    PersonRDF.label IS NOT NULL

  UNION ALL

  SELECT
    PerformanceRDF.OID,
    PerformanceRDF.label AS label7NB6ES
  FROM
    PerformanceRDF
  WHERE
    PerformanceRDF.label IS NOT NULL

  UNION ALL

  SELECT
    Overflow.subj,
    Overflow.obj AS label7NB6ES
  FROM
    Overflow
  WHERE
    Overflow.pred = 'label'
    AND Overflow.obj IS NOT NULL

  UNION ALL

  SELECT
    MusicalWorkRDF.OID,
    MusicalWorkRDF.label AS label7NB6ES
  FROM
    MusicalWorkRDF
  WHERE
    MusicalWorkRDF.label IS NOT NULL) a
```

Consulta 5

```
SELECT
  j.label4ZO3BW AS u,
  k.labelK4VHFL AS m,
  b.addressOB8ZVD AS i,
  p.nameJECP52 AS l
FROM
  (SELECT
    PerformanceRDF.OID,
```

```
   PerformanceRDF.fk_performance_of AS fk_performance_ofJ6TLM9,
   Overflow.obj AS fk_recorded_asYXPD8Z,
   PerformanceRDF.fk_performer AS fk_performerVV1YZG
FROM
   PerformanceRDF,
   Overflow
WHERE
   PerformanceRDF.fk_performance_of IS NOT NULL
   AND Overflow.subj = PerformanceRDF.OID
   AND Overflow.pred = 'fk_recorded_as'
   AND Overflow.obj IS NOT NULL
   AND PerformanceRDF.fk_performer IS NOT NULL) a,

(SELECT
   PersonRDF.OID,
   Overflow.obj AS fk_producedST4Y9Q,
   PersonRDF.name AS nameJECP52
FROM
   PersonRDF,
   Overflow
WHERE
   Overflow.obj IS NOT NULL
   AND Overflow.pred = 'fk_produced'
   AND PersonRDF.OID = Overflow.subj
   AND PersonRDF.name IS NOT NULL) p,

(SELECT
   Overflow.subj AS OID,
   Overflow.obj AS addressOB8ZVD
FROM
   Overflow
WHERE
   Overflow.pred IS NOT NULL) b,

(SELECT
   SignalRDF.OID,
   SignalRDF.label AS label4ZO3BW
FROM
   SignalRDF
WHERE
   SignalRDF.label IS NOT NULL

UNION ALL

SELECT
   Overflow.subj,
   Overflow.obj AS label4ZO3BW
FROM
```

```
        Overflow
    WHERE
        Overflow.pred = 'label'
        AND Overflow.obj IS NOT NULL) j,

    (SELECT
        MusicalWorkRDF.OID,
        MusicalWorkRDF.label AS labelK4VHFL
    FROM
        MusicalWorkRDF
    WHERE
        MusicalWorkRDF.label IS NOT NULL) k
WHERE
    k.OID = fk_performance_ofJ6TLM9
    AND j.OID = fk_recorded_asYXPD8Z
    AND p.OID = fk_performerVV1YZG
    AND b.OID = fk_producedST4Y9Q
```

Apêndice D

Consultas Traduzidas para SQL sem Subconsultas

Consulta 2

```
SELECT
    a.labelY0VF65 AS p,
    a.title_xmlS_stringL4MN74 AS k
FROM
    (SELECT
        SignalRDF.OID,
        SignalRDF.label AS labelY0VF65,
        MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
        SignalRDF, MusicalWorkRDF
    WHERE
        SignalRDF.label IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork = MusicalWorkRDF.OID
        AND MusicalWorkRDF.title_xmlS_string IS NOT NULL

    UNION ALL

    SELECT
        SignalRDF.OID,
        Overflow_SignalRDF.obj AS labelY0VF65,
        MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
        SignalRDF, Overflow_SignalRDF, MusicalWorkRDF
    WHERE
        SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork = MusicalWorkRDF.OID
        AND MusicalWorkRDF.title_xmlS_string IS NOT NULL
        AND Overflow_SignalRDF.subj = SignalRDF.OID
        AND Overflow_SignalRDF.pred = 'label'
```

```
    AND Overflow_SignalRDF.obj IS NOT NULL

  UNION ALL

  SELECT
    SignalRDF.OID,
    SignalRDF.label AS labelY0VF65,
    Overflow_MusicalWorkRDF.obj AS title_xmlS_stringL4MN74
  FROM
    SignalRDF, Overflow_MusicalWorkRDF
  WHERE
    SignalRDF.label IS NOT NULL
        AND Overflow_MusicalWorkRDF.subj = SignalRDF.fk_published_as_MusicalWork
    AND Overflow_MusicalWorkRDF.pred = 'title_xmlS_string'
    AND Overflow_MusicalWorkRDF.obj IS NOT NULL

  UNION ALL

  SELECT
    SignalRDF.OID,
    Overflow_SignalRDF.obj AS labelY0VF65,
    Overflow_MusicalWorkRDF.obj AS title_xmlS_stringL4MN74
  FROM
    SignalRDF, Overflow_SignalRDF, Overflow_MusicalWorkRDF
  WHERE
    SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND Overflow_MusicalWorkRDF.subj = SignalRDF.fk_published_as_MusicalWork
    AND Overflow_MusicalWorkRDF.pred = 'title_xmlS_string'
    AND Overflow_MusicalWorkRDF.obj IS NOT NULL
    AND Overflow_SignalRDF.subj = SignalRDF.OID
    AND Overflow_SignalRDF.pred = 'label'
    AND Overflow_SignalRDF.obj IS NOT NULL) a
```

Consulta 3

```
SELECT
  a.addressWK0MNU AS k,
  a.biography1F77HU AS p
FROM
  (SELECT
    o1.subj,
    o1.obj AS fk_producedLIDKQA,
    o2.obj AS biography1F77HU,
    o3.obj AS addressWK0MNU
  FROM
```

```
    Overflow_PersonRDF o1, Overflow_PersonRDF o2, Overflow_RecordingRDF o3
  WHERE
    o1.subj = o2.subj
    AND o1.obj = o3.subj
    AND o1.pred = 'fk_produced'
    AND o2.pred = 'biography'
    AND o3.pred = 'address'
    AND o1.obj IS NOT NULL
    AND o2.obj IS NOT NULL
    AND o3.obj IS NOT NULL) a
```

Consulta 5

```
      SELECT
  a.label4ZO3BW AS u,
  a.labelK4VHFL AS m,
  a.addressOB8ZVD AS i,
  a.nameJECP52 AS l
FROM
  (SELECT
    PerformanceRDF.OID,
    PersonRDF.name AS nameJECP52,
    Overflow_RecordingRDF.obj AS addressOB8ZVD,
    SignalRDF.label AS label4ZO3BW,
    MusicalWorkRDF.label AS labelK4VHFL
  FROM
    PerformanceRDF,
    Overflow_PerformanceRDF,
    PersonRDF,
    Overflow_PersonRDF,
    Overflow_RecordingRDF,
    SignalRDF,
    MusicalWorkRDF
  WHERE
    PerformanceRDF.fk_performance_of IS NOT NULL
    AND Overflow_PerformanceRDF.subj = PerformanceRDF.OID
    AND Overflow_PerformanceRDF.pred = 'fk_recorded_as'
    AND Overflow_PerformanceRDF.obj IS NOT NULL
    AND PerformanceRDF.fk_performer IS NOT NULL
    AND PersonRDF.OID = PerformanceRDF.fk_performer
    AND Overflow_PersonRDF.obj IS NOT NULL
    AND Overflow_PersonRDF.pred = 'fk_produced'
    AND PersonRDF.OID = Overflow_PersonRDF.subj
    AND PersonRDF.name IS NOT NULL
    AND Overflow_RecordingRDF.subj = Overflow_PersonRDF.obj
    AND Overflow_RecordingRDF.obj IS NOT NULL
    AND Overflow_RecordingRDF.pred = 'address'
```

```
    AND SignalRDF.OID = Overflow_PerformanceRDF.obj
    AND SignalRDF.label IS NOT NULL
    AND MusicalWorkRDF.OID = PerformanceRDF.fk_performance_of
    AND MusicalWorkRDF.label IS NOT NULL

UNION ALL

SELECT
    PerformanceRDF.OID,
    PersonRDF.name AS nameJECP52,
    Overflow_RecordingRDF.obj AS addressOB8ZVD,
    Overflow_SignalRDF.obj AS label4ZO3BW,
    MusicalWorkRDF.label AS labelK4VHFL
FROM
    PerformanceRDF,
    Overflow_PerformanceRDF,
    PersonRDF,
    Overflow_PersonRDF,
    Overflow_RecordingRDF,
    Overflow_SignalRDF,
    MusicalWorkRDF
WHERE
    PerformanceRDF.fk_performance_of IS NOT NULL
    AND Overflow_PerformanceRDF.subj = PerformanceRDF.OID
    AND Overflow_PerformanceRDF.pred = 'fk_recorded_as'
    AND Overflow_PerformanceRDF.obj IS NOT NULL
    AND PerformanceRDF.fk_performer IS NOT NULL
    AND PersonRDF.OID = PerformanceRDF.fk_performer
    AND Overflow_PersonRDF.obj IS NOT NULL
    AND Overflow_PersonRDF.pred = 'fk_produced'
    AND PersonRDF.OID = Overflow_PersonRDF.subj
    AND PersonRDF.name IS NOT NULL
    AND Overflow_RecordingRDF.subj = Overflow_PersonRDF.obj
    AND Overflow_RecordingRDF.obj IS NOT NULL
    AND Overflow_RecordingRDF.pred = 'address'
    AND Overflow_SignalRDF.subj = Overflow_PerformanceRDF.obj
    AND Overflow_SignalRDF.obj IS NOT NULL
    AND Overflow_SignalRDF.pred = 'label'
    AND MusicalWorkRDF.OID = PerformanceRDF.fk_performance_of
    AND MusicalWorkRDF.label IS NOT NULL) a
```

Apêndice E

Consultas Traduzidas para SQL sem Subconsultas e sem Overflow Específico

Consulta 2

```
SELECT
    a.labelY0VF65 AS p,
    a.title_xmlS_stringL4MN74 AS k
FROM
    (SELECT
        SignalRDF.OID,
        SignalRDF.label AS labelY0VF65,
        MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
        SignalRDF, MusicalWorkRDF
    WHERE
        SignalRDF.label IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork = MusicalWorkRDF.OID
        AND MusicalWorkRDF.title_xmlS_string IS NOT NULL

    UNION ALL

    SELECT
        SignalRDF.OID,
        Overflow.obj AS labelY0VF65,
        MusicalWorkRDF.title_xmlS_string AS title_xmlS_stringL4MN74
    FROM
        SignalRDF, Overflow, MusicalWorkRDF
    WHERE
        SignalRDF.fk_published_as_MusicalWork IS NOT NULL
        AND SignalRDF.fk_published_as_MusicalWork = MusicalWorkRDF.OID
        AND MusicalWorkRDF.title_xmlS_string IS NOT NULL
```

```
    AND Overflow.subj = SignalRDF.OID
    AND Overflow.pred = 'label'
    AND Overflow.obj IS NOT NULL

  UNION ALL

  SELECT
    SignalRDF.OID,
    SignalRDF.label AS labelY0VF65,
    Overflow.obj AS title_xmlS_stringL4MN74
  FROM
    SignalRDF, Overflow
  WHERE
    SignalRDF.label IS NOT NULL
    AND Overflow.subj = SignalRDF.fk_published_as_MusicalWork
    AND Overflow.pred = 'title_xmlS_string'
    AND Overflow.obj IS NOT NULL

  UNION ALL

  SELECT
    SignalRDF.OID,
    o1.obj AS labelY0VF65,
    o2.obj AS title_xmlS_stringL4MN74
  FROM
    SignalRDF, Overflow o1, Overflow o2
  WHERE
    SignalRDF.fk_published_as_MusicalWork IS NOT NULL
    AND o2.subj = SignalRDF.fk_published_as_MusicalWork
    AND o2.pred = 'title_xmlS_string'
    AND o2.obj IS NOT NULL
    AND o1.subj = SignalRDF.OID
    AND o1.pred = 'label'
    AND o1.obj IS NOT NULL) a
```

Consulta 5

```
      SELECT
  a.label4ZO3BW AS u,
  a.labelK4VHFL AS m,
  a.addressOB8ZVD AS i,
  a.nameJECP52 AS l
FROM
  (SELECT
    PerformanceRDF.OID,
    PersonRDF.name AS nameJECP52,
    o3.obj AS addressOB8ZVD,
```

```
    SignalRDF.label AS label4ZO3BW,
    MusicalWorkRDF.label AS labelK4VHFL
FROM
    PerformanceRDF,
    Overflow o1,
    PersonRDF,
    Overflow o2,
    Overflow o3,
    SignalRDF,
    MusicalWorkRDF
WHERE
    PerformanceRDF.fk_performance_of IS NOT NULL
    AND o1.subj = PerformanceRDF.OID
    AND o1.pred = 'fk_recorded_as'
    AND o1.obj IS NOT NULL
    AND PerformanceRDF.fk_performer IS NOT NULL
    AND PersonRDF.OID = PerformanceRDF.fk_performer
    AND o2.obj IS NOT NULL
    AND o2.pred = 'fk_produced'
    AND PersonRDF.OID = o2.subj
    AND PersonRDF.name IS NOT NULL
    AND o3.subj = o2.obj
    AND o3.obj IS NOT NULL
    AND o3.pred = 'address'
    AND SignalRDF.OID = o1.obj
    AND SignalRDF.label IS NOT NULL
    AND MusicalWorkRDF.OID = PerformanceRDF.fk_performance_of
    AND MusicalWorkRDF.label IS NOT NULL

UNION ALL

SELECT
    PerformanceRDF.OID,
    PersonRDF.name AS nameJECP52,
    o3.obj AS addressOB8ZVD,
    Overflow_SignalRDF.obj AS label4ZO3BW,
    MusicalWorkRDF.label AS labelK4VHFL
FROM
    PerformanceRDF,
    Overflow o1,
    PersonRDF,
    Overflow o2,
    Overflow o3,
    Overflow_SignalRDF,
    MusicalWorkRDF
WHERE
    PerformanceRDF.fk_performance_of IS NOT NULL
    AND o1.subj = PerformanceRDF.OID
```

```
AND o1.pred = 'fk_recorded_as'
AND o1.obj IS NOT NULL
AND PerformanceRDF.fk_performer IS NOT NULL
AND PersonRDF.OID = PerformanceRDF.fk_performer
AND o2.obj IS NOT NULL
AND o2.pred = 'fk_produced'
AND PersonRDF.OID = o2.subj
AND PersonRDF.name IS NOT NULL
AND o3.subj = o2.obj
AND o3.obj IS NOT NULL
AND o3.pred = 'address'
AND Overflow_SignalRDF.subj = o1.obj
AND Overflow_SignalRDF.obj IS NOT NULL
AND Overflow_SignalRDF.pred = 'label'
AND MusicalWorkRDF.OID = PerformanceRDF.fk_performance_of
AND MusicalWorkRDF.label IS NOT NULL) a
```