Armazenamento Otimizado de dados RDF em um SGBD Relacional

Rafael L. Prado[1], **Rebeca Schroeder**[2], **Carmem S. Hara**[1]

[1]Departamento de Informática – Universidade Federal do Paraná - UFPR
CEP: 81530-900 – Curitiba – PR – Brazil

[2]DCC – Universidade do Estado de Santa Catarina - UDESC
CEP: 89219-710 – Joinville – SC – Brazil

rlprado@inf.ufpr.br, rebeca.schroeder@udesc.br, carmem@inf.ufpr.br

Abstract. *Several methods employ Relational Database Management Systems (RDBMS) to store RDF data. However, the direct mapping from RDF to a table of triples does not produce good query performance. This paper introduces AORR, an optimized method to store RDF data in a RDBMS. AORR identifies data entities in order to define a relational schema. In addition, AORR differs from related work by supporting SPARQL to SQL query translation as well as dynamic data insertions. An experimental study shows that AORR improves the overall query performance, compared to a close related work.*

Resumo. *Diversas propostas utilizam Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDRs) para o armazenamento de dados RDF. O mapeamento direto de RDF para uma tabela de triplas resulta em um desempenho ineficiente no processamento de consultas. Este artigo propõe AORR (**A**rmazenamento **O**timizado de dados **R**DF em SGBD**R**), um método que identifica entidades de dados para gerar tabelas. Além disto, AORR se diferencia de trabalhos relacionados por possibilitar a tradução de consultas SPARQL-SQL, bem como atualizações incrementais da base. Um estudo experimental mostrou que AORR apresenta desempenho superior em consultas, comparado a uma proposta alternativa que também adota o conceito de tabelas de entidades.*

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 de como entidades do mundo real são identificadas. Nesse contexto, o W3C adotou 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 obrigatoriamente um esquema pré-definido. Sendo assim, dados RDF podem ser representados e armazenados em diferentes formatos. 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 Sistema Gerenciador de Banco de Dados Relacional (SGBDR) não deve ser descartado para conjuntos de dados de até milhões de triplas [Zeng et al. 2013]. Uma das vantagens de utilizar um SGBDR é tirar proveito de todo o investimento em pesquisa e desenvolvimento feito na

sujeito	predicado	objeto
s1	p1	'joao'
s1	p2	Pessoa
s1	p3	'0000-0001'
s1	p3	'0000-0002'
s1	p4	s3
s2	p1	'fabiana'
s2	p2	Pessoa
s2	p3	'0000-0003'
s2	p3	'0000-0004'

sujeito	predicado	objeto
s2	p4	s4
s3	p5	'moto1'
s3	p2	Automotor
s4	p5	'moto2'
s4	p2	Automotor
s5	p5	'carro1'
s5	p2	Automotor
s5	p7	'32784738'
s6	p8	'Contato de Debora'

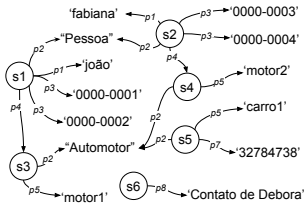

Figura 1. Base RDF em uma tabela SPO e sua representação em grafo

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 2. Exemplo de base relacional gerada pelo AORR

consolidação de sua tecnologia. A maneira mais direta de armazenar dados RDF em um SGBDR é no formato de triplas em uma tabela SPO, como ilustrado na Figura 1. O problema dessa abordagem é que o processamento de consultas SPARQL requer a execução de auto-junções sobre essa tabela, que tem cardinalidade igual a quantidade de triplas.

Exemplo 1: Considere a tabela T_{SPO} da Figura 1, onde são utilizados s_i e p_j para representar IRIs de sujeitos e predicados, respectivamente. Para facilitar a compreensão, uma representação equivalente em 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 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)}*. □

Para minimizar a quantidade de auto-junções para o processamento de consultas e diminuir o volume de dados envolvidos nas operações, este artigo propõe o AORR (Armazenamento Otimizado de dados RDF em um SGBD Relacional). A estratégia adotada pelo AORR é criar tabelas de maior grau, contendo predicados frequentemente associados a um mesmo sujeito. Para permitir que consultas SPARQL possam ser traduzidas para consultas SQL sobre a base relacional, o AORR mantém informações sobre o mapeamento em tabelas de metadados.

Exemplo 2: A base relacional gerada pelo AORR para o armazenamento das triplas na Figura 1 está ilustrada na Figura 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 propriedades 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 apenas com uma projeção sobre *PessoaRDF*. □

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])[1]. 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, neste artigo 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 ainda houve ganho de 1,34%.

As contribuições desse trabalho são: uma proposta de mapeamento de dados RDF para relacional; suporte à atualização incremental da base RDF; armazenamento de metadados que possibilitam a tradução de consultas SPARQL para SQL; e uma análise experimental que mostra a eficiência do sistema proposto. O restante do artigo está organizado da seguinte forma. A Seção 2 discute outras abordagens que utilizam um SGBDR para o armazenamento de RDF. O AORR é apresentado na Seção 3. A análise experimental é descrita na Seção 4 e a Seção 5 finaliza o artigo apresentando trabalhos futuros.

2. Trabalhos Relacionados

Existem diversas propostas para utilizar um SGBDR para o armazenamento RDF. Elas podem ser divididas entre abordagens verticais e horizontais. Nas abordagens verticais cada linha da tabela armazena uma única tripla da base. Já na abordagem horizontal, cada linha

[1]Maiores detalhes sobre as consultas suportadas pelo AORR podem ser obtidos em [Pauluk et al. 2018]

contém um conjunto de triplas. Um exemplo de abordagem vertical são as tabelas SPO, da Figura 1. Dentre os sistemas que adotam esta forma de armazenamento estão o Virtuoso[2] e Jena[3]. Já a proposta de [Abadi et al. 2007] propõe a criação de uma tabela para cada predicado distinto encontrado na base, o que pode resultar em uma grande quantidade de tabelas. Os trabalhos de [Bornea et al. 2013], [Scabora et al. 2017], [Aluç et al. 2014], [Pham et al. 2015] e [Ramunajam et al. 2009] adotam a abordagem horizontal. Os dois primeiros optam por manter um conjunto de k pares (predicado, objeto) ligados ao mesmo sujeito em uma mesma linha da tabela, o que pode resultar em uma grande quantidade de valores nulos. O terceiro utiliza informações da carga de consultas para gerar o esquema. Já os dois últimos fazem a extração de um esquema relacional a partir da base RDF.

A proposta de [Ramunajam et al. 2009] para a extração do esquema relacional tem a preocupação de criar tabelas que possam comportar todas as triplas RDF, baseando-se principalmente nas classes associadas aos sujeitos. Já a proposta ERSR [Pham et al. 2015] cria um esquema baseado em vários critérios, como quantidade mínima de registros na tabela, quantidade máxima de tabelas e frequência mínima da predicados. O esquema do ERSR é obtido a partir da identificação de *Characteristic Sets* (CSs), que correspondem aos conjuntos de predicados que podem estar associados a sujeitos. Avaliações experimentais [MahmoudiNasab and Sakr 2010] mostram que a abordagem horizontal com a geração de um esquema relacional tradicional apresenta melhor desempenho sobre as demais alternativas. A abordagem de [He et al. 2017] corrobora com estes resultados ao avaliar o processamento de consultas SPARQL sobre um SGBDR formado a partir de um esquema RDF pré-definido.

O AORR tomou como base o processo de extração de esquema do ERSR, porém estende este processo com a introdução de tabelas de overflow específico associadas às tabelas de entidades. As tabelas de overflow específico atuam na redução da quantidade de tuplas do overflow geral, além de dar suporte à atualização incremental da base. No ERSR a geração da base relacional ocorre uma única vez, sendo necessário recriar a base relacional para comportar novos dados. Além disso, a estrutura de metadados mantida pelo AORR oferece suporte à transformação de consultas SPARQL para SQL.

3. Armazenamento Otimizado de dados RDF em um SGBDR

Esta seção apresenta o AORR, um método de **A**rmazenamento **O**timizado de dados **R**DF em um SGBD **R**elacional. A estratégia adotada pelo AORR consiste em extrair um esquema relacional no qual predicados frequentemente encontrados em conjunto dão origem a tabelas *de entidades*. A Seção 3.1 apresenta os processos de extração de esquema e carga da base relacional. Para dar suporte à natureza semi-estruturada das bases RDF, o AORR cria tabelas de overflow para comportar dados que não se adequam às tabelas de entidades. As tabelas de overflow adotam o esquema SPO e são de dois tipos: específico e geral, conforme detalhado na Seção 3.2. As informações de mapeamento da base RDF para relacional são mantidas em tabelas de metadados, como detalhado na Seção 3.3.

3.1. Geração da Base Relacional

O processo de geração da base relacional consiste de 7 passos, que compõem o Algoritmo 1. O algoritmo recebe como entrada uma base RDF B e um conjunto de parâmetros de

[2]https://virtuoso.openlinksw.com/

[3]https://jena.apache.org/

configuração, e gera como saída uma base relacional R. Tais parâmetros afetam diretamente a geração da base relacional, dado que o min_t e Ub_{tbl}, quantidade mínima de registros e quantidade máxima de tabelas, respectivamente, determinam se uma tabela será inserida no overflow geral (OverflowG). Já o T_{inf}, frequência mínima, configura o quão frequente uma propriedade deve ser, determinando assim se ela será mantida na tabela de entidade ou no overflow específico (OverflowE). Uma base RDF B é formada por um conjunto de triplas (s, p, o). Sobre B, é definida a função *pred*, que associa um sujeito ao seu conjunto de predicados. Ou seja, *pred(s)* = $\{\ p \mid (s, p, o) \in B\}$. Por exemplo, o conjunto de triplas da Figura 1 compõe uma base RDF, na qual *pred*(s_5) = *{p2, p5, p7}*.

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
```

Passo 1: Criação da base categorizada
A função *criaBaseCategorizada* (Linha 2) identifica os *Characteristic Sets* (CSs) da base, ou seja, os conjuntos de predicados que podem estar associados a algum sujeito. O processo de identificação de CSs gera uma base categorizada, como definido abaixo:

Definição 1: Uma base RDF categorizada B_c de uma base RDF B é definida como um par (S, M), onde: S é um conjunto de CSs, e M é um mapeamento de S para um conjunto de sujeitos, tal que $M(c)$ = $\{\ s \mid pred(s)\ =\ c,\ c \in S\}$. □

A partir da base RDF do Exemplo 1, tem-se CS_1 = $\{p1, p2, p3, p4\}$, CS_2 = $\{p2, p5\}$, CS_3 = $\{p2, p5, p7\}$ e CS_4 = $\{p8\}$. Logo, o conjunto S = $\{CS_1, CS_2, CS_3, CS_4\}$, e $M(CS_1)$ = $\{s1, s2\}$, $M(CS_2)$ = $\{s3, s4\}$, $M(CS_3)$ = $\{s5\}$ e $M(CS_4)$ = $\{s6\}$.

Passo 2: Atribuição de rótulos
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 $M(c)$. A escolha do rótulo leva em consideração 3 características. A primeira é a existência do predicado *type*. Caso ele exista, o predicado *type* com maior frequência no CS, e superior a uma frequência mínima T_{inf}, será selecionado como rótulo. Por exemplo, o rótulo atribuído ao CS_1 é *Pessoa*, uma vez que $M(CS_1)$ = $\{s1, s2\}$ e ambos possuem o predicado *type* com valor *Pessoa*. A segunda característica considerada é o relacionamento entre CSs. A atribuição do rótulo se dá de acordo com o predicado que tem o CS em questão como objeto. Por exemplo, suponha que s_3 e s_4 não possuíssem o predicado *type*. Neste caso, como ambos são objetos da propriedade *veiculo*, este rótulo seria escolhido como $label(CS_2)$. Caso os dois casos anteriores não puderem ser aplicados, o CS recebe o rótulo da sua propriedade mais frequente, realizando apenas a remoção do prefixo. Por

exemplo `http://www.exemplo.br/observacao` é reduzido para `observacao`.

Passo 3: Junção de CSs
O passo de junção de CSs (Linha 4) tem por propósito diminuir a quantidade total de CSs, agrupando dados similares. A primeira junção é de CSs de mesmo rótulo que foram atribuídos pela regra de predicado *type*. No exemplo corrente, CS_2 e CS_3 são agrupados, uma vez que ambos possuem o rótulo *Automotor*, atribuídos devido ao predicado *type*. O parâmetro de configuração T_{inf} é utilizado para agrupar CSs referenciados pelo mesmo predicado ou que possuem similaridade superior a este limiar [Pham et al. 2015]. Como resultado da junção de dois CSs, cs_x e cs_y, o cs_y é removido do conjunto S, e os predicados e sujeitos de cs_y passam a pertencer a cs_x. Ou seja, $pred(cs_x)$ recebe $(pred(cs_x) \cup pred(cs_y))$ e $M(cs_x)$ recebe $(M(cs_x) \cup M(cs_y))$. No exemplo corrente, após a junção, $S = \{CS_1, CS_2, CS_4\}$, $pred(CS_2) = \{p2, p5, p7\}$ e $M(CS_2) = \{s3, s4, s5\}$.

Passo 4: 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.

Definição 2: Considere um conjunto de tipos de literais L e uma base RDF categorizada $B_c = (S, M)$, com função de atribuição de rótulos $label$. A estrutura map é um vetor multidimensional, tal que o elemento $map[c][p][k][t]$ contém uma tupla *(nomeTabela, nomeColuna, flg_multivalorado, flg_emOverflowE)*, 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. Na tupla associada: *nomeTabela* contém o nome da tabela na qual o predicado p de c é armazenado e é obtido pela concatenação de $label(c) +' RDF'$; *nomeColuna* é o atributo em *nomeTabela* que contém p e é obtido pela remoção do prefixo de p; *flg_multivalorado* contém *true* se o predicado p é multivalorado e *false*, caso contrário; *flg_emOverflowE* contém *true* se o predicado p está na tabela de overflow específico de *nomeTabela* e *false*, caso contrário. □

Para exemplificar, um dos elementos da estrutura map no exemplo corrente é $map[CS_1][p1][\texttt{lit}][string] = (' PessoaRDF',' nome', false, false)$, uma vez que o CS_1 contém o predicado $p1$, com valor literal do tipo string, que será armazenado na tabela *PessoaRDF* na coluna *nome*. Além disso, este predicado não é multivalorado (*flg_multivalorado* é $false$) e nem é armazenado na tabela de overflow específico (*flg_emOverflowE* é *false*). Para a geração da estrutura, a base RDF B é percorrida a fim de criar, para cada CS c, 3 conjuntos: *multiValued(c)*, que contém os predicados multivalorados de c; *literalPred(c)* e *linkPred(c)*, que contêm os predicados com valores literais e com ligações para outros CSs, respectivamente, juntamente com seus tipos associados. No exemplo adotado, para CS_1 são criadas as seguintes listas: *multiValued(CS_1) = {$p3$}*, *literalPred(CS_1) = {($p1$,string),($p2$,string),($p3$,string)}* e *linkPred(CS_1) = {($p4$,CS_2)}*. A partir destes conjuntos, a estrutura map é preenchida. Os elementos da estrutura map associados a CS_1 resultantes são:

map[CS_1][$p1$][`lit`][string] = ('PessoaRDF', 'nome', false, false),
map[CS_1][$p2$][`lit`][string] = ('PessoaRDF', 'type', false, false),
map[CS_1][$p3$][`lit`][string] = ('telefoneMultivalueRDF', 'telefone', true, false),
map[CS_1][$p4$][`fk`][CS_2] = ('PessoaRDF', 'fk_veiculo', false, false).

Observe que o nome da tabela é alterado quando o atributo é multivalorado, sendo de-

finido pela concatenação do nome do atributo com 'MultivalueRDF'. De forma similar, atributos que são `fk`, tem o seu nome acrescido do prefixo 'fk_'. Além disso, o valor *false* é atribuído para *flg_emOverflowE* em todos os elementos. Isso porque a transferência de atributos e tabelas para o overflow é feita no passo de filtragem, que corresponde ao processo de refinamento do esquema.

Passo 5: Filtragem
No passo de filtragem (Linha 6), tabelas menos significativas são movidas para o overflow geral, e atributos menos significativos para o overflow específico respectivo. Inicialmente é verificado se cada CS possui a quantidade mínima de sujeitos (min_t), que é um parâmetro de configuração. Caso não possua, ele é migrado para a tabela de overflow geral. No exemplo, se min_t for maior ou igual a 2, o CS_4 é movido para o *Overflow*, uma vez que $|M(CS_4)|$ = 1. Como resultado, a estrutura *map* é alterada, removendo a entrada $map[CS_4][p8]$[`lit`][string] e inserindo a entrada $map[CS_{over}][p8]$[`lit`][string] com o valor *('Overflow', 'observacao', false, false)*. Além disso, os sujeitos em $M(CS_4)$ passam a pertencer a $M(CS_{over})$, onde CS_{over} é o CS introduzido para ser associado ao overflow geral. A filtragem contém ainda a verificação se a quantidade de CSs é maior do que o máximo de tabelas permitido (Ub_{tbl}). Caso seja maior, os CSs com menor quantidade de sujeitos também são migrados para o *Overflow*. A única exceção é feita para tabelas dimensionais, pelo fato delas serem frequentemente referenciadas por outras tabelas. No exemplo, caso $Ub_{tbl} = 1$ e $S = (CS_1, CS_2)$, como $|M(CS_1)| < |M(CS_2)|$ e CS_1 não é uma tabela dimensional, ela seria migrada para o *Overflow*.

Por fim, é realizado o processo de minimização de propriedades infrequentes a fim de que as propriedades com baixa frequência sejam eliminadas das tabelas de entidades e colocadas no overflow específico, uma vez que elas ocasionariam colunas com muitos valores nulos. O parâmetro T_{inf} é utilizado para determinar se um predicado é infrequente. Seguindo o exemplo, suponha que T_{inf} seja 0.4. No CS_2 o predicado $p7$ está presente em apenas um sujeito ($s5$) e $|M(CS_2)| = 3$. Como a presença é de $1/3 = 0.33 < T_{inf}$, $p7$ é movido para o overflow específico de CS_2. Logo, a estrutura *map* é alterada para conter o valor *true* para o *flg_emOverflowE*. Assim, *map*$[CS_2]$*[p7][*`lit`*][int]* passa o conter o valor *('Overflow_AutomotorRDF', 'cod_modelo', false, true)*.

Passo 6: Criação de tabelas
Baseado na estrutura map, as tabelas para armazenar as triplas RDF são geradas. Cada tabela de entidade é criada contendo todos os atributos que *flg_multivalorado* e *flg_emOverflowE* possuem valor *false*. Assim, no exemplo são criadas as seguintes tabelas: *PessoaRDF (OID, nome, type, fk_veiculo)*, onde *OID* é um identificador numérico criado pelo sistema, e *AutomotorRDF (OID, modelo, type)*. Cada tabela de entidade tem uma tabela de overflow específico associada: *Overflow_PessoaRDF* e *Overflow_AutomotorRDF*. Além disso, é criada uma tabela para cada entrada que possui o *flg_multivalorado* com valor *true*. No exemplo, é criada a tabela *telefoneMultivalueRDF*. Estas tabelas são ilustradas na Figura 2. É importante observar que todas as informações necessárias para a criação das tabelas encontram-se na estrutura map.

A estrutura map contém também informações para a inserção de dados nas tabelas de metadados *TB_FullPredicate* e *TB_DatabaseSchema*. A tabela *TB_FullPredicate* tem como objetivo associar as IRIs completas dos predicados (p_i) com os CSs que os possuem e as tabelas e atributos nos quais estão armazenados. Assim, para cada entrada

na estrutura $map[c][p][k][t]$ com valor *(nomeTabela, nomeColuna, flg_multivalorado, flg_emOverflowE)*, é gerada uma linha na tabela *TB_FullPredicate*, com os seguintes valores: *cs_identifier:=* c; *tableName:= null* se *flg_emOverflowE* for *true* e *nomeTabela*, caso contrário; *overflowTable:= null* se *flg_emOverflowE* for *false* e *nomeTabela*, caso contrário; *columnName:= nomeColuna*; e *predicate:=* p. A Figura 2 apresenta a tabela *TB_FullPredicate* do exemplo corrente.

A tabela *TB_DatabaseSchema* armazena informações sobre o mapeamento da base RDF para o relacional. Para cada CS ela mantém: o identificador do CS (*cs_identifier*), nome do predicado sem o prefixo (*propertyName*), tipo do objeto (*valueType*), nome da tabela em que se encontra armazenado o predicado (*tableName*) e o nome do atributo (*tableAttribute*). O *valueType*, quando não for *literal*, contém o identificador do CS referenciado. Considere a tabela *TB_DatabaseSchema* da Figura 2. O *valueType* do predicado *fk_veiculo* de CS_1 é CS_2. Isso significa que o atributo *fk_veiculo* da tabela *PessoaRDF* é uma chave estrangeira que referencia a tabela *AutomotorRDF*.

Passo 7: Carregamento de Dados
Cada tripla da base RDF B só pode ser inserida em uma das quatro tabelas: *Overflow*, tabela de entidade, tabela de overflow específico associada ou tabela de propriedade multivalorada. Estas informações encontram-se na estrutura map. Além disso, para cada CS c, $M(c)$ contém o conjunto de sujeitos que pertencem a c. Portanto, para cada tripla (s, p, o) determina-se a qual $M(c)$ o sujeito s pertence, cria-se um novo OID (caso ele ainda não exista) e insere-se o valor na tabela apropriada, de acordo com a estrutura map. Além disso, a tabela de metadados *TB_Subj_OID* associa as IRIs completas do sujeitos (s_i) aos seus identificadores (OIDs), como mostrado na Figura 2.

Com isso, vale salientar as diferenças do AORR nos passos que se assemelham aos do ERSR. Dado que o AORR tem o intuito de ser utilizado como *backend* de armazenamento, a atribuição por propriedade discriminativa e a junção por ancestral ontológico do ERSR não foram utilizadas no AORR. Já no passo de filtragem a diferenciação se dá por não existir OverflowE no ERSR, ou seja, o que no AORR faz menção a OverflowE, no ERSR seria o OverflowG.

3.2. Atualização da Base

. 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. 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, 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* com pelo menos uma linha com valor igual a o. Caso esta tabela não exista, a tripla é armazenada no overflow geral. Uma situação importante a ser observada é que outros predicados do sujeito s podem ter sido anteriormente inseridos na base e, por não ter sido possível determinar sua tabela de entidade associada, elas foram armazenadas

no overflow geral. Assim, a inserção de uma tripla $(s, type, o)$ pode resultar na migração de triplas do overflow geral para uma tabela de entidade.

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*. 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.

Exemplo 3: Considere a inserção da tripla $(s5, p5,' carro\ popular')$ na base ilustrada na Figura 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)*. □

3.3. 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 3(a) executada sobre a base ilustrada na Figura 2 após a inserção da tripla $(s5, p5,' carro\ popular')$ do Exemplo 3. A consulta SQL resultante da tradução está ilustrada na Figura 3(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*.

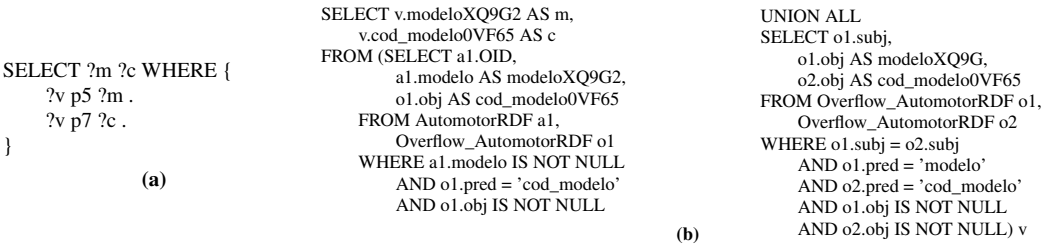

Figura 3. Consulta SPARQL traduzida em consulta SQL

Elas são unidas com o operador UNION ALL para produzir o resultado final.

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.

4. Análise Experimental

As tabelas de overflow específico dividem a carga do overflow geral de forma a diminuir o volume de dados envolvidos no processamento das consultas. O experimento relatado nesta seção avalia o impacto desta estratégia no tempo de processamento de consultas, utilizando a base RDF Peel[4], que possui 276160 triplas e 45 MBytes. As características da base relacional gerada pelo AORR estão descritas na Tabela 1, com um volume total de 29 MBytes. O sistema foi implementado na linguagem Python 2.7, utilizando o SGBD MySQL v 14.14. Os experimentos foram executados em um computador com processador Intel i7-4710HQ, 2.50GHz e 2GB de memória, com S.O Linux 3.13.0. Os parâmetros de entrada de quantidade máxima de tabelas, quantidade mínima de registros e frequência mínima utilizados foram 7, 1000 e 0.1 respectivamente. Tais parâmetros foram adotados para que todos os passos do algoritmo fossem executados e algumas tabelas fossem migradas para o OverflowG pelo processo de filtragem.

Foram elaboradas 5 consultas (C1,...,C5) que exploram diferentes situações nas quais as tabelas de overflow são utilizadas [Pauluk et al. 2018]. Tais consultas foram traduzidas de SPARQL para SQL como detalhado na Seção 3.3. As consultas foram executadas sobre duas bases, sendo que uma delas possui tabelas de overflow específico (com OverflowE) e a outra apenas o overflow geral (sem OverflowE). Foram utilizadas bases de dados com 6 tamanhos: a base Peel original, e 5 outras geradas a partir dela com incrementos de 5000, 10000, 20000, 40000 e 80000 triplas nas tabelas de overflow, para cada predicado utilizado nas consultas. Por exemplo, para uma consulta que envolve 2 predicados associados a um mesmo sujeito, o incremento de 5000 gera 5000 novos sujeitos e triplas suficientes para gerar 5000 novas linhas em tabelas de overflow para cada predicado, totalizando um incremento de 10.000 linhas de overflow. Na discussão abaixo a base com overflow específico e incremento N será chamada de R^N_{AORR} e a base sem overflow específico é denotada por R^N_{ERSR} e contém uma única tabela de overflow geral chamada de *OverflowG*. A Figura 4 apresenta os resultados para as cinco consultas nas bases R_{AORR} e R_{ERSR}, para cada tamanho de base. Os tempos são reportados em milis-

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

Tabela 1. Base relacional gerada pelo AORR

Tabela	# Linhas	Volume (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

Tabela	# Linhas	Volume (MB)
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

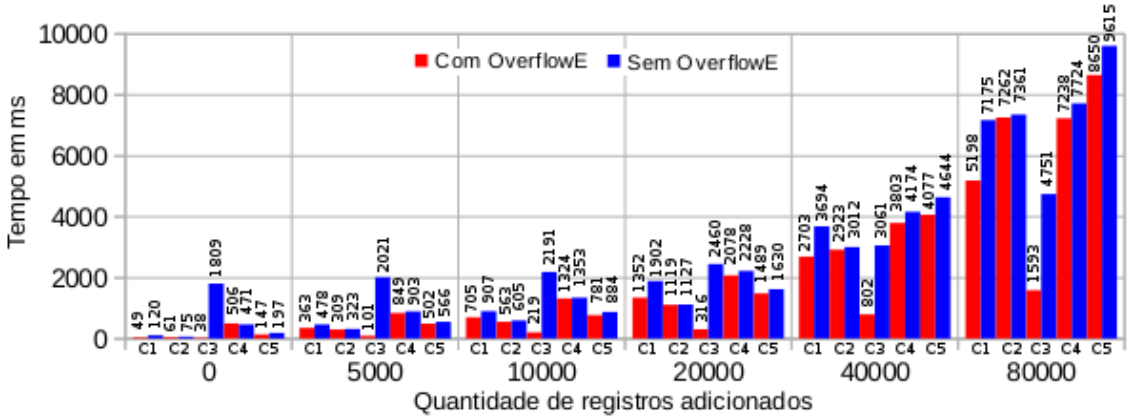

Figura 4. Resultado das consultas para cada incremento em milissegundos

segundos e consideram apenas o tempo de processamento da consulta, sem considerar o tempo de tradução. O único índice criado sobre as tabelas foi sobre o atributo OID.

A consulta na qual o AORR apresentou maior ganho foi a C3 com um tempo de processamento 198% menor para a base R^{80000}_{AORR} comparada à base R^{80000}_{ERSR}. Isso se deve ao fato da consulta envolver apenas tabelas de overflow. Na base R^{80000}_{ERSR}, a consulta executa duas auto-junções sobre o *OverflowG*, que possui aproximadamente 820.000 registros. Em contrapartida, na base R^{80000}_{AORR} é executada uma auto-junção de *Overflow_PersonRDF*, que possui 160.000 registros aproximadamente, e uma junção com *Overflow_RecordingRDF*, com 80.000 registros. Já a consulta C1 busca por dois predicados, ambos contidos tanto na tabela *SignalRDF* como no *Overflow_SignalRDF*. Com isso, a C1 tem 4 subconsultas, precisando realizar três operações UNION ALL, para que todas as combinações sejam contempladas. Como pode ser percebido, o desempenho nas bases R_{AORR} para C1 aumentam proporcionalmente em relação aos incrementos realizados. Já na base R_{ERSR}, o crescimento é maior, dado que a tabela *OverflowG* cresce em uma proporção maior que *Overflow_SignalRDF*. Além disso, como ambos os atributos estão em tabelas de overflow, uma auto-junção sobre *OverflowG* é bem mais custosa do que sobre *Overflow_SignalRDF*. A consulta C4 realiza a busca por apenas um predicado, sendo que ele está contido em 9 tabelas, sendo 2 overflow específicos e 1 overflow geral na base R_{AORR}. Já na base R_{ERSR} a consulta é realizada sobre 7 tabelas, sendo uma delas o *OverflowG*. Como no cenário sem incremento as tabelas de overflow são relativamente pequenas, consultar 2 tabelas de overflow específico no R_{AORR} foi mais custoso do que na base R_{ERSR}. Todavia, já no incremento de 5000, a consulta sobre o R_{AORR} é mais vantajosa do que sobre o R_{ERSR}, dado que o *OverflowG* cresce mais do que as tabelas de overflow específico. Para a C5, como existem 6 junções em cada subconsulta e duas

subconsultas para um UNION ALL, os resultados mostram que a consulta sobre a base R_{AORR} em relação à base R_{ERSR} começa a apresentar um ganho a partir do incremento de 40000. Logo, não houve grande impacto nas consultas de menor incremento.

5. Conclusão

Esse artigo apresentou o AORR, uma abordagem para armazenar dados RDF em um SGBDR e habilitar o processamento otimizado de consultas de casamento básico de grafos. O AORR dá suporte à atualização incremental da base e à tradução de consultas SPARQL em consultas SQL. Os resultados apresentados mostram que consultas realizadas sobre a base gerada pelo AORR apresentaram melhor desempenho do que uma simulação da base gerada pelo ERSR [Pham et al. 2015]. Tal ganho se deu principalmente devido às tabelas de overflow específico. As consultas realizadas sobre as bases de teste foram geradas a partir de um processo de tradução de consulta SPARQL em consulta SQL, que é possível graças às tabelas de metadados propostas pelo AORR. Dentre os trabalhos futuros podem ser citados: desenvolvimento de um processo de migração de triplas na tabela de overflow geral para tabelas de entidades; geração automática dos parâmetros de configuração usados para a geração do esquema relacional; exploração de recursos de otimização do SGBDR, como indexação e caching; e execução de experimentos com outras bases de dados e *benchmarks*.

Referências

Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422.

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

Bornea, M., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., and Bhattacharjee, B. (2013). Building an efficient rdf store over a relational database. In *ACM SIGMOD*.

He, L., Shao, B., Li, Y., Xia, H., Xiao, Y., Chen, E., and Chen, L. J. (2017). Stylus: A strongly-typed store for serving massive rdf data. *Proc. VLDB Endow.*, 11(2):203–216.

MahmoudiNasab, H. and Sakr, S. (2010). An experimental evaluation of relational rdf storage and querying techniques. In *Proc. of DASFAA*, pages 215–226.

Pauluk, J. G., Duarte, M. M. G., Prado, R. L., and Hara, C. S. (2018). Processamento de Consultas SPARQL em uma Base Relacional de Entidades. In *SBBD - Short Papers*.

Penteado, R. R. M., Schroeder, R., and Hara, C. S. (2015). Exploração de grafos RDF com distribuição controlada. In *Anais do XXX SBBD - Short Papers*, pages 69–74.

Pham, M.-D., Passing, L., Erling, O., and Boncz, P. (2015). Deriving an emergent relational schema from rdf data. *Proc. of the 24th WWW Conf.*, pages 864–874.

Ramunajam, S., Gupta, A., Khan, L., Seida, S., and Thurasaisingham, B. (2009). R2d: Extracting relational structure from rdf stores. In *Proc. of the IEEE/ACM WIC*, pages 361–366.

Scabora, L. C., Oliveira, P. H., Kaster, D. S., Traina, A. J. M., and Traina-Jr, C. (2017). Relational graph data management on the edge: Grouping vertices' neighborhood with edge-k. In *Anais do XXXII SBBD*, pages 124–135.

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