Otimização do Mapeamento de Consultas SPARQL para SQL

Mariana Machado Garcez Duarte e Carmem S. Hara

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

marianamgd@gmail.com, carmem@inf.ufpr.br

Resumo. *A Web Semântica tem como princípio a organização e publicação das informações com conteúdo semântico. Ela adota o modelo RDF como padrão de armazenamento e a linguagem SPARQL para consultas. A grande quantidade de dados de RDF existente requer que as consultas SPARQL sejam processadas de forma eficiente. Os trabalhos de [de Lima Prado 2018] e [G. Pauluk and Hara 2016] buscaram uma solução para este problema através do armazenamento de dados RDF em um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) e da tradução de consultas SPARQL para SQL. Este artigo apresenta uma investigação que deu continuidade a esses trabalhos, com o objetivo de otimizar o mapeamento de consultas SPARQL para SQL, utilizando índices, visões e filtros. Os experimentos realizados determinaram o impacto desses recursos no desempenho das consultas. Constatou-se que a implementação de visões é altamente recomendável, com uma redução do tempo de processamento da consulta de até 54,4%. A utilização de filtros que desconsideram tuplas contendo valores nulos resultou em uma redução do tempo de processamento da consulta de até 58,8%.*

1. Introdução

A *Web* Semântica é uma extensão da *World Wide Web*, que organiza informações, de tal forma que permite que computadores e humanos trabalhem em cooperação, através da organização e publicação de dados no formato RDF. Uma base de dados RDF é composta por um conjunto de triplas *(sujeito, predicado, objeto)*. SPARQL é a linguagem utilizada para consultas sobre uma base de dados RDF, na qual padrões de triplas são combinados para a geração de um resultado. A grande quantidade de dados de RDF existente na *Web* requer que as consultas SPARQL sejam processadas de forma eficiente. Há várias formas de obter esse objetivo, sendo uma delas mapear os dados RDF para o modelo relacional e consultas SPARQL para SQL. Assim, é possível aproveitar as otimizações que um SGBDR oferece sobre a linguagem de consulta SQL.

Buscando essa solução, o trabalho de [de Lima Prado 2018] propõe o Sistema de Armazenamento Otimizado de Dados RDF em SGBDR (AORR). A estratégia adotada pelo AORR difere da abordagem direta de armazenamento RDF no modelo relacional, na qual triplas RDF são armazenadas um uma única relação com 3 atributos (uma tabela SPO - sujeito, predicado, objeto). Este mapeamento direto resulta em uma relação com cardinalidade igual ao número de triplas. No AORR, predicados de um mesmo sujeito são agrupados em uma única tupla, gerando tabelas *estruturadas*.

Dada a natureza semi-estruturada das bases RDF, nem todas as informações podem ser mapeadas para as tabelas estruturadas. Assim, o trabalho de [de Lima Prado 2018],

propõe a criação de tabelas de *Overflow* Específico, a tabela de *Overflow* Geral e as tabelas de metadados. As tabelas de *Overflow* correspondem à parte *não estruturada* da base relacional. É criada uma tabela de *Overflow* Específico para cada tabela estruturada para armazenar informações dos sujeitos pertencentes à tabela estruturada, mas que não se adequam ao seu esquema. Assim, se existe uma tabela estrutura para `Pessoa`, é criada também uma tabela SPO `Overflow_Pessoa`. Já a tabela *Overflow* Geral possui os sujeitos que não pertencem a nenhuma tabela estruturada. As tabelas de metadados fornecem informações que permitem a tradução de consultas SPARQL para SQL, sem que o usuário tenha conhecimento do esquema de mapeamento da base RDF para a base relacional. O trabalho de [G. Pauluk and Hara 2016], utiliza essas tabelas de metadados para gerar traduções do SPARQL para o SQL.

O trabalho apresentado neste artigo teve como objetivo dar continuidade a esses trabalhos com a otimização do mapeamento de consultas SPARQL para SQL de [G. Pauluk and Hara 2016], explorando a utilização de índices, filtros e visões. O objetivo dos experimentos realizados foi determinar o impacto destes recursos no desempenho das consultas. Para atingir este objetivo, um plano de atividades foi traçado, o qual possuiu três etapas e testes. Como primeira etapa, foram criados índices nas tabelas estruturadas e seus resultados foram comparados com a situação original. Como etapa seguinte, foram investigadas diferentes estratégias de consultar as tabelas de *Overflow* Específico, como através da criação de visões. Como última etapa, foram utilizadas consultas com filtros que desconsideraram tuplas contendo valores nulos.

O restante do artigo está estruturado da seguinte forma. A Seção 2 apresenta trabalhos relacionados. A Seção **??** detalha o método para transformar o RDF em relacional, proposto em [de Lima Prado 2018] bem como exemplifica a tradução de consultas proposta em [G. Pauluk and Hara 2016]. A Seção 4 descreve os experimentos realizados e seus resultados. A Seção 5 finaliza o artigo enumerando alguns trabalhos futuros.

2. Trabalhos Relacionados

Diversas abordagens para o armazenamento de dados RDF em um SGBDR foram propostas na literatura nos últimos anos. O trabalho de [Abadi et al. 2007] utiliza a abordagem baseada em propriedades, com a criação de uma tabela para cada propriedade distinta. A proposta pode resultar em uma grande quantidade de tabelas, dependendo da base RDF sendo tratada. Já o trabalho de [Bornea et al. 2013] é orientado a entidades*(entity-oriented)* e cria quatro tabelas, duas para tratar atributos multivalorados, uma para sujeitos e a última para objetos. O trabalho de [Scabora et al. 2017] mapeia todas a triplas RDF para uma única tabela, contendo múltiplas colunas. Cada vértice corresponde a uma ou mais linhas da tabela, que é preenchida com suas propriedades, até atingir uma quantidade k de colunas. Após esta quantidade ser atingida, uma nova linha é inserida na tabela. Já o AORR foi inspirado na proposta de [Pham et al. 2015], que tem por objetivo criar tabelas que agrupam sujeitos com estruturas semelhantes ou que sejam semanticamente relacionados. No entanto, a tradução de consultas SPARQL para SQL não é tratada, já que a base relacional é criada para ser diretamente consultada através do SGBDR.

3. Armazenamento de RDF em um SGBDR no Sistema AORR

Para explorar a flexibilidade do RDF com a otimização de um SGBDR, foi proposto o Sistema de Armazenamento Otimizado de Dados RDF em SGBDR (AORR), que utiliza um

SGBDR como *backend* de armazenamento RDF e processamento de consulta SPARQL. O AORR possui um módulo de extração de estrutura de armazenamento inspirado na proposta de [Pham et al. 2015], o qual é baseado no conceito de *characteristic sets* e que tem por objetivo identificar estruturas comuns dos sujeitos da base RDF. Um CS é definido como um conjunto de predicados. Um sujeito na base RDF *pertence* a um determinado *characteristic set* cs_1 se ele possui o conjunto de predicados cs_1. Desta forma, após um processo de agrupamento, remoção e limpeza de CSs, uma tabela *estruturada* é gerada para cada CS resultante. Elas agrupam em uma tabela predicados que são comumente encontrados para um mesmo sujeito. Os dados inseridos nestas tabelas formam a parte *estruturada* da base. Como resultado, além da base relacional, são mantidas informações sobre as relações entre componentes da base original RDF e a base relacional criada. Elas são armazenadas em uma tabela denominada `TB_DatabaseSchema`.

Para dar suporte à heterogeneidade do RDF e permitir atualizações com triplas que não se adequam ao esquema relacional extraído, o AORR mantém um conjunto de tabelas SPO (chamadas de tabelas de *Overflow*), que correspondem à parte *não estruturada* da base relacional. Existem dois tipos de tabelas de *Overflow*: específica e geral. Existe uma tabela de *Overflow* específica para cada tabela estruturada. Ela armazena, por exemplo, predicados que são incomuns aos sujeitos armazenados na tabela estruturada. A tabela de *Overflow* geral armazena informações sobre sujeitos que não se adequam ao esquema das tabelas estruturadas ou que foram inseridos posteriormente à geração da base relacional.

sujeito	predicado	objeto
<http://dbtune.org/.../60678 >	Name	Andy Halstead
<http://dbtune.org/.../60678>	Type	<http://xlmns.com/foaf/0.1/Person>
<http://dbtune.org/.../60678>	Label	AndyHalstead
<http://dbtune.org/.../14002>	Name	Cat Stevens
<http://dbtune.org/.../14002>	Name	Yusuf
<http://dbtune.org/.../14002>	Type	<http://purl.org/ontology/mo/MusicArtist>
<http://dbtune.org/.../25789>	Type	<http://purl.org/ontology/mo/Performance>
<http://dbtune.org/.../25789>	Fk_Performer	<http://dbtune.org/.../60678>
<http://dbtune.org/.../25789>	Fk_Recorded_as	<http://dbtune.org/.../2591>
<http://dbtune.org/.../76229>	Time	1998-07-05

Tabela 1. Tabela SPO

Considere, por exemplo, uma parte de uma tabela SPO apresentada na Figura 1. O resultado da base relacional resultante do processo AORR, está ilustrado na Figura 1. Esta base possui 2 tabelas estruturadas (`MusicArtistRDF`) e (`PerformanceRDF`). Cada uma delas contém uma coluna para cada predicado comumentemente encontrados nos sujeitos que pertencem ao CS. Predicados infrequentes, multivalorados ou com tipos distintos são armazenados na tabela de *Overflow* Específico de cada tabela estruturada. Assim, no exemplo, são ilustradas duas tabelas de *Overflow* específico (`Overflow_MusicArtistRDF`) e (`Overflow_PerformanceRDF`). Além das tabelas de *Overflow* Específico, há a tabela geral chamada de *Overflow*. Ela comporta os sujeitos que não se adequam a nenhuma tabela estruturada.

O AORR gera diversas tabelas de metadados, que contem as informações sobre o mapeamento da base RDF para o esquema relacional. As principais são : `TB_DatabaseSchema` e `TB_Subj_OID`. A tabela `TB_DatabaseSchema` relaciona os predicados de cada CS às tabelas e atributos nos quais eles são armazenados, além do tipo de cada atributo. A Figura 2 ilustra um `TB_DatabaseSchema`.

Music Artist RDF		
OID	Name	Type
60678	Andy Halstead	<http://xmlns.com/foaf/0.1/ Person>
14002	Cat Stevens	<http://purl.org/ontology/mo/ MusicArtist>

Overflow_MusicArtistRDF		
Subj	Pred	Obj
60678	Label	AndyHalstead
14002	Name	Yusuf

PerformanceRDF			
OID	fk_performer	fk_performance_of	Type
25789	60678	NULL	ttp://purl.org/ontology/mo/ Performance>

Overflow_PerformanceRDF		
Subj	Pred	Obj
25789	fk_recorded_as	25791

Overflow		
OID	Pred	Obj
76229	Time	1988-07-05

Figura 1. Estrutura gerado pelo AORR

cs_identifier	PropertyName	VakueType	TableName	TableAttibute
CS1	OID	Literal	MusicArtistRDF	OID
CS1	Name	Literal	MusicArtistRDF	name
CS1	Type	Literal	MusicArtistRDF	type
CS2	OID	Literal	PerformanceRDF	OID
CS2	Type	Literal	PerformanceRDF	type
CS2	fk_performer	CS1	PerformanceRDF	fk_performer
CS2	fk_performance_of	CS5	PerformanceRDF	fk_performance_of
CS1	Name	Literal	Overflow_MusicArtistRDF	pred
CS1	Label	Literal	Overflow_MusicArtistRDF	pred
CS2	fk_recorded_as	CS3	Overflow_PerformanceR DF	pred
...	...	...	...	...
...	...	...	...	...
CSover	Label	Literal	Overflow	pred
CSover	Time	Literal	Overflow	pred
CSover	Type	Literal	Overflow	pred

Figura 2. Exemplo de um `TB_DatabaseSchema`

Na tabela `TB_Subj_OID` encontram-se informações sobre a relação da IRI do sujeito para o OID que a representa, e também, a tabela na qual esse sujeito está armazenado. Essa tabela também é utilizada para identificar se a IRI de um determinado sujeito, ou de um objeto de um relacionamento, já existem na base. A Figura 3 ilustra um exemplo desta tabela. As tabelas `TB_DatabaseSchema` e `TB_Subj_OID` possibilitam a tradução de uma consulta SPARQL para SQL. A Figura 4 apresenta um exemplo de consulta SPARQL e a consulta SQL resultante da tradução proposta em [G. Pauluk and Hara 2016].

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

Figura 3. Exemplo de um `TB_Subj_OID`, como visto em [de Lima Prado 2018]

4. Propostas para otimização do Sistema AORR

Esta seção apresenta os experimentos realizados para determinar o impacto de algumas estratégias de otimização sobre as propostas de armazenamento e tradução de consultas

```
SELECT ?a ?n ?t ?b
WHERE{
    ?a name ?n.
    ?a type ?t.
    ?b fk_performer ?a.
}
```

(a) Consulta SPARQL

```
SELECT b.OID AS b,
       a.typeYLEIFC AS t,
       a.nameEY9TOM AS n,
       a.OID AS a
FROM
  (SELECT MusicArtistRDF.OID,
          MusicArtistRDF.name AS nameEY9TOM,
          MusicArtistRDF.type AS typeYLEIFC
   FROM MusicArtistRDF
  UNION ALL
   SELECT t1.subj as a, t1.obj as t, t2.obj as n
   FROM Overflow_MusicArtistRDF as t1
        FULL OUTER JOIN Overflow_MusicArtistRDF as t2
        on t1.subj=t2.subj
   WHERE t1.pred='name' and t2.pred='type') AS a,
  (SELECT PerformanceRDF.OID,
          PerformanceRDF.fk performer AS fk_performerEKRX08
   FROM PerformanceRDF) AS b
WHERE a.OID = fk performerEKRX08
```

(b) Consulta SQL

Figura 4. Tradução SPARQL para SQL

apresentados na Seção **??**.

Com o objetivo de otimizar o mapeamento de consultas SPARQL para SQL, resultante dos trabalhos de [de Lima Prado 2018] e [G. Pauluk and Hara 2016], um plano de implementação foi traçado, o qual possuiu três etapas e testes. Como primeira etapa, foram criados índices nas tabelas estruturadas e seus resultados foram comparados com a situação original. Como etapa seguinte, foram investigadas diferentes estratégias de consulta à tabelas de *Overflow* Específico, como através da criação de visões. Como última etapa, foram utilizadas consultas com filtros que desconsideraram tuplas contendo valores nulos.

4.1. Experimentos

O computador utilizado para executar os experimentos foi um MacOSX Intel Core m3 1.1 GHz e com 8 GB de memória RAM. O SGBDR utilizado foi o MySQL com o mecanismo de armazenamento InnoDB. Suas especificações estão na Figura 5a.

O banco de dados utilizado possui 26 tabelas, que foram geradas a partir do processo de [de Lima Prado 2018], com o sistema AORR a partir da base de dados Peel, disponível em http://dbtune.org/bbc/peel/. A Figura 5b apresenta as tabelas geradas e seus tamanhos.

As tabelas estruturadas utilizadas nos experimentos são: MusicArtistRDF, PerformanceRDF, RecordingRDF, SignalRDF, TrackRDF. As tabelas multivaloradas, que correspondem a predicados multivalorados da base são: chart_positionMultivalueRDF, createdMultivalueRDF, fk_engineerMultivalueRDF, fk_engineeredMultivalueRDF, fk_performedMultivalueRDF, fk_producedMultivalueRDF, fk_sameAsMultivalueRDF, fk_sub_eventMultivalueRDF, instrumentMultivalueRDF, isrcMultivalueRDF, labelMultivalueRDF.

Como a base de dados Peel não disponibiliza um conjunto de consultas, foram definidas 6 consultas para executar os experimentos. As primeiras 5 consultas variam a complexidade pela quantidade de tabelas, enquanto a consulta 6 inclui uma tabela multivalorada. Seguem as descrições das consultas.

Variable_name	Value
innodb_version	5.7.18
protocol_version	10
slave_type_conversions	
tls_version	TLSv1,TLSv1.1,TLSv1.2
version	5.7.18
version_comment	Homebrew
version_compile_machine	x86_64
version_compile_os	osx10.12

(a) Detalhamento da Configuração do MySQL.

Table Name	Rows Count	Table Size (MB)
MusicArtistRDF	10544	1.52
Overflow	16319	1.52
Overflow_MusicArtistRDF	2842	0.27
Overflow_PerformanceRDF	12958	1.52
Overflow_RecordingRDF	6	0.02
Overflow_SignalRDF	152	0.02
Overflow_TrackRDF	383	0.06
PerformanceRDF	28600	2.52
RDF_Tripla	268590	44.58
RecordingRDF	3924	0.52
SignalRDF	5658	0.42
TB_DatabaseSchema	60	0.02
TB_FullPredicate	38	0.02
TB_Subj_OID	76455	7.52
TrackRDF	19335	2.52
chart_positionMultivalueRDF	1502	0.08
createdMultivalueRDF	1522	0.08
fk_engineerMultivalueRDF	3801	0.16
fk_engineeredMultivalueRDF	3801	0.16
fk_performedMultivalueRDF	11924	0.44
fk_producedMultivalueRDF	3640	0.16
fk_sameAsMultivalueRDF	126	0.02
fk_sub_eventMultivalueRDF	29590	1.52
instrumentMultivalueRDF	9098	0.41
isrcMultivalueRDF	689	0.06
labelMultivalueRDF	5787	0.28

(b) Configuração das entradas de dados por tabela.

Figura 5. Ambiente Experimental

Consulta 1: faz uma busca na tabela `MusicArtistRDF`, extraindo o nome e tipo dos artistas.

Consulta 2: faz uma busca nas tabelas `MusicArtistRDF` e `PerformanceRDF`, retornando o nome e tipo do artista, além do local no qual o artista realizou uma performance. Esta é a consulta ilustrada na Figura 4.

Consulta 3: faz uma busca nas tabelas `RecordingRDF`, `SignalRDF` e `TrackRDF`. Ela procura para cada gravação, o tipo de sinal gravado `signal` e como ele foi publicado.

Consulta 4: faz uma busca nas tabelas `RecordingRDF`, `SignalRDF`, `TrackRDF` e `MusicArtistRDF`. Ela retorna as mesmas informações que a consulta 3 e adicionalmente quem produziu e qual tipo de sinal.

Consulta 5: faz uma busca nas tabelas `RecordingRDF`, `SignalRDF`, `TrackRDF` e `MusicArtistRDF`. Ela retorna todas as informações da Consulta 4, mais informações sobre o Track que corresponde à publicação do sinal.

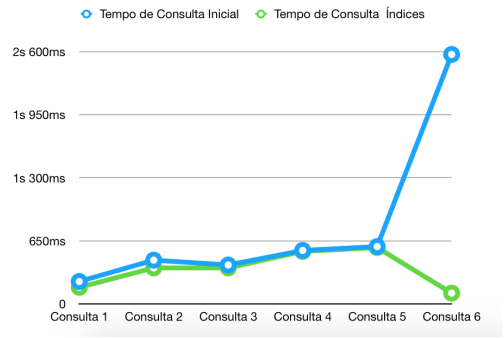
Consulta 6: faz uma busca nas tabelas `SignalRDF`, `TrackRDF` e na tabela multivalorada `chart_positionMultivalueRDF`. Ela retorna o sinal que foi publicado e seu tipo. Busca também o título, seu `label` e tipo do track e quais posições ele esteve, na tabela multivalorada.

4.2. Impacto da Criação de índices

Nesta subseção é determinado o impacto da criação de índices sobre todas as tabelas no atributo OID. As consultas foram executadas com duas configurações. Na primeira, é considerada a existência de índice B+ somente nas tabelas de metadados `TB_DatabaseSchema` e `TB_Subj_OID`. Estes índices são importantes para o processo de tradução das consultas. Na segunda configuração, são gerados índices chamados de `id_OID`, sobre o atributo OID nas tabelas estruturadas `MusicArtistRDF, PerformanceRDF, RecordingRDF, SignalRDF, TrackRDF`. As tabelas utilizam o `OID` como chave primária e podem impactar o tempo de execução da consulta SQL.

	Tempo de Consulta Inicial	Tempo de Consulta Índices	Quantidade de Tabelas Estruturadas	Percentual de ganho
Consulta 1	230ms	170ms	1	26,1%
Consulta 2	450ms	370ms	2	17,8%
Consulta 3	400ms	370ms	3	7,5%
Consulta 4	550ms	540ms	4	1,8%
Consulta 5	590ms	580ms	4	1,7%
Consulta 6	2s 570ms	110ms	2	95,7%

(a) Tempo de execução sem e com índices `id_OID`.



(b) Gráfico do tempo por consulta com a adição dos índices `id_OID`.

Figura 6. Impacto dos índices sobre a atributo OID

Nas Figuras 6a e 6b, é possível observar os tempos de consulta na configuração inicial e após a adição dos índices `id_OID`.

Foi possível concluir que houve um um ganho médio de 11% no tempo de consulta nas tabelas que não são multivaloradas com a criação dos índices. A Consulta 6 utilizou uma tabela multivalorada e obteve um ganho significativo de 96%.

As consultas 1-5 não tiveram um ganho significativo em razão de serem consultas que não filtram o resultado por valores específicos, mas apenas executam junções sobre múltiplas tabelas. Foi também observado que com a junção de mais tabelas, o ganho proporcionado pelo índice é menor. É possível verificar essa constatação na Figura 6a. Portanto, o incremento do ganho da utilização de índices foi pouco significativo. Foi constatado pelo *explain* da consulta SQL, que para as consultas 3, 4 e 5, os índices não são utilizados. É realizado um *Block Nested Loop* e não *Index-Nested Loop*, que utiliza índices.

Para uma melhor análise da melhoria da consulta 6, foi executado o `explain` da consulta na situação inicial e o `explain` da consulta com índices. Como a consulta 6 possui busca em uma tabela multivalorada, o MySQL realiza *join* entre a tabela TrackRDF e a multivalorada chart_positionMultivalueRDF. O MySQL segundo [Oracle 2017], utiliza índices para extrair linhas de outras tabelas quando realiza junção entre tabelas. Ou seja, retira a necessidade de escanear a tabela estruturada TrackRDF múltiplas vezes. Na inexistência do índice, a consulta 6 utiliza *Block Nested Loop*, o que justifica a grande diferença no tempo de execução.

4.3. Criação das visões

Com o objetivo de otimizar as buscas nas tabelas de *Overflow* Específico, foram criadas visões com a mesma estrutura da tabela estruturada a qual ela está associada. Desta forma, consultas que envolvem predicados que pertencem à tabela estruturada podem ser processadas utilizando a visão, facilitando o processo de tradução.

Um exemplo da criação da visão sobre a tabela estruturada MusicArtistRDF está ilustrado na Figura 7(a). A consulta da Figura 4(b) utilizando a visão é apresentada na Figura 7(b).

As 6 consultas foram executadas com as duas formas de tradução: utilizando di-

```
CREATE VIEW MusicArtistRDF_View AS
  SELECT t1.subj as OID,
         t1.obj as name,
         t2.obj as type
  FROM Overflow_MusicArtistRDF as t1
       FULL OUTER JOIN
       Overflow_MusicArtistRDF as t2
       ON t1.subj=t2.subj
  WHERE t1.pre='name' and t2.pred='type'
```

(a) Criação da visão

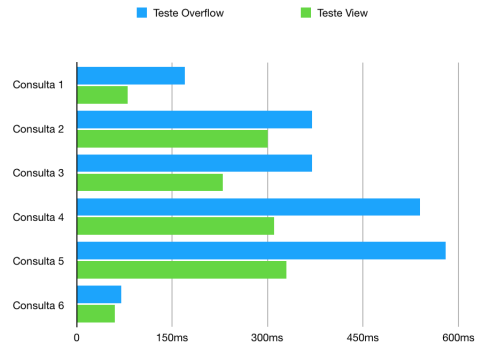
```
SELECT b.OID as b,
       a.typeYLEIFC as t,
       a.nameEY9TOM as n,
       a.OID as a
FROM (SELECT MusicArtistRDF.OID,
             MusicArtistRDF.name as nameEY9TOM,
             MusicArtistRDF.type as typeYLEIFC
      FROM MusicArtistRDF
      UNION ALL
      SELECT MuisicArtistRDF_View.OID,
             MusicArtistRDF_View.name as nameEY9TOM,
             MusicArtistRDF_View.type as typeYLEIFC
      FROM MusicArtistRDF_View) AS a,
     (SELECT PerformanceRDF.OID,
             PerformanceRDF.fk performer
             AS fk_performerEKRX08
      FROM PerformanceRDF) AS b
WHERE a.OID = fk performerEKRX08
```

(b) Consulta SQL utilizando a visão

Figura 7. Tradução SQL utilizando uma visão

	Tempo de Consulta Inicial	Teste Overflow	Teste View
Consulta 1	230ms	170ms	80ms
Consulta 2	450ms	370ms	300ms
Consulta 3	400ms	370ms	230ms
Consulta 4	550ms	540ms	310ms
Consulta 5	590ms	580ms	330ms
Consulta 6	2s 570ms	70ms	60ms

(a) Tempo por consulta da situação inicial, com índices e com visões



(b) Gráfico do tempo por consulta da situação inicial, com índices e com visões

Figura 8. Impacto das Visões

retamente o *Overflow* e utilizando as visões. É importante destacar que os índices criados nos experimentos descritos na SubSeção 4.2 continuaram implementados. Os tempos de execução das consultas utilizando os dois tipos de tradução são apresentados nas Figuras 8a e 8b. Eles mostram um ganho de desempenho considerável utilizando a estratégia visões, que varia de 7% a 54,4%.

4.4. Impacto na Utilização de filtros

Neste experimento foram consideradas consultas que introduzem filtros para desconsiderar atributos com valores nulos `IS NOT NULL`. A consulta da Figura 7(b) com a adição de filtros é apresentada na Figura 9 e os resultados obtidos estão nas Figuras 10a e 10b.

A redução do tempo de processamento com o filtro pode ser explicada pela redução das tabelas intermediárias geradas com a aplicação do filtro. Foi executado o *explain* da Consulta 2, que fez busca direta no *Overflow*. Constatou-se uma redução de 31.190 linhas para 28.069 linhas com a aplicação do filtro já no inicio da busca. Após a união na operação, observa-se que as entradas de dados filtradas reduziram na mesma proporção.

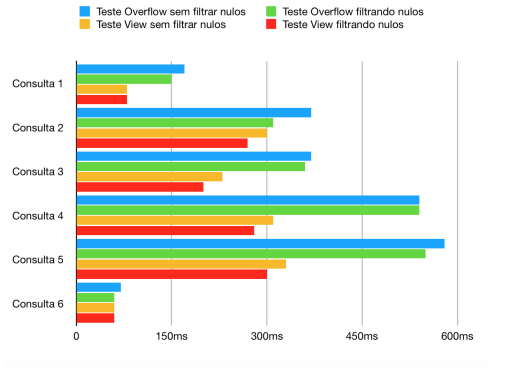
Analogamente, o *explain* da mesma consulta, utilizando visões, reduz de 43.513.880

```
SELECT b.OID as b,
       a.typeYLEIFC as t,
       a.nameEY9TOM as n,
       a.OID as a
FROM (SELECT MusicArtistRDF.OID,
             MusicArtistRDF.name as nameEY9TOM,
             MusicArtistRDF.type as typeYLEIFC
      FROM MusicArtistRDF
      WHERE MusicArtistRDF.name IS NOT NULL and
            MusicArtistRDF.type IS NOT NULL
      UNION ALL
      SELECT MuisicArtistRDF_View.OID,
             MusicArtistRDF_View.name as nameEY9TOM,
             MusicArtistRDF_View.type as typeYLEIFC
      FROM MusicArtistRDF_View
      WHERE MusicArtistRDF_View.name IS NOT NULL and
            MusicArtistRDF_View.type IS NOT NULL) AS a,
     (SELECT PerformanceRDF.OID,
             PerformanceRDF.fk_performer
             AS fk_performerEKRX08
      FROM PerformanceRDF
      WHERE PerformanceRDF.fk_performer IS NOT NULL) AS b
WHERE a.OID = fk performerEKRX08
```

Figura 9. Tradução SQL utilizando filtros

	Tempo de Consulta Inicial	Teste Overflow sem filtrar nulos	Teste Overflow filtrando nulos	Teste View sem filtrar nulos	Teste View filtrando nulos
Consulta 1	230ms	170ms	150ms	80ms	80ms
Consulta 2	450ms	370ms	310ms	300ms	270ms
Consulta 3	400ms	370ms	360ms	230ms	200ms
Consulta 4	550ms	540ms	540ms	310ms	280ms
Consulta 5	590ms	580ms	550ms	330ms	300ms
Consulta 6	2s 570ms	70ms	60ms	60ms	60ms

(a) Tempo por consulta da situação inicial e com visões adicionadas



(b) Gráfico do tempo por consulta da situação inicial e com visões adicionadas

Figura 10. Impacto da utilização de filtros e comparação geral

linhas para 39.162.489, já na primeira operação de busca. Após a união na décima primeira operação, observa-se que as entradas de dados filtradas reduziram na mesma proporção.

Os resultados dos experimentos realizados mostraram que o mapeamento utilizando visões foi bastante vantajoso com relação ao mapeamento processando consultas diretamente no *Overflow* Específico. Além disso, a redução de tabelas intermediárias com a aplicação do filtro `IS NOT NULL` apresentou uma melhoria no tempo de processamento de 7% a 58,5%.

5. Conclusão

Este artigo apresentou a pesquisa que explorou possíveis otimizações de consultas do sistema AORR, que é um modelo de armazenamento de uma base de dados RDF utilizando um SGBDR como *backend* de armazenamento. O AORR é composto de dois módulos: o primeiro realiza a geração de um esquema relacional a partir de uma base RDF e o outro,

converte de uma consulta SPARQL para SQL, compatível com a estrutura de dados criada. O objetivo deste trabalho foi explorar recursos resultantes dos módulos anteriores, visando otimizar o desempenho das consultas.

As técnicas exploradas foram a criação de índices, a utilização de visões no mapeamento de consultas e aplicação de filtros. Os experimentos mostraram que a criação de índices sobre os identificadores das tabelas estruturadas resultam em um ganho médio de 11%, em razão de serem consultas que não filtram o resultado por valores específicos, mas apenas executam junções sobre múltiplas tabelas. O ganho com a criação de visões mostrou-se bastante significativo, apresentando uma melhoria média de 54,4%, sem filtro de valores nulos e de 58,5% com o filtro. Pode-se concluir que a implementação de visões no ambiente AORR foi altamente recomendável para se obter otimização, assim como a utilização filtros de *IS NOT NULL*.

Uma otimização que possivelmente teria um grande impacto no sistema AORR é o armazenamento da tabela de metadados `TB_DatabaseSchema` em uma estrutura em memória, dado que o tempo de tradução utilizando o procedimento de [G. Pauluk and Hara 2016] é alto e utiliza bastante essa estrutura em sua tradução. Outra possível otimização poderia ser a análise de carga das seleções mais frequentes, para a geração de índices clusterizados. A pesquisa não explorou a otimização de buscas no *Overflow* Geral. A criação de índices adicionais sobre o predicado e objeto desta tabela poderia ser também explorada. Por fim, poderiam ser realizados experimentos adicionais com a execução de consultas em outros SGBDRs e com outras estruturas de indexação.

Referências

Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proceedings of the International Conference on Very Large Data Bases*.

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

de Lima Prado, R. (2018). Armazenamento otimizado de dados RDF em um SGBD relacional. Master's thesis, UFPR, Programa de Pós-graduação em Informática.

G. Pauluk, J. and Hara, C. S. (2016). Processamento de consultas SPARQL em um SGBD relacional. Trabalho de Graduação, UFPR, Bacharelado em Ciência da Computação.

Oracle (2017). Mysql 5.6 reference manual.

Pham, M.-D., Passing, L., Erling, O., and Boncz, P. (2015). Deriving an emergent relational schema from RDF data. In *Proceedings of the International World Wide Web Conferences*.

Scabora, L. C., Oliuveira, P. H., Kaster, D. S., Traina, A. J. M., and Junior, C. T. (2017). Relational graph data management on the edge: Grouping vertices' neighborhood with edge-k. In *Proceedings of the Brazilian Symposium on Databases*.