UNIVERSIDADE FEDERAL DO PARANÁ

MARIANA MACHADO GARCEZ DUARTE

OTIMIZAÇÃO DO MAPEAMENTO DE CONSULTAS SPARQL PARA SQL

MONOGRAFIA

CURITIBA

2017

MARIANA MACHADO GARCEZ DUARTE

OTIMIZAÇÃO DO MAPEAMENTO DE CONSULTAS SPARQL PARA SQL

Monografia apresentada à disciplina de Trabalho de Graduação em Banco de Dados, Setor de Ciências Exatas da Universidade Federal do Paraná como requisito para obtenção do grau de Bacharel em Ciencia da Computação.

Orientadora: Profa Dra Carmem Satie Hara

CURITIBA

2017

RESUMO

Machado Garcez Duarte, Mariana. OTIMIZAÇÃO DO MAPEAMENTO DE CONSULTAS SPARQL PARA SQL. 57 f. Monografia – , Universidade Federal do Paraná. Curitiba, 2017.

A *Web* Semântica é uma extensão da *World Wide Web*, que permite aos computadores e humanos trabalharem em cooperação. O objetivo é organizar informações de tal forma que ambos possam lê-las através de padrões de formatação de dados, como o RDF.

SPARQL é a linguagem proposta pela W3C para realizar consultas sobre uma base de dados RDF. Em uma consulta SPARQL, padrões de triplas são combinados para a geração de um resultado. Quando as consultas envolvem grandes volumes de dados, as buscas podem se tornar ineficientes e torna-se necessária a otimização destas consultas. Os trabalhos de (LIMA; HARA, 2016) e (PAULUK; HARA, 2016) buscam uma solução para este problema através do armazenamento de dados RDF em um SGBD Sistema Gerenciador de Banco de Dados relacional e a tradução de consultas SPARQL para SQL.

Esta monografia tem o objetivo de dar continuidade a esses trabalhos e visa otimizar o mapeamento de consultas SPARQL para SQL, utilizando índices e visões. O objetivo dos experimentos realizados nesta monografia foi determinar o impacto desses dois recursos no desempenho das consultas. A partir dos experimentos realizados, constatou-se que a implementação de visões é altamente recomendável para se obter otimização, com uma redução do tempo de processamento da consulta em até 54,4%, e que a utilização de filtros que desconsideram tuplas contendo valores nulos resultam em uma redução de processamento da consulta em até 58,8%.

Palavras-chave: SGBD RDF, Otimização, SPARQL em SGBD relacional, Visões, Índices

LISTA DE FIGURAS

LISTA DE SIGLAS

SGBD	Sistema Gerenciador de Banco de Dados
RDF	*Resource Description Framework*
IRI	*Internationalized Resource Identifier*
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SPO	Sujeito, Predicado e Objeto
AORR	Armazenamento Otimizado de Dados RDF em SGBD Relacional
CS	Characteristics Sets

SUMÁRIO

1 INTRODUÇÃO

A *Web* Semântica é uma extensão da *World Wide Web*, que permite aos computadores e humanos trabalharem em cooperação. O objetivo é organizar informações de tal forma que ambos possam lê-las através de padrões de formatação de dados, como o RDF. O RDF é caracterizado por triplas, onde em cada uma delas, se encontra o sujeito, predicado e objeto.

SPARQL é a linguagem proposta pela W3C para realizar consultas sobre uma base de dados RDF. Em uma consulta SPARQL, padrões de triplas são combinados para a geração de um resultado.

A grande quantidade de dados de RDF existentes requerem 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. Assim, é possivel aproveitar as otimizações que um Sistema Gerenciador de Banco de Dados Relacional oferece, como indexação, visões materializadas, particionamento horizontal, além das otimizações sobre a linguagem de consulta SQL. Buscando essa solução, o trabalho de (LIMA; HARA, 2016) propõe um armazenamento de dados RDF em um Sistema Gerenciador de Banco de Dados (SGBD) relacional, com o Sistema de Armazenamento Otimizado de Dados RDF em SGBD Relacional (AORR). O Sistema AORR tem como objetivo armazenar de forma eficiente os dados RDF. A estratégia adotada pelo AORR difere da abordagem direta, que seria armazenar as triplas RDF em uma relação com 3 atributos relacionais(sujeito, predicado, objeto- SPO). Este mapeamento direto resulta em uma relação de cardinalidade igual ao número de triplas. Atualmente, existem bases RDF com trilhões de triplas, o que pode tomar esta estratégia ineficiente. A proposta do AORR consiste em extrair um esquema dos dados RDF para a geração da base relacional. Nesse processo, é feita a comparação de atributos, gerando tabelas estruturadas, tabelas chamadas de Overflow e as tabelas de metadados. As tabelas chamadas de estruturadas agrupam em uma mesma tupla propriedades de um mesmo sujeito, que correspondem na base RDF a um conjunto de triplas. Para dar suporte à heterogeneidade do RDF e permitir atualizações com tri-

plas que não se adequam ao esquema relacional extraído, o AORR mantém um conjunto de tabelas Overflow, que correspondem à parte *não estruturada* da base relacional. As tabelas de Overflow são mantidas no formato SPO tradicional. As tabelas de metadados correspondem as informações sobre o mapeamento dos dados RDF para o esquema relacional gerado. O armazenamento destas informações é necessário em função do AORR ser proposto para utilização como *backend* de armazenamento da base RDF. Assim, as aplicações usuárias não interagem diretamente com o SGBD relacional, mas através de consultas SPARQL e atualizações na forma de triplas. Assim, é necessário mapear estas atualizações para a tabela adequada na base de dados relacional. Além disso, as tabelas de metadados fornecem informações que permitem a tradução de consultas SPARQL para SQL, sem conhecimento prévio do esquema da base relacional. O trabalho de (PAULUK; HARA, 2016), utiliza essas tabelas de metadados para realizar traduções do SPARQL para o SQL. O trabalho de (PAULUK; HARA, 2016) desenvolve uma metodologia de mapeamento SPARQL para SQL.

Esta monografia tem o objetivo de dar continuidade a esses trabalhos e visa otimizar o mapeamento de consultas SPARQL para SQL de (PAULUK; HARA, 2016), explorando a utilização de índices e visões. O objetivo dos experimentos realizados nesta monografia foi determinar o impacto desses dois recursos no desempenho das consultas. A partir dos experimentos realizados, constatou-se que a implementação de visões é altamente recomendável para se obter otimização, com uma redução do tempo de processamento da consulta em até 54,4%, e que a utilização de filtros que desconsideram tuplas contendo valores nulos resultam em uma redução de processamento da consulta em até 58,8%.

1.1 OBJETIVOS

Esse trabalho tem os objetivos apresentados a seguir.

1.1.1 OBJETIVO GERAL

Otimizar o processamento de consultas SPARQL processadas em um SGBD relacional. Considera-se que o documento RDF foi mapeado para uma base relacional, não no formato de triplas, mas agregando triplas associadas, criando tabelas de maior grau.

1.1.2 OBJETIVOS ESPECÍFICOS

- Determinar as melhores configurações de índices, que possam acelerar o processo de consulta.
- Determinar a melhor maneira realizar buscas, utilizando visões ou realizando diretamente na base de triplas das tabelas Overflow.
- Realizar testes para comprovação da melhor escolha.

1.2 SOLUÇÃO PROPOSTA

Trabalhar com índices e visões para melhorar o desempenho do processamento de SPARQL. Sejam esses índices *hash* ou árvore B+.

1.3 VALIDAÇÃO DA PROPOSTA

A validação será feita a partir de experimentos utilizando uma base de dados RDF que foi mapeada para uma base relacional.

1.4 ORGANIZAÇÃO

O restante desta monografia está organizado da seguinte forma. O Capítulo 2 apresenta as técnicas utilizadas para a otimização de consultas em um SBGD relacional. O Capítulo 3 apresenta definições e trabalhos relacionados. O Capítulo 4 apresenta os estudos experimentais realizados para determinar o impacto da utilização de índices e visões nas consultas apresentadas no Capítulo 3. O Capítulo 5 apresenta as conclusões em relação as técnicas utilizadas e resultados encontrados.

2 ÍNDICES E VISÕES

Nesse Capítulo é mostrado o embasamento para este trabalho de graduação. Nas seções 2.1 e 2.2 são apresentadas estruturas de índices e suas variações. A Seção 2.3 define visões e principais algoritmos associados.

2.1 ÍNDICES

Um índice é composto por um conjunto de entradas de dados, cada uma associada a um valor da chave de busca. Já, uma chave de busca facilita a recuperação de dados associados a um determinado valor da chave de busca, que pode ser composta por um ou mais atributos. Uma entrada de dados pode ser composta por um valor de chave de busca e um registro associado, que corresponde a uma tupla de relação indexada. O registro pode conter o endereço da tupla ou a própria tupla indexada.

Um índice é uma estrutura de dados que agiliza o acesso de tuplas em uma relação armazenada no disco. É possível criar múltiplos índices sobre uma mesma relação, cada um com uma chave de busca diferente. Assim, operações de busca que não são eficientemente auxiliadas pela organização do arquivo usado para manter as tuplas são aceleradas.

2.2 ESTRUTURAS DE DADOS DE ÍNDICE

A principais estruturas de dados utilizadas para a implementação de índices são:

1. Tabelas *hash*.
2. Árvore B+.

2.2.1 INDEXAÇÃO BASEADA EM *HASH*

Com a indexação baseada em *hash* as entradas de dados são agrupadas em *buckets*. No armazenamento físico em disco, *Buckets* consistem em uma página primária e, possivelmente, outras páginas adicionais ligadas em cadeia.

A função *hash* é aplicada na chave de busca para determinar a qual *bucket* a entrada de dados r pertence. Dado o número de um *bucket*, é possível recuperar a página primária desse *bucket* com um acesso a disco. Caso haja páginas adicionais em cadeia, novos acessos a disco são necessários.

Na inserção, a função *hash* é aplicada à chave de busca para determinar seu *bucket* correspondente. Páginas adicionais são alocadas de acordo com a necessidade. A Figura 7 representa esse tipo de indexação. Nesta Figura, são representadas duas estruturas de índice em *hash*. H1 representa uma função *hash* aplicada na chave de busca idade (*age*) e H2 representa representa uma função *hash* aplicada na chave de busca salário (*sal*). Ambas as funções *hash* direcionam para entradas de dados correspondentes, criando índices. Observe que nas entradas de dados do índice H1,os registros associados em cada entrada correspondem aos próprios registros de dados. Já para o índice H2, o registro associado corresponde ao endereço onde o registro de dados está armazenado.

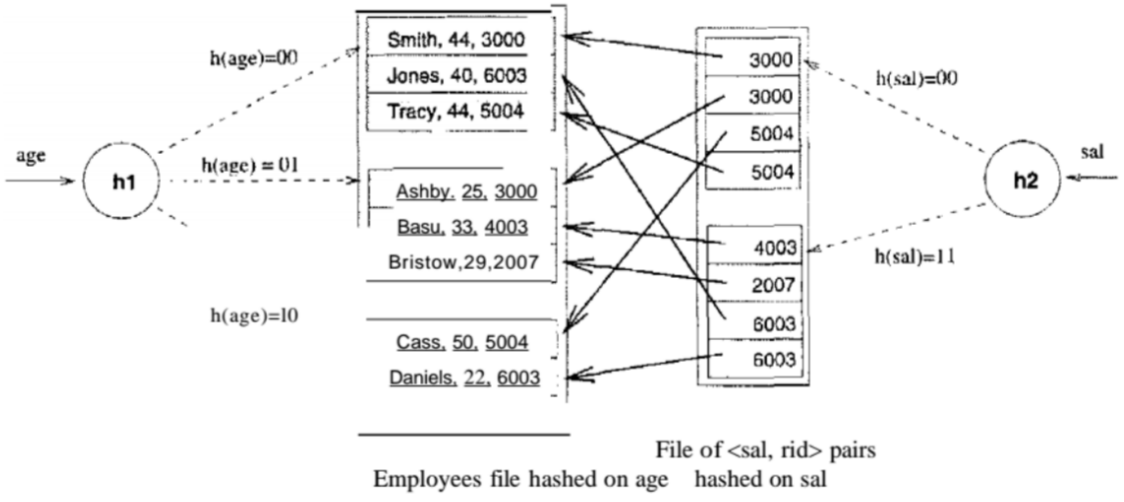

Figura 7: Exemplo de um arquivo organizado com índices baseados em *Hash* (RAMAKRISHNAN; GEHRKE, 2003)

As operações de leitura podem ser por intervalo, na qual se espera vários resultados dentro de um intervalo ordenado. Seleções de igualdade buscam entradas de dados que satisfazem apenas um valor especificado.

Em geral, um índice permite leitura de uma entrada de dados, que satisfaz uma condição de seleção, de forma eficiente. Técnicas de indexação baseadas em *hash* são otimizadas somente para seleções de igualdade, e pouco para seleções de intervalo, no qual tem geralmente um número de acesso a disco mais alto do que buscar em todo o arquivo de entrada de dados.

2.2.2 INDEXAÇÃO BASEADE EM ÁRVORE B+

Na indexação baseada em árvore B+, as entradas de dados são ordenadas pelo valor da chave de busca. Uma estrutura de dados de pesquisa hierárquica é mantida. O nível mais baixo da árvore, chamado de nível de folhas, contem as entradas de dados.

Essa estrutura permite localizar eficientemente todas as entradas de dados com valores de chave de busca em um intervalo desejado. Todas as pesquisas começam no nó mais alto, chamado de raiz. No nível físico, o conteúdo de páginas não-folha direciona as buscas para a página folha correta. Páginas não-folha contêm ponteiros para nós separados pelo valor da chave de busca. O ponteiro de nó que está à esquerda de um valor de chave k aponta para uma sub-árvore que contém somente entradas de dados menores do que k. O ponteiro de nó à direita de um valor de chave k aponta para uma sub-árvore que contém somente entrada de dados maiores ou iguais a k. Portanto, o número de acesso a disco ocorridos durante a busca é igual ao comprimento de um caminho da raiz a uma folha.

Uma árvore B+ é uma estrutura de índices que assegura que todos os caminhos da raiz a uma folha em determinada árvore possuem o mesmo comprimento, ou seja, a estrutura é balanceada. Esse tipo de indexação é ilustrado na Figura 8. Na Figura, é ilustrado um índice baseado em árvore B+. Para realizar a busca do registro de dados que contém a chave, a busca se inicia no nó raiz. Após, o ponteiro de nó verifica que a chave de busca 40 é maior que 12 e menor que 78, então, toma o caminho do meio. O processo é repetido até chegar na página folha que possui o registro de dados procurado.

As operações de leitura podem ser por intervalo, na qual se espera vários resultados dentro de um intervalo ordenado. Seleções de igualdade buscam entradas de dados que satisfazem apenas um valor especificado.

Técnicas de indexação baseadas árvores B+ permitem que seleções de igualdade e de intervalo sejam eficientes, o que explica a sua popularidade.

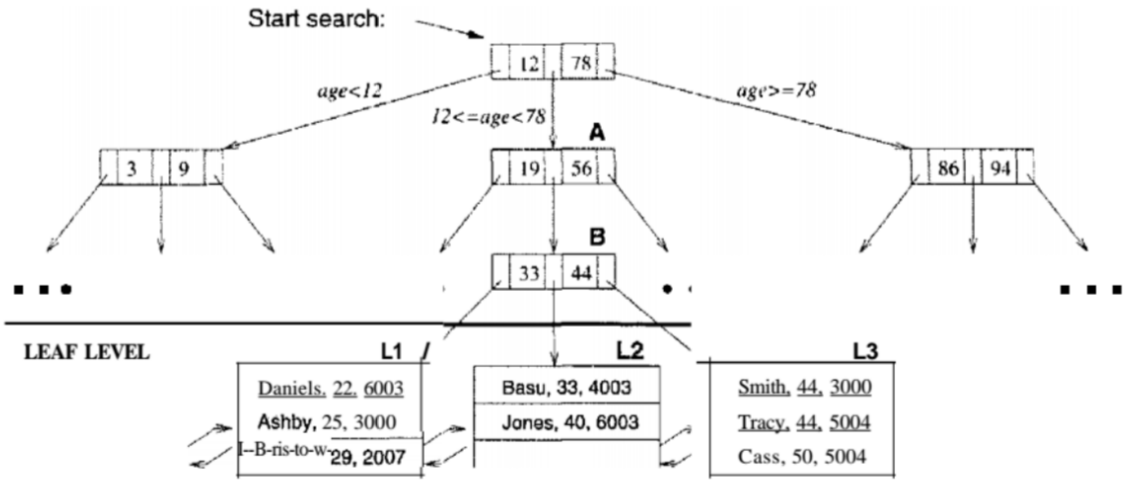

Figura 8: exemplo de um índice basedo em Árvore B+(RAMAKRISHNAN; GEHRKE, 2003)

2.3 VISÕES

Uma visão, conceitualmente, é um conjunto resultado de uma consulta realizada sobre uma ou mais tabelas. O comando de seleção que é usado ao criar uma visão provê a sua definição. Visões não fazem parte do esquema físico do SGBD. Elas são tabelas virtuais computadas e não ocupam espaço em disco. As visões sofrem mudança nos seus dados apenas quando as tabelas utilizadas para a sua definição sofrem mudança.

As visões são utilizadas, tradicionalmente, para proporcionar o aumento da segurança. Com esta recurso, é possível limitar o acesso do usuário a uma tabela utilizando a visão, selecionando apenas o que se deseja dar acesso. Desta maneira, não é preciso dar acesso a tabela real ao usuário. Apesar de visões serem mais utilizadas para segurança, para esta monografia, elas serão usadas para facilitar as buscas em tabelas SPO que armazenam a porção não estruturada.

Visões podem ser virtuais ou materializadas. As Visões virtuais são objetos que apesar de não armazenarem dados, podem ser referenciados como tabelas virtuais, pois é possível realizar buscas sobre eles como em qualquer tabela. Uma visão contém tuplas e atributos, como uma tabela real. Os campos em uma visão são campos de uma ou mais tabelas reais da Base de Dados. Já, uma visão materializada é o resultado de uma busca armazenado em alguma estrutura, geralmente em uma tabela. Essas visões materializadas são usadas quando uma resposta imediata é necessária e a base de dados demoraria muito tempo para produzir resultados sem o uso delas.

Apesar do MySQL tratar visões como tabelas para muitos propósitos, ele não as trata da mesma maneira. Existem algoritmos de implementação que otimizam as visões, tornando-as um recurso para a otimização. O MySql não dá suporte a visões materializadas, segundo (ORACLE, 2017).

2.3.1 ALGORITMOS DE IMPLEMENTAÇÃO DE VISÕES

A maneira mais intuitiva de um servidor implementar uma visão, segundo (SCHWARTZ et al., 2012), é executar o comando *SELECT* e armazenar o resultado em uma tabela temporária. Então, é possível utilizar a tabela temporária como qualquer outra tabela base utilizada na consulta. Existem problemas de otimização com esta abordagem. Uma melhor maneira de implementar a visão é reescrever a consulta que faz referência à visão, utilizando a definição da visão.

Esses dois principais métodos para implementar uma visão podem ser utilizados pelo MySQL, como ilustrado na figura 9. Ele tenta utilizar a estratégia de reescrita de consultas sempre que possível.

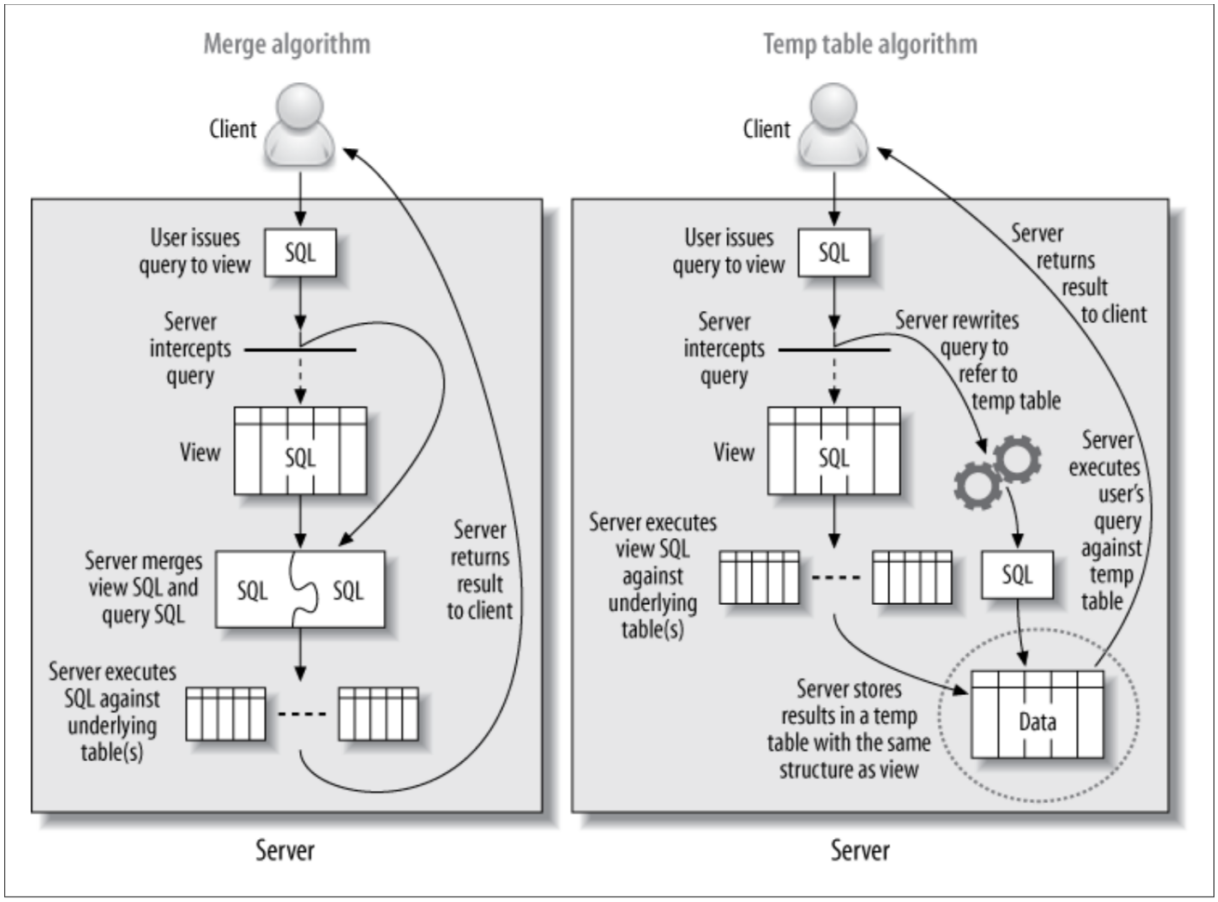

Figura 9: Duas implementações de visões, como visto em (SCHWARTZ et al., 2012)

No entanto, a estratégia de criar uma tabela temporária é adotada quando a consulta contém GROUP BY, DISTINCT, funções agregadas, UNION, subconsultas ou qualquer outra construção que não preserva uma relação um-para-um entre as linhas nas tabelas de base subjacentes e as linhas retornadas de uma visão.

2.4 SUMÁRIO

Neste capítulo foram apresentadas diferentes técnicas para indexação e visões. Essas técnicas serão utilizadas na fase experimental, que é o assunto do Capítulo 4. No Capítulo 3, os trabalhos correlatos serão apresentados.

3 PRELIMINARES E TRABALHOS CORRELATOS

Esse Capítulo apresenta o modelo RDF e tecnologias relacionadas. Os trabalhos sobre os quais esta monografia foi baseada também são detalhados.

Na Seção 3.2, a estratégia AORR, que armazena uma base RDF em um SGBD relacional é apresentado. A Seção 3.3 apresenta o esquema de tradução SPARQL para SQL associado ao AORR. A Seção 2.3 apresenta como foram criadas as visões para esta monografia.

3.1 RDF E SPARQL

O *Resource Description Framework* (RDF) é um modelo padrão para dados interligados na Web. Esse padrão é uma recomendação da W3C para uso na Web Semântica. Uma base RDF é composta por triplas, cada uma delas, contendo um sujeito, um predicado e um objeto. O uso desse modelo simples possibilita que dados estruturados e semi-estruturados sejam misturados, expostos e compartilhados entre diferentes aplicações. Os componentes RDF podem ser de três tipos: IRIs, literais e *blank nodes*. A Figura 11 ilustra esses tês tipos.

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

Figura 11: Exemplo de tipos de dados RDF, como visto em (PAULUK; HARA, 2016)

O *Internationalized Resource Identifier* (IRI) é uma sequência de caracteres que são combinados de maneira a identificar um recurso de maneira única na *Web*.

Os literais representam valores, como datas, números ou textos.

608678 name Andy Halstead
13094 name Graham Coxon
608678 type Music Artist
13094 type Music Artist
16 type Performance
16 fk_performer 15

1- Armazenamento em triplas

Sujeito	Predicado	Objeto
608678	name	Andy Halstead
13094	name	Graham Coxon
608678	type	Music Artist
13094	type	Music Artist
16	Type	Performance
16	fk_performer	15

2- Representação em uma tabela de três colunas

Name	Type	fk_performer	ID
Andy Halstead	Music Artist		608678
Graham Coxon	Music Artist		13094
	Performance	15	16

3- Representação das triplas em um esquema estruturado

Figura 12: Formas de armazenamento RDF.

Os *blank nodes* são usados para representar relações entre recursos, porém não denotam, necessariamente, um recurso. Sujeito de uma tripla pode ser IRIs ou *Black Nodes*. Predicados são somente IRIs. Por fim, objetos podem ser definidos por qualquer um dos tipos definidos na Figura 11.

A Figura 12 ilustra diferentes formas de armazenamento de dados RDF. Os dados podem ser armazenados em triplas em um arquivo, como a Figura 11-1. A representação uma tabela SPO está apresentada na Figura 11-2. Já a representação em uma tabela estrutura, que agrupa informações de um conjunto de triplas com o mesmo sujeito é apresentada na Figura 11-3.

O RDF possui uma representação na forma de grafos. O sujeito e objeto caracterizam vértices. Já o predicado, caracteriza uma aresta direcional entre esses vértices e o relacionamento entre eles. Essa estrutura ligada forma um grafo direcionado e rotulado. A Figura 13 apresenta a mesma base de dados da Figura 12 representada em forma de grafos. A visão do grafo torna o modelo intuitivo e mais fácil de compreender visualmente.

A linguagem SPARQL (W3C, 2004) é o padrão definido pela W3C para manipulações e consultas de dados armazenados em RDF na Web Semântica. Ela pode ser usada para expressar consultas utilizando diversas fontes de dados. Um exemplo de consulta SPARQL pode ser visto na Figura 14. A consulta seleciona duas variáveis chamadas de *?performer* e *?instrument*, do memso sujeito ligados pelo predicado com IRIs com final *performer* e *instrument*.

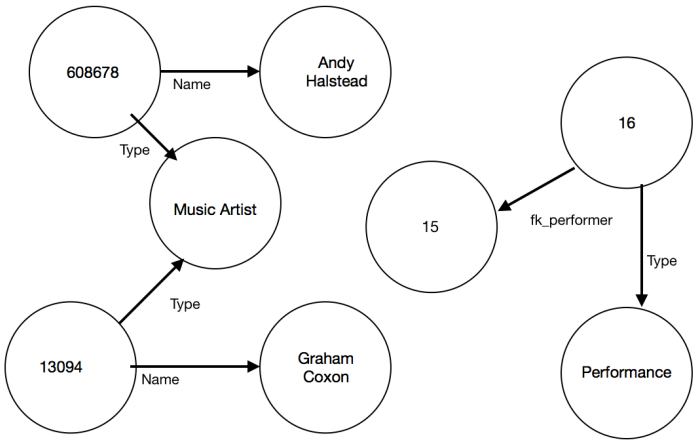

Figura 13: Exemplo de grafo RDF

O SPARQL contém recursos para consultar os padrões de grafo necessários e opcionais, juntamente com suas conjunções e disjunções. O SPARQL também dá suporte a filtragens de consultas. O resultado de uma consulta SPARQL é composto por um conjunto de IRIs e literais.

```
SELECT ?performer, ?instrument
WHERE {
  ?subj
    <http://purl.org/ontology/mo/instrument>
      ?instrument
  ?subj
    <http://purl.org/ontology/mo/performer>
      ?performer
}
```

Figura 14: Exemplo de consulta SPARQL.

3.2 ARMAZENAMENTO DE RDF EM UM SGBD RELACIONAL

A grande quantidade de dados de RDF existentes 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. Assim, é possível aproveitar as otimizações que um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) oferece, como indexação, visões materializadas, particionamento horizontal, além das otimizações sobre a linguagem de consulta SQL (LIMA; HARA, 2016).

Existem várias maneiras de transformar uma base RDF em um esquema relacional. A mais intuitiva é a transformação de triplas RDF em tuplas com 3 atributos. É feita a transformação do sujeito `s`, predicado `p` e objeto `o`, em uma tabela SPO com as colunas `s`, `p` e `o`. Porém, essa transformação torna a busca no Banco de Dados muito custosa, com inúmeras tuplas como descrito por (PHAM et al., 2015). No processamento de consultas, para cada par de padrões de triplas na consulta SPARQL, uma auto-junção é feita na tabela SPO.

Para explorar a flexibilidade do RDF com a otimização de um SGBDR, foi criado o Sistema de Armazenamento Otimizado de Dados RDF em SGBDR (AORR), que utiliza um SGBD relacional como *backend* de armazenamento RDF e processamento de consulta SPARQL. Esse sistema visa diminuir o número de auto-junções feitas em consultas para assim ter um melhor desempenho no processamento de consultas . A Figura 15 retrata os componentes de um sistema AORR.

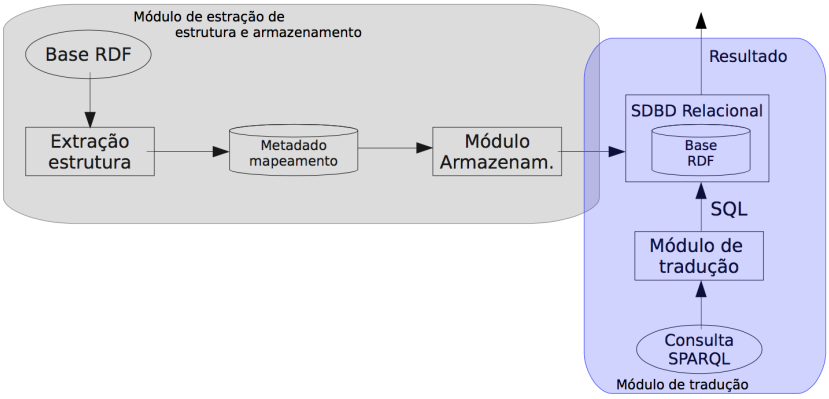

Figura 15: Componentes do sistema AORR

O módulo de extração de estrutura e armazenamento gera um esquema relacional para armazenar uma base de dados em RDF. Já o módulo de tradução de consultas é responsável por receber uma consulta em SPARQL e mapeá-la para SQL a fim de ser processada no sistema AORR.

A proposta de (LIMA; HARA, 2016) é criar um módulo de extração de estrutura e armazenamento. Uma base relacional é criada a partir de uma base RDF. Como resultado deste processo, 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, que correspondem à parte *não estruturada* da base relacional. Os dados inseridos no esquema gerado formam a parte *estruturada* da base.

O processo de geração de esquema do AORR foi inspirado na proposta de (PHAM et al., 2015) e é baseado no conceito de *characteristics sets* (CS). Um CS é definido como um conjunto de predicados. Um sujeito na base RDF *pertence* a um determinado *characteristics set* cs_1 se ele possui o conjunto de predicados cs_1. No exemplo da Figura 12 existem 2 CSs: cs_1= {name, type}, e cs_2= {type, fk_performer}, sendo que os sujeitos 608678, 13094 e 15 pertencem a cs_1 e o sujeito 16 pertence a cs_2. Um exemplo de resultado da base relacional resultante do processo AORR, está ilustrado na Figura 16. 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 tabelas geral chamada de `Overflow`. Ela comporta os sujeitos que não se adequam a nenhuma 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 17 ilustra um TB_DatabaseSchema.

Na tabela TB_Subj_OID encontram-se informações sobre a relação da URI 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 18 ilustra um exemplo desta tabela.

3.3 TRADUÇÃO SPARQL - SQL

O módulo de tradução é responsável por transformar uma consulta SPARQL em uma consulta SQL que será processada sobre o SDBD relacional no sistema AORR.

Uma consulta SPARQL pode ser definida como um par (R, PT), onde R é o

	MusicArtistRDF	
OID	Name	Type
60678	AndyHalstead	<http://xmlns.com/foaf/0.1/Person>
13094	GrahamCoxon	<http://purl.org/ontology/mo/MusicArtist>
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
16	15	NULL	<http://purl.org/ontology/mo/Performance>
19082	19081	NULL	<http://purl.org/ontology/mo/Performance>
25789	25790	NULL	<http://purl.org/ontology/mo/Performance>

	Overflow_PerformanceRDF	
Subj	Pred	Obj
25789	fk_recorded_as	25791

	Overflow	
Subj	Pred	Obj
76229	Type	<http://purl.org/ontology/mo/Transmission>
76229	Time	1988-07-05
76229	Label	Transmissionofsession999onthe1988-07-05

Figura 16: Exemplo de geração AORR

conjunto das variáveis a serem projetadas e PT é um conjunto de padrões de triplas RDF (s, p, o), onde o sujeito s é uma variável ou um IRI, o predicado p um literal e o objeto o pode ser um literal, IRI ou variável.

A proposta de tradução apresentada em (PAULUK; HARA, 2016) possui três etapas:

1. Procura por padrões estrela: São separados todos os sujeitos distintos de PT em grupos. Depois, é realizada a busca no `TB_DatabaseSchema` para obter informações sobre a localização de cada predicado.

cs_identifier	PropertyName	valueType	TableName	TableAttribute
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_PerformanceRDF	pred
...	...	...	...	...
...	...	...	...	...
CSover	Label	Literal	Overflow	pred
CSover	Time	Literal	Overflow	pred
CSover	Type	Literal	Overflow	pred

Figura 17: Exemplo de um `TB_DatabaseSchema`

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 18: Exemplo de um TB_Subj_OID, como visto em (LIMA; HARA, 2016)

2. Filtragem por ligações: É então realizada a filtragem e a montagem do relacionamento entre as variáveis.

A Figura 19 explicita as ligações existentes entre as variáveis. Ligações sujeito com sujeito estão em vermelho. Em azul, ligações objeto - sujeto. A verificação da existência de conexões entre as variáveis possibilita a filtragem de subconsultas desnecessárias no resultado final.

3. Montagem do comando SQL: Com a existência de ligações verificada, a contrução do comando SQL é feita ao organizar projeções, selecionar e relacionar as tabelas obtidas anteriormente.

Existem duas alternativas de tradução de consulta:

1. Utilizando-se as Tabelas de Overflow Específico.

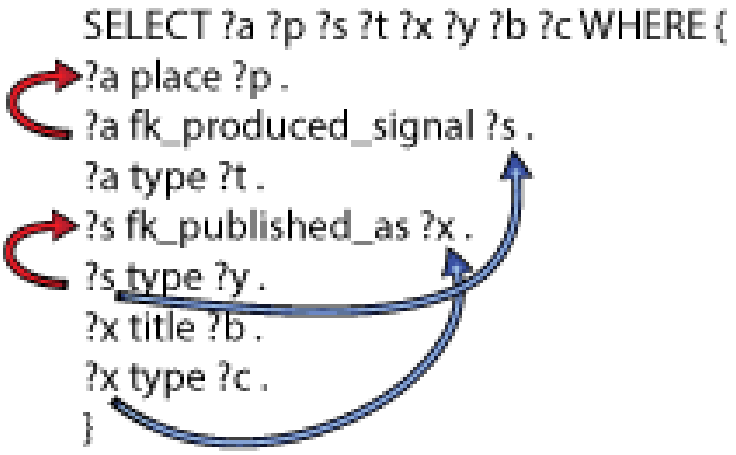

Figura 19: Ligações existentes entre as variáveis

2. Utilizando-se visões.

Um exemplo de tradução da consulta na Figura 20, utilizando Overflow Específico está na figura 21 e utilizando visões na figura 22.

```
SELECT ?a ?s ?n WHERE {
?a name  ?n.
?a type ?s .
}
```

Figura 20: Consulta SPARQL original

Exemplo de consulta traduzida do SPARQL para o SQL utilizando visões.

É importante notar que os experimentos apresentados em (PAULUK; HARA, 2016), não consideram consultas em Overflow Específico e nem em visões. A investigação do efeito destas características no desempenho do processamento de consultas no AORR é o foco desta monografia.

3.4 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. Por exemplo, considere a base ilustrada na Figura 16. O músico com OID 14002 possui 2 nome: `Cat Stevens` e `Yusuf`. Uma visão pode ser criada combinando o nome que

```
SELECT a.nameQD0R7K AS n, a.typePKGAT2 AS s,
        a.OID AS a
FROM
    (SELECT MusicArtistRDF.OID,MusicArtistRDF.name AS nameQD0R7K,
            MusicArtistRDF.type AS typePKGAT2
    FROM MusicArtistRDF
    WHERE MusicArtistRDF.name IS NOT NULL
            AND MusicArtistRDF.type IS NOT NULL
UNION ALL
    SELECT  t1.subj as OID, t1.obj as n,
            t2.obj as s
    from Overflow_MusicArtistRDF as t1
    left outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL
    UNION
    SELECT  t1.subj as OID, t1.obj as n,
            t2.obj as s
    from Overflow_MusicArtistRDF as t1
    right outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL)a;
```

Figura 21: Exemplo de consulta traduzida do SPARQL para o SQL utilizando Overflow Específico.

```
SELECT a.nameQD0R7K AS n,a.typePKGAT2 AS s,
        a.OID AS a
FROM
    (SELECT MusicArtistRDF.OID,MusicArtistRDF.name AS nameQD0R7K,
            MusicArtistRDF.type AS typePKGAT2
    FROM MusicArtistRDF
    WHERE MusicArtistRDF.name IS NOT NULL
    AND MusicArtistRDF.type IS NOT NULL
    UNION ALL
    SELECT  MusicArtistRDF_View.OID,
            MusicArtistRDF_View.name AS nameQD0R7K,
            MusicArtistRDF_View.type AS typePKGAT2
    FROM MusicArtistRDF_View
    WHERE MusicArtistRDF_View.name IS NOT NULL
            AND MusicArtistRDF_View.type IS NOT NULL) a;
```

Figura 22: Exemplo de consulta traduzida do SPARQL para o SQL utilizando visões.

está na tabela `Overflow_MusicArtistRDF(Yusuf)` com os demais predicados associados ao mesmo artista que encontram-se na tabela estruturada. Criando a tupla `(14002, Yusuf, <http://purl.org/ontology/mo/MusicArtist>)` . Um exemplo de visão está

ilustrada na Figura 23.

OID	Name	Type
14002	Yusuf	<http://purl.org/ontology/mo/MusicArtist>

Figura 23: Exemplo de visão sobre a tabela estruturada MusicArtistRDF

O comando usado para criar a visão sobre a tabela estruturada MusicArtistRDF está Registro na Figura 24.

```
CREATE OR REPLACE VIEW musicartistrdf_view AS
 SELECT t1.subj as OID,
    t1.obj as name,
    t2.obj as type
 from Overflow_MusicArtistRDF as t1
 left outer join
 Overflow_MusicArtistRDF as t2
  on t1.subj=t2.subj
 where t1.pred='name' and t2.pred='type'
 UNION
 SELECT t1.subj as OID,
    t1.obj as name,
    t2.obj as type
 from Overflow_MusicArtistRDF as t1
 right outer join
 Overflow_MusicArtistRDF as t2
  on t1.subj=t2.subj
 where t1.pred='name' and t2.pred='type';
```

Figura 24: Comando de Criação da visão musicartistrdf_view

3.5 SUMÁRIO

Este capítulo tratou de trabalhos relacionados e conceitos utilizados nesta monografia. Também mostrou como foram criadas as visões. No Capítulo 4, serão descritos os experimentos realizados aplicando as técnicas de otimização apresentadas no Capítulo 2 e os conceitos introduzidos neste Capítulo.

4 ANÁLISE EXPERIMENTAL

Com o objetivo de otimizar o mapeamento de consultas SPARQL para SQL, resultante dos trabalhos de (LIMA; HARA, 2016) e (PAULUK; HARA, 2016), um plano de implementação foi traçado, o qual possui duas etapas e testes.

Como primeira etapa, são criados índices nas tabelas estruturadas, baseados nas técnicas mencionadas no Capítulo 2, e seus resultados são comparados com a situação original.

Como etapa seguinte, ao invés de efetuar consultas diretamente nas tabelas de Overflow Específico, são efetuadas consultas sobre as visões.

Este capítulo apresenta os estudos experimentais realizados para determinar o impacto da utilização de índices e visões na consulta SQL apresentada no Capítulo 3.

A Seção 4.1 descreve a base de dados e equipamentos utilizados. A Seção 4.2 descreve os experimentos após a criação de índices. A Seção 4.3 descreve experimentos após a criação de visões.

4.1 AMBIENTE DOS EXPERIMENTOS

O computador utilizado foi um MacOSX Intel Core m3 1.1 GHz e com 8 GB de memória RAM. O sistema utilizado é o MySQL com o mecanismo de armazenamento InnoDB. Suas especificações estão na Figura 26.

O banco de dados chamado de TG_TESTE1, possui 26 tabelas que foram geradas a partir do processo de (LIMA; HARA, 2016), descrito no Capítulo 3. Na Figura 27 é possível observar todas as tabelas do TG_TESTE1. O tamanho de cada tabela está ilustrado na Figura 28.

As tabelas estruturadas utilizadas nos experimentos: MusicArtistRDF, PerformanceRDF, RecordingRDF, SignalRDF, TrackRDF. Elas foram geradas utilizando o

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

Figura 26: Detalhamento da Configuração do MySQL

Tables_in_tg_teste1
MusicArtistRDF
Overflow
Overflow_MusicArtistRDF
Overflow_PerformanceRDF
Overflow_RecordingRDF
Overflow_SignalRDF
Overflow_TrackRDF
PerformanceRDF
RDF_Tripla
RecordingRDF
SignalRDF
TB_DatabaseSchema
TB_FullPredicate
TB_Subj_OID
TrackRDF
chart_positionMultivalueRDF
createdMultivalueRDF
fk_engineerMultivalueRDF
fk_engineeredMultivalueRDF
fk_performedMultivalueRDF
fk_producedMultivalueRDF
fk_sameAsMultivalueRDF
fk_sub_eventMultivalueRDF
instrumentMultivalueRDF
isrcMultivalueRDF
labelMultivalueRDF

Figura 27: Tabelas geradas pelo sistema AORR

AORR a partir da base RDF disponível em http://dbtune.org/bbc/peel/ .

As tabelas multivaloradas são: chart_positionMultivalueRDF, createdMultivalueRDF, fk_engineerMultivalueRDF, fk_engineeredMultivalueRDF, fk_performedMultivalueRDF, fk_producedMultivalueRDF, fk_sameAsMultivalueRDF, fk_sub_eventMultivalueRDF, instrumentMultivalueRDF, isrcMultivalueRDF, labelMultivalueRDF.

As consultas consideradas para ambos os experimentos são descritas na sequência:

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 nome e tipo do artista. Além do local onde o artista realizou uma performance.

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

Figura 28: Configuração das entradas de dados por tabela

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 procura como na consulta 3, mais quem produziu e qual tipo de sinal.

Consulta 5: faz uma busca nas tabelas `RecordingRDF`, `SignalRDF`, `TrackRDF` e `MusicArtistRDF`.Ela procura como na consulta 4, mais sobre o Track que corresponde a 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.

Essas consultas expressas em SPARQL encontram-se no apêndice A.

4.2 IMPACTO DA CRIAÇÃO DE ÍNDICES

Os índices iniciais se encontram somente nas tabelas TB_DatabaseSchema e TB_Subj_OID e possuem índices baseados em Árvore B+. Suas configurações estão ilustrados nas Figuras 29 e 30.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
tb_databaseschema	0	PRIMARY	1	id	A	44	NULL	NULL		BTREE		

Figura 29: Índice na tabela TB_DatabaseSchema no estado inicial

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
tb_subj_oid	1	subj_index	1	subj	A	76455	NULL	NULL	YES	BTREE		
tb_subj_oid	1	OID_index	1	OID	A	76455	NULL	NULL	YES	BTREE		

Figura 30: Índices na tabela TB_Subj_OID no estado inicial

Índices chamados de id_OID, sobre o atributo OID, foram criados nas tabelas estruturadas MusicArtistRDF, PerformanceRDF, RecordingRDF, SignalRDF, TrackRDF. As tabelas utilizam o OID como chave primária.

O objetivo dos testes realizados é determinar o impacto da utilização de índices sobre OIDs nas tabelas estruturadas. Foram realizados testes com as 6 consultas, que são descritas na Seção 4.1. A primeira execução utiliza a base de dados descrita na Sessão 4.1 e as consultas traduzidas pelo trabalho de (PAULUK; HARA, 2016) com os índices originalmente criados. A segunda execução utiliza adicionalmente os índices sobre OIDs nas tabelas estruturadas. As consultas traduzidas utilizadas nesta etapa estão no Anexo C. Nas Figuras 31 e 32, é possível observar os tempos de consulta na configuração inicial e após a adição de índices.

	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%

Figura 31: Tempo por consulta da situação inicial e com índices adicionados teste 1

É possível concluir que houve um um ganho médio de 11% no tempo de consulta nas tabelas não multivaloradas e com índices. A Consulta 6 utiliza 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 31. Portanto, o incremento do ganho da utilização de índices foi pouco significativo.

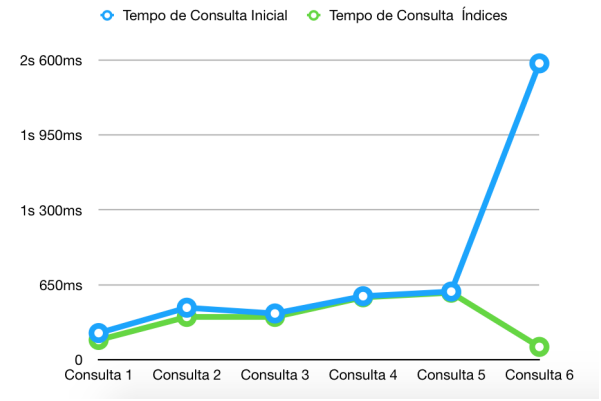

Figura 32: Gráfico do tempo por consulta da situação inicial e com índices adicionados teste 1

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 multiplas vezes. A consulta 6 utiliza *Block Nested Loop* quando não possui índices.

4.3 IMPACTO NA CRIAÇÃO DE VISÃO

Como visto na Seção 3.3, existem duas maneiras de se traduzir uma busca SPARQL em SQL: utilizando o Overflow Específico diretamente ou através de visões. A criação das visões foi descrita na Subseção 3.4. Nessa seção o impacto de criação de visões será analisados.

Foram realizados testes com as 6 consultas, que são descritas na Seção 4.1. As consultas que utilizam diretamente o Overflow estão no Anexo C e as consultas que utilizam as visões estão no Anexo B.

É importante destacar que os índices criados nos experimentos descritos na Seção 4.2 continuam implementados.

Os tempos de execução das consultas utilizando os dois tipos de tradução são

	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

Figura 33: Tempo por consulta da situação inicial e com visões adicionadas teste 2

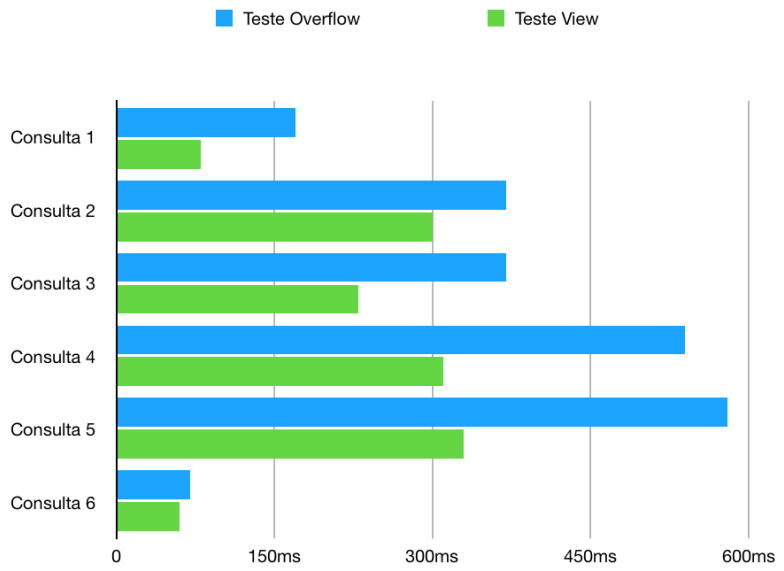

Figura 34: gráfico do tempo por consulta da situação inicial e com visões adicionadas teste 2

apresentados nas Figuras 33 e 34. Eles mostram um ganho de desempenho considerável utilizando a estratégia visões, que varia de 7% a 54,4%.

Além dos dois tipos de tradução, foram considerados também consultas que introduzem filtros para desconsiderar atributos com valores nulos `IS NOT NULL`. Exemplos de consultas sem utilizar este filtro e utilizando estão apresentados nas Figuras 35 e 36. Os resultados do experimento são apresentados nas Figuras 37 e 38.

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 faz busca direta no Overflow. Constatou-se uma redução de 31190 linhas para 28069 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.

```
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 MusicArtistRDF_View.OID,
                 MusicArtistRDF_View.name AS nameEY9TOM,
                MusicArtistRDF_View.type AS typeYLEIFC
        FROM MusicArtistRDF_View) a ,
         (SELECT PerformanceRDF.OID,
                PerformanceRDF.fk_performer AS fk_performerEKRX08
         FROM PerformanceRDF
         UNION ALL
         SELECT PerformanceRDF_View.OID,
                PerformanceRDF_View.fk_performer AS fk_performerEKRX08
         FROM PerformanceRDF_View)b
WHERE a.OID = fk_performerEKRX08;
```

Figura 35: consulta 2 sem filtro *IS NOT NULL*

```
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 MusicArtistRDF_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) a ,
    (SELECT PerformanceRDF.OID,
            PerformanceRDF.fk_performer AS fk_performerEKRX08
    FROM PerformanceRDF
    WHERE PerformanceRDF.fk_performer IS NOT NULL
    UNION ALL
    SELECT PerformanceRDF_View.OID,
           PerformanceRDF_View.fk_performer AS fk_performerEKRX08
    FROM PerformanceRDF_View
    WHERE PerformanceRDF_View.fk_performer IS NOT NULL) b
 WHERE a.OID = fk_performerEKRX08;
```

Figura 36: consulta 2 com filtro *IS NOT NULL*

Analogamente, o *explain* da mesma consulta, utilizando visões, reduz de 43513880 linhas para 39162489 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.

	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

Figura 37: Tempo por consulta da situação inicial e com visões adicionadas

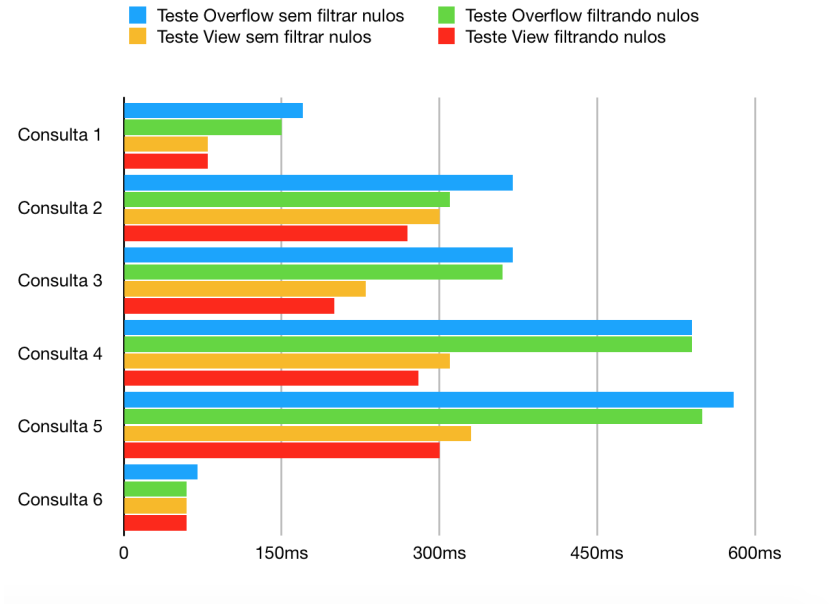

Figura 38: Gráfico do tempo por consulta da situação inicial e com visões adicionadas

Os resultados dos experimentos realizados mostram que o mapeamento utilizando visões é 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` apresentaram uma melhoria no tempo de processamento de 7% a 58,5%.

5 CONCLUSÃO

Este trabalho explorou possíveis otimizações de consultas do sistema AORR, que é um modelo de armazenamento de uma base de dados RDF utilizando um SGBD relacional como backend. 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 era explorar recursos no ambiente resultado dos módulos anteriores, visando otimizar o desempenho das consultas.

As técnicas exploradas foram a criação de índices e utilização de visões no mapeamento de consultas. Os experimentos mostraram que a criação de índices sobre os identificadores das tabelas estruturadas resultam em um ganho médio de 11%. O ganho com a criação de visões mostrou-se bastante significativo, apresentando uma melhoria média de 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 é altamente recomendável para se obter otimização e que a utilização filtros *IS NOT NULL* também.

5.1 LIMITAÇÕES DA SOLUÇÃO

Como consequência da utilização do mecanismo de armazenamento InnoDB, não é possível utilizar índices em *hash* no SGBD MySQL.

Além disso, o MySQL não suporta visões materializadas e, também, visões indexadas, segundo (SCHWARTZ et al., 2012).

5.2 TRABALHOS FUTUROS

Uma otimização que possivelmente teria um grande impacto no sistema AORR é o armazenamento da tabela TB_DatabaseSchema em uma estrutura em memória, dado que o tempo de tradução utilizando o procedimento de (PAULUK; HARA, 2016) é alto

e utiliza muito essa estrutura em sua tradução. Outra possível otimização poderia ser a análise de carga das seleções mais frequentes, para a geração de índices clusterizados em nós de maior carga. Esta monografia 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 SGBD relacionais.

REFERÊNCIAS

LIMA, R.; HARA, C. **Armazenamento otimizado de dados rdf em um sgbd relacional**. 2016.

ORACLE. **MySQL 5.6 Reference Manual**. 2017. Disponível em: <https://dev.mysql.com/doc/refman/5.6/en/create-index.html>. Acesso em: 18 de Outubro de 2017.

PAULUK, J. G.; HARA, C. **Processamento de consultas SPARQL em um SGBD relacional**. 2016.

PHAM, M.-D. et al. **Deriving an emergent relational schema from rdf data**. 2015.

RAMAKRISHNAN, R.; GEHRKE, J. **Database management systems**. 2003.

SCHWARTZ, B.; ZAITSEV, P.; TKACHENKO, V. **High Performance MySQL 3rd Edition**. 2012.

W3C, R. D. A. W. G. **SPARQL Query Language for RDF**. 2004. Disponível em: <https://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 27 de Abril de 2017.

ANEXO A – CONSULTAS EM SPARQL

Consulta 1

```
SELECT ?a ?s ?n WHERE {
?a name  ?n.
?a type ?s .
}
```

Consulta 2

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

Consulta 3

```
SELECT ?a ?p ?s ?t ?x ?y ?b ?c WHERE{
?a place ?p .
?a fk_produced_signal ?s .
?a type ?t .
?s fk_published_as ?x .
?s type ?y .
?x title ?b .
?x type ?c .
}
```

Consulta 4

```
SELECT ?a ?p ?s ?t ?x ?y ?b ?c ?u ?n ?j WHERE {
?a place ?p .
?a fk_produced_signal ?s .
?a fk_producer ?u.
?a type ?t .
?s fk_published_as ?x .
?s type ?y .
?x title ?b .
?x type ?c .
?u name ?n.
?u type ?j.
}
```

Consulta 5

```
SELECT ?a ?p ?s ?t ?x ?y ?b ?c ?u ?n ?j ?l ?m WHERE {
?a place ?p .
?a fk_produced_signal ?s .
?a fk_producer ?u.
?a type ?t .
?a label ?m.
?s fk_published_as ?x .
?s type ?y .
?x title ?b .
?x label ?l.
?x type ?c .
?u name ?n.
?u type ?j.
}
```

Consulta 6

```
SELECT ?s ?x ?y ?b ?l ?c ?z WHERE {
?s fk_published_as ?x.
?s type ?y.
?x title ?b.
?x label ?l.
```

```
?x type ?c.
?x chart_position ?z.
}
```

ANEXO B – CONSULTAS TRADUZIDAS PARA SQL COM VISÃO

Consulta 1

```
SELECT a.nameQD0R7K AS n,a.typePKGAT2 AS s,
        a.OID AS a
FROM
    (SELECT MusicArtistRDF.OID,MusicArtistRDF.name AS nameQD0R7K,
            MusicArtistRDF.type AS typePKGAT2
    FROM MusicArtistRDF
    WHERE MusicArtistRDF.name IS NOT NULL
    AND MusicArtistRDF.type IS NOT NULL
    UNION ALL
    SELECT  MusicArtistRDF_View.OID,
            MusicArtistRDF_View.name AS nameQD0R7K,
            MusicArtistRDF_View.type AS typePKGAT2
    FROM MusicArtistRDF_View
    WHERE MusicArtistRDF_View.name IS NOT NULL
            AND MusicArtistRDF_View.type IS NOT NULL) a;
```

Consulta 2

```
SELECT b.OID AS b,a.typeYLEIFC AS t,
        a.nameEY9TOM AS n, a.OID AS a
          (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  MusicArtistRDF_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) a ,
    (SELECT PerformanceRDF.OID,
            PerformanceRDF.fk_performer AS fk_performerEKRX08
    FROM PerformanceRDF
    WHERE PerformanceRDF.fk_performer IS NOT NULL
    UNION ALL
    SELECT  PerformanceRDF_View.OID,
            PerformanceRDF_View.fk_performer AS fk_performerEKRX08
    FROM PerformanceRDF_View
    WHERE PerformanceRDF_View.fk_performer IS NOT NULL) b
WHERE a.OID = fk_performerEKRX08;
```

Consulta 3

```
SELECT x.typeGR15NX AS c,x.titleXB1CN8 AS b,
        s.typeL4DV29 AS y, s.fk_published_asP8XTH6 AS x,
        a.typePEMY7U AS t, a.fk_produced_signalEDVWV1 AS s,
        a.placeTNLVR9 AS p, a.OID AS a
FROM
    (SELECT RecordingRDF.OID, RecordingRDF.place AS placeTNLVR9,
            RecordingRDF.fk_produced_signal AS fk_produced_signalEDVWV1,
            RecordingRDF.type AS typePEMY7U
    FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
    UNION ALL
    SELECT  RecordingRDF_View.OID, RecordingRDF_View.place AS placeTNLVR9,
            RecordingRDF_View.fk_produced_signal AS fk_produced_signalEDVWV1,
            RecordingRDF_View.type AS typePEMY7U
    FROM RecordingRDF_View
    WHERE RecordingRDF_View.place IS NOT NULL
            AND RecordingRDF_View.fk_produced_signal IS NOT NULL
            AND RecordingRDF_View.type IS NOT NULL) a ,
    (SELECT SignalRDF.OID, SignalRDF.fk_published_as AS fk_published_asP8XTH6,
            SignalRDF.type AS typeL4DV29
```

```
    FROM SignalRDF
    WHERE SignalRDF.fk_published_as IS NOT NULL
            AND SignalRDF.type IS NOT NULL
    UNION ALL
    SELECT  SignalRDF_View.OID, SignalRDF_View.fk_published_as AS fk_published_asP8XTH6,
            SignalRDF_View.type AS typeL4DV29
    FROM SignalRDF_View
    WHERE SignalRDF_View.fk_published_as IS NOT NULL
            AND SignalRDF_View.type IS NOT NULL) s ,
    (SELECT TrackRDF.OID, TrackRDF.title AS titleXB1CN8,
            TrackRDF.type AS typeGR15NX
    FROM TrackRDF
    WHERE TrackRDF.title IS NOT NULL
            AND TrackRDF.type IS NOT NULL
    UNION ALL
    SELECT  TrackRDF_View.OID, TrackRDF_View.title AS titleXB1CN8,
            TrackRDF_View.type AS typeGR15NX
    FROM TrackRDF_View
    WHERE TrackRDF_View.title IS NOT NULL
            AND TrackRDF_View.type IS NOT NULL) x
WHERE s.OID = fk_produced_signalEDVWV1 AND x.OID = fk_published_asP8XTH6;
```

Consulta 4

```
SELECT u.typeHPODZG AS j, u.name05YS7B AS n,
        fk_producerOVBZCW AS u, x.typeR8DI4R AS c,
        x.titleDJ0P30 AS b, s.typeP3QG2Z AS y,
        s.fk_published_asQU35V9 AS x, a.typeK7VHT5 AS t,
        a.fk_produced_signalANQO4Y AS s, a.place57GU3P AS p,
        a.OID AS a
FROM
    (SELECT RecordingRDF.OID,RecordingRDF.place AS place57GU3P,
            RecordingRDF.fk_produced_signal AS fk_produced_signalANQO4Y,
            RecordingRDF.type AS typeK7VHT5,
            RecordingRDF.fk_producer AS fk_producerOVBZCW
    FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
```

```
        AND RecordingRDF.fk_producer IS NOT NULL
UNION ALL
SELECT  RecordingRDF_View.OID, RecordingRDF_View.place AS place57GU3P,
        RecordingRDF_View.fk_produced_signal AS fk_produced_signalANQO4Y,
        RecordingRDF_View.type AS typeK7VHT5,
        RecordingRDF_View.fk_producer AS fk_producerOVBZCW
FROM RecordingRDF_View
WHERE RecordingRDF_View.place IS NOT NULL
        AND RecordingRDF_View.fk_produced_signal IS NOT NULL
        AND RecordingRDF_View.type IS NOT NULL
        AND RecordingRDF_View.fk_producer IS NOT NULL) a ,
(SELECT SignalRDF.OID, SignalRDF.fk_published_as AS fk_published_asQU35V9,
        SignalRDF.type AS typeP3QG2Z
FROM SignalRDF
WHERE SignalRDF.fk_published_as IS NOT NULL
        AND SignalRDF.type IS NOT NULL
UNION ALL
SELECT  SignalRDF_View.OID,
        SignalRDF_View.fk_published_as AS fk_published_asQU35V9,
        SignalRDF_View.type AS typeP3QG2Z
FROM SignalRDF_View
WHERE SignalRDF_View.fk_published_as IS NOT NULL
        AND SignalRDF_View.type IS NOT NULL) s ,
(SELECT TrackRDF.OID, TrackRDF.title AS titleDJ0P30,
        TrackRDF.type AS typeR8DI4R
FROM TrackRDF
WHERE TrackRDF.title IS NOT NULL
        AND TrackRDF.type IS NOT NULL
UNION ALL
SELECT  TrackRDF_View.OID, TrackRDF_View.title AS titleDJ0P30,
        TrackRDF_View.type AS typeR8DI4R
FROM TrackRDF_View
WHERE TrackRDF_View.title IS NOT NULL
        AND TrackRDF_View.type IS NOT NULL) x ,
(SELECT MusicArtistRDF.OID, MusicArtistRDF.name AS name05YS7B,
        MusicArtistRDF.type AS typeHPODZG
FROM MusicArtistRDF
WHERE MusicArtistRDF.name IS NOT NULL
```

```
            AND MusicArtistRDF.type IS NOT NULL
    UNION ALL
    SELECT  MusicArtistRDF_View.OID, MusicArtistRDF_View.name AS name05YS7B,
            MusicArtistRDF_View.type AS typeHPODZG
    FROM MusicArtistRDF_View
    WHERE MusicArtistRDF_View.name IS NOT NULL
            AND MusicArtistRDF_View.type IS NOT NULL) u
WHERE s.OID = fk_produced_signalANQO4Y AND u.OID = fk_producerOVBZCW
AND x.OID = fk_published_asQU35V9;
```

Consulta 5

```
SELECT a.labelJSMREJ AS m, x.label1150HX AS l,u.type3LLJZV AS j,
        u.nameKUSJNW AS n, a.fk_producerBWMTXL AS u, x.typeY5FO4I AS c,
        x.title69E7FO AS b, s.typeG0W1HF AS y, s.fk_published_asFSP5XH AS x,
        a.typeB0OT7G AS t, a.fk_produced_signalSJV0JZ AS s, a.placeQQTYEU AS p,
        a.OID AS a
FROM
    (SELECT RecordingRDF.OID, RecordingRDF.place AS placeQQTYEU,
            RecordingRDF.fk_produced_signal AS fk_produced_signalSJV0JZ,
            RecordingRDF.type AS typeB0OT7G,
            RecordingRDF.fk_producer AS fk_producerBWMTXL,
            RecordingRDF.label AS labelJSMREJ
    FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
            AND RecordingRDF.fk_producer IS NOT NULL
            AND RecordingRDF.label IS NOT NULL
    UNION ALL
    SELECT  RecordingRDF_View.OID, RecordingRDF_View.place AS placeQQTYEU,
            RecordingRDF_View.fk_produced_signal AS fk_produced_signalSJV0JZ,
            RecordingRDF_View.type AS typeB0OT7G,
            RecordingRDF_View.fk_producer AS fk_producerBWMTXL,
            RecordingRDF_View.label AS labelJSMREJ
    FROM RecordingRDF_View
    WHERE RecordingRDF_View.place IS NOT NULL
            AND RecordingRDF_View.fk_produced_signal IS NOT NULL
            AND RecordingRDF_View.type IS NOT NULL
```

```
        AND RecordingRDF_View.fk_producer IS NOT NULL
        AND RecordingRDF_View.label IS NOT NULL) a ,
(SELECT SignalRDF.OID, SignalRDF.fk_published_as AS fk_published_asFSP5XH,
SignalRDF.type AS typeG0W1HF
FROM SignalRDF
WHERE SignalRDF.fk_published_as IS NOT NULL
        AND SignalRDF.type IS NOT NULL
UNION ALL
SELECT  SignalRDF_View.OID,
        SignalRDF_View.fk_published_as AS fk_published_asFSP5XH,
        SignalRDF_View.type AS typeG0W1HF
FROM SignalRDF_View
WHERE SignalRDF_View.fk_published_as IS NOT NULL
        AND SignalRDF_View.type IS NOT NULL) s ,
(SELECT TrackRDF.OID, TrackRDF.title AS title69E7FO,
        TrackRDF.type AS typeY5FO4I, TrackRDF.label AS label1150HX
FROM TrackRDF
WHERE TrackRDF.title IS NOT NULL
        AND TrackRDF.type IS NOT NULL
        AND TrackRDF.label IS NOT NULL
UNION ALL
SELECT  TrackRDF_View.OID, TrackRDF_View.title AS title69E7FO,
        TrackRDF_View.type AS typeY5FO4I, TrackRDF_View.label AS label1150HX
FROM TrackRDF_View
WHERE TrackRDF_View.title IS NOT NULL
        AND TrackRDF_View.type IS NOT NULL
        AND TrackRDF_View.label IS NOT NULL) x ,
(SELECT MusicArtistRDF.OID, MusicArtistRDF.name AS nameKUSJNW,
        MusicArtistRDF.type AS type3LLJZV
FROM MusicArtistRDF
WHERE MusicArtistRDF.name IS NOT NULL
        AND MusicArtistRDF.type IS NOT NULL
UNION ALL
SELECT  MusicArtistRDF_View.OID, MusicArtistRDF_View.name AS nameKUSJNW,
        MusicArtistRDF_View.type AS type3LLJZV
FROM MusicArtistRDF_View
WHERE MusicArtistRDF_View.name IS NOT NULL
        AND MusicArtistRDF_View.type IS NOT NULL) u
```

```
WHERE s.OID = fk_produced_signalSJV0JZ
AND u.OID = fk_producerBWMTXL AND x.OID = fk_published_asFSP5XH;
```

Consulta 6

```
SELECT x.chart_positionHCIRMU AS z, x.typeSFQNJJ AS c, x.labelPBF98U AS l,
        x.title4NERFB AS b, s.typeR4WCVU AS y, s.fk_published_asC41TOV AS x, s.OID AS s
FROM
    (SELECT SignalRDF.OID, SignalRDF.fk_published_as AS fk_published_asC41TOV,
            SignalRDF.type AS typeR4WCVU
    FROM SignalRDF
    WHERE SignalRDF.fk_published_as IS NOT NULL AND SignalRDF.type IS NOT NULL
    UNION ALL
    SELECT SignalRDF_View.OID, SignalRDF_View.fk_published_as AS fk_published_asC41TOV,
            SignalRDF_View.type AS typeR4WCVU
    FROM SignalRDF_View
    WHERE SignalRDF_View.fk_published_as IS NOT NULL
            AND SignalRDF_View.type IS NOT NULL) s ,
    (SELECT chart_positionMultivalueRDF.OID,
            chart_positionMultivalueRDF.chart_position AS chart_positionHCIRMU,
            TrackRDF.title AS title4NERFB, TrackRDF.type AS typeSFQNJJ,
            TrackRDF.label AS labelPBF98U
    FROM chart_positionMultivalueRDF, TrackRDF
    WHERE chart_positionMultivalueRDF.chart_position IS NOT NULL
            AND TrackRDF.OID = chart_positionMultivalueRDF.OID
            AND TrackRDF.title IS NOT NULL AND TrackRDF.type IS NOT NULL
            AND TrackRDF.label IS NOT NULL
    UNION ALL
    SELECT  chart_positionMultivalueRDF.OID,
            chart_positionMultivalueRDF.chart_position AS chart_positionHCIRMU,
            TrackRDF_View.title AS title4NERFB, TrackRDF_View.type AS typeSFQNJJ,
            TrackRDF_View.label AS labelPBF98U
    FROM chart_positionMultivalueRDF,TrackRDF_View
    WHERE chart_positionMultivalueRDF.chart_position IS NOT NULL
            AND TrackRDF_View.OID = chart_positionMultivalueRDF.OID
            AND TrackRDF_View.title IS NOT NULL AND TrackRDF_View.type IS NOT NULL
            AND TrackRDF_View.label IS NOT NULL) x
WHERE x.OID = fk_published_asC41TOV;
```

ANEXO C – CONSULTAS TRADUZIDAS PARA SQL COM OVERFLOW ESPECÍFICO

Consulta 1

```
SELECT a.nameQD0R7K AS n, a.typePKGAT2 AS s,
        a.OID AS a
FROM
    (SELECT MusicArtistRDF.OID,MusicArtistRDF.name AS nameQD0R7K,
            MusicArtistRDF.type AS typePKGAT2
    FROM MusicArtistRDF
    WHERE MusicArtistRDF.name IS NOT NULL
            AND MusicArtistRDF.type IS NOT NULL
UNION ALL
    SELECT  t1.subj as OID, t1.obj as n,
            t2.obj as s
    from Overflow_MusicArtistRDF as t1
    left outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL
    UNION
    SELECT  t1.subj as OID, t1.obj as n,
            t2.obj as s
    from Overflow_MusicArtistRDF as t1
    right outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL)a;
```

Consulta 2

```
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  t1.subj as a,
    t1.obj as t,
    t2.obj as n
    from Overflow_MusicArtistRDF as t1
    left outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL
    UNION
    SELECT  t1.subj as a, t1.obj as t, t2.obj as n
    from Overflow_MusicArtistRDF as t1
    right outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
            AND t1.pred IS NOT NULL AND t2.pred IS NOT NULL)a,
    (SELECT PerformanceRDF.OID,PerformanceRDF.fk_performer AS fk_performerEKRX08
    FROM PerformanceRDF
    WHERE PerformanceRDF.fk_performer IS NOT NULL
    UNION
    SELECT  t1.subj as b, t1.obj as a
    from Overflow_PerformanceRDF as t1
    where t1.pred='fk_performer' AND t1.pred IS NOT NULL
    UNION
    SELECT  t1.subj as OID, t1.obj as fk_performer
    from Overflow_PerformanceRDF as t1
    where t1.pred='fk_performer' AND t1.pred IS NOT NULL) b
WHERE a.OID = fk_performerEKRX08;
```

Consulta 3

```
SELECT x.typeGR15NX AS c,x.titleXB1CN8 AS b,
        s.typeL4DV29 AS y, s.fk_published_asP8XTH6 AS x,
        a.typePEMY7U AS t, a.fk_produced_signalEDVWV1 AS s,
        a.placeTNLVR9 AS p, a.OID AS a
FROM
    (SELECT RecordingRDF.OID, RecordingRDF.place AS placeTNLVR9,
    RecordingRDF.fk_produced_signal AS fk_produced_signalEDVWV1,
    RecordingRDF.type AS typePEMY7U
    FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
    UNION ALL
    SELECT  t1.subj as a, t1.obj as p,
    t2.obj as s, t3.obj as c
    from Overflow_RecordingRDF as t1
    left outer join
    Overflow_RecordingRDF as t2
    on t1.subj=t2.subj
    left outer join
    Overflow_RecordingRDF as t3
    on t2.subj=t3.subj
    where t1.pred='place'
            and t2.pred='fk_produced_signal' and t3.pred='type'
    UNION
    SELECT  t1.subj as a, t1.obj as p,
            t2.obj as s, t3.obj as c
    from Overflow_RecordingRDF as t1
    right outer join
    Overflow_RecordingRDF as t2
    on t1.subj=t2.subj
    right outer join
    Overflow_RecordingRDF as t3
    on t2.subj=t3.subj
    where t1.pred='place'
            and t2.pred='fk_produced_signal' and t3.pred='type') a ,
```

```
(SELECT SignalRDF.OID, SignalRDF.fk_published_as AS fk_published_asP8XTH6,
        SignalRDF.typeAS typeL4DV29
FROM SignalRDF
WHERE SignalRDF.fk_published_as IS NOT NULL
        AND SignalRDF.type IS NOT NULL
UNION ALL
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
left outer join
Overflow_SignalRDF as t2
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type'
UNION
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
right outer join
Overflow_SignalRDF as t2
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type') s ,
(SELECT TrackRDF.OID, TrackRDF.title AS titleXB1CN8,
        TrackRDF.type AS typeGR15NX
FROM TrackRDF
WHERE TrackRDF.title IS NOT NULL
        AND TrackRDF.type IS NOT NULL
UNION ALL
SELECT  t1.subj as x, t1.obj as b, t2.obj as c
from Overflow_TrackRDF as t1
left outer join
Overflow_TrackRDF as t2
on t1.subj=t2.subj
where t1.pred='title' and t2.pred='type'
UNION
SELECT  t1.subj as x, t1.obj as b, t2.obj as c
from Overflow_TrackRDF as t1
right outer join
Overflow_TrackRDF as t2
on t1.subj=t2.subj
where t1.pred='title' and t2.pred='type') x
```

```
WHERE s.OID = fk_produced_signalEDVWV1 AND x.OID = fk_published_asP8XTH6;
```

Consulta 4

```
SELECT u.typeHPODZG AS j,u.name05YS7B AS n,
        a.fk_producerOVBZCW AS u, x.typeR8DI4R AS c,
        x.titleDJ0P30 AS b, s.typeP3QG2Z AS y,
        s.fk_published_asQU35V9 AS x, a.typeK7VHT5 AS t,
        a.fk_produced_signalANQO4Y AS s, a.place57GU3P AS p,
        a.OID AS a FROM
    (SELECT RecordingRDF.OID,RecordingRDF.place AS place57GU3P,
            RecordingRDF.fk_produced_signal AS fk_produced_signalANQO4Y,
            RecordingRDF.type AS typeK7VHT5,
            RecordingRDF.fk_producer AS fk_producerOVBZCW
    FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
            AND RecordingRDF.fk_producer IS NOT NULL
    UNION
    SELECT  t1.subj as a, t1.obj as p,
            t2.obj as s, t3.obj as t,
            t4.obj as u
    from Overflow_RecordingRDF as t1
    left outer join
    Overflow_RecordingRDF as t2
    on t1.subj=t2.subj
    left outer join
    Overflow_RecordingRDF as t3
    on t2.subj=t3.subj
    left outer join
    Overflow_RecordingRDF as t4
    on t3.subj=t4.subj
    where t1.pred='place'
            and t2.pred='fk_produced_signal'
            and t3.pred='type' and t4.pred='fk_producer'
    UNION
    SELECT  t1.subj as a, t1.obj as p,
            t2.obj as s, t3.obj as t,
```

```
            t4.obj as u
from Overflow_RecordingRDF as t1
right outer join
Overflow_RecordingRDF as t2
on t1.subj=t2.subj
right outer join
Overflow_RecordingRDF as t3
on t2.subj=t3.subj
right outer join
Overflow_RecordingRDF as t4
on t3.subj=t4.subj
where t1.pred='place'
            and t2.pred='fk_produced_signal' and t3.pred='type'
            and t4.pred='fk_producer') a ,
(SELECT SignalRDF.OID,SignalRDF.fk_published_as AS fk_published_asQU35V9,
            SignalRDF.type AS typeP3QG2Z
FROM SignalRDF
WHERE SignalRDF.fk_published_as IS NOT NULL
            AND SignalRDF.type IS NOT NULL
UNION ALL
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
left outer join
Overflow_SignalRDF as t2
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type'
UNION
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
right outer join
Overflow_SignalRDF as t2
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type') s,
(SELECT TrackRDF.OID,TrackRDF.title AS titleDJ0P30,
            TrackRDF.type AS typeR8DI4R
FROM TrackRDF
WHERE TrackRDF.title IS NOT NULL
            AND TrackRDF.type IS NOT NULL
```

```
    UNION ALL
    SELECT  t1.subj as x, t1.obj as b, t2.obj as c
    from Overflow_TrackRDF as t1
    left outer join
    Overflow_TrackRDF as t2
    on t1.subj=t2.subj
    where t1.pred='title' and t2.pred='type'
    UNION
    SELECT  t1.subj as x, t1.obj as b, t2.obj as c
    from Overflow_TrackRDF as t1
    right outer join
    Overflow_TrackRDF as t2
    on t1.subj=t2.subj
    where t1.pred='title' and t2.pred='type') x,
    (SELECT MusicArtistRDF.OID,MusicArtistRDF.name AS name05YS7B,
            MusicArtistRDF.type AS typeHPODZG
    FROM MusicArtistRDF
    WHERE MusicArtistRDF.name IS NOT NULL
            AND MusicArtistRDF.type IS NOT NULL
    UNION
    SELECT  t1.subj as u, t1.obj as n, t2.obj as j
    from Overflow_MusicArtistRDF as t1
    left outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
    UNION
    SELECT  t1.subj as u, t1.obj as n, t2.obj as j
    from Overflow_MusicArtistRDF as t1
    right outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type' ) u
WHERE s.OID = fk_produced_signalANQO4Y
AND u.OID = fk_producerOVBZCW AND x.OID = fk_published_asQU35V9;
```

Consulta 5

```
SELECT a.labelJSMREJ AS m, x.label1150HX AS l,
```

```
        u.type3LLJZV AS j, u.nameKUSJNW AS n,
        a.fk_producerBWMTXL AS u, x.typeY5FO4I AS c,
        x.title69E7FO AS b, s.typeG0W1HF AS y,
        s.fk_published_asFSP5XH AS x, a.typeB0OT7G AS t,
        a.fk_produced_signalSJV0JZ AS s, a.placeQQTYEU AS p,
        a.OID AS a
FROM
    (SELECT RecordingRDF.OID,RecordingRDF.place AS placeQQTYEU,
            RecordingRDF.fk_produced_signal AS fk_produced_signalSJV0JZ,
            RecordingRDF.type AS typeB0OT7G,
            RecordingRDF.fk_producer AS fk_producerBWMTXL,
            RecordingRDF.label AS labelJSMREJ
            FROM RecordingRDF
    WHERE RecordingRDF.place IS NOT NULL
            AND RecordingRDF.fk_produced_signal IS NOT NULL
            AND RecordingRDF.type IS NOT NULL
            AND RecordingRDF.fk_producer IS NOT NULL
            AND RecordingRDF.label IS NOT NULL
    UNION ALL
    SELECT  t1.subj as a, t1.obj as p,
            t2.obj as s, t3.obj as t,
            t4.obj as u, t5.obj as m
    from Overflow_RecordingRDF as t1
    left outer join
    Overflow_RecordingRDF as t2
    on t1.subj=t2.subj
    left outer join
    Overflow_RecordingRDF as t3
    on t2.subj=t3.subj
    left outer join
    Overflow_RecordingRDF as t4
    on t3.subj=t4.subj
    left outer join
    Overflow_RecordingRDF as t5
    on t4.subj=t5.subj
    where t1.pred='place' and t2.pred='fk_produced_signal'
    and t3.pred='type' and t4.pred='fk_producer'
            and t5.pred='label'
```

```
UNION
SELECT  t1.subj as a, t1.obj as p,
        t2.obj as s, t3.obj as t,
        t4.obj as u, t5.obj as m
from Overflow_RecordingRDF as t1
right outer join
Overflow_RecordingRDF as t2
on t1.subj=t2.subj
right outer join
Overflow_RecordingRDF as t3
on t2.subj=t3.subj
right outer join
Overflow_RecordingRDF as t4
on t3.subj=t4.subj
right outer join
Overflow_RecordingRDF as t5
on t4.subj=t5.subj
where t1.pred='place' and t2.pred='fk_produced_signal'
        and t3.pred='type' and t4.pred='fk_producer'
        and t5.pred='label') a ,
(SELECT SignalRDF.OID,
        SignalRDF.fk_published_as AS fk_published_asFSP5XH,
        SignalRDF.type AS typeG0W1HF
FROM SignalRDF
WHERE SignalRDF.fk_published_as IS NOT NULL
        AND SignalRDF.type IS NOT NULL
UNION ALL
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
left outer join
Overflow_SignalRDF as t2
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type'
UNION
SELECT  t1.subj as s, t1.obj as x, t2.obj as y
from Overflow_SignalRDF as t1
right outer join
Overflow_SignalRDF as t2
```

```
on t1.subj=t2.subj
where t1.pred='fk_published_as' and t2.pred='type' ) s ,
(SELECT TrackRDF.OID,TrackRDF.title AS title69E7FO,
        TrackRDF.type AS typeY5FO4I, TrackRDF.label AS label1150HX
FROM TrackRDF
WHERE TrackRDF.title IS NOT NULL
        AND TrackRDF.type IS NOT NULL
        AND TrackRDF.label IS NOT NULL
UNION ALL
SELECT  t1.subj as x, t1.obj as b,
        t2.obj as c, t3.obj as l
from Overflow_TrackRDF as t1
left outer join
Overflow_TrackRDF as t2
on t1.subj=t2.subj
left outer join
Overflow_TrackRDF as t3
on t2.subj=t3.subj
where t1.pred='title' and t2.pred='type' and t3.pred='label'
UNION
SELECT  t1.subj as x, t1.obj as b,
        t2.obj as c, t3.obj as l
from Overflow_TrackRDF as t1
right outer join
Overflow_TrackRDF as t2
on t1.subj=t2.subj
right outer join
Overflow_TrackRDF as t3
on t2.subj=t3.subj
where t1.pred='title' and t2.pred='type' and t3.pred='label' ) x ,
(SELECT MusicArtistRDF.OID, MusicArtistRDF.name AS nameKUSJNW,
        MusicArtistRDF.type AS type3LLJZV
FROM MusicArtistRDF
WHERE MusicArtistRDF.name IS NOT NULL
        AND MusicArtistRDF.type IS NOT NULL
UNION ALL
SELECT  t1.subj as u, t1.obj as n, t2.obj as j
from Overflow_MusicArtistRDF as t1
```

```
    left outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type'
    UNION
    SELECT  t1.subj as OID, t1.obj as n, t2.obj as j
    from Overflow_MusicArtistRDF as t1
    right outer join
    Overflow_MusicArtistRDF as t2
    on t1.subj=t2.subj
    where t1.pred='name' and t2.pred='type') u
WHERE s.OID = fk_produced_signalSJV0JZ
AND u.OID = fk_producerBWMTXL AND x.OID = fk_published_asFSP5XH;
```

Consulta 6

```
SELECT x.typeSFQNJJ AS c, x.labelPBF98U AS l,
        x.title4NERFB AS b, s.typeR4WCVU AS y,
        s.fk_published_asC41TOV AS x, s.OID AS s
FROM
    (SELECT SignalRDF.OID, SignalRDF.fk_published_as AS
            fk_published_asC41TOV, SignalRDF.type AS typeR4WCVU
    FROM SignalRDF
    WHERE SignalRDF.fk_published_as IS NOT NULL
            AND SignalRDF.type IS NOT NULL
    UNION ALL
    SELECT  t1.subj as s, t1.obj as x, t2.obj as y
    from Overflow_SignalRDF as t1
    left outer join
    Overflow_SignalRDF as t2
    on t1.subj=t2.subj
    where t1.pred='fk_published_as'
            and t2.pred='type'
    UNION
    SELECT  t1.subj as s, t1.obj as x, t2.obj as y
    from Overflow_SignalRDF as t1
    right outer join
    Overflow_SignalRDF as t2
    on t1.subj=t2.subj
```

```
    where t1.pred='fk_published_as'
            and t2.pred='type') s ,
    (SELECT chart_positionMultivalueRDF.OID,
            chart_positionMultivalueRDF.chart_position
                        AS chart_positionHCIRMU,
            TrackRDF.title AS title4NERFB,
            TrackRDF.type AS typeSFQNJJ,
            TrackRDF.label AS labelPBF98U
    FROM chart_positionMultivalueRDF, TrackRDF
    WHERE chart_positionMultivalueRDF.chart_position IS NOT NULL
            AND TrackRDF.OID = chart_positionMultivalueRDF.OID
            AND TrackRDF.title IS NOT NULL
            AND TrackRDF.type IS NOT NULL
            AND TrackRDF.label IS NOT NULL
    UNION ALL
    SELECT  chart_positionMultivalueRDF.OID,
            chart_positionMultivalueRDF.chart_position
                        AS chart_positionHCIRMU,
            TrackRDF_View.title AS title4NERFB,
            TrackRDF_View.type AS typeSFQNJJ,
            TrackRDF_View.label AS labelPBF98U
    FROM chart_positionMultivalueRDF,TrackRDF_View
    WHERE chart_positionMultivalueRDF.chart_position IS NOT NULL
            AND TrackRDF_View.OID = chart_positionMultivalueRDF.OID
            AND TrackRDF_View.title IS NOT NULL
            AND TrackRDF_View.type IS NOT NULL
            AND TrackRDF_View.label IS NOT NULL) x
WHERE x.OID = fk_published_asC41TOV;
```