UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MACEDO SOUSA MAIA

ASBJOIN: UMA ESTRATÉGIA ADAPTATIVA PARA CONSULTAS ENVOLVENDO OPERADORES DE JUNÇÃO EM LINKED DATA

FORTALEZA, CEARÁ

2013

MACEDO SOUSA MAIA

ASBJOIN: UMA ESTRATÉGIA ADAPTATIVA PARA CONSULTAS ENVOLVENDO OPERADORES DE JUNÇÃO EM LINKED DATA

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: Profa. Dra. Vania Maria Ponte Vidal

Co-Orientador: Prof. Dr. José Maria da Silva Monteiro Filho e Prof. Dr. Fábio André Machado Porto

FORTALEZA, CEARÁ

2013

A000z Maia, M. S..

ASBJoin: Uma estratégia adaptativa para consultas envolvendo operadores de junção em Linked Data / Macedo Sousa Maia. 2013.

97p.;il. color. enc.

Orientador: Profa. Dra. Vania Maria Ponte Vidal

Co-Orientador: Prof. Dr. José Maria da Silva Monteiro Filho e Prof. Dr. Fábio André Machado Porto

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, 2013.

1. Operações de Junção 2. Consultas Federadas 3. Adaptatividade I. Profa. Dra. Vania Maria Ponte Vidal(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD:000.0

MACEDO SOUSA MAIA

ASBJOIN: UMA ESTRATÉGIA ADAPTATIVA PARA CONSULTAS ENVOLVENDO OPERADORES DE JUNÇÃO EM LINKED DATA

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Profa. Dra. Vania Maria Ponte Vidal
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. José Maria
Universidade Federal do Ceará - UFC
Co-orientador

Prof. Dr. Fabio Porto
Laboratório Nacional de Computação
Cientifica - LNCC
Co-orientador

Profa. Dra. Ana Maria Moura
Laboratório Nacional de Computação
Cientifica - LNCC

Aos meus Pais.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus e aos meus pais(Jacinta e Paulo) pelo apoio constante e pelo incentivo em atingir as metas que a vida me impõe.

Agradeço também ao apoio e incentivo constante de minha namorada Camila que sempre esteve presente nos momentos bons e ruins, dando o suporte necessário.

Aos Professores Vânia Vidal e José Maria pela orientação e coorientação, respectivamente, bem como por todo o apoio recebido durante o período da pesquisa.

Ao Professor Fábio Porto que sempre foi muito atencioso e preciso em suas observações que foi de total importância para a pesquisa. Agradeço também a sua equipe do Laboratório DEXL-LNCC e à Professora Ana Maria Moura que tão bem me recepcionaram no LNCC.

Ao amigo Regis Pires pelos vários esclarecimentos práticos e teóricos sobre Web Semântica e Linked Data que foram de grande utilidade para o desenvolvimento deste trabalho.

Agradeço também a Dona Creusa, que me hospedou em sua casa durante minha estadia em Petrópolis e que também foi muito atenciosa e solidária em vários momentos.

Aos meus amigos de mestrado e doutorado Livio Freire, Wellington Franco, David Araújo, Jeovane Reges, Vinícius Pires, Adyson, Rainara, Jefferson Carvalho, Diego Victor, Mônica Regina, Manoel Siqueira, Dionísio, Diego Sá, Carlos Eduardo (Cadu), Thiago Vinutto, Arlino Henrique, Franzé, Luís Eufrásio, Igo Brilhante, Henrique Viana, Ticiana Linhares, Lívia Almada, Flávio Sousa, Leonardo Moreira, Paulo Rego, Mariano, Dener, André Fonteles, Gustavo, Rodrigo Félix, Rafael Lima, Bruno Leal e Ticianne Ribeiro pelos excelentes momentos de convívio e aprendizado que pudemos compartilhar ao longo dessa caminhada.

Agradeço a Rute Castro, Bruno Sabóia e toda a equipe do GREat que tão bem me recepcionaram quando comecei a trabalhar em um dos projetos dessa instituição.

Aos meus amigos Alessandra, Andrea, Isabel, Kleison, Luzianne e João Paulo pelo apoio diário.

Agradeço ao Orley pelas vezes que precisei resolver problemas relacionados ao mestrado.

Ao apoio financeiro recebido da CAPES durante o mestrado e durante a missão de estudos em Petrópolis.

”Portanto, a tarefa não é tanto ver o que ninguém viu ainda, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê”

(Arthur Schopenhauer)

RESUMO

Motivado pelo sucesso de Linked Data e impulsionado pelo crescimento do número de fontes de dados em formato RDF disponíveis na Web, novos desafios para processamento de consultas estão emergindo, especialmente em configurações distribuídas. No ambiente de Linked Data, é possível executar consultas federadas, as quais envolvem junções de dados fornecidos por múltiplas fontes. O termo consulta federada é usado quando queremos prover soluções baseadas em informações obtidas de diferentes fontes. Nesse sentido, a concepção de novos algoritmos e estratégias adaptativas para a execução de junções de forma eficiente constitui um desafio importante. Nesse trabalho, apresentamos uma solução para a execução adaptativa de operações de junções de dados em consultas federadas. A execução da operação de junção adaptativa entre informações contidas em fontes de dados distribuídas baseia-se em estatísticas, que são coletadas em tempo de execução. Uma informação estatística sobre uma determinada fontes seria, por exemplo, o tempo decorrido (Elapsed Time) para obter algum resultado. Para obter as informações estatísticas atualizadas, usamos uma estratégia que coleta essas informações durante a execução da consulta e,logo após, são armazenadas em uma base de dados local, na qual denominamos como catálogo de informações estatísticas.

Palavras-chave: Operações de Junção. Consultas Federadas. Adaptatividade.

ABSTRACT

Motivated by the success of *Linked Data* and driven by the growing number of data sources into RDF files available on the web, new challenges for query processing are emerging, especially in distributed settings. These environments allow distributed execution of federated queries, which involve joining data provided by multiple sources, which are often unstable. In this sense, the design of new algorithms and adaptive strategies for efficiently implementing joins is a major challenge. In this paper, we present a solution to the adaptive joins execution in federated queries. The adaptative context of distributed data sources is based on statistics that are collected at runtime. For this, we use a module that updates the information in the catalog as the query is executed. The module works in parallel with the query processor.

Keywords: Join Operator. Federated Queries. Adaptivity.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Web é atualmente um enorme espaço global de documentos e dados distribuídos em múltiplas fontes heterogêneas. Esta nova Web, denominada Web de Dados, visa pavimentar o caminho para a Web Semântica funcional, onde haverá a disponibilidade de uma grande quantidade de dados vinculados no modelo RDF.

A Web Semântica provê tecnologias para publicação, recuperação e integração de dados distribuídos na Web de dados. Resumidamente essas tecnologias são: **RDF** (Resource Description Framework), um modelo de dados simples, expressivo, extensível e que permite interligar itens entre diferentes fontes de dados; **URI (ou IRI)**, usado como mecanismo de nome global; **linguagem SPARQL** (PRUD'HOMMEAUX; SEABORNE, 2008), uma linguagem de consulta de alto nível capaz de abstrair detalhes da sequência de passos necessária para a execução de consultas sobre fontes heterogêneas; e possibilidade de uso de **mecanismos de inferência** sobre os dados. A integração de dados em grande escala é provavelmente uma das melhores aplicações para essas tecnologias, tendo em vista que o volume de dados e a quantidade de conjuntos de dados disponíveis se expandem a cada dia, dificultando cada vez mais seu consumo de forma integrada. Além disso, um conjunto de melhores práticas para publicação e conexão de dados estruturados na Web, conhecido como *Linked Data*, possibilita a interligação de itens entre diferentes fontes de dados e, portanto, a conexão dessas fontes em um único espaço de dados global (HEATH; BIZER, 2011). *Linked Data* se baseia em tecnologias da Web Semântica, e permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Além disso, a adoção de um modelo de dados padronizado (modelo RDF) e de um mecanismo padronizado de acesso aos dados, juntamente com a natureza autodescritiva dos dados também simplificam sua integração. A arquitetura aberta de *Linked Data* ainda permite a descoberta de novos dados e mesmo conjuntos de dados em tempo de execução. Desse modo, *Linked Data* tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados distintos a fim de facilitar sua integração e fornecendo, portanto, um novo cenário à integração de dados.

Diversas soluções para a integração de dados no contexto de *Linked Data* podem ser encontradas na literatura (HEATH; BIZER, 2011). Porém, em geral, podemos distinguir duas abordagens principais (HARTIG; LANGEGGER, 2010) : o enfoque materializado (ou *Data Warehousing*) e o enfoque virtual.

1. **Data warehousing (Enfoque materializado)**. Nesta abordagem os dados são coletados e armazenados em um banco de dados central. Consultas são executadas sobre esse banco de dados. A ideia de *datawarehouse* pode ser aplicada a um serviço de consulta sobre uma coleção de *Linked Data* copiados a partir de múltiplas fontes na Web. Essa coleção de dados pode ser construída através do carregamento de *dumps* dos dados em formato RDF para um *RDF Store*. Também é possível extrair esses dados ao percorrer e analisar informações sobre URIs na Web de Dados, ou através de consultas a um *Endpoint SPARQL*. A principal desvantagem dessa abordagem é a replicação dos dados, que além de reque-

rer espaço de armazenamento adicional, ainda possibilita o uso de dados desatualizados em relação aos dados originais. A possível desatualização dos dados leva à necessidade de percorrer a web de dados com certa frequência para atualizá-los. O principal desafio dessa abordagem é, portanto, criar e manter a visão de integração.

2. **Enfoque Virtual**. O enfoque virtual permite a execução de consultas federadas sobre um conjunto fixo de fontes de dados. Seu principal desafio consiste em encontrar planos de execução com desempenho satisfatório sobre múltiplas fontes de dados. A **federação de consultas** baseia-se na distribuição do processamento de consultas para múltiplas fontes de dados autônomas. A visão virtual de integração dos dados pode ser construída com ou sem o uso de um mediador, conforme as explicações a seguir:

   - **Uso de mediador**. Um mediador permite a execução de consultas *ad-hoc* sobre os conjuntos de dados registrados. Um mediador decompõe a consulta em subconsultas de forma transparente, direciona as subconsultas a múltiplos serviços de consulta distribuídos, e, finalmente, integra os resultados das subconsultas. O uso de um mediador aumenta a complexidade da arquitetura. No entanto, dá maior flexibilidade para formulação de novas consultas.
   - **Uso de ambiente para execução de consultas pré-definidas**. O uso de consultas pré-definidas permitem que planos de execução sejam gerados em tempo de projeto. Isso possibilita a análise e otimização das consultas sem as restrições que seriam impostas se essa tarefa ocorresse em tempo de execução. Aplicar a maior otimização possível em tempo de projeto pode viabilizar operações complexas cujo tempo de execução da consulta e transmissão dos resultados dominam o tempo de execução total.

1.1 Motivação

Motivados pelo sucesso da iniciativa *Linking Open Data*[1] e pelo grande crescimento da quantidade de fontes de dados disponíveis na web, novas abordagens de processamento de consultas estão surgindo. Enquanto o processamento de consultas no contexto do modelo RDF era tradicionalmente realizado usando armazenamento centralizado, ultimamente pode-se observar uma mudança de paradigma (SCHWARTE et al., 2011) para a adoção de abordagens federadas em decorrência da estrutura descentralizada da web. Na prática, inúmeros cenários existem em que mais de uma fonte de dados pode contribuir com informações, tornando o processamento de consultas mais complexo. O caminho natural segue a busca por soluções eficientes para o processamento de consultas federadas sobre fontes de dados distribuídas na web.

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

1.2 Caracterização do Problema

Abordagens tradicionais de federação de consultas baseiam-se na obtenção de informações úteis para a realização de otimizações, balanceamento de carga ou mesmo uso de dados locais, a partir da cooperação entre as fontes de dados. No entanto, fontes de *Linked Data* são normalmente publicadas por provedores independentes e autônomos, sobre os quais não temos controle. Essas fontes podem ficar sobrecarregadas ou mesmo inativas em certos momentos não previstos. Além disso, o protocolo *SPARQL* somente define como a consulta e os resultados são trocados entre clientes e *Endpoints*, mas não permite a cooperação entre eles. Realizar consultas federadas eficientes nesse cenário é um grande desafio que vem sendo abordado por algumas estratégias tratadas ao longo deste trabalho. No entanto, as estratégias atuais que endereçam esse problema ainda são relativamente recentes e muitas delas realizam apenas consultas federadas de forma bastante simplificada e sem maiores preocupações com otimização. Nesse sentido, a concepção de novos algoritmos e estratégias adaptativas para a execução de junções de forma eficiente constitui um desafio importante. A execução da operação de junção adaptativa entre informações contidas em fontes de dados distribuídas baseia-se em estatísticas, que são coletadas em tempo de execução. Uma informação estatística sobre uma determinada fontes seria, por exemplo, o tempo decorrido (*Elapsed Time*) para obter algum resultado. Para obter as informações estatísticas atualizadas, usamos uma estratégia que coleta essas informações durante a execução da consulta que,logo após, são armazenadas em uma base de dados local, na qual denominamos como catálogo de informações estatísticas.

1.3 Objetivo

A principal contribuição deste trabalho é a implementação e a validação, por meio de experimentos, de uma estratégia adaptativa para processamento de operações de junção, da álgebra *SPARQL*, em *Linked Data*. O operador de junção concebido, denominado ASBJoin, pode ser utilizado com até três fontes de dados. O ASBJoin define dinamicamente a ordem com que as fontes envolvidas na operação de junção são acessadas. Essa definição baseia em informações estatísticas sobre as fontes de dados envolvidas na junção. Essas informações são coletadas e atualizadas periodicamente por um módulo denominado monitor. O módulo monitor executa de forma concorrente com a execução da operação de junção. Na seção 4.3, discutiremos em maiores detalhes as informações estatísticas utilizadas pela abordagem proposta neste trabalho.

1.4 Estrutura da Dissertação

O restante desse trabalho é composto por seis capítulos, os quais estão estruturados da seguinte forma:

O Capítulo 2 – Fundamentação Teórica – apresenta uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dis-

sertação. Ele expõe os principais conceitos e ferramentas relacionados a integração de dados, Consultas Federadas, *Linked Data*, Junções de Dados sobre *Linked Data*, além de tratar também sobre a linguagem e a álgebra *SPARQL*.

O Capítulo 3 – Trabalhos Relacionados – trata das principais ferramentas existentes para lidar com a execução de operações de junção sobre *Linked Data*, destacando funcionalidades e desvantagens de cada uma delas.

O Capitulo 4 descreve as principais características do ASBJoin, uma estratégia para a execução de operações de junções adaptativas sobre *Linked Data*. Adicionalmente, a arquitetura concebida para a implementação do ASBJoin.

O Capitulo 5 descreve o operador de junção adaptativo denominado ASBJoin. O ASBJoin é um operador de junção da álgebra *SPARQL* que permite a junção de até três fontes de dados. O ASBJoin adapta dinamicamente a ordem com que as fontes envolvidas na operação de junção são acessadas.

O Capítulo 6 – Experimentos e Resultados – explica os experimentos realizados para avaliar a viabilidade do QEF-LD em comparação com outras estratégias de execução de consultas federadas, e analisa os resultados obtidos.

Por fim, o Capítulo 7 – Conclusão – tece as considerações finais sobre o trabalho e apresenta possíveis trabalhos futuros para dar prosseguimento ao que obtivemos até aqui.

2 FUNDAMENTAÇÃO TEÓRICA

A Web atual deixou de ser apenas um espaço global de documentos interligados e está se tornando também um enorme espaço global de dados vinculados constituído de bilhões de triplas RDF que cobrem os mais variados domínios, denominada Web de Dados. *Linked Data* define um conjunto de princípios que formam a base para a difusão e uso de dados na Web. Desde 2007 conjuntos de dados dos mais diversos domínios têm sido publicadas de acordo com estes princípios, gerando um volume crescente de dados e, consequentemente, uma demanda por seu consumo.

A Web de Dados fornece um novo cenário à integração de dados, mas também traz novos desafios. Neste capítulo trataremos desses assuntos, cujo entendimento é essencial para a compreensão do contexto em que esta dissertação está inserida, bem como da fundamentação necessária ao entendimento dos capítulos seguintes.

2.1 Integração de Dados

Integração de dados é o processo de permitir acesso transparente à múltiplos sistemas de informação heterogêneos e distribuídos (LANGEGGER, 2010).

O principal objetivo dos sistemas de integração de dados é permitir que usuários consultem simultaneamente múltiplas fontes de dados heterogêneas, distribuídas e autônomas por meio de uma única interface de consultas, mantendo transparentes os procedimentos de acesso, extração e integração dos dados. Assim o sistema de integração de dados deve tratar de forma transparente problemas de heterogeneidade (estrutural, conceitual e tecnológica), distribuição e autonomia das fontes durante a execução de consultas, que são descritos a seguir.

Distribuição. As fontes de dados estão dispersas geograficamente, sendo interligadas por meio de uma rede de computadores. Como consequência da distribuição, é necessário lidar com os problemas envolvidos nas redes, como replicação, fragmentação e custo da transmissão dos dados e a capacidade de processamento de cada servidor.

Autonomia. Refere-se ao nível de independência de operação de cada fonte de dados que participe de um sistema de integração, em que as fontes possuem controle total sobre os dados e, geralmente, não podem afetar suas funcionalidades e requerer modificações.

Heterogeneidade. Como os esquemas das fontes são desenvolvidos independentemente, eles possuem estruturas e terminologias diferentes (heterogeneidade estrutural e semântica), o que ocorre tanto com os esquemas que vêm de domínios diferentes, quanto com os modelados no mesmo domínio do mundo real, pelo fato de serem desenvolvidos por pessoas diferentes, em diferentes contextos. Para serem efetivos, os sistemas de integração de dados devem ser capazes de transformar dados de diferentes fontes para responder a consultas feitas sobre esse esquema.

2.2 Linked Data

A Web atual deixou de ser apenas um espaço global de documentos interligados e está se tornando um enorme espaço global de dados vinculados constituído de bilhões de triplas RDF que cobrem os mais variados domínios (HEATH; BIZER, 2011). Esta nova Web, denominada Web de Dados, visa pavimentar o caminho para a Web Semântica funcional, onde haverá a disponibilidade de uma grande quantidade de dados vinculados em formato RDF. A Web Semântica fornece tecnologias para efetivamente publicar, recuperar e descrever dados distribuídos na Web. A integração de dados em grande escala é provavelmente uma das melhores aplicações para essas tecnologias. A Web de Dados baseia-se nos princípios *Linked Data* delineados pelo diretor geral do W3C, o pesquisador Tim Berners-Lee. De fato, *Linked Data* é um conjunto de melhores práticas para publicação e conexão de dados estruturados na Web que se baseia em tecnologias da Web Semântica, e que permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Desse modo, Linked Data tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados. Devido a esta conexão, aplicações Web que visam integrar dados, tais como Mashups, podem se beneficiar do uso de Linked Data.

Duas razões tornam Linked Data uma solução promissora para integrar dados semanticamente (RIBEIRO, 2011):

1. é mais fácil criar e manter um Linked Data Mashup do que um mashup convencional, uma vez que, no primeiro caso, as relações entre os vocabulários das fontes já são identificadas e uma linguagem de consulta de alto nível – como SPARQL – pode ser usada para consultar conjuntos de dados heterogêneos;

2. há uma quantidade crescente de dados RDF publicados na Web de acordo com princípios de Linked Data. No entanto, um problema recorrente é de como especificar automaticamente uma ordem para execução de uma consulta sobre diversos conjuntos de dados. Atualmente, embora existam soluções capazes de executar tais planos de consulta sobre conjuntos de dados de Linked Data, a tarefa de especificá-los ainda é executada manualmente pelo usuário. Portanto, há a necessidade de um processo comum, bem definido, que permita a construção dessa especificação, de forma simples.

A Web de Dados cria inúmeras oportunidades para a integração semântica de dados, fomentando o desenvolvimento de novos tipos de aplicações e ferramentas. Muito esforço tem sido despendido pela comunidade para o desenvolvimento de navegadores, mecanismos de busca e outras ferramentas específicas para consumo de dados vinculados. Além disso, muitas ferramentas para publicação de dados seguindo os princípios de *Linked Data* foram desenvolvidas e disponibilizadas, motivando a publicação de dados de forma aberta. Essas forças têm revelado desafios a ser superados para o uso efetivo da Web de Dados, o que tem aumentado o interesse de pesquisa nesta área.

2.2.1 Definição

Linked Data é um conjunto de melhores práticas para publicação e consumo de dados estruturados na Web, permitindo estabelecer ligações entre itens de diferentes conjuntos de dados para formar um único espaço de dados global (HEATH; BIZER, 2011). Os dados publicados na Web de acordo com essas melhores práticas podem ser processados por máquinas, possuem significado explicitamente definido e podem estar ligados a outras fontes de dados. (BIZER; HEATH; BERNERS-LEE, 2009b) resume *Linked Data* como o uso da Web para criar ligações tipadas entre dados de diferentes conjuntos de dados.

2.2.2 Princípios e Boas Práticas

As melhores práticas relacionadas à *Linked Data* foram inicialmente propostas por (BERNERS-LEE, 2006) e ficaram conhecidas como os princípios de *Linked Data* que são enumerados a seguir:

1. Usar URIs como nomes para coisas.
2. Usar URIs HTTP para que as pessoas possam procurar esses nomes.
3. Quando alguém procurar uma URI, prover informação útil, usando os padrões (RDF, SPARQL).
4. Incluir links para outras URIs, de modo que possam permitir a descoberta de mais coisas.

Esses princípios fornecem a base para a publicação e interligação de dados estruturados na Web. Posteriormente, eles foram estendidos por documentos originados a partir das experiências da comunidade de *Linked Data* (BIZER; CYGANIAK; HEATH, 2007; SAUERMANN; CYGANIAK; VÖLKEL, 2007), resultando em boas práticas de publicação e consumo de *Linked Data*. Algumas dessas boas práticas relacionadas a *Linked Data* são tratadas na seção 2.2.7, após a exposição de alguns fundamentos necessários ao seu entendimento.

2.2.3 Dados Abertos

Tim Berners-Lee sugeriu uma forma de classificar o nível de abertura dos dados baseada em estrelas (BERNERS-LEE, 2006). Nesse tipo de classificação, os dados abertos podem ter entre uma e cinco estrelas. Assim, quanto mais abertos forem os dados, mais estrelas eles terão. Dados sob licença aberta possuem uma estrela. Se, além disso, os dados estiverem disponibilizados de forma estruturada, como em forma de planilha, por exemplo, eles terão duas estrelas. Três estrelas são atribuídas aos dados que usam formato não proprietário como CSV ou XML. Caso os dados estejam usando os padrões Web recomendados pelo W3C como modelo RDF, URIs e linguagem SPARQL, eles são classificados com quatro estrelas. Por fim, as cinco estrelas são atribuídas aos dados que além das características já mencionadas em relação as demais estrelas, ainda estão conectados com dados de outros conjuntos de dados através de links RDF.

2.2.4 Conjuntos de Dados

Um **conjunto de dados** é um conjunto de triplas RDF publicadas, mantidas ou agregadas por um único provedor. Para facilitar a identificação de conjuntos de dados relevantes na Web é importante que cada conjunto de dados publique metadados com informações de proveniência, que podem incluir o mantenedor do conjunto de dados, informações de licença e tópicos de assuntos relacionados ao conjunto de dados. O vocabulário padronizado pelo W3C para publicação de metadados sobre conjunto de dados disponíveis como Linked Data é o *VoiD – Vocabulary of Interlinked Datasets* (ALEXANDER et al., 2009, 2010). Além disso, os conjuntos de dados devem ser registrados em repositórios de informações sobre vários conjuntos de dados disponíveis, como por exemplo: CKAN[1], VOIDstore[2] e VoiD Browser[3]. A ferramenta WoDQA[4] – Web of Data Querying Application – pode ser usada para automatizar a identificação de conjuntos de dados nos repositórios de documentos VoiD a partir de padrões de triplas fornecidos.

2.2.5 Web de Documentos e Web de Dados

Para facilitar o entendimento da Web de dados, podemos estabelecer um paralelo com a Web de documentos que já conhecemos. A Web de dados pode ser acessada a partir de navegadores RDF, assim como os navegadores HTML são usados para acessar a Web de documentos. Enquanto na Web de documentos usamos links HTML para navegar entre diferentes páginas, na Web de dados os links RDF são usados para acessar dados de outras fontes. Portanto, os links de hipertexto são capazes de conectar os documentos, assim como os links RDF interligam os dados. A Tabela 2.1 ilustra algumas comparações entre a Web de Documentos e a Web de Dados.

Web de Documentos	**Web de Dados**
Navegadores HTML	Navegadores RDF
Links HTML conectando documentos	Links RDF interligando dados
Mecanismo de identificação – URIs	Mecanismo de identificação – URIs
Mecanismo de acesso – HTTP	Mecanismo de acesso – HTTP
Formato de conteúdo – HTML	Modelo de dados – RDF
—	Linguagem de consulta – SPARQL

Tabela 2.1: Comparações entre a Web de Documentos e a Web de Dados

Além disso, a Web de documentos está alicerçada em um pequeno conjunto de padrões: um mecanismo de identificação global e único (*URIs – Uniform Resource Identifiers*), um mecanismo de acesso universal (*HTTP – Hypertext Transfer Protocol*) e um formato de conteúdo amplamente usado (*HTML – Hypertext Markup Language*). De modo semelhante, a Web de dados também tem por base alguns padrões bem estabelecidos como: o mesmo mecanismo

[1] http://ckan.org/
[2] http://void.rkbexplorer.com/
[3] http://kwijibo.talis.com/voiD/
[4] http://seagentdev.ege.edu.tr:8180/seagent/wodqa/wodqa.seam

de identificação usado na Web de documentos (URIs), um modelo de dados comum (RDF) e uma linguagem de consulta para acesso aos dados (SPARQL). O modelo RDF (MANOLA; MILLER, 2004) é baseado na ideia de identificar os recursos da Web usando identificadores chamados *Uniform Resource Identifiers – URIs*[5], e descrever tais recursos em termos de propriedades, as quais podem apontar para outras URIs ou ser representadas por literais. Esses padrões serão abordados a seguir.

2.2.6 Padrões

Os padrões abertos adotados em *Linked Data* são amplamente difundidos e suportados pelas mais variadas linguagens de programação.

URI – Uniform Resource Identifier
Um Identificador Uniforme de Recursos (*URI – Uniform Resource Identifier*) é uma sequência compacta de caracteres que identifica um recurso físico ou abstrato (BERNERS-LEE; FIELDING; MASINTER, 1998). URIs são usadas no contexto de *Linked Data* para identificar objetos e conceitos, permitindo que eles sejam dereferenciados para obtenção de informações a seu respeito. Assim, uma URI dereferenciada resulta em uma descrição RDF do recurso identificado. Por exemplo, a URI *http://www.w3.org/People/Berners-Lee/card#i* identifica o pesquisador Tim Bernes-Lee.

Protocolo *HTTP – Hypertext Transfer Protocol*
O protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*) é o mecanismo padrão de acesso aos documentos e dados na Web. É um protocolo genérico, sem estado e no nível de aplicação para sistemas distribuídos, colaborativos e hipermídia. Uma característica do HTTP é a tipagem e negociação de representação de dados, que permitem a construção de sistemas de forma independente dos dados transferidos (FIELDING et al., 1999).

RDF – Resource Description Framework
A utilização um modelo de dados comum – modelo RDF – torna possível a implementação de aplicações genéricas capazes de operar sobre o espaço de dados global (HEATH; BIZER, 2011). O modelo RDF (MANOLA; MILLER, 2004) é um modelo de dados descentralizado, baseado em grafo e extensível, possuindo um alto nível de expressividade e permitindo a interligação entre dados de diferentes conjuntos de dados. Ele foi projetado para a representação integrada de informações originárias de múltiplas fontes. Os dados são descritos na forma de triplas com sujeito, predicado e objeto, onde o sujeito é uma URI, o objeto pode ser uma URI ou um literal e o predicado é uma URI que define como sujeito e predicado estão relacionados. Por exemplo, a afirmação em português *'http://www.w3.org/People/Berners-Lee/card#i tem uma propriedade denominada creator cujo valor é Tim Bernes-Lee'* pode ser definida através de uma tripla RDF conforme ilustrado na Figura 2.1.

Cada tripla faz parte da Web de Dados e pode ser usada como ponto de partida para explorar esse espaço de dados. Triplas de diferentes conjuntos de dados podem ser facilmente combinadas para formar um único grafo. Além disso, é possível usar termos de diferentes

[5]http://www.ietf.org/rfc/rfc2396.txt

```
Sujeito: <http://www.w3.org/People/Berners-Lee/card#i>
Predicado: <http://purl.org/dc/elements/1.1/creator>
Objeto: "Tim Bernes-Lee"
```

Figura 2.1: Exemplo de tripla RDF

vocabulários para representar os dados. O modelo RDF ainda permite a representação de dados em diferentes níveis de estruturação, sendo possível representar desde dados semiestruturados a dados altamente estruturados.

Links RDF

No contexto de *Linked Data* os *Links RDF* descrevem relacionamentos entre dois recursos (HEATH; BIZER, 2011). Um *link RDF* consiste de uma tripla com três URIs. As URIs referentes ao sujeito e objeto identificam os recursos relacionados. A URI referente ao predicado define o tipo de relacionamento entre os recursos. Uma distinção útil que pode ser feita é com relação a *links* internos e externos. *Links RDF* internos conectam recursos dentro de um único conjunto de dados. *Links* externos conectam recursos servidos por diferentes conjuntos de dados *Linked Data*. No caso de *links* externos, as URIs referentes ao sujeito e predicado do link pertencem a espaço de nomes *(namespaces)* distintos. *Links* externos são cruciais para a Web dos Dados visto que eles permitem interligar as fontes de dados dispersas em um espaço global de dados.

A figura 2.2 apresenta dois exemplos de *links RDF*. O primeiro exemplo interliga o perfil FOAF do pesquisador Tim Berners-Lee localizado em um arquivo RDF ao recurso que o identifica na fonte de dados do DBLP. No segundo exemplo, o recurso que identifica Tim Berners-Lee na fonte DBpedia também é ligado ao recurso na fonte DBLP que o identifica. A propriedade *http://www.w3.org/2002/07/owl#sameAs* define que os recursos interligados representam a mesma entidade do mundo real.

```
Sujeito:   <http://www.w3.org/People/Berners-Lee/card#i>
Predicado: <http://www.w3.org/2002/07/owl\#sameAs>
Objeto:    <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007>
```

```
Sujeito:   <http://dbpedia.org/resource/Tim\_Berners-Lee>
Predicado: <http://www.w3.org/2002/07/owl\#sameAs>
Objeto:    <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007>
```

Figura 2.2: Exemplos de links RDF

O armazenamento de dados no modelo RDF pode ser realizado através de grafo em memória, arquivo texto ou banco de dados específico para armazenamento de triplas RDF, chamado de *RDF Store, Triple Store* ou *Quad Store*. Normalmente uma *Triple Store* é de fato uma *Quad Store*, pois suporta Grafos Nomeados *(Named Graphs)*. Um grafo nomeado é simplesmente uma coleção de triplas RDF nomeada por uma URI que identifica o grafo, com a finalidade de caracterizar a proveniência dos dados RDF. O armazenamento de triplas em arquivo texto usa algum formato de serialização de RDF, como RDF/XML, Notation3 (N3),

Turtle, NTriples ou RDF/JSON.

2.2.7 Boas Práticas

A adoção de boas práticas relacionadas a *Linked Data* facilita a descoberta de informações relevantes para a integração de dados entre diferentes fontes. A seguir serão descritas algumas dessas práticas.

- **Selecionar URIs adequadas.** Deve-se evitar URIs contendo algum detalhe de implementação ou do ambiente em que estão publicadas. Como exemplo a evitar, consideremos o URI *http://lia.ufc.br:8080/ regispires/cgi-bin/resource.php?id=ufc* que possui detalhes da porta 8080 usada em seu ambiente de publicação e do script implementado em PHP necessário à sua execução.

  É frequente o uso de três URIs relacionadas a cada recurso: (i) um identificador para o recurso; (ii) um identificador para informações sobre o recurso para visualização através de navegadores HTML; (iii) um identificador para informações sobre o recurso em formato RDF/XML. A figura 2.3 apresenta um exemplo de três URIs relacionadas à pesquisadora Vânia Vidal na fonte DBLP. A representação de um recurso através diferentes URIs permite que a interface Linked Data realize o dereferenciamento da URI de acordo com o tipo de conteudo requisitado no cabeçalho HTTP (i.e. Text/HTML, application/rdf+xml, etc.).

```
http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal
http://dblp.l3s.de/d2r/page/V%C3%A2nia_Maria_Ponte_Vidal
http://dblp.l3s.de/d2r/data/V%C3%A2nia_Maria_Ponte_Vidal
```

Figura 2.3: Exemplos de URIs relacionadas a um mesmo recurso

  A figura 2.4 apresenta dois exemplos de requisições HTTP referente à URI da pesquisadora Vânia Vidal na fonte DBLP. No exemplo referente ao item (a), a requisição define como tipo MIME, dados no modelo RDF e recebe como resposta, através do redirecionamento 303, a URI referente aos dados da pesquisadora. No exemplo referente ao item (b), a requisição solicita os dados no formato HTML e recebe como resposta o redirecionamento para a URI referente à pagina HTML da pesquisadora.

- **Usar URIs dereferenciáveis** para que a descrição do recurso possa ser obtida da Web.

- **Utilizar URIs estáveis.** A alteração de URIs quebra links já estabelecidos, criando um problema para a localização de recursos. Para evitar esse tipo de alteração, recomenda-se um planejamento meticuloso das URIs que serão usadas e também que o responsável pela publicação detenha a propriedade do espaço de nomes.

- **Criar links para outras fontes de dados** de modo a permitir a navegação entre as fontes de dados. Os *links* podem ser criados de forma manual ou automatizada.

```
(a)
$ curl -H "Accept: application/rdf+xml"
   http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal

303 See Other: For a description of this item,
see http://dblp.l3s.de/d2r/data/V%C3%A2nia_Maria_Ponte_Vidal
```

```
(b)
$ curl -H "Accept: text/html"
   http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal

303 See Other: For a description of this item,
see http://dblp.l3s.de/d2r/page/V%C3%A2nia_Maria_Ponte_Vidal
```

Figura 2.4: Exemplos de URIs relacionadas a um mesmo recurso

- **Publicação de Metadados.** Análise dos metadados facilita a seleção dos dados relevantes. Devem ser fornecidos metadados sobre proveniência e licenciamento dos dados. Também é recomendável a disponibilização de metadados sobre a fonte de dados. O vocabulário mais usado atualmente para publicação de metadados sobre conjunto de dados disponíveis é o *VoiD – Vocabulary of Interlinked Datasets*, conforme foi mencionado na Seção 2.2.4.

- **Utilizar termos de vocabulários amplamente usados**. Embora não haja restrições para seleção de vocabulários, é considerada uma boa prática o reuso de termos de vocabulários RDF amplamente usados para facilitar o processamento de *Linked Data* pelas aplicações clientes (BIZER; CYGANIAK; HEATH, 2007). Novos termos só devem ser definidos se não forem encontrados em vocabulários já existentes. A seguir apresentamos alguns vocabulários bastante difundidos: *Friend-of-a-Friend* (FOAF), *Semantically-Interlinked Online Communities* (SIOC), *Simple Knowledge Organization System* (SKOS), *Description of a Project* (DOAP), *Creative Commons* (CC) e *Dublin Core* (DC). Uma relação mais extensa desses vocabulários é mantida pelo projeto *Linking Open Data* no *ESW Wiki*[6].

- **Estabelecer relações entre os termos de vocabulários proprietários para termos de outros vocabulários.** Isso pode ser feito através do uso das propriedades *owl:equivalentClass*, *owl:equivalentProperty*, *rdfs:subClassOf*, *rdfs:subPropertyOf*. A figura 2.5 mostra que a classe Pessoa de um vocabulário local é equivalente à definição da classe *Person* no vocabulário da *DBpedia*. A definição de relações entre vocabulários facilita a integração de dados que utilizam esses vocabulários.

- **Explicitar formas de acesso adicional aos dados** como *endpoints SPARQL* e *RDF dumps*.

[6]http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/CommonVocabularies

```
<http://lia.ufc.br/Pessoa> owl:equivalentClass <http://dbpedia.org/ontology/Person> .
```

Figura 2.5: Relação de equivalência entre termo proprietário e termo da DBpedia

2.2.8 O Projeto Linking Open Data e outras iniciativas para fomentar a Web de Dados

Inúmeras iniciativas voltadas para fomentar a criação da Web de Dados surgiram nos últimos anos, como por exemplo, o projeto *Linking Open Data (LOD)*[7] que é um esforço comunitário iniciado em 2007 e suportado pelo W3C para identificar fontes de dados publicadas sob licenças abertas, convertê-las para RDF e publicá-las na Web usando os princípios de *Linked Data*. Até agosto de 2010, este projeto havia publicado 295 conjuntos de dados compostos por mais de 31 bilhões de triplas RDF e 500 milhões de *links* RDF englobando os mais variados domínios como informações geográficas, censo, pessoas, empresas, comunidades online, publicações científicas, filmes, músicas, livros, além de outros (BIZER, 2011). A figura 2.6 mostra um diagrama de nuvem[8] com as fontes de dados publicadas pelo projeto LOD e as interligações entre elas em setembro de 2011. O tamanho dos círculos corresponde ao número de triplas de cada fonte de dados. As setas indicam a existência de pelo menos 50 links entre duas fontes. A origem de uma seta indica a fonte que possui o link e a fonte referenciada é a fonte para a qual a seta está apontando. Setas bidirecionais representam fontes que se referenciam mutuamente. A espessura da seta corresponde ao número de links.

Outra importante iniciativa foi a criação do *Workshop Linked Data on the Web* (LDOW)[9] dentro da programação da *International World Wide Web Conference* (WWW2008), tendo, desde então, entre os membros de seu comitê organizador o pesquisador Tim Berners-Lee (W3C/MIT). No Brasil foi estabelecida no âmbito governamental a Infraestrutura Nacional de Dados Abertos (INDA)[10], uma importante iniciativa criada em 2011 para aplicar os princípios de *Linked Data* na publicação de dados governamentais abertos.

Para termos uma ideia da quantidade de dados que estão disponíveis na Web sob o formato de *Linked Data*, a Figura 2.7 ilustra algumas informações estatísticas sobre os principais domínios de conhecimento atualmente utilizados em *Linked Data*.

A seguir, listamos dois domínios de conhecimento (*Cross Domain* e *Life Science*) e algumas das fontes de dados que foram classificados como parte de um desses dois domínios. As fontes de dados, que serão descritas a seguir, foram utilizadas nos experimentos realizados com a finalidade de validar a abordagem proposta nesta dissertação e que serão discutidos, com mais detalhes, no capitulo 5.

CROSS DOMAIN(CD)

Cross Domain é o domínio de dados no qual as informações podem estar inseridos em dois ou mais domínios. Normalmente, uma base é classificada como *Cross Domain* quando

[7] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[8] http://lod-cloud.net/
[9] http://events.linkeddata.org/ldow2008/
[10] http://wiki.gtinda.ibge.gov.br/

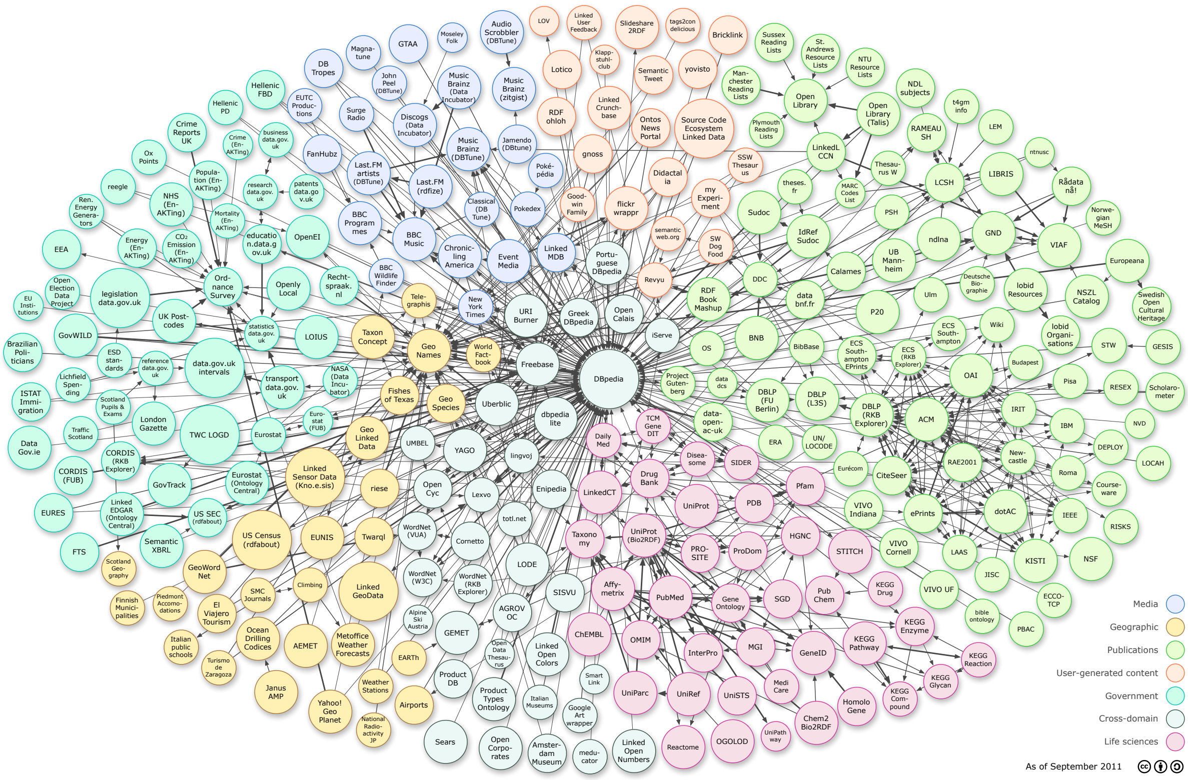

Figura 2.6: Diagrama de nuvem *Linking Open Data*, por Richard Cyganiak e Anja Jentzsch. Atualizado em 19/09/2011.

ela possui entidades que assumem diversos papéis não restringindo-se, apenas, a um único domínio em comum. Alguns exemplos de bases que fazem parte desse domínio são as que contém dados de enciclopédias digitais, jornais e revistas.

DBPedia[11]**:** Conjunto de dados amplamente conhecido por conter dados da Wikipédia em formato RDF. Atualmente, é um dos maiores conjuntos de dados, em número de dados, da Web de Dados e um dos que possuem mais ligações com outras fontes de dados.

NyTimes[12]**:** Contém informações triplificadas sobre noticias e informações publicadas no jornal The New York Times. Em Janeiro de 2010, o *The New York Times* publicou, aproximadamente, 10.000 diferentes assuntos publicados em seus periódicos no formato *Linked Data* sob a licença *Creative Commons (CC)*. Esse número de assuntos vem aumentando a cada dia. Nesse conjunto de dados são descritos diferentes tipos de recursos como, por exemplo: personalidades, organizações, lugares conhecidos e outras.

LIFE SCIENCE(LS)

Alguns dados sobre Ciências da Vida (*Life Science*) estão publicados na Web de Dados. *Life Science* é a área do conhecimento que envolve o estudo científico de organismos vivos como, por exemplo, microrganismos, plantas e animais em geral. Projetos como BioToRdf [13]

[11]http://dbpedia.org
[12]http://data.nytimes.com
[13]http://bio2rdf.org/

e Linked Life Data[14] tem como objetivo publicar dados sobre *Life Science* em formato *RDF*. A seguir serão mencionados alguns dos conjuntos de dados disponibilizados por esses projetos.

Kegg[15]: *KEGG* é um conjunto de dados para a compreensão de funções de alto nível e utilitários de sistemas biológicos, como células, organismos e ecossistemas. As informações providas pelo *KEGG* contém informações a nível molecular, especialmente conjuntos de dados moleculares em larga escala.

Diseasome: Possui informações úteis sobre doenças conhecidas e possibilita integração de dados com fontes de informações sobre drogas e medicamentos, por exemplo.

Dailymed[16]: Esse conjunto de dados contém dados sobre drogas e medicamentos. Dados do *Dailymed* são bastante usados por farmacêuticos e profissionais da área de saúde.

LinkedGeoData[17]: O projeto *LinkedGeoData* é um esforço para adicionar informações geográficas na Web de Dados. O objetivo do *LinkedGeoData* é utilizar dados geográficos coletados pelo projeto *OpenStreetMap*[18] e transformá-los em dados disponíveis em formato *RDF*, os quais são publicados de acordo com os princípios do *Linked Data*.

GeoNames[19]: *GeoNames* é um conjunto de dados que contém dados geográficos integrados, tais como nomes de lugares, altitude e população. Atualmente, o *GeoNames* é a segunda maior base de dados em formato *RDF*, sendo superado, apenas, pela *DBPedia*.

Domain	Number of datasets	Triples	%	(Out-)Links	%
Media	25	1,841,852,061	5.82 %	50,440,705	10.01 %
Geographic	31	6,145,532,484	19.43 %	35,812,328	7.11 %
Government	49	13,315,009,400	42.09 %	19,343,519	3.84 %
Publications	87	2,950,720,693	9.33 %	139,925,218	27.76 %
Cross-domain	41	4,184,635,715	13.23 %	63,183,065	12.54 %
Life sciences	41	3,036,336,004	9.60 %	191,844,090	38.06 %
User-generated content	20	134,127,413	0.42 %	3,449,143	0.68 %
	295	31,634,213,770		503,998,829	

Figura 2.7: Informações sobre os Domínios de Conhecimento da Web de Dados

A Figura 2.7 ilustra algumas informações estatísticas sobre os mais variados domínios de dados que compõem a Web de Dados. Na Figura 2.7 podemos observar, por exemplo, o número de conjuntos de dados por domínio de conhecimento, a quantidade total de triplas, a quantidade total de triplas contendo ligações com dados externos e a porcentagem de dados total por domínio. Desta forma, pode-se notar que, o domínio de dados governamentais compõe 42,09 porcento do total dos dados contido na Web de Dados.

[14] http://linkedlifedata.com
[15] http://www.genome.jp/kegg/
[16] http://dailymed.nlm.nih.gov/dailymed/
[17] http://linkedgeodata.org
[18] http://www.openstreetmap.org/
[19] http://www.geonames.org

A Figura 2.8 ilustra a quantidade de dados de cada domínio. Já a Figura 2.9 exibe a quantidade de triplas contendo ligações com dados externos por domínio. Nesse caso, as triplas de ligações com dados externos permitem ligar recursos que estão em diferentes conjunto de dados.

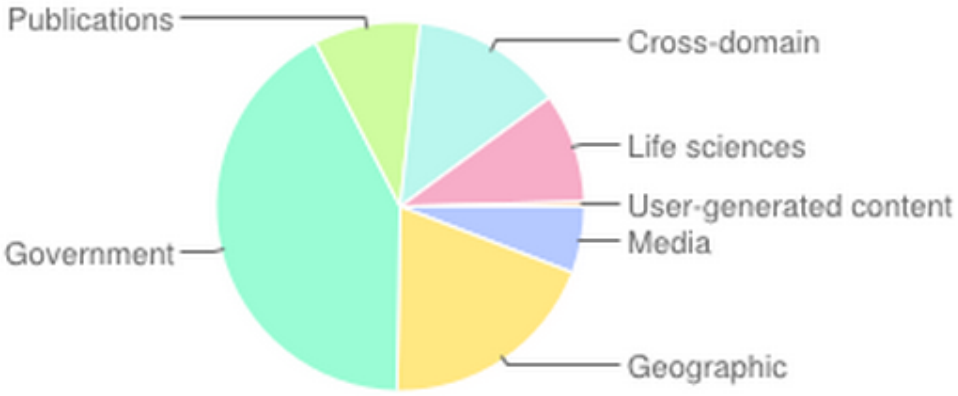

Figura 2.8: Quantidade de Dados por Domínio de Conhecimento

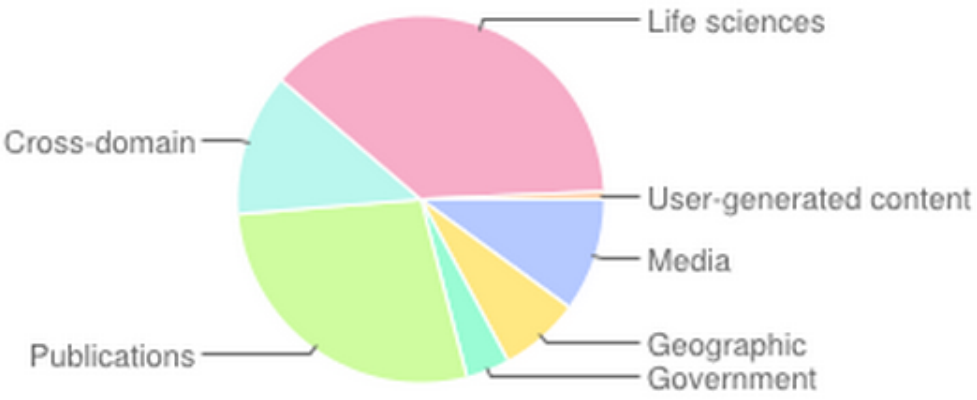

Figura 2.9: Quantidade de Triplas de Ligações por Domínio de Conhecimento

2.2.9 Conversão de dados para o modelo RDF e Wrappers RDF

Como RDF é o modelo de dados comum usado em *Linked Data*, os dados publicados devem estar nesse modelo ou serem convertidos para ele. Caso os dados não adotem o modelo RDF, há duas abordagens possíveis para tratar essa heterogeneidade:

(i) **Materialização dos dados no modelo RDF** – Usar um processo de **conversão**, onde os dados não RDF são usados para gerar um arquivo RDF através de uma ferramenta específica[20]. Desse modo, através de conversores específicos é possível converter bancos de dados relacionais, planilhas, arquivos CSV, arquivos XML e outros documentos para o formato RDF. O

[20]http://www.w3.org/wiki/ConverterToRdf

projeto *RDFizer* [21] contém informações de ferramentas para conversão de vários formatos de dados para RDF, além de hospedar algumas dessas ferramentas. Após geração do arquivo em formato RDF, seus dados podem ser carregados em uma *RDF Store* e publicados. Uma vantagem dessa abordagem é a melhoria de desempenho que pode ser obtida ao usar formas de armazenamento especificamente otimizadas para realizar a persistência de triplas RDF. No entanto, o armazenamento das triplas requer espaço extra em relação aos dados originais. Além disso, a conversão demanda certo tempo para ser realizada e os dados em RDF podem ficar desatualizados em relação aos dados originais.

(ii) **Fornecimento de uma visão RDF (abordagem virtual)** de dados que não estão no modelo RDF através de um *Wrapper RDF*. A conversão dinâmica realizada pelo *wrapper* baseia-se em mapeamentos estabelecidos entre o modelo nativo e o modelo RDF, devendo haver um *wrapper* específico para cada tipo de modelo. Um *Wrapper RDF* também pode prover uma visão RDF a dados que precisam ser acessados através de uma Web API. *RDF Book Mashup* (BIZER; CYGANIAK; HEATH, 2007) é uma aplicação *mashup* escrita em PHP que funciona como um *RDF Wrapper* usado para combinar dados obtidos a partir das APIs proprietárias *Amazon Web Service* e *Google Base*. Desse modo, informações sobre livros, autores, revisões e comparações de ofertas entre diferentes livrarias podem ser usados por clientes genéricos, incluindo navegadores e mecanismos de busca RDF. Essa abordagem tende a ter um desempenho inferior à abordagem anterior devido às traduções dinâmicas entre os modelos que devem ser realizadas a cada uso da visão RDF; e também devido ao modelo original não estar otimizado especificamente para uso de triplas. No entanto, o uso de *Wrappers RDF* traz grandes vantagens, pois como o acesso ocorre sobre os dados originais, a visão RDF não requer espaço de armazenamento extra e não corre o risco de ter dados desatualizados.

A ampla difusão dos Bancos de Dados Relacionais motiva a necessidade de publicação dos dados no modelo relacional como *Linked Data*. Assim, seguindo a abordagem (ii), há *Wrappers RDB para RDF* que criam visões RDF a partir de mapeamentos entre as estruturas relacionais e os grafos RDF. A plataforma D2RQ[22] (BIZER; SEABORNE, 2004) fornece a infraestrutura necessária para acessar bancos de dados relacionais como grafos RDF virtuais. Ela disponibiliza uma interface *Linked Data*, *SPARQL endpoint* e *dumps RDF* baseados em um mapeamento declarativo entre o esquema do banco de dados e os termos RDF. A plataforma D2RQ possui os seguintes componentes:

- **Linguagem de mapeamento D2RQ** é uma linguagem declarativa para descrever as correspondências entre o modelo relacional e o modelo RDF. Os mapeamentos escritos em D2RQ são documentos RDF.
- **Mecanismo D2RQ** é um plug-in para os *frameworks Jena* e *Sesame* que usa os mapeamentos escritos na linguagem D2RQ para converter chamadas às APIs desses *frameworks* em consultas SQL ao banco de dados para obtenção dos resultados.
- **Servidor D2R**[23] (BIZER; SEABORNE, 2004) é um servidor HTTP que usa o meca-

[21] http://simile.mit.edu/RDFizers/
[22] http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/
[23] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

nismo D2RQ para prover uma interface *Linked Data* e um *SPARQL endpoint* sobre o banco de dados relacional.

Os componentes anteriormente descritos funcionam de forma integrada como pode ser observado na figura 2.10 que apresenta a arquitetura da plataforma D2RQ.

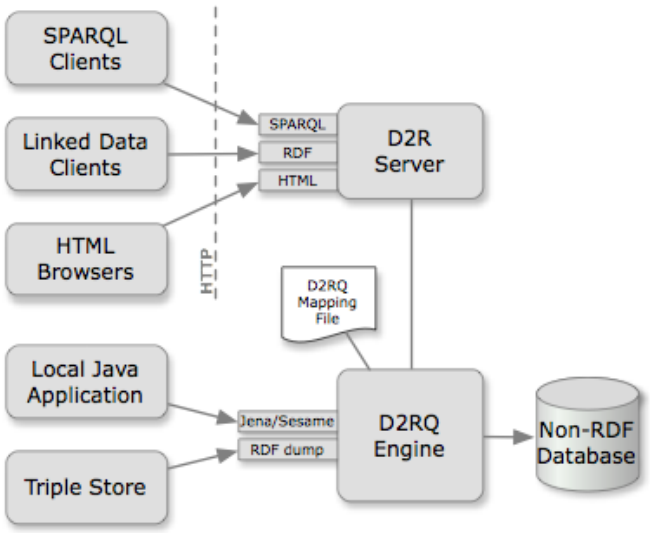

Figura 2.10: Arquitetura da plataforma D2RQ extraída de http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/

Além do D2R, duas outras ferramentas se destacam como *Wrappers RDB para RDF*: o *Virtuoso RDF Views* (ERLING; MIKHAILOV, 2006) e o *Triplify*[24] (AUER et al., 2009). Este último é um pequeno *plugin* para aplicações Web que permite mapear os resultados de consultas SQL em RDF, JSON e *Linked Data*. Depois disso, os dados podem ser compartilhados e acessados na web de dados. *Triplify* consiste de poucos arquivos, totalizando menos de 500 linhas de código. Importante ressaltar que D2R e Triplify além de fornecerem uma visão RDF aos dados relacionais, também permitem a geração de dumps RDF (materialização dos dados).

Um importante passo para a padronização desse tipo de solução para lidar com o modelo relacional foi a criação em 2009 do grupo de trabalho RDB2RDF[25] do W3C. Desde então, o grupo tem definido a linguagem padrão R2RML (DAS; SUNDARA; CYGANIAK, 2011) visando o mapeamento de dados e esquemas relacionais para RDF, que tende a substituir as soluções de mapeamento já existentes.

2.2.10 Aplicações Linked Data

URIs, palavras-chave e consultas SPARQL são usados como ponto de partida para o consumo de *Linked Data*. Assim, todas as aplicações que consomem a Web de dados usam direta ou indiretamente pelo menos um desses itens.

[24] http://triplify.org/
[25] http://www.w3.org/2001/sw/rdb2rdf/

Segundo (HEATH; BIZER, 2011), o consumo de *Linked Data* é realizado basicamente através de dois tipos de aplicações: **aplicações genéricas** que fazem uso de *Linked Data* de qualquer domínio e **aplicações de domínio específico** que são especificamente desenvolvidas para lidar com *Linked Data* relacionado a um determinado domínio.

2.2.10.1 Aplicações Genéricas

A utilização de padrões Web e um modelo de dados comum torna possível a implementação de aplicações genéricas capazes de operar sobre o espaço de dados global. Essas aplicações permitem o consumo de dados relacionados a múltiplos domínios distribuídos pela web de dados. Ao percorrer os *links RDF* é possível explorar e descobrir novas informações. A seguir serão abordados alguns tipos de aplicações genéricas normalmente usadas para acessar *Linked Data*.

Navegadores *Linked Data*
Navegadores *Linked Data* são aplicações executadas a partir dos navegadores Web convencionais que dereferenciam URIs e exibem uma visualização desse resultado, possibilitando a navegação aos dados de fontes relacionadas a partir dos *RDF links*. O *LOD Browser Switch*[26] é uma aplicação web que obtém detalhes a respeito de uma URI especificada pelo usuário a partir da seleção de um dos vários navegadores *Linked Data* disponibilizados pela aplicação. Assim é possível comparar a visualização de uma URI através de vários desses navegadores.

Alguns dos navegadores *Linked Data* são citados a seguir: *Explorator*[27] (ARAÚJO; SCHWABE, 2009; ARAÚJO; SCHWABE; BARBOSA, 2009), *Disco Hiperdata Browser*[28], *Marbles* [29], *Tabulator*[30] (BERNERS-LEE et al., 2007), *LinkSailor*[31] e *Graphite RDF Browser*[32]. A figura 2.11 exibe informações obtidas a partir do dereferenciamento da URI *http://dblp.l3s.de/d2r/re*
A._Casanova pelo *Disco*.

Mecanismos de Busca Linked Data O acesso à Web de Dados pode ocorrer a partir de mecanismos de busca específicos capazes de realizar pesquisas que levam em consideração a semântica dos dados. Esses mecanismos de busca permitem localizar recursos de diferentes fontes normalmente através de palavras-chave. A consulta pode ser realizada pelo usuário através de uma interface web ou através de serviços web providos pelos mecanismos de busca. Mecanismos de busca *Linked Data* percorrem a Web de dados percorrendo os *links* entre as fontes de dados e fornecendo a possibilidade de consultas sobre os dados dessas fontes. Os resultados das buscas são URIs que podem ser dereferenciadas e visualizadas através dos navegadores RDF. Atualmente há vários mecanismos de busca *Linked Data*. A seguir citamos alguns deles: *Sin-*

[26] http://browse.semanticweb.org/
[27] http://www.tecweb.inf.puc-rio.br/explorator
[28] http://www4.wiwiss.fu-berlin.de/rdf_browser/
[29] http://www5.wiwiss.fu-berlin.de/marbles/
[30] http://dig.csail.mit.edu/2005/ajar/ajaw/tab
[31] http://linksailor.com/
[32] http://graphite.ecs.soton.ac.uk/browser/

Figura 2.11: Visualização de informações sobre recurso através do navegador Disco

dice[33] (OREN et al., 2008), *VisiNav*[34], *Watson*[35] (D'AQUIN et al., 2007) e *Swoogle*[36] (DING et al., 2004).

Linked Data Mashups

Linked Data Mashups permitem aos usuários executar consultas e integrar dados estruturados e vinculados na web. Dados manipulados por *Linked Data Mashups*, em geral são dinamicamente recuperados através de um conjunto de especificações dos dados sobre fontes distribuídas e das definições de relacionamento entre estas fontes.

Especificar, construir e manter *Linked Data Mashups* não são tarefas fáceis, devido à necessidade de conhecer as URIs das fontes e o vocabulário utilizado por cada fonte, além da dificuldade para encontrar e estabelecer relações entre dados destas fontes, quando não estão vinculadas. Um dos objetivos deste trabalho é reduzir a complexidade da construção e manutenção de *Linked Data Mashups*.

[33] http://sindice.com/
[34] http://visinav.deri.org/
[35] http://watson.kmi.open.ac.uk/WatsonWUI/
[36] http://swoogle.umbc.edu/

Sig.ma[37] (TUMMARELLO et al., 2010) é um exemplo de *Linked Data Mashup* de uso genérico. Ele permite a busca de dados estruturados a partir de uma palavra-chave e os exibe em uma única página, integrando os dados de múltiplas fontes. A visão criada pelo *Sig.ma* baseia-se em resultados fornecidos pelo mecanismo de busca *Sindice*[38] (OREN et al., 2008). O usuário pode aprovar, rejeitar ou acrescentar fontes para estabelecer uma visão dos dados relevantes. Ao selecionar uma entidade da lista de resultados, uma nova visão é apresentada ao usuário. Um *link* permanente pode ser criado para futuros acessos ou compartilhamento dessa visão. As filtragens das fontes de dados realizadas pelos usuários coletivamente ajudam a classificar melhor a relevância das fontes e aperfeiçoar a qualidade dos resultados futuros. Além da interface web do usuário, *Sig.ma* ainda provê uma API destinada aos desenvolvedores de aplicações. A figura 2.12 ilustra o resultado de uma consulta sobre a pesquisadora Vânia Vidal envolvendo dezesseis fontes, onde quatro delas foram rejeitadas.

Figura 2.12: Visão criada pelo Sig.ma sobre a pesquisadora Vânia Vidal

Outras aplicações genéricas

Informações adicionais sobre determinado recurso podem ser obtidas através da localização de objetos referenciados pelas propriedades `rdfs:seeAlso` e `owl:sameAs`. Serviços online de coreferenciamento como o *sameAs*[39] são usados para encontrar URIs de diferentes fontes de dados que representam um mesmo conceito.

LDSpider[40] é um framework capaz de navegar pela web de dados seguindo *links* para obter dados de fontes *Linked Data* e os armazenar em uma *RDF Store* através de SPARQL Update ou como arquivo RDF.

[37] http://sig.ma/
[38] http://sindice.com/
[39] http://sameas.org
[40] http://code.google.com/p/ldspider/

2.2.10.2 Aplicações de domínio específico

Várias aplicações têm sido desenvolvidas para integrar *Linked Data* em domínios específicos. Essas aplicações também podem ser classificadas como *Linked Data Mashups*, mas voltados para um determinado domínio. A seguir descreveremos algumas delas.

Revyu[41] é uma aplicação web para crítica e classificação de qualquer item passível de avaliação. *Revyu* também disponibiliza uma API e um *SPARQL endpoint* para serem usados pelos desenvolvedores de aplicações.

DBpedia Mobile[42] (BECKER; BIZER, 2008) é uma aplicação cliente para dispositivos móveis consistindo de uma visão com um mapa e do navegador *Linked Data Marbles*. Baseado na localização geográfica de um dispositivo móvel, a aplicação exibe um mapa indicando localizações próximas a partir de dados extraídos das fontes *DBpedia*, *Revyu* e *Flickr*. O acesso ao *Flickr* é realizado através de um *Wrapper*. O usuário pode explorar informações sobre essas localizações e navegar em conjuntos de dados interligados. Também é possível a publicação de informações como *Linked Data*, de modo que possam ser usadas por outras aplicações.

Talis Aspire[43] é uma aplicação web voltada para que alunos e professores e possam encontrar os principais recursos educacionais em universidades do Reino Unido. O serviço é gratuito e provê ferramentas para criar e editar listas de leitura, além da produção e publicação de materiais educativos. Quando o usuário publica conteúdo, a aplicação cria triplas RDF em uma *RDF store*. Itens publicados são interligados de forma transparente a itens correspondentes de outras instituições.

BBC Programmes[44] e *BBC Music*[45] são projetos desenvolvidos pela *BBC Audio and Music Interactive*. A aplicação web *BBC Programmes* disponibiliza informações detalhadas sobre tipos, séries e episódios de todos os programas de TV e rádio transmitidos pela BBC. *BBC Music* fornece informações sobre artistas, vinculando-os aos programas da BBC. Assim é possível escolher um artista e obter todos os episódios de programas relacionados a ele. As aplicações mencionadas usam *Linked Data* como tecnologia de integração de dados, inclusive fazendo uso de vocabulários amplamente conhecidos como *DBpedia* e *MusicBrainz*.

2.2.11 APIs para manipulação de Linked Data

A seguir descreveremos algumas APIs para manipulação de dados na web semântica que são usadas no desenvolvimento de aplicações de domínio genérico ou específico para consumo de *Linked Data*.

[41] http://revyu.com/
[42] http://beckr.org/DBpediaMobile/
[43] http://www.talisaspire.com/
[44] http://www.bbc.co.uk/programmes
[45] http://www.bbc.co.uk/music

Sesame[46] e *Jena*[47] são *frameworks* de web semântica implementados em Java que fornecem APIs para manipulação de grafos RDF. Ambos possuem processador de consultas com suporte a consultas federadas em SPARQL (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). A maioria dos mediadores e demais ferramentas que lidam com federação de consultas SPARQL usam internamente uma dessas duas APIs.

Sesame permite armazenamento, consulta e manipulação de dados RDF. Além disso, o *framework* é extensível e configurável em relação a formas de armazenamento (memória e *RDF store*), mecanismos de inferência, formatos de arquivo RDF e linguagens de consulta (SPARQL e SeRQL).

Jena foi desenvolvido no *HP Labs* entre 2000 e 2009. Atualmente faz parte do projeto *Apache* e suas principais características são: suporte a RDF, RDFa, RDFS, OWL e SPARQL; armazenamento de triplas RDF em memória, banco de dados relacional *(Jena SDB)* ou *RDF store (Jena TDB)*; processamento de consultas SPARQL *(Jena ARQ)*; disponibilização de *SPARQL endpoint* (*Joseki* ou *Fuseki*); disponibilização de mecanismos de inferência embutidos e interfaces para mecanismos de inferência externos.

Named Graphs API for Jena (NG4J)[48] é uma extensão ao framework Jena para análise, manipulação e serialização de conjuntos de grafos nomeados representando os grafos como modelos ou grafos do Jena. NG4J permite o armazenamento de grafos em memória ou em banco de dados. Consultas SPARQL podem ser realizadas sobre os grafos nomeados.

O *Semantic Web Client Library* (SWClLib)[49] (HARTIG; BIZER; FREYTAG, 2009) faz parte do NG4J e é capaz de representar a web de dados como um único grafo RDF. Ele recupera informações dereferenciando URIs, seguindo *links rdfs:seeAlso* e consultando o mecanismo de busca *Sindice*. O SWClLib considera todos os dados como um único conjunto global de grafos nomeados, sendo usado na implementação de vários navegadores *Linked Data*. Os grafos recuperados são mantidos em um *cache* local para melhorar o desempenho de buscas futuras.

ARQ2[50] é uma biblioteca escrita em PHP que contempla armazenamento de Triplas RDF, *SPARQL endpoint* e interface *Linked Data* em uma única ferramenta. As triplas RDF são armazenadas em um banco de dados MySQL. A infraestrutura necessária para o funcionamento do ARQ2 é muito simples por requerer apenas um servidor Web com suporte a PHP e um banco de dados MySQL, sendo facilmente encontrada em qualquer serviço de hospedagem Web.

2.2.12 Abordagens para execução de consultas sobre múltiplas fontes de dados

Aplicações podem acessar *Linked Data* na web através de consultas a um *SPARQL endpoint* de um determinado conjunto de dados. Embora esse acesso possa prover dados va-

[46] http://www.openrdf.org/
[47] http://jena.apache.org/
[48] http://www4.wiwiss.fu-berlin.de/bizer/ng4j/
[49] http://www4.wiwiss.fu-berlin.de/bizer/ng4j/semwebclient/
[50] http://arc.semsol.org/

liosos para a aplicação, essa abordagem ignora o grande potencial da web de dados, pois não explora as possibilidades deste imenso espaço de dados que integra um grande número de conjuntos de dados interligados. Essas possibilidades podem ser alcançadas pela execução de consultas complexas e estruturadas sobre múltiplos conjuntos de dados. O trabalho desenvolvido em (HARTIG; LANGEGGER, 2010) discute diferentes abordagens para realizar essas consultas sobre a web de dados, classificando-as basicamente em dois tipos: tradicionais e inovadoras.

Abordagens Tradicionais

Data warehousing e federação de consultas são abordagens amplamente discutidas na literatura de banco de dados para realização de consultas sobre dados distribuídos em fontes autônomas. Consultas sobre a web de dados podem utilizar essas abordagens tradicionais que requerem o conhecimento prévio das fontes de dados relevantes e, portanto, limitam as fontes de dados que serão levadas em conta para obter as respostas de uma consulta. A seguir descreveremos a aplicação dessas abordagens sobre a web de dados.

Data warehousing usa uma base de dados centralizada que coleta e armazena os dados das fontes. No contexto de *Linked Data*, é possível materializar dados das fontes relevantes em uma base centralizada para a execução de consultas sobre ela. Tal estratégia também pode ser usada em mecanismos de busca sobre a web de dados. Além disso, ela possui o melhor desempenho dentre as abordagens que serão aqui discutidas, já que os dados podem ser acessados diretamente na base centralizada, sem a necessidade de comunicações adicionais através da rede. No entanto, em fontes de dados cujo volume de dados é muito grande, a materialização dos dados tende a requerer bastante tempo e espaço de armazenamento. Outro problema é que atualizações sobre as fontes não são imediatamente refletidas sobre o repositório central, podendo ocasionar consultas com resultados desatualizados em relação aos dados originais. Outra questão a ser considerada é que as consultas somente são realizadas sobre os dados materializados e não sobre toda a web de dados.

Federação de consultas baseia-se na distribuição do processamento de consultas para múltiplas fontes de dados autônomas. Seu objetivo é dar ao usuário acesso aos dados por meio de algum vocabulário padrão especificado em uma ontologia de domínio. Consultas podem ser formuladas baseadas nessa ontologia. Um mediador decompõe, de forma transparente, a consulta em subconsultas, direciona as subconsultas a múltiplos serviços de consulta distribuídos, e, finalmente, integra os resultados das subconsultas. Em mais detalhe, o processamento de uma consulta requer as seguintes tarefas: particionamento, adaptação, mapeamento, otimização e execução. A tarefa de adaptação consiste na modificação e extensão da consulta, por exemplo, através da inclusão de termos similares ou mais abrangentes, a partir de relacionamentos com outros vocabulários, expandindo assim o escopo do espaço de busca de forma a obter melhores resultados. A tarefa de mapeamento consiste na seleção conjuntos de *Linked Data* que têm potencial para retornar resultados para as expressões contidas na consulta. A tarefa de otimização avalia o custo de diferentes estratégias para processar a consulta, preparando um plano de execução para a consulta. Finalmente, a tarefa de execução implementa uma via de comunicação com os conjuntos de *Linked Data* e processa o plano de execução preparado pela tarefa de otimização, possivelmente adaptando-o dinamicamente. Uma vantagem da federação de consultas é que ela não requer tempo ou espaço adicional para materialização de dados. Por outro lado, a

execução de consultas é mais lenta devido às transmissões de rede necessárias para realização das subconsultas sobre as fontes de dados. Além disso, as consultas não podem ser realizadas sobre toda a web de dados, mas somente sobre as fontes de dados registradas no mediador. *DARQ* (QUILITZ; LESER, 2008) é um mediador baseado no processador de consultas *Jena ARQ* capaz de realizar consultas distribuídas sobre a web dados. *SemWIQ* (LANGEGGER, 2010) é outro mediador que estende o *Jena ARQ* a fim de consultar a web de dados fazendo uso de estatísticas (LANGEGGER; WOSS, 2009) para otimizar as consultas. FedX (SCHWARTE et al., 2011) é um mediador que estende o Framework Sesame com uma camada de federação que possibilita o processamento eficiente de consultas sobre fontes distribuídas de Linked Data. (VIDAL et al., 2011) apresentam um *framework* baseado em mediador de três níveis para integração de dados sobre *Linked Data*. Desafios relacionados à eficiência de consultas federadas e uma abordagem para otimização dessas consultas baseada em programação dinâmica foram tratados por (GöRLITZ; STAAB, 2011).

Abordagens Inovadoras

As abordagens inovadoras surgiram para eliminar a restrição imposta pelas abordagens tradicionais de limitarem as consultas sobre as fontes previamente conhecidas. Assim, elas permitem a descoberta das fontes durante a execução das consultas, podendo atuar sobre toda a web de dados. (HARTIG; LANGEGGER, 2010) caracterizam duas abordagens inovadoras: descoberta ativa baseada em federação de consultas e consultas exploratórias (também conhecidas como *link traversal*).

Descoberta ativa baseada em federação de consultas é uma estratégia baseada na combinação de processamento de consultas federado com uma descoberta ativa de fontes de dados relevantes, para possibilitar o uso de fontes de dados desconhecidas. Essa estratégia parece não ter sido implementada até o momento da publicação do presente trabalho, mas é uma estratégia que vale a pena ser objeto de investigações futuras, já que pode combinar as vantagens da federação de consultas com a possibilidade de obter dados de fontes ainda desconhecidas pelo mediador.

Consultas exploratórias *(link traversal)*. No enfoque exploratório, proposto por (HARTIG; BIZER; FREYTAG, 2009), dados são descobertos e recuperados em tempo de execução da consulta. Este enfoque é baseado na busca de URIs, onde uma consulta SPARQL é executada através de um processo interativo, onde URIs são dereferenciadas de modo a recuperar suas descrições em RDF na Web e os resultados da consulta construídos a partir dos dados recuperados. Desse modo, consultas exploratórias seguem *RDF links* para obter mais informações sobre os dados já existentes. Através do uso de dados recuperados a partir das URIs usadas em uma consulta como ponto de partida, o processador de consultas avalia partes da consulta. Soluções intermediárias resultantes dessa avaliação parcial geralmente contêm URIs adicionais que possuem ligações para outros dados que por sua vez, podem prover novas soluções intermediárias para a consulta. Para determinar o resultado completo da consulta, o processador de consultas avalia as partes da consulta e dereferencia URIs. O conjunto de dados usado na consulta é continuamente ampliado com dados potencialmente relevantes da web, cuja descoberta é realizada a partir das URIs de soluções intermediárias que podem estar em espaços de nomes distintos.

2.2.13 Desafios para integração de dados sobre Linked Data

A integração de dados de um grande número de fontes de dados heterogêneos na web ainda é complexa e ocorre no nível de vocabulário e identidade (links sameAs). No nível vocabulário, a integração deve ocorrer entre diversos vocabulários distintos. No nível de identidade é possível ter identificadores e conceitos distintos interligados através de links 'owl:sameAs' para um mesmo conceito do mundo real.

(HARTIG; LANGEGGER, 2010) afirmam a necessidade de tornar mais transparente a integração de dados entre múltiplas fontes. Isso requer mapeamentos entre termos de diferentes vocabulários usados por fontes de dados com conteúdos similares. Além disso, pode ser necessário aplicar técnicas de fusão de dados para obter uma representação consistente de dados descritos diferentemente em fontes distintas, bem como, ajudar a resolver questões relacionadas a conflitos e qualidade dos dados. Muito ainda precisa ser feito também em relação à inferência e descoberta de conhecimento em dados provenientes de múltiplas fontes.

Permitir o mapeamento dos diversos vocabulários existentes, para que seja possível identificar e escolher dados de fontes diferentes sobre uma mesma entidade também é uma questão que requer maior aprofundamento.

Criação, edição e manutenção de *Linked Data* por vários usuários são desafios. Outro desafio está relacionado à manutenção desses dados para evitar problemas de acesso a informações que não estejam mais disponíveis. A Web de Dados é dinâmica e deve permitir que aplicações possam fazer atualizações e utilizar técnicas avançadas para a detecção de inconsistências. A Web de Dados é alimentada com dados provenientes dos mais diversos domínios, causando problemas quanto à confiabilidade e qualidade daquilo que é disponibilizado.

As possibilidades criadas por esses dados integrados podem infringir os direitos de privacidade dos usuários. Proteger os direitos dos indivíduos se torna difícil, pois os dados estão em fontes descentralizadas e sob diversas jurisdições legais. Prover ferramentas para explicitar os direitos de cópia e reprodução sobre os dados é uma das lacunas no contexto de *Linked Data*.

Já existem várias aplicações funcionais e em desenvolvimento que permitem consultas complexas na Web de Dados, porém, ainda existem muitas oportunidades de pesquisa relacionadas à forma na qual os usuários poderão navegar por esses dados para tornar essa interação mais intuitiva, simples e objetiva.

Determinar as informações mais relevantes, assim como detectar sua validade para melhorar a qualidade da informação, também são desafios que precisam ser superados através de algum *feedback* do usuário ou mesmo de forma automatizada.

Encontrar *endpoints SPARQL* relevantes normalmente é uma tarefa complexa devido à falta de descrição conceitual das fontes de dados. Para simplificar essa tarefa, é possível adotar as estratégias abordadas na Seção 2.2.4. No entanto, muito ainda precisa ser realizado para reduzir ainda mais a complexidade da descoberta dessas fontes.

2.3 QEF

O QEF (*Query Execution Framework*) (PORTO et al., 2007) é uma ferramenta que provê um ambiente de consulta que auxilia o usuário na tarefa de definir e executar várias requisições de consultas. Um plano de execução de consulta (*QEP*) é uma representação das consultas baseada em operadores algébrico no formato de árvore. Nessa árvore, os nós são os operadores, as folhas são as fontes de dados e as arestas são os relacionamentos entre operadores na forma de produtor-consumidor. O *QEF* realiza consultas atráves de um *QEP* que determinará a sequencia de operações a ser executada. No *QEF*, um plano de execução é representado por um arquivo XML composto de uma lista de *templates* de operadores, onde cada operador é definido por uma identificador, um nome, uma lista de produtores e uma lista de parâmetros. A Figura 2.13 ilustra um exemplo de um operador que compõe um determinado *QEP*. Nesse exemplo, temos um operador denominado "Operador A", cujo id é igual a 1. Através dos *templates,* podemos construir uma arvore de operadores. A Figura 2.14 mostra um *QEP* formado por quatro operações (A,B,C e D). Cada operador consome uma tupla[51] e produz uma tupla modificada. Assim, durante a execução de um *QEP*, há um fluxo de tuplas de um operador para outro em uma árvore; isso é chamado de fluxo de dados (*dataflow)* do modelo produtor-consumidor. Os operadores em um *QEP* são organizados segundo uma topologia de árvore em que cada um dos pares de operadores subsequentes são ligados através de uma protocolo de comunicação seguido de um modelo iterativo, proposto por (GRAEFE, 1993).

```
<op:Operator id="1" prod="2">
        <Name>Operator A</Name>
        <ParameterList>
                <!—Operator's parameters -->
        </ParameterList>
</op:Operator>
```

Figura 2.13: Definição de um Operador em um QEP

Um QEP é representado por um arquivo XML composto de uma lista de modelos de operadores, onde cada operador é definido por um identificador (id), um nome, uma lista de produtores e uma lista de parâmetros (exemplificado pela Figura 2.15). Com esse modelo, podemos construir uma arvore de operações (Douglas Ericson and Fabio Porto, 2010).

Uma política de otimização de consultas comumente utilizada em Sistemas de Gerenciamento de Banco de Dados (SGBD) Relacionais consiste na utilização de informações históricas. Por meio dessas informações, pode-se estimar o custo de execução de cada operação que compõe um determinado plano de execução de consulta. Além disso, sabe-se que a qualidade de um plano de execução de consulta gerado pelo otimizador de consultas é tão boa quanto seu modelo de custo (BRUNO, 2011). Ou seja, dado um plano de execução de consultas inicial, o otimizador terá que escolher, dentre vários planos de execução equivalentes ao plano inicial, um determinado plano de execução que retorne os mesmos resultados do plano inicial, mas que apresente uma estimativa de custou e/ou tempo inferior. Atualmente, essa estratégia

[51] Como tradicionalmente adotada no modelo de dados relacional nós chamamos a estrutura de dados consumida e produzida pelos operadores como tupla

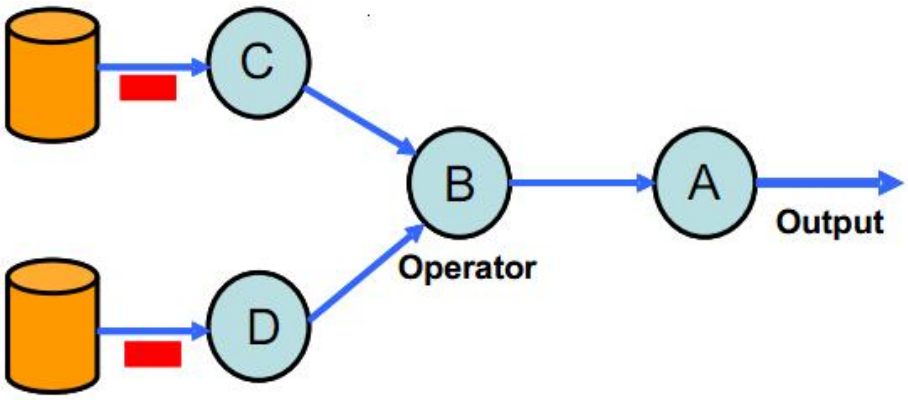

Figura 2.14: Exemplo de um QEP

não se mostra viável no contexto de *Linked Data*, uma vez que a maioria das fontes de dados não mantém informações históricas sobre a execução das consultas *SPARQL*.

2.4 Considerações finais

Este capítulo apresentou uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dissertação. Foram expostos os principais conceitos e ferramentas relacionados a integração de dados, Consultas Federadas, *Linked Data*, Junções sobre *Linked Data*, além de tratar também sobre a linguagem e a álgebra SPARQL.

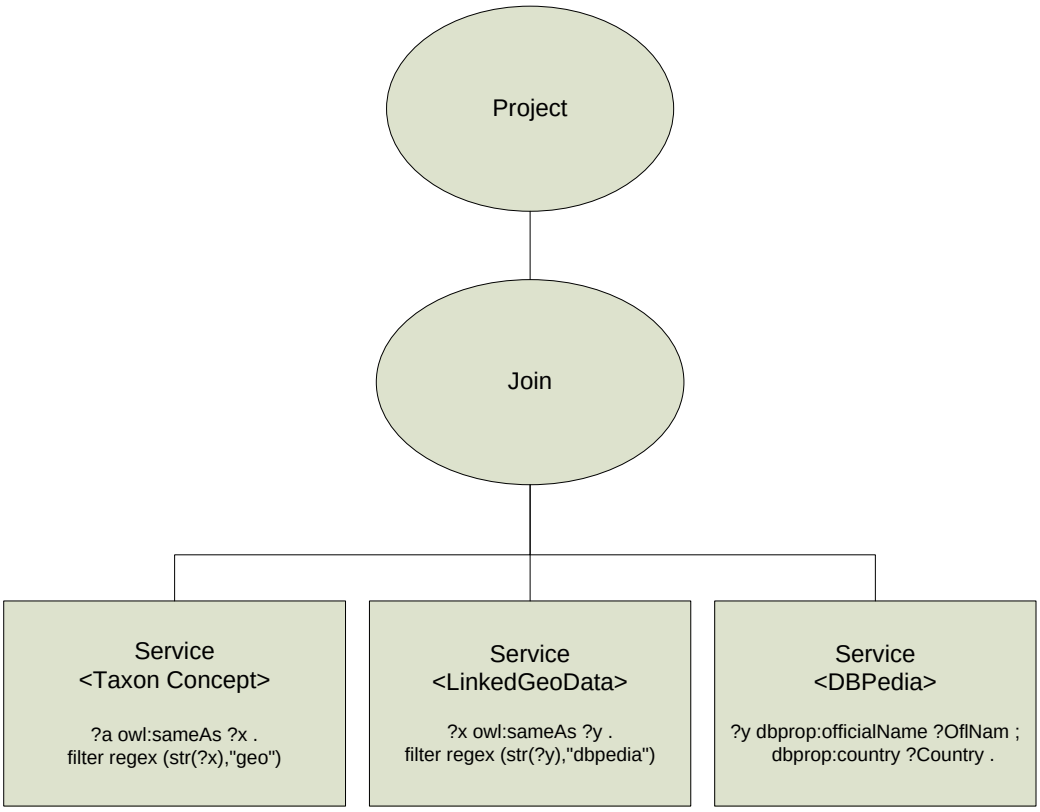

Figura 2.15: Exemplo de uma Consulta SPARQL e seu respectivo QEP

3 TRABALHOS RELACIONADOS

3.1 Introdução

Diversas soluções para processamento de junções de dados em consultas federadas sobre fontes de dados *RDF* heterogêneas têm sido propostas, tendo em vista sua relevância para integração de dados no contexto de *Linked Data* e o desafio de tornar o processo de integração de dados mais eficiente. As soluções existentes para processar consultas federadas em *Linked Data* utilizam diversas estratégias para melhorar o desempenho da execução das operações de junção. A seguir, iremos discutir as principais abordagens, encontradas na literatura, para o processamento de junções de dados em consultas federadas sobre *Linked Data*. Essas Soluções foram classificadas em dois grandes grupos: i) soluções que não utilizam estatística e ii) soluções baseadas em informações estatísticas sobre as fontes de dados *RDF* .

3.2 Soluções que não utilizam estatísticas sobre as bases de dados

Nessa seção, serão apresentadas as principais soluções para o processamento de junções em consultas federadas sobre *Linked Data* que não utilizam estatísticas sobre as fontes de dados.

3.2.1 Jena ARQ

O processador de consultas SPARQL *Jena ARQ*[1] é integrado ao framework de web semântica Jena[2]. Embora não seja uma solução abrangente de processamento distribuído de consultas, ele dá suporte a execução de consultas federadas, conforme o documento que estabelece um padrão oficial para federação de consultas em SPARQL 1.1 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). De acordo com esse padrão, o operador SERVICE permite a determinação da URI do SPARQL endpoint, bem como da consulta SPARQL que será executada naquele endpoint. No entanto, a especificação é bastante simples e não prevê otimizações ou outras estratégias para melhoria do desempenho das consultas. Apesar de ser uma referência dentre as tecnologias que envolve conceitos da web semântica, Jena ARQ é bastante simples e não adota nenhuma estratégia para otimizar ou adaptar a execução de consultas federadas sobre *Linked Data*. Jena ARQ é uma das estratégias de execução de consultas federadas usada em nossos experimentos, conforme detalhado no Capítulo 6.

3.2.2 Sesame

Sesame[3] é um *framework* Java de código aberto voltado para o armazenamento e recuperação de dados *RDF*. Além disso, o Sesame, assim como o Jena, disponibiliza um

[1] http://jena.apache.org/documentation/query/
[2] http://jena.apache.org/
[3] http://www.openrdf.org/

conjunto de bibliotecas que podem ser utilizadas para o desenvolvimento de ferramentas para consumir dados *RDF*. Nas versões anteriores, o Sesame suportava consultas SPARQL usando o sistema AliBaba [4] (até o ano de 2009). Nas versões anteriores, o Sesame não era capaz de executar consultas federadas no padrão SPARQL 1.1. Assim, como o Sesame executava somente consultas segundo o padrão SPARQL 1.0, os resultados vindos de várias fontes de dados eram integrados em um repositório único. A versão atual do Sesame segue a especificação SPARQL 1.1, fornecendo assim os recursos necessários para a execução de consultas federadas, incluindo o operador SERVICE. Assim como o SPARQL Jena ARQ, o Sesame é uma das ferramentas de execução de consultas federadas utilizadas nos nossos experimentos.

3.2.3 Fedx

FedX (SCHWARTE et al., 2011) é um framework para acesso transparente a fontes de dados através da federação de consultas. Ele oferece processamento de consultas federadas eficiente e usa os padrões e protocolos suportados pela maioria dos endpoints SPARQL disponíveis atualmente. O FedX estende o framework Sesame com uma camada dedicada à federação. A infraestrutura do Sesame permite que fontes de dados heterogêneas sejam usadas como endpoints no contexto da integração de dados. A Figura 3.1 apresenta a arquitetura de aplicação que faz uso do FedX.

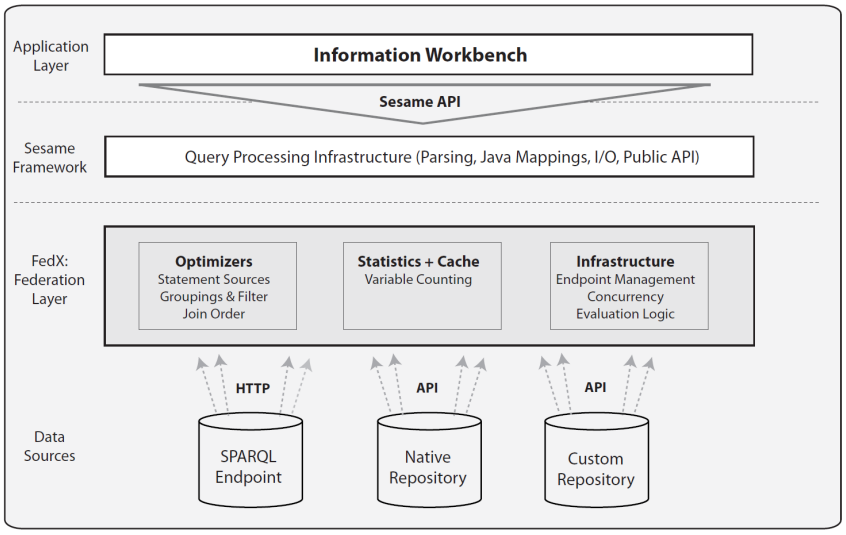

Figura 3.1: Arquitetura de uma aplicação usando FedX. Fonte: (SCHWARTE et al., 2011)

A camada de aplicação provê o *frontend* para o processador de consulta e é necessária para qualquer tipo de interação com a federação. A segunda camada é composta do Sesame e provê a infraestrutura básica para o processador de consultas, incluindo recursos de tradução de consultas, mapeamentos Java, componentes de entrada e saída, e a API para interação com o cliente. A camada de federação é implementada como uma extensão do Sesame e constitui o FedX, que adiciona as funcionalidades necessárias para gerenciamento de fontes de dados, comunicação com endpoints e, principalmente, otimizações para processamento de consultas

[4] http://www.openrdf.org/doc/alibaba/2.0-alpha2/alibaba-sail-federation/index.html

distribuído. Fontes de dados podem ser adicionadas na forma de implementação de repositórios do Sesame. Implementações padrões são providas para repositórios local, nativo (Sesame) e *endpoints* SPARQL remotos. Com isso, as seguintes federações são possíveis: puramente local consistindo de repositórios Sesame nativos, federações de *endpoints* ou formas híbridas.

O processamento de consultas no FedX consiste dos passos a seguir: uma consulta global é formulada sobre a federação de fontes de dados. Esta consulta é traduzida e otimizada para um plano de consulta federado, constituído de subconsultas que podem ser respondidas pelas fontes de dados individuais. Os resultados dessas consultas locais são integrados pelo federador e, finalmente, retornados. O processo é completamente transparente para o usuário, dando a impressão de que os dados estão virtualmente integrados em um único grafo RDF.

FedX vai além da federação de consultas definidas para SPARQL 1.1. Ele possibilita a configuração dinâmica de federações sobre fontes de dados distribuídas e executar consultas SPARQL transparentemente sobre as diferentes fontes de dados, mesmo sem o uso das extensões de federação, como a operação SERVICE.

O uso de técnicas de otimização são cruciais para o bom desempenho do processador de consultas. Especialmente no cenário distribuído, é essencial aplicar abordagens para reduzir o número de chamadas aos *endpoints* remotos. A combinação de uma otimização própria para ordenação de junções e agrupamento de subconsultas reduzem bastante o número de resultados intermediários e requisições, sendo as principais contribuições do FedX para a melhoria no desempenho de consultas federadas. O FedX implementa as seguintes técnicas de otimização:

- Fontes de sentenças RDF - examina fontes de sentenças RDF usando consultas ASK em SPARQL;
- Execução antecipada de filtros (o mais cedo possível)
- Processamento paralelo - uso de *threads* em operações de junção e união.
- Ordem de junções - reordenação de junções, técnicas de contagem de variáveis e heurísticas são usadas para estimar o custo de cada junção. Seguindo uma abordagem gulosa, as junções são executadas em ordem crescente de custo;
- Uso de *block nested loop joins*
- Agrupamento de sentenças RDF de uma mesma fonte de dados para execução em uma única consulta SPARQL a um endpoint.

Todas as funcionalidades do FedX são compatíveis com os *endpoints* compatíveis com SPARQL 1.0, sendo adequado aos ambientes atuais. O FedX não requer metadados preprocessados como estatísticas ou índices, o que o torna adequado para o processamento sob demanda de consultas *ad-hoc*.

3.2.4 CQELS

CQELS (*Continuous Query Evaluation over Linked Stream*) (LE-PHUOC et al., 2011), é um mecanismo de processamento de consultas sobre *Linked Data*. Ele define o seu próprio modelo de processamento, que é implementado no mecanismo de consulta. CQELS fornece uma estrutura de execução de consulta, com o processador de consultas que se adapta dinamicamente às mudanças sobre os dados de entrada. Durante a execução da consulta, o CQELS reordena continuamente os operadores, baseando-se na estimativa do tempo de resposta final e na complexidade da consulta para alcançar um melhor desempenho.

As consultas são realizadas usando a linguagem nativa do CQELS, que é uma extensão da linguagem declarativa SPARQL 1.1. Os dados obtidos são codificadas e o motor de busca utiliza algumas estratégias de acesso aos dados da memória, como *caching* e indexação.

3.3 Soluções que utilizam estatísticas sobre as fontes de dados RDF

Ao contrário das abordagens apresentadas anteriormente, algumas estratégias utilizam estatísticas sobre as fontes de dados. Essas informações podem ser obtidas, dinamicamente, em tempo de execução. Posteriormente, essas estatísticas podem ser armazenadas em um catálogo (metabase) ou mesmo em memória. Uma estratégia comumente utilizada para o armazenamento de informações estatísticas no cenário de *Linked Data* consiste em utilizar fontes de dados RDF e representá-las através do vocabulário VoID (ALEXANDER et al., 2009). VoID é um vocabulário que contém termos úteis para descrever dados estatísticos. Informações como, por exemplo, número de triplas e quantidade de predicados em uma fonte de dados são algumas das informações que podem ser representadas pelo VoID.

3.3.1 Distributed ARQ

O DARQ (*Distributed ARQ*)(QUILITZ; LESER, 2008) é uma extensão do *framework* ARQ que realiza consultas federadas sobre fontes de dados distribuídas através de um mediador de consultas. O DARQ provê uma descrição declarativa dos dados disponíveis em uma fonte de dados, a definição das limitações em padrões de acesso e informações estatísticas sobre os dados disponíveis, as quais são utilizadas para otimizar as consultas federadas. Duas estratégias de junção são implementadas pelo DARQ: *Nested Loop Join* (VALDURIEZ; GARDARIN, 1984) e *Bind Join* (FLORESCU et al., 1999).

O DARQ teve seu projeto cancelado no ano de 2008. Além disso, ele possui diversas limitações, como por exemplo:

- Uma consulta só funciona se esta envolver predicados que indiquem para onde devem ser enviadas as subconsultas baseadas em seu padrão de triplas. Consultas com padrões de tripla, contendo predicados em forma de variáveis, não são suportadas (por exemplo, ?s ?p ?o).

- Junções usando *Blank Nodes* não são suportadas. Se uma operação de junção encontrar um *Blank Nodes*, uma exceção será disparada;
- O processamento de consultas, sobre várias fontes de dados ou sobre fontes muito grandes (contém um número grande de triplas armazenadas) não possui um bom desempenho.

O principal objetivo do DARQ é possibilitar a federação de consultas SPARQL, fornecendo acesso transparente a múltiplos SPARQL *endpoints*. Resumidamente, o DARQ decompõe uma consulta SPARQL em subconsultas e as submete aos SPARQL *endpoints* correspondentes, sendo os resultados obtidos de cada subconsulta integrados no mediador. Do ponto de vista arquitetural, DARQ é um mediador similar à arquitetura definida em (WIEDERHOLD, 1992). Sua arquitetura é apresentada na Figura 3.2. No entanto, diferente de um sistema baseado em mediador, o DARQ não prevê a integração de esquemas.

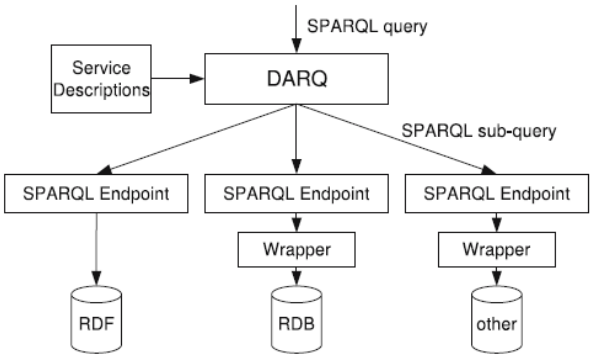

Figura 3.2: Arquitetura de integração de dados do mediador DARQ. Fonte: (QUILITZ; LESER, 2008)

Para usar o DARQ é necessário ter previamente armazenada a capacidade de cada fonte de dados em um arquivo chamado de descrição de serviço (*service descriptions*), um catálogo em RDF que descreve as fontes de dados. Essas capacidades são armazenadas localmente no mediador e descrevem os predicados presentes em cada fonte de dados com suas cardinalidades e também definem algumas informações de seletividade para expressões de filtro.

A reformulação da consulta é bem simples: a consulta principal é decomposta (particionada) em várias subconsultas de acordo com as informações obtidas na descrição de serviço. Cada uma delas deve ser respondida por um SPARQL *endpoint* conhecido.

Para otimização do processamento de consulta, o DARQ aplica otimização lógica e física. A otimização lógica usa regras para reescrever a consulta original antes da geração do plano; e a otimização física utiliza programação dinâmica interativa, empregando informações fornecidas pelo catálogo (*service description*) para definir a ordem de execução da junção.

O DARQ não possui um serviço para registro e monitoramento de fontes, dificultando o registro de novas fontes, bem como a obtenção de estatísticas atualizadas sobre elas. A configuração do DARQ requer que o usuário forneça explicitamente um arquivo de configuração contendo as descrições do serviço. Uma limitação do DARQ é que os predicados de

todas as consultas necessariamente devem estar definidos, não sendo possível o uso de variáveis SPARQL como predicado nas consultas.

O DARQ surgiu como uma prova de conceito no HP Labs em 2006. Seu autor diz que ele ainda está em estágio de desenvolvimento ainda muito inicial, não sendo indicado para uso em ambiente de produção. A última atualização do projeto ocorreu em 2008, e o autor afirma em seu blog pessoal [5] que não pretende continuar seu desenvolvimento.

3.3.2 ANAPSID

ANAPSID (ACOSTA et al., 2011) é um *framework* voltado para gerenciar a execução de consultas no que diz respeito à disponibilidade de dados e condição de execução. Utiliza o operador XJoin (URHAN; FRANKLIN, 2000) combinado ao *Symmetric Hash Join* (DESHPANDE; IVES; RAMAN, 2007) para integrar dados provenientes de várias fontes. O ANAPSID utiliza um catálogo para armazenar dados estatísticos. As informações de *timeout* da execução dos *endpoints* SPARQL são adicionadas ao catálogo de dados, que é atualizado em tempo real. Além disso, ANAPSID pode adaptar um o plano de consulta em tempo de execução.

3.3.3 ADERIS

ADERIS (*Adaptive Distributed Endpoint RDF Integration System*) (LYNDEN et al., 2010) é um processador de consultas adaptativo que integra dados RDF armazenados em diferentes fontes de dados e que suporta consultas SPARQL. O ADERIS baseia-se em informações sobre cada fonte de dados RDF para selecionar a ordem de execução das sub-consultas que compõem uma consulta federada. Para integrar os resultados obtidos sobre as fontes de dados SPARQL, é utilizado o operador *Nested-Loop Join*. Além disso, o ADERIS utiliza as informações das fontes de dados para verificar o estado de cada uma das fontes e, assim, mudar o plano de execução caso seja necessário.

3.3.4 GDS

Graph Distributed SPARQL (GDS) é uma ferramenta que estende o *framework* Jena, implementando uma árvore geradora mínima (*MST - Minimum Spanning Tree*) com o objetivo de melhorar a representação do BGP (*Basic Graph Pattern*) em uma consulta. O termo BGP está relacionado ao padrão de triplas que deve ser utilizado para encontrar um resultado para uma determinada consulta. Normalmente o BGP é o bloco delimitado pela cláusula WHERE de uma consulta SPARQL. Com base na descrição da consulta, grafos MST são gerados a partir da exploração realizada pelo algoritmo de Kruskal, que visa estimar o conjunto mínimo de padrões de triplas que devem ser usados e a ordem de execução das fontes que compõem uma consulta federada.

[5] http://blog.quilitz.de/2010/01/darq-federated-sparql-queries-status/

3.3.5 SemWIQ

O SemWIQ (LANGEGGER, 2010) é outro sistema de integração de dados em que as consultas são expressas em SPARQL. Também foi implementado usando o processador de consultas Jena ARQ. É baseado em uma arquitetura de mediadores-wrappers para geração do plano de execução, adotando uma estratégia própria de otimização.

O sistema foi desenvolvido com foco no compartilhamento de dados científico e faz parte de um projeto maior chamado *Semantic Data Access Middleware for Grids (GSDAM)*, para permitir que dados científicos fornecidos por diferentes grupos de pesquisa sejam acessados de maneira transparente e de forma compartilhada. No entanto, SemWIQ pode servir genericamente como sistema de integração sobre Linked Data.

A arquitetura foi desenvolvida considerando três princípios básicos: os dados podem ser estruturados (por exemplo, usando XML, RDF/RDFS, OWL), são geograficamente distribuídos e armazenados em formatos heterogêneos.

O funcionamento do SemWIQ é explicado resumidamente a seguir. (i) O cliente estabelece conexão com o mediador e submete uma consulta SPARQL ao esquema de mediação; (ii) o tradutor calcula um plano de execução que é modificado pelo otimizador; (iii) o otimizador analisa a consulta e procura no catálogo as fontes de dados relevantes para responder à consulta. A saída do otimizador é um plano global para a execução da consulta que é enviado ao engine de execução; (iv) o engine delega a execução das subconsultas aos SPARQL endpoints; (v) quando a fonte de dados não tem suporte nativo a consultas SPARQL é preciso um wrapper capaz de reescrever a consulta original, escrita em SPARQL, para o formato específico da fonte de dados consultada; (vi) o mediador provê um wrapper local quando a fonte de dados é acessada por um serviço Web para envio dos dados; (vii) o catálogo armazena descrições e estatísticas sobre as bases de dados registradas. Essas estatísticas são geradas pela ferramenta RDFStats (LANGEGGER; WOSS, 2009); (ix) finalmente, o componente de monitoramento atualiza as estatísticas sobre as fontes de dados registradas que enviam periodicamente consultas SPARQL aos endpoints a fim de gerar estatísticas.

A última atualização realizada no SemWIQ ocorreu em 2010 e há uma nota no site do projeto[6] afirmando que ele não é mais mantido.

3.3.6 Módulo de Execução de consultas (QEF-LD

O módulo *Query Plan Executor* é responsável pela execução de planos de consulta federados sobre a Web de Dados. Este componente também pode ser chamado de QEF-LD por se tratar de uma extensão do *QEF - Query Evaluation Framework* (PORTO et al., 2007) com suporte a *Linked Data*. O QEF foi estendido pelo QEF-LD para permitir a execução desse tipo de planos de consultas.

[6] http://semwiq.sourceforge.net/

3.4 Considerações Finais

Este capítulo apresentou as principais ferramentas existentes para lidar com a execução de junções em consultas federadas sobre *Linked Data*, destacando funcionalidades e desvantagens de cada uma delas. Pretende-se com esse estudo, obter subsídios para a formulação de contribuições relevantes ao contexto de execução de junções em consultas federadas. Essas contribuições serão apresentadas nos capítulos seguintes deste trabalho.

4 ASBJOIN - UMA ESTRATÉGIA ADAPTATIVA PARA A EXECUÇÃO DE JUNÇÕES EM CONSULTAS FEDERADAS SOBRE LINKED DATA

Nessa seção, apresentamos o ASBJoin, uma estratégia adaptativa para a execução de junções em consultas federadas sobre *Linked Data*.

4.1 Visão Geral

A solução proposta neste trabalho, denominada ASBJoin, consiste em uma estratégia adaptativa para a execução de junções, envolvendo até três fontes de dados, em consultas federadas sobre *Linked Data*. O ASBJoin baseia-se em informações estatísticas coletadas, dinamicamente, a partir das fontes de dados utilizadas em uma determinada operação de junção. Com base nessas informações, o ASBJoin pode detectar, por exemplo, instabilidades nas fontes de dados utilizadas durante a realização de uma determinada operação de junção j. Uma vez detectada estas instabilidades, o ASBJoin reage, podendo mudar a ordem com que as fontes de dados são acessadas para computar a junção.

A Figura 4.1 exibe uma consulta federada SPARQL Q. Já a Figura 4.2 ilustra um exemplo de um plano de execução de consultas P (QEP - Query Execution Plan) gerado para uma determinada consulta federada Q, a qual envolve uma operação de junção entre 3 fontes de dados *RDF*. Nesse exemplo, a operação de junção envolve três fontes de dados distintas: *TaxonConcept*, *GeoLinkedData* e *DBPedia*.

```
Select  * where{
SERVICE <http://lsd.taxonconcept.org/sparql>{
?a owl:sameAs ?x .
}
SERVICE <http://virtuoso.mooo.com/lgd-dbpedia/sparql>{
?x owl:sameAs ?y .
}
SERVICE <http://dbpedia.org/sparql>{
?y dbprop:officialName ?w ;
  dbprop:country ?u .
}
}
```

Figura 4.1: Consulta Federada SPARQL Q

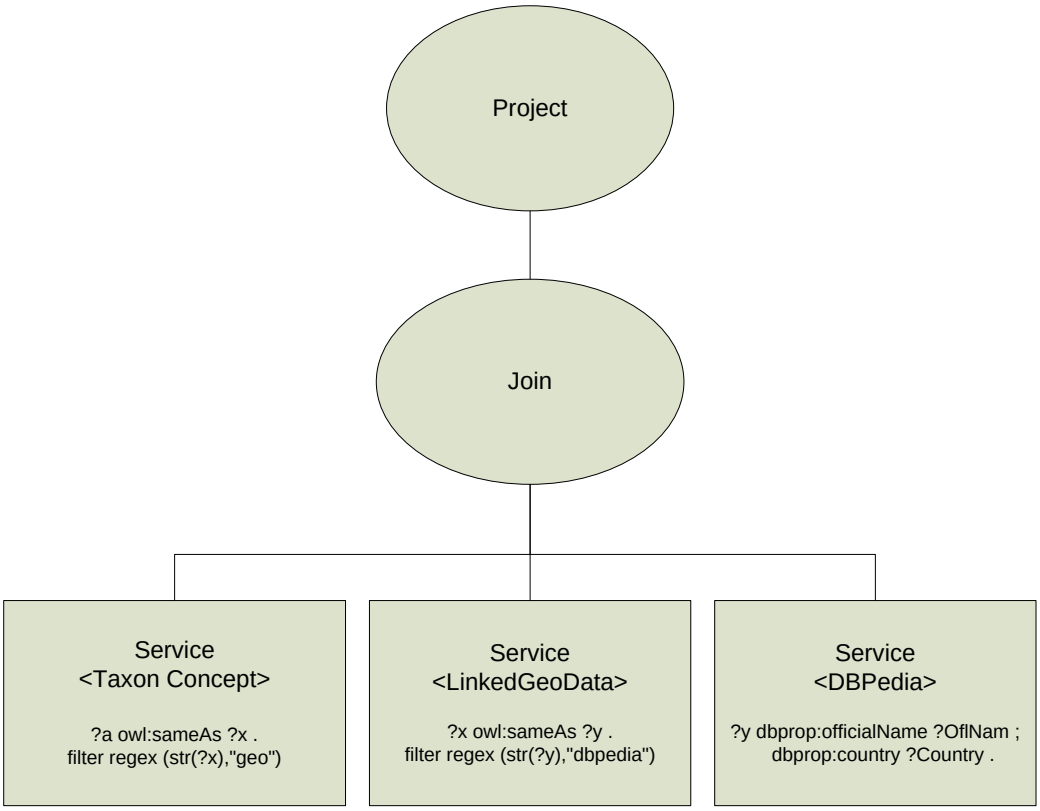

Figura 4.2: Exemplo de um Plano de Execução de Consulta P gerado para a Consulta Q

A estratégia ASBJoin, pode alterar, dinamicamente, a ordem com que as fontes de dados são acessadas durante a execução de uma operação de junção j que compõe um plano de execução de consultas (QEP) P. A Figura 4.3 ilustra as possíveis ordens de acesso às fontes de dados utilizadas em um operador de junção j que compõe um *QEP* P. Vale destacar que a ordem com que as fontes de dados são acessadas influencia diretamente o desempenho da operação de junção.

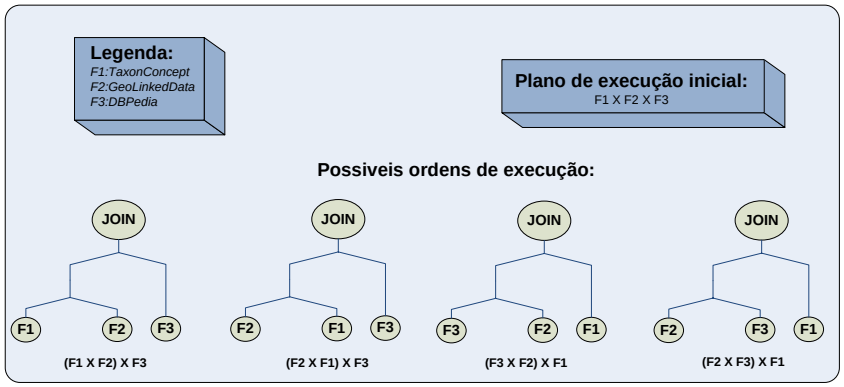

Figura 4.3: Possíveis Planos de Execução para a Consulta Q

Além disso, é comum que fontes de dados na Web tornem-se instáveis em determinados momentos. Neste caso, o tempo de acesso aos dados nessas fontes pode variar. Para

tomar decisões sobre a ordem em que as fontes são acessadas, o ASBJoin monitora informações atualizadas sobre as fontes de dados utilizadas na operação de junção. Essas informações são verificadas, periodicamente, durante a execução da operação de junção. Na seção 4.4 serão apresentados maiores detalhes sobre as estatísticas coletadas a partir das fontes de dados e como estas informações são utilizadas para definir, dinamicamente, a ordem de acesso às fontes de dados.

Nas seções seguintes, a estratégia ASBJoin será discutida em detalhes. Inicialmente, na seção 4.2, vamos discutir a arquitetura do ambiente de execução do ASBJoin. Em seguida, na seção 4.3, discutiremos o modelo de custo e as estatísticas utilizadas para verificar a melhor ordem de acesso às fontes de dados em um operador de junção j. Finalmente, a seção 4.4 conclui esse capítulo.

4.2 Arquitetura do Ambiente de Execução do ASBJoin

O ambiente de execução do ABSJoin foi concebido a partir do *QEF-LD* (MAGALHãES, 2012), uma extensão do *QEF* (PORTO et al., 2007) voltada para a execução de consultas federadas sobre fontes de dados *RDF*. O *QEF* é um *framework* que provê um ambiente para definição e execução de planos de execução de consultas(QEP) em um ambiente distribuído, a comunicação entre os componentes do *QEP* e o acesso a fontes de dados heterogêneas (MAGALHãES, 2012). Neste trabalho, estendemos o *QEF-LD*, implementando uma estratégia adaptativa para a execução de junções em consultas federadas, denominada ASBJoin, e um operador de junção adaptativo, denominado ASBJoin, o qual será descrito com mais detalhes no Capítulo 5.

A Figura 4.4 ilustra a arquitetura do ambiente de execução do ASBJoin, incluindo os seus principais componentes. A seguir, descrevemos em maiores detalhes cada um desses componentes:

Engine: O motor de execução de consultas (*Engine*) é constituído pelo Processador de Consultas e pelo Monitor. O motor de execução engloba todos os componentes responsáveis pelo processamento da operação de junção, bem como pelo processo de adaptação, o qual baseia-se nas informações estatísticas obtidas a partir das fontes de dados utilizadas na junção.

Parser: Obtem as informações contidas no plano de execução da consulta (*QEP*), o qual é representado por um arquivo *XML*.

Processador de Consultas: Responsável pela execução de um determinado plano de execução da consulta (*QEP*).

Monitor: Tem como objetivo coletar as informações estatísticas sobre cada uma das fontes de dados envolvidas em uma determinada operação de junção j. As informações obtidas pelo monitor são armazenadas no catálogo (metabase).

Catálogo: Banco de dados que contém informações estatísticas sobre as fontes de dados.

Fontes de Dados: São *triplestores* que armazenam dados no formato de triplas *RDF*.

Para atualizar as informações estatísticas sobre as fontes de dados, o monitor consulta, periodicamente, as fontes de dados e armazena as informações obtidas no catálogo. O processador de consultas e o monitor funcionam de maneira independente e em paralelo. Para verificar se houve mudanças no ambiente de execução, o processador de consultas utiliza as informações estatísticas sobre as fontes de dados utilizadas na operação de junção que estão armazenadas no catalogo. Com base nessas informações, o processador de consultas avalia a necessidade de adaptação, ou seja, verifica se é necessário alterar a ordem de acesso às fontes de dados que compõem uma determinada operação de junção j.

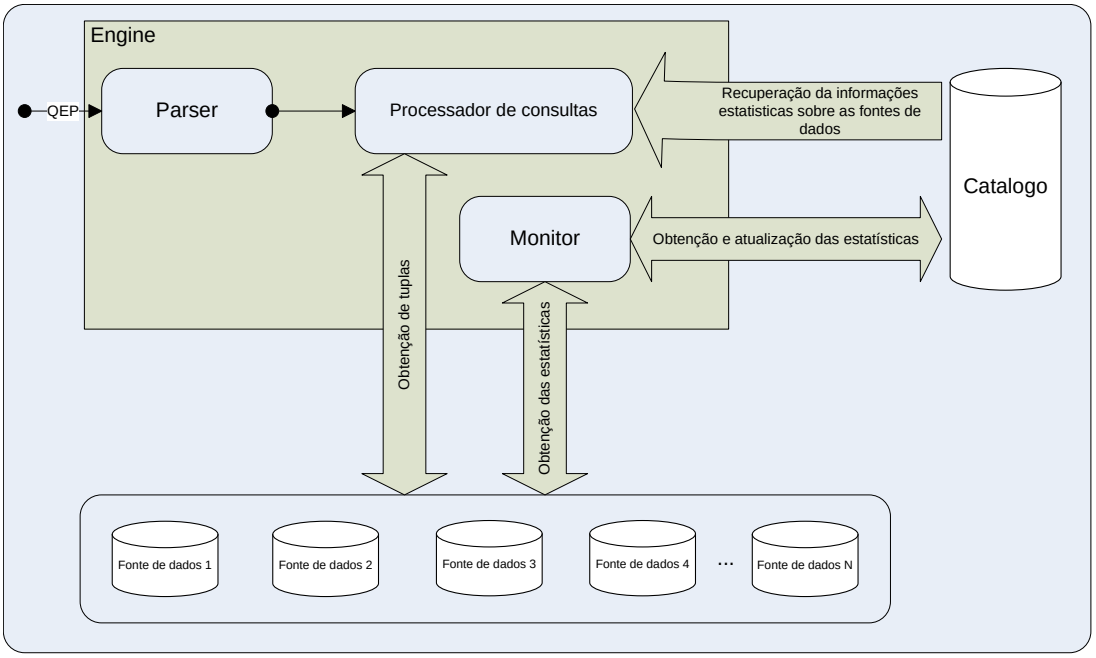

Figura 4.4: Arquitetura do Ambiente de Execução do ASBJoin

Na Figura 4.5 apresentamos um diagrama de estados que representa, de uma forma simples, os passos de execução de uma operação de junção j. A Figura representa os estados ou situações em que o processamento da junção pode se encontrar no decorrer de sua execução.

4.3 Estatisticas e Modelo de Custo

O ASBJoin seleciona dinamicamente a ordem em que as fontes são acessadas em uma operação de junção. Para isso, o ASBJoin calcula, para cada fonte de dados envolvida na operação de junção, uma métrica denominada FEP(Fator de Eficiência do Produtor). Essa métrica é calculada com base em 3 parâmetros: O tempo decorrido (*Elapsed Time*) (ET), o número total de tuplas (NTR) e o grau de confiança (GC) nas consultas realizadas em cada uma das fontes de dados.

O *Elapsed Time* (ET) é o tempo que o monitor leva para obter a primeira instância de

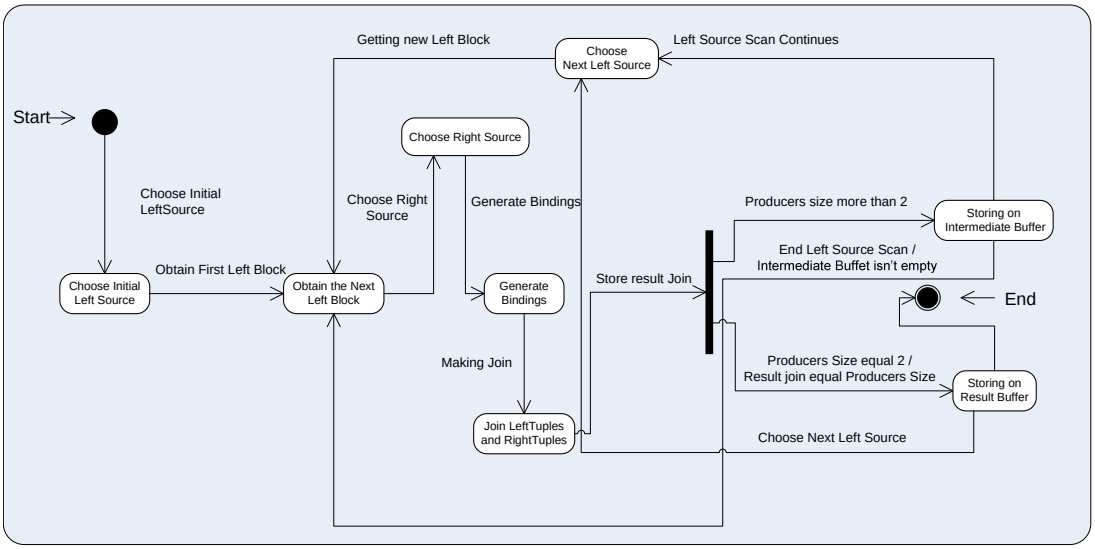

Figura 4.5: Diagrama de Estados da Execução de uma Operação de Junção usando a estratégia ASBJoin

uma consulta realizada sobre uma determinada fonte de dados. O número de tuplas recuperadas (NTR) em uma determinada fonte de dados refere-se à quantidade de instancias obtidas através da consulta sobre essa fonte de dados. E, finalmente, o grau de confiança (GC) corresponde a razão entre o número de vezes em que uma consulta a uma fonte de dados obtém alguma resposta e o número de vezes que essa fonte de dados é requisitada. A expressão para calcular o valor FEP de cada uma das fontes de dados durante a execução de uma consulta que envolve operações de junção é definida como:

$$FEP = (ET \,.\, GC) / NTR$$

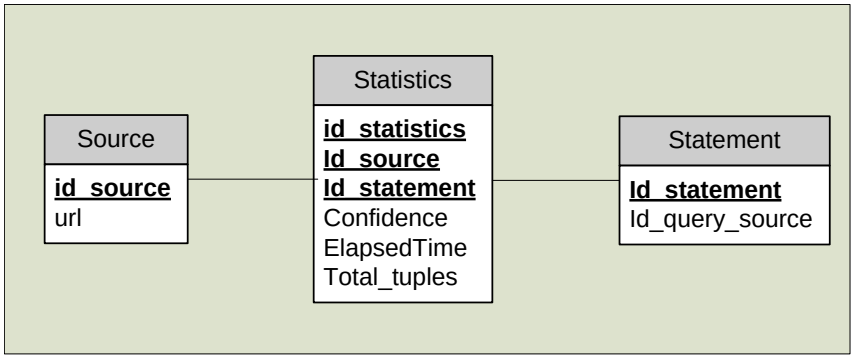

Figura 4.6: Diagrama Entidade-Relacionamento do Catálogo (Metabase)

Na figura 4.6, são representadas as informações sobre cada uma das fontes de dados acessadas, bem como sobre cada uma das consultas realizadas nessas fontes são armazenadas no Catálogo, mais precisamente nas tabelas *source* e *statement*, respectivamente. Cada uma das tuplas da tabela *source*, que representa as fontes acessadas, contém como atributos um id e a URL de acesso à fonte de dados. Na tabela *statement*, cada tupla representa uma consulta realizada sobre as fontes de dados. Esta tabela possui como atributos o id e o texto da consulta. A tabela *statistics* relaciona as fontes de dados e as consultas a elas submetidas. Na tabela

statistics são armazenas os parâmetros *confidence*, *elapsedTime* e *totalTuples* para cada fonte de dados.

Para definir a ordem de acesso às fontes de dados, que são produtoras da operação de junção, utilizamos o valor da métrica FEP para cada uma das fontes de dados utilizadas na operação de junção. No Capítulo 5, apresentaremos o operador ASBJoin, o qual irá definir a ordem de acesso às fontes, com base no valor da métrica FEP para cada fonte.

Algoritmo 1 : Monitor

```
1: if sourceExists(source) then
2:     currentSource ← getSource(source);
3: else
4:     newSource.setUrl(source);
5:     setSource(newSource);
6:     currentSource ← getSource(source);
7:     newStatistics.setIdSource(currentSource.getIdSource());
8: end if
9: if statementsExists(query) then
10:    currentStatements ← getStatements(query);
11: else
12:    newStatement.setIdSource(currentSource.getIdSource());
13:    newStatement.setQuerySource(query);
14:    setStatement(newStatement);
15:    currentStatements ← getStatements(query);
16:    newStatistics.setIdStatements(currentStatements.getIdStatements());
17: end if
18: if existStatistics(currentSource,currentStatistics) then
19:    newStatistics ← getStatistics(currentSource,currentStatements);
20:    elapsedTime ← getElapsedTimeJena(currentStatements.getQuerySource);
21:    if elapsedTime > −1 then
22:        confidence ← (newStatistics.getConfidence() + 1)/2;
23:    else
24:        confidence ← (newStatistics.getConfidence() − 1)/2;
25:    end if
26:    newStatistics.setElapsedTime(elapsedTime);
27:    newStatistics.setConfidence(confidence);
28:    updateStatistics(newStatistics)
29: else
30:    newStatistics.setIdSource(newSource.getIdSource());
31:    numTuples ← selectStatementJena(query,source);;
32:    newStatistics.setIdStatements(currentStatements.getIdStatements());
33:    newStatistics.setTotalTuples(numTuples);
34:    newStatistics.setElapsedTime(MaxValue);
35:    newStatistics.setConfidence(1.0);
36:    setStatistics(newStatistics);
37: end if
```

Para armazenar coletar e armazenar as informações estatísticas sobre as fontes de dados no catálogo, foi desenvolvido um algoritmo que realiza as operações do monitor. No Algoritmo 1 podemos observar o conjunto de passos realizados pelo monitor. Inicialmente, o Algoritmo 1 verifica a existência de informações estatísticas referente à uma determinada fonte de dados no catalogo. Da linha 1 até a linha 8 são verificadas as informações sobre uma determinada fonte de dados cuja a URL é expressa através de um valor fornecido como entrada. Da linha 9 até a linha 17, são verificadas as informações referentes à uma determinada consulta,

realizada sobre uma fonte de dados específica, por meio da tabela *Statements*. Da linha 18 até a linha 37 são analisadas as informações estatísticas sobre as fontes de dados na tabela *statistics* do catálogo.

4.4 Conclusão

Nesse capitulo, apresentamos a estratégia ASBJoin e discutimos as suas principais características. Foi descrita a visão geral do ambiente de execução concebido para implementar a estratégia ASBJoin. Em seguida foi apresentada a arquitetura do ambiente de execução do ASBJoin. Por fim, foi apresentado o m;odulo monitor e as informações estatísticas utilizadas para avaliar o estado das fontes de dados utilizadas em uma determinada operação de junção.

5 ASBJOIN - UM OPERADOR DE JUNÇÃO ADAPTATIVO PARA LINKED DATA

A principal contribuição deste trabalho é a implementação de um operador de junção adaptativo, denominado ASBJoin, capaz de executar junções entre até três fontes de dados distintas, o qual pode ser utilizado em consultas federadas sobre *Linked Data*. O ASBJoin foi inspirado no projeto Eddies (AVNUR; HELLERSTEIN, 2000). Entretanto, o ASBJoin é adaptado ao contexto de *Linked Data*. O ASBJoin foi implementado como um operador algébrico no QEF-LD(MAGALHãES, 2012). A principal característica do ASBJoin é a possibilidade de alterar, dinamicamente, a ordem com que as fontes de dados são acessadas durante a execução da operação de junção.

5.1 Principais Características

O operador de junção adaptativo ASBJoin foi implementado junto ao *QEF-LD*. No *QEF-LD* os planos de consulta são representados por um QEP – *Query Execution Plan* (Plano de Execução de Consulta) – e constituídos de operadores algébricos e operadores de controle (OLIVEIRA; PORTO, 2010). As operações de um QEP comunicam-se entre si para a obtenção de um resultado. Um QEP é representado como uma árvore, onde os nós são operadores, as folhas são fontes de dados (*data sources*) e as arestas são os relacionamentos entre operadores no modo produtor-consumidor. A estrutura de dados consumida e produzida pelos operadores é chamada de tupla. Os **operadores algébricos** implementam a álgebra de um determinado modelo de dados e agem sobre o conteúdo de uma tupla, realizando processamentos de acordo com uma semântica específica. Os **operadores de controle** são metaoperadores que implementam características de execução associadas ao fluxo de dados.

Os operadores seguem o modelo de iterador (GRAEFE, 1990), implementando as seguintes operações: *open*, *getNext* e *close*. A operação *open* prepara o operador para a produção de dados. Essa preparação consiste em executar a operação *open* de todos os produtores do operador e também em definir os metadados sobre as tuplas que serão produzidas pelo operador, através da operação *setMetadata*. O Algoritmo 2 apresenta a operação *open* usada pelo operador *Operator* do QEF. A operação *setMetadata* (Linha 7) recebe os metadados dos produtores do operador e a partir deles, define os metadados que serão usados para produzir as tuplas resultantes. Usualmente a operação *open* também inicializa algumas variáveis usadas pelo operador.

Algoritmo 2 : Operator – open

```
Input: producers
1: i ← 0;
2: for each producer in producers do
3:     producer.open();
4:     metadata[i] ← producer.getMetadata();
5:     i++;
6: end for
7: setMetadata(metadata);
```

A operação *getNext* produz uma tupla sob demanda do consumidor. As tuplas produzidas devem possuir as características definidas pelos metadados do operador. Neste capítulo serão apresentados os algoritmos da operação *getNext* dos operadores implementados como contribuições deste trabalho.

Finalmente, a operação *close* conclui a execução do operador. Isso ocorre ao se executar a operação *close* em todos os seus produtores, como pode ser observado no Algoritmo 3, usado no operador *Operator* do QEF. Também é comum a liberação de recursos na implementação de *close*.

Algoritmo 3 : Operator – close

```
Input: producers
1: i ← 0;
2: for each producer in producers do
3:     producer.close();
4: end for
```

A chamada do operador localizado na raiz da árvore é propagada aos operadores filhos até alcançar as folhas (fontes de dados). A operação *getNext* requisita a produção de uma tupla a ser consumida por um consumidor dentro da cadeia de execução. O resultado é uma execução *pipeline* de operadores sincronizados pela chamada da operação *getNext* entre pares de operadores produtor-consumidor. Esse modo de execução *pipeline* permite a produção de resultados assim que a primeira tupla chega ao operador raiz.

Cada plano de execução no *QEF* é definido por um *template* que é um arquivo XML composto de uma lista de operadores, onde cada operador é definido por um *id*, um nome, uma lista de produtores e uma lista de parâmetros.

O *QEF-LD* possui ainda alguns operadores que são utilizados pelo ASBJoin. A seguir, descrevemos esses operadores.

Operador *Service*

O operador *Service* é um operador de acesso a fontes de dados do tipo *endpoint SPARQL*. Ele recebe como parâmetros o nome da fonte de dados responsável pela execução da consulta (*DataSourceName*), a string da consulta SPARQL (*SPARQLQuery*) e a URI do *endpoint* SPARQL (*ServiceURI*) onde a consulta será executada. Os dois últimos parâmetros são repassados à fonte de dados definida no primeiro parâmetro. A string da consulta SPARQL pode conter parâmetros nomeados que serão substituídos na string por seus respectivos valores. Os nomes dos parâmetros são precedidos pelos caracteres '?:'. A Figura 5.1 apresenta um trecho de um *template* do QEF com um exemplo de definição de um operador *Service*, onde o parâmetro nomeado `?:dsname` será substituído pelo seu valor durante a execução da consulta.

```
<op:Operator id="1" prod="0" type="Scan" numberTuples="?">
<Name>Service</Name>
<ParameterList>
    <DataSourceName>SparqlEndpoint</DataSourceName>
    <ServiceURI>http://www4.wiwiss.fu-berlin.de/diseasome/sparql</ServiceURI>
    <SPARQLQuery>
    <![CDATA[
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX dsome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>

    SELECT ?ds ?dsn
    WHERE {
         ?ds dsome:name ?dsn .
         FILTER regex(?dsn, ?:dsname, "i")
    }
    ORDER BY ?dsn
    LIMIT 100
    ]]>
    </SPARQLQuery>
</ParameterList>
</op:Operator>
```

Figura 5.1: Definição de um operador Service em template do QEF

Operador *Project*
O operador *Project* é usado para definir as variáveis SPARQL cujos valores serão projetados nos resultados da consulta.

5.2 O Operador ASBJoin

O operador *BindJoin*, já implementado no *QEF-LD*, obtém o resultado da junção entre os seus produtores. Para cada tupla obtida do produtor esquerdo da junção, são recuperadas as tuplas do produtor direito, relacionadas à tupla obtida. O predicado de junção é dado pelas variáveis comuns entre os produtores do *join*. Os valores dessas variáveis comuns obtidos do produtor esquerdo do *join* são usados para fazer o (*bind*) das variáveis existentes na(s) consulta(s) realizadas pelo produtor direito do *join*. Depois disso, são retornadas as tuplas resultantes da junção entre a tupla obtida do produtor esquerdo e as tuplas do produtor direito do *join*. Os passos descritos são repetidos para cada uma das demais tuplas obtidas pelo produtor esquerdo da junção.

O operador *ASBJoin*, assim como o operador *BindJoin*, também obtém o resultado da junção entre os seus produtores. No entanto, ele difere do operador *BindJoin* por agrupar os resultados do produtor esquerdo em um conjunto (*set*) cujo tamanho pode ser configurado no plano de execução. Depois, são obtidas as tuplas do produtor direito do *join* que possuem relação com as tuplas do conjunto. Por fim, é realizada a junção entre as tuplas do conjunto e as tuplas obtidas do produtor direito do *join*, cujas variáveis comuns entre ambos os produtores do *join* possuem o mesmo valor. Além disso, pode-se habilitar o uso de *threads* para paralelizar a obtenção de resultados, permitindo que novos conjuntos do lado esquerdo sejam formados para obtenção de resultados, independente de conjuntos anteriores já terem sido completamente

processados para obtenção de resultados.

O funcionamento do operador *ASBJoin* pode ser dividido em duas etapas: i) etapa de pré-processamento e ii) etapa de processamento. Essas etapas serão detalhadas a seguir.

5.3 Etapa de Pré-Processamento

O Algoritmo 4 representa a etapa de pré-processamento do ASBJoin. Nessa etapa são verificadas, inicialmente, as informações estatísticas, armazenadas no catálogo, sobre as fontes de dados utilizadas na operação de junção. Em seguida, com base nas informações estatísticas obtidas, o operador ASBJoin define a ordem inicial de acesso às fontes de dados (produtores). Além disso, diversas variáveis globais são inicializadas. As variáveis booleanas *firstPass* e *endProcess* referem-se ao estado de execução da operação de junção. A variável *firstPass* refere-se ao processo de execução do método *firstPassProcess()* (algoritmo 8). O valor *true* indica que o processo de execução do primeiro primeiro passo da junção está ativo, ou seja, que ainda está executando a junção entre as duas primeiras fontes (e não a junção entre o resultado desta com a terceira fonte de dados). A variável *endProcess* indica se a execução da junção chegou ou não ao final. A variável *firstPass* é inicializada com o valor *true* e *endProcess* como valor *false*.

A variável *isSharedVariable* é utilizada para identificar os pares de produtores da junção que possuem variáveis comuns. A variável em comum determina as operações que possibilitam a ligação (*binding*) entre os produtores. A variável *producerMetadata* contém os metadados dos produtores. Desta forma, esta variável armazena informações sobre as consultas a serem realizadas em cada fonte de dados (produtor). O método *keyVariables* verifica a existência de variáveis comuns entre dois produtores, a partir de seus metadados.

5.4 Fase de Processamento

Depois que a etapa de pré-processamento é concluída e as variáveis globais inicializadas, o processo de execução da operação de junção é iniciado (Algoritmo 5).

O operador ASBJoin possibilita que os resultados da junção sejam obtidos a partir do processamento de conjuntos (*sets*) de tuplas gerados pelo produtor esquerdo da junção. O agrupamento das tuplas obtidas do produtor esquerdo da junção em conjuntos, permite a redução do número de consultas remotas a *endpoints* SPARQL relacionados ao produtor direito da junção. Além disso, também funciona como limitador do número de tuplas obtidas, visto que o *binding* das variáveis comuns usadas nos produtores ocasionará a formulação de uma consulta com uma seletividade menor e, portanto, mais restritiva.

O processamento de cada um desses conjuntos pode ser resumidamente dividido nas seguintes etapas:

(i) Formar um conjunto *S* de tuplas obtidas a partir do produtor esquerdo da junção.
(ii) Obter as tuplas do produtor direito da junção que se relacionam com as tuplas do conjunto

Algoritmo 4 : 3 – ASBJoin – open

Input: Um vetor denominado url que contém os endereços de acesso a cada fonte de dados e um vetor denominado query que contém as consultas realizadas sobre cada fonte de dados.

```
1: firstPass ← true;
2: endProcess ← false;
3: processStarted ← true
4: for i = 0 → totalProducers do
5:     allIdSources.add(i);
6:     sourceUri_i ← url_i;
7:     if existSource(URL_i) then
8:         source_i ← getSource(URL_i);
9:         if existStatement(URL_i) then
10:            statement_i ← getStatement(query_i);
11:        else
12:            addQuery(query_i);
13:        end if
14:        if existStatistics(url_i) then
15:            statistics_i ← getStatistics(url_i);
16:        else
17:            statistics_i.setIdSource(source_i.getIdSource());
18:            statistics_i.setTotalTuples(MAXVAL);
19:        end if
20:    else
21:        addSource(URL_i);
22:        if existStatement(URL_i) then
23:            statement_i ← getStatement(query_i);
24:        else
25:            addQuery(query_i);
26:            statement_i ← getStatement(query_i);
27:        end if
28:        statistics_i.setIdQuery(statement_i.getIdQuery(query_i));
29:        statistics_i.setIdSource(source_i.getIdSource(uri_i));
30:        statistics_i.setElapsedTime(MAXVAL);
31:        statistics_i.setConfidence(1.0);
32:        statistics_i.setTotalTuples(MAXVAL);
33:    end if
34: end for
35: for i = 0 → totalProducers − 1 do
36:    for j = i + 1 → totalProducers − 1 do
37:        keyColumns_ij ← JoinMetadata(ProducerMetadata_i, ProducerMetadata_j);
38:        keys_ij ← keyVariables(ProducerMetadata_i, ProducerMetadata_j);
39:        if keys_ij ≠ null then
40:            isSharedVariable_i.add(j);
41:            isSharedVariable_j.add(i);
42:        end if
43:    end for
44: end for
```

S.
(iii) Gerar as tuplas resultantes a partir da junção entre as tuplas do conjunto *S* e as tuplas obtidas do produtor direito.

As etapas citadas são detalhadas a seguir.

(i) Formar um conjunto *S* de tuplas obtidas a partir do produtor esquerdo da junção.

O operador ASBJoin (Algoritmos 5, 8 e 9) agrupa as tuplas obtidas do produtor esquerdo da junção, em conjuntos (*sets*). Os conjuntos possuem um tamanho máximo de tuplas previamente configurado no operador *ASBJoin* do plano de consulta.

(ii) Obter as tuplas do produtor direito da junção que se relacionam com as tuplas do conjunto *S*.
À medida que o conjunto *S* é preenchido com a quantidade de tuplas estipulada em sua configuração, o método *cloneOperator* é executado sobre o produtor direito da junção (Linha 1), para cloná-lo e alterá-lo de modo que a execução do método *getNext*, somente obtenha tuplas relacionadas às tuplas do conjunto *S*. Assim, todas as tuplas obtidas a partir do produtor direito da junção, efetivamente participarão da construção das tuplas resultantes. A alteração do produtor direito consiste em reformular as consultas existentes nele, de modo a fazerem a ligação (*binding*) dos valores das variáveis comuns entre os produtores da junção. Esse valores são extraídos das tuplas do conjunto *S*. A reformulação consiste em alterar as consultas existentes no produtor direito da junção para realizarem o *binding* dos valores das variáveis a partir do uso da operação UNION sobre a mesma estratégia de *binding* adotada no operador *ASBJoin*. A título de exemplo, suponhamos que o conjunto de tuplas obtido do produtor esquerdo do *join* possua as tuplas ilustradas da Figura 5.2 e que o produtor direito do *join* possua a consulta SPARQL mostrada na Figura 5.3.

```
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> ),
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83> ),
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11> ),
```

Figura 5.2: Exemplo de conjunto de tuplas geradas pelo produtor esquerdo do join

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT * WHERE {
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11>
    )
  }
}
```

Figura 5.3: Exemplo de conjunto de tuplas geradas pelo produtor esquerdo do join

A Figura 5.4 mostra mesma consulta após sua reformulação pelo método *cloneOperator*. Observa-se o uso de uma operação UNION para cada tupla proveniente do conjunto de resultados do produtor esquerdo da junção, exceto para a primeira tupla. Assim, para um conjunto de n tuplas, haverá $(n-1)$ operações UNION após a reformulação.

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT * WHERE {
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11>
    )
  }
}
```

Figura 5.4: Exemplo de consulta SPARQL do produtor direito do join reformulada

Todas as tuplas obtidas pelo produtor esquerdo da junção do conjunto *S* são armazenadas em uma tabela *Hash* (*hash table*) chamada *leftTupleHashTable* que possui como chave, uma representação dos valores das variáveis comuns entre os produtores da junção; e como valor, uma lista de tuplas que possuem aquela chave.

(iii) Gerar as tuplas resultantes a partir da junção entre as tuplas do conjunto *S* e as tuplas obtidas do produtor direito.
A partir da chave de cada tupla proveniente do produtor direito da junção, obtém-se da *leftTupleHashTable* a lista com todas as tuplas obtidas à esquerda que possuem a mesma chave. Depois disso, percorre-se essa lista para efetuar a junção de cada um de seus elementos com o elemento obtido à direita, produzindo-se, assim, as tuplas resultantes da operação.

As tuplas resultantes de todos os conjuntos processados em paralelo são armazenadas em uma única fila bloqueante ligada (*linked blocking queue*) chamada de *resultBuffer*. O método *take* da fila *resultBuffer* obtém e remove o elemento do início da fila, caso a fila não esteja vazia. Se a fila estiver vazia, o método *take* entra em estado de espera até que algum novo elemento seja adicionado. Para inserir um elemento no final da fila, utiliza-se o método *put*. O método *put* entra em estado de espera caso não haja espaço disponível para inserção de um novo elemento na fila. Assim que o espaço for disponibilizado, a fila sai do estado de espera e permite a inserção de novos elementos.

Vale destacar que o ASBJoin seleciona, de maneira dinâmica e autônoma, as fontes de dados que serão inicialmente utilizadas como o produtor da esquerda e o produtor da direita. Neste sentido, primeiramente o ASBJoin seleciona o produtor da esquerda (Linha 2 do Algoritmo 5), por meio do método *sourceSelectionCriterion()*, o qual irá retornar o identificador da fonte de dados com maior valor para a métrica FEP(definido no capitulo 4), o qual é armazenado na variável *betterSource*. Em seguida, o ASBJoin irá selecionar a fonte de dados que será utilizada como produtor da direita. a seleção do produtor a direita é definida através do método getRightSource(algoritmo 6). O valor do identificador da fonte de dados selecionada é armazenado na variável *rightBetterSource*(linha 5).

Após a escolha dos produtores da esquerda e da direita, o método *processBlockQuery()* (Algoritmo 7) é executado. A variável *instance*(Algoritmo 10) recebe as tuplas que compõem o resultado final da junção. O resultado da execução da junção é armazenado em uma estrutura de dados denominada *resultBuffer*.

Quando o Algoritmo 7 (*processBlockQuery()*) é chamado, os produtores da esquerda e da direita já foram definidos.Assim, o Algoritmo 7 funciona como método principal para a execução do operador de junção ASBJoin. Quando a operação de junção envolve apenas duas fontes de dados (F1 e F2, por exemplo) o ASBJoin será executado em apenas um passo (representado pelo método *firstPassProcess()*). Contudo, quando a operação de junção envolve três fontes de dados (F1, F2 e F3, por exemplo) o ASBJoin será executado em dois passos (representados pelos métodos *firstPassProcess()* e *secondPass()*). O método *firstPassProcess()* é responsável por montar os blocos do produtor da direita (F1, por exemplo) que serão utilizados para fazer a ligação (*binding*) com o produtor da direita (F2, por exemplo). Já o método *firstPassProcess()* é responsável por montar os blocos resultantes da junção entre dois produto-

Algoritmo 5 : 3 – ASBJoin – getNext

Input: As listas denominadas source, statistics e statement **Output:** uma instancia do resultado da consulta.

```
 1: if processStarted ≠ FALSE then
 2:     betterSource ← sourceSelectionCriterion(source, statistics, statement);
 3:     count = 0;
 4:     if isSharedVariable[betterSource].size() > 1 then
 5:         rightBetterSource ← getRightSource(betterSource);
 6:     else
 7:         if isSharedVariable[betterSource].size() = 1 then
 8:             rightBetterSource ← isSharedVariable[betterSource].get(0);
 9:         end if
10:     end if
11:     processStarted ← true;
12:     processBlockQuery();
13:     instance ← resultBuffer.take;
14:     if instance = TOKENNULL then
15:         instance ← null;
16:     else
17:         produced++;
18:     end if
19:     return instance;
20: end if
```

Algoritmo 6 : 3 – ASBJoin – getRightSource

Input: A variável betterSource. **Output:** A variável rightBetterSource.

```
 1: for i = 0 → numSource do
 2:     if i ∈ isSharedVariable[betterSource] then
 3:         sourceAux[count] ← source_i;
 4:         statisticsAux[count] ← statistics_i;
 5:         statementAux[count] ← statement_i;
 6:         count++;
 7:     end if
 8: end for
 9: rightBetterSource ← sourceSelectionCriterion(sourceAux, statisticsAux, statementAux);
10: return rightBetterSource;
```

res (F1 e F2, por exemplo) que serão utilizados para fazer a ligação (*binding*) com o terceiro produtor (F3, por exemplo). A junção entre um bloco produzido pelo produtor da esquerda com os dados do produtor da direita é realizada pelo método *fillResultBuffer()*(Algoritmo 10).

Algoritmo 7 : 3 – ASBJoin – processBlockQuery()

```
1: while endprocess = false do
2:     if firstPass = true then
3:         firstPassProcess();
4:     end if
5:     if firstPass = false and IntermediateBuffer ≠ empty then
6:         secondPass();
7:     end if
8:     fillResultBuffer();
9: end while
```

O Algoritmo 8 descreve o método *firstPassProcess()*, primeiro passo para processar uma operação de junção com até três fontes. O método é concluído quando um bloco de tuplas, obtidas do produtor da esquerda, é preenchido. Os blocos de tuplas possuem tamanho pré-definido (número máximo de tuplas em um bloco). O tamanho dos blocos é um parâmetro do operador ASBJoin. No algoritmo 8, um bloco de tuplas é representado pela lista *leftTupleMap*. Esses blocos de tuplas (produzidas pelo produtor da esquerda) serão utilizados para a reformulação da consulta realizada no produtor da direita , a fim de se obter tuplas correspondentes.

Adicionalmente, o Algoritmo 8 é responsável pelo processo de adaptação do ASBJoin, ou seja, pela alteração dinâmica da ordem de acesso às fontes de dados. Caso ocorra uma adaptação (ou seja, um novo produtor da esquerda seja selecionado), torna-se necessário verificar os casos de valores duplicados através do método changeSourceVerify(linha 31). As tuplas recuperadas anteriormente, nos antigos produtores a esquerda, são armazenadas em uma lista denominada *instanceObtained*. Nesta lista serão armazenadas as tuplas já recuperadas, com a finalidade de evitar tuplas duplicadas no resultado final. Além de verificar os casos de valores duplicados, o método changeSourceVerify verifica se a necessidade de adaptação, ou seja, da alteração do produtor da esquerda (no caso de uma junção entre duas fontes e dados) ou da alteração tanto do produtor da esquerda quanto do produtor da direita (no caso de uma junção envolvendo três fontes de dados). Na linha 29 é realizada a escolha da nova fonte por meio do método *getBetterSource(linha 29)*, o qual elege o identificador do novo produtor da esquerda (caso tenha ocorrido alguma instabilidade nas fontes de dados). Essa verificação é realizada sempre que um novo bloco do produtor da esquerda é produzido (completamente preenchido, por exemplo).

A lista *leftTupleMap*(linha 16) é uma lista global que é utilizada no algoritmo 10, o qual os elementos contidos nela são parâmetros necessários para a reformulação da consulta no produtor à direita.

O Algoritmo 9, descreve o método *secondPass* do ASBJoin. Esse método é utilizado somente quando o ASBJoin envolve três fontes de dados. Neste caso, o método *secondPass* é utilizado para construir blocos de tuplas resultantes da junção entre dois produtores (tuplas geradas pelos métodos *firstPassProcess()* e *fillResultsBuffer* e armazenadas em uma estrutura de dados global denominada *intermediateBuffer*), a fim de que este bloco seja utilizado

Algoritmo 8 : 3 − ASBJoin − firstPassProcess()

```
1: leftTuple ← getProducer(betterSource).getNext();
2: while leftTuple ≠ null do
3:     numberOfLeftTuples ← 0;
4:     while true do
5:         if leftTuple ≠ null then
6:             firstPass ← false;
7:         end if
8:         if numberOfLeftTuples >= this.leftTuplesSetSize and leftTuple ≠ null then
9:             break;
10:        end if
11:        if !instancesObtained[rightBetterSource].containsKey(keyObtained) then
12:            numberOfLeftTuples++;
13:        end if
14:        key ← getKey(leftTuple);
15:        if leftList = null then
16:            leftTuplesMap.put(key,leftList);
17:        end if
18:        leftList.add(leftTuple);
19:        if numberOfLeftTuples = this.leftTuplesSetSize − 1) then
20:            for i = 0 → numProducers do
21:                if ed.existSource(sourceUri[i]) then
22:                    source[i] = ed.getSource(sourceUri[i]);
23:                    if ed.existStatistics(sourceUri[i]) then
24:                        statistics[i] = ed.getStatistics(sourceUri[i]);
25:                    end if
26:                end if
27:            end for
28:            oldSource = betterSource;
29:            betterSource = getBetterSource(source,statitics,statements);
30:        end if
31:        rightSource ← getRightSource(betterSource)
32:        changeSourceVerify(betterSource,oldSource);
33:    end while
34: end while
```

para se realizar a junção com o terceiro produtor.

Na linha 1 do Algoritmo 9, é obtido o primeiro elemento da estrutura de dados *intermediateBuffer*. Cada elemento obtido de *intermediateBuffer* é armazenado em um bloco de tuplas que é representado pela estrutura de dados denominada *leftTupleMap*. A variável booleana *boolChange* (linha 3) indica se as tuplas obtidas de *intermediateBuffer* são resultantes da junção entre as mesmas duas fontes. Supondo que as três primeiras tuplas obtidas de *intermediateBuffer* tenham sido geradas a partir da junção entre as fontes F1 e F2, então podemos formar um bloco de tuplas com essas três tuplas. Agora suponha que a quarta tupla tenha sido resultante da junção entre as fontes F2 e F3. Neste caso, deve-se evitar que tuplas formadas a partir de diferentes fontes sejam parte do mesmo bloco de tuplas. Para isso, a variável *boolChange* recebe o valor *true* (condição que pode ser verificada da linha 24 até a linha 30). Assim, um bloco é montado com as três primeiras tuplas, deixando a quarta tupla para ser armazenada no bloco seguinte . A variável *key* (linha 13) recebe o valor da variável comum da tupla obtida em *intermediateBuffer* com o produtor da direita.

Algoritmo 9 : $3-\text{ASBJoin}-\text{secondPass}()$

```
 1: leftIntermediateTuple ← intermediateBuffer.take();
 2: actualLeftTuple ← leftIntermediateTuple;
 3: boolChange ← false;
 4: while leftIntermediateTuple.getTuple() ≠ null do
 5:     numberOfLeftTuples ← 0;
 6:     while true do
 7:         if leftIntermediateTuple.getTuple() = null then
 8:             endProcess ← true;
 9:         end if
10:         if boolChange = true or numberOfLeftTuples = leftTuplesSetSize then
11:             boolchange = false;
12:             break;
13:         end if
14:         key ← getKey(leftIntermediateTuple.getTuple());
15:         leftList ← leftTupleMap.get(key);
16:         if leftIntermediateTuple.getTuple() ≠ null then
17:             leftTupleMap.put(key,leftIntermediateList);
18:         end if
19:         rightSource ← chooseSource(leftTuple)
20:         if !intermediateBuffer.isEmpty() then
21:             leftIntermediateTuple ← intermediateBuffer.take();
22:         end if
23:         numberOfLeftTuples++;
24:         if leftIntermediateBuffer.tuple() = null then
25:             boolChange = true;
26:         else
27:             if actualLeftTuple.getSources() ≠ leftIntermediateBuffer.getSource() then
28:                 boolChange = true;
29:             end if
30:         end if
31:         actualLeftTuple ← leftIntermediateTuple;
32:     end while
33: end while
```

O Algoritmo 10 descreve a função *fillResultBuffer*. Esse algoritmo é responsável pela reformulação da consulta a ser executada no da direita. Adicionalmente, o Algoritmo 10

é responsável pela junção dos dados, bem como pela atualização das variáveis *resultBuffer* e *intermediateBuffer*, as quais armazenam os resultados da operação de junção.

Na estrutura de dados *rightProducer* (linha 1) são armazenadas as tuplas geradas pelo método *getProducer()* através da reformulação da consulta no produtor da direita. A reformulação é realizada a partir do bloco de tuplas vindo do produtor da esquerda (representado pela estrutura de dados *leftTuplesMap*).

A variável *tuple* recebe as tuplas geradas a partir da junção realizada sobre os produtores pelo método *join* (linha 14).

A variável *rightTuple* recebe as tuplas obtidas do produtor da direita. A estrutura *sourcesQuery* armazena as informações sobre as consultas realizadas pelos produtores do ASB-Join. Essa estrutura é utilizada para informar o número de produtores do ASBJoin e, assim, saber em qual *buffer* a nova tupla gerada deve ser armazenada (condição prevista da linha 15 até a linha 23).

Algoritmo 10 : $3 - \text{ASBJoin} - \text{fillResultBuffer}()$

```
 1: rightProducer ← getProducer(rightBetterSource, leftTuplesMap); ;
 2: rightTuple ← rightProducer.getNext();
 3: while rightTuple ≠ null do
 4:     key ← getKey(rightTuple, getRightSharedVarsPositions());
 5:     leftInstancesList ← leftInstancesMap.get(key);
 6:     while leftInstanceList.hasNext() do
 7:         leftTuple ← leftIterator.next();
 8:         tuple = join(leftTuple, rightTuple);
 9:         if hasJoinCount = totalProducers − 1 then
10:             getResultBuffer().add(tuple);
11:         else
12:             IntermediateResultBuffer().add(tuple);
13:         end if
14:     end while
15:     rightTuple ← rightProducer.getNext();
16: end while
```

A Figura 5.5 ilustra um exemplo de uma operação de junção realizada pelo ASB-Join. Considere a consulta *SPARQL* Q21, ilustrada na Figura 6.3, a qual está relacionada aos domínios de drogas e doenças. Suponha que a junção entre os padrões de triplas utilizados nas subconsultas enviadas para as fontes S1 e S2 foi executada por completo e que o tamanho do bloco seja 3.

Na Figura 5.5, o item (1) representa as três fontes de dados envolvidas na operação de junção. Considere que a fonte S1 (*Diseasome*) tenha sido eleita como produtor à esquerda. Em seguida, uma operação de *Scan* é realizada sobre a fonte S1 e as tuplas agrupadas em blocos de tamanho 3. O item (2) ilustra esses blocos. O item (3) representa cada bloco construído a partir das tuplas de S1 sob a forma da condição de *Union*. No passo (4) é realizada a etapa de reformulação da consulta na fonte S2. Em (5) é representado o resultado da junção entre S1 e S2.

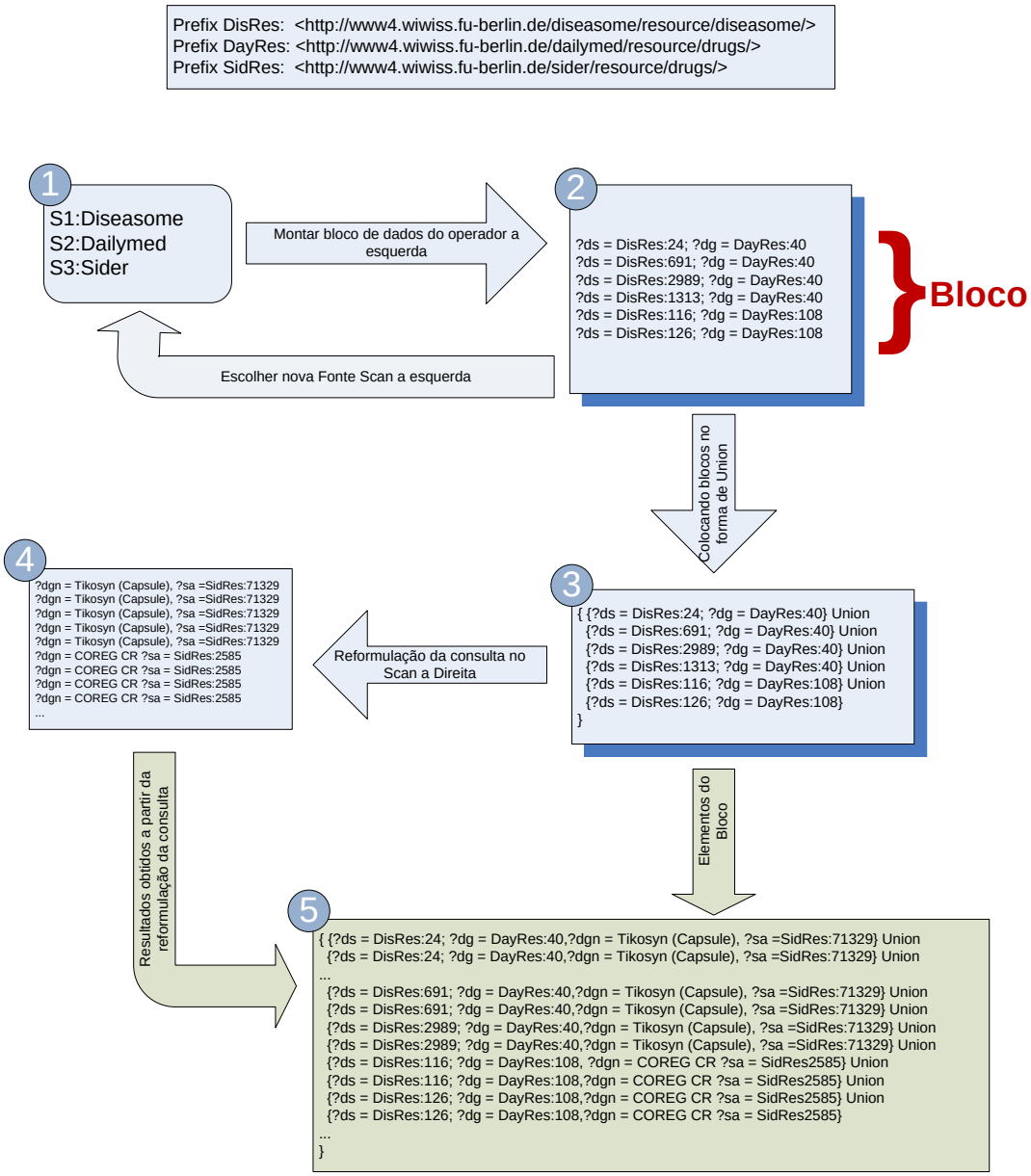

Figura 5.5: Execução do Operador ASBJoin sobre 3 Fontes de Dados (Fase 1)

Depois que todas as tuplas de um dos produtores do ASBJoin são obtidas, inicia-se a Fase 2, descrita na Figura 5.6. Nessa segunda fase, o objetivo é gerar o resultado final a partir dos dados intermediários obtidos na Fase 1. Para isso, será realizada a etapa de junção dos dados intermediários com o terceiro produtor. Em seguida, o resultado final da junção é retornado. Da mesma maneira como foi realizada na Fase 1, as instancias contidas nos resultados intermediários farão o papel de "produtor"a esquerda, enquanto a fonte de dados restante será o produtor da direita. Nesse caso, blocos são criados a partir das instancias resultantes da junção entre S1 e S2, por exemplo. Com isso, a reformulação da consulta é realizada sobre S3.

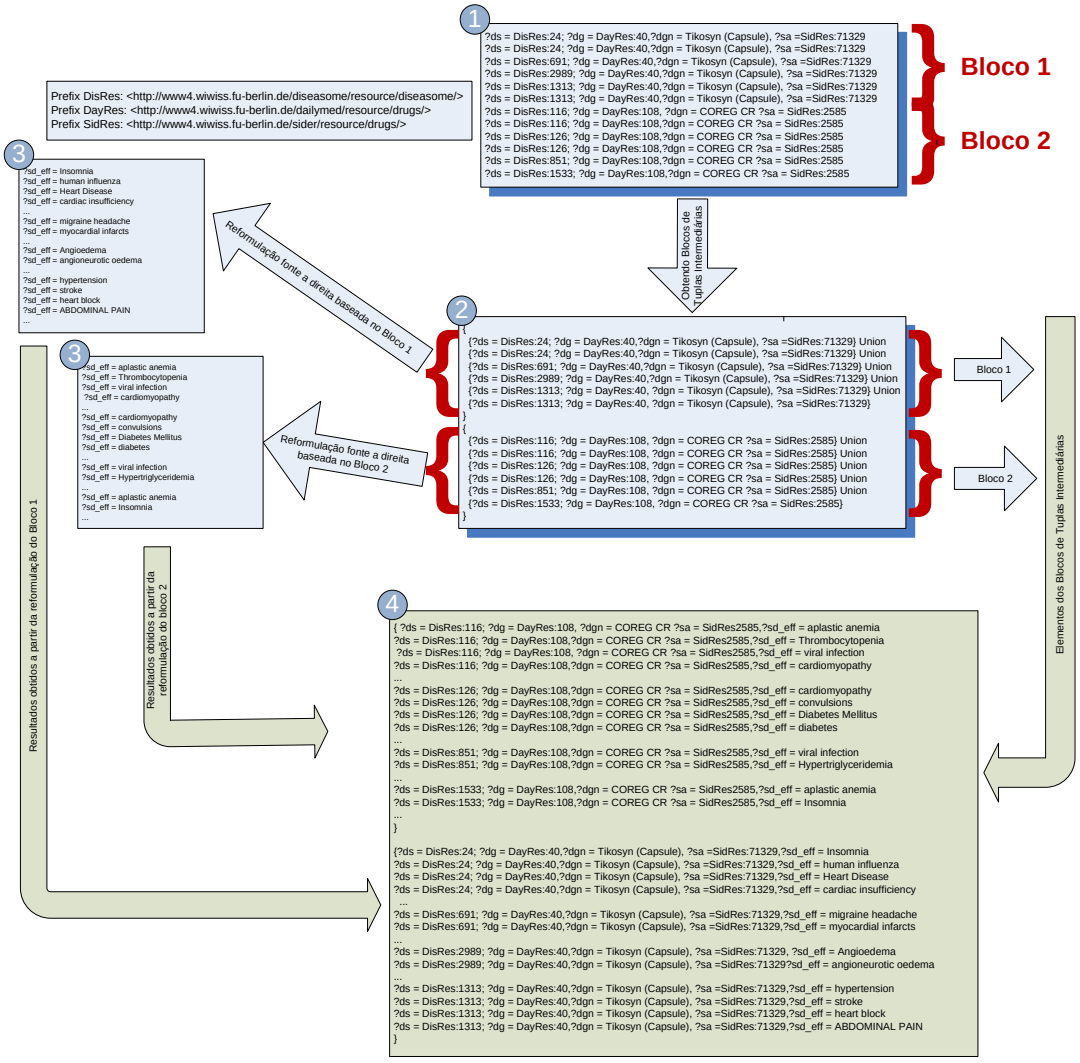

Figura 5.6: Exemplo do Operador ASBJoin sobre 3 Fontes de Dados (Fase 2)

5.5 Considerações Finais

Nesse capitulo, apresentamos os algoritmos que foram utilizados para implementar o operador de junção adaptativo ASBJoin. Adicionalmente, um exemplo para ilustrar o funcionamento do ASBJoin foi discutido.

6 EXPERIMENTOS E RESULTADOS

Essa seção descreve os experimentos realizados com a finalidade de avaliar a estratégia proposta para execução de junções adaptativas em consultas federadas sobre *Linked Data*. Adicionalmente, os resultados obtidos são discutidos e analisados.

Os testes foram executados utilizando-se duas cargas de trabalho distintas. Uma carga de trabalho consiste de um conjunto de consultas *SPARQL* federadas. As ferramentas Jena, Sesame e FedX foram avaliadas e comparadas com a estratégia proposta nesta dissertação, denominada ASBJoin. Vale destacar que as ferramentas Jena, Sesame e FedX executam as consultas *SPARQL* federadas diretamente. Já no caso do ASBJoin, como este foi implementado junto ao *QEF-LD*, foi necessário converter cada consulta *SPARQL* federada em um plano de consulta correspondente compatível com o *QEF-LD*.

6.1 Cargas de Trabalho

As cargas de trabalho utilizadas nos experimentos basearam-se nas consultas pertencentes ao Benchmark FedBench [1], que contém um conjunto de consultas para a análise da eficiência de estratégias de processamento de consultas federadas sobre dados que seguem as boas práticas propostas pelo *Linked Data* (HAASE; MATHäSS; ZILLER, 2010).

As consultas que compõem as cargas de trabalho utilizam fontes de dados que se enquadram em dois dos domínios de dados mais interessantes da Web: *Cross Domain* e Ciências da Vida (*Life Science*). A razão para a escolha desses domínios reside na natureza das consultas, na preferência por parte da maioria dos usuários de *Linked Data* e na importância em relação às informações que podemos obter nesses tipos de fontes.

A primeira carga de trabalho, denominada C1, é constituída por seis consultas (pertencentes ao *FedBench*), as quais envolvem operações de junção sobre duas fontes de dados. Essas consultas foram nomeadas Q1i, onde i = 1..6.

A segunda carga de trabalho, denominada C2, é constituída por quatro consultas sintéticas (baseadas nas consultas do *Benchmark FedBench*), as quais envolvem operações de junção sobre três fontes de dados. Essas consultas foram nomeadas Q2i, onde i = 1..4.

As consultas que compõem a carga de trabalho C1 podem ser visualizadas nas figuras 6.1 e 6.2. Já as consultas que compõem a carga de trabalho C2 são ilustradas na Figura 6.3.

[1] Página do projeto do Fedbench: http://code.google.com/p/fbench/

Q_{11} : Recuperar o dados sobre URL, partido Politico e pagina na Web de Presidentes na DBPedia e NYTimes.

```
SELECT ?president ?party ?page WHERE {
  service<http://dbpedia.org/sparql>{
  select * where{
  ?president <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/President> .
  ?president <http://dbpedia.org/ontology/party> ?party .
 }
}
  service<http://virtuoso.mooo.com/nytimes/sparql>{
  ?x <http://data.nytimes.com/elements/topicPage> ?page .
  ?x <http://www.w3.org/2002/07/owl#sameAs> ?president .
  filter regex(str(?president),"dbpedia")
}
}
```

Q_{12} : Recuperar o nome dos atores que participaram do filme Tarzan e as noticias relacionadas a eles.

```
select ?actor ?news WHERE {
service<http://virtuoso.mooo.com/nytimes/sparql>{
    ?y <http://www.w3.org/2002/07/owl#sameAs> ?x .
  ?y <http://data.nytimes.com/elements/topicPage> ?news .
}
service<http://virtuoso.mooo.com/linkedmdb/sparql>{
  ?film <http://purl.org/dc/terms/title> 'Tarzan' .
  ?film <http://data.linkedmdb.org/resource/movie/actor> ?actor .
  ?actor <http://www.w3.org/2002/07/owl#sameAs> ?x.
}
}
```

Q_{13} : Recuperar os dados sobre o genero e o director do filme After Life.

```
SELECT ?film ?director ?genre WHERE {
service<http://dbpedia.org/sparql>{
  select ?film ?director where{
  ?film <http://dbpedia.org/ontology/director>  ?director .
}
}
service<http://virtuoso.mooo.com/linkedmdb/sparql>{
  ?x <http://www.w3.org/2002/07/owl#sameAs> ?film .
  ?x <http://data.linkedmdb.org/resource/movie/genre> ?genre .
  filter regex(str(?film),"After.Life")
}
}
```

Figura 6.1: Carga de Trabalho C1 - Consultas Q11, Q12 e Q13

Q_{14} : Recuperar as URIs de doenças, nome das drogas que combatem cada doença e sua indicação.

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
select ?dg ?dgn where{
service<http://virtuoso.mooo.com/diseasome/sparql>{
                        ?ds diseasome:possibleDrug ?dg .
                           filter regex( str(?dg),"dailymed" )
}
service<http://virtuoso.mooo.com/dailymed/sparql>{
                              ?dg dailymed:fullName ?dgn ;
                              owl:sameAs ?sa ;
                              dailymed:indication ?indication .
                        }
}
```

Q_{15} : Recuperar o a URIs das drogas, a interação e o efeito da interação da droga.

```
SELECT ?Drug ?IntDrug ?IntEffect WHERE {
service<http://dbpedia.org/sparql>{
   ?Drug <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Drug> .
}
service<http://virtuoso.mooo.com/drugbank/sparql>{
   ?Int <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/interactionDrug1> ?y .
   ?Int <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/interactionDrug2> ?IntDrug .
   ?Int <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/text> ?IntEffect .
   ?y <http://www.w3.org/2002/07/owl#sameAs> ?Drug .
   filter regex (str(?Drug),"dbpedia")
}
}
```

Q_{16} : Recuperar a URI daa descrição, do substrato e da equação molecular das drogas que fazem parte da categoria "cathartics".

```
SELECT ?drug ?drugDesc ?cpd ?equation WHERE {
service<http://virtuoso.mooo.com/kegg/sparql>{
  ?enzyme <http://bio2rdf.org/kegg_vocabulary:xSubstrate> ?cpd .
  ?reaction <http://bio2rdf.org/kegg_vocabulary:xEnzyme> ?enzyme .
  ?reaction <http://bio2rdf.org/kegg_vocabulary:equation> ?equation .
}
service<http://virtuoso.mooo.com/drugbank/sparql>{
  ?drug <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/drugCategory> drugbank-category:cathartics .
  ?drug <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/keggCompoundId> ?cpd .
  ?drug <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/description> ?drugDesc .
}
}
```

Figura 6.2: Carga de Trabalho C1 - Consultas Q14, Q15 e Q16

Q_{21}: Recuperar a URL das drogas, a possível droga que combate a doença e os seus efeitos colaterais.

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
prefix owl: <http://www.w3.org/2002/07/owl#>
select  ?ds ?dg ?dgn ?sd_eff where{
service<http://virtuoso.mooo.com/diseasome/sparql>{
                                ?ds diseasome:possibleDrug ?dg .
                                      filter regex( str(?dg),"dailymed" )
}
service<http://virtuoso.mooo.com/dailymed/sparql>{
                                   ?dg dailymed:fullName ?dgn ;
                                   owl:sameAs ?sa ;
                                   dailymed:indication ?indication .
                                   filter regex(?dgn,"Capsule")
}
service<http://virtuoso.mooo.com/sider/sparql>{
                               ?sa sider:sideEffect ?se .
                               ?se sider:sideEffectName ?sd_eff .
}
}
```

Q_{22}: Recuperar o Nome do produtor e a URI da sua profissão no DBPedia.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix dbprop:<http://dbpedia.org/property/>
prefix dbpedia:<http://dbpedia.org/resource/>
prefix movie:<http://data.linkedmdb.org/resource/movie/>
select  ?occupation ?i where{
service<http://virtuoso.mooo.com/linkedmdb/sparql>{
    ?a  movie:producer_name  ?i .
}
service<http://virtuoso.mooo.com/dbplinkedmdb/sparql>{
                                  ?z  dbprop:occupation ?occupation;
                                     owl:sameAs ?a .
                                        filter regex(str(?a),"producer")
                                   filter regex(str(?a),"linkedmdb")
}
service<http://virtuoso.mooo.com/yago-dbpedia/sparql>{
                        ?x owl:sameAs ?z .
}
}
```

Q_{23}: Recuperar a URL de um recurso da base geospecies, informações sobre esse recurso e sua ordem na cadeia animal.

```
prefix dbprop:<http://dbpedia.org/property/>
prefix rdf-schema:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia-owl:<http://dbpedia.org/ontology/>
prefix owl:<http://www.w3.org/2002/07/owl#>
select ?s ?o ?y where{
service<http://virtuoso.mooo.com/geospecies/sparql>{
                           ?s   rdf-schema:seeAlso  ?o .
                                 filter regex(str(?o),"dbpedia")
}
service<http://lsd.taxonconcept.org/sparql>{
                        ?o dbpedia-owl:order ?y .
}
service<http://virtuoso.mooo.com/yago-dbpedia/sparql>{
                      ?h owl:sameAs ?y .
}
}
```

Q_{24}: Recuperar o a URI de drogas do Medicare, organismos afetados por elas e os efeitos colaterias.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
prefix dbpprop: <http://dbpedia.org/property>
PREFIX medicare: <http://www4.wiwiss.fu-berlin.de/medicare/resource/medicare/>
PREFIX dailymed:<http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX drugbank:  <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
SELECT ?dbmed ?koo  ?sd_efc
WHERE {
SERVICE <http://virtuoso.mooo.com/drugbank/sparql> {
                                  ?dbdg drugbank:affectedOrganism ?koo .
                                    ?dbdg owl:sameAs ?sdg .
}
SERVICE <http://virtuoso.mooo.com/medicare/sparql> {
                       ?dbmed    owl:sameAs ?dbdg .
                        filter regex (str(?dbdg),"drugbank")
}
Service <http://virtuoso.mooo.com/sider/sparql> {
                  ?sdg sider:sideEffect ?sd_efc .
}
}
```

Figura 6.3: Carga de Trabalho C2

Fontes usadas em cada uma das consultas		
Consulta - C1	Fonte 1	Fonte 2
Q11	dbpedia	nytimes
Q12	nytimes	linkedmdb
Q13	linkedmdb	dbpedia
Q14	diseasome	dailymed
Q15	dbpedia	drugbank
Q16	drugbank	kegg

Tabela 6.1: Fontes de dados utilizadas em cada uma das consultas que compõem a Carga de Trabalho C1

Número de tuplas resultantes de cada fonte que compõe cada consulta		
Consulta - C1	Fonte 1	Fonte 2
Q11	1525	6389
Q12	2	21683
Q13	4	75165
Q14	6124	25468
Q15	5195	8968
Q16	2	34298

Tabela 6.2: Número de tuplas obtidas em operações de *Scan* sobre as Fontes de Dados utilizadas em cada uma das consultas que compõem a Carga de Trabalho C1

A seguir, apresentaremos maiores detalhes sobre as fontes de dados utilizadas nas consultas que compõem as cargas de trabalho C1 e C2. A tabela 6.1 ilustra as fontes de dados utilizadas em cada uma das consultas que compõem a carga de trabalho C1. A Tabela 6.2 apresenta o número de triplas armazenadas nas fontes de dados utilizadas em cada uma das consultas que compõem a carga de trabalho C1. Já a Tabela 6.3 mostra o número de tuplas retornado ao final da execução de cada uma das consultas que compõem a carga de trabalho C1. Essas mesmas informações podem ser observadas nas tabelas 6.4, 6.5 e 6.6 para as consultas que compõem a carga de trabalho C2.

Número de tuplas retornadas pela consulta	
Consulta - C1	Número de tuplas
Q11	61
Q12	1
Q13	1
Q14	80224
Q15	8968
Q16	3

Tabela 6.3: Número de tuplas retornado ao final da execução de cada uma das consultas que compõem a carga de trabalho C1

Fontes usadas em cada uma das consultas			
Consultas - C2	Fonte1	Fonte2	Fonte3
Q21	diseasome	dailymed	sider
Q22	dbpedia-linkedmdb	linkedmdb	yago-dbpedia
Q23	geospecies	taxonconcept	yago-dbpedia
Q24	medicare	drugbank	sider

Tabela 6.4: Fontes de dados utilizadas em cada uma das consultas que compõem a Carga de Trabalho C2

Número de tuplas resultantes de cada fonte que compõe cada consulta			
Consultas -C2	Source1	Source2	Source3
Q21	6124	2850	126006
Q22	369	14882	1048576
Q23	12925	127367	1144824
Q24	276	8575	68423

Tabela 6.5: Número de tuplas obtidas em operações de *Scan* sobre as Fontes de Dados utilizadas em cada uma das consultas que compõem a Carga de Trabalho C2

Número de tuplas retornadas pela consulta	
Consultas -C)	Número de tuplas
Q21	40761
Q22	48
Q23	24
Q24	7701

Tabela 6.6: Número de tuplas retornado ao final da execução de cada uma das consultas que compõem a carga de trabalho C2

6.1.1 Ambiente de Testes

Todos os nossos experimentos foram executados em uma máquina cliente com processador Intel Core 2 Duo 2.93GHz e 2GB de memória RAM 667 MHz. Nessa mesma maquina foram instalados o sistema operacional Ubuntu Desktop 12.04 e a Java Virtual Machine (JVM) 32 bits na versão 1.6.0.22. Nessa máquina foram instaladas cada uma das ferramentas avaliadas nos experimentos (Fedx, Sesame, Jena e ASBJoin).

O servidor utilizado possui um processador Intel Core i7 2.93GHz e 16 GB de memória RAM DDR3 1333 MHz. No servidor foram instalados o sistema operacional Ubuntu Server 12.04 LTS e a aplicação *OpenLink Virtuoso OpenSource* como ambiente para armazenar as fontes de dados *R*DF que são utilizadas nos experimentos. Vale destacar que dentre todas as fontes de dados utilizadas nos experimentos, somente a fonte *DBPedia* foi acessada via Web. Todas as outras fontes de dados foram instaladas no computador servidor, o qual estava conectado na mesma rede local da máquina cliente. Assim, os dados *RDF* foram importados para a R*DF Store* a partir de dumps de conjuntos de dados disponíveis na Web sob licença aberta. Os dados de *diseasome, dailymed, sider, drugbank* e *dblp* foram importados por completo. Os dados importados de *linkedgeodata* incluíram somente as triplas com ligações para *dbpedia*. Já os dados usados da *DBpedia* foram apenas aqueles relacionados a coordenadas geográficas.

Durante os experimentos, tanto o servidor quanto a máquina cliente ficaram dedicados a essas tarefas em um ambiente controlado, onde a conexão entre ambos ocorreu através de uma rede local. Nesse contexto, a intenção foi evitar que interferências de outras atividades viessem adulterar os resultados dos experimentos.

Por várias vezes tentamos também realizar os mesmos experimentos sobre os dados originais disponíveis na Web. No entanto, as consultas sobrecarregavam os endpoints utilizados, chegando a causar a interrupção do serviço em certos casos. Esse problema ocorreu principalmente quando consultamos *Endpoints* que adotavam o servidor D2R. Dos sete conjuntos de dados que usamos, cinco deles (*diseasome, dailymed, sider, drugbank* e *dblp*) usavam o *D2R Server* como *Endpoint SPARQL*. Em outros casos, o servidor limitava os resultados inserindo um fim de arquivo antecipado que ocasionava uma exceção com a mensagem *"Premature end of file"* antes mesmo de chegar aos quatro minutos de execução da consulta. Infelizmente esses problemas dificultaram os experimentos no ambiente não controlado e real da Web que contém os conjuntos de dados originais que usamos.

Em cada um dos experimentos realizados, executamos cada consulta cinco vezes, desprezamos o tempo da primeira execução e calculamos a média das quatro execuções seguintes (teste conhecido como *cache* vazio ou *cache* frio). Vale ressaltar, ainda, que todas as ferramentas avaliadas foram executas utilizando-se sua configuração padrão.

6.2 Análise dos Experimentos

Nesta seção, serão apresentados os resultados dos experimentos realizados.

6.2.1 Cenário 1: Avaliando o *Overhead*

O primeiro experimento teve por objetivo avaliar o *overhead* gerado pelas atividades de monitoramento e de adaptação do ASBJoin. Para isso, executamos cada uma das cargas de trabalho utilizando o ASBJoin em duas circunstâncias distintas: i) com o módulo de monitoramento e de adaptação ativados; e ii) com o módulo de monitoramento e a adaptação desativados. As Figuras 6.4 e 6.5 ilustram os resultados deste experimento para as cargas de trabalho C1 e C2, respectivamente.

Analisando essas figuras, podemos observar que o ASBJoin com a adaptação (ou seja, com o módulo de monitoramento e a adaptação ativados) apresentou um tempo de execução levemente superior ao ASBJoin sem adaptação, como mostrado na Tabela 6.7. Entretanto, esse aumento de tempo já era esperado, por causa da execução, em paralelo, do módulo monitor, bem como pelas consultas ao catálogo realizadas pelo Processador de Consultas a fim de avaliar a necessidade de adaptação, que é executado sempre que um bloco de tuplas é construído no produtor da esquerda.

A Tabela 6.7 mostra os tempos de execução das consultas obtidos neste experimento e ilustrado nas Figuras 6.4 e 6.5.

Por outro lado, acreditamos que, apesar do pequeno *overhead* proporcionado pela adaptação, o ASBJoin com adaptação proporcionaria ganhos de desempenho em ambientes reais, onde as fontes de dados *RDF* na Web são instáveis. Nestes ambientes, encontrar formas de ajustar a execução da operação de junção e otimizar o processo de obtenção de resultados torna-se de fundamental importância.

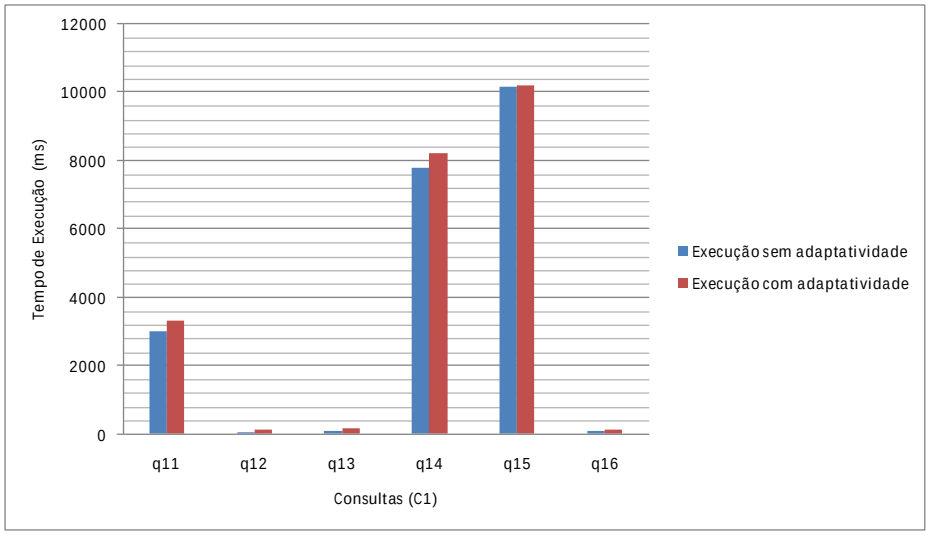

Figura 6.4: Tempo de Execução das Consultas da Carga de Trabalho C1 com e sem adaptação

Tempo de execução das consultas das cargas de trabalho C1 e C2 com e sem adaptatividade			
Consultas - C1 e C2	Resultado sem adaptatividade	Resultado com adaptatividade	Overhead
Q11	3003	3307	304
Q12	56	136	80
Q13	89	145	56
Q14	7777	8202	425
Q15	10186	10225	39
Q16	90	104	14
Q21	29712	31299	1587
Q22	2550	2982	432
Q23	99548	101328	1820
Q24	10057	11362	692

Tabela 6.7: Tempo de Execução das Consultas das Cargas de Trabalho C1 e C2 com e sem adaptatividade

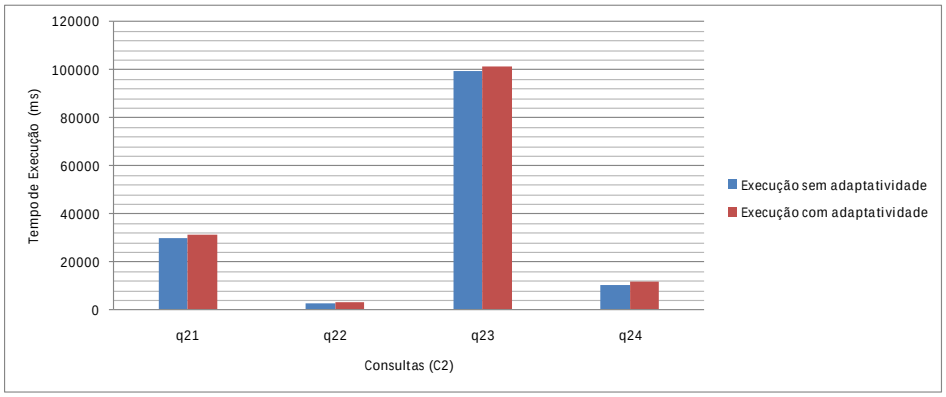

Figura 6.5: Tempo de Execução das Consultas da Carga de Trabalho C2 com e sem adaptação

6.3 Cenário 2: Avaliando o Impacto do Tamanho dos Blocos

O segundo experimento realizado teve por objetivo avaliar o impacto da variação do tamanho dos blocos de tuplas formados pelo produtor à esquerda no ASBJoin. Para isso, executamos cada uma das cargas de trabalho com diferentes valores para o tamanho do bloco de tuplas: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 e 55.

As Figuras 6.6 e 6.7 ilustram, para cada tamanho de bloco avaliado, os tempos de execução das consultas que compõem as cargas de trabalho C1 e C2, respectivamente. Pode-se observar que o tamanho do bloco influencia diretamente o desempenho das consultas. Além disso, podemos concluir que o desempenho das consultas melhora (ou seja, o tempo de execução diminui) à medida em que o tamanho do bloco cresce. Além disso, para determinadas consultas, o seu tempo de execução alcança um determinado limiar em relação ao tamanho do bloco. Esse fato pode ser observado nas consultas Q11 e Q15. As Tabelas 6.8 e 6.9 mostram os tempos de execução (em milisegundos) obtidos neste experimento.

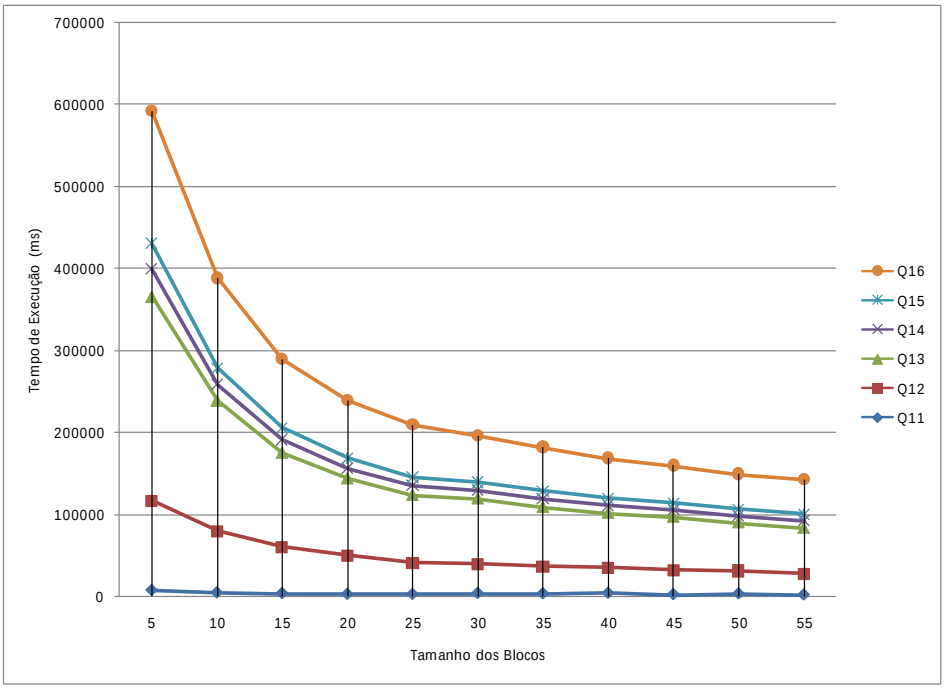

Figura 6.6: Variação dos tempos de execução das consultas que compõem a carga de trabalho C1 em função da variação do tamanho do bloco do produtor mais a esquerda do ASBJoin

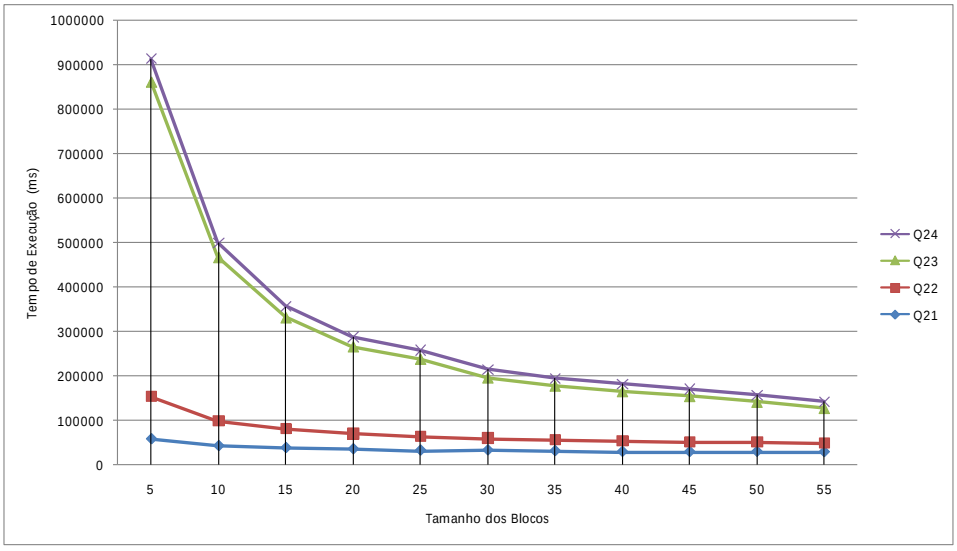

Figura 6.7: Variação dos tempos de execução das consultas que compõem a carga de trabalho C2 em função da variação do tamanho do bloco do produtor mais a esquerda do ASBJoin

6.4 Cenário 3: Comparação com as Ferramentas Jena, Sesame e FedX

O terceiro experimento realizado teve por finalidade comparar o desempenho do ABSJoin com as principais ferramentas relacionadas: Jena, Sesame e FedX. Para isso, as cargas de trabalho C1 e C2 foram executadas em cada uma dessas ferramentas. No ASBJoin, setamos

Tempo de execução das consultas por tamanho de bloco											
	5	10	15	20	25	30	35	40	45	50	55
Q11	7646	4673	3857	2839	2780	3937	3576	4161	2466	3455	2024
Q12	108908	75211	56924	47292	37966	35602	33118	30707	29949	28120	26062
Q13	249676	159237	114637	93789	82943	79848	72347	66990	64644	58514	55378
Q14	33550	19804	15692	12346	11159	10124	9390	8913	8436	8118	8719
Q15	30475	19004	14012	11884	10422	9524	9530	8445	8010	8090	8027
Q16	161749	110278	84226	71153	63881	57113	53910	49378	45934	43140	42556

Tabela 6.8: Tempos de execução das consultas que compõem a carga de trabalho C1 em função da variação do tamanho do bloco do produtor mais a esquerda do ASBJoinn

Tempo de execução das consultas por tamanho de bloco											
	5	10	15	20	25	30	35	40	45	50	55
Q21	57169	41816	37433	34800	31841	32223	30575	28897	28627	29121	28262
Q22	96407	56736	43004	34709	30289	26686	24978	23127	21625	20925	19763
Q23	708608	368245	250556	196210	175600	136734	121803	112762	103567	90745	78966
Q24	51517	32625	25338	21809	19745	18534	17266	16547	16016	15558	14777

Tabela 6.9: Variação dos tempos de execução das consultas que compõem a carga de trabalho C2 em função da variação do tamanho do bloco do produtor mais a esquerda do ASBJoin

o tamanho do bloco de tuplas obtido no produtor mais a esquerda com o valor 55 (por ter o melhor tempo na maioria das consultas durante o teste do Cenário 2).

As Figuras 6.8 e 6.9 resumem os resultados obtidos neste experimento. Nessas figuras podem-se observar os tempos de execução das consultas (em milissegundos) obtidos nas ferramentas Jena ARQ, Sesame, Fedx e o ASBJoin. As Tabelas 6.10 e 6.11 mostram, em detalhes, os resultados obtidos. Por meio da análise dos resultados obtidos, podemos observar que o ASBJoin proporcionou ganhos de desempenho significativos em relação às demais abordagens para a maioria das consultas. Por outro lado, o ASBJoin apresentou um desempenho levemente pior nas consultas com tempo de execução inferior a um segundo. Esse fato pode ser observado, por exemplo, na consulta Q16.

Vale destacar também que todas as consultas submetidas ao ABSJoin foram concluídas com sucesso. Por outro lado, nenhuma das demais ferramentas analisadas conseguiu concluir todas as consultas com sucesso (como podemos observar nas consultas Q14 e Q23, por exemplo). O Sesame apresentou dificuldades e problemas na execução de determinadas consultas. O Fedx se mostrou mais eficiente que o Jena ARQ e que o Sesame durante os experimento. Entretanto, o FedX falhou ao executar duas do total de dez consultas utilizadas.

6.5 Detalhes Observados Durante as Análises dos Experimentos

Durante a análise dos experimentos, foram observadas alguns fatores e detalhes importantes referentes à execução das consultas. A seguir, discutimos esses observações.

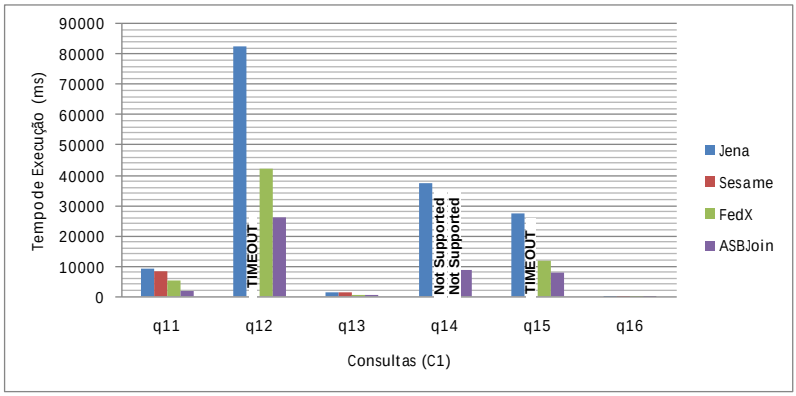

Figura 6.8: Tempos de execução das consultas que compõem a Carga de Trabalho C1

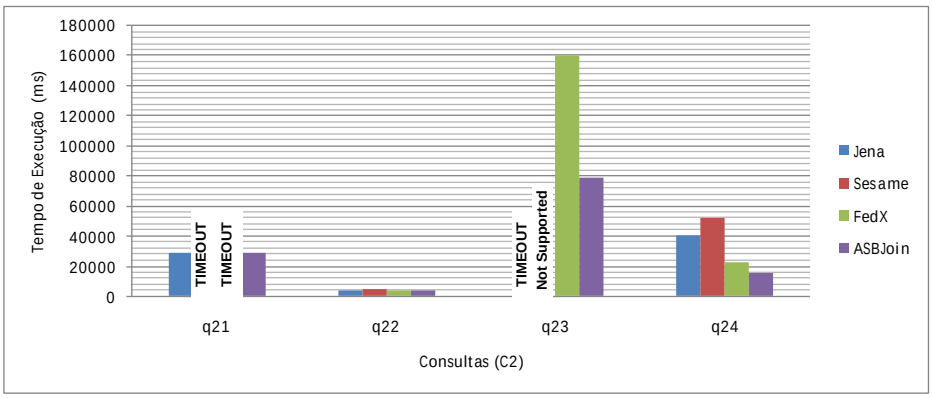

Figura 6.9: Tempos de execução das consultas que compõem a Carga de Trabalho C2

6.5.1 Adaptação das Fontes

Para mostrarmos que a adaptação funciona, elaboramos um experimento controlado. Neste experimento utilizamos apenas a consulta Q21, a qual possui uma operação de junção envolvendo três fontes de dados distintas: *Diseasome* (F1), *DailyMed* (F2) e *Sider* (F3). O ambiente de experimentação foi composto por duas máquinas virtuais (VM1 e VM2), onde cada VM é independente uma da outra e ambas possuem a mesma configuração em relação aos recursos da máquina servidor descrito na seção 6.1.2. . Na VM1 instalamos a fonte de dados *Diseasome* (F1). Já na VM2 instalamos as fontes de dados *DailyMed* (F2) e *Sider* (F3). Inicialmente, o ASBJoin selecionou como produtor da esquerda a fonte de dados *Diseasome* (F1) e como produtor da direita a fonte de dados *DailyMed* (F2). Em seguida, para analisar a influência dos atrasos na rede sobre o desempenho da consulta, forçamos a redução da banda de rede em diferentes tamanhos, variando entre 100 kbps até 10 kbps, na máquina virtual VM1. Ao reduzir a largura de banda para 10 kbps observamos que o ASBJoin realizou uma adaptação, alterando tanto o produtor da esquerda que deixou de ser a fonte *Diseasome* (F1) e passou a ser a fonte *DailyMed* (F2), quanto o produtor da direira, que deixou de ser a fonte *DailyMed* (F2) e passou a ser a fonte *Sider* (F3). Posteriormente, quando a largura de banda retornou para 100 kbps, o ASBJoin realizou uma segunda adaptação, retornando para a ordem de acesso selecio-

Tempo de execução das consultas				
Consultas - C1	Jena	Sesame	FedX	ASBJoinJoin
Q11	9148	8126	5212	2024
Q12	82549	Timeout	42151	26062
Q13	1408	1569	419	447
Q14	37397	Not Supported	Not Supported	8719
Q15	27543	Timeout	11609	8027
Q16	87	63	76	102

Tabela 6.10: Tempos de execução das consultas que compõem a Carga de Trabalho C1 em cada umas das abordagens avaliadas

Tempo de execução das consultas				
Consultas - C2	Jena	Sesame	FedX	ASBJoin
Q21	28610	Timeout	Timeout	28262
Q22	3633	4235	3616	3549
Q23	Timeout	Not Supported	160434	78966
Q24	40029	52369	22624	14777

Tabela 6.11: Tempos de execução das consultas que compõem a Carga de Trabalho C2 em cada umas das abordagens avaliadas

nada inicialmente, com a fonte *Diseasome* (F1) como produtor da esquerda e a fonte *DailyMed* (F2) como produtor da direita.

6.5.2 Problemas Relacionados às Fontes de Dados

Nessa seção relatamos os principais problemas observados em relação às consultas executadas sobre as fontes de dados na Web. Desta forma, apresentaremos as bases de dados de acordo com a aplicação que permite o acesso aos dados. Com isso, classificamos cada *dataset* em três grupos: D2R, OpenLink Virtuoso e os *datasets* que não podem ser acessados diretamente na Web. O D2R [2] e o OpenLink Virtuoso [3] são aplicações que permitem o acesso aos dados armazenados em *triplestores*. Os *datasets* que não podem ser acessados diretamente na Web de Dados estão disponíveis em documentos *RDF*, porém não existe nenhum endereço público de onde se possa acessá-los publicamente na Web.

- ***D2R:*** Dailymed [4], Sider [5], DrugBank [6] e LinkedMdb [7]
- ***OpenLink Virtuoso:*** DBpedia [8], KEGG [9]

[2] http://d2rq.org/d2r-server
[3] http://virtuoso.openlinksw.com/
[4] http://wifo5-04.informatik.uni-mannheim.de/Dailymed/sparql
[5] http://wifo5-04.informatik.uni-mannheim.de/Sider/sparql
[6] http://wifo5-04.informatik.uni-mannheim.de/drugbank/sparql
[7] http://www.linkedmdb.org/snorql/
[8] http://dbpedia.org/sparql
[9] http://dr. Bio2rdf.org/sparql

- ***Datasets que não podem ser acessados diretamente na WEB:*** Yago-DBpedia, Diseasome e NYTimes.

Durante os testes experimentais alguns problemas foram constatados, ao tentarmos realizar consultas contendo operações de junção em fontes de dados de acesso direto na Web. Alguns desses problemas serão, resumidamente, descritos a seguir:

- ***Falta de determinados endpoints na Web:*** Alguns dos dados em *datasets* como *NYTimes*, *Diseasome* e *Yago* utilizados nos experimentos foram encontradas no formato *RDF* e disponíveis para *download*. Entretanto, não foram encontrados fontes na Web correspondentes a esses dados, dificultando assim a realização de consultas federadas nesse ambiente.

- ***Processamento de consultas sobre dados na Web:*** Mesmo que fontes de dados possam acessar diretamente outras fontes Web, ainda existe a possibilidade de que determinadas consultas não possam ser processadas. Este foi o caso de algumas consultas executadas em nossos experimentos. Como exemplo, podemos citar a consulta Q15, descrita na Figura 6.2, que acessa dados de *Drugbank*. Tentamos executar essa consulta sob diversas maneiras. Esse problema foi observado, principalmente, em fontes *D2R*. A conclusão é que fontes *D2R* não são tão estáveis assim, e com isso, provavelmente não suporte um grande número de requisições feitas por determinadas consultas. Consultas a fontes acessadas através do *Virtuoso* se mostraram mais estáveis que em fontes acessadas através do aplicativo *D2R*. Outro problema está relacionado a má formação de URIs, o que pode ocasionar problemas na hora de realizar operações de junção entre diferentes fontes. Um exemplo pode ser visto na Figura 6.10, onde uma mesma URI, nativa da *DBPedia*, foi malformada em *LinkedMdb* e, devido a isso, ficou impossível fazer correspondências entre determinados dados contidos nessas duas fontes de dados.

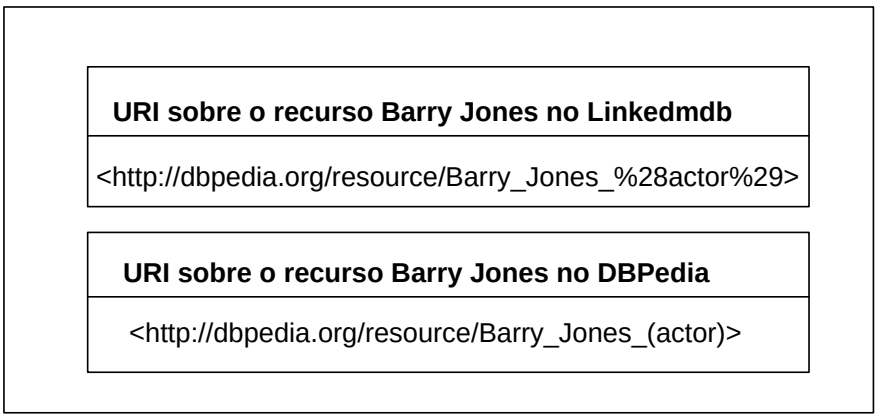

Figura 6.10: Exemplo de uma URI da DBPedia que foi malformada no *dataset Linkedmdb*

- ***Recursos com diferentes URIs e semanticamente similares:*** Determinados *datasets* contêm recursos similares. Entretanto, em alguns casos, esses recursos não possuem nenhuma ligação que indique que esses estejam ligados através de um predicado owl:sameAs.

Predicados owl:sameAS são utilizados para indicar que recursos de diferentes *datasets* são semanticamente iguais. Problemas relacionados a isso podem ser vistos em projetos que possuem *datasets* que contêm dados e domínios similares. Nesse caso, a URI desses recursos são diferentes e sem ligações entre esses termos. Um exemplo disso pode ser visto nos projetos Bio2RDF e Fuberlin, onde ambos descrevem dados que pertencem ao domínio de ciências da vida (*Life Science*). A base *Drugbank*, por exemplo, contida nesses dois projetos, possue informações equivalentes sobre dados de drogas. Entretanto, o *Drugbank* em *Bio2RDF* não possui tantas informações. No entanto, se houvessem ligações owl:sameAs entre recursos similares nos dois projetos, as informações contidas nas bases *Drugbank* dos dois projetos poderiam se enriquecer ainda mais. Talvez esse problema esteja atrelado à dificuldade de ligar milhares de recursos similares de forma automática, o que evitaria esforço manual. Existem algumas ferramentas que permitem esse tipo de ligação, a exemplo ao *SILK* [10] e o LIMES [11].

- ***Atualização dos prefixos das URIs de recursos:*** A questão da atualização automática dos prefixos de URIs de recursos ainda é um problema em aberto. Esse processo de atualização exige um certo esforço humano para verificar a existência de mudanças e também atualizar os mesmos recursos alterados na base original contidos em outras bases. Um exemplo disso foi a mudança do prefixo dos recursos contidos nas bases *Drugbank*, *Dailymed* e *Sider*. Atualmente o *Drugbank*, por exemplo, utiliza o prefixo <http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/drugbank/> para identificar cada recurso existente na sua base. Porém, antes o prefixo dos recursos dessa base era <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>. Algumas outras bases que tem ligações owl:sameAs, por exemplo, se ligam a recursos do *Drugbank*. Entretanto, muitas dessas bases não atualizaram suas informações levando em consideração essa mudança.

6.6 Considerações Finais

Nesse capitulo, descrevemos os resultados dos experimentos utilizando o operador ASBJoin. Inicialmente, foi apresentado as consultas utilizadas nos experimentos e foi descrito o ambiente onde os experimentos foram realizados. Para medir a performance do ASBJoin, foi realizado uma comparação entre diversas abordagens baseando-se no tempo de execução das consultas em cada dessas abordagens com o ASBJoin. Com isso, foi mostrado que o ASBJoin foi mais eficiente que as demais abordagens apresentadas nesse capítulo.

[10] http://lod2.eu/Project/Silk.html
[11] http://aksw.org/Projects/LIMES.html

7 CONCLUSÃO E TRABALHOS FUTUROS

7.1 Contribuições

Nesta dissertação propomos uma estratégia adaptativa para a execução de junções em consultas federadas sobre *Linked Data*, denominada ASBJoin. A estratégia foi implementada como uma extensão do *QEF-LD*. Um operador de junção adaptativo da álgebra *SPARQL*, denominado ASBJoin, foi implementado junto ao *QEF-LD*. O ambiente de execução concebido para executar a estratégia proposta inclui um módulo denominado Monitor. O módulo Monitor é responsável por coletar, periodicamente, e armazenar informações estatísticas sobre as fontes de dados utilizadas em uma determinada operação de junção. Essas estatísticas são armazenadas em um catálogo de dados. Durante a execução de uma operação de junção, as informações estatísticas armazenadas no catálogo são consultadas. Essas informações são utilizadas pelo operador ASBJoin para verificar a necessidade de adaptação, ou seja, de alterar a ordem com que as fontes de dados são acessadas.

Por fim, foram realizados experimentos para validar a estratégia proposta, ASBJoin, e comparar o seu desempenho com as principais ferramentas relacionadas: Jena, Sesame e FedX. Durante os testes foi possível constatar que a mudança na ordem de execução das fontes produtoras do ASBJoin afeta diretamente no desempenho da consulta.

7.2 Trabalhos Futuros

Durante a implementação e os testes, foram detectados alguns problemas relacionados às fontes e à consistência dos dados. Além disso, diversas melhorias podem ser implementadas, como, por exemplo:

- ***Conceber modelos de custo baseados em estatísticas para fontes RDF:*** Apesar da nossa estratégia de adaptação ter sido eficiente durante os testes, é possível conceber estratégias mais sofisticadas. Para isso, modelos de custo mais precisos devem ser concebidos. Além disso, torna-se necessário novos métodos que permitam obter informações estatísticas sobre as fontes de dados de forma mais rápida e eficiente. Uma possibilidade consiste em utilizar bases de dados auxiliares para armazenar os dados estatísticos sobre as fontes de dados. Essas bases auxiliares representarão informações estatísticas sobre os dados da sua fonte correspondente sob um vocabulário próprio para esse domínio de dados. Uma alternativa para isso consiste em utilizar o modelo descrito no VoID (ALEXANDER et al., 2009).

- ***Estender o ASBJoin para realizar junções entre n produtores:*** Devido a falta de tempo, não foi possível implementar um operador para realizar junções para um número qualquer de fontes de dados.

- ***Aprimorar o processamento paralelo das tarefas realizadas durante o processamento da consulta no ASBJoin:*** Apesar do ASBJoin ter sido superior as outras abordagens na

maioria dos casos, ele ainda utiliza processamento sequencial durante toda a etapa da consulta. A utilização de *threads* para possibilitar o processamento paralelo tornaria a consulta mais rápida e eficiente.

- ***Construção de um tradutor que converta uma consulta SPARQL para um QEP no formato do QEF-LD:*** O QEF-LD ainda utiliza um QEP como parâmetro de entrada. Devido ao fato de estarmos trabalhando com consultas federadas em *Linked Data* e também devido a popularidade da linguagem de consulta declarativa SPARQL, é necessário o uso de algum artifício de conversão de uma consulta SPARQL, fornecida pelo usuário, para um QEP, que é usado como entrada pelo QEF-LD.

- ***Estratégia para analisar outros tipos de operadores produtores do ASBJoin:*** O ASBJoin ainda não é capaz de classificar produtores diferente dos que obtém informações de fontes de dados na Web como, por exemplo, um outro operador de junção. É necessário encontrar formas de classificar diferentes tipos de produtores do ASBJoin.

REFERÊNCIAS BIBLIOGRÁFICAS

ACOSTA, M. et al. Anapsid: An adaptive query processing engine for sparql endpoints. In: *International Semantic Web Conference (1)*. [S.l.: s.n.], 2011. p. 18–34.

ALEXANDER, K. et al. Describing Linked Datasets - On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'. In: *WWW 2009 Workshop: Linked Data on the Web (LDOW2009)*. Madrid, Spain: [s.n.], 2009.

ALEXANDER, K. et al. *Describing Linked Datasets with the voiD Vocabulary*. December 2010. Disponível em: <http://www.w3.org/2001/sw/interest/void/>.

ARAÚJO, S. F. C.; SCHWABE, D. Explorator: a Tool for Exploring RDF Data Through Direct Manipulation. In: *LDOW 2009: Linked Data on the Web*. [S.l.: s.n.], 2009.

ARAÚJO, S. F. C.; SCHWABE, D.; BARBOSA, S. D. J. Experimenting with Explorator: a Direct Manipulation Generic RDF Browser and Querying Tool. In: *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*. [S.l.: s.n.], 2009.

AUER, S. et al. Triplify: Light-weight linked data publication from relational databases. In: QUEMADA, J. et al. (Ed.). *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*. [S.l.]: ACM, 2009. p. 621–630. ISBN 978-1-60558-487-4.

AVNUR, R.; HELLERSTEIN, J. M. Eddies: continuously adaptive query processing. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 29, n. 2, p. 261–272, maio 2000. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/335191.335420>.

BECKER, C.; BIZER, C. DBpedia Mobile: A Location-Enabled Linked Data Browser. In: *Linked Data on the Web (LDOW2008)*. [S.l.: s.n.], 2008.

BECKETT, D.; BROEKSTRA, J. *SPARQL Query Results XML Format*. January 2008. World Wide Web Consortium, Recommendation REC-rdf-sparql-XMLres-20080115.

BERNERS-LEE, T. *Linked Data ? Design Issues*. 2006. Http://www.w3.org/DesignIssues/LinkedData.html.

BERNERS-LEE, T. et al. The world-wide web. *Commun. ACM*, ACM, New York, NY, USA, v. 37, n. 8, p. 76–82, ago. 1994. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/179606.179671>.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *Uniform Resource Identifiers (URI): Generic Syntax*. United States: RFC Editor, 1998.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific American*, v. 284, n. 5, p. 34–43, maio 2001. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>.

BERNERS-LEE, T. et al. *Tabulator Redux: Writing Into the Semantic Web*. [S.l.], 2007. Disponível em: <http://eprints.soton.ac.uk/264773/>.

BIZER, C. Evolving the web into a global data space. In: *Proceedings of the 28th British national conference on Advances in databases*. Berlin, Heidelberg: Springer-Verlag, 2011. (BNCOD'11), p. 1–1. ISBN 978-3-642-24576-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2075914.2075915>.

BIZER, C.; CYGANIAK, R.; HEATH, T. *How to publish Linked Data on the Web*. 2007. Disponível em: <http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/>.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, v. 5, n. 3, p. 1–22, 2009.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, v. 5, n. 3, p. 1–22, 2009. Disponível em: <http://www.igi-global.com/article/linked-data-story-far/37496>.

BIZER, C.; SEABORNE, A. D2rq - treating non-rdf databases as virtual rdf graphs. In: *ISWC2004 (posters)*. [s.n.], 2004. Disponível em: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/Bizer-D2RQ-ISWC2004-Poster.pdf>.

BRUNO, N. *Automated Physical Database Design and Tuning*. [S.l.]: CRC Press, 2011. 256 p.

CLARK, K. G. *SPARQL Protocol for RDF*. [S.l.], 2006.

CLARK, K. G.; FEIGENBAUM, L.; TORRES, E. *Serializing SPARQL Query Results in JSON*. 2008.

D'AQUIN, M. et al. Characterizing knowledge on the semantic web with watson. In: *Evaluation of Ontologies and Ontology-Based Tools: 5th International EON Workshop*. [S.l.: s.n.], 2007.

DAS, S.; SUNDARA, S.; CYGANIAK, R. *R2RML: RDB to RDF Mapping Language*. 2011.

DESHPANDE, A.; IVES, Z.; RAMAN, V. Adaptive query processing. *Found. Trends databases*, Now Publishers Inc., Hanover, MA, USA, v. 1, n. 1, p. 1–140, jan. 2007. ISSN 1931-7883. Disponível em: <http://dx.doi.org/10.1561/1900000001>.

DING, L. et al. Swoogle: a search and metadata engine for the semantic web. In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, 2004. (CIKM '04), p. 652–659. ISBN 1-58113-874-1.

Douglas Ericson and Fabio Porto. *QEF Manual do Usuário*. 2010. Http://dexl.lncc.br/QEF_-Manual.html.

DÜRST, M.; SUIGNARD, M. *RFC 3987 – Internationalized Resource Identifiers (IRIs)*. 2005.

ERLING, O.; MIKHAILOV, I. *Mapping Relational Data to RDF in Virtuoso*. 2006.

FIELDING, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. United States: RFC Editor, 1999.

FLORESCU, D. et al. Query optimization in the presence of limited access patterns. In: *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1999. (SIGMOD '99), p. 311–322. ISBN 1-58113-084-8. Disponível em: <http://doi.acm.org/10.1145/304182.304210>.

GöRLITZ, O.; STAAB, S. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: *Proceedings of the 2nd International Workshop on Consuming Linked Data*. Bonn, Germany: [s.n.], 2011. Disponível em: <http://uni-koblenz.de/ goerlitz/publications/-GoerlitzAndStaab_COLD2011.pdf>.

GRAEFE, G. Encapsulation of parallelism in the volcano query processing system. In: *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1990. (SIGMOD '90), p. 102–111. ISBN 0-89791-365-5.

GRAEFE, G. Query evaluation techniques for large databases. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 25, p. 73–169, June 1993. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/152610.152611>.

HAASE, P.; MATHäSS, T.; ZILLER, M. An evaluation of approaches to federated query processing over linked data. In: *Proceedings of the 6th International Conference on Semantic Systems*. New York, NY, USA: ACM, 2010. (I-SEMANTICS '10), p. 5:1–5:9. ISBN 978-1-4503-0014-8. Disponível em: <http://doi.acm.org/10.1145/1839707.1839713>.

HARTIG, O.; BIZER, C.; FREYTAG, J.-C. Executing sparql queries over the web of linked data. In: *Proceedings of the 8th International Semantic Web Conference*. Berlin, Heidelberg: Springer-Verlag, 2009. (ISWC '09), p. 293–309. ISBN 978-3-642-04929-3.

HARTIG, O.; LANGEGGER, A. A database perspective on consuming linked data on the web. *Datenbank-Spektrum*, v. 10, n. 2, p. 57–66, 2010.

HEATH, T.; BIZER, C. *Linked Data: Evolving the Web into a Global Data Space*. 1st. ed. Morgan & Claypool, 2011. 136 p. ISBN 9781608454303. Disponível em: <http://linkeddatabook.com/>.

HORROCKS, I. et al. Semantic web architecture: Stack or two towers? In: *PPSWR*. [S.l.: s.n.], 2005. p. 37–41.

LANGEGGER, A. *A Flexible Architecture for Virtual Information Integration based on Semantic Web Concepts*. Tese (Doutorado) — J. Kepler University Linz, 2010.

LANGEGGER, A.; WOSS, W. Rdfstats - an extensible rdf statistics generator and library. In: *Proceedings of the 2009 20th International Workshop on Database and Expert Systems Application*. Washington, DC, USA: IEEE Computer Society, 2009. (DEXA '09), p. 79–83. ISBN 978-0-7695-3763-4. Disponível em: <http://dx.doi.org/10.1109/DEXA.2009.25>.

LE-PHUOC, D. et al. A native and adaptive approach for unified processing of linked streams and linked data. In: *Proceedings of the 10th international conference on The semantic web - Volume Part I*. Berlin, Heidelberg: Springer-Verlag, 2011. (ISWC'11), p. 370–388. ISBN 978-3-642-25072-9. Disponível em: <http://dl.acm.org/citation.cfm?id=2063016.2063041>.

LENZERINI, M. Data integration: A theoretical perspective. In: *PODS*. [S.l.: s.n.], 2002. p. 233–246.

LYNDEN, S. J. et al. Adaptive integration of distributed semantic web data. In: KIKUCHI, S.; SACHDEVA, S.; BHALLA, S. (Ed.). *Databases in Networked Information Systems, 6th International Workshop, DNIS 2010, Aizu-Wakamatsu, Japan, March 29-31, 2010. Proceedings*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science, v. 5999), p. 174–193. ISBN 978-3-642-12037-4.

MAGALHãES, R. P. *Um Ambiente para Processamento de Consultas Federadas em Linked Data Mashups*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2012.

MANOLA, F.; MILLER, E. RDF primer. *W3C Recommendation*, W3C, v. 10, p. 1–107, 2004. Disponível em: <http://www.w3.org/TR/rdf-primer/>.

OLIVEIRA, D. E. de; PORTO, F. *QEF User Manual*. [S.l.], September 2010.

OREN, E. et al. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 3, n. 1, p. 37–52, nov. 2008. ISSN 1744-2621. Disponível em: <http://dx.doi.org/10.1504/IJMSO.2008.021204>.

PORTO, F. et al. Qef - supporting complex query applications. In: *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007. (CCGRID '07), p. 846–851. ISBN 0-7695-2833-3. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2007.89>.

PRUD'HOMMEAUX, E.; BUIL-ARANDA, C. *SPARQL 1.1 Federated query*. 2011. Http://www.w3.org/TR/sparql11-federated-query/.

PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF*. 2008. Http://www.w3.org/TR/rdf-sparql-query/.

QUILITZ, B.; LESER, U. Querying Distributed RDF Data Sources with SPARQL. In: *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*. Berlin, Heidelberg: Springer-Verlag, 2008. (ESWC'08), p. 524–538. ISBN 3-540-68233-3, 978-3-540-68233-2.

RIBEIRO, T. d. G. *Uma Abordagem Baseada em Ontologias para a Geração de Serviços Web de Integração de Dados*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2011.

SAUERMANN, L.; CYGANIAK, R.; VÖLKEL, M. *Cool URIs for the Semantic Web*. Deutsches Forschungszentrum f ür Künstliche Intelligenz GmbH, February 2007. Disponível em: <http://www.dfki.uni-kl.de/dfkidok/publications/TM/07/01/tm-07-01.pdf>.

SCHWARTE, A. et al. Fedx: Optimization techniques for federated query processing on linked data. In: *International Semantic Web Conference (1)*. [S.l.: s.n.], 2011. p. 601–616.

SEABORNE. *SPARQL 1.1 Query Results CSV and TSV Formats*. [S.l.], 2011. Disponível em: <http://www.w3.org/TR/sparql11-results-csv-tsv/>.

SOUZA, R. R.; ALVARENGA, L. A web semântica e suas contribuições para a ciência da informação. *Ciencia da Informação*, scielo, v. 33, p. 132 – 141, 04 2004. ISSN 0100-1965.

TUMMARELLO, G. et al. Sig.ma: Live views on the web of data. In: *Proceedings of the 19th International Conference on World Wide Web*. New York, NY, USA: ACM, 2010. (WWW '10), p. 1301–1304. ISBN 978-1-60558-799-8. Disponível em: <http://doi.acm.org/10.1145/1772690.1772907>.

URHAN, T.; FRANKLIN, M. J. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, v. 23, n. 2, p. 27–33, 2000.

VALDURIEZ, P.; GARDARIN, G. Join and semijoin algorithms for a multiprocessor database machine. *ACM Trans. Database Syst.*, ACM, New York, NY, USA, v. 9, n. 1, p. 133–161, mar. 1984. ISSN 0362-5915. Disponível em: <http://doi.acm.org/10.1145/348.318590>.

VIDAL, V. M. P. et al. Query Processing in a Mediator Based Framework for Linked Data Integration. *IJBDCN*, v. 7, n. 2, p. 29–47, 2011.

WIEDERHOLD, G. Mediators in the architecture of future information systems. *Computer*, v. 25, n. 3, p. 38–49, 1992.