UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ezequiel Rodrigues Ribeiro

IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON

Santa Maria, RS
2019

Ezequiel Rodrigues Ribeiro

IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Ciência da Computação**

Orientadora: Profª. Drª. Deise de Brum Saccol

Santa Maria, RS

2019

Rodrigues Ribeiro, Ezequiel

IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON / por Ezequiel Rodrigues Ribeiro. – 2019.

92 f.: il.; 30 cm.

Orientadora: Deise de Brum Saccol

Trabalho de Conclusão de Curso - Universidade Federal de Santa Maria, Centro de Tecnologia, Graduação em Ciência da Computação, RS, 2019.

1. NoSQL. 2. JSON. 3. Esquemas. 4. Extração. I. de Brum Saccol, Deise. II.IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON.

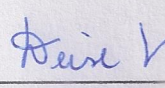
E-mail: ezequielrr@inf.ufsm.br

Ezequiel Rodrigues Ribeiro

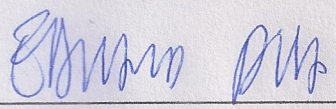
IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Ciência da Computação**

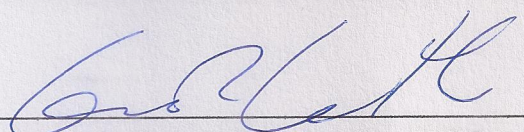
Aprovado em 08 de julho de 2019:



Deise de Brum Saccol, Drª. (UFSM)
(Presidente/Orientadora)



Eduardo Kessler Pivetta, Dr. (UFSM)



Giovani Rubert Librelotto, Dr. (UFSM)

Santa Maria, RS

2019

AGRADECIMENTOS

Em primeiro lugar, sou grato a Deus, por ter me acompanhado, provido forças e oportunidades para um constante aperfeiçoamento, em seguida, à minha família, por todo apoio e compreensão, à orientadora, Profa. Dra. Deise de Brum Saccol, pela orientação, repreensões, paciência e palavras de ânimo durante todo o processo de desenvolvimento do trabalho. À banca, composta pelo Prof. Dr. Giovani Rubert Librelotto e Prof. Dr. Eduardo Kessler Pivetta, que ajudaram elucidar o trabalho, através das sugestões de melhorias. Pelo carinho, bom humor, conselhos e constantes incentivos de minha namorada Aldiara, principalmente nos momentos mais difíceis da vida acadêmica. Aos meus atuais chefes, Cleiton e Diego, pela disposição ao diálogo e paciência ao flexibilizar a carga horária do trabalho para que eu pudesse desempenhar as obrigações acadêmicas. Aos amigos dona Rosa, Carina, Lucas, Cátia, Carlos e Jovani, pela amizade, apoio e apontamentos referentes ao texto do trabalho. À Sra. Regina e família, pela companhia, amizade, conselhos e suporte fornecidos. Aos amigos Luís e Nayra, por todos os bons conselhos e troca de conhecimentos. Também aos amigos Diogo, Elmo, João e Rodrigo, pela companhia e apoio. Aos irmãos e irmãs de coração da Igreja em Santa Maria e da 17º Igreja do Evangelho Quadrangular, por todo apoio e amizades conquistadas. À Universidade Federal de Santa Maria, por todo conhecimento adquirido e infraestrutura proporcionados durante todo o percurso acadêmico.

RESUMO

IMPLEMENTAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE ESQUEMAS IMPLÍCITOS EM BASES DE DADOS JSON

AUTOR: EZEQUIEL RODRIGUES RIBEIRO
ORIENTADORA: DEISE DE BRUM SACCOL

Este estudo aborda reflexões sobre implementação de algoritmos para extração de esquemas implícitos em bases de dados JSON. Diante as tecnologias e aplicações presentes e sua consequente demanda massiva de dados, questões como, por exemplo, suporte à escalabilidade, se tornaram pertinentes. O objetivo geral do referido estudo foi implementar algoritmos para a extração de esquemas a partir de fontes de dados JSON e os específicos foram: estudar e analisar o processo de extração de esquemas a partir de fontes de dados JSON proposto por Machado (2017); projetar e implementar a ferramenta, detalhando o funcionamento dos algoritmos utilizados; testar e validar a ferramenta em um estudo de caso. Quanto à metodologia, este estudo consiste em uma pesquisa exploratória, descritiva com apresentação de análises quantitativas e qualitativas. Num primeiro momento buscou-se reflexões consistentes na revisão bibliográfica dos principais assuntos abordados, entre eles estão o banco de dados NoSQL, Mongo DB, Java e o processo de extração de esquemas implícitos em bases de dados NoSQL proposto pela aurora. No segundo momento consistiu-se na especificação da ferramenta, sendo detalhada sua estrutura e funcionamento. Foi utilizada a linguagem Java para o desenvolvimento dos algoritmos e os dados serão salvos em arquivos de texto JSON, CSV ou TXT. No terceiro momento consistiu-se na validação da ferramenta desenvolvida, através de sua explicação, descrita em um estudo de caso. Pode-se dizer que dessa forma, o presente trabalho apresentou uma implementação para as atividades, Consolidar Estrutura, Remontar estrutura, Montar Mapeamentos e Adaptar à Notação de Agregados, propostos pela autora do processo.

Palavras-chave: NoSQL. JSON. Esquemas. Extração.

ABSTRACT

IMPLEMENTATION OF ALGORITHMS FOR EXTRACTION OF IMPLICIT SCHEMES IN JSON DATABASES

AUTHOR: EZEQUIEL RODRIGUES RIBEIRO
ADVISOR: DEISE DE BRUM SACCOL

This paper discusses the implementation of algorithms for extracting schemas implicit in JSON databases. Provided the technologies and applications present and their consequent massive demand for data, issues such as scalability support have become relevant. The general objective of this study was to implement algorithms for the extraction of schemas from JSON data sources and the specific ones were: to study and to analyze the schema extraction process from JSON data sources proposed by Machado (2017); design and implement the tool, detailing the operation of the algorithms used; test and validate the tool in a case study. Regarding the methodology, this study consists of an exploratory, descriptive research with presentation of quantitative and qualitative analyzes. Firstly, we sought to make consistent reflections on the bibliographic review of the main issues addressed, among them are the database NoSQL, Mongo DB, Java and the process of extracting schemas implicit in NoSQL databases proposed by aurora. In the second moment it consisted of the specification of the tool, being detailed its structure and operation. The Java language was used for the development of the algorithms and the data will be saved in JSON, CSV or TXT text files. In the third moment we consisted of the validation of the developed tool, through its explanation, described in a case study. It can be said that in this way, the present work presented an implementation for the activities, Consolidate Structure, Reassemble structure, Mount Mappings and Adapt to Aggregate Notation, proposed by the author of the process.

Keywords: NoSQL. JSON. Schemes. Extraction.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE LISTAGENS

LISTA DE ABREVIATURAS E SIGLAS

IDEF1X	Integration DEFinition for information modeling
NoSQL	Not only SQL
JSON	JavaScript Object Notation
BD	Banco de Dados
SQL	Structured Query Language
SGBD	Sistema Gerenciador de Banco de Dados
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
BASE	Basically Available, Soft State, Eventual Consistency
BSON	Binary JSON
XML	Extensible Markup Language
CSV	Comma-separated values
PDF	Portable Document Format

SUMÁRIO

1 INTRODUÇÃO

Diante de novas tecnologias e aplicações presentes nas últimas décadas e sua consequente geração massiva de dados, questões como por exemplo, suporte à escalabilidade, se tornaram pertinentes. Naturalmente, para que um banco de dados relacional possa ser expandido para atender determinada demanda, o mais comum é a adição de armazenamento, processamento e memória RAM (escalabilidade vertical), que pode ser custoso e até mesmo inviável a depender do contexto. Em um sistema distribuído, existe a escalabilidade horizontal, que consiste na adição de novos computadores ao complexo do serviço prestado, permitindo também a expansão da capacidade de um sistema para atender certa demanda, entretanto, os bancos de dados relacionais originalmente não foram projetados para execução em um sistema de dados distribuído (VIEIRA et al., 2012).

Frente a esta limitação, soluções como, por exemplo, o BigTable da Google Inc. (CHANG et al., 2008) foram desenvolvidas com o intuito de suprir tais necessidades. Esses novos modelos de bancos de dados são nomeados pelo termo NoSQL (Not only SQL) e são caracterizados por poder ser executados de forma distribuída, não possuir relacionamentos gerenciados pelo SGBD (Sistema Gerenciador de Banco de Dados) e não possuir um esquema definido. São, geralmente, oriundos de projetos de código aberto (NOSQL DATABASES, 2019).

Entretanto, a flexibilidade de esquemas nas bases de dados NoSQL (BRITO, 2010), permite que determinado campo esteja presente em alguns documentos e em outros não, como também a existência de campos que contenham a mesma informação, mas são descritos por nomes diferentes (MACHADO, 2017), sendo necessário o desenvolvimento de processos que visam a extração do esquema implícito unificado a partir das bases de dados.

Portanto, a extração de um esquema implícito em bases de dados NoSQL se torna importante no momento em que é preciso acessar um banco de dados em ambientes distribuídos (na web, por exemplo), composto por arquivos com diferenças estruturais, através de uma única ferramenta ou software. A extração dos esquemas implícitos presentes em bases de dados NoSQL viabiliza o desenvolvimento de aplicações com o intuito de acessar diversas fontes de dados de forma unificada.

1.1 OBJETIVO GERAL

O presente trabalho tem como objetivo implementar algoritmos para extração de esquemas a partir de fontes de dados JSON.

1.2 OBJETIVOS ESPECÍFICOS

Para alcançar o objetivo geral, o presente trabalho possui os seguintes objetivos específicos:

- Estudar e analisar o processo de extração de esquemas a partir de fontes de dados JSON proposto por Machado (2017);
- Projetar e implementar a ferramenta, detalhando o funcionamento dos algoritmos utilizados;
- Testar e validar a ferramenta em um estudo de caso;

1.3 METODOLOGIA

O presente trabalho se caracteriza como uma pesquisa de caráter exploratório e descritivo, com apresentação de análises quantitativas e qualitativas. O desenvolvimento do trabalho será dividido em três etapas.

A primeira etapa consiste na revisão bibliográfica dos principais assuntos abordados, entre eles destacam-se os bancos de dados NoSQL, MongoDB, Java e o processo de extração de esquemas implícitos em bases de dados NoSQL proposto pela autora. Será dada ênfase ao modelo de banco de dados baseado em documentos, no qual se encaixa o MongoDB, por ser, dentre os tipos de bancos de dados NoSQL existentes, o mais usado.

A segunda etapa consiste na especificação da ferramenta, sendo detalhada sua estrutura e funcionamento. Será utilizada a linguagem Java para o desenvolvimento dos algoritmos e os dados serão salvos em arquivos de texto JSON, CSV ou TXT.

A terceira etapa consiste na validação da ferramenta desenvolvida, através de sua aplicação, descrita em um estudo de caso.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho se divide em cinco capítulos. O primeiro apresenta a introdução, objetivos e metodologia do trabalho. O Segundo capítulo apresenta a fundamentação teórica para a compreensão e desenvolvimento do trabalho. O terceiro capítulo apresenta a documentação da implementação dos algoritmos. O quarto capítulo se destina à descrição do estudo de caso e validação dos algoritmos desenvolvidos. O quinto capítulo apresenta a conclusão e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentada uma revisão dos principais conceitos para o entendimento e desenvolvimento do presente trabalho. Será explicado sobre Bancos de Dados NoSQL orientados a documentos, uma visão geral do processo de extração de esquemas implícitos a partir de bancos de dados NoSQL no formato JSON e os algoritmos propostos por (MACHADO, 2017).

2.1 BANCOS DE DADOS NOSQL

Os BDs NoSQL surgiram como uma solução para a demanda de grandes empresas no começo do século XX, referentes ao processamento e armazenamento de grandes volumes de dados. Um banco de dados NoSQL pode ser definido como um sistema distribuído, que não exige esquemas para sua estrutura, evitam operações de join e são escaláveis horizontalmente, não são manipulados via SQL e têm sua origem a partir de projetos open source (TUDORICA; BUCUR, 2011). Ao oferecer alto desempenho e alta disponibilidade, os bancos de dados NoSQL não priorizam a consistência, pois, de acordo com o teorema CAP, alta disponibilidade e consistência são propriedades que não são possíveis de serem garantidas de forma absoluta e simultânea, quando se opta por tolerância a partição de rede, que é o caso de sistemas distribuídos (BREWER, 2000).

Por não ser priorizada a consistência, os bancos de dados NoSQL acabam perdendo as características ACID, presentes em Bancos de Dados relacionais, em troca de manter as propriedades BASE. Segundo Brewer (2012), o foco de um sistema BASE é a alta disponibilidade, enquanto um sistema ACID tem seu foco na consistência. De forma resumida, um sistema BASE é um sistema que funciona basicamente todo o tempo (Basically Available) e não precisa ser consistente todo o tempo (Soft-state), pois em um momento indeterminado, o sistema estará em um estado consistente (Eventually Consistent). Assim, os bancos de dados NoSQL garantem que as atualizações são eventualmente propagadas para todos os nós, mas não existe garantia sobre a consistência das leituras. Sendo assim, conseguem garantir alto desempenho em operações de leitura e escrita, além de alta disponibilidade e escalabilidade (SADALAGE; FOWLER, 2013).

2.2 BANCOS DE DADOS ORIENTADOS A DOCUMENTOS

Os BDs orientados a documentos, ou simplesmente BDs de documentos, não têm seu foco com alto desempenho em operações de leitura e escrita concorrentes, mas sim, em garantir um armazenamento eficiente de grandes volumes de dados e bom desempenho em operações de consulta (HAN et al., 2011). Sendo assim, BDs NoSQL orientados a documentos tentam melhorar a disponibilidade através da replicação de dados, utilizando uma configuração do tipo mestre-escravo (SADALAGE; FOWLER, 2013). Os mesmos dados ficam disponíveis em múltiplos nodos, os quais os clientes podem acessar mesmo quando o nodo primário estiver indisponível. Esses BDs podem ser considerados como uma subcategoria dos BDs chave-valor, tendo como diferença o fato de que, enquanto os BDs chave-valor armazenam valores como BLOB, os BDs orientados a documentos armazenam os valores como documentos, os quais podem ser armazenados em um formato de troca de dados padrão, como XML, JSON ou BSON (HASHEM; RANC, 2016; INDRAWAN-SANTIAGO, 2012; HAN et al., 2011). Um dos recursos interessantes dos BDs orientados a documentos, comparados aos BDs chave-valor, é a possibilidade de consultar os dados dentro dos documentos sem ter de recuperar o documento inteiro por sua chave e depois examiná-lo (SADALAGE; FOWLER, 2013).

Em geral, BDs orientados a documentos não possuem esquema, por isso, o número de campos não é limitado e novos campos podem ser adicionados dinamicamente a um documento. Um documento, por sua vez, pode ser representado como uma estrutura de dados em forma de árvore, constituída por valores escalares (string, boolean, numérico), por documentos aninhados e coleções. Como outros BDs NoSQL, BDs de doLOcumentos também não fornecem propriedades de transação ACID. BDs populares nessa categoria, segundo DB-Engines Ranking são: MongoDB, CouchDB e Couchbase.

2.3 PROCESSO DE EXTRAÇÃO DE ESQUEMAS

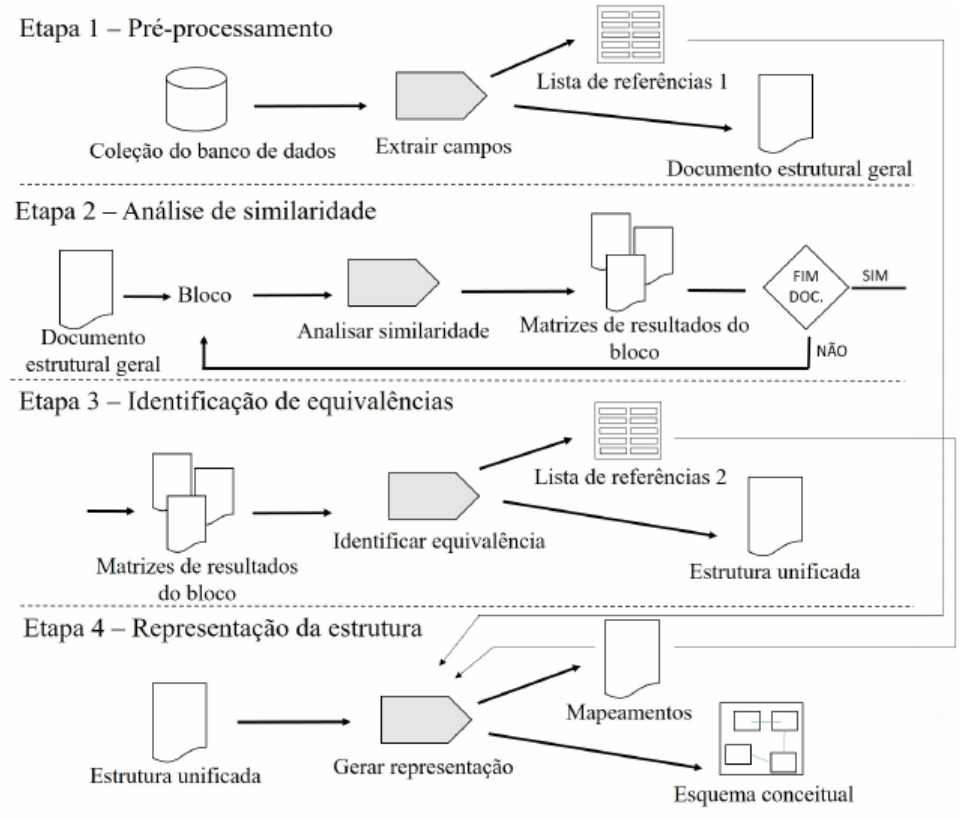
O presente trabalho utiliza como base para sua execução o processo proposto por Machado (2017) para a extração do esquema da base de dados NoSQL. Será então apresentada uma visão geral do processo e um detalhamento das etapas que são o alvo da implementação do trabalho.

2.3.1 Visão Geral

Um banco de dados NoSQL orientado a documentos é composto por um conjunto de coleções e cada coleção é constituída por um conjunto de documentos. Cada documento é um registro em formato JSON que pode conter campos aninhados como objetos e arrays. Cada registro possui um identificador único, gerado pelo SGBD ou atribuído via código ao ser criado. Frente à flexibilidade de esquema presente nos bancos de dados NoSQL, campos que representam a mesma informação podem acabar por ser nomeados de forma diferente e, para a geração de uma representação conceitual consolidada do sistema, é necessária a eliminação de tais diferenças.

O processo proposto pela autora, como apresentado na Figura 1, é constituído por 4 etapas principais (pré-processamento, análise de similaridade, identificação de equivalências e representação da estrutura), sendo estas apresentadas a seguir.

Figura 1 – Visão geral do processo para extração de esquemas.



Fonte: (MACHADO, 2017).

1. **Pré-processamento:** executada entre os documentos da coleção, com a finalidade de percorrer e unificar todos os documentos em um só, que contém apenas os campos, sendo

estes analisados nas etapas posteriores. Para os documentos que apresentam campos distintos do documento de unificação, é armazenada uma lista, que contém as referências dos campos distintos a seus respectivos documentos de origem;

2. **Análise de similaridade:** o documento provindo da etapa anterior é então processado por blocos, através de iterações, até atingir o seu fim. Para cada bloco, são aplicadas técnicas de similaridade baseadas em cadeias de caracteres, conhecimento e também do extrator de radicais, que visa identificar equivalências entre campos com nomes diferentes, porém representam a mesma informação. Cada técnica aplicada sobre o bloco gera como resultado uma matriz de similaridade, resultando num total de três matrizes de similaridade geradas para cada bloco processado;

3. **Identificação de equivalências:** assim como a etapa 2, é realizada por blocos. Este processo é responsável pela definição de equivalência ou não. Para tal, são realizados cruzamentos entre os dados das três matrizes resultantes (obtidas na etapa anterior) e é então calculado o resultado. É executada repetidas vezes até chegar ao fim do documento. Gera como saída um arquivo com a estrutura unificada, mais uma segunda lista contendo as equivalências entre os campos;

4. **Representação da estrutura:** etapa com regras que percorrem a estrutura unificada com o intuito de detectar entidades, atributos e relacionamentos. Conforme o que for identificado, é então gerada uma representação conceitual do banco de dados NoSQL. Ao consultar as listas de referências geradas no processo, os mapeamentos são definidos. Gera como saída o esquema conceitual da coleção presente no banco de dados NoSQL e os mapeamentos dos campos consolidados com seus respectivos correspondentes.

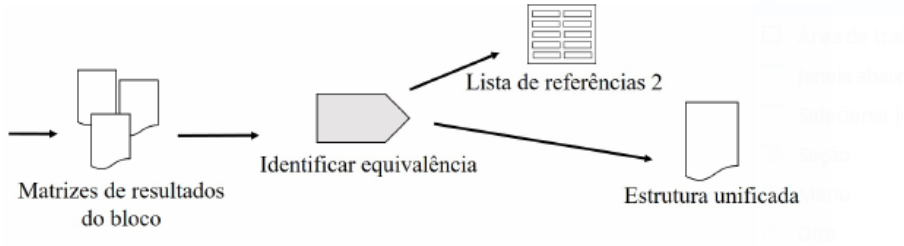
Cada uma das etapas descritas é composta por um subconjunto de atividades que visam atingir seu objetivo. Entretanto, a concepção do presente trabalho tem como foco as etapas 3 (Identificação de equivalências) e 4 (Representação da estrutura), que serão detalhadas a seguir.

2.3.2 Identificação de Equivalências

Segundo a autora, nessa etapa são executados cálculos e testes por bloco (segundo a estrutura do documento unificado gerado) e logo após os resultados são sintetizados em uma nova estrutura unificada. Durante essa etapa é também gerada a segunda lista de referências que,

basicamente, consiste em apresentar as relações de correspondência ($A = B$) encontradas entre os campos dos documentos da coleção. Uma visão geral da etapa é apresentada na figura 2.

Figura 2 – Etapa de identificação de equivalências.



Fonte: (MACHADO, 2017).

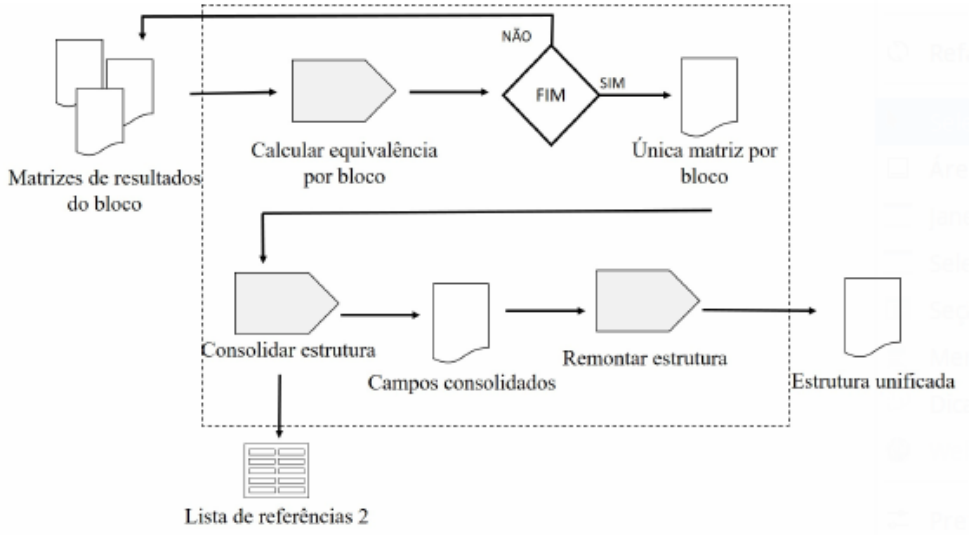
Esta etapa é subdividida em três atividades (Figura 3), sendo estas:

1. **Calcular equivalência por bloco:** é executada em blocos, até o fim do documento. Possui como entrada três matrizes de resultados, referentes a cada técnica aplicada na etapa anterior. São efetuados testes, comparações e cálculos que têm como saída uma única matriz constituída de valores 0 ou 1 referentes ao bloco processado;

2. **Consolidar estrutura:** possui como entrada as matrizes geradas a partir do processamento dos blocos. Os blocos são executados em ordem, sendo que, ao final, formam um único arquivo contendo os campos consolidados. Também gera uma segunda lista de referências, formada pelos termos e seus equivalentes;

3. **Remontar estrutura:** reorganiza os blocos em um único documento e/ou adiciona novamente os delimitadores para caso o documento tenha sido executado como um todo.

Figura 3 – Detalhamento da etapa de identificação de equivalências.



Fonte: (MACHADO, 2017).

As subatividades apresentadas serão detalhadas nas subseções seguintes. Entretanto, visto que o desenvolvimento do trabalho não inclui a subatividade *Calcular Equivalência por Bloco*, esta não será abordada, sendo abordados os detalhes a partir da subatividade *Consolidar Estrutura*.

2.3.2.1 Consolidar Estrutura

O processo de consolidar estrutura (Algoritmo 1) tem como objetivo ler as matrizes geradas no processo de Cálculo da média por bloco e selecionar os campos consolidados, guardando a equivalência em uma segunda lista de referências. Nesta etapa os elementos possuem valor binário, ou seja, zero ou um. Durante esta atividade, para os campos que são considerados equivalentes, é montada uma segunda lista de referências.

Algoritmo 1: Atividade consolidar estrutura

Entrada: Única matriz por bloco
Saída: Campos consolidados, Lista de referências 2

```
1  início
2      Carrega matriz de resultados do bloco;
3      Consolida todas as palavras da coluna;
4      para cada elemento acima da diagonal principal faça
5          se elemento da Matriz_ab é igual a 1 então
6              Mantém a primeira ocorrência do termo (palavra da linha);
7              Apaga a palavra da coluna dos campos consolidados;
8              Grava equivalência na Lista de referências 2;
9          fim
10     fim
11 fim
```

Fonte: (MACHADO, 2017).

2.3.2.2 Remontar Estrutura

A atividade (Algoritmo 2) tem por objetivo reorganizar e remontar o documento completo, adicionando os delimitadores corretamente. Caso o processo tenha sido executado em blocos, a atividade os organiza em ordem. Caso tenha sido executado como um único documento, os delimitadores que foram desconsiderados para a análise são recolocados.

Algoritmo 2: Atividade remontar estrutura

Entrada: Campos consolidados, Lista de referências 1 e Lista de referências 2
Saída: Estrutura consolidada

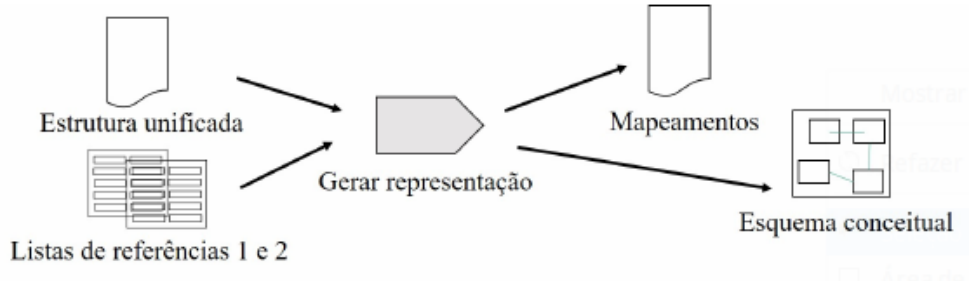
```
1  início
2      Carrega campos consolidados e listas;
3      selecione modo de execução faça
4          caso único bloco faça
5              Escolhe o documento de origem para referência (com maior número de
                blocos);
6              para cada campo do documento de referências faça
7                  se campo está consolidado então mantém;
8                  senão verifica os correspondentes que estão consolidados e os
                    substitui;
9              fim
10             Os campos consolidados que não estão no documento de referência, são
                incluídos em um novo objeto delimitado por chaves;
11         fim
12         caso por blocos faça
13             Remonta os blocos em ordem mantendo os símbolos delimitadores de
                origem;
14         fim
15     fim
16 fim
```

Fonte: (MACHADO, 2017).

2.3.3 Representação da Estrutura

A etapa de representação da estrutura, apontada pela figura 4, tem por objetivo, a partir da estrutura unificada, aplicar a conversão desta para um diagrama, bem como gerar mapeamentos dos campos e seus equivalentes. A atividade principal desta etapa é gerar representação. O arquivo de entrada apresenta os campos juntamente com os delimitadores que auxiliam tanto no processo de montar o esquema conceitual como para representar os mapeamentos com a notação JSONPath.

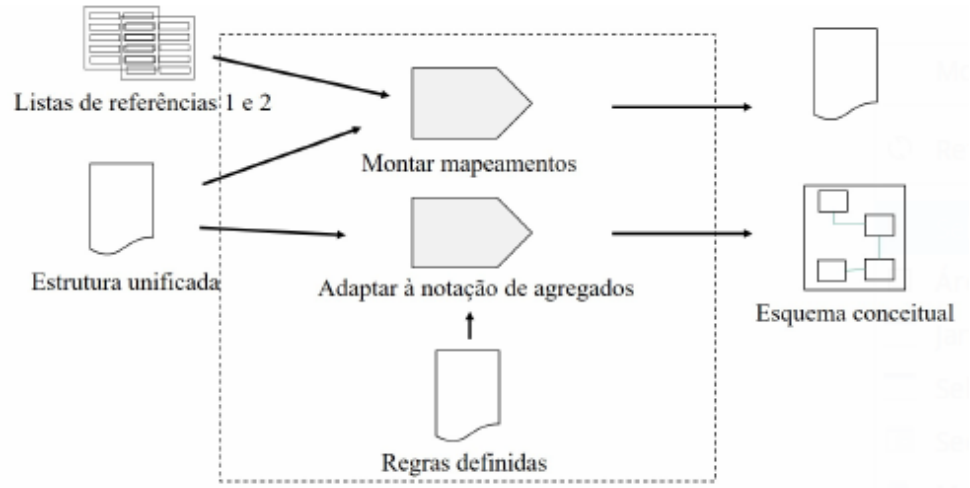
Figura 4 – Processo de representação de estrutura.



Fonte: (MACHADO, 2017).

A atividade principal é gerar representação. Esta é dividida em duas subatividades (Figura 5), conforme especificado a seguir.

Figura 5 – Detalhamento do processo de representação de estrutura.



Fonte: (MACHADO, 2017).

De acordo com a Figura 5, pode-se observar:

- **Listas de referências 1 e 2:** artefatos gerados durante todo o processo de extração. A primeira lista é gerada na atividade principal (extrair campos) e a segunda é saída da atividade principal de identificação de equivalências;
- **Atividade montar mapeamentos:** atividade que tem como objetivo consultar ambas as listas para inferir novas equivalências entre os campos, montando uma tabela final com o caminho do termo consolidado representado pela notação JSONPath;
- **Regras definidas:** este artefato apresenta as regras definidas. Estas são empregadas du-

rante a identificação dos blocos em objetos ou arrays que auxiliam na definição de entidades, atributos e relacionamentos do esquema conceitual;

- **Atividade adaptar à notação de agregados:** atividade que tem por objetivo montar a representação através da notação IDEF1X adaptada ao modelo NoSQL proposta por (JOVANOVIC; BENSON, 2013) aplicando as regras definidas.

2.3.3.1 Montar Mapeamentos

Como a proposta do processo abrange a consolidação de um esquema para que seja possível realizar futuras consultas, permitindo assim o acesso de forma unificada à base, torna-se necessário guardar os mapeamentos. Segundo a autora, tais informações relacionam um termo consolidado aos seus correspondentes em outros documentos pertencentes à mesma coleção, para que ao acessar determinado termo, tenha-se conhecimento de quais outros estão relacionados ao mesmo. Dessa maneira, a atividade, descrita no Algoritmo 3, tem por objetivo, com base nas listas de referências e na estrutura unificada, identificar possíveis equivalências internas e montar uma tabela com as respectivas informações: termo consolidado, seu correspondente e documento de origem do correspondente. Para representação do termo consolidado é utilizada a notação JSONPath, que especifica caminhos para acesso a atributos e objetos dentro de um documento JSON.

Algoritmo 3: Atividade montar mapeamentos

Entrada: Listas de referências 1, Lista de referências 2 e Estrutura consolidada
Saída: Mapeamentos

```
1  início
2      Carrega ambas listas de referências e estrutura unificada;
3      para cada campo consolidado ∈ estrutura unificada faça
4          Procura-o na estrutura unificada montando a referência JSONPath;
5          Verifica na lista de referências 2 quais termos são equivalentes;
6          Verifica equivalências do tipo A = B e B = C ⇒ A = C;
7          para cada termo equivalente faça
8              Verifica documento de origem na Lista de referências 1;
9          fim
10     fim
11     Grava mapeamentos;
12 fim
```

Fonte: (MACHADO, 2017).

2.3.3.2 Regras Definidas

De acordo com a autora, este artefato tem por objetivo fornecer as regras definidas para a conversão da estrutura unificada em um esquema conceitual. De modo geral, este artefato visa identificar entidades, atributos e relacionamentos para esquemas em NoSQL. No contexto do referido trabalho, uma entidade pode corresponder a um bloco e os relacionamentos identificados são do tipo um para um ($1:1$) e um para muitos ($1:N$). Na referida proposta, por exemplo, ao encontrar o início de um objeto ou array infere-se uma nova classe de relacionamento $1:N$. Regras. As regras para construção do esquema conceitual são descritas e numeradas por C_n onde n aponta o número da regra. Inicialmente é identificado o delimitador que marca o início do documento d, que deve ser encerrado ao encontrar o mesmo símbolo de fechamento. As regras definidas são como segue:

- C_1: cada "{" de nome "String" gera uma nova classe *C* de nome "String" e cria um relacionamento do tipo $0:1$. A classe *C* só é fechada ao encontrar um "}" e o número de chaves abertas corresponda ao número de chaves fechadas;

- C_2: cada campo de valor que seja de um tipo primitivo (se configurando como $var = valor$) gera um atributo "*att*" na classe *C*. Se este já existe nada acontece, senão o atributo é criado;

- C_3: cada "[" de nome "String" gera uma nova classe *C* de nome "String" com atributo de nome *att* e cria um relacionamento $0:N$. A classe *C* só é fechada quando encontrar "]" e o número de colchetes abertos corresponda ao número de colchetes fechados;

- C_4: cada "[{" gera uma nova classe c de nome String e cria um relacionamento $0:N$. Ela aplica C_2 novamente. Quando encontrar "}", volta ao início da classe *C* para testar novamente os atributos *att*. Se o atributo já existe nada acontece, senão ele é criado dentro da classe *C*. A classe só é encerrada quando for encontrado "]";

Conforme observado, a regra C_1 é aplicada para gerar uma nova classe cujo nome é a String que antecede a abertura da chave. Esta regra ocorre ao identificar o início de um objeto. Este relacionamento é do tipo $0:1$. A regra C_2 transforma em atributos os campos nas classes correspondentes. Ela é executada após a identificação de início de uma classe. A regra C_3 ocorre quando é aberto um array com enumeração de itens, sendo gerada uma nova classe com um único atributo de nome *att*. A regra C_4 também gera uma nova classe cujo nome é a String

que antecede a abertura do colchete. Este relacionamento é do tipo $1 : n$. Este caso, porém, diferencia-se por conter objetos aninhados. Assim os atributos desta classe são criados através de iterações nos objetos, ou seja, é aplicada C_2 cada vez que encerrar um objeto interno. Desse modo, o processo retorna ao início do objeto para verificar se o atributo *att* já existe. Caso seja diferente, ele é acrescentado ao esquema.

2.3.3.3 Adaptar à Notação de Agregados

Esta atividade (Algoritmo 4) tem por objetivo montar a representação do esquema conceitual, artefato final do processo proposto pela autora. Para tal, a atividade aplica as regras de conversão definidas sobre o arquivo de entrada. Os relacionamentos entre coleções, isto é, do tipo muitos para muitos ($n : n$), não estão no escopo do referido trabalho, pois o mesmo trata apenas das relações entre documentos e não entre coleções. O arquivo de entrada apresenta os campos, juntamente com os delimitadores como chaves e colchetes, que auxiliam a identificar entidades, atributos e relacionamentos, gerando uma representação na notação IDEF1X própria ao modelo NoSQL de acordo com JOVANOVIC; BENSON (2013). Esta notação parte de um nó raiz sendo os demais aninhados no mesmo.

Algoritmo 4: Atividade adaptar à notação de agregados

Entrada: Listas de referências 1, Lista de referências 2 e Estrutura consolidada
Saída: Mapeamentos

```
1  início
2      Carrega arquivo com a estrutura unificada;
3      Primeiro objeto é considerada entidade raiz;
4      selecione evento faça
5          caso { faça Aplica regra C1;
6          caso value faça Aplica regra C2;
7          caso [ faça Aplica regra C3;
8          caso [{ faça Aplica regra C4;
9      fim
10     Monta a representação visual;
11 fim
```

Fonte: (MACHADO, 2017).

3 DESENVOLVIMENTO

Neste capítulo será apresentada a proposta da ferramenta desenvolvida, também serão abordados os detalhes da implementação dos algoritmos, bem como o funcionamento da ferramenta em si e detalhes sobre os artefatos gerados durante o processo de geração do esquema.

3.1 PROPOSTA DA FERRAMENTA

O software no presente trabalho é desenvolvido na linguagem de programação Java e armazena os dados em arquivos de texto.

Trata-se de uma aplicação java que irá processar arquivos JSON e artefatos provindos do processo de extração de esquemas proposto por Machado (2017) para a geração de uma lista de mapeamentos e um esquema conceitual, ambos em arquivos de texto. O principal objetivo do software é viabilizar o acesso unificado a bases de dados JSON, através da disponibilização de informações estruturais da base (esquema).

Este trabalho usa como base o processo para fazer a extração do esquema implícito de uma base de dados NoSQL orientada a documentos no formato JSON ou um determinado conjunto de fontes de dados JSON que se tenha o objetivo de acessar de maneira unificada, desde que pertençam ao mesmo domínio.

A aplicação é constituída por implementações dos algoritmos referentes às seguintes atividades, respectivamente:

- consolidar estrutura;
- remontar estrutura;
- detectar mapeamentos;
- adaptar à notação de agregados.

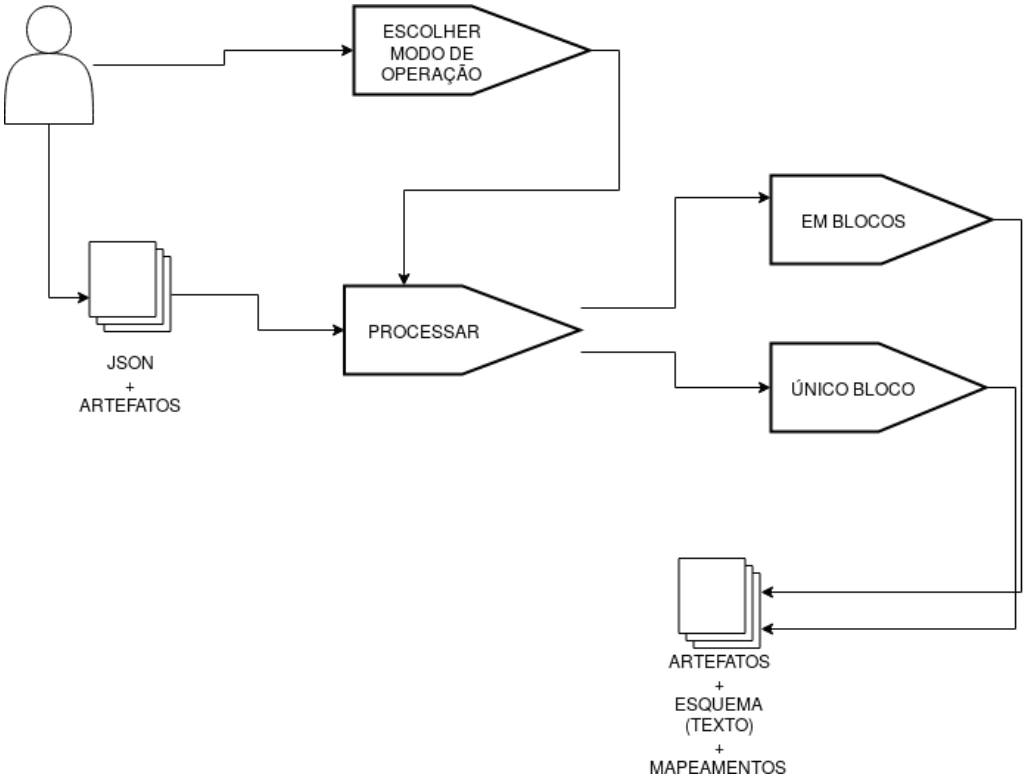
Para que possa ser executada, a ferramenta deverá contar com todos os arquivos necessários para isto (arquivos JSON, lista de referências 1, estrutura unificada e matriz(es) única(s) de resultados). Em caso de falta de um destes arquivos, o processo será cancelado. A ferramenta proposta irá suportar dois modos de aplicação: em blocos ou em um único bloco. Para operar no modo de bloco único, onde todos os campos dos arquivos JSON são testados entre si, independente da hierarquia, o usuário deverá indicar esta opção na interface da aplicação. Para

operar no modo em blocos, será necessário o usuário adicionar à pasta de artefatos (*artefatos-entrada*) um documento de texto contendo as informações sobre os blocos equivalentes entre os documentos de entrada.

3.2 ARQUITETURA

Nesta seção serão apresentados os detalhes sobre a arquitetura da aplicação desenvolvida no trabalho, segundo as Figuras 6 e 7.

Figura 6 – Arquitetura da aplicação



Fonte: Autor

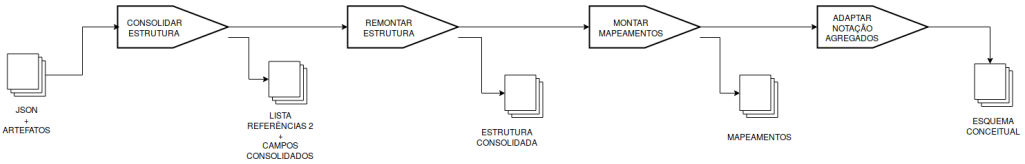
A atividade *processar em blocos* se refere ao modo de processo semi-automático, onde um especialista do domínio intervém no processo de geração do esquema referente ao banco. Esta atividade recebe como entrada os arquivos JSON, os artefatos *lista de referências 1* e *matriz única de resultados*, gerados anteriormente, e uma lista auxiliar, chamada *estrutura do especialista*, onde está expressa a equivalência dos blocos definida pelo especialista.

A atividade *processar único bloco* se refere ao modo de processo automático, onde são realizados testes entre todos os blocos do dos documentos de entrada como se estivessem em um mesmo nível. Esta atividade recebe como entrada os os arquivos JSON e os artefatos *lista*

de referências 1 e *matriz única de resultados*.

Ambas as atividades, apesar das diferenças referentes aos arquivos de entrada e ao algoritmo para *remontar estrutura* utilizado, são efetuadas de maneira semelhante, ou seja, obedecem uma mesma sequencia de passos para sua execução, segundo a Listagem 7.

Figura 7 – Passos e artefatos gerados



Fonte: Autor

A atividade *consolidar estrutura* recebe como entrada a *matriz única de resultados*. No modo de operação por blocos, recebe n matrizes para n blocos, sendo uma matriz para cada bloco. No modo de execução por único bloco, recebe apenas uma matriz. Esta atividade tem como artefatos de saída a *lista de referências 2* e a *lista de campos consolidados*.

A atividade *remontar estrutura* tem como entrada os artefatos *lista de referências 1*, *lista de referências 2* e *lista de campos consolidados*. Esta atividade possui critérios diferentes (ou seja, é executada por algoritmos distintos) para cada modo de execução. No modo *por blocos*, recebe adicionalmente como entrada a estrutura auxiliar *estrutura do especialista*, que é usada para a recuperação das informações sobre os campos que são "comuns" ao bloco, ou seja, em um conjunto de documentos JSON com a divisão lógica em n blocos, todo o documento que possua o bloco b possui também o campo c. Campos que se encaixam nessa regra não são incluídos na *lista de referências 1*, pois esta armazena a referência do campo c que seja única ao bloco, ou seja, apenas um subconjunto menor do conjunto de documentos que possuam o bloco b possui o campo c. Para o modo de execução de *bloco único*, recebe como entrada apenas a *lista de campos consolidados* e as *listas de referências 1* e *2*. Esta atividade gera como artefato de saída a *estrutura consolidada*.

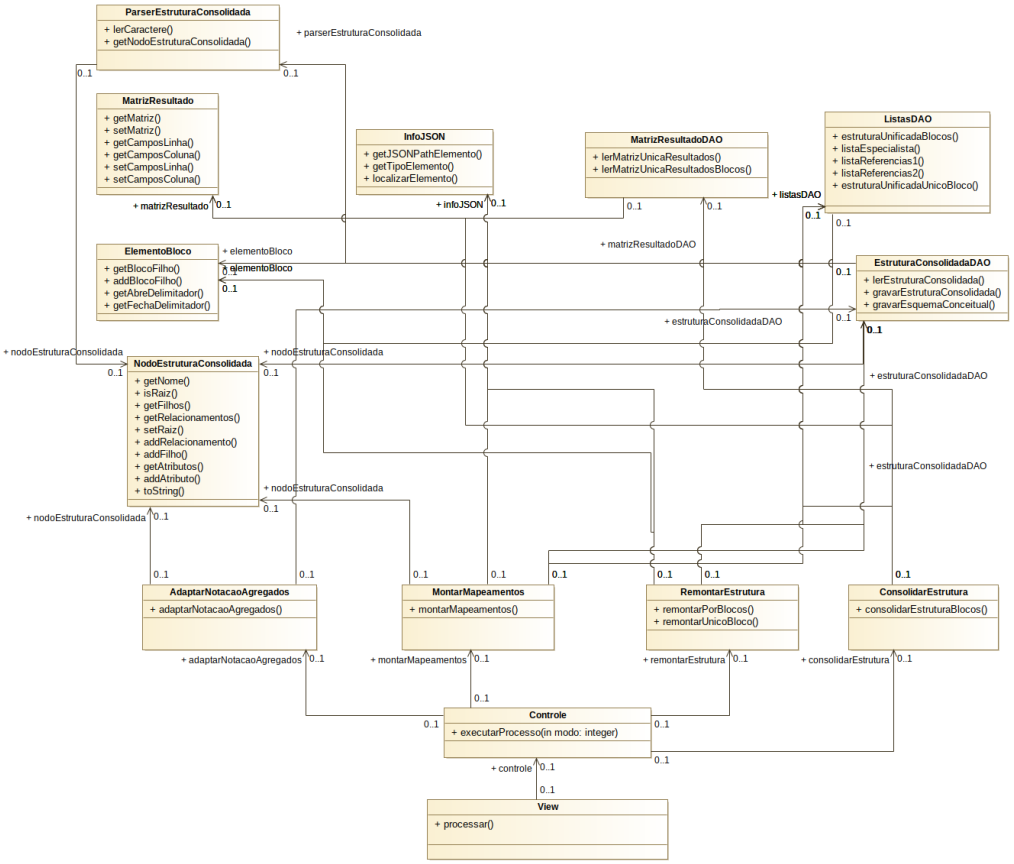
A atividade *montar mapeamentos* recebe como entrada os artefatos *lista de referências 1*, *lista de referências 2* e *estrutura consolidada*. Em ambos os modos de execução, é executada da mesma forma. Gera como saída o artefato *mapeamentos*.

A atividade *adaptar à notação de agregados* recebe como entrada o artefato *lista consolidada* e gera como saída o artefato contendo a estrutura consolidada expressa conforme a notação IDEF1X, no caso da presente aplicação, este artefato é um arquivo de texto.

3.3 ESTRUTURA DA APLICAÇÃO

A implementação da ferramenta obedece o padrão de desenvolvimento MVC, onde a camada de *Modelo* é composta pelas classes controladas pela classe *Controle* (e as classes controladas por essas, consequentemente), a camada Controle é composta pela classe *Controle* e a camada *View* é composta pela classe *View*.

Figura 8 – Diagrama de classes da implementação realizada



Fonte: Autor

A classe *View* é composta pela subclasse *main*, e executa apenas os comandos de entrada (por teclado) do usuário e repassa esses parâmetros ao método *executarProcesso* da classe *Controle*, também exibe as mensagens de erro ou sucesso, como mostra a Listagem 1.

Listagem 1 – Classe *View*

```
1 public class View {
2     public static void main(String[] args) throws
          FileNotFoundException, IOException {
3         Controle c = new Controle();
4         Scanner s = new Scanner(System.in);
5
```

```
6           System.out.println("\n\n\nEXTRACAO DE ESQUEMAS A
               PARTIR DE ARQUIVOS JSON\n\n\n");
7           System.out.println("Escolha o modo de operacao (1 = em
               blocos;"
8                   + " 2 = unico bloco)\n\n");
9           int modo = s.nextInt();
10          try {
11              c.executarProcesso(modo);
12          } catch (Exception ex) {
13              System.out.println(ex.getMessage());
14          }
15          System.out.println("Extracao de esquemas a partir de
               arquivos JSON"
16                  + " executada. Verifique os arquivos de saida
                       gerados.");
17      }
18  }
```

Fonte: Autor

A classe *Controle*, por sua vez, é responsável por todo o fluxo de execução da aplicação, pois, a partir do valor que o método *executarProcesso* recebe, determina quais algoritmos serão executados, se serão executados no modo de execução *em blocos*, através do método *processoEmBlocos* ou em *único bloco*, através do método *processoUnicoBloco*, como mostra a Listagem 2. Caso o valor do parâmetro *modo* for um número diferente de 1 ou 2, o método dispara uma exceção, interrompendo a execução da aplicação e informando ao usuário que a opção selecionada é incorreta.

Listagem 2 – Método *executarProcesso*

```
1  public void executarProcesso(int modo) throws Exception {
2      if (modo == 1) {
3          this.processoEmBlocos();
4      } else if (modo == 2) {
5          this.processoUnicoBloco();
6      } else {
7          throw new Exception("Voce pode apenas selecionar um
               modo "
8                   + "(1 = blocos, 2 = unico bloco)");
9      }
10 }
```

Fonte: Autor

Os métodos *processoEmBlocos* (Listagem 3) e *processoUnicoBloco* (Listagem 4) executam todo o processo de extração de esquemas, sendo o primeiro referente ao modo de execu-

ção *em blocos* e o segundo em um *único bloco*, respectivamete. Entretanto, somente a classe *RemontarEstrutura* possui um método possui versões diferentes para cada modo de execução.

Listagem 3 – Método *processoEmBlocos*

```
1  private void processoEmBlocos() throws IOException, Exception {
2      ConsolidarEstrutura cons = new ConsolidarEstrutura();
3      RemontarEstrutura rem = new RemontarEstrutura();
4      MontarMapeamentos mm = new MontarMapeamentos();
5      AdaptarNotacaoAgregados an = new AdaptarNotacaoAgregados();
6      //Consolidar Estrutura
7      cons.consolidarEstruturaBlocos();
8
9      // Remontar Estrutura
10     rem.remontarPorBlocos();
11
12     //Gerar mapeamentos
13     mm.montarMapeamentos();
14
15     //Adaptar a notacao de agregados (esquema conceitual)
16     an.adaptarNotacaoAgregados();
17 }
```

Fonte: Autor

O método *processoEmBlocos* (Listagem 3) consiste em inicializar as classes que representam as atividades e executá-las em sequencia, chamando os métodos de cada atividade em sua versão no modo de execução *em blocos*.

Listagem 4 – Método *processoUnicoBloco*

```
1  public void processoUnicoBloco() throws IOException, Exception
       {
2      ConsolidarEstrutura cons = new ConsolidarEstrutura();
3      RemontarEstrutura rem = new RemontarEstrutura();
4      MontarMapeamentos mm = new MontarMapeamentos();
5      AdaptarNotacaoAgregados an = new AdaptarNotacaoAgregados();
6      //Consolidar Estrutura
7      cons.consolidarEstruturaBlocos();
8
9      // Remontar Estrutura
10     rem.remontarUnicoBloco();
11
12     //Gerar mapeamentos
13     mm.montarMapeamentos();
14
15     //Adaptar a notacao de agregados (esquema conceitual)
16     an.adaptarNotacaoAgregados();
17 }
```

Fonte: Autor

O método *processoUnicoBloco* (Listagem 4), assim como o método *processoEmBlocos*, consiste em inicializar as classes que representam as atividades e executar os métodos em sequencia.

Os métodos utilizados tanto em *processoEmBlocos* (Listagem 3) quanto em *processoUnicoBloco* (Listagem 4) são pertencentes às classes que representam as atividades propriamente ditas, nessas cloasses que se encontram os algoritmos citados no capítulo da revisão bibliográfica e detalhados nos próximos capítulos. Os métodos públicos dessas classes possuem um padrão de execução, ou seja, cada método chamado segue a seguinte sequencia básica de ações, com poucas variações:

1. carregar artefatos de entrada, via classes DAO;
2. executar algoritmo (caracterizado como um método auxiliar), recebendo via parâmetro os artefatos carregados;
3. gravar os artefatos de saída gerados pelo algoritmo em arquivo(s), via classes DAO.

Como exemplo será mostrado o método *montarMapeamentos* (Listagem 5), da classe *MontarMapeamentos*.

Listagem 5 – Método *montarMapeamentos* (exemplo)

```
1  public void montarMapeamentos() throws FileNotFoundException,
       Exception {
2      ListasDAO l = new ListasDAO();
3      EstruturaConsolidadaDAO e = new EstruturaConsolidadaDAO();
4      //Carrega os artefatos
5      //Estrutura consolidada
6      NodoEstruturaConsolidada estrCons =
           e.lerEstruturaConsolidada();
7      List<String> listaCamposConsolidados =
           estrCons.getListaNomes();
8      List<String> listaJsonPathCamposConsolidados
9              = estrCons.getListaJsonPath();
10     //Lista de referencias 1
11     List<List<String>> listaRef1 = l.lerListaReferencias1();
12
13     //Lista de referencias 2
14     List<String[]> listaRef2 = l.lerListaReferencias2();
15
16     //Monta os mapeamentos
```

```
17      List<String[]> listMapeamentos =
           getMapeamentos(listaCamposConsolidados,
           listaJsonPathCamposConsolidados, listaRef2, listaRef1);
18
19      //Grava a tabela de mapeamentos
20      l.gravarMapeamentos(listMapeamentos);
21  }
```

Fonte: Autor

No método *montarMapeamentos* (Listagem 5), o passo (1) é executado ao carregar a *estrutura consolidada* (Linha 6) e as *listas de referências 1 e 2* (linhas 11 e 14), o passo (2) é executado ao obter a lista de mapeamentos através do método *getMapeamentos* (linha 17) e por fim, o passo (3) é executado ao gravar a lista de mapeamentos obtida (linha 20).

3.4 DADOS DA APLICAÇÃO

Os dados presentes na aplicação desenvolvida neste trabalho consistem, basicamente, nos itens:

1. arquivos de entrada no formato JSON;
2. artefatos provindos de atividades anteriores (*primeira lista de referências* e *matrizes únicas de resultados*);
3. artefatos gerados durante o processo (*lista de referências 2*, *campos consolidados*, *estrutura consolidada*, *mapeamentos* e *esquema conceitual*);
4. artefatos adicionados para atender uma necessidade em específico (*estrutura do especialista*).

Tanto os arquivos de entrada quanto os artefatos de (2) foram obtidos a partir da pasta pública da autora, que contém os artefatos e resultados do estudo de caso do processo proposto. O artefato de (4) foi adicionado especificamente ao modo de operação em blocos.

Nas próximas subseções deste capítulo será detalhada a coleta, o pré-processamento e a a estrutura dos dados presentes na aplicação.

3.4.1 Arquivos JSON

Os arquivos JSON são referências exportadas a partir de diferentes trabalhos científicos e artigos presentes em bibliotecas digitais como DBLP[1], Scopus[2], PubMed[3] e Bibsonomy[4]. Estas fontes foram escolhidas pela autora por permitirem exportar referências em formatos XML e CSV, sendo possível, consequentemente, serem convertidas para o formato JSON. Os arquivos foram escolhidos de modo aleatório, uma vez que o processo desconsidera os valores associados aos campos durante o processo de extração de esquema.

Para o modo de execução por blocos, foi utilizado o documento *Caso1-dataset-bkp.json*, mostrado na Listagem 6, presente na pasta pública da autora. Por motivos de tamanho, o documento não será exibido totalmente. As reticências(...) indicam continuidade do conteúdo.

Listagem 6 – Arquivo (parcial) *Caso1-dataset-bkp.json*

```
{
"dblp": {
        "article": {
                "@key": "journals/ijbpim/AlmeidaBF15",
                "@mdate": "2016-01-05",
                "author": [
                        "Rafael Almeida",
                        "Jorge Bernardino",
...
}
{
"dblp": {
        "proceedings": {
                "@key": "conf/naa/2016",
                "@mdate": "2017-04-13",
                "editor": [
                        "Ivan Dimov",
                        Istvan Farago",
...
}
{
"dblp": {
        "inproceedings": {
                "@key": "conf/ACMse/ParkerPV13",
                "@mdate": "2015-05-07",
                "author": [
                        "Zachary Parker",
```

[1] http://dblp.uni-trier.de/
[2] https://www.scopus.com/home.uri
[3] https://www.ncbi.nlm.nih.gov/pubmed
[4] https://www.bibsonomy.org/

```
                        "Scott Poe",
...
}
{
        "Authors": "Gonzalez", Aparicio M.T., Ogunyadeka A.,
           Younas M., Tuya J., Casado R.",
        "Title": "Transaction processing in consistency-aware
           user's applications deployed on NoSQL databases"
...
}
```

Fonte: (MACHADO, 2017)

O documento *Caso1-dataset-bkp.json* (Listagem 6) continha quatro estruturas JSON, cada estrutura representava um documento diferente. O documento foi então dividido em quatro documentos individuais, *doc1.json*, *doc2.json*, *doc3.json* e *doc4.json*. A Listagem 7 mostra o arquivo *doc1.json*, um dos documentos obtidos a partir do *Caso1-dataset-bkp.json*.

Listagem 7 – *doc1.json*, obtido a partir do *Caso1-dataset-bkp.json*

```
{
    "dblp": {
            "article": {
                    "@key": "journals/ijbpim/AlmeidaBF15",
                    "@mdate": "2016-01-05",
                    "author": [
                            "Rafael Almeida",
                            "Jorge Bernardino",
                            "Pedro Furtado"
                    ],
                    "title": "\nTesting SQL and NoSQL
                       approaches for big data warehouse
                       systems.\n",
...
            }
    }
}
```

Fonte: (MACHADO, 2017)

Ainda no modo de execução em blocos, a autora, para a execução do processo, subdividiu os documentos em blocos, de forma que as comparações fossem feitas somente entre campos pertencentes a um grupo de campos (blocos) considerados equivalentes. Entretanto, essa subdivisão em blocos não gerou novos arquivos, como foi a divisão do arquivo *Caso1-dataset-bkp.json*, tendo sido esta apenas definida na aplicação, através do artefato *estrutura do especialista*, que será detalhado nas próximas subseções.

Para o modo de execução em um único bloco, foi utilizado o documento *Caso2-dataset-bkp.json* (Listagem 8), presente na pasta pública da autora.

Listagem 8 – Parte do arquivo *Caso2-dataset-bkp.json*

```
{
 "Source": {
  "Author": {
   "Author": {
    "NameList": {
     "Person": {
      "First": "Rick",
      "Last": "Cattell"
...
}
{
  "dblp": {
    "article": {
      "@key": "journals/ijbpim/AlmeidaBF15",
      "@mdate": "2016-01-05",
      "author": [
        "Rafael Almeida",
        "Jorge Bernardino",
        "Pedro Furtado"
...
}
{
  "PubmedArticle": {
    "MedlineCitation": {
      "@Status": "In-Process",
      "@Owner": "NLM",
      "PMID": {
        "@Version": 1,
        "$": 27936191
      },
      "DateCreated": {
        "Year": 2016,
        "Month": 12,
        "Day": "09"
      },
...
}
{
  "Authors": "Reniers V., Rafique A., Van Landuyt D., Joosen
     W.",
  "Title": "Object-NoSQL Database Mappers: a benchmark study
     on the performance overhead",
  "Year": 2017,
  "Source title": "Journal of Internet Services and
```

```
      Applications",
  "Volume": 8,
  "Issue": 1,
...
}
```

Fonte: (MACHADO, 2017)

O documento *Caso2-dataset-bkp.json* (Listagem 8), assim como o *Caso1-dataset-bkp.json* continha quatro estruturas JSON, cada estrutura representa um documento diferente. Também foi dividido em quatro documentos individuais (*doc1.json*, *doc2.json*, *doc3.json* e *doc4.json*) para ser processado. A Listagem 9 mostra o arquivo *doc1.json*, um dos documentos obtidos a partir do *Caso2-dataset-bkp.json*.

Listagem 9 – *doc1.json*, obtido a partir do *Caso2-dataset-bkp.json*

```
{
    "Source": {
        "Author": {
            "Author": {
                "NameList": {
                    "Person": {
                        "First": "Rick",
                        "Last": "Cattell"
                    }
                }
            }
        },
        "BIBTEX_KeyWords": "cloud-computi       ng, nosql",
        "City": "New York, NY, USA",
...
    }
}
```

Fonte: Autor

Neste caso, os documentos resultantes não foram subdivididos em blocos, sendo que todas as operações foram feitas entre todos os campos de todos os blocos. Tendo sido apresentados os arquivos de entrada, será abordado na próxima subseção sobre o artefato *Matriz única de resultados por bloco*.

3.4.2 Matriz única de resultados por bloco

A matriz única de resultados consiste é, na aplicação desenvolvida neste trabalho, um dos primeiros artefatos a ser processado, na atividade *Consolidar Estrutura*. Nesse caso, ela

foi subdividida em dois arquivos de texto CSV, um arquivo contendo a matriz propriamente dita (*matriz-unica-resultados-X.csv*), como mostra a Listagem 11, e outro arquivo contendo os nomes dos campos a serem consolidados (*matriz-unica-resultados-campos-X.csv*), exibido na Listagem 10, onde X é equivalente ao número do bloco menos um, por exemplo, para o primeiro bloco (bloco 1), X assume o valor 0. Importante ressaltar que a ordem dos valores no arquivo contendo os campos corresponde à ordem dos valores no arquivo contendo a matriz, ou seja, o primeiro item da lista de nomes equivale à primeira linha e à primeira coluna do arquivo da matriz, o segundo, à segunda linha e segunda coluna, a *n-ésima* palavra, à *n-ésima* linha e *n-ésima* coluna do arquivo de matrizes.

Listagem 10 – Matriz única de resultados - *matriz-unica-resultados-campos-X.csv*

```
1 article; proceedings; inproceedings
```

Fonte: Autor

Listagem 11 – Matriz única de resultados - *matriz-unica-resultados-X.csv*

```
1 1;0;0
2 0;1;1
3 0;0;1
```

Fonte: Autor

Para a execução por blocos, o número de matrizes equivale ao número (máximo) de blocos em que os arquivos forem divididos. Ou seja, se em determinado caso for definida uma estrutura de divisão em 10 blocos, deverão existir 10 matrizes, uma por bloco. Para a execução por bloco único, deverá conter apenas uma matriz.

O conteúdo das matrizes foi obtido a partir dos estudos de caso da autora do processo, para ambos os modos de execução. Uma vez apresentada a *Matriz Única de Resultados por Bloco*, será abordado sobre a *Lista de Referências 1*.

3.4.3 Lista de Referências 1

Trata-se de um arquivo de texto CSV (nomeado *lista-referencias1.csv*), separado por ponto e vírgula, contendo nomes de campos e o(s) nome(s) do(s) documento(s) onde há ocorrências destes campos. Por exemplo, na Listagem 12, a linha 3 da lista indica que o campo *author* possui ocorrências nos documentos *doc1.json*, *doc2.json* e *doc3.json*. As listas utilizadas neste trabalho são ambas originárias dos estudos de caso da autora, pois esta é gerada após a execução da atividade *Mesclar Estrutura*, no caso, não implementada no presente trabalho.

Listagem 12 – Parte do arquivo *lista-referencias1.csv*

```
source;doc1.json
@validyn;doc3.json
author;doc1.json; doc2.json; doc3.json
affiliation;doc3.json
namelist;doc1.json
language;doc3.json
...
```

Fonte: Autor

A sua estrutura não sofre alterações, independentemente do modo de execução adotado (em blocos ou em único bloco). Para a execução em blocos, de acordo com a autora, a lista contém apenas os elementos considerados distintos aos blocos, ou seja, dentro de um bloco b, um determinado elemento e (atributo, objeto ou array) incide parcialmente no bloco, ou seja, nos documentos que contenham o bloco b nem em todos existem ocorrências de e. Caso contrário, o elemento não é adicionado à lista, por ser comum a todos os documentos que contenham o bloco b. Para a execução em único bloco, todos os elementos existentes em todos os arquivos analisados são adicionados à lista. Tendo sido apresentado a *lista de referências 1*, será detalhada a estrutura *campos consolidados*.

3.4.4 Campos Consolidados

Trata-se de um arquivo de texto CSV (*campos-consolidados.csv*), separado por ponto e vírgula, gerado pela atividade *consolidar estrutura*, nele estão contidos os nomes dos campos considerados únicos dentro do conjunto de documentos JSON de entrada (Listagem 13).

Listagem 13 – *campos-consolidados.csv* - execução em blocos

```
dblp
article; proceedings
@key; @mdate; author; title; pages; year; volume; journal;
   number; ee; url; editor; booktitle; series; Page start;
   Page end; Page count; Cited by; DOI; Link; EID
@href; $
```

Fonte: Autor

Para o modo de execução em blocos, o artefato assume a configuração como apresentada na Listagem 13, onde cada linha do documento equivale a um conjunto de campos consolidados de um bloco, ou seja, a primeira linha refere-se aos campos consolidados do primeiro bloco, a segunda, do segundo bloco, a *n-ésima*, do *n-ésimo* bloco. No modo de execução em um único bloco, o artefato possui apenas uma linha preenchida (Listagem 14).

Listagem 14 – *campos-consolidados.csv* (parcial) - execução em único bloco

```
1 source; author; namelist; person; first; last;
    bibtex_keywords; city; journalname; month; pages;
    publisher; sourcetype; standardnumber; tag; title; url;
    volume; year; dblp; article; @key; @mdate; journal; number;
    ee;...
```

Fonte: Autor

Apresentado foi o artefato *campos consolidados*. Na próxima subseção será abordado sobre o artefato *lista de referências 2*.

3.4.5 Lista de Referências 2

A segunda lista de referências é um artefato gerado pela atividade *consolidar estrutura*, assume o formato de um arquivo CSV (*lista-referencias2.csv*), separado por ponto e vírgula, que contém tuplas formadas pelos termos correspondentes, onde o primeiro termo é o termo consolidado e o segundo, o termo considerado equivalente pelo algoritmo da atividade. Como mostrado na Listagem 15, por exemplo, entende-se que o campo *author* é o campo consolidado, e o campo *authors* foi considerado equivalente a ele. Tanto no modo de operação em blocos quanto no modo de único bloco, a lista tem o mesmo formato.

Listagem 15 – Lista de Referências 2

```
1 author; authors
2 number; issue
3 proceedings; inproceedings
```

Fonte: Autor

Após abordar sobre a *lista de referências 2*, será abordado, na próxima subseção, sobre a *estrutura consolidada*.

3.4.6 Estrutura Consolidada

Trata-se de um arquivo texto (*estrutura-consolidada.txt*), que possui o conteúdo da *estrutura unificada* após a execução da tarefa *remontar a estrutura*, adicionalmente, com os delimitadores recolocados, conforme a Listagem 16. Neste trabalho, optou-se por manter o ponto e vírgula após o nome do elemento, antes dos delimitadores, com o intuito de facilitar o reconhecimento de nomes contendo espaços.

Listagem 16 – Estrutura Consolidada

```
dblp;{
    article;
    proceedings;{
        @key;
        @mdate;
        author;[]
        title;
        pages;
        year;
        volume;
        journal;
        number;
        ee;
        url;
        editor;[]
        booktitle;
        series;{
            @href;
            $;
        }
        isbn;[]
        crossref;
        source title;
        art no;
        Page start;
        Page end;
        Page count;
        Cited by;
        DOI;
        Link;
        EID;
    }
}
```

Fonte: Autor

Em ambos os modos de execução (*por blocos* ou em *único bloco*), a lista assume a mesma configuração, apesar de ser montada por algoritmos diferentes para cada modo de execução. Apresentada a *Estrutura Consolidada*, será apresentada na próxima subseção a estrutura *mapeamentos*.

3.4.7 Mapeamentos

A lista de mapeamentos é um arquivo de texto no formato CSV, chamado *mapeamentos.csv* (Listagem 17), separado por ponto e vírgula, gerado após a execução da atividade *montar*

mapeamentos. Sua estrutura é uma tabela, onde cada linha contém:

1. caminho JSONPath para o elemento consolidado segundo a *estrutura consolidada*;
2. caminho JSONPath para o elemento equivalente segundo a estrutura do seu documento JSON de origem;
3. nome do documento de origem do elemento equivalente.

Listagem 17 – Arquivo *mapeamentos.csv*

```
1 $.dblp.proceedings.author; $..authors; doc4.json
2 $.dblp.proceedings.number; $..issue; doc4.json
3 $.dblp.proceedings; $..inproceedings; doc3.json
```

Fonte: Autor

Por exemplo, a primeira linha do documento de mapeamentos apresentado na Listagem 17 representa o termo consolidado *author*, que é filho dos nodos *dblp* e *proceedings*, segundo a estrutura consolidada (Listagem 16) e o endereço para acessar o nodo equivalente, *authors*, localizado no *doc4.json*. Em ambos os modos, a lista assume o mesmo formato. Apresentado o artefato *mapeamentos*, será então abordado na próxima subseção o artefato *mapa conceitual*.

3.4.8 Mapa Conceitual

O mapa conceitual consiste em um arquivo de texto (*mapa-conceitual.txt*) gerado após a execução da tarefa *Adaptar à Notação de Agregados*, que exibe as descrições de cada entidade presente no mapa e seus relacionamentos, obedecendo à notação IDEF1X. O formato exibido dos resultados, de acordo com a Listagem 18, percorre a estrutura consolidada em profundidade, ou seja, serão visitados todos os "filhos" de determinado objeto antes de serem exibidos os detalhes do objeto vizinho. A entidade "raiz" é exibida com a palavra-chave "ROOT" antes, indicando assim que aquele é o elemento raiz da estrutura consolidada. O campo *entidade* representa o nome do objeto JSON presente na estrutura. O campo *atributos* lista todos os campos (inteiros, strings, booleanos e nulos) presentes no objeto JSON apresentado. O campo REFI, obedecendo à notação IDEF1X, lista todos os objetos aninhados ao objeto apresentado (relacionamentos $0:1$) e o campo EMBED lista todos os arrays e arrays de objetos anexados ao objeto (relacionamentos $0:N$). O algoritmo de exibição mostra os "detalhes" apenas dos

objetos filhos que sejam do tipo objeto (objeto ou array de objetos), sendo os arrays simples e atributos apenas listados nos campos *EMBED* e *atributos*, respectivamente.

Listagem 18 – Mapa Conceitual

```
1  ----------------------------------------------------
2  ROOT
3  Entidade:dblp
4  Atributos: article
5  REFI: , proceedings
6  ----------------------------------------------------
7  ----------------------------------------------------
8  Entidade:proceedings
9  Atributos: @key, @mdate, title, pages, year, volume, journal,
      number, ee, url, booktitle, crossref, source title, art no,
      Page start, Page end, Page count, Cited by, DOI, Link, EID
10 REFI: , series
11 EMBED: , author, editor, isbn
12 ----------------------------------------------------
13 ----------------------------------------------------
14 Entidade:series
15 Atributos: @href,$
16 ----------------------------------------------------
```

Fonte: Autor

No exemplo exibido na Listagem 18, temos como objeto raiz o objeto de nome *dblp*, que possui como atributos um campo de nome *article* e possui um objeto filho (relacionamento $0:1$) chamado *proceedings*. Logo após, são exibidas as informações sobre o objeto aninhado *proceedings* e, por fim, os detalhes do seu objeto aninhado *series*. Apresentado o artefato *mapa conceitual*, será apresentado, na próxima subseção, o artefato adicional *Estrutura do Especialista*.

3.4.9 Estrutura do Especialista

Trata-se de um artefato adicional, feito para auxiliar especificamente no modo de execução por blocos, na atividade *remontar estrutura*. A estrutura foi montada para driblar uma limitação detectada na atividade *remontar estrutura* no modo de execução por blocos. Elementos comuns ao bloco, ou seja, de todos os documentos que contém o bloco b, todos contém o elemento e, não são adicionados à *lista de referências 1*, não sendo reconhecidos pelo algoritmo, pois a lista, no caso, armazena os elementos não comuns ao bloco, ou seja, do grupo de documentos que contém o bloco b, não são todos que contém o elemento e.

Essa estrutura obedece o mesmo princípio da estrutura unificada na execução em blocos, onde cada linha representa um bloco dos documentos JSON de entrada. Neste caso, representa as ocorrências dos blocos nos documentos de entrada. Por exemplo, segundo a Listagem 19, o bloco 2 (segunda linha) está presente nos documentos *doc1.json*, *doc2.json* e *doc3.json*, já o bloco 4 (quarta linha) está presente apenas no *doc2.json*.

Listagem 19 – Definições do Especialista

```
1 doc1.json; doc2.json; doc3.json
2 doc1.json; doc2.json; doc3.json
3 docc1.json; doc2.json; doc3.json; doc4.json
4 doc2.json
```

Fonte: Autor

3.5 ALGORITMOS IMPLEMENTADOS

Nesta seção será apresentado o detalhamento de cada algoritmo do processo descrito no referencial teórico, sendo apresentada uma versão teórica do algoritmo, baseada no algoritmo proposto pelo processo, logo após detalhado seu funcionamento, por fim, é apresentada uma versão implementada na linguagem Java (baseada na versão teórica) aplicado ao desenvolvimento da ferramenta proposta pelo trabalho. Nas próximas subseções serão apresentados os algoritmos referentes às atividades *Consolidar Estrutura*, *Remontar Estrutura*, *Montar Mapeamentos* e *Adaptar à Notação de Agregados*.

3.5.1 Consolidar Estrutura

O algoritmo apresentado (Algoritmo 5) consiste em percorrer o artefato *matriz única de resultados por bloco* e detectar as semelhanças apontadas na matriz, ou seja, elementos acima da diagonal principal que sejam de valor igual a 1, o que indica a equivalência entre dois campos, removendo então o campo considerado equivalente da lista de campos consolidados e adicionando a equivalência detectada ao artefato *lista de referências 2*.

Algoritmo 5: Algoritmo Consolidar Estrutura

Entrada: Matriz única por bloco (m)
Saída: Lista de campos consolidados (lcc), Lista de referências 2 (lr_2)

```
1  início
2      lcc ← todas as palavras associadas às colunas de m;
3      k ← tamanho da matriz quadrada m;
4      para j ← 0 até k − 1 faça
5          se j = 0 então
6              pular;
7          fim
8          para i ← (j − 1) até 0 faça
9              se m_ij = 1 então
10                 Adiciona as palavras lcc_i e lcc_j à lr_2;
11                 Remove a palavra lcc_j da lista lcc;
12             fim
13         fim
14     fim
15 fim
```

Fonte: Autor

Neste algoritmo (Algoritmo 5), por se tratar de uma matriz quadrada e diagonal superior que representa o relacionamento *todos-com-todos*, onde todos os elementos pertencentes à diagonal principal se referem ao relacionamento de igualdade entre os elementos com eles mesmos, a primeira posição (no caso, 0) não é testada, por se tratar da posição inicial (0, 0) da matriz, ou seja, não existe nenhum elemento acima deste. Por fim, é feita uma busca sequencial na matriz, somente nos endereços acima da diagonal principal, por elementos m_{ij} que sejam de valor igual a 1. Ao encontrá-los, são adicionadas as palavras associadas à linha i e à coluna j ao artefato *lista de referências 2* e então é removida a palavra associada à coluna j do artefato *campos consolidados*. A seguir é apresentada a versão implementada do Algoritmo 5 na linguagem Java (Listagem 20).

Listagem 20 – Método *consolidaEstruturaBloco*

```
1 private void consolidaEstruturaBloco(MatrizResultados
    matrizUnicaResultados,
2         List<String[]> listaReferencias2,
3         List<String> listaPalavrasConsolidadas) {
4     List<String> aRemover = new ArrayList<>();
5     // Caminhamento em diagonal (acima da diagonal principal)
6     // Movimentacao das colunas
7
```

```
8       List<List<Integer>> matrizBloco =
            matrizUnicaResultados.getMatriz();
9       // Consolida todas as pralavras da coluna
10      List<String> palavrasConsolidadas = new ArrayList<>(0);
11      for (String palavra :
            matrizUnicaResultados.getCamposColuna()) {
12          palavrasConsolidadas.add(palavra);
13      }
14
15      for (int j = 0; j < matrizBloco.size(); j++) {
16          if (j == 0) {
17              continue;
18          }
19          // Movimentacao das linhas
20          for (int i = j - 1; i >= 0; i--) {
21              if (matrizBloco.get(i).get(j) == 1) {
22                  String[] equivalencia = new String[2];
23                  equivalencia[0] = palavrasConsolidadas.get(i);
24                  equivalencia[1] = palavrasConsolidadas.get(j);
25                  // Adiciona as equivalencias a lista de
                        equivalencias
26                  listaReferencias2.add(equivalencia);
27                  /* Adiciona a palavra da coluna a uma lista de
                        palavras a
28                   * serem excluidas da lista de palavras
                         consolidadas
29                   * posteriormente */
30                  aRemover.add(palavrasConsolidadas.get(j));
31              }
32          }
33      }
34      //Remove da lista de elementos consolidados os elementos
            presentes
35      // na lista temporaria aRemover
36      for (String removerPalavra : aRemover) {
37          palavrasConsolidadas.remove(removerPalavra);
38      }
39      //Adiciona as palavras consolidadas a lista de palavras
            consolidadas
40      for (String palavraConsolidada : palavrasConsolidadas) {
41          listaPalavrasConsolidadas.add(palavraConsolidada);
42      }
43  }
```

Fonte: Autor

A versão implementada do Algoritmo 5 é representada pelo método *consolidaEstrutura-Bloco* (Listagem 20), que recebe como parâmetros um objeto chamado *matrizUnicaResultados*,

que representa o artefato *matriz única de resultados*, uma lista chamada *listaReferencias2*, que representa o artefato *lista de referencias 2* e a lista chamada *listaPalavrasConsolidadas*, que representa o artefato *palavras consolidadas*. O algoritmo começa inicializando uma lista auxiliar chamada *aRemover*, esta lista vai conter todos os elementos a serem removidos da lista de palavras consolidadas, logo após é então extraído a partir do objeto *matrizBloco* a *matriz única de resultados*, então em seguida iniciada uma lista temporária chamada *palavrasConsolidadas*, que vai conter todas as palavras consolidadas do processo executado no bloco. É feita a varredura da matriz, onde para todo elemento da matriz com valor igual a 1 é então criado um vetor de duas posições, que vai conter a palavra extraída a partir da posição i da lista de palavras consolidadas (representa a palavra associada à linha i da matriz) e a palavra extraída a partir da posição j da lista (representa a palavra associada à coluna j da matriz), sendo esta também adicionada à lista *aRemover*. Após percorrer a matriz, é então percorrida a lista *aRemover*, sendo removidas todas as palavras que existam nessa lista e na lista *palavrasConsolidadas*. Por fim, todos os elementos contidos na lista *palavrasConsolidadas* são então adicionados à lista *listaPalavrasConsolidadas*. Estas listas serão persistidas em disco fora do método. A sua execução, tanto no modo em blocos quanto em único bloco, é executada da mesma maneira, sendo a diferença no número de blocos onde é executado, pois o método *consolidaEstruturaBloco* é executado dentro de um laço de repetição em outro método, chamado *consolidarEstruturaBlocos* (Listagem 21), pertencente à classe *ConsolidarEstrutura*.

Listagem 21 – Método *consolidarEstruturaBlocos*

```
1  public void consolidarEstruturaBlocos() throws
       FileNotFoundException,
2          IOException {
3      // Carrega os artefatos de entrada
4      MatrizResultadosDAO carrega = new MatrizResultadosDAO();
5      List<MatrizResultados> matrizBlocos
6              = carrega.lerMatrizUnicaResultadosBlocos();
7      List<List<String>> palavrasConsolidadas = new
         ArrayList<>();
8      List<List<String[]>> listaReferencias2 = new ArrayList<>();
9
10     // Consolidar blocos da estrutura
11     for (MatrizResultados matrizBloco : matrizBlocos) {
12         List<String> palavrasConsolidadasBloco = new
             ArrayList<>();
13         List<String[]> listaReferencias2Bloco = new
             ArrayList<>();
```

```
14
15          // Executa o algoritmo de consolidar estrutura
16          consolidaEstruturaBloco(matrizBloco,
               listaReferencias2Bloco,
17                  palavrasConsolidadasBloco);
18
19          palavrasConsolidadas.add(palavrasConsolidadasBloco);
20          listaReferencias2.add(listaReferencias2Bloco);
21      }
22
23      //Une os resultados da lista de referencias 2 em uma unica
           lista
24      List<String[]> listaReferencias2Final = new ArrayList<>();
25      for (List<String[]> lista : listaReferencias2) {
26          for (String[] elemento : lista) {
27              listaReferencias2Final.add(elemento);
28          }
29      }
30
31      //Grava em arquivos os artefatos:
32      //lista de referencias 2 e campos consolidados
33      ListasDAO l = new ListasDAO();
34      l.gravarListaCamposConsolidados(palavrasConsolidadas);
35      l.gravarListaReferencias2(listaReferencias2Final);
36  }
```

Fonte: Autor

O método *consolidarEstruturaBlocos*, por sua vez, consiste em carregar as matrizes existentes na pasta da aplicação, a partir do método *lerMatrizUnicaResultadosBlocos*, pertencente à classe *MatrizResultadosDAO*. Os artefatos de saída (*palavras consolidadas* e *lista de referências 2*) são representadas por listas, inicializadas sem nenhum valor prévio. Logo, são percorridos os blocos carregados (cada bloco contendo uma *matriz única de resultados* associada) e é executado em cada bloco o método *consolidaEstruturaBloco* (Listagem 20), onde cada execução gera uma lista de campos consolidados e uma lista de referências individual, referentes ao bloco onde foi aplicado o método. Após percorrer todos os blocos, a lista de referências é unificada, pois está "repartida" em blocos e, por fim, os artefatos são gravados em arquivos de texto, através dos métodos *gravarListaCamposConsolidados* e *gravarListaReferencias2*, pertencentes à classe *ListasDAO*. Com o detalhamento da atividade *Consolidar Estrutura*, será apresentado, na próxima subseção, o algoritmo referente à atividade *Remontar Estrutura*.

3.5.2 Remontar Estrutura

O algoritmo *Remontar Estrutura* (Algoritmo 5) consiste em montar novamente o artefato *lista de campos consolidados* em uma estrutura, recolocando os delimitadores de cada elemento. Esta atividade não gera um arquivo JSON válido, mas uma estrutura que pode ser interpretada pelas próximas atividades e que representa o esquema da fonte de dados JSON. Sua execução consiste em dois modos, o modo *por blocos* ou o modo em *único bloco*, para cada modo é executada uma rotina diferente do algoritmo, como mostra o Algoritmo 6.

Algoritmo 6: Algoritmo Remontar Estrutura

Entrada: Lista de campos consolidados(lcc) separada por blocos, Estrutura com configurações do especialista (ce), Lista de referências 1 (lr_1) e 2 (lr_2)
Saída: Estrutura Consolidada

```
1  início
2      selecione modo de execução faça
3          caso único bloco faça
4              Escolhe o documento de origem para referência (com maior número de
                blocos;
5              para cada campo do documento de referências faça
6                  se campo está consolidado então mantém;
7                  senão verifica os correspondentes que estão consolidados e os
                    substitui;
8              fim
9              Os campos consolidados que não estão no documento de referência, são
                incluídos em um novo objeto delimitado por chaves;
10         fim
11         caso por blocos faça
12             ordenar lcc de acordo com critério do especialista, se necessário;
13             s1 ← tamanho da lista lcc;
14             para i ← (s1 − 1) até 0 faça
15                 s2 ← tamanho da sub-lista lcc_i, que contém os elementos do bloco i;
16                 para j ← (s2 − 1) até 0 faça
17                     obter informações sobre o termo consolidado lcc_ij, de acordo
                        com os dados de ce, lr1 e lr2;
18                     se lcc_ij for um objeto OU lcc_ij for um array de objetos então
19                         se fila_blocos estiver vazia então
20                             considera o objeto lcc_ij como se fosse um atributo (sem
                                "filhos"), removendo os delimitadores;
21                         fim
22                         senão
23                             adiciona o primeiro bloco armazenado em fila_blocos
                                como filho do objeto lcc_ij;
24                         fim
25                     fim
26                 fim
27                 adiciona lcc_i a fila_blocos;
28             fim
29         fim
30     fim
31     grava a estrutura obtida em um arquivo de texto;
32 fim
```

Fonte: Autor

No modo de execução *por blocos*, a estrutura consolidada está dividida em blocos, cada bloco com um conjunto de palavras consolidadas (lcc_i). Com os blocos devidamente ordena-

dos, o algoritmo percorre a lista de campos consolidados por blocos (a partir do final da lista até o início), e, a cada bloco visitado (lcc_i), busca por informações nos arquivos JSON de entrada referentes a cada termo consolidado pertencente ao bloco (lcc_{ij}), de acordo com as informações fornecidas pelos artefatos *lista de referências 1*, *lista de referências 2* e *configurações do especialista*. Cada bloco (lcc_i) é percorrido a partir do seu final até o início, sendo visitado a partir do último elemento do bloco (lcc_{ij}) até o primeiro. É então feita a verificação do tipo de elemento, caso o elemento seja um objeto, é verificado na fila de objetos a existência de um bloco, caso este não exista, então o objeto será tratado como um atributo, caso contrário, o objeto receberá o primeiro bloco na fila como seu bloco filho. A cada iteração, são adicionados os blocos à fila de blocos, aguardando pelos objetos aos quais serão adicionados como filhos. Após percorrer todos os blocos, grava a estrutura obtida em um arquivo de texto. Na Listagem 22 é mostrada uma implementação na linguagem Java do algoritmo para remontar estrutura no modo de execução em blocos.

Listagem 22 – Método *remontarPorBlocos*

```
1  private List<ElementoBloco> remontarPorBlocos(
2          List<List<ElementoBloco>> blocosCamposConsolidados,
3          List<String[]> listaRef2, List<List<String>> listaRef1,
4          List<List<String>> listaEspecialista) throws
              IOException {
5
6      int i;
7      for (i = blocosCamposConsolidados.size() - 1; i >= 0; i--)
          {
8          List<ElementoBloco> elementos =
              blocosCamposConsolidados.get(i);
9          for (int j = elementos.size() - 1; j >= 0; j--) {
10             ElementoBloco eb = elementos.get(j);
11             atualizaElementoBloco(eb, listaRef1,
                  listaEspecialista, i);
12             if (eb.getTipo() == ElementoBloco.OBJETO
13                     || eb.getTipo() ==
                        ElementoBloco.ARR_OBJETO) {
14
15                 if (!filaBlocos.isEmpty()) {
16                     eb.setBlocoFilho(filaBlocos.remove());
17                 } else {
18                     eb.setTipo(ElementoBloco.ATRIBUTO);
19                 }
20
21             }
22         }
23         filaBlocos.add(elementos);
```

```
24      }
25      // Seta o primeiro no como o raiz do objeto
26      return filaBlocos.poll();
27  }
```

Fonte: Autor

A implementação mostrada na Listagem 22 é exclusivamente voltada à execução em blocos, representada pelo método *remontarPorBlocos* (pertencente à classe *RemontarEstrutura*), que recebe como parâmetro a *lista de campos consolidados* separados por bloco (*blocoCamposConsolidados*), a *lista de referências 2* (*listaRef2*), a *lista de referências 1* (*listaRef1*) e a *estrutura do especialista* (*listaEspecialista*). O algoritmo começa fazendo a varredura da lista, como instruído no Algoritmo 6, obtém-se as informações sobre o objeto, através do método auxiliar *atualizaElementoBloco*, que recebe como parâmetros o elemento do bloco a ser atualizado (*eb*), a *lista de referências 1* (*listaRef1*), a *estrutura do especialista* (*listaEspecialista*) e, adicionalmente, o número do bloco atual, com o intuito de agilizar as buscas. Logo após é feita a verificação de tipo de elemento (objeto, array, array de objetos ou atributo). Caso seja um objeto, é aplicada a regra descrita no Algoritmo 6, caso contrário, o elemento é considerado um atributo pertencente ao bloco. Logo após, o algoritmo encerra sua execução, retornando o primeiro elemento da fila de blocos (uma estrutura auxiliar utilizada durante o processo de remontar estrutura), sendo este representante do nodo raiz do artefato *estrutura consolidada*. O carregamento dos artefatos e a gravação dos artefatos de saída são feitos pelo método também nomeado *remontarPorBlocos* (também pertencente à classe *RemontarEstrutura*), exibido na Listagem 23.

Listagem 23 – Método *remontarPorBlocos (2)*

```
1  public void remontarPorBlocos() {
2      try {
3          //Carrega os artefatos de entrada
4          ListasDAO l = new ListasDAO();
5          //Lista de referencias 1
6          List<List<String>> listaRef1 =
               l.lerListaReferencias1();
7          //Lista de referencias 2
8          List<String[]> listaRef2 = l.lerListaReferencias2();
9          //Configuracoes especialista
10         List<List<String>> listaEspecialista =
               l.lerListaEspecialista();
11         //Lista de campos consolidados
```

```
12          List<List<ElementoBloco>> blocosCamposConsolidados =
               l.lerListaCamposConsolidados();
13
14          // Remonta a estrutura
15          List<ElementoBloco> blocoPrincipal =
               remontarPorBlocos(blocosCamposConsolidados,
               listaRef2, listaRef1, listaEspecialista);
16
17          // Grava a estrutura em arquivo, como estrutura
               consolidada
18          EstruturaConsolidadaDAO est = new
               EstruturaConsolidadaDAO();
19          est.gravarEstruturaConsolidada(blocoPrincipal);
20      } catch (FileNotFoundException ex) {
21          Logger.getLogger(RemontarEstrutura.class.getName())
               .log(Level.SEVERE, null, ex);
22      } catch (IOException ex) {
23          Logger.getLogger(RemontarEstrutura.class.getName())
               .log(Level.SEVERE, null, ex);
24      }
25  }
```

Fonte: Autor

O método *remontarPorBlocos*, mostrado na Listagem 23 consiste em carregar os artefatos *lista de referências 1* (*listaRef1*), *lista de referências 2* (*listaRef2*), *estrutura do especialista* (*listaEspecialista*), *lista de campos consolidados* (*blocosCamposConsolidados*), através dos métodos *lerListaReferencias1*, *lerListaReferencias2*, *lerListaEspecialista* e *lerListaCamposConsolidados*, todos pertencentes à classe *ListaDAO*. Logo após, remonta a estrutura através do método *remontarPorBlocos*, que recebe todos os artefatos carregados por parâmetro. Após obter o artefato *estrutura consolidada*, o mesmo é gravado em disco, através do método *gravarEstruturaConsolidada* da classe *EstruturaConsolidadaDAO*.

No modo de execução como *único bloco*, é escolhido um documento do conjunto, que contenha o maior número de blocos, para ser usado como referência para a estrutura a ser remontada. Após a escolha do documento com mais blocos, é iniciada uma lista contendo todos os nomes de campos existentes no documento. Outra lista também é inicializada, inicialmente vazia, que contém os nomes (segundo a lista de campos consolidados) de todos os elementos (objetos, arrays, etc) que já foram adicionados anteriormente à estrutura, para evitar a adição de elementos repetidos à estrutura final. A lista de campos então é percorrida, e cada palavra é testada, sendo obtido o tipo de dados que esta palavra está associada (a um objeto, atributo ou array de objetos). Caso seja um atributo, primeiro verifica-se se o seu nome está na lista de

campos consolidados, caso negativo, é então obtido a partir da *lista de referências 2* o termo consolidado correspondente, verifica-se se o mesmo termo já não foi utilizado anteriormente, caso não, então ele é adicionado o atributo à estrutura do documento, caso contrário, o atributo é ignorado. Caso seja um objeto, é também feita a substituição de seu nome pelo nome consolidado correspondente (se necessário), é verificado se o termo foi utilizado anteriormente, se sim, caso o objeto contenha ao menos um atributo consolidado, então seu nome é removido e é então adicionado à estrutura com um nome genérico, *e_(número randômico)*. Caso o objeto não possua nenhum atributo e o termo consolidado associado ao seu nome não tenha sido utilizado anteriormente, então o objeto será tratado como um atributo (sem filhos), sendo adicionado à estrutura somente se o termo consolidado associado a seu nome não tiver sido utilizado anteriormente. No caso de arrays de objetos, todos os atributos de cada objeto aninhado é reunido em uma única estrutura, sendo eliminadas as repetições. Após percorrer todo documento, é então armazenada a estrutura obtida em um arquivo de texto. Na Listagem 24 é apresentada a implementação utilizada no trabalho, no caso, é representada pelo método *remontarUnicoBloco*.

Listagem 24 – Método *remontarUnicoBloco*

```
1  public void remontarUnicoBloco() throws FileNotFoundException,
       IOException {
2      ListasDAO l = new ListasDAO();
3      List<String[]> listaReferencias2 =
           l.lerListaReferencias2();
4      List<List<ElementoBloco>> listaConsolidada
5              = l.lerListaCamposConsolidados();
6      List<String> visitados = new ArrayList<>();
7      EstruturaConsolidadaDAO e = new EstruturaConsolidadaDAO();
8
9      String arquivo = arquivoComMaisBlocos();
10     InfoJSON info = new InfoJSON(arquivo);
11     ElementoBloco raiz = new ElementoBloco("RAIZ",
           ElementoBloco.OBJETO);
12
13     if (info.temUnicoObjetoRaiz()) {
14         // Se sim, seta o nome do raiz com o nome do primeiro
               objeto
15         File f = new File(arquivo);
16         FileInputStream fi = new FileInputStream(f);
17         JsonParser parser = Json.createParser(fi);
18         Event evt = null;
19         while (parser.hasNext()) {
20             evt = parser.next();
21             if (evt == Event.KEY_NAME) {
```

```
22                  raiz.setNome(parser.getString());
23                  parser.close();
24                  fi.close();
25                  break;
26              }
27          }
28          montaArvore(raiz, listaReferencias2, listaConsolidada,
               info, visitados);
29      } else {
30          montaArvore(raiz, listaReferencias2, listaConsolidada,
               info, visitados);
31      }
32      e.gravarEstruturaConsolidada(raiz);
33  }
```

Fonte: Autor

O método consiste em carregar os artefatos, logo após obtém o nome do documento pertencente ao conjunto de arquivos de entrada JSON que contenha o maior número de blocos, através do método *arquivoComMaisBlocos*. É verificado se o documento escolhido possui um objeto raiz único, caso não possua, é criado um objeto raiz genérico para a estrutura consolidada que será gerada, caso contrário é utilizado então o primeiro objeto do arquivo como raiz. É então chamado o método *montaArvore*, que implementa todas as regras para a geração (em formato de árvore) da estrutura consolidada, recebendo como parâmetro o objeto raiz, os artefatos carregados e a lista inicial de palavras consolidadas já utilizadas (parâmetro de nome *visitados*). Após a formação da estrutura consolidada na memória, a árvore (no caso, representada apenas pelo objeto *raiz*) é gravada em um arquivo de texto através do método *gravarEstruturaConsolidada*, pretencente à classe *EstruturaConsolidadaDAO*.

Agora é possível abordar sobre a atividade *montar mapeamentos* na próxima subseção.

3.5.3 Montar Mapeamentos

O algoritmo *Montar Mapeamentos* (Algoritmo 7) consiste em analisar o artefato *lista de referências 2* na busca de novas equivalências e, ao final, formar uma tabela, contendo o endereço JSONPath para o elemento consolidado, o endereço JSONPath para o(s) elemento(s) considerado(s) equivalente(s) e o nome do(s) documento(s) JSON que contém o(s) elemento(s) equivalente(s). O principal detalhe não presente no algoritmo inicial proposto (Algoritmo 3) é o fato de que um campo pode estar presente em mais de um documento, sendo necessário incluir estes documentos na lista também. Para ambos modos de execução (*por blocos* e *único bloco*),

o seu procedimento é o mesmo.

Algoritmo 7: Algoritmo Montar Mapeamentos

Entrada: Estrutura de Campos Consolidados (lcc), Listas de Referências 1 (lr_1) e 2 (lr_2)
Saída: Mapeamentos (map)

```
1  início
2      para cada e_k ∈ lcc faça
3          termo_equiv ← termos equivalentes ao elemento e_k (e_k = a) de acordo
             com a lr_2;
4          termo_equiv ← termos equivalentes ao elemento
             e_k (e_k = a, a = b ⇒ e_k = b) de acordo com a lr_2;
5          json_path_e ← JSONPath para o elemento e_k, segundo a estrutura
             consolidada;
6          para cada t_m ∈ termo_equiv faça
7              doc_orig ← nome(s) do(s) documento(s) de origem que contém o
                 elemento t_m, de acordo com a lr_1;
8              para cada doc_n ∈ doc_orig faça
9                  json_path_t ← JSONPath para o elemento t_m em doc_n;
10                 map_n ← valores de json_path_e, json_path_t e doc_n obtidos;
11             fim
12         fim
13     fim
14     grava conteúdo de map em um arquivo de texto;
15 fim
```

Fonte: Autor

O Algoritmo 7 inicia percorrendo o artefato *lista de campos consolidados* (lcc) e adicionando os termos equivalentes ao o elemento consolidado e_k à lista $termos\_equiv$, através de análises do tipo $(e_k = a)$ e $(e_k = a,\ a = b \Rightarrow e_k = b)$. Em seguida, o endereço JSONPath para e_k é armazenado na variável temporária $json\_path\_e$, segundo o artefato *estrutura consolidada*, previamente remontada na atividade *remontar estrutura*. Logo após, é percorrida a lista $termos\_equiv$, sendo obtidos os documentos de origem que contenham o termo equivalente t_m, de acordo com informações fornecidas pelo artefato *lista de referências 1*, que contém os nomes dos elementos e seus respectivos documentos de origem. Então, para cada documento de origem (doc_n) associado ao termo t_m, é obtido o JSONPath para o termo t_m no documento doc_n e armazenado na variável temporária $json\_path\_t$, por fim, é adicionado à lista de mapeamentos (map) uma nova linha contendo os valores de $json\_path\_e$, $json\_path\_t$ e doc_n, representando o endereço JSONPath para o elemento consolidado, endereço JSONPath para o elemento equivalente e o nome do documento de origem do termo, respectivamente. É exibido na Listagem 25 uma implementação na linguagem Java para o Listagem 7.

Listagem 25 – Método *getMapeamentos*

```
1  private List<String[]> getMapeamentos(
2          List<String> listaCamposConsolidados,
3          List<String> listaJsonPathCamposConsolidados,
4          List<String[]> listaRef2, List<List<String>>
              listaRef1) {
5      int i = 0;
6
7      List<String> fixados = new ArrayList<>(0);
8      List<String[]> mapeamentos = new ArrayList<>(0);
9      for (String elem : listaCamposConsolidados) {
10
11         i++;
12         List<String> termosEquivalentes = new ArrayList<>();
13         obterTermosEquivalentes(elem, listaRef2,
              termosEquivalentes, fixados);
14
15         for (String termo : termosEquivalentes) {
16             List<String> docOrig = getDocOrigem(termo,
                  listaRef1);
17             for (String documentoOrig : docOrig) {
18                 i = listaCamposConsolidados.indexOf(elem);
19                 String[] dados =
                      {listaJsonPathCamposConsolidados.get(i),
20                       termo, documentoOrig};
21                 mapeamentos.add(dados);
22             }
23         }
24     }
25     return mapeamentos;
26 }
```

Fonte: Autor

O método *getMapeamentos*, apresentado na Listagem 25, pertence à classe *MontarMapeamentos*, recebe como parâmetro os artefatos *lista de campos consolidados* (*listaCamposConsolidados*), as *listas de referências 1* (*listaRef1*) e *2* (*listaRef2*) e a lista auxiliar *listaJsonPathCamposConsolidados*, que é uma lista que contém o endereço JSONPath de cada elemento presente na lista *listaCamposConsolidados*. O algoritmo obtém os termos equivalentes através do método *obterTermosEquivalentes*, que recebe como parâmetros o nome do elemento a ser analisado (*elem*), a *lista de referências 2* (*listaRef2*), a lista de termos equivalentes (*termosEquivalentes*) e a lista auxiliar *fixados* (Listagem 26).

Listagem 26 – Método *obterTermosEquivalentes*

```
1  private void obterTermosEquivalentes(String termo,
       List<String[]> listaRef2,
2          List<String> equivalencias, List<String> fixados) {
3      if (fixados.contains(termo)) {
4          return;
5      }
6      for (String[] termoAtual : listaRef2) {
7          if (termoAtual[0].equals(termo)) {
8              if (!fixados.contains(termoAtual[1])
9                      && !equivalencias.contains(termoAtual[1]))
                           {
10                 equivalencias.add(termoAtual[1]);
11                 fixados.add(termo);
12                 obterTermosEquivalentes(termoAtual[1],
                     listaRef2,
13                          equivalencias, fixados);
14             }
15         } else if (termoAtual[1].equals(termo)) {
16             if (!fixados.contains(termoAtual[0])
17                     && !equivalencias.contains(termoAtual[0]))
                           {
18                 equivalencias.add(termoAtual[0]);
19                 fixados.add(termo);
20                 obterTermosEquivalentes(termoAtual[0],
                     listaRef2,
21                          equivalencias, fixados);
22             }
23         }
24     }
25 }
```

Fonte: Autor

O método auxiliar, mostrado na Listagem 26, consiste em percorrer a *lista de referências 2* de modo recursivo, a fim de detectar relações do tipo $(a = b)$ e $(a = b,\ b = c \Rightarrow a = c)$, testando ambos os "lados" da lista. Para evitar *loops* indefinidos e, consequentemente, o "estouro de pilha", a lista auxiliar *fixados* serve como um histórico de palavras já visitadas pelo algoritmo, impedindo-o de testar novamente algum termo que já tenha sido testado anteriormente.

Por fim, o método *getMapeamentos* é executado por outro método, chamado *montarMapeamentos*, na classe *MontarMapeamentos* (Listagem 27), sendo este responsável por carregar os artefatos e gravar a lista de mapeamentos na memória secundária. Este também é o método principal, que é chamado pela classe *Controle*.

Listagem 27 – Método *montarMapeamentos*

```
1  public void montarMapeamentos() throws FileNotFoundException,
       Exception {
2      ListasDAO l = new ListasDAO();
3      EstruturaConsolidadaDAO e = new EstruturaConsolidadaDAO();
4      //Carrega os artefatos
5      //Estrutura consolidada
6      NodoEstruturaConsolidada estrCons =
           e.lerEstruturaConsolidada();
7      List<String> listaCamposConsolidados =
           estrCons.getListaNomes();
8      List<String> listaJsonPathCamposConsolidados
9              = estrCons.getListaJsonPath();
10     //Lista de referencias 1
11     List<List<String>> listaRef1 = l.lerListaReferencias1();
12
13     //Lista de referencias 2
14     List<String[]> listaRef2 = l.lerListaReferencias2();
15
16     //Monta os mapeamentos
17     List<String[]> listMapeamentos =
           getMapeamentos(listaCamposConsolidados,
18             listaJsonPathCamposConsolidados, listaRef2,
                   listaRef1);
19
20     //Grava a tabela de mapeamentos
21     l.gravarMapeamentos(listMapeamentos);
22 }
```

Fonte: Autor

O método carrega o artefato *Estrutura Consolidada* (*estrCons*) através do método *lerEstruturaConsolidada* da classe *EstruturaConsolidadaDAO* e os artefatos *lista de referências 1* (*listaRef1*) e *lista de referências 2* (*listaRef2*) através dos métodos *lerListaReferencias1* e *lerListaReferencias2* respectivamente. As estruturas auxiliares *listaCamposConsolidados* e *listaJsonPathCamposConsolidados* são obtidas a partir do objeto da estrutura consolidada (*estrCons*). Logo após, é executado o método *getMapeamentos*, recebendo como parâmetros os itens carregados anteriormente e, por fim, a lista obtida com os mapeamentos é gravada em um arquivo através do método *gravarMapeamentos* da classe *ListasDAO*. Detalhada a implementação *Montar Mapeamentos*, agora é possível apresentar e detalhar o funcionamento da última atividade, *Adaptar à Notação de Agregados*.

3.5.4 Adaptar à Notação de Agregados

O algoritmo a seguir apresentado (Algoritmo 8) consiste em interpretar o artefato *Estrutura Consolidada* e gerar uma representação da estrutura, ou seja, um esquema conceitual, no formato IDEF1X a partir da interpretação desta. Este algoritmo, para interpretar a estrutura consolidada, lê o arquivo caractere por caractere, e vai formando a estrutura na memória de acordo com os símbolos que encontra ao longo do documento.

Algoritmo 8: Adaptar à Notação de Agregados

Entrada: Arquivo com a Estrutura Unificada, Regras definidas
Saída: Esquema Conceitual

```
1  início
2      estr_unif ← preparar para leitura arquivo com a Estrutura Unificada;
3      cont_obj ← 0;
4      para cada s_k ∈ estr_unif faça
5          selecione s_k faça
6              caso { faça
7                  aplicar C_1;
8                  cont_obj + +;
9                  se cont_obj = 1 então
10                     primeiro objeto: considerar como raiz;
11                 fim
12             fim
13             caso value faça aplicar C_2;
14             caso [ faça aplicar C_3;
15             caso [{ faça aplicar C_4;
16         fim
17     fim
18     monta esquema conceitual a partir da estrutura obtida;
19 fim
```

Fonte: Autor

O algoritmo começa ao preparar o artefato *Estrutura Unificada* para leitura, armazenando o ponteiro para a primeira posição do documento na variável $estr\_unif$, logo após, inicializa a variável $cont\_obj$ com o valor 0, esta é uma variável auxiliar para ajudar a determinar se o objeto é um objeto raiz ou não. Então começa a percorrer a estrutura unificada, aplicando as regras definidas (C_1, C_2, C_3 e C_4) de acordo com o símbolo que encontra ao ler a estrutura. Após ler todo o documento, monta um esquema conceitual a partir da estrutura obtida através da interpretação do artefato. Na Listagem 28 é apresentado um fragmento da versão implementada na linguagem Java do Algoritmo 8.

Listagem 28 – Fragmento do método *adaptarNotacao*

```
1  private void adaptarNotacao(char simbolo) {
2      NodoEstruturaConsolidada novoNodo = null;
3      switch (simbolo) {
4          case FIM_PALAVRA:
5              if (bufferPalavras.empty()) {
6                  bufferPalavras.push(palavraAtual.toString()
                       .trim());
7              } else {
8                  nodoAtual.addAtributo(bufferPalavras.pop());
9                  bufferPalavras.push(palavraAtual.toString()
                       .trim());
10             }
11             palavraAtual = new StringBuilder();
12             break;
13         case INI_CLASSE:
14             novoNodo = new
                   NodoEstruturaConsolidada(bufferPalavras.pop());
15             if (nodoAtual != null) {
16                 nodoAtual.addFilho(novoNodo);
17                 nodoAtual.addRelacionamento(
                       NodoEstruturaConsolidada.ZERO_UM);
18                 pilha[NODOS_ABERTOS].push(nodoAtual);
19             } else {
20                 novoNodo.setRaiz(true);
21             }
22             nodoAtual = novoNodo;
23             break;
24         case FECHA_CLASSE:
25             if (!bufferPalavras.empty()) {
26                 nodoAtual.addAtributo(bufferPalavras.pop());
27             }
28             if (!pilha[NODOS_ABERTOS].empty()) {
29                 pilha[NODOS_FECHADOS].push(nodoAtual);
30                 nodoAtual = (NodoEstruturaConsolidada)
                       pilha[NODOS_ABERTOS].pop();
31             } else {
32                 pilha[NODOS_FECHADOS].push(nodoAtual);
33             }
34             break;
35 ...
```

Fonte: Autor

O método *adaptarNotacao*, pertencente à classe *ParserEstruturaConsolidada*, apresentado na Listagem 28 é a implementação do Algoritmo 8 e das regras definidas juntamente. Devido sua extensão, foi mostrado apenas a parte do código referente ao reconhecimento de

palavras e aplicação da regra C_1. Tanto o método *adaptarNotacao* quanto os demais referentes à interpretação do arquivo da estrutura unificada foram implementados na classe *ParserEstruturaConsolidada*, devido a sua complexidade. Este método consiste em reconhecer o símbolo e então formar uma estrutura na memória de acordo com as regras definidas descritas no trabalho. O método *adaptarNotação* interpreta os símbolos lidos a partir do arquivo, interpretados pelo método *lerCaractere* (Listagem 29).

Listagem 29 – Fragmento do método *lerCaractere*

```
1  public void lerCaractere(char c) {
2      switch (c) {
3          case ESPACO:
4              if (leituraPalavra) {
5                  bufferSimbolos = PALAVRA;
6                  palavraAtual.append(c);
7              }
8              break;
9          case '\n':
10             break;
11         case FIM_PALAVRA:
12             if (leituraPalavra) {
13                 bufferSimbolos = FIM_PALAVRA;
14                 leituraPalavra = false;
15                 adaptarNotacao(FIM_PALAVRA);
16             }
17             break;
18         case INI_CLASSE:
19             if (bufferSimbolos == INI_ARRAY) {
20                 adaptarNotacao(INI_ARRAY_OBJ);
21             } else {
22                 adaptarNotacao(INI_CLASSE);
23             }
24             break;
25         case FECHA_CLASSE:
26             bufferSimbolos = FECHA_CLASSE;
27             adaptarNotacao(FECHA_CLASSE);
28             break;
29 ...
30 }
```

Fonte: Autor

O método *lerCaractere* recebe como parâmetro o caractere obtido a partir da leitura do artefato *Estrutura Consolidada* e faz uma análise léxica para o método *adaptarNotacao*, transformando a entrada em símbolos e valores, simplificando a análise de símbolos como "[{", que são tratados na aplicação como um único símbolo, mas sua leitura corresponde a dois

caracteres, “[” e “{”. Os símbolos utilizados internamente no método e na classe (*ESPAÇO*, *FIM_PALAVRA*, *INI_CLASSE*, etc) foram definidos como constantes da classe, com o intuito de facilitar sua manipulação e compreensão (Listagem 30).

Listagem 30 – Constantes da classe *ParserEstruturaConsolidada*

```
1 private final static char ESPACO = ' ';
2 private final static char FIM_PALAVRA = ';';
3 private final static char INI_CLASSE = '{';
4 private final static char FECHA_CLASSE = '}';
5 private final static char INI_ARRAY = '[';
6 private final static char FECHA_ARRAY = ']';
7 private final static char INI_ARRAY_OBJ = '?';
8 private final static char FECHA_ARRAY_OBJ = '*';
9 private final static char PALAVRA = '~';
```

Fonte: Autor

O método *lerCaractere* é o método de entrada da classe *parserEstruturaConsolidada*, enquanto o método *getEstruturaConsolidada*, que retorna a árvore (o nodo raiz) que representa a estrutura consolidada na memória, é o método de saída da classe. Ambos são utilizados na classe *EstruturaConsolidadaDAO*, no método *lerEstruturaConsolidada* (Listagem 31)

Listagem 31 – Método *lerEstruturaConsolidada*

```
1  public NodoEstruturaConsolidada lerEstruturaConsolidada()
2          throws FileNotFoundException, IOException, Exception {
3      String nomeArquivo =
          "artefatos-saida/estrutura-consolidada.txt";
4      ParserEstruturaConsolidada notacoes = new
          ParserEstruturaConsolidada();
5      FileReader arq = new FileReader(nomeArquivo);
6      BufferedReader lerArq = new BufferedReader(arq);
7      int c;
8
9      while ((c = lerArq.read()) != -1) {
10         notacoes.lerCaractere((char)c);
11     }
12     lerArq.close();
13     arq.close();
14     return notacoes.getNodoEstruturaConsolidada();
15 }
```

Fonte: Autor

O método *lerEstruturaConsolidada* (Listagem 31) consiste em abrir o arquivo de texto para leitura, ler o documento caractere por caractere, até o seu final e repassar a entrada para o método *lerCaractere* da classe *ParserEstruturaConsolidada*, que fará a análise e por fim a

elaboração da estrutura consolidada na memória. Ao final do arquivo, o arquivo é fechado e então o método retorna a árvore da Estrutura Consolidada (através do seu nodo raiz). Por sua vez, o método *lerEstruturaConsolidada* é utilizado pelo método *adaptarNotacaoAgregados* da classe *AdaptarNotacaoAgregados* para ser feita a interpretação da estrutura consolidada para a notação IDEF1X e o esquema conceitual resultante ser armazenado em um arquivo de texto (Listagem 32).

Listagem 32 – Método *adaptarNotacaoAgregados*

```
1 public void adaptarNotacaoAgregados() throws Exception {
2     // Carrega os artefatos de entrada
3     EstruturaConsolidadaDAO ecd = new
          EstruturaConsolidadaDAO();
4     NodoEstruturaConsolidada n = ecd.lerEstruturaConsolidada();
5
6     // Grava esquema conceitual
7     ecd.gravarEsquemaConceitual(n);
8 }
```

Fonte: Autor

O método *adaptarNotacaoAgregados* (Listagem 32) consiste em carregar a estrutura consolidada, através do método *lerEstruturaConsolidada* e gravar a estrutura no formato de esquema conceitual, através do método *gravarEsquemaConceitual*, também da mesma classe (Listagem 33).

Listagem 33 – Método *gravarEsquemaConceitual*

```
1 public void gravarEsquemaConceitual(NodoEstruturaConsolidada
      estrutura)
2           throws IOException {
3     FileWriter arq = new
          FileWriter("artefatos-saida/esquema-conceitual.txt");
4     PrintWriter gravarArq = new PrintWriter(arq);
5     gravarArq.println(estrutura.toString());
6     gravarArq.close();
7     arq.close();
8 }
```

Fonte: Autor

O método *gravarEsquemaConceitual* (Listagem 33) consiste em abrir o arquivo de destino (*esquema-conceitual.txt*) em modo de gravação e gravar o conteúdo no arquivo, através do método *println* da biblioteca do java *PrintWriter*. Foi utilizando o método *toString* modificado da classe *NodoEstruturaConsolidada* para gerar uma string com o conteúdo da árvore em memória da estrutura consolidada, listando seu conteúdo de forma legível (Listagem 34).

Listagem 34 – Método *toString* da classe *NodoEstruturaConsolidada*

```
@Override
public String toString() {
    if (!this.atributos.isEmpty() &&
       this.atributos.get(0).equalsIgnoreCase("att")) {
        return "";
    }

    String tmp =
       "-------------------------------------------------
       --\n";
    if (raiz) {
        tmp += "ROOT\n";
    }
    tmp += "Entidade:" + nome + "\n";

    //Impressao de atributos
    tmp += "Atributos: ";
    for (int i = 0; i < this.atributos.size(); i++) {
        if (i > 0) {
            tmp += ",";
        }
        tmp += this.atributos.get(i);
    }
    tmp += "\n";

    // Imprimir os relacionamentos 0:1
    if (relacionamentos.contains(ZERO_UM)) {
        tmp += "REFI: ";
        int i = 0;
        for (String relacionamento : relacionamentos) {
            if (relacionamento.equalsIgnoreCase(ZERO_UM)) {
                tmp += ", " + filhos.get(i).getNome();
            }
            i++;
        }
        tmp += "\n";
    }

    // Imprimir os relacionamentos 0:N
    if (relacionamentos.contains(ZERO_MUITOS)) {
        tmp += "EMBED: ";
        int i = 0;
        for (String relacionamento : relacionamentos) {
            if (relacionamento.equalsIgnoreCase(ZERO_MUITOS)) {
                tmp += ", " + filhos.get(i).getNome();
            }
            i++;
```

```
45          }
46          tmp += "\n";
47      }
48      tmp +=
           "-------------------------------------------------------
           --\n";
49      for (NodoEstruturaConsolidada n : filhos) {
50          tmp += n.toString();
51      }
52      return tmp;
53  }
```

Fonte: Autor

O método apresentado na Listagem 34 consiste em uma substituição do método padrão do java por um método que, basicamente, imprime is dados da entidade e de seus nodos filhos, fazendo o caminhamento em profundidade, ou seja, serão exibidas primeiro informações sobre os filhos do nodo atual e dos filhos dos filhos antes de serem exibidas informações sobre os nodos vizinhos pertencentes à árvore.

4 ESTUDO DE CASO

Este capítulo apresenta a aplicação da ferramenta desenvolvida segundo o processo de extração proposto em um estudo de caso. O processo tem início a partir de parte do processo proposto pela autora do método, mais precisamente na atividade *Consolidar Estrutura*, que tem como entrada a matriz única por bloco, entretanto, durante o decorrer do processo, serão necessários os artefatos Estrutura Unificada e Lista de Referências 1, além dos documentos JSON de entrada. Ao final do processo é gerado um esquema conceitual (em um arquivo de texto) e uma tabela com os mapeamentos.

4.1 VISÃO GERAL

Este estudo de caso utilizará como base o estudo de caso realizado pela autora, o qual o fez manualmente, para testar o método. Para tal, serão reutilizados alguns artefatos já existentes e disponibilizados pela autora em uma pasta pública do OneDrive[5], referentes a atividades que não estão no escopo do desenvolvimento do trabalho. Será realizado, assim como no estudo de caso de origem, testes envolvendo duas formas de aplicação, a primeira corresponde à execução por blocos, onde é necessária a intervenção de um especialista do domínio para especificar quais blocos são equivalentes, a segunda trata o documento como um único bloco, comparando todos os campos, de todos os blocos, entre si.

Os arquivos de entrada utilizados são os mesmos arquivos utilizados no estudo de caso de Machado (2017). Os arquivos são documentos no formato JSON provindos de bibliotecas digitais como DBLP[6], Scopus[7], PubMed[8] e Bibsonomy[9]. Como os algoritmos desenvolvidos no presente trabalho se encaixam nas últimas etapas do processo proposto, são incluídos também os artefatos gerados até a atividade *Calcular equivalência por bloco*, sendo estes:

- lista de referências 1;
- matriz única de resultados.

Os artefatos foram pré-processados manualmente para que pudessem ser corretamente manipulados pela aplicação. Sendo assim, o processo pode ser executado de duas maneiras, a

[5] https://1drv.ms/f/s!AkZi7x6fPkh7xzxEXLfNvOF29VUS
[6] http://dblp.uni-trier.de/
[7] https://www.scopus.com/home.uri
[8] https://www.ncbi.nlm.nih.gov/pubmed
[9] https://www.bibsonomy.org/

primeira divide os testes por bloco, requerendo a intervenção de um especialista para identificar os blocos equivalentes. O primeiro modo de operação será aplicado nos testes com a adição de uma estrutura auxiliar que represente as definições de um especialista, indicando os blocos considerados equivalentes. Para o segundo modo nenhum acréscimo será feito, pois este opera de forma automática. Os resultados de ambos estudos de caso e o código-fonte da aplicação encontram-se disponíveis em uma pasta pública do *Google Drive*[10] e no *GitHub*[11] do autor do trabalho, respectivamente.

4.2 FERRAMENTAS

Esta seção descreve as ferramentas utilizadas no processo de extração de esquema.

O processo tem início com os arquivos em formato JSON e os artefatos e gera, ao final, um esquema conceitual e uma tabela de mapeamentos. Desse modo, o processo consiste na execução das quatro últimas atividades (*consolidar estrutura*, *remontar estrutura*, *montar mapeamentos* e *adaptar à notação de agregados*) presentes nas duas últimas etapas (*identificação de equivalências* e *representação da estrutura*) do processo.

Antes de dar início, os arquivos JSON e os artefatos precisam ser pré-processados para que possam ser processados. Para isto, é feita uma formatação manual nos arquivos JSON, que consiste em dividir um único arquivo JSON (que continha em sua estrutura quatro arquivos) em quatro arquivos individuais(*doc1.json*, *doc2.json*, *doc3.json* e *doc4.json*) e alterar a formatação dos artefatos, resultando em arquivos no formato CSV (*lista-referencias1.csv*, *estrutura-unif.csv* e *matriz-unica-resultados.csv*).

Ao final do processo, o diagrama entidade-relacionamento é obtido manualmente a partir da interpretação do arquivo de texto gerado após a execução da atividade “Adaptar para a Notação de Agregados”, que apresenta em formato texto a estrutura consolidada e as informações sobre os relacionamentos entre as entidades. Para o desenho dos diagramas ER é utilizada a ferramenta web *Draw.io*[12]. Nas próximas seções são demonstrados os casos de testes propostos para cada forma de aplicação.

[10] https://drive.google.com/open?id=1BAUNyMaZlAaCmvIRFU5nyRe1REnnWoPD
[11] https://github.com/ezequielrribeiro/algoritmos-tcc
[12] https://www.draw.io/

4.3 EXECUÇÃO POR BLOCOS

Nesta seção serão apresentados os passos efetuados para o processamento dos artefatos segundo o modo de execução por blocos, onde as comparações são feitas somente entre os elementos pertencentes a um conjunto predeterminado, sendo essa divisão por blocos definida por um especialista, que define esses conjuntos.

4.3.1 Divisão de blocos utilizada

Para o processo de execução em blocos, a autora definiu uma divisão prévia para os blocos segundo os documentos disponíveis. O primeiro bloco é composto, nos documentos 1, 2 e 3, pelo nível onde se encontra a palavra *dblp*, sendo apenas esta a palavra que constitui o primeiro bloco. O segundo bloco, nos documentos 1, 2 e 3, é constituído pelo nível onde se encontram as palavras *article*,*proceedings* e *inproceedings*, sendo estas as palavras que constituem o segundo bloco, o terceiro bloco é composto pelos documentos 1, 2, 3 pelo nível que abrange palavras como *@key*, *editor*, *author* e as demais que compõem este nível, e pelo primeiro nível do documento 4, composto por *Authors*, *Title*, etc, sendo a junção de todas essas palavras o conjunto que constitui o terceiro bloco. O quarto bloco é composto apenas pelo documento 2, pelo nível que abrange as palavras *@href* e *$*, sendo estas as palavras que compõem o quarto bloco. Segundo a autora, os testes de semelhança serão feitos especificamente entre palavras que pertençam ao mesmo conjunto, não sendo feitos testes com as demais, sendo necessário, em alguns casos, artefatos individuais para cada bloco, que, no caso do trabalho, acontece em relação ao artefato de entrada *matriz única de resultados*.

4.3.2 Pré-processamento

4.3.2.1 Arquivos JSON

A primeira etapa consistiu no pré-processamento dos artefatos de entrada (*arquivos JSON*, *lista de referências 1* e *matrizes únicas de resultados por bloco*) presentes na pasta pública da autora. O processo de pré-processamento foi manual e efetuado com a ajuda da ferramenta de edição *Sublime Text*[13].

Os arquivos JSON foram obtidos a partir de um único arquivo, *Caso1-dataset-bkp.json*,

[13] https://www.sublimetext.com/

disponível na pasta pública. O arquivo é composto por 4 estruturas, como mostrado pela Listagem 35, sendo que cada estrutura representa um arquivo JSON distinto. As estruturas foram separadas e então foram obtidos 4 documentos JSON (*doc1.json*, *doc2.json*, *doc3.json* e *doc4.json*) a partir do documento original.

Listagem 35 – Fragmento do arquivo *Caso1-dataset-bkp.json*

```
{
"dblp": {
        "article": {
                "@key": "journals/ijbpim/AlmeidaBF15",
                "@mdate": "2016-01-05",
                "author": [
                        "Rafael Almeida",
                        "Jorge Bernardino",
...
}
{
"dblp": {
        "proceedings": {
                "@key": "conf/naa/2016",
                "@mdate": "2017-04-13",
                "editor": [
                        "Ivan Dimov",
                        Istvan Farago",
...
}
{
"dblp": {
        "inproceedings": {
                "@key": "conf/ACMse/ParkerPV13",
                "@mdate": "2015-05-07",
                "author": [
                        "Zachary Parker",
                        "Scott Poe",
...
}
{
        "Authors": "Gonzalez", Aparicio M.T., Ogunyadeka A.,
           Younas M., Tuya J., Casado R.",
        "Title": "Transaction processing in consistency-aware
           user's applications deployed on NoSQL databases"
...
}
```

Fonte: (MACHADO, 2017)

4.3.2.2 Lista de Referências 1

A *lista de referências 1* utilizada foi fornecida no arquivo PDF *Estudo de caso 1 - Documentação.pdf*, que é um documento que contém a descrição dos artefatos gerados, arquivos de entrada e resultados obtidos pelo processo desenvolvido pela autora. A Figura 9 exibe a lista presente no arquivo PDF.

Figura 9 – Primeira Lista de Referências - original da pasta pública

Article – doc1	Pages – doc1, 3	editor – doc 2	Issue – doc 4
Proceedings – doc 2	Year – doc1, 2, 3, 4	Booktitle – doc 2,3	Art no – doc 4
Inproceedings – doc3	Volume – doc1, 2,4	Series – doc 2	Page start - doc4
@key – doc1, 2, 3	Journal – doc 1	Isbn – doc 2	Page end – doc 4
@mdate – doc 1, 2, 3	Number – doc1	Crossref – doc 3	Page count – doc 4
author – doc 1, 3	Ee – doc1, 2, 3	Authors – doc4	Cited by – doc 4
Title – doc1, 2, 3, 4	url – doc1 ,2, 3	Source title – doc4	DOI – doc 4
Link – doc 4	EID - doc 4		

Fonte: (MACHADO, 2017)

O pré-processamento da *lista de referências 1* consistiu em copiar o conteúdo a partir do arquivo PDF, colar na janela principal do *Sublime Text*, substituir os separadores "-" por ponto e vírgula, completar os nomes dos documentos (por exemplo, substituir "*doc1, 2, 3*" por "*doc1.json*; *doc2.json*; *doc3.json*;"). O resultado do pré-processamento é exibido na Listagem 36.

Listagem 36 – Fragmento do arquivo *lista-referencias1.csv*

```
Article;doc1.json
Pages;doc1.json; doc3.json
editor;doc2.json
Issue;doc4.json
Proceedings;doc2.json
Year;doc1.json; doc2.json; doc3.json; doc4.json
Booktitle;doc2.json; doc3.json
Art no;doc4.json
Inproceedings;doc3.json
Volume;doc1.json; doc2.json; doc4.json
Series;doc2.json
Page start;doc4.json
@key; doc1.json; doc2.json; doc3.json
...
```

Fonte: Autor

4.3.2.3 Matriz Única de Resultados

Por se tratar de um processo realizado em blocos, para cada bloco definido será necessária uma matriz de resultados para o bloco em questão. A matriz única de resultados foi extraída a partir do arquivo XLSX *Estudo de Caso 1 - Algoritmo 6.xlsx* disponível na pasta pública. A tabela com a matriz única de resultados se localiza na planilha (dentro do documento XLSX) *ResultadoFinal*, como mostrado na Figura 9.

Figura 10 – Matriz única de resultados - Bloco 2

Bloco 2			
	article	proceedings	inproceedings
article		0	0
proceedings			1
inproceedings			

Fonte: (MACHADO, 2017)

Figura 11 – Matriz única de resultados - Bloco 3

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1										Bloco 3								
2		@key	@mdate	author	title	pages	year	volume	journal	number	ee	url	editor	booktitle	series	isbn	crossref	authors
3	@key		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	@mdate		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	author		1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
6	title		1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	pages		1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
8	year		1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
9	volume		1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	journal		1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
11	number		1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
12	ee		1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
13	url		1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
14	editor		1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
15	booktitle		1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
16	series		1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
17	isbn		1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
18	crossref		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
19	authors		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	source title		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
21	issue		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	art no		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
23	Page start		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
24	Page end		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
25	Page count		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fonte: (MACHADO, 2017)

O pré-processamento consistiu na separação das matrizes e acréscimo dos separadores nos arquivos CSV, sendo gerado, a partir de cada matriz, dois documentos, um que contém o conteúdo da matriz e outro contém os nomes dos campos. A planilha continha as matrizes referentes aos blocos 2 e 3, a partir destes foram gerados os arquivos *matriz-unica-resultados-1.csv*, *matriz-unica-resultados-campos-1.csv*, *matriz-unica-resultados-2.csv* e *matriz-unica-resultados-campos-2.csv* respectivamente (pois a numeração de blocos na aplicação começa a partir de 0).

Listagem 37 – Arquivo *matriz-unica-resultados-1.csv*

```
1;0;0
0;1;1
0;0;1
```

Fonte: Autor

Listagem 38 – Arquivo *matriz-unica-resultados-campos-1.csv*

```
1 article; proceedings; inproceedings
```

Fonte: Autor

Listagem 39 – Fragmento do Arquivo *matriz-unica-resultados-2.csv*

```
1 1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
2 0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
3 0;0;1;0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0
4 0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
5 0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
6 0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
7 0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
8 0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
9 0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0
10 ...
```

Fonte: Autor

Listagem 40 – Arquivo *matriz-unica-resultados-campos-2.csv*

```
1 @key; @mdate; author; title; pages; year; volume; journal;
    number; ee; url; editor; booktitle; series; isbn; crossref;
    authors; source title; issue; art no; Page start; Page end;
    Page count; Cited by; DOI; Link; EID
```

Fonte: Autor

Foram acrescentados os arquivos *matriz-unica-resultados-0.csv*, *matriz-unica-resultados-campos-0.csv*, *matriz-unica-resultados-3.csv*, *matriz-unica-resultados-campos-3.csv* referentes ao primeiro e quarto blocos respectivamente, estas foram acrescentadas manualmente (pois não estavam disponíveis na planilha) para que a aplicação pudesse reconhecer todos os blocos que foram definidos pela autora no estudo de caso, pois o algoritmo reconhece todas as entradas a partir dos rótulos das matrizes de resultados.

Listagem 41 – Arquivo *matriz-unica-resultados-campos-0.csv*

```
1 1
```

Fonte: Autor

Listagem 42 – Arquivo *matriz-unica-resultados-campos-0.csv*

```
1 dblp
```

Fonte: Autor

Como o primeiro bloco é composto por uma palavra apenas, resultou em uma matriz de uma posição somente (Listagem 41), assim como um arquivo com apenas um campo (Listagem 42).

Listagem 43 – Arquivo *matriz-unica-resultados-campos-3.csv*

```
1; 0
0; 1
```

Fonte: Autor

Listagem 44 – Arquivo *matriz-unica-resultados-campos-3.csv*

```
@href; $
```

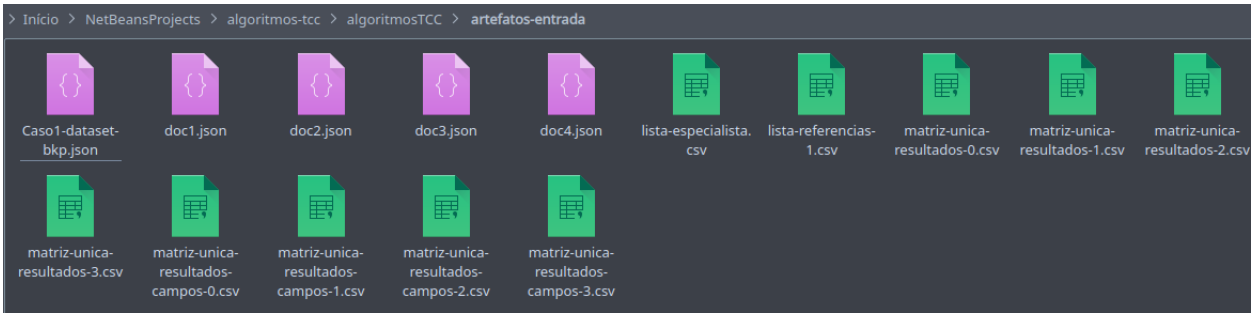
Fonte: Autor

Como as duas palavras que compõem o quarto bloco são visivelmente diferentes, deduziu-se que a sua matriz de resultados seria uma matriz identidade, de duas linhas e duas colunas, não identificando nenhuma relação entre os campos (Listagem 43), acompanhada por um arquivo com os nomes dos dois campos (Listagem 44).

4.3.3 Processamento

O próximo passo consistiu no processamento dos artefatos para a obtenção do esquema conceitual e da lista de mapeamentos. Os artefatos de entrada foram dispostos na pasta *artefatos-entrada* (Figura 12) presente no diretório da aplicação e, ao ser executada a ferramenta, foi digitada a opção 1, que é referente ao modo de execução em blocos, conforme a Figura 13. Todos os artefatos gerados no processo foram gravados na pasta *artefatos-saida* (Figura 14), também presente no diretório da aplicação.

Figura 12 – Diretório da aplicação - arquivos de entrada



Fonte: Autor

No diretório *artefatos-entrada* foram colocados os documentos após o processo manual de pré-processamento.

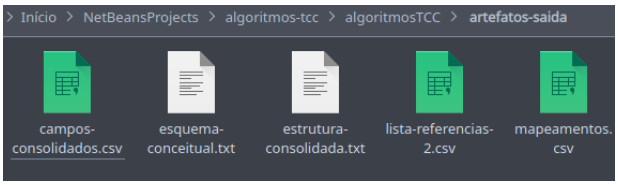
Figura 13 – Interface da aplicação - execução em único bloco

```
EXTRAÇÃO DE ESQUEMAS A PARTIR DE ARQUIVOS JSON


Escolha o modo de operação (1 = em blocos; 2 = único bloco)


1
Extração de esquemas a partir de arquivos JSON executada. Verifique os arquivos de saída gerados.
------------------------------------------------------------------------
```

Fonte: Autor

Logo após, foi executada a aplicação foi selecionada a opção 1, que instrui a ferramenta a operar em modo por blocos.

Figura 14 – Diretório da aplicação - arquivos de saída



Fonte: Autor

Após a mensagem de êxito, a pasta *artefatos-saida* continha os arquivos (artefatos) gerados durante o processo. Cada um deles será mostrado a seguir. Serão mostrados apenas fragmentos dos arquivos em alguns casos, por motivos de tamanho dos mesmos. Os arquivos completos encontram-se na pasta pública do *Google Drive*[14] do autor do trabalho.

Os artefatos gerados pela atividade *consolidar estrutura* foram os arquivos *campos-consolidados.csv* (Listagem 45) e *lista-referencias-2.csv* (Listagem 46), referentes aos artefatos *estrutura unificada* e *lista de referências 2* respectivamente.

Listagem 45 – Arquivo *campos-consolidados.csv*

```
1 dblp
2 article; proceedings
3 @key; @mdate; author; title; pages; year; volume; journal;
     number; ee; url; editor; booktitle; series; isbn; crossref;
     source title; art no; Page start; Page end; Page count;
     Cited by; DOI; Link; EID
4 @href; $
```

Fonte: Autor

[14] https://drive.google.com/open?id=1BAUNyMaZlAaCmvIRFU5nyRe1REnnWoPD

Listagem 46 – Arquivo *lista-referencias2.csv*

```
proceedings;inproceedings
author;authors
number;issue
```

Fonte: Autor

O artefato gerado pela atividade *remontar estrutura* foi o arquivo *estrutura-consolidada.txt* (Listagem 47), referente ao artefato *estrutura consolidada*.

Listagem 47 – Arquivo *estrutura-consolidada.txt*

```
dblp;{
        article;
        proceedings;{
                @key;
                @mdate;
                author;[]
                title;
                pages;
                year;
                volume;
                journal;
                number;
                ee;
                url;
                editor;[]
                booktitle;
                series;{
                        @href;$;
                }
                isbn;[]
                crossref;
                source title;
                art no;
                Page start;
                Page end;
                Page count;
                Cited by;
                DOI;
                Link;
                EID;
        }
}
```

Fonte: Autor

O artefato gerado pela atividade *montar mapeamentos* foi o arquivo *mapeamentos.csv* (Listagem 48), referente ao artefato *mapeamentos*.

Listagem 48 – Arquivo *mapeamentos.csv*

```
$.dblp.proceedings; $..inproceedings; doc3.json
$.dblp.proceedings.number; $..issue; doc4.json
$.dblp.proceedings.author; $..authors; doc4.json
```

Fonte: Autor

O artefato gerado pela atividade *adaptar à notação de agregados* foi o arquivo *esquema-conceitual.txt* (Listagem 49), referente ao artefato *esquema conceitual*.

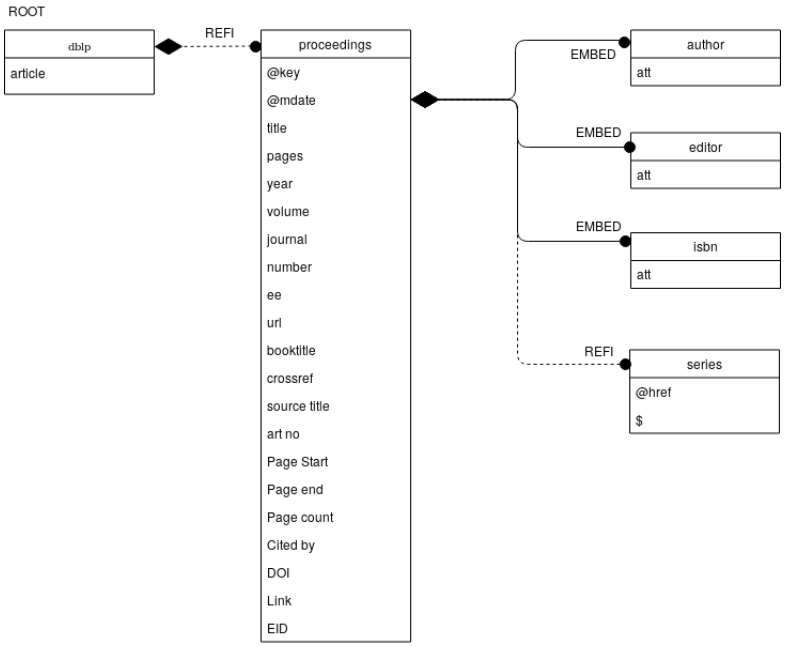
Listagem 49 – Arquivo *esquema-conceitual.txt*

```
-----------------------------------------------------
ROOT
Entidade:dblp
Atributos: article
REFI: , proceedings
-----------------------------------------------------
-----------------------------------------------------
Entidade:proceedings
Atributos: @key, @mdate, title, pages, year, volume, journal,
   number, ee, url, booktitle, crossref, source title, art no,
   Page start, Page end, Page count, Cited by, DOI, Link, EID
REFI: , series
EMBED: , author, editor, isbn
-----------------------------------------------------
-----------------------------------------------------
Entidade:series
Atributos: @href, $
-----------------------------------------------------
```

Fonte: Autor

Através da interpretação manual do arquivo *mapa-conceitual.txt*, apresentado na Listagem 49, obteve-se o esquema gráfico representado na Figura 15.

Figura 15 – Diagrama do Esquema Conceitual - processamento em blocos



Fonte: Autor

4.4 EXECUÇÃO POR ÚNICO BLOCO

Nesta seção serão apresentados os passos efetuados para o processamento dos artefatos segundo o modo de execução por único bloco, onde as comparações são feitas entre todos os campos do arquivo JSON em todos os níveis, de forma automática, sem a necessidade de um especialista para definir blocos.

4.4.1 Pré-processamento

A primeira etapa consistiu no pré-processamento dos artefatos de entrada (*arquivos JSON*, *lista de referências 1* e *matriz única de resultados*) presentes na pasta pública da autora. O processo de pré-processamento foi manual e efetuado através da ferramenta de edição *Sublime Text*[15].

4.4.1.1 Arquivos JSON

Os arquivos JSON foram obtidos a partir de um único arquivo, *Caso2-dataset-bkp.json*, disponível na pasta pública. O arquivo é composto por 4 estruturas, sendo que cada estrutura representa um arquivo JSON distinto. As estruturas foram separadas e então foram obtidos

[15] https://www.sublimetext.com/

4 documentos JSON (*doc1.json*, *doc2.json*, *doc3.json* e *doc4.json*) a partir deste documento apresentado na Listgem 50. O arquivo foi resumido a fim de caber nas páginas do trabalho. As reticências indicam continuidade do conteúdo.

Listagem 50 – Fragmento do arquivo *Caso2-dataset-bkp.json*

```
{
 "Source": {
  "Author": {
   "Author": {
    "NameList": {
     "Person": {
      "First": "Rick",
      "Last": "Cattell"
    }
  }
}
},
"BIBTEX_KeyWords": "cloud-computing, nosql",
"City": "New York, NY, USA",
"JournalName": "SIGMOD Rec.",
...
}
{
  "dblp": {
    "article": {
      "@key": "journals/ijbpim/AlmeidaBF15",
      "@mdate": "2016-01-05",
      "author": [
        "Rafael Almeida",
        "Jorge Bernardino",
        "Pedro Furtado"
...
}
{
  "PubmedArticle": {
    "MedlineCitation": {
      "@Status": "In-Process",
      "@Owner": "NLM",
      "PMID": {
        "@Version": 1,
        "$": 27936191
      },
      "DateCreated": {
        "Year": 2016,
        "Month": 12,
        "Day": "09"
      },
      "DateRevised": {
```

```
        "Year": 2017,
        "Month": "02",
        "Day": 24
      },
...
}
{
  "Authors": "Reniers V., Rafique A., Van Landuyt D., Joosen
     W.",
  "Title": "Object-NoSQL Database Mappers: a benchmark study
     on the performance overhead",
  "Year": 2017,
  "Source title": "Journal of Internet Services and
     Applications",
  "Volume": 8,
  "Issue": 1,
  "Art. No.": 1,
  "Page start": "",
...
}
```

Fonte: (MACHADO, 2017)

4.4.1.2 Lista de Referências 1

A *lista de referências 1* utilizada foi fornecida no arquivo PDF *Estudo de caso 2 - Documentação.pdf*, que é um documento que contém a descrição dos artefatos gerados, arquivos de entrada e resultados obtidos pelo processo desenvolvido pela autora. A Figura 16 exibe a lista obtida a partir do arquivo PDF.

Figura 16 – Primeira Lista de Referências - original da pasta pública

Article – doc1	Pages – doc1, 3	editor – doc 2	Issue – doc 4
Proceedings – doc 2	Year – doc1, 2, 3, 4	Booktitle – doc 2,3	Art no – doc 4
Inproceedings – doc3	Volume – doc1, 2,4	Series – doc 2	Page start - doc4
@key – doc1, 2, 3	Journal – doc 1	Isbn – doc 2	Page end – doc 4
@mdate – doc 1, 2, 3	Number – doc1	Crossref – doc 3	Page count – doc 4
author – doc 1, 3	Ee – doc1, 2, 3	Authors – doc4	Cited by – doc 4
Title – doc1, 2, 3, 4	url – doc1 ,2, 3	Source title – doc4	DOI – doc 4
Link – doc 4	EID - doc 4		

Fonte: (MACHADO, 2017)

O pré-processamento da *lista de referências 1* consistiu em copiar o conteúdo a partir do arquivo PDF, colar na janela principal do *Sublime Text*, substituir os separadores "-" por ponto e vírgula, completar os nomes dos documentos (por exemplo, substituir "*doc1, 2, 3*"

por “*doc1.json*; *doc2.json*; *doc3.json*;”). O resultado do pré-processamento é exibido na Listagem 51

Listagem 51 – Fragmento do arquivo *lista-referencias-1.csv*

```
source;doc1.json
@validyn;doc3.json
author;doc1.json; doc2.json; doc3.json
affiliation;doc3.json
namelist;doc1.json
language;doc3.json
person;doc1.json
publicationtypelist;doc3.json
first;doc1.json
publicationtype;doc3.json
last;doc1.json
...
```

Fonte: Autor

4.4.1.3 Matriz Única de Resultados

Em seguida, a matriz única de resultados foi extraído a partir do arquivo XLSX *Estudo de Caso 2 - Algoritmo 6.xlsx* disponível na pasta pública. A tabela com a matriz única de resultados se localiza na planilha (dentro do documento XLSX) *ResultadoFinal*, como mostrado na Figura 17.

Figura 17 – Parte da matriz única de resultados - Arquivo original

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		source	author	namelist	person	first	last	bibtex_key	city	journalnar	month	pages
3	source		1	0	0	0	0	0	0	0	0	0
4	author			0	0	0	0	0	0	0	0	0
5	namelist				0	0	0	0	0	0	0	0
6	person					0	0	0	0	0	0	0
7	first						0	0	0	0	0	0
8	last							0	0	0	0	0
9	bibtex_keywords								0	0	0	0
10	city									0	0	0
11	journalname										0	0
12	month											0
13	pages											

Fonte: (MACHADO, 2017)

O pré-processamento consistiu na no acréscimo dos separadores nos arquivos CSV, sendo gerado, a partir da matriz, dois documentos, um que contém o conteúdo da matriz e outro contém os nomes dos campos. A partir da matriz existente na planilha foram gerados

dois arquivos, *matriz-unica-resultados-0.csv* (Listagem 52), *matriz-unica-resultados-campos-0.csv*(Listagem 53).

Listagem 52 – Fragmento do arquivo *matriz-unica-resultados-0.csv*

```
1;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;...
0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;...
0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;...
0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;...
...
```

Fonte: Autor

Listagem 53 – Fragmento do arquivo *matriz-unica-resultados-campos-0.csv*

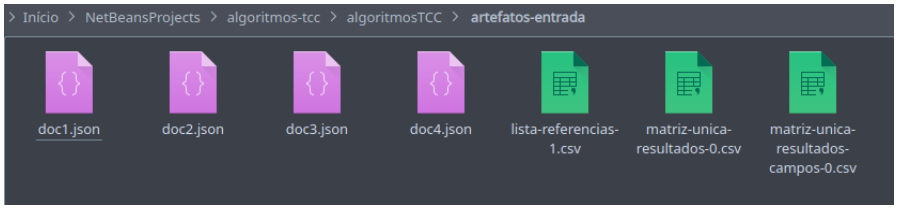
```
source; author; namelist; person; first; last;
   bibtex_keywords; city; journalname; month; pages;
   publisher; sourcetype; standardnumber; tag; title; url;
   volume; year; dblp; article; @key; @mdate; journal; number;
   ee; pubmedarticle; medlinecitation; @status; @owner; pmid;
   @version; $; datecreated; day; daterevised; @pubmodel;
   issn; ...
```

Fonte: Autor

4.4.2 Processamento

O próximo passo consistiu no processamento dos artefatos para a obtenção do esquema conceitual e da lista de mapeamentos. Os artefatos de entrada foram dispostos na pasta *artefatos-entrada* (Figura 18) presente no diretório da aplicação e, ao ser executada, foi digitada a opção 2, que é referente ao modo de execução em único bloco. Todos os artefatos gerados no processo foram gravados na pasta *artefatos-saida* (Figura 20), também presente no diretório da aplicação.

Figura 18 – Diretório da aplicação - arquivos de entrada



Fonte: Autor

No diretório *artefatos-entrada* foram colocados os documentos após o processo manual de pré-processamento.

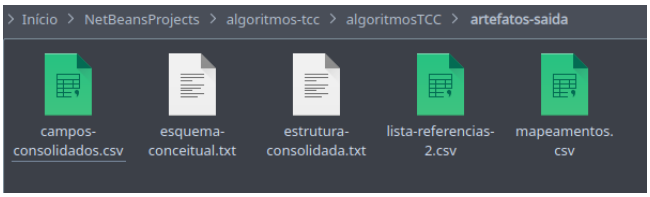
Figura 19 – Interface da aplicação - execução em único bloco

```
EXTRAÇÃO DE ESQUEMAS A PARTIR DE ARQUIVOS JSON


Escolha o modo de operação (1 = em blocos; 2 = único bloco)

2
Extração de esquemas a partir de arquivos JSON executada. Verifique os arquivos de saída gerados.
------------------------------------------------------------------------
```

Fonte: Autor

Logo após, foi executada a aplicação foi selecionada a opção 2 (Figura 19), que instrui a ferramenta a operar em modo de bloco único.

Figura 20 – Diretório da aplicação - arquivos de saída



Fonte: Autor

Após a mensagem de êxito, a pasta *artefatos-saida* continha os arquivos (artefatos) gerados durante o processo (Figura 20). Cada um deles será mostrado a seguir. Serão mostrados apenas fragmentos dos arquivos em alguns casos, por motivos de tamanho dos mesmos. As reticências indicam continuidade do conteúdo.

Os artefatos gerados pela atividade *consolidar estrutura* foram os arquivos *campos-consolidados.csv* (Listagem 54) e *lista-referencias-2.csv* (Listagem 55), referentes aos artefatos *estrutura unificada* e *lista de referências 2*, respectivamente.

Listagem 54 – Fragmento do arquivo *campos-consolidados.csv*

```
1 source; namelist; person; first; last; bibtex_keywords; city;
    journalname; month; pages; publisher; sourcetype;
    standardnumber; tag; title; url; volume; year; dblp;
    article; @key; @mdate; journal; ...
```

Fonte: Autor

Listagem 55 – Fragmento do arquivo *lista-referencias2.csv*

```
1 source;author
2 number;issue
3 source;@source
4 affiliationinfo;affiliation
5 publicationtypelist;publicationtype
```

```
articletitle;articledate
...
```

Fonte: Autor

O artefato gerado pela atividade *remontar estrutura* foi o arquivo *estrutura-consolidada.txt*, referente ao artefato *estrutura consolidada*. A identação na Listagem 56 foi inserida com a finalidade de visualização, pois o resultado é gerado, normalmente, sem identação, em uma linha somente.

Listagem 56 – Fragmento do arquivo *estrutura-consolidada.txt*

```
PubmedArticle;{
        MedlineCitation;{
                @Status;
                @Owner;
                PMID;{
                        @Version;
                        $;
                }
                DateCreated;{
                        Year;
                        Month;
                        Day;
                }
                DateRevised;
                Article;{
                        @PubModel;
                        Journal;{
                                ISSN;{
                                        @IssnType;
                                }
...
```

Fonte: Autor

O artefato gerado pela atividade *montar mapeamentos* foi o arquivo *mapeamentos.csv*, referente ao artefato *mapeamentos* (Listagem 57). Os caminhos JSONPath no arquivo *mapeamentos.csv* foram abreviados para caber na largura da página.

Listagem 57 – Fragmento do arquivo *mapeamentos.csv*

```
$..number;$..issue;doc3.json
$..number;$..issue;doc4.json
$..source;$..author;doc1.json
$..source;$..author;doc2.json
...
```

Fonte: Autor

O artefato gerado pela atividade *adaptar à notação de agregados* foi o arquivo *esquema-conceitual.txt* (Listagem 58), referente ao artefato *mapa conceitual*.

Listagem 58 – Fragmento do arquivo *esquema-conceitual.txt*

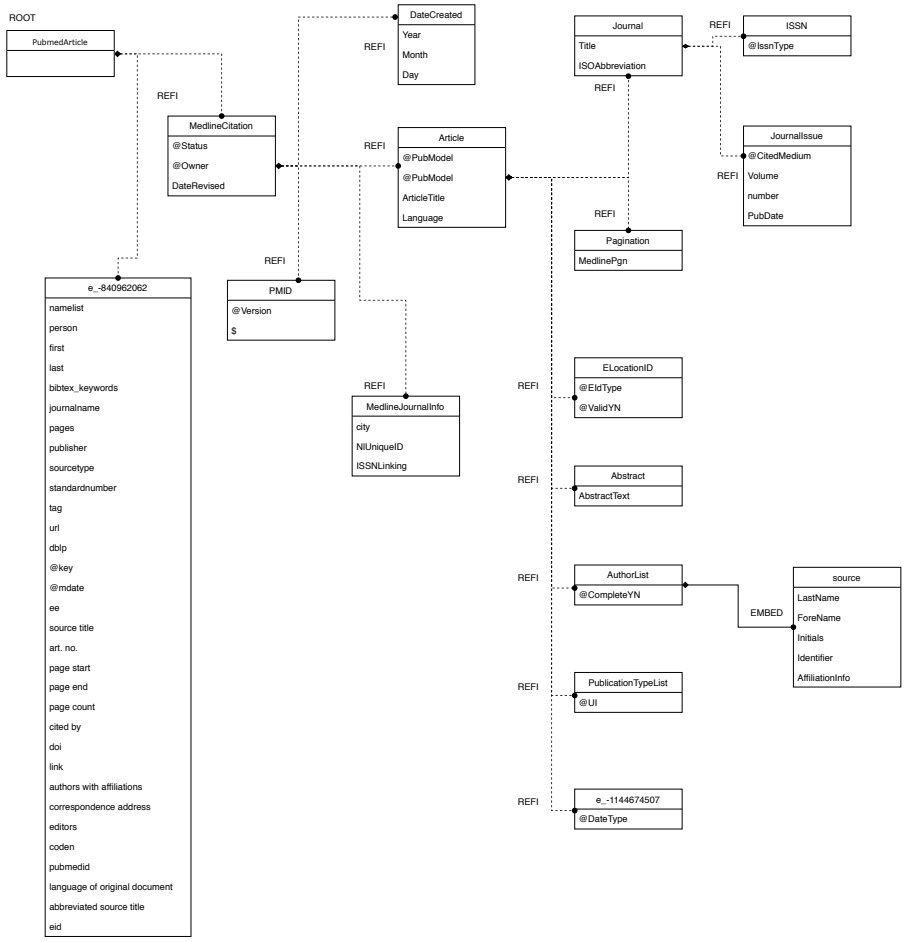
```
-------------------------------------------------
ROOT
Entidade:PubmedArticle
Atributos:
REFI: , MedlineCitation, e_-1276392649
-------------------------------------------------
-------------------------------------------------
Entidade:MedlineCitation
Atributos: @Status,@Owner,DateRevised
REFI: , PMID, DateCreated, Article, MedlineJournalInfo
-------------------------------------------------
-------------------------------------------------
Entidade:PMID
Atributos: @Version,$
-------------------------------------------------
-------------------------------------------------
Entidade:DateCreated
Atributos: Year,Month,Day
-------------------------------------------------
...
```

Fonte: Autor

Através da interpretação manual do arquivo *esquema-conceitual.txt*, obteve-se o mapa representado na Figura 21. O mapa foi elaborado com auxílio da ferramenta online *draw.io*[16].

[16] https://www.draw.io/

Figura 21 – Diagrama do Esquema Conceitual - processamento em único bloco



Fonte: Autor

5 CONCLUSÃO

O presente trabalho teve como objetivo a implementação de algoritmos para extração de esquemas implícitos em fontes de dados JSON, visto que o foco é para bancos de dados NoSQL orientados a documentos, os quais possuem como característica principal a flexibilidade de esquemas, sendo então refinados, durante o processo de desenvolvimento, os algoritmos propostos por Machado (2017), resultando em algoritmos com foco na implementação, complementando os algoritmos propostos pela autora, que se encontram em linguagem de alto nível, sendo então suprimidos detalhes sobre seu funcionamento, importantes para o desenvolvimento de ferramentas e software em geral.

Durante o processo de desenvolvimento dos algoritmos foram elaboradas estratégias para sua implementação que não foram abordadas pela autora na definição do processo de extração, tal como, por exemplo, a adição de um artefato ao modo de execução em blocos que expresse as definições do especialista quanto aos blocos considerados equivalentes, assim como também um detalhamento das atividades propostas pela autora. Os testes da implementação, no qual foram utilizados como base os artefatos e parâmetros do estudo de caso realizado pela autora, apresentou resultados semelhantes aos apresentados pelo estudo de caso feito pela autora, indicando que a implementação realizada no trabalho obedece, até onde foi testada, o mesmo padrão de funcionamento proposto pela autora.

Entretanto, para uma efetiva validação da ferramenta, serão necessários testes em diferentes bases de dados, além das bases testadas no estudo de caso apresentado. Também, como sugestão de melhoria futura para a ferramenta seria a implementação das atividades anteriores às atividades desenvolvidas no trabalho (*consolidar estrutura*, *remontar estrutura*, *montar mapeamentos* e *adaptar à notação de agregados*) ou integração com uma implementação preexistente destas atividades. Outra possível melhoria à ferramenta desenvolvida seria a inclusão da funcionalidade de gerar e editar o mapa conceitual de forma gráfica ou mesmo integrar a ferramenta à um software existente no mercado.

Dessa forma, o presente trabalho apresentou uma implementação para as atividades *Consolidar Estrutura*, *Remontar Estrutura*, *Montar Mapeamentos* e *Adaptar à Notação de Agregados*, propostos pela autora do processo.

REFERÊNCIAS

BREWER, E. A. Towards robust distributed systems. In: PODC. **Anais...** [S.l.: s.n.], 2000. v.7.

BREWER, E. CAP Twelve years Later: how the. **Computer**, [S.l.], n.2, p.23–29, 2012.

BRITO, R. W. Bancos de dados NoSQL x SGBDs relacionais: análise comparativa. **Faculdade Farias Brito e Universidade de Fortaleza**, [S.l.], 2010.

CHANG, F. et al. Bigtable: a distributed storage system for structured data. **ACM Transactions on Computer Systems (TOCS)**, [S.l.], v.26, n.2, p.4, 2008.

HAN, J. et al. Survey on NoSQL database. In: 2011 6TH INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND APPLICATIONS. **Anais...** [S.l.: s.n.], 2011. p.363–366.

HASHEM, H.; RANC, D. Evaluating NoSQL document oriented data model. In: IEEE 4TH INTERNATIONAL CONFERENCE ON FUTURE INTERNET OF THINGS AND CLOUD WORKSHOPS (FICLOUDW), 2016. **Anais...** [S.l.: s.n.], 2016. p.51–56.

INDRAWAN-SANTIAGO, M. Database research: are we at a crossroad? reflection on nosql. In: INTERNATIONAL CONFERENCE ON NETWORK-BASED INFORMATION SYSTEMS, 2012. **Anais...** [S.l.: s.n.], 2012. p.45–51.

JOVANOVIC, V.; BENSON, S. Aggregate data modeling style. In: SOUTHERN ASSOCIATION FOR INFORMATION SYSTEMS CONFERENCE. **Proceedings...** [S.l.: s.n.], 2013. p.70–75.

MACHADO, F. T. d. S. **Um Processo Para Extração De Esquemas Conceituais Em Fontes De Dados Json Baseado Em Técnicas De Similaridade De Texto**. 2017. Dissertação de Mestrado — UFSM.

NOSQL Databases. Acessado em: 18 de julho de 2019, http://nosql-database.org/index.html.

SADALAGE, P. J.; FOWLER, M. **NoSQL distilled**: a brief guide to the emerging world of polyglot persistence. [S.l.]: Pearson Education, 2013.

TUDORICA, B. G.; BUCUR, C. A comparison between several NoSQL databases with comments and notes. In: ROEDUNET INTERNATIONAL CONFERENCE 10TH EDITION: NETWORKING IN EDUCATION AND RESEARCH, 2011. **Anais...** [S.l.: s.n.], 2011. p.1–5.

VIEIRA, M. R. et al. Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de big data. **Simpósio Brasileiro de Bancos de Dados**, [S.l.], 2012.