FuzzyScript: Um método de consulta a banco de dados de documento usando informação imperfeita

Leandro Brito do Nascimento Nogueira

Dezembro / 2022

Dissertação de Mestrado em Ciência da Computação

FuzzyScript: Um método de consulta a banco de dados de documento usando informação imperfeita

Esse documento corresponde à Dissertação apresentada à Banca Examinadora para Defesa de Dissertação no curso de Mestrado em Ciência da Computação da Faculdade Campo Limpo Paulista.

Campo Limpo Paulista, 2 de março de 2023.

Leandro Brito do Nascimento Nogueira

Luiz Mariano Val Del Cura (Orientador)

Ficha catalográfica elaborada pela

Biblioteca Central da Unifaccamp

N712L

Nogueira, Leandro Brito do Nascimento

FuzzyScript: um método de consulta a banco de dados de documento usando informação imperfeita / Leandro Brito do Nascimento Nogueira. Campo Limpo Paulista, SP: Unifaccamp, 2022.

Orientador: Prof. Dr. Luis Mariano Del Val Cura

Dissertação (Programa de Mestrado Profissional em Ciência da Computação) – Centro Universitário Campo Limpo Paulista – Unifaccamp.

1. Bancos de dados. 2. NoSQL. 3. Banco de dados de documentos. 4. Conjuntos nebulosos. 5. Informações imperfeitas. I. Del Val Cura, Luis Mariano. II. Centro Universitário Campo Limpo Paulista. III. Título.

CDD – 005.75

Agradecimentos

Primeiramente a Deus pela oportunidade de cursar o mestrado e por todo esse tempo ter me guiado a fazer esse projeto, sendo minha base e sustentação durante todo o projeto.

Aos meus pais, José Brito do Nascimento e Luciana Brito do Nascimento (ambos em memória) por me proporcionarem educação, de forma que eu possa seguir a vida da melhor maneira possível.

A minha avó Luciléia Dornelas do Nascimento (em memória), por ter me criado e dado a educação necessária para ser quem sou hoje, me ensinando a nunca desistir, mas persistir naquilo que é correto.

Aos meus tios, Lucélia Brito do Nascimento, Luciano Brito do Nascimento, Lúcio Brito do Nascimento, Lúcio Mauro Brito do Nascimento e Lorival Brito do Nascimento, por me apoiarem na decisão de prestar o mestrado e sempre me motivarem a concluí-lo.

Ao meu pastor Denílson da Silva Roque, por me motivar nos estudos não deixando que a preguiça ou desânimo me dominassem, assim como, por me instruir a expandir minha área de estudos como o doutorado.

Ao meu orientador o Professor Dr. Luiz Mariano Val Del Cura, que durante esse trabalho me ajudou a pesquisar a fundo sobre esse tema, abrindo horizontes para evolução e novos pensamentos.

E por fim quero agradecer aos meus amigos e colegas do programa de mestrado Renato, Igor, André, Márcio, Raphael e Jader por todo esse tempo que estivemos juntos.

Resumo.*Esta dissertação propõe um método de consulta em banco de dados de documentos utilizando a lógica fuzzy, o intuito de utilizar esse método é para que haja uma cobertura maior de dados obtidos no retorno da consulta. Um dos motivos de existir essas perdas de cobertura é o uso de métodos lógicos, que não são flexíveis e muito restritivos na consulta em banco de dados, que é o padrão usado atualmente. Com a lógica fuzzy podemos explorar mais condições de consulta, que podem ser aplicadas ao se realizar uma busca no banco de dados de documentos. Aumentando as possibilidades de se recuperar informações do banco de dados sem perder dados que podem ser importantes para usuário que está pesquisando.*

O trabalho propõe, com base na bibliografia selecionada, um método que utiliza a lógica fuzzy nos comandos de consulta, mostrando o problema e as necessidades disto de forma concreta. Será também mostrada uma API que é responsável por captar as informações imperfeitas colocadas pelo usuário da API, para fazer o tratamento necessário antes de ir ao banco, e quando o banco retornar os dados da consulta ela também vai aplicar no dado retornado uma porcentagem de associação que ela tem com a informação imperfeita pesquisada.

Por fim esta dissertação apresenta os testes de utilização da API com um comparativo de consultas realizadas pela API e outra utilizando métodos tradicionais por meio de comandos normais de consulta, para provar o valor da API. E com isso será apresentado o documento JSON nebuloso com a porcentagem de associação de cada dado, com base no critério pesquisado pelo usuário.

Palavras-chave: *Bancos de Dados, NoSQL, Banco de Dados de Documentos, Conjuntos Nebulosos, Informações Imperfeitas.*

Abstract.*This dissertation proposes a method of querying a document database using fuzzy logic. One of the reasons for these coverage losses is the use of logical methods, which are not flexible and very restrictive in the database query, which is the standard currently used. With fuzzy logic we can explore more query conditions, which can be applied when performing a search in the document database. Increasing the possibilities of recovering information from the database without losing data that may be important to the user who is searching.*

The work proposes, based on the selected bibliography, a method that uses fuzzy logic in query commands, showing the problem and the needs of this in a concrete way. An API will also be shown that is responsible for capturing the imperfect information placed by the API user, to carry out the necessary treatment before going to the bank, and when the bank returns the query data it will also apply a percentage of association to the returned data. it has with the imperfect information searched.

Finally, this dissertation presents the API usage tests with a comparison of queries performed by the API and another using traditional methods through normal query commands, to prove the value of the API. And with that, the hazy JSON document will be presented with the percentage of association of each data, based on the criteria searched by the user.

Keywords: *Databases, NoSQL, Document Database, Fuzzy Sets, Imperfect Information.*

Sumário

Glossário

- **BDD** - Document Oriented Database
- **API** - Application Program Interface
- **JSON** - JavaScript Object Notation
- **XML** - eXtensible Markup Language
- **REST** - Representational State Transfer
- **SGML** - Standard Generalized Markup Language
- **HTML** - HyperText Markup Language
- **OME** - Object Manager Enterprise
- **NQ** - Native Queries
- **XED** - XML Enabled Database
- **NXD** - Native XML Database
- **GUI** - Graphical User Interface
- **ANTLR** - Another Tool for Language Recognition
- **ZB** - Zettabyte
- **PB** - Petabyte
- **MVVM** - View-Model-Model-View
- **MVC** - Model-View-Controller
- **XSD** - XML Schema Definition

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O uso de Banco de Dados de Documentos (BDD) cresceu ao longo destes últimos anos, fundamentalmente por seus recursos serem utilizados pelas novas tecnologias de tipo *Big Data*. Essas novas tecnologias combinam a eficiência de suas arquiteturas, a escalabilidade e a agilidade das consultas executadas. Estes bancos têm como resultado das consultas documentos XML ou JSON (Nurseitov et al. (2009)), um bom fator que também contribui para a sua aceitação e uso pela comunidade. Essa aceitação acontece pelo uso desses documentos em diversos sistemas e aplicativos, para a obtenção de dados em diferentes requisições.

Nós, humanos, conseguimos avaliar e processar categorias de dados incertos e ambíguos. Termos de linguagem relativamente vagos como "rápido e lento" , "quente e frio" , "jovem e velho" , etc. são usados para se julgar algum parâmetro que está sendo avaliado. Essa categoria de comunicação é definida como "informação imperfeita" (Ma (2007)).

Quando tentamos utilizar essa categoria de informação para realizar uma busca em um banco de dados, em especial de documentos, o banco não consegue interpretar essa informação imperfeita como aconteceria com um dado preciso. Com este dado preciso é possível, na lógica booleana, definir se esse valor no banco é verdadeiro ou falso, ou seja, se a resposta é sim ou não.

Por esta razão, devemos primeiramente entender como vamos organizar esse con-

junto de dados de informações imperfeitas. A lógica e os conjuntos nebulosos (*fuzzy*) definem o embasamento matemático para essa categoria de informação imperfeita (Zadeh et al. (1996)). Geralmente cada consulta permite operadores utilizando a lógica booleana, isto é com valores verdadeiros e falsos. Neste caso, com lógica nebulosa, vamos permitir que cada atributo possa ter valores entre 0 e 1. Desta maneira, vamos elaborar um método que possa converter esses atributos em uma linguagem de consulta nebulosa de um BDD.

Na área de bancos de dados existem muitos trabalhos que exploram a utilização de informação imperfeita. Em bancos de dados relacionais é definida a informação imperfeita como atributos das tabelas, no entanto, também aparecem definições associadas a informação imperfeita nas tuplas das relações (Ma & Yan (2007b); Ma et al. (2000); Chen & Jong (1997); Prade & Testemale (1984); Raju & Majumdar (1988)).

Em bancos de dados orientados a objetos também foi explorada a informação imperfeita similar a bancos de dados relacionais, mas associando também esse tipo de informação a classes, assim como na relação de um objeto com a herança. (Shukla et al. (2011); Yazici & Koyuncu (1997); Bordogna et al. (1999); Umano et al. (1998); Zicari (1990))

Nos trabahos de (Castelltort & Laurent (2014); Pivert et al. (2014, 2016); Ponsoni (2019)), foi abordado a área de bancos de dados NOSQL, mostrando uma asssociação de valores da informação imperfeita tanto em campos como vértices e arestas. E também a inserção de informação imperfeita no banco de dados e em possíveis consultas.

Vários trabalhos exploram as definições nebulosas em bancos de dados semiestruturados como XML (Ma & Yan (2016); Marrara & Pasi (2016); Jin & Veerappan (2010)). Outros trabalhos apresentaram linguagens de consulta nebulosa sobre esses bancos como XQuery para XML (Fredrick & Radhamani (2009)). Note-se que esses elementos associados a XML estão fundamentalmente associados a documentos semiestruturados e não a um BDD NoSQL.

Existem vários bancos de dados de documentos: *OrientDB* *CouchDB* , *MongoDB* , dentre outros (Tesoriero (2013); Suter (2012); Lennon (2009)). Para este projeto, optamos por utilizar o banco MongoDB. Por conta de sua facilidade em lidar com os atributos dos documentos.

É possível inicialmente colocar alguns questionamentos referente a ideia apresentada, optamos por colocar algumas abaixo com suas respectivas respostas:

1. Quais os problemas que pretendemos resolver com essa pesquisa?

   R: O principal problema que procuramos resolver com essa pesquisa é utilizar a informação imperfeita como parâmetro de busca dentro do BDD.

   Isto é não depender da lógica booleana para se obter os resultados, mas sim a lógica difusa com o intuito de não haver perda de dados e futuramente obter um número maior de resultados. Porém, esses resultados ainda vão precisar ser relevantes ao usuário, para não perder a credibilidade da consulta.

2. Como será feito a interpretação da informação imperfeita que esta sendo colocada na consulta?

   R: A interpretação será feita da seguinte forma, vamos categorizar a informação imperfeita inserida pelo usuário, essa categorização será feita por uma API (Interface de Programação de Aplicação), que vai obter o tipo de informação imperfeita por meio de um padrão pré-estabelecido. A informação imperfeita pode ser, por exemplo, termos como "jovem" e "velho", a API vai categorizar e transformar essa informação em termos precisos, e assim realizar a consulta normalmente.

3. Esse método que irá usar a informação imperfeita, vai ser melhor do que as consultas já realizadas de forma tradicional?

   R: Sim, esse inclusive é um dos resultados que buscamos com essa pesquisa. Pois assim vamos conseguir ampliar os termos utilizados na consulta, conseguindo mais resultados importantes e sem perdas.

   Esse método propõe o uso de Javascript e JSON dentro para realizar as buscas em BDD utilizando a lógica *fuzzy*. Assim, explorando o uso da informação imprecisa e, por consequência, trazer resultados que refletem mais ao que o usuário final deseja ter de informação.

4. Que tipo de informação imperfeita esse método vai conseguir aproveitar? E quais não serão?

   R: Essa dissertação apresenta 5 categorias distintas de informação imperfeita, onde após detalhar cada uma em um capítulo futuro, inicialmente vamos trabalhar com 3 categorias e apresentar como lidaremos com as 2 categorias restantes

em trabalhos futuros.

1.1. Proposta

A proposta desta dissertação é mostrar um método que facilite a forma que as consultas em bancos de dados de documentos são feitas utilizando a informação imperfeita nos termos da consulta. Para estas consultas utilizaremos o documento JSON. A partir desse modelo de documento a informação imperfeita será inserida, interpretada e depois ser encaminhada ao banco. No capítulo 3 faremos uma maior exposição dos documentos JSON.

Não vamos levar a informação imperfeita para dentro do banco, antes disso vamos tratar a informação imperfeita através de uma API. A API vai atuar como um intermediador entre o usuário e o banco de dados. Ela vai garantir que a informação imperfeita enviada pelo usuário seja interpretada corretamente.

A apresentação dos dados ao usuário também será feita de uma forma mais elaborada, os dados serão acompanhados de uma porcentagem mostrando o quanto o dado teve de afinidade com a pesquisa do usuário que utilizou a API.

1.2. Contribuição

O objetivo desta dissertação é o contribuir com tratamento de informação imperfeita através de consultas embasadas em conjuntos nebulosos em bancos de dados de documentos.

Como contribuições desde trabalho devemos destacar:

1. A definição formal de um documento JSON nebuloso: Neste projeto vamos oferecer um documento JSON que se corresponda com as informações imperfeitas na consulta e também na resposta das consultas.
2. A definição de um método que chamamos de *FuzzyScript*: Por meio de uma API que atua como intermediária entre um cliente JavaScript e o BDD.

Será apresentada a API, bem como alguns trechos da implementação inicial e testes realizados com o resultado atual da pesquisa e, por fim, a conclusão e os trabalhos futuros.

A API é responsável por todas essas transformações sem modificar o BDD. Vamos revisar a conceituação de Banco de Dados Orientados a Documentos e apresentar uma revisão bibliográfica de trabalhos relacionados ao projeto sendo descrito. Também será discutido a caracterização de informação imperfeita com a apresentação dos conceitos básicos relacionados à Teoria de Conjuntos Nebulosos.

Propomos de maneira prática um modo de pesquisar em BDD, sem estar sob as amarras de condições lógicas existentes que influenciam diretamente no resultado das buscas. Assim equilibrando ou trazendo um resultado mais satisfatório aos usuários da aplicação.

Uma das áreas que abordaremos neste trabalho, que pode se beneficiar bastante, é a da medicina, visto que a porcentagem de uso da informação imperfeita nessa área é bem grande em diferentes setores. Como, por exemplo, usar a informação imperfeita para buscar a eficiência de um medicamento no combate a uma doença, o composto mais efetivo para a manipulação de um remédio, etc.

Outra categoria de contribuição que podemos mencionar também é a tecnológica, visto que não ha muitos trabalhos que usam o Javascript como intermediário entre o banco de dados e o usuário que envia a informação imperfeita ao banco.

As abordagens e conceitos encontrados aqui podem servir de ponte para novos trabalhos e propostas que possam explorar esse mesmo recurso em outras áreas, colocando em prática o uso de novas linguagens de programação para realizar a comunicação com BDD.

1.3. Orgranização do Trabalho

Esta dissertação é composta pelos seguintes capítulos:

No *Capítulo 1*, é apresentada a proposta do trabalho, em que ele contribui, os desafios e a organização das informações a serem apresentadas.

No *Capítulo 2*, apresentamos o conceito de informação imperfeita e classificamos algumas categorias de informação imperfeita, aplicados em BDD, para melhor entendimento do problema que estamos abordando neste projeto.

No *Capítulo 3*, são apresentados os fundamentos e conceitos de BDD, junto de 3

exemplos de BDD famosos pela comunidade. Também são mostrados nos subcapítulos os conceitos dos documentos XML e JSON, e também sua aplicação nos BDD e, por fim, um comparativo entre esses dois modelos.

No *Capítulo 4*, é mostrado os trabalhos relacionados desta dissertação com o processo de mapeamento sistemático realizado, mostrando uma tabela com a contribuição de cada trabalho usado.

No *Capítulo 5*, é mostrado os objetivos da pesquisa, colocando com mais detalhes as contribuições propostas na introdução.

No *Capítulo 6*, apresentamos a API com os métodos já existentes e também o motivo de utilizarmos Javascript para escrevê-la, além dos benefícios dessa escolha.

No *Capítulo 7*, é apresentado os resultados que obtivemos com o uso da API e também quais os próximos passos para otimizar os métodos, pesquisa e obtenção de resultados da mesma.

No *Capítulo 8*, é colocada a conclusão que chegar até o momento e o que pretendemos fazer no futuro junto de outros trabalhos.

Capítulo 2

Informação Imperfeita e sua Aplicação em Banco de Dados de Documentos

Neste capítulo, é explicado o que é a informação imperfeita, em seguida é apresentado 5 categorias que a informação imperfeita pode receber. Também são abordados os casos em que essa categorização ocorre dentro de um banco de dados de documentos. Por fim, é explicado sobre a lógica nebulosa e os conjuntos nebulosos.

2.1. Informações Imperfeitas

Informações imperfeitas são representadas por dados que não têm um sentido completo, que podem ser imprecisos ou vagos no entendimento, elas são onipresentes de acordo com (Parsons (1996)). Quase todas as informações que temos sobre o mundo real não são certas, completas ou precisas. Um dos propósitos que os sistemas de informação tem é modelar o mundo real; para que isso se cumpra, eles devem lidar com a informação imperfeita também.

É necessário um estudo persistente não só em certas informações, mas também no contexto geral para abranger boa parte desse problema. Porém, a maioria dos trabalhos em bancos de dados estuda uma pequena parte do problema conforme a necessidade.

Ao tentar recuperar um tipo de informação imperfeita em um banco de dados é preciso considerar principalmente o raciocínio que tal informação terá na tomada de decisão e não o quanto ela está imperfeita no conjunto de informações que ela esta arma-

zenada. Também é necessário classificá-la, para que informação imperfeita possa ter um pouco mais de referência a elementos mais exatos do mundo real.

Por exemplo, o gerente de uma empresa de varejo deseja achar no banco de dados os vendedores com melhor desempenho nos últimos 10 dias. Para isso funcionar ele deverá colocar como critério os vendedores que tiveram vendas "boas" onde o termo "boas" será considerado como um valor igual ou maior que 4000 dentro de 10 dias.

Note que não definimos o valor real do termo "boas" apenas colocamos que os valores aceitos serão aqueles que passarem pela condição de serem maiores ou iguais a 4000, assim deixando o termo "boas" catalogado como informação imperfeita, a seguir observaremos as possibilidades de se classificar uma informação imperfeita.

2.2. Classificação das Informações Imperfeitas

Muitos estudos ao longo dos anos apresentaram muitas tentativas de se criar uma categorização classificasse as categorias de informação imperfeita e com isso poder definir uma relação entre elas. Com isso, muitas dessas tentativas conseguiram chegar a um parecer consistente e intuitivo, até chegando a mencionar que não existe uma forma de classificação melhor. Porém, diversos estudos estão conseguindo levantar possíveis classificações e padrões para especificar cada categoria de informação imperfeita. Por exemplo, no estudo feito por (Parsons (1996)), conseguimos ver que pode-se classificar a informação imperfeita em 5 tipos principais: *imprecisão*, *incerteza*, *ambiguidade*, *inconsistência* e *vagueza*. Veremos como são cada uma delas, as diferenças entre elas e quais os exemplos que elas podem ter de aplicação em nosso trabalho nas sub-seções a seguir:

2.2.1. Imprecisão

Imprecisão é uma característica ou particularidade daquilo que é impreciso, com ausência de precisão, exatidão e/ou clareza. A imprecisão ocorre quando existe um valor que não da para ser medido de forma precisa. Ela pode também ser avaliada como se fosse um intervalo, ou seja, ela pode ser um conjunto de números que terá um valor mínimo e máximo de determinada informação.

Seguindo esse raciocínio, poderíamos ter como exemplo uma consulta para se obter a idade uma pessoa chamada "Sílvia". Em bancos de dados tradicionais, seria

necessário apenas retornar o valor preciso que corresponde a idade indicando as possíveis.

Entretanto, há a possibilidade de essa idade estar armazenada de uma forma imprecisa, como um conjunto de idades seguindo um formato similar a este "{40, 55, 60, 72, 80}" podendo a idade a ser cerca de uma dessas, no conjunto.

Além de vir acompanhado de mais uma descrição como: "Sílvia pode ter cerca de 80 anos", ou "a idade de Sílvia pode ser entre 40 e 80 anos", assim gerando o intervalo com qualquer sequência de valores que vão ser associados.

2.2.2. Incerteza

A incerteza é quando não se pode se definir com toda certeza a precisão de determinado dado, ou seja, não da para ter confiança total neste dado. É uma condição ou natureza do que é incerto, qualidade daquilo que incita dúvida ou indecisão.

A um dado incerto pode ter sua verdade estimada usando a probabilidade de alguma proposição que seja verdadeira, utilizando as possibilidades. Normalmente, são tratadas anexando um número, este que representa a medida de certeza do elemento e a manipulação desse número vai depender de toda a ideia por trás dele. Por exemplo, em uma empresa temos uma funcionária chamada Ana e queremos saber o salário dela, porém só temos a seguinte informação: "Existe 70 por cento de certeza que o salário da Ana seja de 2 mil reais".

E com isso, poderíamos formar, a partir da indução, um possível conjunto de valores que poderiam compor o armazenamento ou consulta do salário de Ana ("{(1000,60), (2000,70),(3000,30),(4000,20), (5000,10)}") Diferente da imprecisão aqui o conjunto de valores sempre será acompanhando da porcentagem que o valor tem de ser verdadeiro.

2.2.3. Vagueza

Quando a informação possui falta de clareza, ela é dita como uma informação vaga. Conforme o dicionário, é uma característica ou condição do que é vago; que se apresenta de maneira incerta ou imprecisa. Ela é mais caracterizada como um termo dentro de um conjunto de informações onde o mesmo se encontra incompleto, em alguns pode estar representado por algum fragmento de informação ex: "alto", "baixo".

E nesses fragmentos podemos ampliar o sentido desta vagueza os intensificando

como, por exemplo: “muito alto”, “muito baixo”, “pouco alto”, e “pouco baixo”. Podemos também adicionar termos de negação nestes mesmo fragmentos ficando algo mais amplo para ser explorado como: “não tão alto” e “não tão baixo”. E, por fim, podemos ligar esses termos com operadores lógicos como “e” e “ou”, assim aumentando as possibilidades e complexidade das combinações.

Outro termo que podemos encaixar no contexto de vagueza são os valores apresentados como nulos (ou NULL no banco de dados). Esse termo demostra a falta de valor em determinada informação ou conjunto de informações que podem estar sendo buscados no banco de dados. Visto que pode ser um valor indefinido, valor desconhecido ou até mesmo um valor que não foi declarado ainda. Considere que ao buscar um dado de determinado usuário, a coleção de documentos encontrados seja essa: User:{Nome:“Ana”, Idade: 25, Sexo: NULL}, esse é um dos desafios que vamos encontrar mais no banco do que na formação de uma consulta ao banco de dados.

2.2.4. Ambiguidade

Ambiguidade é quando a informação pode ter diversos sentidos, que possui diversas formas de se ver determinado valor. Conforme o dicionário é uma duplicidade de sentidos; característica de alguns termos, expressões, sentenças que expressam mais de uma acepção ou entendimento possível.

A diferença dela para outras categorias de informação imperfeita como imprecisão ou até mesmo a incerteza é que na ambiguidade temos um valor real de determina informação, estando imperfeito nesse caso é o entendimento concreto da mesma visto que há várias interpretações dela.

Podemos utilizar o seguinte exemplo para representar esse caso tanto na realidade quanto em banco de dados, suponhamos que existe o seguinte dado: ”Guilherme está gordo”. Para uma pessoa “x”, o peso de Guilherme pode ser de 100 quilos, e para a pessoa “y” o peso pode ser de 110 quilos; nesse momento, não há uma forma de se determinar qual das duas interpretações está correta. Essa informação necessita ser clara para haver uma compreensão verdadeira, e para isso podemos ver todos os detalhes da mesma, como que haja algo que traga mais clareza e determine um sentido único para a informação.

2.2.5. Inconsistência

Inconsistência principalmente no banco de dados pode ser considerada com uma falha que gera falta de consistência, de estabilidade ou de firmeza, conforme o dicionário. Isso acontece devido a uma possível falha de integridade da base de dados, por conta de sincronização ou de atualização, o que pode gerar resultados diferentes de uma mesma consulta realizada.

Quando essas operações dão errado o banco perde a integridade dos dados afetados, visto que eles podem ter sido atualizados em dois lugares diferentes assim perdendo a originalidade do dado, ou foi atualizada em apenas um local e não replicado nos demais ou ser atualizada em todos, mas não em um específico.

Um exemplo bem simples disso é quando temos uma empresa de correspondências que faz entregas em uma cidade específica, la no banco de dados existe o registro das encomendas enviadas e em cada lugar que a encomenda passa é atualizada o local que ela percorreu.

Caso da matriz até o lugar que a encomenda se encontra não tenha se atualizado corretamente as informações de local, pode acontecer de a enconda ter sido entregue, mas ter sido perdido seu paradeiro nos registros ou ela constar que saiu para entrega, mas não saiu do centro de distribuição, assim causando transtorno ao usuário que precisa da encomenda. Essa categoria de imperfeição está mais ligado a problema de modelagem de dados do que problema na hora da consulta do que na forma de armazenamento do dado de forma individual.

2.3. Loǵica Nebulosa e Conjuntos Nebulosos

A lógica nebulosa (ou fuzzy) é a lógica que usa o raciocínio por aproximação ao invés do raciocínio exato (Gomide & Gudwin (1994)), o termo “fuzzy”, vem de origem inglesa, ele significa incerto, vago ou impreciso. O conceito dessa lógica foi apresentado incialmente em 1965 no formato de publicação pelo matemático Lotfi Asker Zadeh. Essa lógica tem como base desenvolver métodos e algorítimos para atuar como um solucionador de problemas. Esses problemas antes eram intratáveis por métodos tradicionais, por se limitarem a ter o resultado somente com modelos exatos.

A lógica nebulosa trabalha com valores além do sistema tradicional de lógica binária, a lógica binária assume somente dois valores: verdadeiro (1) e falso (0). Já na lógica nebulosa podemos ter um elemento ou um conjunto de elementos dentro de um intervalo. Isto acontece porque os valores podem ser expressos por expressões linguísticas (ex: muito alto, pouco alto, muito verdade, muito falso, etc.), assim gerando um subconjunto dentro desse valor.

Em (Zadeh (1965)) é nos apresentado algumas definições baseadas na teoria dos conjuntos clássica:

Definição I: Seja U o conjunto universo e A um subconjunto de U, um elemento x pertence a U (x $\in U$), com isso temos a função $\mu A(x)$, onde um conjunto $A \subseteq U$ é dado por: $\mu A(x)$ = { 1 se $x \in A$ ou 0 se $x \notin A$}, por fim resultando na função $\mu A(x)$: U $\rightarrow$ { 0,1 }. $\mu A(x)$ = 1 indica que o elemento x está em A, enquanto $\mu A(x)$ = 0 indica que x não é elemento de A.

Na teoria dos conjuntos tradicional o elemento tende a pertencer a um conjunto ou ele não pertence. Porém, existem situações em que o grau de pertinência entre os elementos e conjuntos não é precisa, ou seja, não sabemos dizer se um elemento pertence efetivamente a um conjunto ou não.

O que podemos fazer nesse caso é definir qual elemento do conjunto universo se encaixa com mais compatibilidade ao termo que caracteriza o subconjunto. Por exemplo, consideremos o subconjunto dos números reais "próximo de 2": C = { x $\in$ R: se x é próximo de 2 }.

Podemos então questionar se os números 7 e 2,001 pertencem ao conjunto A. A resposta é incerta, pois não sabemos até que ponto podemos dizer com certeza o quanto um número está próximo de 2. O que pode ser afirmado é que 2,001 está mais próximo de 2 do que 7. A definição a seguir formaliza os conceitos da teoria dos conjuntos fuzzy com a noção de subconjunto fuzzy.

Definição II: Seja U o conjunto universo. Um subconjunto F(fuzzy) de U é caracterizado por uma função: $\mu F(x)$: U $\rightarrow$ [0,1], essa função é chamada de função de pertinência ou grau de pertinência do subconjunto F(fuzzy).

O valor de $\mu F(x)$ indica o grau que um determinado elemento x do conjunto U está no conjunto F. Assim como na teoria dos conjuntos tradicional $\mu F(x)$ = 0 e $\mu F(x)$ = 1 vão indicar a não pertinência e a pertinência completa do elemento x no conjunto F. O que mudou de uma definição para outra, foi que ampliamos o domínio da função característica, que antes era um conjunto $\{0,1\}$ para um intervalo [0,1].

Partindo da **Definição II**, vamos voltar ao exemplo pontuado acima, existe um subconjunto dos números reais "próximo de 2": C = $\{$ x $\in$ R: se x é próximo de 2 $\}$.

Se utilizarmos a função $\mu C(x)$: U $\rightarrow$ [0,1] , para associar cada valor x real ao valor próximo 2 teremos a expressão $\mu C(x)$ = $\{$ (1 - | x - 2|) se $x \in [1,3]$ ou 0 se $x \notin [1,3]\}$. Com isso podemos agora calcular o quanto 7 e 2,001 estão próximo de 2, obtendo como resultado $\mu C(2,001)$ = 0,999 e $\mu C(7)$ = 0. Concluímos que 2,001 é um número real próximo de 2 com grau de pertinência 0,999 e 7 não pois tem o grau de pertinência 0.

Podemos usar essa lógica para modelar qualquer sistema, mas antes de tudo é necessário pontuar o comportamento do sistema. Isso é feito para se desenhar com exatidão as etapas que vão ser efetuadas, seja uma simulação, um teste de mesa ou uma análise mais aberta dos casos. Para ser colocado em prática, a ideia inicial é que essa modelagem seja feita a partir do conhecimento de um especialista, em vez de modelar o sistema sem uma experiência prévia.

É possível representar a variação dos conjuntos de diversas formas, aqui vamos considerar dois modelos (sendo os mais utilizados), estes tem como base a função trapezoidal e a função triangular. As funções têm como papel representar os intervalos de valores no conjunto U (universo) e o valor que referencia cada elemento.

Na Figura 2.1 é apresentado a função trapezoidal, dentro dessa função temos um conjunto de 4 valores (a, b, c, d). Os elementos desse conjunto são compostos por (a, d) sendo considerados como elementos de limite e (b, c) considerados como intervalo na função. O papel deles é formar os valores com grau de pertinência mais próximo de 1.

Na função triangular temos um conjunto de 3 valores (a, b, c). Os elementos desse conjunto são compostos por (a, c) sendo considerados como elementos de mínimo

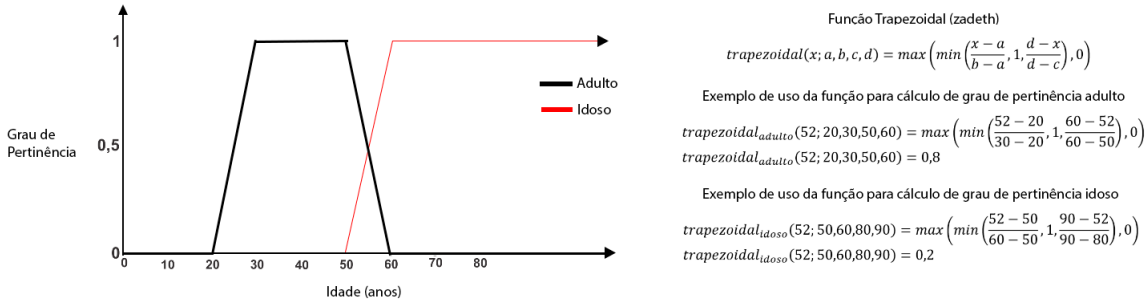

Figura 2.1. Exemplo de um Gráfico que Representa a Função Trapezoidal (do autor)

e máximo e (b) considerados como intervalo na função. O papel deles é formar os valores com grau de pertinência mais próximo de 1, mais detalhes da construção da função podem ser vistos na figura 2.2:

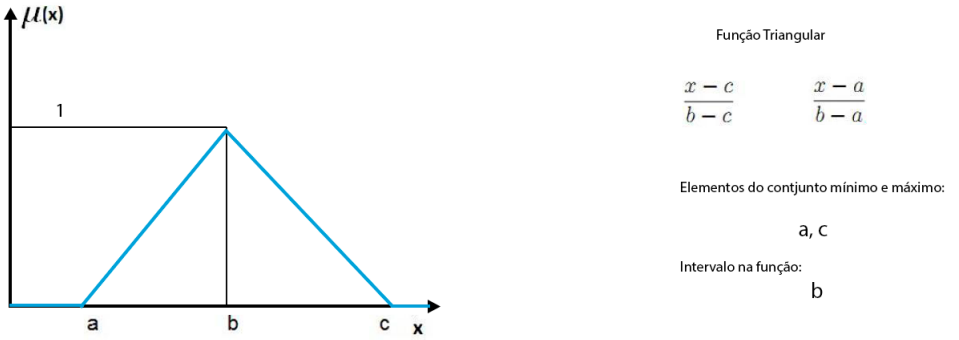

Figura 2.2. Exemplo de um Gráfico que Representa a Função Triangular (do autor)

2.4. Formas de Representar as Informações Imperfeitas

A informação precisa ser entendida, deve ser possível que o ser humano possa interpretá-la, com base nas suas experiências e conhecimento prévio.

Podemos fazer com que a informação imperfeita tenha um significado real, para isso podemos usar os conceitos apresentados na seção anterior (Lógica Nebulosa e Conjuntos Nebulosos). Deste modo assimilando a informação com algumas possíveis respostas.

É possível fazer diversas estratégias para chegar a essa assimilação, podendo usar formas que representam a informação de modo numérico ou textual, no trabalho

de (Üstünkaya et al. (2007)) podemos encontrar as seguintes definições: intervalo de valores, valores nulos, conjunto, valores que se relacionam e valores com base em variável linguística. Veremos um pouco de cada nos subtópicos abaixo:

Intervalo de Valores

A informação será representada dentro de um intervalo de valores, isto é feito para se estabelecer os limites dos possíveis conjuntos. Por exemplo, se quisermos representar a altura de uma determinada pessoa, inserimos nesse intervalo os seguintes valores [1.60 - 2.00] (de 1 metro e 60 a 2 metros).

Valores Nulos

Essa definição foi comentada um pouco na seção de classificação de informação imperfeita do tipo vagueza, ela ocorre quando há falta de informação de um determinado elemento dentro de um conjunto de elementos. O que pode ocasionar em diferentes categorias de indefinições.

Conjunto de Valores

Essa categoria de representação resumidamente se baseia em um conjunto de elementos, por exemplo, se quisermos representar uma idade, mas não temos com exatidão a mesma, podemos dizer: "a idade do fulano é 19, 24, 30 ou 39 anos", representado pelo conjunto { 19, 24, 30, 39 }.

Valores que se Relacionam

Esses valores costumam referenciar valores vindos de outros conjuntos ou subconjuntos, como se fosse uma chave estrangeira. Caso haja uma alteração na informação de determinado conjunto, todos os que referenciam essa informação também sofrem alterações.

Valores com Base em Variável Linguística

São valores que tem a função de representar um dado desconhecido, eles também são chamados dado nebuloso. Eles podem ser simples ou compostos, o simples podemos considerar termos como “alto” e “baixo”.

Já os compostos podem trazer um adjetivo a mais ao termo, como “muito alto” e “mais ou menos baixo”. Assim expandindo as possibilidades de assimilação, é possível inclusive acrescentar um termo negativo ou termo lógico como “ou” e “não”.

Uso de Limiares na Informação Imperfeita

Limiares é o plural de limiar, que de acordo com o dicionário significa: ponto que constitui um limite. Basicamente o limiar vai nos ajudar a atribuir um valor que tem o papel de limitar a quantidade de dados que vão surgir ao utilizar a informação imperfeita na consulta.

Isso é importante ser estabelecido, pois sem esse elemento em nosso projeto toda a consulta vai retornar dados que não fazem muito sentido ao usuário, mesmo se encaixando no critério de busca. Isso acontece pelo fato de que mesmo fazendo parte dos resultados, o valor de amostra é tão pequeno que faz esse dado ser irrelevante.

É nessa parte que seria feita aplicação dos limiares, seria estabelecido um valor limiar que vai lidar com qualquer uma das categorias de informação imperfeita mencionadas acima, após isso somente viria nos resultados da consulta os dados que realmente atendam aos requisitos.

As consultas vão permitir inclusive que o usuário faça o uso de limiares que ele mesmo definir, assim não ficará preso a muitas regras estabelecidas, dando liberdade para achar os limites que mais fazem sentido a ele no momento da busca.

2.5. Considerações Finais

Foi pontuado nesse capítulo a definição de informação imperfeita com 5 tipos definidos, também foi explicado quando um desses tipos podem ocorrer dentro de um banco de dados de documentos. Por fim explicamos a lógica nebulosa e como ela pode

ser usada na modelagem de sistema.

Nessa dissertação vamos trabalhar com duas categorias de informação imperfeita, a categoria de informação imperfeita chamada imprecisão e a chamada vagueza. Trabalhar com às cinco categorias estenderia muito esse trabalho atual, então optamos por utilizar as categorias restantes em trabalhos futuros.

O próximo capítulo vai pontuar o conceito de banco de dados de documentos e também vai apresentar dois modelos de documentos bem usados atualmente.

Capítulo 3

Modelos e Banco de Dados de Documentos

Neste capítulo apresentamos dois modelos de documentos: os documentos XML e os documentos JSON. Após a definição de ambos, colocamos um comparativo entre eles apontando suas diferenças mais comuns e na sequência, três tipos de bancos de dados são apresentados: MongoDB, CouchDB, EXist-db e Db4o.

3.1. Documento XML (eXtensible Markup Language)

O XML (da sigla *Extensible Markup Language*) é um tipo de linguagem de marcação que contém diversas regras para a codificação de documentos, tendo sido projetada para armazenar e transportar dados que podem ser classificados como semiestruturados ou não estruturados (Elmasri et al. (2005)).

A linguagem surgiu por volta de 1996, quando um grupo de trabalho SGML (Standard Generalized Markup Language) do W3C propôs um novo padrão, denominado XML, sendo um subconjunto do SGML (Deutsch et al. (1998)), para oferecer muitos dos benefícios do SGML não disponíveis em HTML (Hypertext Markup Language) e fornecê-los em uma linguagem que seja mais fácil de aprender e usar do que o SGML completo.

Os benefícios incluem extensão arbitrária das tags e atributos de um documento, ou seja, que não seguem princípios lógicos nem dependem de regras e normas. Su-

porte para documentos com estrutura complexa e validação da estrutura do documento em relação a uma gramática de estrutura de documento opcional, chamada Definição de Tipo de Documento (DTD).

A figura 3.1 mostra a estrutura de um documento XML, a linguagem XML não possui tags predefinidas, sendo de responsabilidade do autor do documento defini-las junto da estrutura. O XML pode ser considerado um modelo padrão para o intercâmbio de dados na internet, dentro dele é também usado a estrutura de árvores hierárquicas.

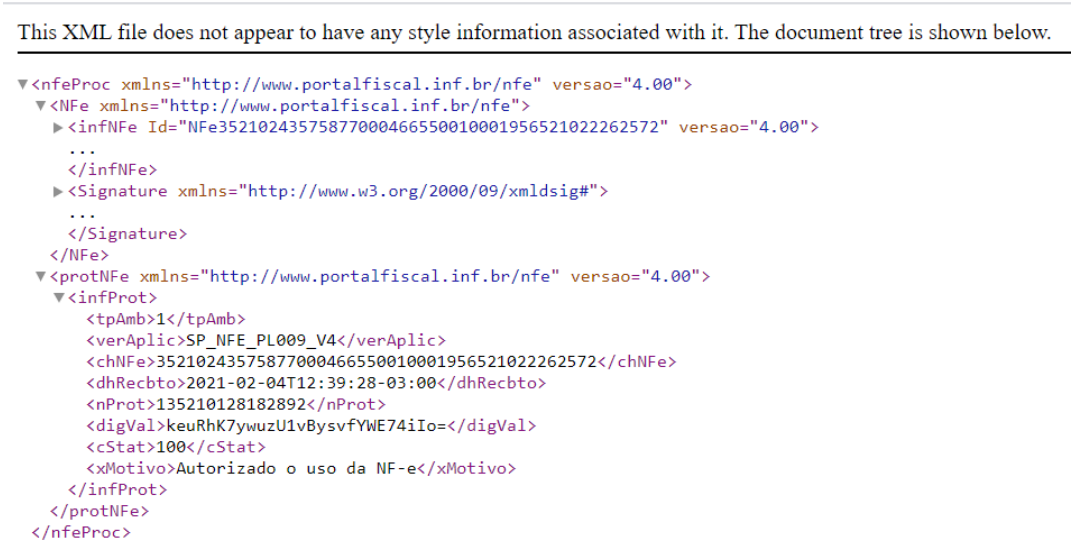

Figura 3.1. Exemplo de Documento XML (do autor)

Os dados contidos nele podem ser representados como elementos que podem ser alinhados com o intuito de criar estruturas hierárquicas de grande complexidade. A especificação de sua formatação pode ser feita separadamente utilizando, por exemplo, o XSL (eXtensible Stylesheet Language), que traduzindo fica Linguagem estendida de folhas de estilo.

A informação armazenada no documento XML pode ser classificada como dados semiestruturados e não estruturados. Dados semiestruturados são dados que podem ter atributos novos inseridos em seus registros mais recentes a qualquer momento sem que dependa de um esquema predefinido. Dados não estruturados são aqueles que não possuem um modelo definido ou estrutura, sendo, portanto, informações que aparecem mais como um grande bloco de texto, datas ou números.

3.1.1. Banco de Dados de Documentos XML

A estrutura do documento XML é composta por dois itens: elementos e atributos (Elmasri et al. (2005)). É importante frisar que esses itens se diferem dos itens que vemos nos tradicionais bancos de dados e se assemelham mais com o que vemos em um documento HTML.

Os elementos são representados em um documento XML com tag de início e tag de fim, sendo a tag de início representado por "$\langle ... \rangle$" e a tag de fim por "$\langle / \rangle$", os nomes ficam contidos no sinal de menor e maior.

Esse tipo de estrutura traz o benefício da simplicidade, pois os elementos serão organizados em sequência. A única restrição é que cada elemento que for aberto pela marcação "$\langle \rangle$" (exemplo "$\langle cidade \rangle$") deve ser devidamente fechado pela marcação "$\langle / \rangle$" (exemplo "$\langle / cidade \rangle$").

Um documento XML pode ser considerado bem formatado caso ele consiga respeitar algumas condições, sendo que essas condições são especificadas em um arquivo XML separado chamado DTD (Definição de Tipo de Documento) (Elmasri et al. (2005)). O DTD especifica os nomes das tags e o alinhamento de sua estrutura, para o documento XML ser válido ele precisa estar no padrão que o DTD especificou.

A comunidade de banco de dados menciona que os dados precisam ser gerenciados através de uma arquitetura com três níveis, cujo objetivo é separar o modelo conceito do interno e, consequentemente, de um conjunto de modelos externos de dados. A exigência para que esse modelo seja aplicado em um banco de dados de documento XML é que o modelo conceitual deve ter todos os componentes do documento disponibilizado para que qualquer aplicativo XML possa interagir com ele.

Com isso, pode ocorrer de um banco de dados de documento XML incluir modelos externos que enxergarão o banco de dados como apenas um recurso de documento para uso corporativo, por exemplo, ou qualquer combinação de uso que são necessárias várias classes de aplicativo.

No trabalho de (Salminen & Tompa (2001)), é comentado que muitos documentos XML são tratados como representações temporárias com o intuito de troca de dados em

diferentes tipos de aplicativos. Porém, há a necessidade de existirem meios eficazes de gerenciar dados do XML em um banco de dados. São discutidos quais os requisitos para o desenvolvimento de novos modelos e sistemas adequados para o gerenciamento de banco de dados XML abordando questões como modelagem, definição e manipulação.

O XML tem a finalidade de atuar especialmente na internet, sendo que as referências ao XML também referenciam aos recursos de internet. Por isso, os sistemas que gerenciam banco de dados de documento XML devem também incluir o gerenciamento de recursos de internet. Também é importante a integração de um gerenciamento de documentos estruturados com a possibilidade de gerir outros tipos de documentos.

Existem duas maneiras distintas de se realizar a consulta em um documento XML. Uma delas é utilizando o padrão XPath (XML Path Language) (*XML path language (xpath) 3.1* (2017)), que utiliza construtores de linguagem para especificar expressões de caminho e identificar os elementos no documento que corresponde a um padrão. A outra é o padrao XQuery (XML Query Language) (*XQuery 3.1: An XML query language* (2017)), sendo uma linguagem de consulta que usa expressões do XPath, porém possuiu construtores adicionais. A forma mais comum de consulta da XQuery é pela expressão FLWR(FOR,LET,WHERE,RETURN), que pega variáveis ligadas a elementos individuais e faz a junção com variáveis ligadas a coleção, especifica as condições atribuídas e retorna o resultado da consulta.

3.2. Documento JSON (JavaScript Object Notation)

JavaScript Object Notation (Pezoa et al. (2016)) é o formato de dados de maior popularidade para fins de envio de dados em requisições e resposta de API's em aplicativos web por meio do protocolo HTTP. Utilizada desde 2000, quando foi especificada por Douglas Crockford, ela é baseada nos tipos de dados da linguagem de programação JavaScript, que pode ser facilmente lida e compreendida tanto pelos humanos quanto por máquinas.

Mesmo sendo um formato de grande popularidade na web, conforme mencionado em (Bourhis et al. (2017)), existem poucos estudos a respeito de JSON, deixando assim um vazio a respeito do consenso sobre uma estrutura correta para lidar com ele.

O documento JSON possui um formato estilo chave-valor (Peng et al. (2011)), documentos com esse formato armazenam dados como um conjunto de pares de chave-valor onde uma chave funciona como um identificador exclusivo. A chave e os valores podem ser qualquer coisa, desde objetos simples até objetos compostos complexos.

Porém, a estrutura do documento JSON sempre deve ser preservada. Em outras palavras, em cada nó da árvore contida no documento, os filhos correspondentes a esse nó devem ter a sua representação alinhados a ele. O documento JSON também oferece suporte a matrizes e tipos de dados atômicos (ex: inteiros e strings). Esse formato também é muito utilizado em sistemas de bancos de dados de documentos ou banco de dados de grafos.

A especificação JSON completa define 7 tipos de valores: objetos, arrays, strings, números e os valores true / false. Para a extração de informações de um documento JSON, a maioria dos sistemas utiliza as instruções de navegação da linguagem, variando de sistema para sistema, porém sempre seguindo princípios iguais.

Inicialmente, essa estatégia não é muito poderosa, mas o que a torna interessante é que os algoritmos e implementações têm um tamanho muito leve, sendo algo que combina com o tamanho dos formatos de data que há no JSON.

Se j for um objeto JSON, ele consegue acessar o valor JSON em um par de chave-valor específico deste objeto. Caso j for um array, ele vai conseguir acessar o i-ésimo elemento de j. Por exemplo, se J é um objeto simples (ex: ”primeiro”: ”João”, ”último”: ”Doe”) poderíamos utilizar a instrução J[primeiro] para se obter o valor do par ”primeiro”: ”João”, que trará o nome João. Também é possível utilizar a instrução J [último] para se obter o valor do último que é Doe, mas não há uma instrução que possa recuperar os índices dessas instruções no caso primeiro e último.

O mesmo vale para as matrizes onde o acesso é essencialmente aleatório. Em outras palavras, pode ser acessado o i-ésimo termo como também existem maneiras primitivas de se acessar o primeiro e o último elemento de matrizes, a figura 3.2 mostra a estrutura básica de um documento JSON:

Figura 3.2. Exemplo de Documento JSON (do autor)

3.3. Definição de um Documento JSON

O documento JSON é muito usado em BDD que usam o paradigma NoSQL, ou BDD de grafos. Pode se obter do documento JSON uma estrutura como uma árvore de dados com uma ordenação semelhante ao XML. Isso é, cada chave aparece uma vez no máximo dentro de um dicionário.

Contudo, no momento da consulta de um documento JSON podem ocorrer várias implicações, uma delas seria a restrição de chaves. Por outro lado, dependendo da situação, lidamos com uma estrutura mais simples do que o XML.

No trabalho de (Lv et al. (2018)) é mostrado a definição de um documento JSON. O valor contido em um documento JSON pode ser um valor do tipo string, inteiro, numerico, booleano, nulo ou um valor do tipo matriz. Na estrutura de composição dos dados JSON, cada valor pode ser recursivamente declarado em um novo conjunto de objetos JSON no mesmo documento.

Em primeiro lugar, vamos definir melhor esses valores que o documento JSON pode conter:

1) String = Uma sequência de caracteres, geralmente utilizada para representar palavras, frases ou textos.

2) Inteiro = São números inteiros sendo eles positivos ou negativos. Não suportando números decimais.

3) Numérico = São números que suportam casas decimais entre outros sendo eles positivos e negativos.

4) Booleano = É um tipo de dado lógico que pode ter apenas um de dois valores possíveis: verdadeiro ou falso.

5) Nulo = Um valor nulo pode indicar que a informação existe mas é desconhecida, ou seja representa uma variável sem valor.

6) Matriz = Uma matriz é uma coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente na memória. A individualização de cada variável de um vetor é feita através do uso de índices.

Em segundo lugar, damos a definição de um documento JOSN da seguinte forma:

1.Documento = { Objeto [,Objeto, . . .] }
2.Objeto = Chave: { Valor } [, Chave: { Valor }, . . .]
3.Chave = String
4.Valor = String / Inteiro / Número / Matriz / Nulo / Objeto.

No item (1) mostra o documento como é montado ele pode ser composto por um objeto ou uma matriz de objetos. Em (2) descreve os possíveis valores no objeto, podendo ser um atributo de chave simples ou uma matriz de chaves. É dito em (3) que a chave só pode ser do tipo string, e por fim em (4) mostra que o valor na chave pode ser de diversos tipos incluindo um novo objeto, assim repetindo a sequência de (2), (3) e (4).

Com a estrutura definida, podemos montar um documento JSON como mostra a figura 3.3

Figura 3.3. Exemplo de um documento JSON (do autor)

Note que na figura 3.3 utilizamos alguns dos valores definidos anteriormente, no campo nome e email é utilizado valores do tipo string, no campo idade é usado um valor do tipo inteiro e em habilidades é utilizando um valor do tipo matriz que contem um valor do tipo string e um valor do tipo inteiro. Porém, o documento não se restringe somente a esse tipo de estrutura, podendo se expandir para diferentes estruturas, conforme a necessidade do usuário.

3.3.1. Banco de Dados de Documentos JSON

(Bourhis et al. (2017)) explica que, como as instruções para navegação dentro do JSON são básicas para ser uma linguagem de consulta completa, a maioria dos sistemas desenvolveu diversas maneiras para efetuarem uma consulta em documentos JSON e, como não existe um padrão nem diretriz geral, não é possível comparar as operações realizadas de um sistema para o outro.

Existem diversas linguagens de consulta que tiveram inspiração da expressão FLWR ou expressões relacionais, que propõe construir uma linguagem de consulta que pode unir, mesclar ou até gerar um novo documento JSON. Elas, em sua maioria, têm base na XQuery e possuem muitos recursos. Porém, ainda não foi realizado nenhum estudo para que se possar estruturar formalmente esses recursos.

O formato do JSON é muito usado em sistemas de banco de dados que usam o paradigma NoSQL, ou banco de dados de grafos. Como já dito anteriormente, pode se obter do documento JSON uma estrutura como uma árvore de dados com uma ordenação semelhante ao XML. Porém, a árvore obtida do documento JSON são determinísticas. Isso é, cada chave aparece uma vez no máximo dentro de um dicionário.

Contudo, no momento da consulta de um documento JSON podem ocorrer várias implicações, uma delas seria a restrição de chaves e isso pode ocasionar em uma análise mais complicada, mesmo para consultas mais simples. Por outro lado, dependendo da situação, lidamos com linguagens mais simples do que o XML.

Neste trabalho, foi mostrado que se usou uma variedade de sistemas que lidam com JSON, desde linguagens de programação como Python até sistemas de gerenciamento de banco de dados de documento JSON como MongoDB, com foco principal de se navegar pela estrutura da árvore JSON.

3.4. Diferenças Entre o Documento XML e o Documento JSON

Os formatos de transmissão de dados evoluíram a um ponto onde possuem até sua própria marcação, para oferecer um suporte adicional de exibição dos dado em sua forma estrutural.

Embora muito similares, existem algumas diferenças entre XML e JSON, (Bourhis et al. (2017)):

1. JSON mistura dados ordenados e não ordenados.
2. XML tem uma estrutura orientada a marcação (tags).
3. Vetores de JSON não são listas, mas podem ser considerados conjuntos.
4. O conteúdo do XML não precisa ser delimitado com aspas.
5. O XML possui um arquivo chamado DTD para validar a sua estrutura
6. Árvores de JSON são determinísticas.
7. O documento XML suporta dados de imagens e gráficos.
8. A notação de estrutura do documento JSON é mais simples.
9. A codificação do JSON é UTF-8.
10. O XML possui diversas codificações.

XML e JSON tem o mesmo papel de transmitir dados entre aplicações, esses dois formatos ficaram com mais popularidade devido à flexibilidade e agilidade de transportar os dados. (Nurseitov et al. (2009)).

O que nos fez optar pela escolha do formato de documento ser o JSON foi que além das diferenças citadas acima, ele é mais compatível ainda com o Javascript, que é a linguagem que vamos usar para fazer a arquitetura intermediária.

A normalização dos dados do JSON será mais fácil de se realizar do que em XML e caso seja necessário utilizar o XML basta usar o Javascrit para converter o documento JSON em um documento XML, sem grandes complicações.

Esses formatos foram bem aceitos pela comunidade, pelo modo de como pode se atribuir e controlar os dados recebidos. A diferença principal entre os dois é que o XML conta com um DTD, e este define a estrutura padrão que o XML vai aceitar de dados.

Já o documento JSON não possui uma estrutura específica, e isso tanto pode ser bom quanto não, pois, enquanto não ficamos presos a um padrão quando tivermos que restringir alguns dados não serão de uma forma tão automatizada, já que precisaremos criar nossa própria estrutura JSON e declarar isso em várias partes da aplicação.

De todo o caso esse é um dos desafios desta dissertação, implantar um documento JSON com uma estrutura difusa para se poder utilizar a consulta com base na informação imperfeita em diversos sistemas de banco de dados de documentos.

3.5. Banco de Dados de Documentos

Os SBD são modelos de banco de dados não relacional que armazenam e consultam dados em formato de documentos (Nayak et al. (2013)), hoje há dois formatos de documentos populares na comunidade, um deles é o documento XML e outro é o documento JSON. Esses formatos buscam facilitar o armazenamento e consulta para quem os utiliza, pois esses dois modelos usam o mesmo formato de código que é usado no código da aplicação, e também possuem uma estrutura flexível, permitindo que o modelo evolua com as necessidades da aplicação que o utiliza.

Em geral, o conceito por trás dos bancos de dados de documentos aponta em sua definição que eles utilizam o formato de dados e documentos auto-contidos e auto-

descritivos, assim já definindo sua apresentação e o significado por trás de sua estrutura de armazenamento. Por ter essa característica de poder possuir todas as informações importantes em um único documento, ele acaba também sendo livre de esquemas visto que não é necessário atribuir toda estrutura do banco inicialmente como é feito no modelo relacional.

O termo correto para se nomear esses modelos de bancos seria NoSQL (Not Only SQL), esse termo é usado quando o modelo de banco não possui os conceitos e métodos da linguagem SQL padrão e também por ser uma das características de um modelo não relacional. Modelos não relacionais são modelos que não possuem relação entre as tabelas/coleções de dados, geralmente o relacionamento é feito de forma indireta no próprio documento ou em documentos separados, mas sem especificar a relação apenas os ligando de forma indireta.

Os documentos no banco de dados possuem como endereço uma chave única, esta serve para identificar o documento no registro e nas demais operações do banco de dados. Essa chave pode ser constituída de uma string simples ou uma string que possa ser uma URI ou um endereço, o ideal é que esses bancos sejam usados para aplicações que os dados não tenham necessidade de serem armazenados em tabelas com campos de tamanho uniforme. Assim dando lugar ao armazenamento através de um documento com as devidas propriedades.

Dentro dos bancos de dados de documentos os registros são armanezados em coleções, que seriam como as tabelas no banco de dados de modelo relacional, e dentro dessas coleções é aonde ficam armazenados os documentos, a figura 3.4 apresenta um exemplo resumindo um pouco este conceito.

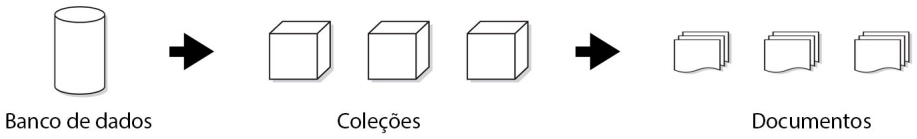

Figura 3.4. Estrutura Simples de um Banco de Dados de Documentos (do autor)

Esse tipo de armazenamento apresenta um bom funcionamento quando o modelo pode ser dividido e particionado em alguns documentos, porém o desenvolvedor que utilizar esse modelo de banco de dados precisa ficar atento ao fato do banco de dados possuir várias relações e normalizações. Caso isso ocorra o armazenamento deve ser evitado, podemos citar nesse trabalho três exemplos de sistemas de banco de dados de documentos: MongoDB, CouchDB, EXist-db e Db4o.

3.5.1. Banco de Dados de Documentos MongoDB

Desenvolvido pela 10gen e lançado por volta de 2009 o MongoDB é um banco desenvolvido na linguagem C++, possui grande eficiência e alto desempenho além de possuir tolerância a falhas de consistência e persistência dos dados. Existem outros recursos oferecidos pelo MongoBD como, agregação de valores, consultas ad hoc, possibilidade de indexação e fragmentação automática.

O formato de armazenamento de documentos é no formato BSON (Binary JSON) aonde estes contem uma lista de elementos ordenada que consistem no nome do campo, tipo do campo e valor do campo, esse formato tem uma grande eficiência de espaço quanto de velocidade de varredura ao se comparar com o JSON normal. Ele conta também com o GridFS para armazenar arquivos grandes.

Ele é recomendado para se trabalhar com aplicativos de gerenciamento de conteúdo, arquivamento e análise de tempo real, a figura 3.5 mostra um comparativo entre a formatação de dados utilizada no banco Mongo DB e um banco de dados relacional.

Figura 3.5. Comparativo do Banco MongoDB Para um Banco Relacional (do autor)

3.5.2. Banco de Dados de Documento CouchDB

Desenvolvido pela apache software e lançado em 2005 o CouchDB também foi desenvolvido em C++, além de armazenar os documentos JSON ele também fornece uma RESTful API HTTP para poder manipular esses documentos no banco de dados. Ele utiliza na estrutura de consulta javascript, assim como nosso trabalho irá fazer na API, porém a diferença é que essa estrutura atua de uma forma adaptada para trabalhar somente em seu modelo.

O Banco de dados CouchDB também fornece um aplicativo na web usado na administração chamado FULTON, este trabalha no padrão MVCC (Multi-Version Concurrency Control) possibilitando assim o acesso simultâneo aos usuários. Conta também com recursos de sincronização e replicação, a recomendação é que ele seja utilizado para aplicações que façam alteração ocasional dos dados aonde deve ser usado consultas já pré-definidas. Conta também com disponibilidade offline, ou seja pode funcionar sem a internet (por conta da sincronização) algo muito útil para aplicativos em dispositivos móveis. A figura 3.6 resume como é a arquitetura do CouchDB.

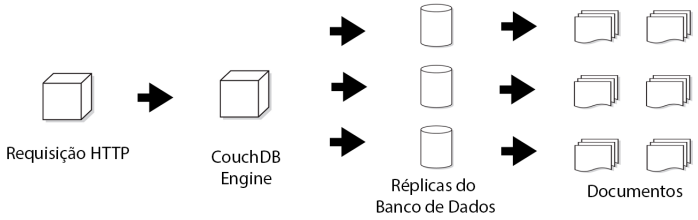

Figura 3.6. Arquitetura do CouchDB (do autor)

3.5.3. Banco de Dados de Documento Db4o

Criado por Carl Rosenberger em 2000 e lançado em 2004 o db4o é um banco de dados de documento desenvolvido em C# e JAVA, possui uma interface gráfica chamada OME (Object Manager Enterprise) que tem a finalidade de fazer a conexão com o banco, navegação, construção e funções administrativas no mesmo.

Ele fornece Native Queries (NQ) permitindo os usuários usarem linguagens de programação que possuem o paradigma de orientação a objetos com o JAVA,C# e o VB.Net, ao invés de usar uma linguagem de consulta própria. Com a possibilidade de ar-

mazenar um objeto apenas executando um único comando, podendo também sincronizar um modelo de back end relacional com o db4o.

Sua desvantagem é que ele não fornece suporte para exportar ou importar dados de um documento JSON ou documento XML ou arquivos de texto, também não conta com recursos de integridade referencial ou ferramentas OLAP, a figura 3.7 mostra a diferença do modelo de banco relacional para o db4o.

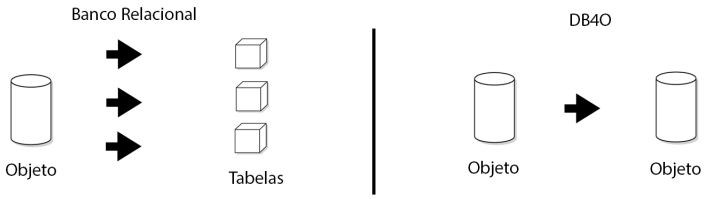

Figura 3.7. Comparativo da Estrutura entre o Banco relacional e o Banco Db4o (do autor)

3.5.4. Banco de Dados de Documento eXist-db

Exist-db é um banco de dados de documento feito a partir de um projeto de código aberto para bancos de dados NoSQL com XML. Ele é um banco de dados XML nativo, possuindo como linguagens de consulta dentro desse banco o XQuery e XSLT, foi lançado em 2000 em sua versão 2.1. A figura 3.8 mostra a interface gráfica do eXist-db rodando um script de busca em sua base.

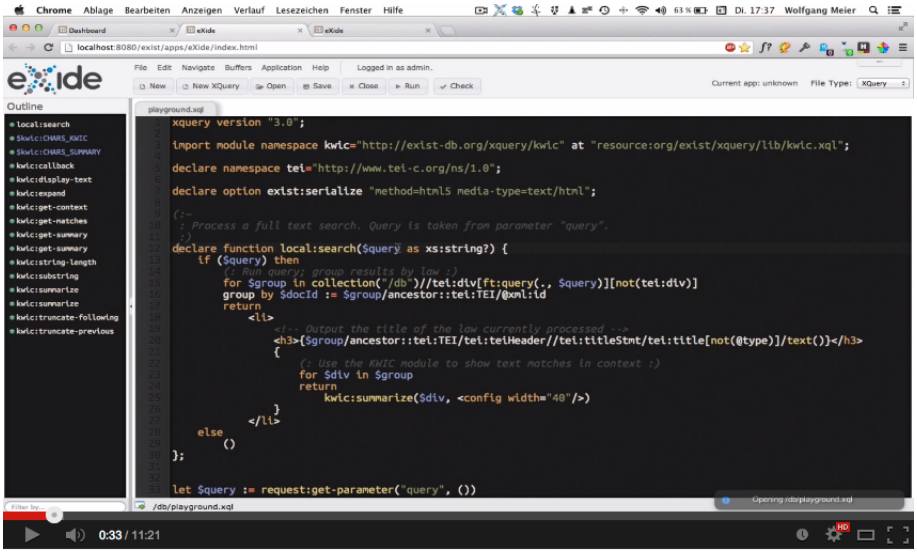

Figura 3.8. Aplicação de um Scrip Dentro do eXists-db (Exist-DB, Website (2018))

O eXist-db possibilita que desenvolvedors consigam persistir documentos XML sem escrever um tipo de serviço completo para isso, ele segue e estende muitos padrões da W3C e suporta a interface REST para interface com formulários. Ele também permite a indexação de documentos automaticamente usando um sistema de indexação de palavras-chave.

3.6. Considerações Finais

Nesse capítulo falamos um pouco dos modelos de documento XML e JSON e também pontuamos a diferença entre os dois para melhor entendimento de cada estrutura usada nesta dissertação.

Após foi apresentado o conceito de banco de dados de documentos com alguns exemplos e diferenças entre eles. O próximo capítulo vai mostrar a bibliografia pesquisada e no fim introduzir a fundamentação teórica.

Capítulo 4

Trabalhos Relacionados

Neste capítulo revisamos alguns trabalhos e separamos algumas ideias contidas em cada trabalho, com o propósito de formar a ideia principal deste trabalho baseado nelas. Após a separação foi feita a comparação das ideias de cada artigo, para a montagem do plano de ação, as ideais foram organizadas em uma tabela comparando a proposta de cada uma.

4.1. Levantamento com Mapeamento Sistemático

Mapeamento Sistemático é um método que consistem em identificar, avaliar e interpretar a relevância dos trabalhos para determinada questão ou área de interesse conforme dito em (Kitchenham (2004)).

É de suma importância a realização desse método pois garante o valor da pesquisa por meio de questões de pesquisa que vão ser levantadas como palavras chaves para a filtragem de propostas de um artigo para o outro, assim após a extração restará somente os que são compatíveis realmente com a proposta inicial desta dissertação.

Com isso em mente é preciso definir os parâmetros para direcionar o processo de extração e seleção dos trabalhos individuais que vão contribuir de forma positiva para o Mapeamento Sistemático, essa definição será realizada de forma sistemática. Vamos chamar esses trabalhos de estudos primários, de acordo com (Kitchenham (2004)) é necessário o uso de um protocolo para a seleção e análise dos estudos primários, pois assim evita que os trabalhos sejam escolhidos primeiramente por intenções tendenciosas do pes-

quisador, deixando os resultados não confiáveis.

Para o Mapeamento Sistemático realizado nesse trabalho foram usados como base artigos do ano 2000 até o ano 2020, no idioma inglês. Os artigos utilizados nesse intervalo de tempo não se tornaram obsoletos com o tempo para que não pudesse ser usados nessa pesquisa, pois um dos requisitos era verificar os métodos utilizados em cada artigo para a manipulação de informação imperfeita.

As bases utilizadas foram IEEEXplore, ResearchGate, ACM, ScienceDirect e MDPI. O propósito principal do Mapeamento Sistemático foi buscar na literatura estudos primários sobre o uso de lógica fuzzy em Bancos de Dados Orientados a Documentos. Com esse propósito também formamos a questão principal que é: " Quais são os estudos primários que abordam o uso de lógica fuzzy em Bancos de Dados Orientados a Documentos?".

A seleção dos artigos vai ter o papel de responder as seguintes questões de pesquisa:

Q1: Tem efeito positivo utilizar a lógica fuzzy em banco de dados?

Q2: Quais técnicas estão sendo utilizadas para se chegar nesse resultado?

Q3: Quais trabalhos apresentam técnicas mais expressivais que podem ser usadas como base para uma nova usada em nossa pesquisa?

É preciso agora para dar continuidade com o processo de pesquisa criar uma string de busca para sintetizar os parâmetros de pesquisa para que haja um possível retorno dos estudos primários que vão responder as questões levantadas anteriormente.

Para formar a string de busca foi considerado os seguintes campos: título, palavras-chave e resumo. Resultando nas seguintes palavras para a string de busca:

1.Fuzzy

2.Database System

A string de busca utilizada foi Fuzzy Database System, ela foi definida dessa forma para ser compatível com o objetivo dessa pesquisa, convém destacar que incluímos a seguinte string adicional "Javascript Fuzzer" nas buscas para obter alguns resultados que lidassem com Javascript para ter estudos primários que usassem tal tecnologia.

Com a junção dessas strings de busca já foi o suficiente para o retorno de estudos primários o suficiente para a confecção do Mapeamento Sistemático.

4.2. Trabalhos Retornados das Bases

Durante o mês de Abril de 2020, a string de busca foi aplicada nas bases apresentadas, retornando um total de 714 artigos que atendiam os critérios estabelecidos anteriormente. Ao pesquisar na base do IEEEXplore tivemos um retorno de 77 artigos, na base do ResearchGate 100 artigos, na base do ACM DL 291 artigos, na base do MDPI 216 artigos e por fim na base ScienceDirect 30 artigos. A Firugra 4.1 apresenta um gráfico que foi elaborado para representar melhor esses resultados:

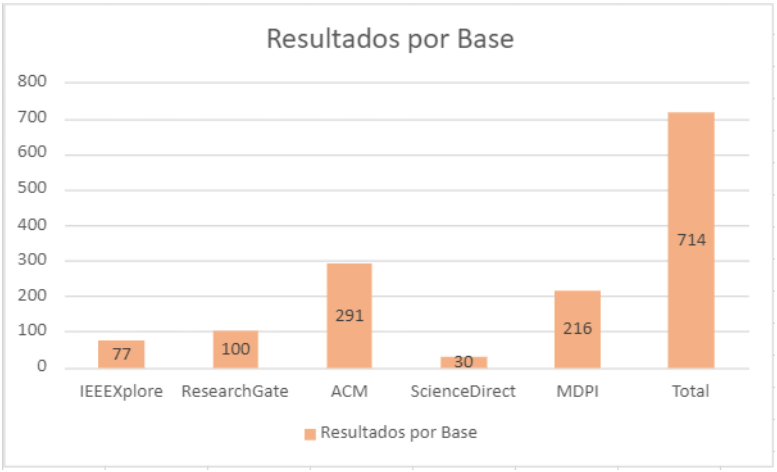

Figura 4.1. Resultado das Bases de Artigos Pesquisadas (do autor)

O próximo passo agora é dentre esses 409 artigos selecionar os mais relevantes para o objetivo principal afim de finalizar o Mapeamento Sistemático.

4.3. Processo de Inclusão e Exclusão dos Artigos

Foi necessário criar alguns critérios para a seleção dos trabalhos mais relevantes dentro dos 498 artigos obtidos na etapa anterior para dar continuidade ao Mapeamento Sistemático.

Esses critérios vão ser responsáveis por incluir os artigos com maior compatibilidade e também por excluir os que tiverem menor compatibilidade. Para serem compatíveis com os critérios de inclusão, os artigos devem cumprir os seguintes critérios:

•Abordem de alguma forma o uso de lógica fuzzy para otimizar o Banco de Dados de Documentos

•Que essa abordagem seja em Banco de Dados de Documentos XML ou JSON

Os critérios de exclusão, vão ter como finalidade justificar o motivo da remoção dos artigos no Mapeamento Sistemático por não cumprem os requisitos para se tornarem compátiveis. Estabelecemos os seguintes critérios:

•Artigos que não lidem em sua totalidade com Bancos de Dados de Documentos

•Artigos com conteúdo duplicado

•Artigos que não estão na língua inglesa

•Trabalhos não categorizados como artigos

Critérios Para Inclusão
Artigos que abordem o uso de lógica fuzzy para otimizar o Banco de Dados de Documentos
Que a abordagem seja realizada em Banco de Dados de Documentos XML ou JSON

Critérios Para Exclusão
Artigos que não propõe o uso de lógica fuzzy para otimizar o Banco de Dados de Documentos
Trabalhos que não estejam na lingua inglesa ou que não sejam categorizados como artigos

Tabela 4.1. Critérios para a inclusão e exclusão dos artigos

Em posse dos critérios de inclusão e exclusão, foi seguido para a próxima etapa que foi realizar uma análise dos artigos obtidos, seja pela leitura do título e resumo do artigo ou a leitura do artigo completo.

Utilizar como um dos métodos de análise a leitura do resumo dos artigos é válido, pois podemos de imediato fazer uma filtragem dos artigos que podem ser válidos no Mapeamento Sistemático utilizando os critérios de exclusão para escolher os artigos com maior relevância.

Após essa filtragem do resumos, foi realizada a leitura dos artigos completos que foram considerados válidos, e desses artigos extrair os que de fato atendiam aos critérios de inclusão.

Com esse processo, dos 498 artigos retornados anteriormente 20 foram escolhidos para uma análise mais aprofundada. E isso resultou na figura 4.2, que mostra os artigos excluídos por base:

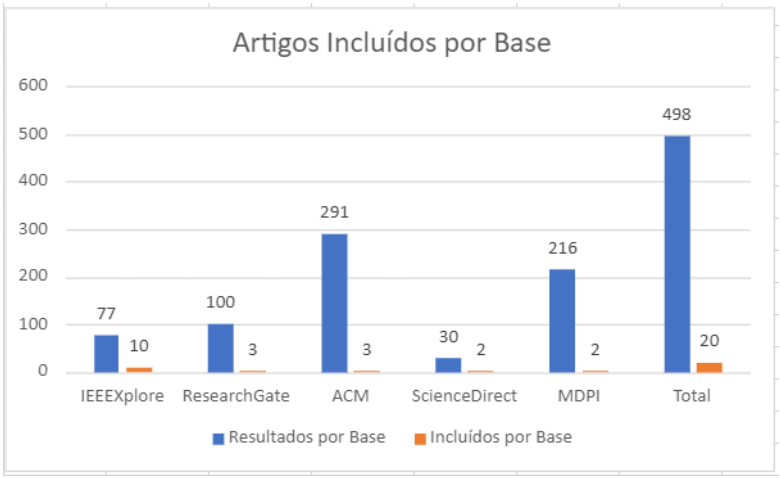

Figura 4.2. Artigos Que Foram Incluídos por Base (do autor)

Dos 20 artigos que ficaram pós filtragem, 15 atenderam o primeiro critério de inclusão imposto e 5 artigos atenderam o segundo critério de inclusão. Os 478 artigos excluídos atenderam ao primeiro critério de exclusão somente.

Depois foi realizada mais uma análisa em cima desses 20 artigos pré-selecionados, para selecionar os que mais teria relação com essa pesquisa. Dos 20 artigos 7 se mostrara mais apropriados para serem utilizados, resultando no grafico apresentado na figura 4.3

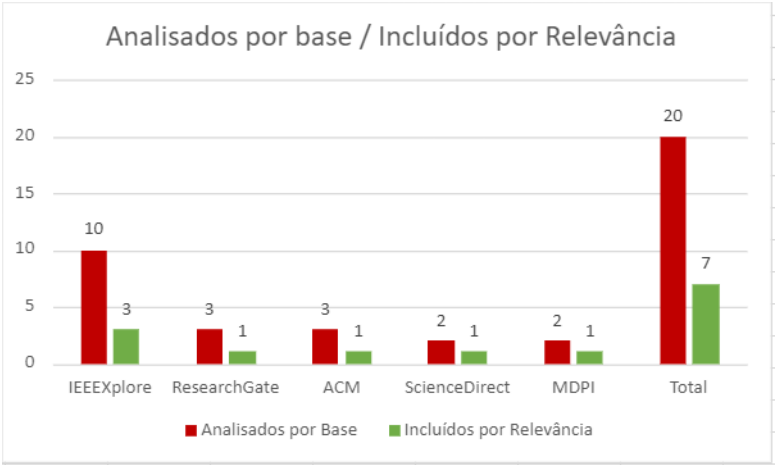

Figura 4.3. Analisados por base / Incluídos por Relevância (do autor)

4.4. Análise dos Trabalhos Incluídos por Relevância

Essa seção vai apresentar as características dos artigos escolhidos, de forma mais precisa, apontando os pontos que os fizeram ser selecionados como referência para essa pesquisa.

4.4.1. Trabalho de Psaila & Marrara

O trabalho de Psaila & Marrara (2019) tem como objetivo explorar as coleções de documentos JSON, com o intuito de flexibilizar os resultados obtidos de determinada consulta. Com a finalidade de que esses resultados sejam mais coerentes com aquilo que o usuário que está realizando a busca espera. Nos processos descritos neste artigo, são mostrados os passos de como fazer um framework fuzzy para analisar coleções de documentos JSON.

Um framework é um pacote de códigos ou instruções prontas que podem ser utilizadas no desenvolvimento de diversas aplicações. A proposta para o uso dessa ferramenta é aplicar funcionalidades, comandos e estruturas já prontas para garantir a qualidade no projeto e produtividade. Uma das motivações de seu trabalho era a necessidade de criar consultas mais flexíveis, pois as consultas atualmente seguem mais o caminho das condições booleanas clássicas (ou condições nítidas).

Dentro do nosso trabalho foi aproveitado essa ideia de flexibilidade e de como pegar termos imprecisos e inseri-los nos comandos de consulta do banco de dados de documento para gerar uma consulta válida.

Sem essa condição não seria possível realizar uma consulta com resultados diferentes ou mais maleáveis do que foi proposto no ”corpo”da consulta. Assim eliminando possíveis resultados que teriam relevância significativa para a tomada de decisão do usuário, assim prejudicando seu julgamento final.

Esses resultados, às vezes não são considerados porque ainda não se atingiu a condição necessária, por ser um dado novo na coleção ou por faltar uma pequena parte para atingir totalmente a condição solicitada. Para o usuário, às vezes isso não faz mal, pois em nossa realidade tal dado pode ser usado de maneira eficiente, algo que a máquina ainda não consegue entender de maneira aproximada.

Nós seres humanos conseguimos avaliar e processar categorias de dados incertos e ambíguos, sendo assim é ideal levar esse tipo de raciocínio para as aplicações que trabalhamos.

Dentro ainda dessa motivação, podemos colocar que os autores propõem o uso de uma consulta que utiliza mais associação do que regra booleana. Isso tem um impacto muito positivo, pois garante, em teoria, que a consulta vai retornar mais dados que utilizando uma regra booleana.

Afinal, a regra de associação utiliza como critério o quanto aquele dado é relevante para o usuário em um intervalo de 0 a 1, quanto mais próximo de 1 significa que o dado possui maior relevância e mais próximo de 0 indica que o dado não possui nenhuma relevância.

Em outras palavras, nessa categoria de consulta abordada no trabalho, o dado agora só precisa estar mais próximo de 1 para ser apresentado ao usuário, não tendo mais nenhum impedimento. Esse foi um dos conceitos que levamos para a implementação de nosso trabalho.

Outro ponto que pode ser mencionado neste trabalho foi como aproveitar melhor o grau de pertinência da informação nebulosa colocada na consulta. O grau de pertinência é um valor de associação que vai ser obtido através de uma função triangular ou trapezoidal.

Além do uso de funções matemáticas para classificar o grau de pertinência das informações nebulosas, eles utilizaram padrões de intervalos (identificados com o atributo de nome range) para determinar um intervalo dos menores aos maiores elementos mapeados para o grau de associação, de modo que os valores possam ser comparados dentro desse intervalo e, assim, cada resultado seja tratado para que, após passar pelas funções de associação, tenha seu valor aceito nos resultados de busca ou não.

O que no final nos leva a uma classificação mais flexível do que a booleana, por conta de ter uma aceitação bem mais abrangente e pode garantir uma maior cobertura de informação relevante ao usuário.

Nossa proposta de consulta deve também ser flexível, queremos contribuir em diversas áreas que necessitam extrair dados sem muitas com maior cobertura, por conta

dos critérios de busca existentes, para uma tomada de decisão mais precisa.

Um exemplo disso é quando temos vendedores de uma loja e queremos saber se algum vendedor já bateu a meta de vendas de 10 dias, porém estando no 4.º dia do período. Podem existir vendedores que já cumpriram a meta, porém haverá vendedores que ainda não, devido aos 6 dias retantes ou os outros vendedores estão realizando outra estratégia para que até o 10.º dia a meta seja cumprida.

Se utilizássemos a lógica booleana, a mais comum nos bancos de dados atuais, essa situação dificilmente seria entendida, visto que nessa lógica as condições são diretas e restritas ao contexto de ”sim”e ”não”ou ”verdadeiro”e ”falso”.

Além desse exemplo comentado acima, outros exemplos serão abordados dentro deste trabalho, de modo a apresentar melhor a aplicação final que tem como objetivo ser usada na maioria dos bancos de dados de documento. Desde que se respeitem alguns padrões de compatibilidade para que todos tenham a mesma experiência ao efetuar as consultas no banco, esses padrões serão pontuados ao longo dos capítulos para melhor entendimento.

4.4.2. Trabalho de Guo

No trabalho de Guo (2017) é explorado o banco de dados de documento chamado MongoBD, que trabalha com coleções de documentos no formato JSON. Foi proposto misturar nos comandos de busca do MongoDB comandos da linguagem JavaScript, para a realização de testes randômicos no banco de dados de modo a se caçar falhas na plataforma.

Isso é bastante interessante, pois mostra como é possível trabalhar bem com mongoBD e JavaScript juntos para se obter uma aplicação que conta com recursos combinados de ambos (sendo uma das estratégias de nossa pesquisa). O trabalho explora muito o conceito de árvores de sintaxe para definir onde seria colocado no fluxo de execução os comandos JavaScript para gerar os casos de testes.

Esses testes contam com tratamento de erros, evitando assim que a aplicação pare ao encontrar uma falha muito específica, podendo continuar com a execução dos testes.

Foi comentado neste trabalho que durante um teste, utilizando uma técnica para

testar o software usando ferramentas automáticas, foi descoberto que o MongoDB não manipulava corretamente strings de expressões regulares contendo bytes nulos. Esse caso de teste causou uma falha no servidor, sendo que essa informação é muito valiosa para nosso trabalho por nos informar os perigos que ocorrem ao não se tratar devidamente as strings que possam vir juntas dos resultados de uma consulta no MongoDB.

Essa falha dentro da nossa aplicação pode ocasionar um erro tanto para quem efetua a consulta quanto no sistema inteiro, porque geralmente vamos tratar a informação imperfeita como se fosse um dado do tipo string. É necessário aplicar regras de tratamento de erros para as exceções que virem para garantir o maior controle possível.

Em outra parte do trabalho, foi comentado o uso de um conjunto de instruções try/catch, essas instruções são responsáveis por tratar os "erros comuns" que ocorrem a fim de evitar que falsos positivos travem a aplicação deixando que somente ocorram em casos de erros mais críticos.

Assim, não perdemos tempo com questões triviais e deixamos a aplicação com o mínimo de erros possíveis até o lançamento do primeiro protótipo. Entretanto, o trabalho usado em referência possui algumas limitações.

Uma delas é que quando um teste possui um estado configurado por operações anteriores, que não está relacionado com outros estados, é necessário examinar e compreender o ocorrido para identificar a falha , levando a ocorrer intervenção humana durante os testes.

No final do trabalho, o autor conseguiu aumentar a cobertura de detecção de erros significativamente com pouco esforço, considerando que ele busca os testes de casos extremos e que ainda não foram tradados.

Deixando de lado os erros mais comuns encontrados no MongoBD, visto que o restante dos erros durante o projeto estava sendo devidamente tratados, levaremos para o nosso trabalho alguns pontos comentados aqui.

Como a junção de comandos do Javascript e o MongoDB, as estratégias de tratamentos de erros sendo eles triviais ou medianos para evitar travamentos e erros fatais na aplicação e entre outros.Para fazer sentido todo o esforço gasto e a contribuição desse

projeto, pegaremos o melhor de cada tecnologia com o diferencial de cada uma, assim resolvendo de forma diferente um problema existente.

Lembrando que não será somente ao MongoDB que a aplicação vai ser adaptável, mas sim aos bancos de dados de documentos que atenderem aos requisitos necessários para poderem se conectar e realizar as consultas de maneira satisfatória sem perdas de dados.

4.4.3. Trabalho de Jin & Veerappan

Em Jin & Veerappan (2010), os autores falam como eles conseguiram incorporar a lógica difusa em um sistema de banco de dados XML e também como conseguiram implementar uma linguagem de consulta difusa baseada no XQuery, permitindo que se use expressões difusas em qualquer condição de consulta realizanda no banco de dados XML.

Um processador de consulta também foi construído para permitir o processamento e execução de uma consulta difusa sobre dados XML difusos e nítidos. Para ilustrar o funcionamento desses dois recursos mencionados, os autores utilizaram um aplicativo de livraria que inclui os dados nítidos e difusos.

O trabalho inicialmente fala da capacidade do ser humano de processar todas as categorias de dados, e também das decisões que eles manipulam com base no raciocínio com termos vagos. Posteriormente, o trabalho apresenta o conceito da função de pertinência que não trabalha definindo se a informação é verdadeira ou falsa, mas sim lhe atribuindo um grau de pertinência.

Por exemplo, pegamos a frase: “A temperatura está quente”, a temperatura pode ser uma variável difusa ou variável linguística, sendo assim é necessário que o grau de pertinência atribua a essa variável um valor numérico com base em uma função de associação.

Essa função pode ser uma função trapezoidal, que vai utilizar um valor limite para especificar o grau de associação do valor da variável entre 0 e 1, onde mais próximo de 0 quer dizer que a variável é menos compatível com o valor especificado e que quanto mais próximo de 1, mais compatível ela é com esse valor.

Os autores utilizaram esse conceito primeiramente criando um atributo no registro de livros que continha a popularidade dos livros, porém esse atributo necessita que seja definido como um valor fuzzy mesmo que dentro dele tenha um valor nítido (para que a lógica de consulta possa ser aplicada).

Isso acontece porque se o valor de popularidade do livro for definido, esse valor pode ser substituído normalmente sem afetar nenhuma parte do sistema que já foi construído, assim facilitando outras futuras implementações.

A função do processador de consultas neste trabalho é avaliar e executar essas consultas. Ele identifica a variável, faz a varredura para compreender o operador de comparação utilizado na expressão e identifica o tipo de distribuição. Se o atributo é definido como fuzzy, ele pode ser consultado utilizando qualquer um dos três tipos de distribuição (trapezoidal, triângular ou intervalo).

Esse sistema foi implementado em Java, o que nos deixa com uma alternativa a mais sobre qual tecnologia usar para implementação do protótipo nos próximos passos deste trabalho. No final do projeto, eles conseguiram implementar o aplicativo da livraria que, por padrão, consegue armazenar e consultar dados difusos com funcionalidades nativas do sistema.

Esse trabalho nos deu como base estratégias para a criação de campos difusos em nosso esquema de documento, gerando um documento nebuloso (que vai ser abordado em um tópico específico mais adiante). E também de como trabalhar com o grau de pertinência, para que possamos no nosso protótipo, adotar uma estratégia similar, mas de uma forma totalmente diferente que consiga explorar outras possibilidades do uso de limites e intervalos.

4.4.4. Trabalho de Ueng & Skrbic

No trabalho de Ueng & Škrbić (2012) eles implementam uma extensão da XQuery, sendo a linguagem padrão para consultar um documento XML, para ela poder lidar com dados incertos e imprecisos.

Foi proposto inserir a lógica fuzzy nos conceitos da linguagem XQuery com um interpretador para entender a extensão aplicada. Com isso, as consultas que são estendidas

são convertidas para consultas do modelo padrão de XQuery para serem executadas no banco nativo.

O Trabalho aborda o problema de satisfação de restrição difusa de prioridade generalizada (GPFCSP), sendo um problema definido com um conjunto de objetos que seu estado deve satisfazer algumas restrições e limitações.

Esse problema pode ser estendido com a lógica fuzzy também introduzindo a noção de prioridade, deixando assim que o valor com maior prioridade tenha maior impacto no resultado. Desse modo, a estrutura de construção desse problema pode ser semelhante à cláusula XQuery que os autores procuraram estender, utilizando cláusulas específicas e palavras-chave que podem ser usadas como variáveis GPFCSP com domínios associados.

O XQuery é estruturado na instrução FLWOR ("For, Let, Where, Order By, and Return"), sendo que o For é um loop que permite percorrer vários elementos da coleção de um elemento para o outro; o Let permite criar uma variável e atribuir um valor a ela; o Where filtra os itens a serem usados na coleção; o Order By classifica a coleção resultante do item enviado de volta pelo Return; o Return mostra os itens de dados específicos que devem retornar ao loop For.

Aqui, é mostrado também dois tipos de banco de dados XML o XED (XML Enabled Database) e o NXD (Native XML Database). O XED é um banco de dados relacional que transfere dados entre documentos XML e tabelas relacionais. Ele recupera dados com o intuito de manter as propriedades de relacionamento em tabelas, em vez de modelar os documentos XML. Já o NXD armazena os dados XML diretamente e os acessa sem realizar nenhuma conversão, fornecendo um bom desempenho.

É explicado ainda mais sobre a proposta de armazenagem de valores fuzzy no documento XML, que pode ser usado na XQuery estendida para reconhecer a lógica difusa. Também é comentado sobre a variável linguística com o tipo padrão de função de associação (seja triangular, trapezoidal ou ombro) esses termos ficaram em tags " $<>$ " conforme o padrão do documento XML, porém as variáveis difusas devem ser definidas pelos usuários dentro de uma GUI (Graphical User Interface).

Essa extensão da XQuery teve uma modificação comentada no comando WHERE, tendo sido introduzido o conceito de prioridade e limite. A prioridade tem o efeito de definir o nível de influência de uma expressão e o limite, que tem a função de eliminar os resultados que estão abaixo do limite especificado.

Tanto prioridade quanto limite estão no intervalo de [0,1]. Se não houver um valor de prioridade definido para o termo, ele vai assumir o valor 1, mas caso não seja especificado um limite o valor dele é 0. Esses valores no intervalo de [0,1] são obtidos através das funções de associação apresentadas.

No fim do trabalho, é falado o estado atual do interpretador XQuery fuzzy, que ainda está em fase de implementação. Para isso, foi utilizada a linguagem de programação Java e a ferramenta compiladora ANTLR (Another Tool for Language Recognition).

Foi utilizado como back-end o eXist, por ser um banco de dados XML nativo de código aberto que também é escrito em Java com suporte ao XQuery padrão. O eXist usa o Jetty como servidor web e fornece uma interface de módulo plugável, que permite o desenvolvimento fácil de extensões para a ferramenta com o acesso total ao banco de dados.

Foram desenvolvidos três componentes principais: SyntaxValidation, Transform e CalculateMembershipFn. O SyntaxValidation é usado quando o usuário insere a consulta. Ele verifica e valida a sintaxe do comando segundo a gramática EBNF. O Transform é utilizado para transformar a XQuery fuzzy em consulta XQuery.

Ele fará a verificação do elemento fuzzy na consulta, eliminara a restrição fuzy e por fim removera a expressão fuzzy da consulta. O CalculateMembershipFn será responsável pelo cálculo da função de associação, ele também é responsável por fazer os elementos fuzzy não serem mostrados no resultado da consulta.

4.4.5. Trabalho de Shakhovska

O trabalho de Shakhovska (2017) se baseia inicialmente na exploração da definição do Big Data e a descrição de suas principais características, além de explorar a linguagem XML com uma arquitetura e padrões modernos.

Um modelo de associação entre entidades e características é construído durante

esse trabalho com um método de compartilhamento de dados heterogêneos, os trazendo para o modelo de dados relacional.

No início do trabalho é comentado sobre o grande aumento do número total de dados globais coletados: em 2005, eram 130 Exabytes; em 2011, esse número aumentou para 1227 EB; nos anos seguintes, subiu para 3 ZB (Zettabyte) com a pesquisa estipulando que em 2015 o volume teria um crescimento para 7,9 ZB (Zettabyte).

E finaliza a contagem apontando que o tamanho dos bancos de dados individuais está chegando no PB (Petabyte) e que a maioria desses dados coletados não são analisados, ou se são, apenas superficialmente.

A autora descreve que os principais problemas que surgem no processamento de dados é a falta de métodos analíticos adequados devido à boa escalabilidade.

O modelo proposto no trabalho tem suporte à associação difusa (que da possibilidade de usar variáveis e funções difusas). Para isso acontecer, foi proposto o uso de uma arquitetura moderna denominada XML MVVM ("Visão- Model","Model-View").

A diferença desse modelo para o MVC (Model-View-Controller) é a falta de requisitos de vinculação de dados para sua apresentação.

O modelo "valor de entidade" possui 5 componentes: (1) Fuzzy L-markup Fuzzy queries, (2) Mining of XML, XSD, DTD, XQEURY (3) XML, XSD, DTD, XQEURY processing (4) Syntax cheking of fuzzy XML-tegs (5) Transformation to entity and characteristic.

O primeiro componente (1) tem a função de permitir a descrição de XML com variáveis e funções difusas. Esse tipo de implementação de funções difusas no documento XML difuso é feita através de um aplicativo externo com o poder de computação apropriado.

Para descrever os elementos difusos de uma forma conveniente é utilizado o XSD (2), com a função de associação que apresenta para questão de limite um valor mínimo e uma valor máximo. O XSD é ainda manipulado nos componentes 3 e 4, sendo que no último componente (5) é onde ocorre a transformação de entidades e características.

Esse trabalho nos ajudou a entender melhor as questões de limite com valor máximo

e mínimo usando a função de associação, a caracterizar melhor a informação difusa no documento com as devidas marcações.

Isso trouxe conhecimento da possibilidade de trabalhar com informação difusa e funções difusas no banco de dados de documentos, podendo processá-los e manipulá-los (conforme o modelo proposto no trabalho).

A contribuição desse trabalho em comparação aos outros foi menor, contudo as dúvidas em relação ao tema também foram menores, visto que este trabalho estava sendo bem mais direto aos assuntos de banco de dados de documentos, extração de dados e uso de informação difusa e XML difuso.

4.4.6. Trabalho de Turowski & Weng

O trabalho de (Turowski & Weng (2002)) se baseia em criar um DTD nebuloso para representar as informações nebulosas no banco de dados de documento XML, e com isso ampliar as integrações existentes em sistemas e aplicações.

Esse DTD vai definir como os elementos ou atributos do documento XML devem apresentar as informações imperfeitas, isso é possível devido à capacidade do XML de poder compartilhar diferentes categorias de informação em diferentes sistemas.

O DTD difuso vai tomar o papel de intermediador e assim comunicar os sistemas como se deve ler as informações contidas no XML que vai ser consumido. Além deste trabalho outros trabalhos também abordam a criação de um DTD nebulos como Tseng et al. (2005), Ma & Yan (2007a), Panić et al. (2014).

Mesmo que nosso trabalho não aborde o XML em sua totalidade, o modo que esses trabalhos se proporão a chegar na ideia de um DTD nebuloso funcional vai servir como um guia, para podermos chegar em uma definição de estrutura de um documento JSON com a mesma finalidade.

Isto é dentro do JSON existirá uma formatação que os outros sistemas ao consumi-lo entenderão a estrutura assim evitando o máximo de conflitos de dados possíveis.

Por exemplo, dentro do JSON podemos ter no documento um campo de nome idade e seu valor ao invés de ser representado por um número ele pode estar sendo representado pelo termo ”Adulto”ou por um grau de associação 0.6 (60 por cento adulto).

4.4.7. Trabalho de Fosci & Psaila

O trabalho de Fosci & Psaila (2021b) tem o objetivo de explorar as variedades de dados dentro de um documento JSON em portais de dados abertos, para isso eles se aproveitavam dos recursos do framework J-CO e da linguagem de consulta J-CO-QL.

Eles queriam fazer isso de uma forma que as consultas fossem flexíveis, uma vez que os dados são integrados no documento JSON, pra isso eles usaram o J-CO Framework, ele é usado para gerenciar grandes coleções de documentos JSON, podendo também fornecer recursos para consulta baseados em conjuntos fuzzy.

Neste trabalho podemos ver a primeira vez como seria a estrutura de um documento JSON nebuloso, onde os campos colocados são propriamente ajustados para aparecer as informações nebulosas que eles desejam.

Porém ao analisar o trabalho percebe-se que ele aborda no geral apenas uma forma de representar a informação difusa dentro do documento JSON, que também utiliza os limiares para reduzir a quantidade de dados que não fazem sentido dentro da consulta.

Em nosso trabalho pretendemos abordar pelo menos três das formas (categorias de informação imperfeita) mencionadas acima e assim conseguir flexibilizar ainda mais as consultas em qualquer repositório de dados que a API seja conectada (com o devido suporte prestado).

Na próxima seção listamos os trabalhos utilizados em uma tabela, comparamos os recursos que cada oferece e montamos o propósito inicial que este trabalho terá ao ser concluído além dos recursos necessários.

4.5. Conclusão da Revisão

A tabela da figura 4.4, apresenta cada trabalho analisado de forma resumida, trazendo o título do trablho, os autores do trabalho, uma breve descrição do trabalho e por último resultado que este trabalho obteve.

Trabalhos Relacionados			
Título	Autor (Ano)	Descrição	Resultados
A First Step Towards a Fuzzy Framework for Analyzing Collections of Json Documents	Psaila & Marrara (2019)	Tem como objetivo explorar as coleções de documentos JSON, com o intuito de flexibilizar os resultados obtidos de determinada consulta. Com a finalidade de que esses resultados sejam mais coerentes com aquilo que o usuário que está realizando a busca espera.	Conseguem propor a primeira solução para o problema de como consultar coleções heterogêneas de documentos JSON de forma flexível, por meio de conceitos vagos.
Mongodb's Javascript Fuzzer	Guo (2017)	É explorado o banco de dados de documento chamado MongoBD, que trabalha com coleções de documentos no formato JSON. Foi proposto misturar nos comandos de busca do MongoDB comandos da linguagem JavaScript, para a realização de testes randômicos no banco de dados de modo a se caçar falhas na plataforma.	Com base no conjunto existente de testes JavaScript executados por eles, conseguiram aumentar significativamente a cobertura dos testes com relativamente pouco esforço.
A fuzzy XML database system: Data storage and query processing	Jin & Veerappan (2010)	Os autores falam como eles conseguiram incorporar a lógica difusa em um sistema de banco de dados XML e também como conseguiram implementar uma linguagem de consulta difusa baseada no XQuery, permitindo que se use expressões difusas em qualquer condição de consulta realizanda no banco de dados XML	O artigo descreveu a implementação da representação de dados, sintaxe de consulta difusa e lógica de processamento de consulta difusa. O sistema foi implementado em Java. Uma direção futura é a avaliação de desempenho do sistema.
Implementing XQuery Fuzzy Extensions Using a Native XML Database	Ueng & Skrbic (2012)	Os autores implementaram uma extensão da XQuery, sendo a linguagem padrão para consultar um documento XML, para ela poder lidar com dados incertos e imprecisos.	Proporam a extensão da linguagem XQuery como forma de fornecer uma linguagem XQuery mais flexível com lógica fuzzy que permitiria adicionar prioridade e limiar na consulta.
The Method of Big Data Processing	Shakhovska (2017)	Se baseia inicialmente na exploração da definição do Big Data e a descrição de suas principais caracteísticas, além de explorar a linguagem XML com uma arquitetura e padrões modernos.	O artigo resolveu importante tarefa científica de organizar e integrar recursos de informação Big data e pesquisa para identificar as principais tendências de previsão processos ambientais e econômicos na região.
Representing and Processing Fuzzy Information—an XML-Based Approach	Turowski & Weng (2002)	Criar um DTD nebuloso para representar as informações nebulosas no banco de dados de documento XML, e com isso ampliar as integrações existentes em sistemas e aplicações.	Introduziram uma sintaxe formal para tipos de dados difusos importantes. Além disso, tornaram essa sintaxe operável definindo DTDs apropriadas e mostraram como informações fuzzy, cuja descrição é baseada nessas DTDs, podem ser trocadas entre sistemas aplicativos por meio de XML.
Towards Flexible Retrieval, Integration and Analysis of JSON Data Sets through Fuzzy Sets: A Case Study	Fosci & Psaila (2021)	Explorar as variedades de dados dentro de um documento JSON em portais de dados abertos, para isso eles se aproveitavam dos recursos do framework J-CO e da linguagem de consulta J-CO-QL com consultas flexíveis.	Ilustraram os benefícios da adoção de uma nova tecnologia para gerenciar conjuntos de dados em um estudo de caso prático.
FuzzyScript: Um método de consulta a banco de dados de documento usando informação imperfeita	Leandro Brito & Luiz Mariano (2022)	Criação de uma API que vai realizar a interpretação da informação imperfeita através de um documento JSON nebuloso para ques seja possível realizar consultas em um BDD com um retorno de dados maior do que se fosse feito com lógica booleana.	A API consegue lidar com a interpretação de 3 categorias de informação imperfeita, conseguindo levar a informação tradada para o banco, assim obtendo um retorno significativo dos dados que tem a ver com a consulta solicitada.

Figura 4.4. Tabela de Trabalhos Relacionados (do autor)

A finalizade do Mapeamento Sistemático foi indentificar dentro dos trabalhos da

área científica a seguinte questão: "Quais são os estudos primários que abordam o uso de lógica fuzzy em Bancos de Dados Orientados a Documentos?".

Os trabalhos relacionados nos ajudaram a responder as questões levantadas anteriormente:

Q1: Tem efeito positivo utilizar a lógica fuzzy em banco de dados?

Resposta: Com base nos artigos analisados, temos sim um efeito positivo ao utilizar a lógica fuzzy em banco de dados, os experimentos apresentados mostraram eficiência em sua proposta, mesmo aqueles que ainda estão na fase conceitual sem um protótipo apresentável.

Eles consideram principalmente o fato de dados que não eram antes retornados aparecerem agora, variando de conceito e técnica utilizada para chegar nesse resultado.

Q2: Quais técnicas estão sendo utilizadas para se chegar nesse resultado?

Resposta: Com base nos artigos analisados, encontramos diversas abordadens para trabalhar com informação imperfeita em banco de dados de documentos. No trabalho de Psaila & Marrara (2019) eles optaram por flexibilizar as consultas no banco de dados utilizando grau de similariedade.

Em Guo (2017) eles misturam os comandos de busca do banco de dados MongoDB com javascript. No trabalho de Jin & Veerappan (2010) eles modificaram a linguagem de consulta XQuery a estentendo para um novo formato para trabalhar com termos difusos. No trabalho de Turowski & Weng (2002) eles apresentaram um modelo de estrutura para um documento para tipos de dados nebuloso.

Em Fosci & Psaila (2021b), os autores introduziram o conceito de limites na consulta e a classificação do atributo difuso (atributo da informação nebulosa). No trabalho de Shakhovska (2017) é construído um modelo de associação entre entidades e características para compartilhamento de dados heterogêneos.

No trabalho de Ueng & Škrbić (2012) foi proposto inserir a lógica fuzzy nos conceitos da linguagem XQuery com um interpretador para entender a extensão aplicada.

Q3: Quais trabalhos apresentam técnicas mais expressivais que podem ser usadas

como base para uma nova usada em nossa pesquisa?

Resposta: Dos sete trabalhos selecionados, os que apresentaram técnicas mais expressivas e foram possíveis de usar como base para uma nova técnica foram os de Turowski & Weng (2002), Ueng & Škrbić (2012) e Psaila & Marrara (2019), pois esses trabalhos nos mostraram técnicas que lidavam mais com programação e interação com documentos seja XML ou JSON.

E assim foi dado início ao desenvolvimento do nosso projeto na parte técnica, pegamos a ideia de modificar o XML para a inclusão de elementos difusos e colocamos para ser o documento JSON a contemplar essa estrutura. Com o Javascript a manipulação desse documento JSON será a alternativa mais acessível, já que assim não precisamos nos preocupar de realizar uma conversão de documento para outro.

Para a elaboração desse documento JSON difuso vamos utilizar alguns dos fundamentos do trabalho de Fosci & Psaila (2021b), introduzindo o conceito de limites na consulta, classificação do atributo difuso e outros elementos que vamos construindo durante a dissertação.

Por fim também vai ser apresentado a nossa API, que vai ser a responsável por receber e devolver os dados pesquisados pelo usuário e devolver eles no formato que definirmos por padrão.

4.6. Considerações Finais

Neste capítulo foi selecionado, em um mapeamento sistemático os trabalhos que consideramos mais relevantes e criamos um fluxo para representar melhor elas dentro de uma tabela.

Um ponto a se destacar é que existe um grande volume de materiais que abordam o uso da informação imperfeita em banco de dados de documentos. Entretanto, o modo que os autores usam para elaborar as técnicas e os protótipos ainda não usufruem totalmente de alguns recursos tecnológicos, como, por exemplo, uma linguagem de programação. Prevalecendo o uso de métodos tradicionais que são efetivos, porém mais trabalhosos.

No próximo capítulo vamos apresentar o objetivo principal da dissertação com as ideias apresentadas anteriormente nesse capítulo.

Capítulo 5

Concretização das Contribuições Fundamentais da Pesquisa

A partir dos trabalhos relacionados mais os elementos vistos em capítulos anteriores, podemos dizer que esta dissertação propõe a criação de um documento JSON que vai conseguir lidar com a informação imperfeita. Com a definição desse documento JSON também vamos implementar uma API (Interface de Programação de Aplicação) feita em JavaScript que vai intermediar os dados entre o usuário e o Banco de Dados de Documentos.

Colocamos as seguintes contribuições como as mais importantes desta dissertação:

1. A definição de um documento JSON nebuloso.
2. A definição de uma arquitetura intermediária que chamamos de *FuzzyScript*

A seguir vamos ver em detalhes cada uma dessas contribuições.

5.1. Definição de um Documento JSON Nebuloso

Esse documento é uma extensão do documento JSON tradicional. Ele foi desenvolvido para trabalhar com a informação imperfeita (ou dado nebuloso), de forma que possa ser usado para obter dados de consultas realizadas com esse documento em um BDD.

Sua estrutura segue a mesma do documento JSON padrão apresentada na definição

anterior. No entanto, foram feitas algumas adições dentro desse novo documento que permitem a entrada e saída de informações imperfeitas no retorno da consulta realizada.

Quando este documento é utilizado para consulta, é possível inserir a informação imperfeita em qualquer uma das formas de interpretação mencionadas anteriormente, desde que seja respeitado os padrões de estrutura já estabelecidos no documento JSON padrão. Quando este documento é utilizado para retorno dos dados, ele definirá, dentro de sua estrutura o dado imperfeito pesquisado, junto do grau de associação nebulosa, para ser comprovado o quanto aquele dado tem de associação com a consulta.

A definição de um documento JSON Nebuloso pode ser representada da seguinte forma:

1. Documento = { Objeto [,Objeto, . . .] }
2. Objeto = Chave: { Valor, Termo Nebuloso: Valor } [, Chave: { Valor, Termo Nebuloso: Valor }, . . .]
3. Chave = String
4. Valor = String | Inteiro | Número | Matriz | Nulo | Objeto.
5. Termo Nebuloso = Número | Matriz | Objeto.

No item (1) mostra o documento como é montado, ele pode ser composto por um objeto ou uma matriz de objetos. Em (2) descreve os possíveis valores no objeto, podendo ser um atributo de chave simples ou uma matriz de chaves, e dentro dessas chaves podemos ter valores ou termos nebulosos. É dito em (3) que a chave só pode ser do tipo string, e por fim em (4) mostra que o valor na chave pode ser de diversos tipos, incluindo um novo objeto, assim repetindo a sequência de (2), (3) e (4).

Em (5) mostra que no termo nebuloso só podem aparecer números, matrizes ou objetos por conta do tratamento que a informação imperfeita recebe ao ser enviada ao BDD, esse tratamento será realizado por uma Interface de Programação de Aplicativos (API), vamos nos aprofundar mais dessa API nos próximo capítulo.

Os atributos do documento JSON Nebuloso serão como um espelho dos campos que o banco de dados possui, já que não é possível realizar uma consulta em um BDD de um campo não existente. O usuário montará esse documento com base no que lhe interessa pesquisar com o parâmetro desejado, seja um valor comum ou um valor difuso.

Por exemplo, em nossos testes estamos usando o BDD de uma escola para validar as consultas realizadas. Dentro desse BDD existe uma tabela de usuários contendo os campos: ”id”, ”name”, ”age”e ”size”. Será excluído o campo ”id”do documento de consulta porque, esse campo é um campo de identificação do registro gerado automaticamente, por padrão ele é não nulo e se preenche sozinho, assim não dando a possibilidade de receber um valor nebuloso.

Como parte desta proposta sugerimos a introdução em JSON de determinados elementos difusos assim como também limiares para a definição em uma consulta. Seguindo como referência o trabalho de (Fosci & Psaila (2021a)), onde eles utilizaram o framework J-CO, sendo uma coleção de códigos para expandir o documento JSON deixando alguns atributos ou um conjunto de atributos inseridos em uma coleção de dados difusos.

Nesse trabalho eles se aprofundaram mais na categoria de informação imperfeita vagueza, utilizando a mesma ideia de outro trabalho anterior, que até foi mencionada na revisão bibliográfica desse projeto de pesquisa.

Eles filtraram vários documentos de regiões e temperaturas e após o filtro acrescentaram as propriedades fuzzy nos documentos restantes, assim gerando um tipo de documento JSON Nebuloso.

A diferença desse trabalho para o nosso é que não vamos apenas limitar e filtrar a informação imperfeita, mas também tratar ela para poder ser compatível com os demais BDD por meio da API estabelecendo um padrão de compatibilidade.

No início dessa seção definimos a estrutura do documento JSON nebuloso, agora será mostrado sua estrutura utilizando as categorias de informação imperfeita já apresentadas. Por conta do documento JSON suportar estruturas de vetores dentro de um atributo ou itens aninhados, as possibilidades de produzir diversos documentos aumentam bastante e podemos considerar isso um bom desafio para nossa API, pois ela deverá processar todos eles e aprender com cada padrão específico.

5.1.1. Documento JSON Difuso com Imprecisão

Vamos supor que o usuário, para fazer uma consulta no BDD forneça o documento conforme a figura 5.1:

Figura 5.1. Documento JSON Difuso com Dado da Categoria Imprecisão Para Entrada de Dados (do Autor)

Perceba que no campo "age" há um valor difuso, mais precisamente da categoria de informação imperfeita imprecisa, como descrito na introdução, a informação estando nesse formato não conseguimos efetuar uma consulta no BDD de maneira tradicional, pois o banco não está preparado para lidar com esse tipo de dado, fazendo com que essa consulta retorne provavelmente uma coleção de documentos vazia ou indefinida.

Para contornar esta situação, além de desenvolvermos o modelo de documento JSON nebuloso, teremos a API que vai interpretar esse JSON difuso e com isso conseguir realizar a consulta no BDD retornando uma coleção de documentos. Com dados que podem ter mais compatibilidade com o que usuário espera. Além deste exemplo também desenvolvemos outros modelos de documento JSON difuso para a entrada e saída do banco de dados por intermédio da API.

Perceba que aqui está sendo feito um trabalho de analisar os campos do documento e após a detecção do tipo de cada atributo é determinado se aquela informação é imperfeita ou não, caso houvesse mais de um campo ambos iriam passar pelo mesmo processo.

A partir do documento da figura 5.1 fazendo os devidos tratamentos da informação imperfeita da categoria imprecisão, temos a seguinte resposta da API na figura 5.2. Re-

tornando com a idade certa do usuário João da Silva, e não com um vetor de idade como antes.

Figura 5.2. Documento JSON Difuso com Dado da Categoria Imprecisão Para Saída de Dados (do Autor)

É pretendido com esses modelos tratar todas as categorias de informação imperfeita, conforme a evolução do projeto, com isso a consulta sempre vai conter um retorno, seja ela um documento com resultados ou um documento vazio caso não haja dados que satisfaçam as condições de consulta solicitadas.

5.1.2. Documento JSON Difuso com Incerteza

Esse documento pode ter um dado nebuloso contendo um valor composto por um vetor que armazena um valor e sua probabilidade de ser o valor verdadeiro. Ele não traz um valor exato para fazer a consulta no banco de forma tradicional, a figura 5.3 mostra um documento com esse tipo de estrutura, no campo "height(altura)" em centímetros, no vetor temos o peso seguido da probabilidade de que esse é o peso do João da Silva (que vai de 0 a 1 em decimal):

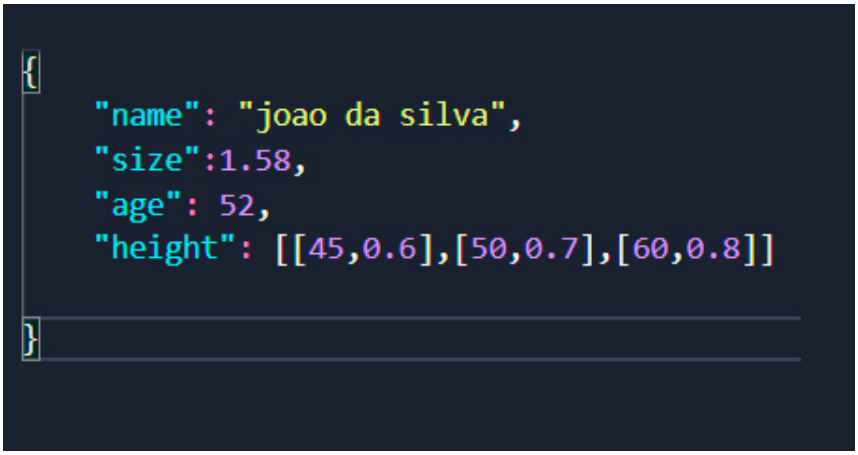

Figura 5.3. Documento JSON Difuso com Dado da Categoria Incerteza Para Entrada de Dados (do Autor)

No final da consulta, após ter o valor tratado pela API o resultado será o documento da figura 5.4, com a altura do usuário:

Figura 5.4. Documento JSON Difuso com Dado da Categoria Incerteza Para Saída de Dados (do Autor)

5.1.3. Documento JSON Difuso com Vagueza

Esse documento conforme figura 5.5 comporta um dado nebuloso do tipo string de nome adulto, poderia ser outro nome dependo do campo, normalmente por ser tratar de uma idade deveria existir nesse campo um valor numérico do tipo inteiro, a figura 5.5 mostra esse documento contendo esse dado no campo “age(idade)”:

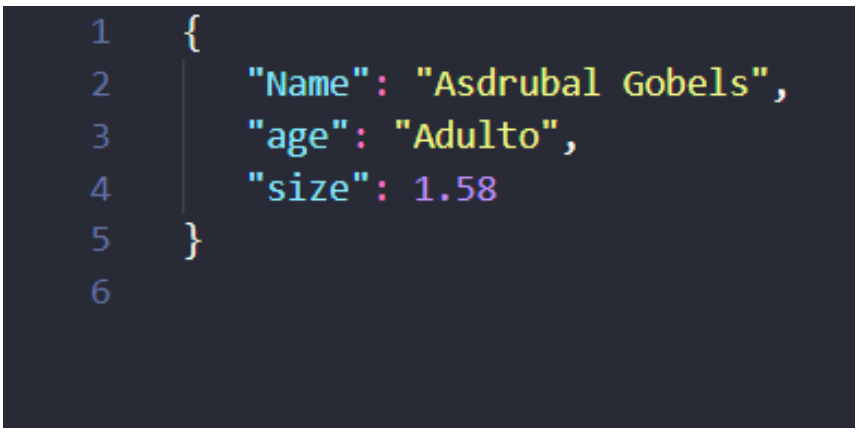

Figura 5.5. Documento JSON Difuso com Dado da Categoria Vagueza Para a Entrada de Dados (do Autor)

Depois de ter o valor tratado pela API o resultado será o documento da figura 5.6, a idade será mostrada de forma exata, mas perceba que vamos retornar também o quanto o usuário é adulto em uma escala de 0 a 1. Isso vale também caso seja pesquisado usando o atributo idoso ao invés de adulto.

Figura 5.6. Documento JSON Difuso com Dado da Categoria Vagueza Para a Saída de Dados (do Autor)

Nesse caso em específico utilizamos o limiar de consulta, deixando com o valor

0.6 assim qualquer documento encontrado com um limiar menor que esse não irira ser retornado na consulta apenas os que fosse maiores ou iguais a esse valor de limiar.

5.2. Definição de uma Arquitetura Intermediária: *FuzzyScript*

Para melhor entender o funcionamento da API, desenvolvemos um fluxograma mostrando de forma mais clara as etapas que vão ocorrer desde o momento da consulta até o retorno da coleção de documentos, conforme a figura 5.7:

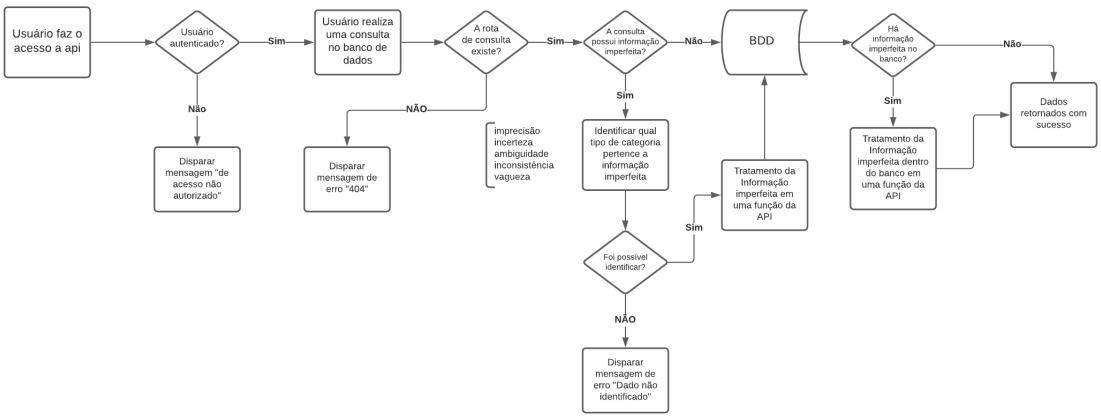

Figura 5.7. Fluxograma dos Processos Existentes na API

Note que a API lida com três situações dependendo do dado inserindo na consulta, quando o dado não é uma informação imperfeita, quando ele é uma informação imperfeita, e quando é uma informação imperfeita, mas está fora dos padrões estabelecidos nas cinco categorias apresentadas anteriormente.

Tudo começa pela entrada de dados que usuário realiza na API, isto vai ser feito de maneira autenticada para evitar acesso não autorizado a possíveis dados sigilosos do BDD, assim evitando vazamentos de dados. Ao ser autenticado a API vai gerar um token e com esse token o usuário vai ter livre acesso as interações que a API possui.

Então o usuário poderá fazer, por intermédio da API uma consulta no banco de dados, ele deve escolher uma rota que a API possui e assim enviar o JSON difuso na entrada de dados. Perceba que a rota deve exisitr pois ela é a ponte entre o JSON nebuloso e as funções da API que vão lidar exatamente com esse tipo de dado.

Em outras palavras se o usuário realizar a consulta em uma rota diferente daquela

informação imperfeita, a API vai sinalizar isso com um aviso alertando ele a escolher a rota certa para que essa consulta seja feixa com êxito.

O próximo passo então é identificar a informação imperfeita e depois ver a qual categoria ela pertence, pois se não for possível definir qual tipo de informação é a consulta não vai retornar nada a não ser uma mensagem de erro.

Depois de identificar essa informação vamos chamar algumas funções na API que vão tratar essa informação, e após o tratamento com informação normalizada, será feito o envio da mesma para o banco de dados, para que assim a consulta ocorra normalmente e ser possível obter um resultado.

De uma forma mais rápida e clara, a API vai receber o documento JSON, identificar se em um atributo do documento há uma informação imperfeita, vai tentar identificar o tipo dela com base nos 5 tipos já declarados, e após a indentificação vai tratar a informação e continuar a consulta no BDD. O resultado esperado será o das figuras apresentadas nas seções anteriores, um documento JSON difuso que tem como saída de dados as informações tratadas.

Para desenvolver essa arquitetura vamos usar como base o trabalho de Fosci & Psaila (2021b), que conforme falado no capítulo que aborda o Mapeamento Sistemático tem o objetivo explorar as variedades de dados dentro um documento JSON em portais de dados abertos.

Em nosso trabalho pretendemos abordar inicialmente três categorias das cinco mencionadas no trabalho acima, e assim conseguir flexibilizar ainda mais as consultas em qualquer repositório de dados que a API for conectada (com o devido suporte prestado).

Queremos que o *FuzzyScript* em um futuro distante consiga lidar com mais tipos de informação imperfeita, para isso vamos treiná-lo inicialmente com as categorias já mencionados nesse trabalho, para que assim possamos colocar outros tipos de categoria melhorando a capacidade de reconhecimento da informação imperfeita.

5.3. Considerações Finais

Neste capítulo apresentamos as contribuições e soluções para atingir o objetivo principal dessa dissertação, com as ideias que consideramos mais relevantes de cada tra-

balho do Mapeamento Sistemático. Uma delas que já falamos aqui é o conceito do documento JSON difuso e como ele vai funcionar dentro da API.

O próximo capítulo descreve melhor a estrutura da API e o que já está funcional na API com os comandos documentados.

Capítulo 6

Apresentação do Protótipo do *FuzzyScript*

Nesse capítulo falaremos com mais detalhes como funciona o *FuzzyScript* e os processos que ele realiza. Depois mostraremos a arquitetura por trás dele com algumas informações técnicas, como estrutura de pastas, linguagem utilizada, etc.

6.1. Funcionalidades do *FuzzyScript*

A função principal da API é receber um documento JSON (possuindo atributos normais ou difusos), identificar se há algum tipo de informação imperfeita dentro desse documento, caso haja a informação fazer o tratamento desse JSON e após realizar uma busca no banco de dados usando esse mesmo JSON. A API irá retornar os dados da consulta em um novo documento JSON nebuloso com a informação imperfeita em evidência.

Para esses processos internos serem executados foram criados métodos dentro da API que envolvem cálculos de associação e tratamentos das informações imperfeitas mencionadas no capítulo 2. Caso o usuário deseje usar outro BDD ele precisa definir o BDD e a estrutura de conexão nas configurações da API, esse procedimento irá garantir a portabilidade de uma estrutura de BDD para outra, não perdendo nenhuma funcionalidade em um BDD específico.

Cada rota da API é usada para um determinado tipo de consulta, assim podendo ser expandida conforme a necessidade. Por exemplo, existe uma rota que ao ser acionada

na API irá realizar uma consulta no banco de dados de documentos, e também irá aplicar a função trapezoidal nos resultados obtidos com base no dado nebulosos informado pelo usuário.

Ao optar por usar esse modelo ao invés de instruções que já vem por padrão nos BDD, ganhamos mais possibilidades de manipular o documento JSON antes e depois da consulta.

Abaixo é apresentado um exemplo de rota que a categoria da informação imperfeita que veio da API é das informações imprecisas. O código começa fazendo uma busca no banco de dados MongoDB pelos usuários sem nenhuma filtragem das informações.

Em seguida, o retorno da consulta é enviado a função trapezoidal, onde cada retorno da consulta será calculado, e aqueles que tiverem menos de 0.6 de associação são removidos da coleção de documentos. Assim, o resultado é somente removido da coleção se ele tiver uma associação matematicamente baixa e não por condição lógica.

Estamos usando para a construção da API a linguagem de programação Javascript. Javascript é uma linguagem de programação bem estruturada e de alto nível, que antigamente era interpretada somente pelos navegadores, atualmente sua versatilidade fez com que pudesse ser reconhecido em diversos dispositivos, podendo até ser usado para a construção de aplicativos de celular para os sistemas operacionais Android e IOS.

A figura 6.1 mostra a aplicação do código em Javascript que tem o papel de tratar o dado nebuloso passado para ela. Ao se utilizar o Javascript (Silva (2010)) para este protótipo ganhamos uma boa variedade de recursos, pois o Javascript lida bem com JSON.

Figura 6.1. Rota Dentro da API Usada Para se Fazer uma Busca por Idade Contendo o Dado Nebuloso Idoso

Existem funções do próprio Javascript que podemos usar para tratar o documento JSON, pois o documento JSON pode ser interpretado como um objeto na linguagem Javascript, e com isso podemos manipular qualquer dado que esse documento venha conter. Podemos descrever um objeto como uma estrutura de coleção de dados ou funções relacionadas, ele é independente podendo ter diversas propriedades e tipos.

Pense no objeto como se fosse uma xícara, a xícara possui propriedades, ela tem uma cor, uma forma, um peso, um material que a compõe, etc. Essa é a mesma forma que o objeto funciona.

Para exemplificar melhor podemos imaginar uma variável chamada data, esta variável recebe a resposta da API em JSON ao receber essa resposta e a mesma não for vazia já podemos executar as funções de manipulação. Uma delas é a função ”map()”onde mapeamos o conjunto de objetos armazenados na variável data e com esse mapeamento podemos atribuir esses valores a um novo conjunto, organizando as propriedades conforme desejamos. Esse exemplo pode ser mais entendido ao visualizar a figura 6.2:

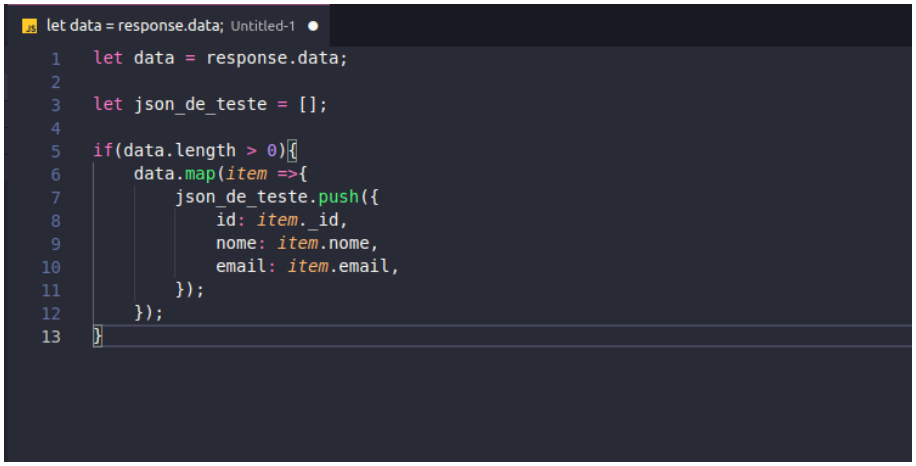

Figura 6.2. Uso da Função map() Para a Criação da Estrutura do Documento JSON

Esse é o primeiro passo para formarmos a estrutura do documento JSON desejado. O segundo passo é manipular os dados que vierem na função ”map()”, devemos lembrar que o dado nebuloso pode não estar nos comandos de consulta e sim nos resultados obtidos no banco.

Por exemplo, se vir um dado nulo na função ”map()”, vai existir outra função que vai lidar com os dados nulos, não os excluindo, mas os tratando para que a informação referente aquele dado não se perca. Esse tratamento vai envolver técnicas de média matemática e normalização para que o dado não sofra muitas modificações e perca sua essência original (mesmo estando nulo aparentemente).

Dentro do protótipo será aproveitado diversas funções do Javascript, seja para receber os dados, para tratar e também para reconstruir o JSON em um padrão pré-estabelecido, seja um documento JSON de saída normal ou documento JSON de sáida nebuloso. Conforme citado no capítulo 3 o documento JSON não possui uma estrutura definida, e isso da possibilidade de criar um padrão de estrutura para diversos bancos de dados de documentos. Lembrando que vamos estar inserindo também dados nebulosos no JSON gerando assim um documento JSON difuso.

6.2. Arquitetura do *FuzzyScript*

O protótipo que será apresentado junto desta dissertação é uma Interface de Programação de Aplicativos (API) em Javascript que permite a associação de valores nebulosos para

um BDD.

Essa API foi batizada durante o projeto de FuzzyScript, uma vez que usa o Javascript com lógica fuzzy para interpretar as informações imperfeitas. O objetivo principal da API é que no retorno das consultas realizadas por ela tenha mais dados retornados do que em uma consulta feita de maneira tradicional (utilizando a lógica booleana).

Para isso ela precisa receber dados, sendo eles difusos ou nao, interpretá-los e após realizar as devidas conversões para que se possa enviar esses dados ao banco para realizar a consulta.

A estrutura da API pode ser vista na figura 6.3, ela possui tanto estrutura de pastas como arquivos separados na raiz do projeto:

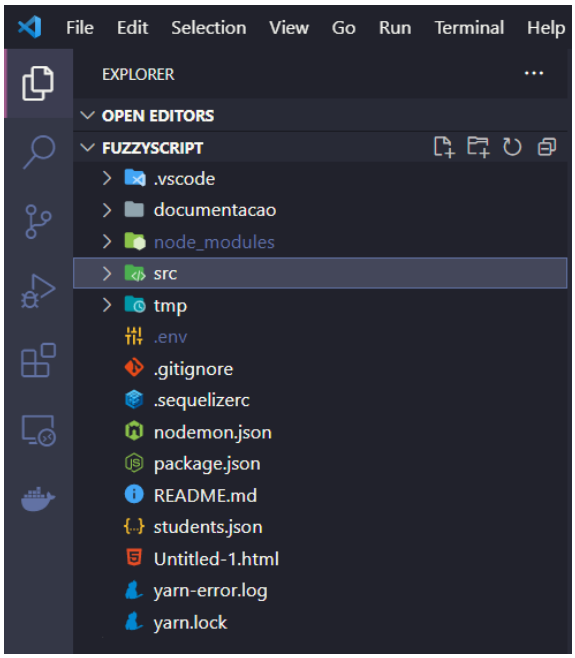

Figura 6.3. Estrutura de Arquivos que Compõe a API

Podemos resumir a estrutura de arquivos da API da seguinte forma:

- As Pastas .vscode, tmp e node-modules são arquivos de biblioteca, de estrutura do projeto e editor que usamos para programar a API e arquivos temporários compilados/armazenados durante a execução do mesmo
- A pasta documentação contém a documentação do projeto e um manual de instruções para que outros usuários possam tirar dúvidas a respeito da arquitetura da api e das funções e métodos existentes.

- Os arquivos .env, .sequelizerc, .gitignore, nodemon.json, package.json, README.md, yarn-error.log e yarn.lock fazem parte da configuração geral da estrutura, contendo bibliotecas instaladas via linha de comando. Também são encontradas as regras de estrutura e de como alguns códigos dentro da API devem se comportar.
- Os arquivos da pasta src são os arquivos que compõem as funcionalidades da API.

Essa API foi feita utilizando o conceito de back-end, podemos dizer que o back-end é a estrutura por trás de toda a aplicação. É onde ficam as regras de negócios para que um sistema ou aplicação consiga executar as devidas funções projetadas para fazer.

O servidor dessa API é feita em Node.JS e este recurso possui comunicação com diversos bancos de dados de documentos, assim facilitando a portabilidade para um banco diferente do abordado nesta dissertação.

A pasta principal da API é a pasta **src**, dentro dessa pasta esta localizado as funcionalidades da API como por exemplo, as regras para receber o documento JSON e enviar ao BDD, as regras para se conectar a diversos BDD, os qrquivos que validam as coleções existentes dentro do BDD, as rotas para se realizar determinada consulta e os arquivos que fazem essas rotas serem autenticadas para evitar perda de cobertura de dados. A figura 6.4 mostra a estrutura da pasta src com mais detalhes.

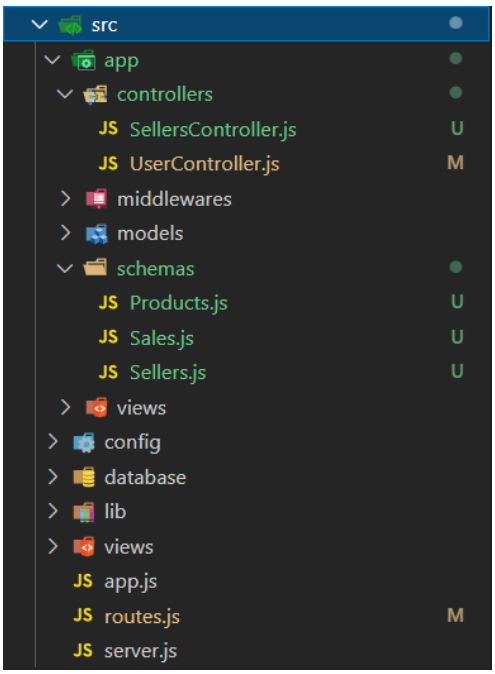

Figura 6.4. Estrutura de Arquivos da Pasta SRC

Dentro da pasta **app**, que esta dentro da pasta **src**, existem mais cinco pastas: controllers, middlewares, models, schemas e views. A pasta controllers guarda todos os controllers da API. Controller é um arquivo responsável por receber todas as requisições do usuário por meio do arquivo routes.js. Seus métodos chamados actions são responsáveis por dizer model usar e qual view será mostrado ao usuário.

A pasta middlewares contém arquivos que possuem funções que podem tratar os inputs e outputs das rotas antes ou depois que uma rota é processada, ou seja, pode-se criar um middleware que intercepta e verificar se uma requisição está enviando um header específico, caso o mesmo não esteja enviando o header a requisição vai retornar uma tela de erro para o usuário, negando a requisição de acessar uma determinada rota da aplicação.

A pasta models contém arquivos do tipo Model. Model é um arquivo composto por uma classe, que também é conhecido como Business Object Model (Objeto Modelo de Negócio). Sua responsabilidade é gerenciar e controlar como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas. É nesses arquivos que vão ter um espelho da estrutura que a coleção de documentos do BDD possui, para poder, por exemplo, validar se tal campo pesquisado realmente existe na coleção.

O mesmo conceito é usado na pasta Schemas, onde dentro dessa pasta estão os arquivos do tipo Schema que são arquivos que possuem um objeto JSON que define a estrutura e o conteúdo de seus dados. Eles estendem o esquema de documento JSON padrão, para definir o modelo de dados da aplicação e validar documentos sempre que eles são criados, alterados ou excluídos. Ele é o detentor dos dados que recebe as informações do Controller, válida se ela está correta ou não e envia a resposta mais adequada.

A pasta views contem os arquivos do tipo View, uma view tem a função de exibir uma representação dos dados modelados em um template padrão. Isso acontece após os dados solicitados pelo controller serem obtidos através do model, a view é responsável por usar as informações disponibilizadas para produzir qualquer interface de apresentação que a aplicação possa necessitar, a figura 6.5 mostra o fluxo padrão que é seguido nas aplicações que usam o MVC.

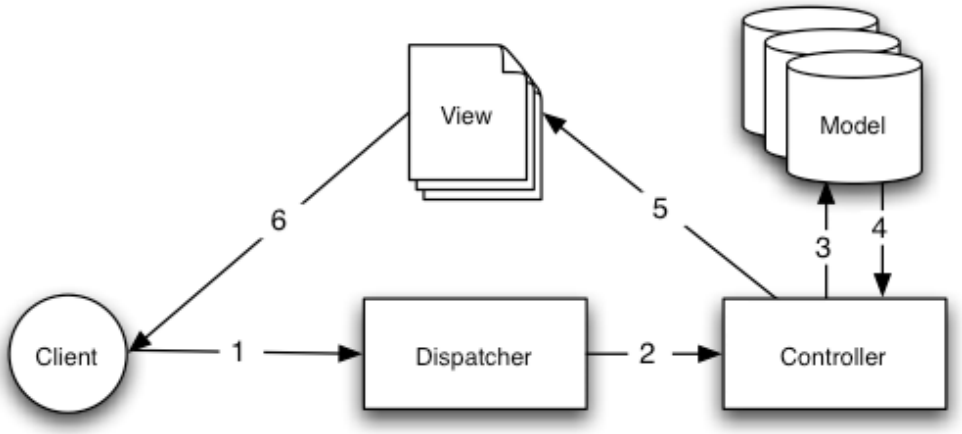

Figura 6.5. Exemplo do padrão de fluxo executado em uma aplicação MVC (CakePHP, Website (2022))

Esta é somente a estrutura inicial do projeto, ela pode ser expandida a cada nova implementação, podendo ser mais robusta e com diversas funcionalidades ao longo de seu desenvolvimento conforme novas necessidades apareçam.

6.3. Considerações Finais

Aqui apresentamos a API em sua totalidade abordando sua estrutura, funcionalidades e o motivo de ser feito dessa forma, além de introduzir também um pouco dos conceitos de MVC e a linguagem de programação Javascript, onde a API foi construída. Com direito a apresentação de alguns comandos que fizeram a API ter mais possibilidades de manipular o documento JSON proposto.

O próximo capítulo fala sobre os resultados já alcançados desse trabalho e os próximos passos desta dissertação.

Capítulo 7

Resultados do Projeto

Neste capítulo apresentamos os resultados que alcançamos durante todo o período dessa dissertação. Para ter documentado os avanços que o *Fuzzyscript* teve. Este processo envolveu testes em um BDD com documento JSON normal e JSON nebuloso, revisão de código, etc.

7.1. Resultados Alcançados

Como primeiro resultado alcançado, tivemos a API, que teve seu desenvolvimento finalizado e já está funcional. Para comprovar o seu funcionamento utilizamos um BDD feito em Mongo DB com três coleções de documentos para testar alguns tipos de consulta utilizando informação imperfeita. Assim conseguimos mostrar diversos recursos que API explora e quanto ela consegue se aprofundar na base de dados para trazer os resultados desejados.

Foi decido usar um BDD com dados voltados a área de vendas, o BDD utilizado foi nomeado para QuickSale e as coleções de documentos dentro desse BDD para products (coleção de produtos cadastrados), sales (coleção de vendas cadastradas) e sellers (coleção de vendedores cadastrados).

A API foi conectada a esse banco do Mongo DB utilizado uma biblioteca e um formato de conexão e driver recomendados pela própria documentação do Mongo DB. Os dados utilizados nesse banco do Mongo DB são todos fictícios, gerados pela ferramenta Mockaroo (https://www.mockaroo.com/), mas utilizando contextos e valores facilmente

encontrados no mundo real para que os resultados nao fossem fora da realidade.

Na coleção de documentos de produtos cadastrados existem cerca de 1000 registros, onde esses registros possuem a seguinte estrutura de dados:

1. id: representa o identificador único do documento dentro da coleção de documentos.
2. name: representa o nome do produto dentro da coleção de documentos.
3. price: representa o preço do produto na coleção de documentos.
4. stock: representa a empresa responsável pelo estoque do produto dentro da coleção de documentos.

A figura 7.1 mostra um documento dessa coleção dentro do mongoDB.

Figura 7.1. Exemplo de um Documento da Coleção de Produtos (do Autor)

Na coleção de documentos de vendedores cadastrados existem cerca de 1000 registros, onde esses registros possuem a seguinte estrutura de dados:

1. id: representa o identificador único do documento dentro da coleção de documentos.
2. name: representa o nome do vendedor dentro da coleção de documentos.
3. age: representa a idade do vendedor na coleção de documentos.
4. weight: representa o peso do vendedor dentro da coleção de documentos.
5. gender: representa o gênero do vendedor dentro da coleção de documentos.

A figura 7.2 mostra um documento dessa coelção dentro do mongoDB.

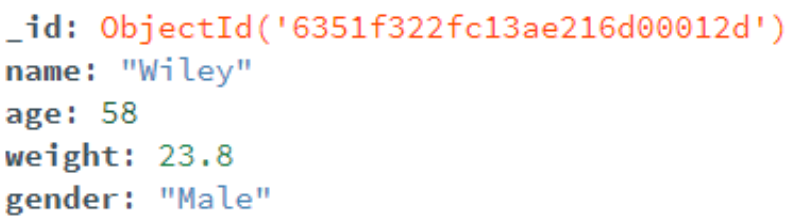

Figura 7.2. Exemplo de um Documento da Coleção de Vendedores (do Autor)

Na coleção de documentos de vendas cadastradas existem cerca de 107 registros, onde esses registros possuem a seguinte estrutura de dados:

1. id: representa o identificador único do documento dentro da coleção de documentos.
2. name: representa o nome do vendedor que efetuou a venda dentro da coleção de documentos.
3. days: representa o total de dias que as vendas ocorreram na coleção de documentos.
4. satisfaction: representa a satisfação dos clientes pelo atendimento prestado pelo vendedor dentro da coleção de documentos.

A figura 7.3 mostra um documento dessa coelção dentro do mongoDB:

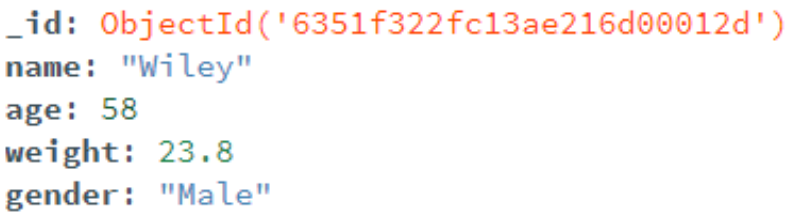

Figura 7.3. Exemplo de um Documento da Coleção de Vendas (do Autor)

O primeiro teste realizado foi na tabela sellers, onde enviamos um documento JSON nebuloso para a API utilizando a ferramenta de requisição Insomnia REST, como mostra a figura 7.4 optamos por pesquisar vendedores que tenha a idade categorizada como idoso. Note que antes do atributo “age” do documento, que corresponde a idade do vendedor, tem um atributo chamado “queryIsFuzzy”, esse atributo é responsável por

informar a API se a consulta será com atributos difusos ou não, pois a API também funciona com consultas que envolvem a lógica booleana. Isso possibilita que a API possa buscar a informação em ambos os fluxos após identificar isso na consulta.

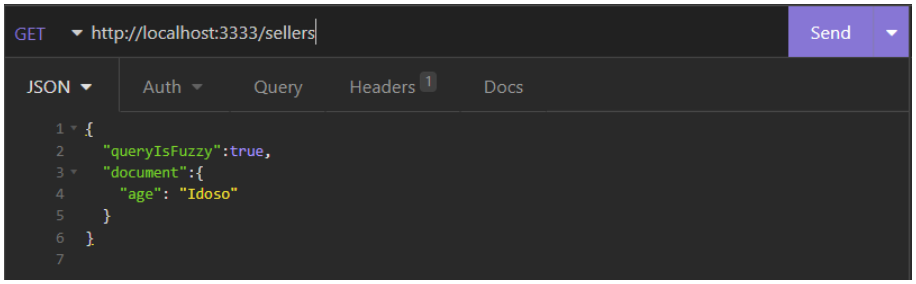

Figura 7.4. Exemplo de um Documento da Coleção de Vendas (do Autor)

No final de cada requisição da API, se a consulta é realizada com sucesso, será retornado os dados solicitados do banco de dados em um documento JSON que será armazenado em uma variável de retorno. Essa variável de retorno receberá um tratamento para que o documento JSON contido nela seja um documento JSON nebuloso, ou seja, dentro dele também irá conter o dado nebuloso pesquisado. Será colocado também o grau de pertinência que a função trapezoidal retornou, no exemplo abaixo o JSON retornou uma lista de pessoas que foram possíveis de se obter com o termo nebuloso idoso conforme mostrado na figura 7.5.

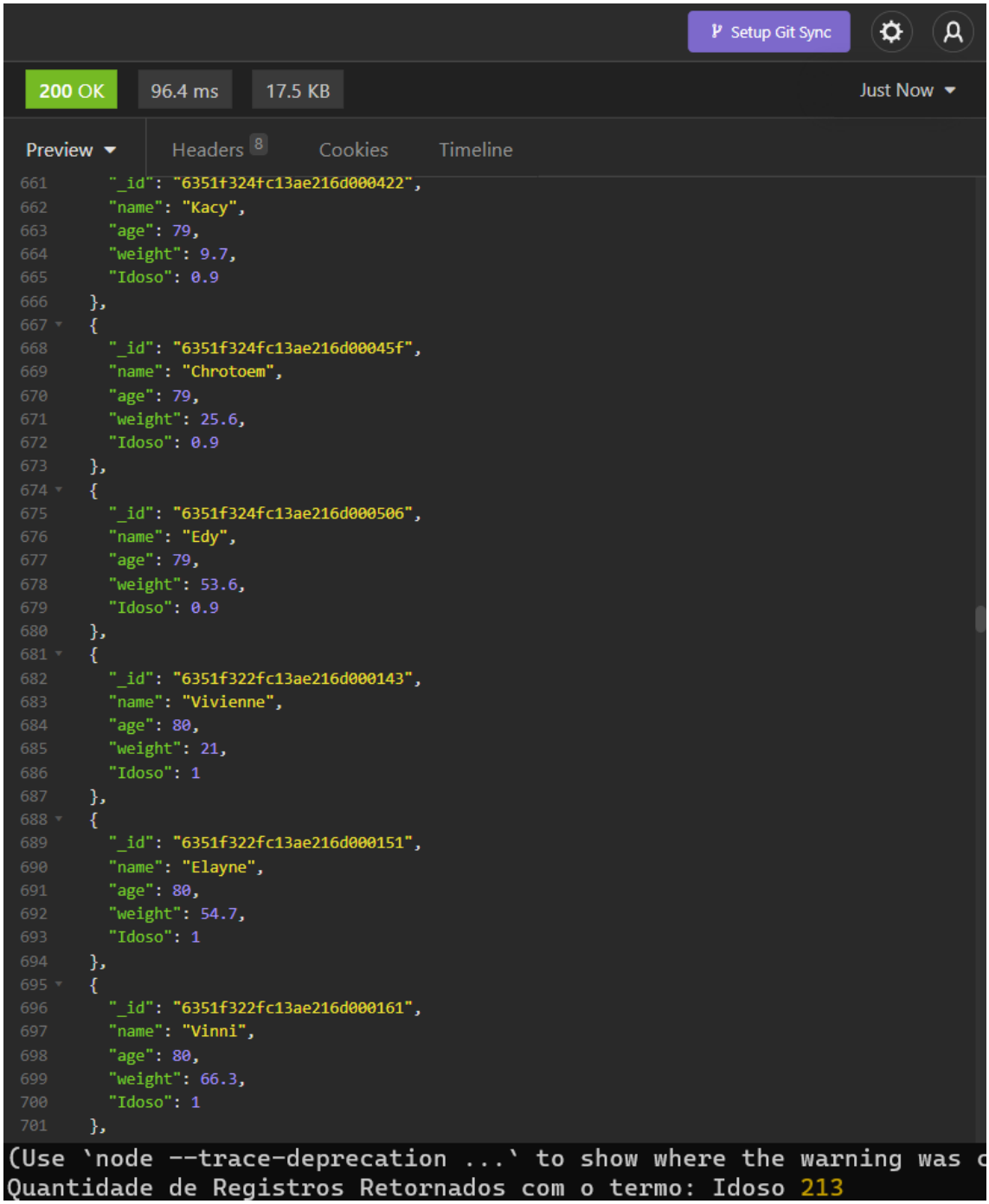

Figura 7.5. Resultado de uma Consulta Realizada na API Usando o Termo Nebuloso Idoso (do autor)

A consulta utilizando o termo idoso obteve um retorno de 213 registros de uma base de 1000 vendedores, podemos dizer então que nessa base matematicamente 21,30% dos vendedores podem ser considerados idosos (por conta dos resultados da função trapezoidal). É possível filtrar ainda mais esse resultado caso queira aumentar o grau de associação do atributo nebuloso assim trazendo usuários que seu termo nebuloso idoso superior a 0.6, por exemplo. Nos resultados apresentados foi optado por não realizar a filtragem, pois o intuito era mostrar os resultados por termo pesquisado em sua totalidade.

Foram realizados mais testes utilizando outros termos nebulosos, utilizamos os termos adulto e criança, para que pudéssemos retornar os vendedores nessa faixa de idade respectivamente. Sabemos que por se tratar de um banco fictício poderíamos colocar usuários que tivessem idade de uma criança como vendedor, isso foi propositalmente colocado para mostrar a capacidade da API em lidar com diversas idades.

A consulta utilizando o termo aulto obteve um retorno de 420 registros de uma base de 1000 vendedores, podemos dizer então que nessa base matematicamente 42% dos vendedores podem ser considerados aultos (por conta dos resultados da função trapezoidal). É possível filtrar ainda mais esse resultado caso queira aumentar o grau de associação do atributo nebuloso assim trazendo usuários que seu termo nebuloso idoso superior a 0.6, por exemplo. Nos resultados apresentados foi optado por não realizar a filtragem, pois o intuito era mostrar os resultados por termo pesquisado em sua totalidade.

A consulta utilizando o termo criança obteve um retorno de 124 registros de uma base de 1000 vendedores, podemos dizer então que nessa base matematicamente 12,4% dos vendedores podem ser consideradas crianças (por conta dos resultados da função trapezoidal). Lembrando que esse cenário foi propositalmente colocado para mostrar que a API poderia lidar com diversas idades que fossem apresentadas. Nos resultados apresentados foi optado por não realizar a filtragem, pois o intuito era mostrar os resultados por termo pesquisado em sua totalidade.

A figura 7.6 e figura 7.7 mostra os resultados obtidos usando os termos criança e adulto:

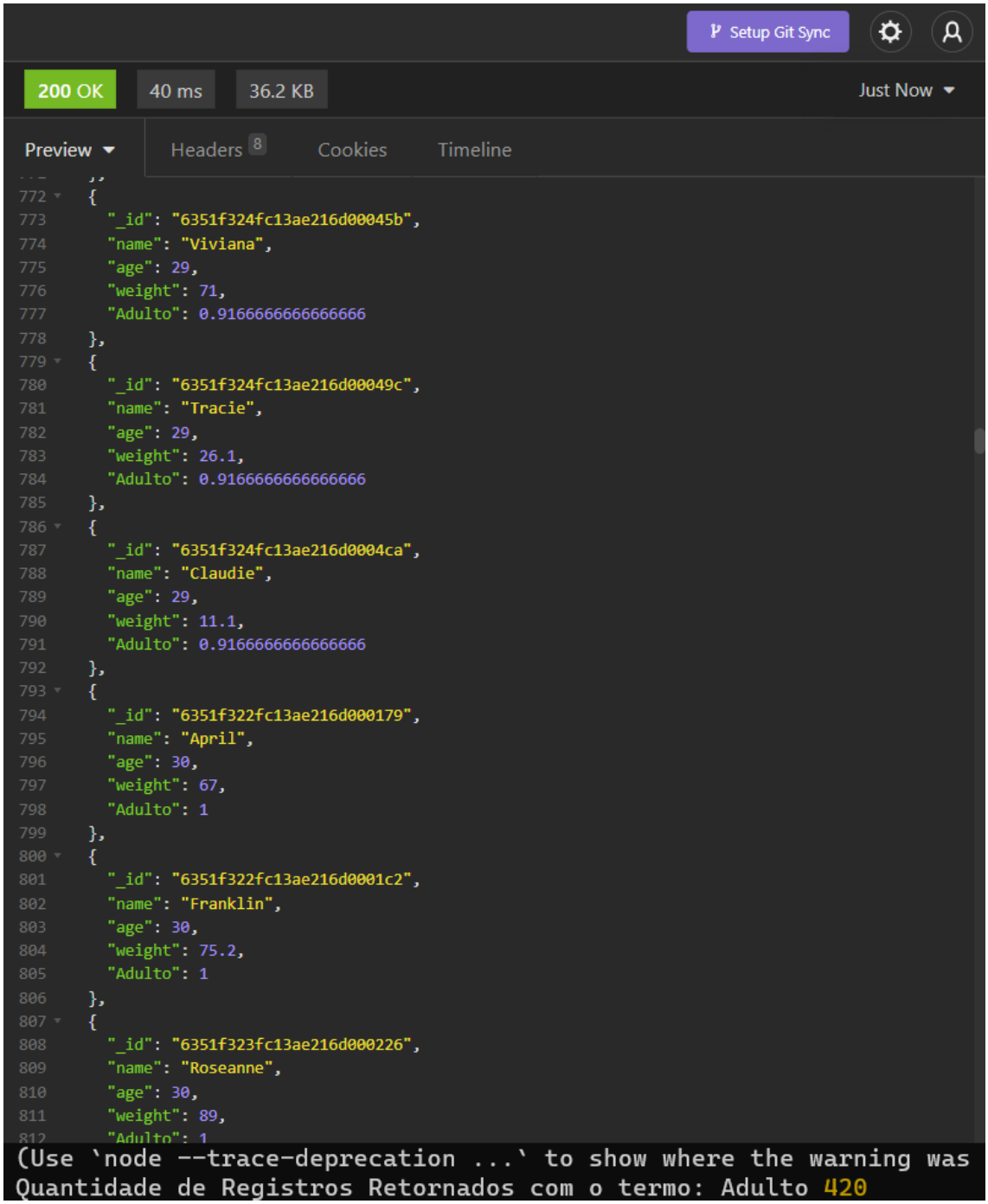

Figura 7.6. Resultado de uma Consulta Realizada na API Usando o Termo Nebuloso Adulto (do autor)

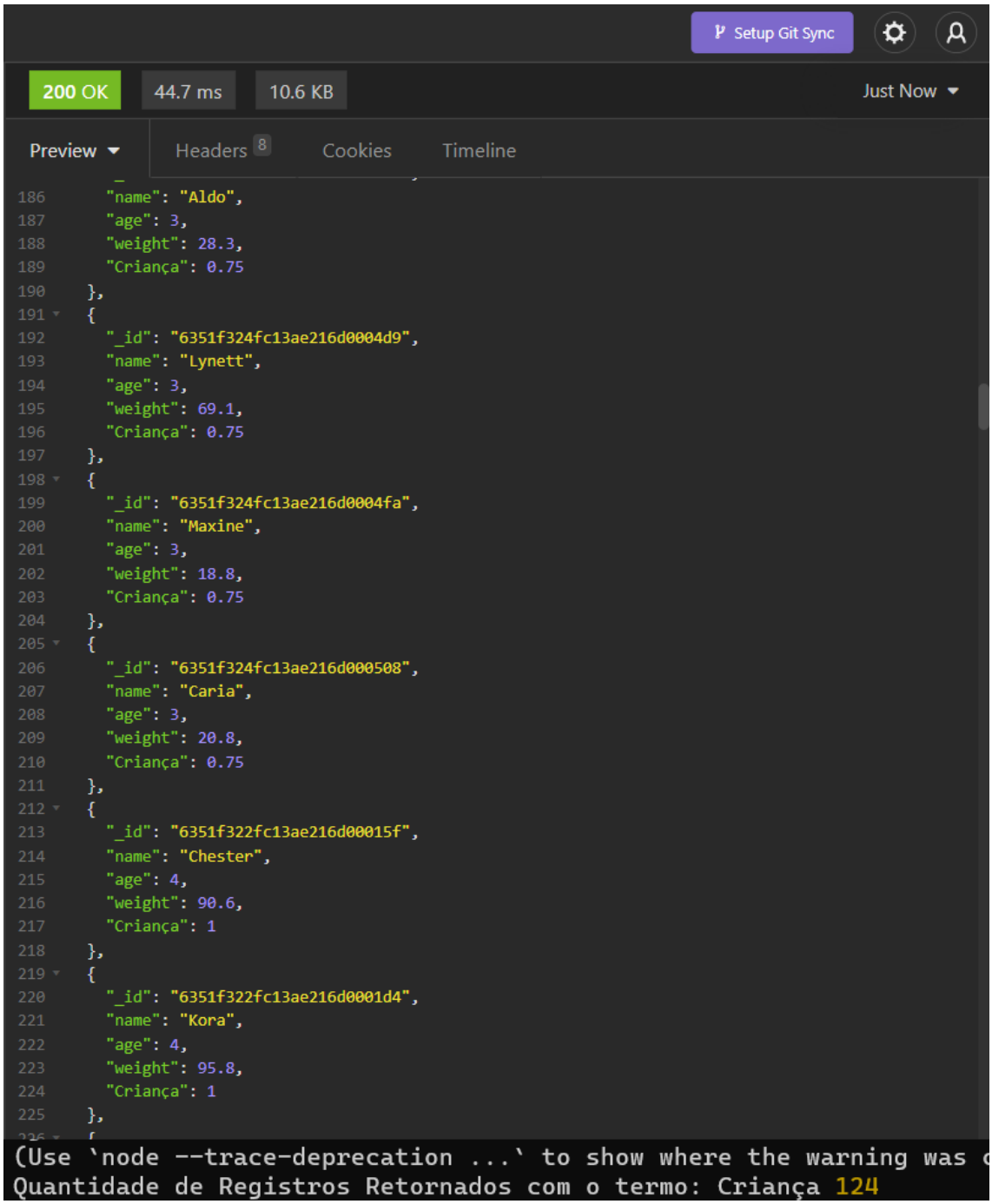

Figura 7.7. Resultado de uma Consulta Realizada na API Usando o Termo Nebuloso Criança (do autor)

Hoje a API consegue trabalhar com 3 categorias de informação imperfeita, mas como trabalho futuro exploraremos todas as cinco classificações de informação imperfeita e ao longo do tempo alimentar os resultados obtidos e alocar isso em um aprendizado de máquina. Alocando novos recursos que vão ser implementados na API em um futuro distante. A API no futuro não irá ser limitada apenas a fazer consulta de dados, mas sim conseguir realizar outras funções que API tradicionais já fazem, como criação, edição e remoção de dados. Assim podendo ser usada em outros tipos de aplicação também, visto que com uma interface gráfica bem definida esta API pode se converter em um sistema médico, por exemplo.

Nesse cenário o usuário colocaria em suas consultas as informações imperfeitas e após a captura de dados a interface gráfica iria submeter os dados até a API e a mesma faria o que já foi descrito nos capítulos acima para efetuar as consultas e se obter os resultados desejados para uma tomada de decisão mais precisa.

No próximo capítulo será abordado o cronograma de desenvolvimento envolvendo também as etapas de construção da API.

Capítulo 8

Conclusão e Considerações Finais

8.1. Considerações das Contribuições

Esta dissertação teve como contribuições importantes os seguintes elementos:

1) A definição e criação de um documento JSON nebuloso para consultas envolvendo informação imperfeita.

Durante o desenvolvimento desta pesquisa foram realizados diversas análises voltadas a criação um recurso que conseguisse enviar a informação imperfeita ao BDD de uma forma similar ao que existe hoje, sendo por meio de um documento JSON ou XML.

Nos trabalhos analisados encontramos diversas estratégias que poderiam ser usadas para se chegar nesse resultado, mas o que nos chamou mais a atenção para implementar foi a definição e criação do documento JSON nebuloso, embora os outros métodos também possuam sua relevância. Ao desenvolver a extensão de uma estrutura baseada em uma já existente, devemos considerar inicialmente que essa nova estrutura irá respeitar as regras de formatação e estrutura que já existem e formar novas, expandindo os padrões existentes para fazer sentido o seu uso e criação.

2) A definição e implementação de uma arquitetura intermediária que chamamos de *FuzzyScript*, para tratamento e realização de consultas envolvendo informação imperfeita.

Para conclusão e testes dessa definição de documento foi elaborado uma API em

Javascript para interpretar o documento JSON nebuloso e com isso ser possível realizar buscas em um BDD usando dados nebulosos (ou informação imperfeita), hoje ja conseguimos por meio dessa implementação interpretar três categorias diferentes de informação imperfeita e como trabalho futuro desejamos englobar as duas restantes totalizando as cinco categorias.

O banco utilizado para esses testes foi o BDD Mongo DB, porém a sua estrutura foi pensada para lidar com outros BDD existentes, bastando apenas mudar o driver e a biblioteca de conexão com o BDD (caso exista). É necessário lembrar que o BDD deve ter suporte a conexões esternas, pois isso é um fator determinante para se trabalhar com API's.

8.2. Conclusão

Conforme os resultados obtidos foi mostrado que é possível trabalhar com informação imperfeita nas consultas de BDD, o que será considerado agora como trabalho futuro é a possibilidade de implementar outras funções tradicionais que as API já possuem no *FuzzyScript* para que ele se torne uma ferramenta mais robusta e que possa ser uma solução completa para desenvolvimento de aplicações que evolvem tomadas de decisão a partir de dados pesquisados.

É possível expandir ainda mais o documento JSON Nebuloso podendo ter a possibilidade de fazer consultas com mais de um atributo nebuloso, assim aumentando a eficiência da API, pois estaremos deixando as consultas mais complexas, porém garantindo o retorno de dados com mais integridade.

Também está sendo desenvolvido um artigo abordando os principais focos e resultados dessa pesquisa. Esse artigo foi submetido à conferência DBKDA 2023 no dia 26 de novembro de 2022, e estamos esperançosos para ser aprovado e assim na conferência apresentar melhor os conceitos e aplicações dessa ferramenta. Caso seja aprovado o trabalho será publicado na IARIA Journal edição 2023, o artigo pode ser visualizado após a Referência Bibliográfica dessa dissertação. A API pode ser baixada gratuitamente e também ter seu código-fonte analisando para novas implementações em seu repositório no github (Nogueira, Leandro & Cura, Luiz (2022)).

8.3. Trabalhos Futuros

Como trabalho futuro está no planejamento a criação de uma interface humano-computador para interagir com a API, a fim de deixar o uso da API um pouco mais interativa e não deixa apenas que seu uso seja diretamente focado nos programadores. Deixando também outros tipos de nicho usufruírem da tecnologia.

Está em pauta também a realização de benchmark's com outros tipos de tecnologias e trabalhos para sabermos as vantagens e desvantagens, cobertura de dados e performance do FuzzyScript em comparação a outros trabalhos utilizados nessa dissertação.

Como próximo passo começaremos a analisar os cenários para patentear tanto a API quanto o formato de documento JSON nebuloso, porém não é descartado deixas essas estruturas públicas para poderem também receber atualizações e novas funcionalidades por parte da comunidade de desenvolvedores.

O desenvolvimento de novas publicações para divulgar melhor todo o método, a fim de expor melhor as contribuições que o FuzzyScript tem a oferecer.

Referências Bibliográficas

Bordogna, G., Pasi, G. & Lucarella, D. (1999). A Fuzzy Object-Oriented Data Model for Managing Vague and Uncertain Information, *International Journal of Intelligent Systems* **14**(7): 623–651.

Bourhis, P., Reutter, J. L., Suárez, F. & Vrgoč, D. (2017). JSON: Data Model, Query Languages and Schema Specification, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 123–135.

CakePHP, Website (2022). Cakephp 1.3 cookbook. [Online; accessed October 24, 2022].
URL: *https://book.cakephp.org/1.3/pt/The-Manual/Beginning-With-CakePHP/Understanding-Model-View-Controller.html*

Castelltort, A. & Laurent, A. (2014). Fuzzy Queries Over NoSQL Graph Databases: Perspectives for Extending the Cypher Language, *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer, pp. 384–395.

Chen, S.-M. & Jong, W.-T. (1997). Fuzzy Query Translation for Relational Database Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **27**(4): 714–721.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A. & Suciu, D. (1998). XML-QL: A query Language for XML.

Elmasri, R., Navathe, S. B., Pinheiro, M. G. et al. (2005). Sistemas de Banco de Dados.

Exist-DB, Website (2018). Exist-db. [Online; accessed October 27, 2021].
URL: *http://exist-db.org/exist/apps/homepage/resources/img/first-dev-steps.png*

Fosci, P. & Psaila, G. (2021a). J-CO, A Framework for Fuzzy Querying Collections of JSON Documents, *International Conference on Flexible Query Answering Systems*, Springer, pp. 142–153.

Fosci, P. & Psaila, G. (2021b). Towards Flexible Retrieval, Integration and Analysis of JSON Data Sets through Fuzzy Sets: A Case Study, *Information* **12**(7): 258.

Fredrick, E. T. & Radhamani, G. (2009). Fuzzy Logic Based XQuery Operations for Native XML Database Systems, *International Journal of Database Theory and Application* **2**(3): 13–20.

Gomide, F. A. C. & Gudwin, R. R. (1994). Modelagem, Controle, Sistemas e Lógica Fuzzy, *SBA controle & Automação* **4**(3): 97–115.

Guo, R. (2017). Mongodb's Javascript Fuzzer, *Queue* **15**(1): 38–56.

Jin, Y. & Veerappan, S. (2010). A Fuzzy XML Database System: Data Storage and Query Processing, *2010 IEEE International Conference on Information Reuse & Integration*, IEEE, pp. 318–321.

Kitchenham, B. (2004). Procedures for performing systematic reviews, *Keele, UK, Keele University* **33**(2004): 1–26.

Lennon, J. (2009). Introduction to CouchDB Views, *Beginning CouchDB*, Springer, pp. 107–123.

Lv, T., Yan, P. & He, W. (2018). Survey on json data modelling, *Journal of Physics: Conference Series*, Vol. 1069, IOP Publishing, p. 012101.

Ma, Z. M. (2007). A Literature Overview of Fuzzy Database Modeling, *Intelligent Databases: Technologies and Applications* pp. 167–196.

Ma, Z. M. & Yan, L. (2007a). Fuzzy XML Data Modeling With the UML and Relational Data Models, *Data & Knowledge Engineering* **63**(3): 972–996.

Ma, Z. M. & Yan, L. (2007b). Generalization of Strategies for Fuzzy Query Translation in Classical Relational Databases, *Information and Software Technology* **49**(2): 172–180.

Ma, Z. & Yan, L. (2016). Modeling Fuzzy Data With XML: A Survey, *Fuzzy Sets and Systems* **301**: 146–159.

Ma, Z., Zhang, W.-J. & Ma, W. (2000). Semantic Measure of Fuzzy Data in Extended Possibility-Based Fuzzy Relational Databases, *International Journal of Intelligent Systems* **15**(8): 705–716.

Marrara, S. & Pasi, G. (2016). Fuzzy Approaches to Flexible Querying in XML Retrieval, *International Journal of Computational Intelligence Systems* **9**(sup1): 95–103.

Nayak, A., Poriya, A. & Poojary, D. (2013). Type of NOSQL Databases and its Compari-

son With Relational Databases, *International Journal of Applied Information Systems* **5**(4): 16–19.

Nogueira, Leandro & Cura, Luiz (2022). Fuzzyscript api. [Online; accessed October 27, 2021].

URL: *https://github.com/aikid/fuzzyscript*

Nurseitov, N., Paulson, M., Reynolds, R. & Izurieta, C. (2009). Comparison of JSON and XML Data Interchange Formats: A Case Study., *Caine* **9**: 157–162.

Panić, G., Racković, M. & Škrbić, S. (2014). Fuzzy XML and Prioritized Fuzzy XQuery With Implementation, *Journal of Intelligent & Fuzzy Systems* **26**(1): 303–316.

Parsons, S. (1996). Current Approaches to Handling Imperfect Information in Data and Knowledge Bases, *IEEE Transactions on Knowledge and Data Engineering* **8**(3): 353–372.

Peng, D., Cao, L. & Xu, W. (2011). Using JSON for Data Exchanging in Web Service Applications, *Journal of Computational Information Systems* **7**(16): 5883–5890.

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M. & Vrgoč, D. (2016). Foundations of JSON schema, *Proceedings of the 25th International Conference on World Wide Web*, pp. 263–273.

Pivert, O., Slama, O., Smits, G. & Thion, V. (2016). SUGAR: A Graph Database Fuzzy Querying System, *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, IEEE, pp. 1–2.

Pivert, O., Thion, V., Jaudoin, H. & Smits, G. (2014). On a Fuzzy Algebra for Querying Graph Databases, *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, IEEE, pp. 748–755.

Ponsoni, Bruno, d. V. L. . (2019). An Neo4j Implementation for Designing Fuzzy Graph Databases, *Proceedings of the 23rd International Database Applications & Engineering Symposium*, pp. 1–6.

Prade, H. & Testemale, C. (1984). Generalizing Database Relational Algebra for the Treatment of Incomplete or Uncertain Information and Vague Queries, *Information sciences* **34**(2): 115–143.

Psaila, G. & Marrara, S. (2019). A First Step Towards a Fuzzy Framework for Analyzing Collections of Json Documents, *IADIS AC 2019*, pp. 19–28.

Raju, K. & Majumdar, A. K. (1988). Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems, *ACM Transactions on Database Systems (TODS)* **13**(2): 129–166.

Salminen, A. & Tompa, F. W. (2001). Requirements for XML Document Database Systems, *Proceedings of the 2001 ACM Symposium on Document Engineering*, pp. 85–94.

Shakhovska, N. (2017). The Method of Big Data Processing, *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, Vol. 1, IEEE, pp. 122–126.

Shukla, P. K., Darbari, M., Singh, V. K. & Tripathi, S. P. (2011). A Survey of Fuzzy Techniques in Object Oriented Databases, *International Journal of Scientific and Engineering Research* **2**(11): 1–11.

Silva, M. S. (2010). *JavaScript-Guia do Programador: Guia Completo das Funcionalidades de Linguagem JavaScript*, Novatec Editora.

Suter, R. (2012). MongoDB An Introduction and Performance Analysis, *Rapperswil* .

Tesoriero, C. (2013). *Getting Started With OrientDB*, Packt Publishing Ltd.

Tseng, C., Khamisy, W. & Vu, T. (2005). Universal Fuzzy System Representation With XML, *Computer Standards & Interfaces* **28**(2): 218–230.

Turowski, K. & Weng, U. (2002). Representing and Processing Fuzzy Information—an XML-Based Approach, *Knowledge-Based Systems* **15**(1-2): 67–75.

Ueng, P. S. & Škrbić, S. (2012). Implementing XQuery Fuzzy Extensions Using a Native XML Database, *2012 IEEE 13th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE, pp. 305–309.

Umano, M., Imada, T., Hatono, I. & Tamura, H. (1998). Fuzzy Object-Oriented Databases and Implementation of its SQL-Type Data Manipulation Language, *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36228)*, Vol. 2, IEEE, pp. 1344–1349.

Üstünkaya, E., Yazici, A. & George, R. (2007). Fuzzy Data Representation and Querying in XML Database, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **15**(supp01): 43–57.

XML path language (xpath) 3.1 (2017). [Online; accessed February 25, 2023].
URL: *https://www.w3.org/TR/xpath-31/*

XQuery 3.1: An XML query language (2017).
URL: *https://www.w3.org/TR/xquery-31/*

Yazici, A. & Koyuncu, M. (1997). Fuzzy Object-Oriented Database Modeling Coupled With Fuzzy Logic, *Fuzzy Sets and Systems* **89**(1): 1–26.

Zadeh, L. (1965). Fuzzy sets, *Information and Control* **8**(3): 338–353.
URL: *https://www.sciencedirect.com/science/article/pii/S001999586590241X*

Zadeh, L. A., Klir, G. J. & Yuan, B. (1996). *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers*, Vol. 6, World Scientific.

Zicari, R. (1990). Incomplete Information in Object-Oriented Databases, *ACM SIGMOD Record* **19**(3): 5–16.