UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Modelagem e incorporação de um banco de dados orientado a grafos ao chatbot LIA

Trabalho de Conclusão de Curso

Nívea Neyara Bomfim Melo

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Nívea Neyara Bomfim Melo

Modelagem e incorporação de um banco de dados orientado a grafos ao chatbot LIA

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Leonardo Nogueira Matos
Coorientador(a): Thiago Dias Bispo

São Cristóvão – Sergipe

2021

Dedico este trabalho aos meus pais, amigos e professores
que acreditaram e contribuíram nesta caminhada.

Resumo

Com os avanços da tecnologia, percebe-se um aumento no número de aplicações que atendem as diversas áreas da computação, desde sistemas de CRM (Customer Relationship Management) até redes que fazem uso da Internet das Coisas. Estas aplicações costumam necessitar de um sistema de banco de dados com a finalidade de guardar e recuperar as informações necessárias de forma rápida e segura. Apesar de todos os bancos existentes cumprirem este propósito, os vários modelos de bancos de dados se diferenciam por conta de fatores cujo grau de importância depende da aplicação. Estes fatores incluem, mas não se limitam, a quantidade de dados a serem armazenados, como eles devem estar organizados, e o tempo de leitura e escrita no banco. Levando em conta as aplicações do tipo *chatbot*, sabe-se que há uma busca contínua por métodos que façam com que o bot conduza conversas mais fluidas, seja ágil na tomada de decisões e preste um serviço de qualidade. Um dos mecanismos principais para atingir estes objetivos é agregar uma base de conhecimento de modo que o *chatbot* tenha acesso a todos os recursos necessários para tomar decisões autônomas e inteligentes. Assim, este trabalho se fundamenta na modelagem de um banco orientado a grafos que sirva como uma base de conhecimento para a assistente virtual LIA, criada para atender o ramo imobiliário. O primeiro passo consistiu em realizar uma revisão sistemática a fim de buscar trabalhos relacionados com o tema, ou seja, a integração de bancos orientados a grafos em *chatbots*. Em seguida, fez-se uma pesquisa de mercado para selecionar o banco orientado a grafos mais adequado para a LIA considerando vários critérios objetivos pré-estabelecidos. Por fim, analisou-se as vantagens na aplicação deste novo modelo de Banco de dados em comparação com a versão original do *chatbot*. Constatamos que o Grakn foi o mais adequado dentre todos os bancos de dados orientados a grafo analisados e que ele permitiu a modelagem de uma base de conhecimento mais rica e flexível que a versão atual do LIA.

Palavras-chave: Bancos Orientados a Grafos, Chatbot, Base de Conhecimento.

Abstract

Considering the advances in technology, there was an increase in the number of applications that serve the diverse areas of computing, from CRM (Customer Relationship Management) systems to networks that make use of the Internet of Things. These applications usually need a database system to save and retrieve the necessary information quickly and safely. Although all existing databases fulfill this purpose, the various database models differ due to factors whose degree of importance depends on the application. These factors include, but are not limited to, the amount of data to be stored, how this data should be organized, and the reading and writing time made by the database. Taking into account chatbot applications, it is known that there is a continuous search for methods that make the bot conduct more fluid conversations, be agile in decision-making and provide a quality service. One of the main mechanisms to achieve these objectives is to add a knowledge base so that the chatbot has access to all the resources necessary to make autonomous and intelligent decisions. Thus, this work is based on the modeling of a graph database that acts as a knowledge base for the virtual assistant LIA, which was created to serve the real estate industry. The first step consisted of conducting a systematic review to search for related works to this subject, that is, the integration of graph databases in chatbots. Then, market research has been made to select the most suitable graph database for the chatbot LIA considering several pre-established objective criteria. Finally, the advantages of applying this new database model were compared to the original version of the chatbot. It was found that Grakn was the most suitable among all the graph databases analyzed and that it allowed the modeling of a knowledge base richer and more flexible than the current version of the LIA.

Keywords: Graph Databases. Chatbot. Knowledge Base.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Sumário

1 Introdução

Na busca por uma melhor qualidade no atendimento ao cliente e com a intenção de alcançar metas cada vez maiores quanto à execução de um serviço ou entrega de um produto, uma empresa visa expandir seu leque de estratégias incorporando tecnologias que são tendências no mercado. Nos setores de atendimento ao cliente, por exemplo, percebe-se que as tecnologias existentes procuram aproximar humano e máquina, uma vez que é neste departamento que o cliente busca informações a respeito do negócio da empresa. Nesse contexto, destacam-se os assistentes virtuais e os *chatbots*, sendo que este último grupo representa o artefato utilizado neste trabalhado.

De acordo com Khan e Das (2018), *chatbot* é um programa de computador que consegue analisar as mensagens codificadas em linguagem natural de um usuário e retornar respostas coerentes. Uma outra definição dada por Brandtzaeg e Følstad (2017) introduz *chatbot* como a representação de uma mudança potencial na maneira como as pessoas interagem com dados e serviços online. Esta tecnologia se apresenta como uma solução capaz de automatizar conversas, evitar que atendentes lidem com processos repetitivos e proporcionar um atendimento ininterrupto com alta disponibilidade.

Empresas das mais diversas áreas (financeiro, educacional, entretenimento etc.) possuem seus próprios *chatbots* a fim de servir as necessidades dos seus usuários finais. O *chatbot* LIA, por exemplo, foi desenvolvido por Silva (2019) e Santos (2019b) como Trabalho de Conclusão do Curso de Ciência da Computação da Universidade Federal de Sergipe. Esta assistente virtual visa servir empresas do mercado imobiliário com o propósito de acelerar o processo de aquisição e venda de imóveis por parte de clientes e construtoras, respectivamente. Deste modo, a construtora consegue interagir com mais clientes e estes não têm que esperar muito tempo para conseguir qualquer tipo de informação relacionado ao domínio de imóveis.

Com a criação da empresa Fechô [1] em 2020, os criadores alteraram o nome da assistente

[1] Fechô <https://fecho.app/>

para “Fê”. Portanto, este trabalho a referenciará por este nome de agora em diante. Antes de citar os objetivos deste trabalho, criou-se o tópico a seguir que traz uma visão geral de como está a Fê atualmente, o que levou a escolha das ferramentas propostas no seu desenvolvimento e a motivação para o emprego de um banco de dados orientado a grafos.

1.1 Definição do Problema

1.1.1 Características do MVP da Fê

O Mínimo Produto Viável (MVP) da Fê foi implementado a partir do framework Rasa [2] e integrado ao banco de dados não relacional (NoSQL), MongoDB [3]. O Rasa é um *framework* destinado para o desenvolvimento de *chatbots* contextuais, ou seja, que utilizam recursos de inteligência artificial para se comunicar com o usuário e não dependem de regras pré-definidas. De acordo com Silva (2019), o MongoDB foi escolhido por se tratar de uma estrutura de dados NoSQL orientada a documentos, *open source* e distribuída, ou seja, os dados são dispersos e tratados em diferentes máquinas a fim de seguir o conceito de replicação.

Após analisar 21 estudos que discutiam sobre o armazenamento e a recuperação de dados para o contexto do chatbot, Silva (2019) selecionou a arquitetura proposta em Karve et al. (2018) em que o autor apresenta as etapas que um agente de conversação executa durante uma consulta ao banco de dados. As etapas consistem em pré-processamento, identificação de intenção e tratamento de contexto e geração de resposta. O Rasa visa simplificar estes processos posto que ele já implementa essa arquitetura que auxilia na identificação das palavras-chave nas mensagens trocadas entre o usuário e o chatbot.

Para extrair estas informações, o módulo de Processamento de Linguagem Natural (PLN), denominado Rasa NLU, treina o chatbot gerando um modelo que, então, será capaz de identificar a intenção da mensagem do usuário e extrair as entidades definidas para o domínio de imóveis. Além do Rasa NLU, existe um segundo módulo denominado Rasa Core que lida como diálogo entre o usuário e a aplicação. Dentro dele, existe um submódulo de ações em que é definido o que o chatbot deve fazer quando recebe uma mensagem. Assim, ele pode enviar uma mensagem ao usuário, acessar a Interface de Programação de Aplicação (API) de um serviço, ou até mesmo consultar um banco de dados.

1.1.2 Por que usar um banco orientado a grafos em chatbots ?

Onlim [4] é uma das várias empresas de software que fornece soluções para criar *chatbots* e gerenciar bases de conhecimento, sendo que estas consistem de uma biblioteca de informações sobre um serviço ou produto. Um tipo específico de base de conhecimento é o grafo de

[2] Rasa <https://www.rasa.com/>
[3] MongoDB <https://www.mongodb.com/>
[4] Onlim <https://onlim.com/en/>

conhecimento, que representa os dados através de relacionamentos entre entidades (como pessoas, lugares ou coisas). A empresa afirma que grafos de conhecimento oferecem vários benefícios ao *chatbot* quando se trata de otimizar a conversação com o usuário:

- Aprimora e estende o entendimento de linguagem natural;
- Possibilita responder a perguntas específicas e complexas;
- Pode ser conectado a outros grafos de conhecimento formando novos relacionamentos e ampliando conhecimento;

Outra empresa do mesmo ramo da Onlim é a Kore.ai [5], que é especializada no desenvolvimento de assistentes virtuais. Ela destaca a importância dos grafos de conhecimento na elaboração de FAQs em que é possível definir termos e criar uma hierarquia entre estes de modo que a entrada do usuário é associada aos termos exatos. Agora que tem-se as razões para usar grafos de conhecimento combinados com *chatbots*, faz-se necessário buscar motivos para construir grafos de conhecimentos com bancos orientados a grafos.

O advento e popularização de aplicações, como o facebook e o twitter, exigiu a busca por tecnologias que lidassem com um grande volume de dados (*big data*). Em comparação com bancos de dados relacionais, os NoSQL não possuem as mesmas propriedades ACID (atomicidade, consistência, isolamento e durabilidade) ou as mesmas verificações de integridade de dados, mas lidam com *big data* de forma eficiente (BHOGAL; CHOKSI, 2015). Por conta disso, essas e outras grandes empresas adicionaram tecnologias NoSQL aos seus ecossistemas, cada uma de acordo com as necessidades dos serviços prestados. Dentre os tipos de bancos de dados NoSQL, têm-se os modelos denominados colunar, orientado a grafos, orientado a documentos e de chave-valor, cada qual projetado para propósitos específicos.

Resource Description Framework (RDF) consiste em um padrão para modelagem de dados que fornece uma maneira de interpretar e fazer suposições sobre os dados. Este padrão é mencionado no trabalho de (Tang et al., 2020), que propõe a elaboração de um grafo de conhecimento com base na modelagem orientada a grafos, e compara a performance de armazenamento e consultas entre o formato RDF e o orientado a grafos considerando um caso de uso específico, gerenciamento de equipamentos elétricos. Apesar de que o RDF ser bastante utilizado na representação de conhecimento (do inglês, Knowledge Representation - KR), o autor apresenta em seu trabalho que pesquisadores começaram a considerar o uso de bancos NoSQL para solucionar os problemas de armazenamento e consulta emergentes da escalabilidade de dados com RDF.

Dadas as quatro categorias de bancos NoSQL e que o RDF representa os dados como uma tripla equivalente a uma aresta associadas a um rótulo, (Tang et al., 2020) adiciona que o

[5] Kore.ai <https://kore.ai/>

grafo de conhecimento se porta naturalmente como uma estrutura em grafos. Desta forma, o modelo orientado a grafos consegue representar a semântica como no modelo RDF, além de armazenar os dados no formato de grafos e conter algoritmos que possibilitam projetar o esquema e consultas baseadas em inferência sobre os dados. O principal ponto que o autor apresenta como vantagem no uso de um banco orientado a grafos como grafo de conhecimento é o suporte a escalabilidade em larga escala. O banco pode percorrer nós e arestas em paralelo e a quantidade de dados armazenados não afeta a velocidade da operação.

Em uma publicação de (Zhang, 2017), este afirma que os bancos orientados a grafos são especialmente úteis quando lidam com dados complexos, ricos em relacionamentos. Ele discute o uso de bancos orientados a grafos como o componente central em sistemas de gestão de conhecimento por facilitar no planejamento, desenvolvimento e implementação destes tipos de sistemas. Ao comparar um simples caso de uso representado em modelos relacional e orientado a grafos, o autor ilustra como o último modelo reduz consideravelmente a complexidade do problema perante relações do tipo muitos-para-muitos.

Em uma publicação feita pela comunidade do Rasa (BERGMANN, 2019), é mencionada as vantagens em criar uma base de conhecimento para *chatbots* a depender do domínio aplicado a estes. Quando um usuário interage com um *chatbot*, espera-se que ele não só faça perguntas sobre uma entidade específica (ex. um imóvel), como também queira comparações entre entidades, ou informações de uma entidade já mencionada na conversa. Essa publicação, juntamente com os outros argumentos aqui discutidos, motivaram a adoção de um banco de dados orientados a grafos como base de conhecimento e na execução deste trabalho.

O propósito da Fê é sugerir imóveis (a partir de um algoritmo de recomendação, por exemplo) com base em dados fornecidos pelo usuário, como o número de quartos, tipo, preço e localização do imóvel. No entanto, é importante que a Fê não só realize esse tipo de recomendação, mas que também esteja preparada para responder outras perguntas relacionadas ao domínio de imóveis. Nessa situação, aplicar um banco orientado a grafos para servir de base de conhecimento da Fê pode resultar em conversas mais ricas e na execução mais eficiente de tarefas que dependam de relações entre entidades diferentes, que é o ponto forte deste modelo.

1.2 Objetivos

1.2.1 Objetivo Geral

Modelar o contexto da Fê em um banco de dados orientado a grafos, persistir os dados de imóveis e usuários, ampliar o conhecimento do chatbot sobre o domínio da aplicação e aprimorar o poder de interpretação das mensagens do usuário por parte do chatbot.

1.2.2 Objetivos Específicos

- Realizar uma revisão sistemática a fim de analisar trabalhos que usaram bancos de dados orientados a grafos no ramo de inteligência artificial;
- Realizar uma pesquisa de mercado para avaliar os banco de dados orientados a grafos existentes;
- Modelar o banco com base na ontologia de domínio do *chatbot*;
- Integrar o banco de dados selecionado à Fê;
- Discutir as vantagens da adoção do banco orientado a grafos quanto ao nível de expressividade da linguagem e quanto à integração com o Rasa.

1.3 Metodologia

Antes de executar a etapa prática deste trabalho, foi necessário fazer uma revisão sistemática para reunir trabalhos que empregaram bancos orientados a grafos em aplicações do campo de inteligência artificial. Isso permitiu verificar as causas da adoção deste modelo e como ele foi implantado no projeto em questão. Em seguida, fez-se uma busca de mercado selecionando os bancos orientados a grafos mais populares e analisando as peculiaridades de cada um. Como resultado, o Grakn foi escolhido e iniciou-se a etapa de modelagem que levou em conta o domínio de imóveis considerado pela Fê.

A documentação do Grakn foi o recurso principal para criar o esquema e importar os dados. O fato de a versão em produção da Fê estar executando com o MongoDB e, assim, já possuir esquemas para imóveis e usuários definidos, facilitou bastante a elaboração do modelo criado com o Grakn. Para esta fase foi importante esboçar novos diálogos, entre o usuário e a Fê, que fariam uso da base de conhecimento gerada com o banco. Após conectar o banco à Fê, o chatbot foi treinado com estes diálogos para gerar um novo modelo que visa identificar a intenção do usuário. Os termos importantes da conversa são extraídos e tratados no submódulo de ações do chatbot em conjunto com as consultas feitas no banco que retornam as informações desejadas.

Ao final, o banco MongoDB será comparado com o Grakn usando como critérios a representação dos dados e o nível de expressividade das consultas. Vale destacar que essa comparação não é apenas entre dois modelos de bancos de dados, mas considerando o contexto de um *chatbot*, especificamente, a Fê.

1.4 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em três capítulos, que são:

- Capítulo 1 - Introdução:

  Corresponde ao presente capítulo, que compreende a metodologia e os objetivos deste trabalho.

- Capítulo 2 - Fundamentação Teórica:

  Contém os conceitos centrais que guiam este trabalho com base em publicações relevantes para a área de estudo em questão.

- Capítulo 3 - Trabalhos Relacionados:

  Aborda os trabalhos encontrados a partir da revisão sistemática, bem como a pesquisa de mercado feita sobre o objeto de estudo deste trabalho.

- Capítulo 4 - Desenvolvimento:

  Descreve a execução do trabalho, desde a modelagem do banco de dados até a integração com a Fê.

- Capítulo 5 - Resultados:

  Compara o MongoDB e o Grakn quanto à expressividade de suas linguagens de consulta e apresenta o resultado da integração do Grakn com a Fê.

- Capítulo 6 - Conclusão e Trabalhos Futuros:

  Discute os resultados encontrados e cita os trabalhos futuros considerando a abordagem deste trabalho.

2

Fundamentação Teórica

Esse capítulo apresenta os conceitos básicos necessários para o entendimento deste trabalho. As seções seguintes abordam conceitos sobre processamento de linguagem natural, bancos de dados relacionais e não relacionais, e grafos de conhecimento.

2.1 Processamento de Linguagem Natural

De acordo com Chowdhury (2003), processamento de linguagem natural (PLN) é uma área de pesquisa que explora como os computadores podem ser usados para reconhecimento de fala ou análise de texto em linguagem natural. O termo "linguagem natural" refere-se a liguagem usada para se comunicar no dia-a-dia, como português, inglês ou mandarim (BARBOSA et al., 2017). O ato de fazer com que uma máquina entenda a linguagem humana não é uma tarefa simples devido às particularidades de cada linguagem, como a existência de ambiguidades e anáforas.

Além de analisar o texto como uma mera sequência de caracteres, o PLN considera a estrutura hierárquica da linguagem, fazendo uso de conceitos linguísticos como classes de palavras (substantivo, verbo, adjetivo, etc.), também denominado *Part-Of-Speech*. Assim, os sistemas de PLN podem ser utilizados em aplicações como tradutores automáticos, reconhecimento ótico de caracteres (OCR), sumarização automática, sistemas de diálogos, entre outros (BARBOSA et al., 2017).

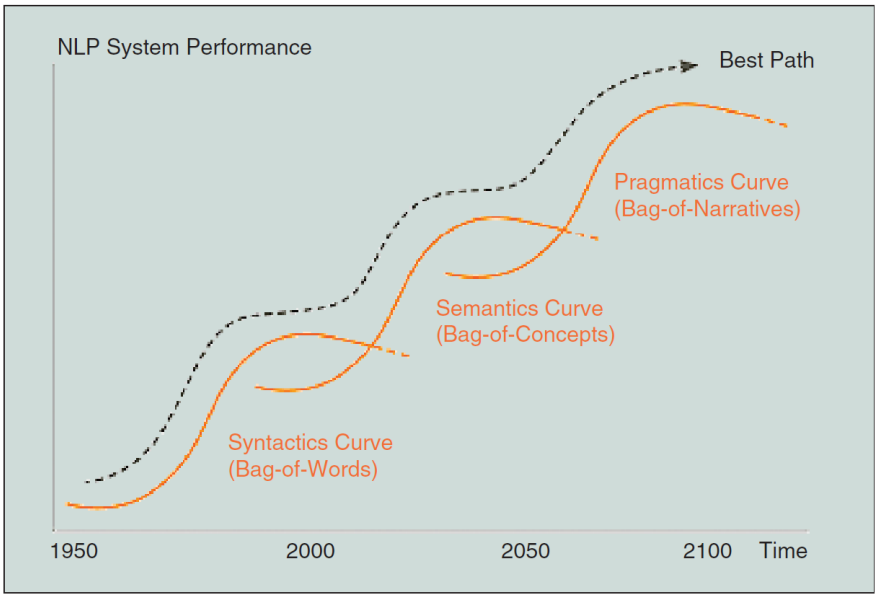
Apesar de o PLN ter cumprido seu propósito de forma satisfatória até o momento, analisando textos com base em fatores como palavras-chave, pontuações e frequência de ocorrência de palavras, os algoritmos comuns ao PLN estão se tornando menos eficazes. Segundo Cambria e White (2014), isso se deve ao crescimento do chamado conteúdo gerado pelo usuário (do inglês, User-generated content - UGC) e à eclosão de fenômenos falsos como *spam* social e *trolls* na internet. O primeiro diz respeito à disposição de conteúdo de forma espontânea sobre

determinado assunto, o que é comumente observado em blogs ou redes sociais. Já o segundo se trata da divulgação em massa de mensagens sem propósito, e do ato de certos usuários da web em iniciar discussões polêmicas para provocar outras pessoas.

Por conta disso, é fundamental que os sistemas PLN também tenham acesso a uma quantidade considerável de conhecimento sobre o mundo e o domínio em questão. Isso faz com que exista uma menor dependência quanto ao uso de técnicas que realizam uma análise sintática das palavras e inicia-se uma fase de constante análise semântica (CAMBRIA; WHITE, 2014). A Figura 1 apresenta um gráfico com a evolução do PLN como a interseção de três curvas sobrepostas, sendo que cada curva representa um estágio do PLN (sintático, semântico e pragmático). Essas curvas representam o percurso do PLN que eventualmente o levará ao ponto de não apenas processar, mas também entender a linguagem natural (Natural Language Understanding - NLU).

A curva sintática do PLN normalmente está relacionada à análise gramatical e corresponde a tarefas como extração e recuperação de informação, categorização, modelagem de tópicos, etc. A curva semântica leva em consideração o significado agregado ao texto e envolve a questão de ensinar à máquina conhecimentos de senso comum, equivalente ao que as pessoas aprendem no estágios iniciais da vida, e conhecimento comum, equivalente ao que as pessoas aprendem no dia-a-dia. Esta curva, portanto, requer que a máquina tenha acesso a conhecimentos gerais sobre o mundo ou domínio em questão. Por fim, a curva pragmática se trata de uma hipótese sobre o futuro do PLN em que se alcança a habilidade de processar informações com precisão assim como ocorre na comunicação humana.

Figura 1 – Previsão da evolução do PLN sobre três diferentes curvas



Fonte: (CAMBRIA; WHITE, 2014)

2.2 Bancos de Dados

De acordo com RICHARDSON (2015), "the topic of data storage is one that does not need to be well understood until something goes wrong (data disappears) or something goes really right (too many customers)". Com isso, ele revela o descuido normalmente observado durante a escolha de um banco de dados, que por vezes é considerada como uma ferramenta que apenas armazena dados e os devolve quando requisitado. Compreender os recursos e garantias que os bancos de dados a disposição oferecem é um fator importante que pode evitar futuros descontentamentos por parte de usuários por conta de problemas como lentidão ou indisponibilidade.

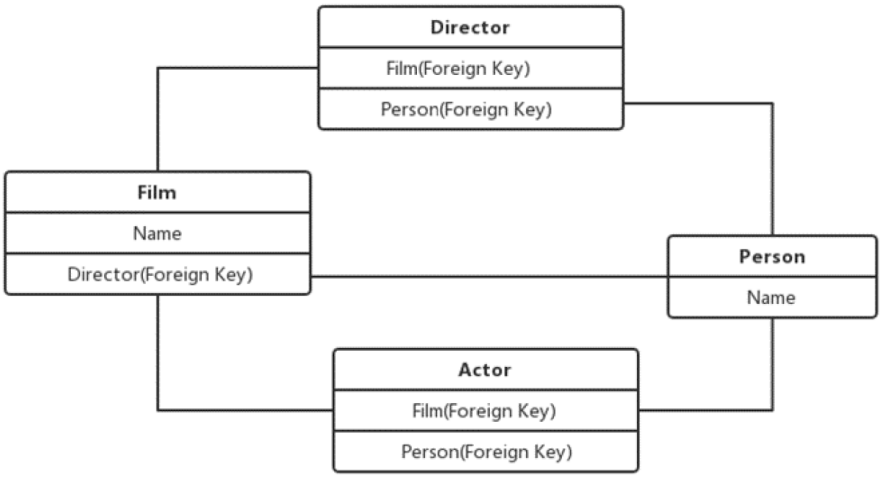
Os tópicos a seguir buscam levar o leitor a refletir quanto a importância de analisar os bancos de dados durante o desenvolvimento de uma aplicação e, também, apresentar as vantagens e desvantagens de cada tipo de banco de dados, destacando os bancos relacionais e os orientados a grafos.

2.2.1 Banco de Dados Relacional

Em termos de adoção, o bancos relacionais ainda são bastante utilizados por oferecerem flexibilidade e modelos bem definidos. Esse tipo de banco permite que partes constituintes de um todo sejam modificadas independentemente através do conceito de tabelas e criando relações entre elas. Outras vantagens são que a linguagem de consulta SQL é bem conhecida entre os desenvolvedores e os bancos relacionais oferecem um esquema que mantém a consistência dos dados, entre outros benefícios providos pela propriedade ACID (Atomicidade, Consistëncia, Isolamento e Durabilidade). Uma grande desvantagem desse tipo de banco, no entanto, é que ele possui limitações quanto ao tamanho de dados que se pretende armazenar ou recuperar. Se for uma quantidade de dados consideravelmente grande, a performance do sistema pode ser afetada, ou até mesmo a complexidade para lidar com os dados pode ser alta (RICHARDSON, 2015).

Um problema de alta complexidade normalmente envolve várias entidades e relações. Para cada entidade cria-se uma tabela e existem relações entre entidades em que pode ser necessário a criação de novas tabelas. Estas tabelas cuja função é associar uma entidade a outra geralmente não armazenam nenhum outro dado além das chaves estrangeiras. Este aumento considerável do número de tabelas por conta da necessidade de representar relações pode afetar a execução do banco de dados tornando-o ineficiente (YAO, 2018). A Figura 2 apresenta uma parte do modelo de um banco de dados relacional para persistir dados relacionados a filmes. Observe que para definir as relações que representam pessoas que podem dirigir um filme e pessoas que podem atuar em um filme, foi preciso criar novas tabelas e estas armazenam um número pequeno de informações.

Figura 2 – Representação de um modelo em um banco de dados relacional



Fonte: (YAO, 2018)

2.2.2 Banco de Dados Não Relacional

Os bancos de dados não relacionais ganharam foco com o *Big Data*, termo que descreve o volume de informações que cresce continuamente em alta velocidade e que necessitam de técnicas ou tecnologias apropriadas para lidar com um volume grande e complexo de dados (SEGAL, 2019). Para processar essa vasta quantidade de dados, portanto, é necessário o uso de bancos de dados distribuídos com alta flexibilidade e alta performance (LI; MANOHARAN, 2013).

Conforme RICHARDSON (2015), o grande feito dos bancos não relacionais é que eles executam operações mais rápido em relação aos bancos relacionais pelo fato de não incluírem transações e tabelas. Mas a maioria destes bancos, como o MongoDB, adotaram alguns recursos SQL como consultas, filtros e agregações. Mais tarde alguns bancos NoSQL passaram a suportar certas transações das propriedades ACID a fim de corrigir alguns problemas de consistência dos dados.

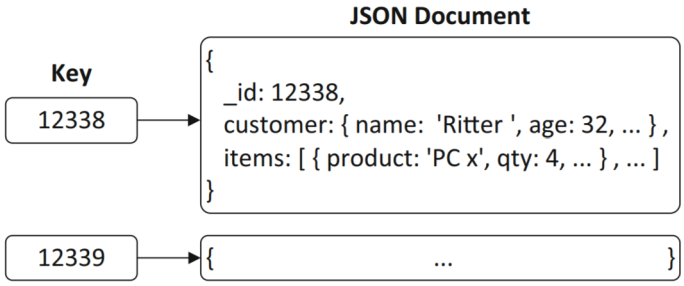
Com isso, percebe-se que os bancos NoSQL se proliferaram nos últimos anos destrichando-se em quatro modelos que buscam atender diferentes nichos tecnológicos da melhor maneira possível. Logo, tem-se os bancos do tipo chave-valor, orientado a colunas, orientado a documentos, e orientado a grafos. O tópico a seguir descreve as características dos modelos orientado a documentos e orientado a grafos, uma vez que o primeiro foi empregado na primeira versão da assistente virtual Fê e o último é o objeto de estudo deste trabalho.

2.2.2.1 Modelo Orientado a Documentos

Esse modelo normalmente armazena dados organizados no formato JSON com atributos que consistem de um conjunto de pares chave-valor, como ilustrado na Figura 3. Uma característica deste modelo é o fato de não ser necessário criar um esquema explicitamente através de uma

linguagem de definição de dados (DDL, do inglês Data Definition Language) e, portanto, a estrutura dos dados armazenados é descrita na lógica da aplicação. Essa característica define os bancos orientados a documentos como *schemaless* (GESSERT et al., 2017).

Figura 3 – Representação de um objeto em um banco orientado a documentos



Fonte: (GESSERT et al., 2017)

Como apresentado na página oficial do MongoDB [1], as vantagens de um banco orientado a documentos são:

1. Possui um esquema flexível, ou seja, os campos podem variar de documento para documento e a estrutura pode ser modificada a qualquer momento sem a necessidade de realizar migração dos dados;
2. Não há necessidade de decompor os dados como os bancos relacionais fazem com o uso de tabelas, evitando assim a execução de JOINs que tendem a encarecer o custo da leitura dos dados;
3. É projetado para ser distribuído e fácil de escalar quando deve-se lidar com um grande número de dados.

Vale destacar, contudo, que se houver decomposição dos dados em um banco orientado a documentos, este não permite realizar JOINs entre vários registros. Apenas em versões mais recentes, o MongoDB apresentou uma solução para isto com a operação de *lookup*, equivalente a um *left join*, no método de agregação. Ainda assim, seu uso é recomendado apenas quando necessário pelo fato de que fazer a ligação de registros é uma operação custosa, e não deve ser considerado como uma justificativa para sempre manter os dados normalizados em um banco orientado a documentos.

Outro ponto é que não há integridade referencial no modelo orientado a documentos, ou seja, é possível incluir links entre documentos de modo semelhante à atribuição de chaves estrangeiras em um banco relacional, mas não há mecanismos que garantam, por exemplo, deleção em cascada. Para lidar com situações como essa, o desenvolvedor deve tomar providências, também, a partir da camada de aplicação.

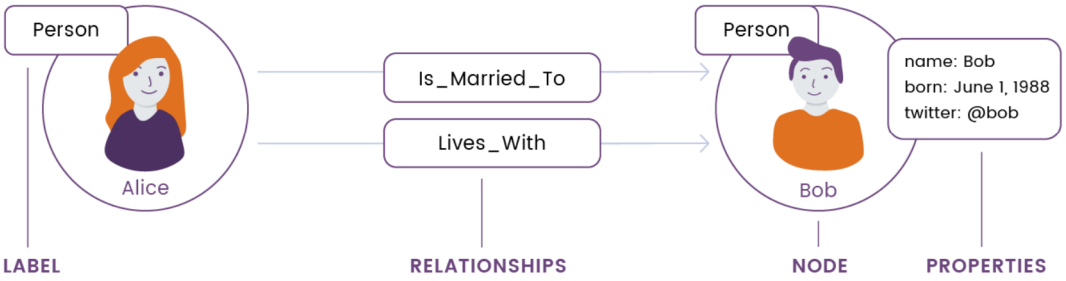
[1] https://www.mongodb.com/document-databases

2.2.2.2 Modelo Orientado a Grafos

De modo geral, o banco orientado a grafos oferece um modelo de representação natural e mais próximo de como os dados se comportam no mundo real. Isso não só permite que a representação faça sentido para todos os envolvidos, como também aproveita ao máximo as interações complexas entre os dados. Aliás, a representação do mundo através de grafos não é algo incomum, uma vez que eles podem ser vistos em árvores genealógicas, mapas de metrôs e até na própria internet (COX, 2017).

Os dados são organizados como nós, relações e propriedades. Os nós são as entidades do grafo e possuem um número qualquer de atributos, organizados em pares chave-valor, que são denomidados propriedades. As relações proveem conexões direcionais, nomeadas e semanticamente relevantes entre duas entidades. Tanto os nós como as relações possuem rótulos (ou *labels*) para representar os diferentes papeis das entidades no domínio em questão e para auxiliar na identificação da relação entre elas, respectivamente. A Figura 4 apresenta um exemplo indicando cada um desses elementos.

Figura 4 – Exemplo representando o modelo orientado a grafos



Fonte: (GLEB; VLAD, 2018)

Cox (2017) também acrescenta que este tipo de banco geralmente não possui um esquema - permitindo a flexibilidade que se observa em bancos orientados a documentos -, mas representa relacionamentos de maneira semelhante à de um banco relacional. Se comparado a outros modelos de bancos de dados, os grafos apresentam os dados de forma mais elegante sem envolver estratégias como JOINs entre tabelas, ou documentos aninhados para alcançar o mesmo nível de expressividade. Ao aplicar a teoria dos grafos, é possível verificar conexões entre os dados que não são óbvias em outros tipos de bancos.

2.3 Grafo de Conhecimento

Segundo Yan et al. (2018), o papel de um grafo de conhecimento (do inglês, *knowledge graph*) é extrair dados importantes de uma grande quantidade de informação e apresentar as relações complexas entre esses dados de uma forma compreensível e organizada. Obviamente, um grafo de conhecimento representa uma coleção de informações a partir de um grafo. Logo, os

dados são representados a partir de entidades (nós) e relações (arestas) que ajudam a lidar com modelos grandes e complexos de informações de modo que é possível percorrer as conexões existentes e descobrir como partes remotas de um domínio se relacionam.

Outra característica de um grafo de conhecimento é o fato de ele ser semântico, isto é, ele é autodescritivo ao fornecer um único local para encontrar os dados e entender o que eles representam como um todo. Para Stichbury (2017), além de fornecer a descrição de conceitos e propriedades de um domínio de maneira compreensível tanto para o usuário como para o computador, um grafo permite fazer consultas em um estilo muito mais próximo da linguagem natural. Ou seja, o significado dos dados é expresso de acordo com os nomes das entidades e das relações, considerando que os interessados no domínio especificado estão familiarizados com esses termos. Isso permite buscas mais eficientes e diminui a lacuna de comunicação entre os consumidores e os provedores dos dados.

Empresas das mais diversas áreas utilizam grafos de conhecimento para armazenar e analisar um grande volume de dados. Como exemplificado por Yan et al. (2018), empresas de e-commerce, como o Walmart, fazem recomendações e busca de produtos a partir dessa tecnologia. Na área da medicina, vários medicamentos são desenvolvidos graças às análises feitas em cima de informações coletadas sobre doenças ajudando, assim, na evolução de diagnósticos médicos.

À princípio, Nickel et al. (2015) relata que é possível utilizar grafos de conhecimentos para mapear fatos. Supondo que pretende-se representar a informação "Leonardo DiCaprio é um ator que interpretou o personagem Jack Dawson no filme de romance Titanic" a partir de um grafo de conhecimento, o autor propõe conectar os fatos como relacionamentos binários no formato SPO (sujeito, predicado, objeto). O sujeito e o objeto são entidades, enquanto que o predicado define a relação entre eles. Assim, do ponto de vista do grafo, os fatos são apresentados como na Tabela 1.

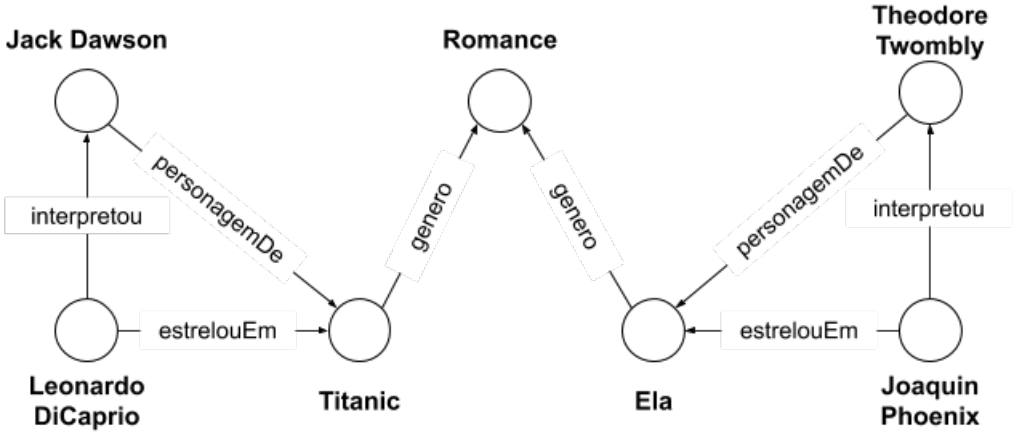
Tabela 1 – Representação de um fato como uma tripla SPO.

Sujeito	**Predicado**	**Objeto**
(Leonardo DiCaprio	profissao	Ator)
(Leonardo DiCaprio	estrelouEm	Titanic)
(Leonardo DiCaprio	interpretou	Jack Dawson)
(Jack Dawson	personagemDe	Titanic)
(Titanic	genero	Romance)

Fonte: Adaptado de (NICKEL et al., 2015)

(NICKEL et al., 2015) afirma ainda que é possível combinar cada relação no formato SPO para criar um multigrafo em que os nós representam as entidades (sujeitos e objetos), e arestas direcionadas representam as relações. Além disso, as relações podem ser diferenciadas associando rótulos a cada aresta, resultando na Figura 5.

Figura 5 – Grafo de conhecimento



Fonte: Adaptado de (NICKEL et al., 2015)

3

Trabalhos Relacionados

Este capítulo apresenta os resultados obtidos a partir da realização de revisões sistemáticas e pesquisa de mercado. Por um lado, a revisão sistemática propõe reunir informações acerca do tema proposto neste trabalho a partir do estudo de artigos publicados em bases científicas. Por outro, a pesquisa de mercado contribui para a tomada de decisão quanto ao produto que deve ser empregado neste trabalho, que neste caso trata-se de um banco de dados orientado a grafos, visto aqueles que estão à disposição no mercado.

3.1 Revisão Sistemática

A revisão sistemática, segundo Sampaio e Mancini (2007), é uma forma de pesquisa realizada sobre determinado tema com base em estudos primários. Um ponto importante, referido por Cordeiro et al. (2007), é que uma boa revisão sistemática depende da formulação adequada de uma pergunta uma vez que esta define as estratégias necessárias para identificar tanto os estudos que serão incluídos, como os dados que devem ser coletados de cada estudo.

Para este trabalho, decidiu-se formular mais de uma pergunta acerca do tema para identificar os termos mais importantes e, então, criar uma string de busca genérica. As perguntas estabelecidas foram:

Q1. Quais bancos orientados a grafos já foram integrados a *chatbots*?

Q2. Quantos bancos orientados a grafos são *open source*?

Q3. Quais são as características destes bancos?

Q4. Como o uso de bancos orientados a grafos contribuem no aprendizado de *chatbots*?

Os termos selecionados para aplicar à string de busca genérica estão na Tabela 2. Eles foram usados em seis bases científicas acessadas através do Portal Periódicos CAPES [1]. As bases são: ACM [2], Compendex [3], IEEE [4], Science Direct [5], Scopus [6] e Web of Science [7].

Tabela 2 – Termos considerados para a criação da string de busca.

Termo	**Significado**	**Sinônimo**
Graph Database	Banco de Dados Orientado a Grafos	-
Chatbot	Chatbot	Chatterbot

Fonte: O Autor.

As strings de busca aplicadas às bases de dados (apresentadas na Tabela 3) possuem pequenas diferenças quanto a sintaxe, pois este é um fator característico de cada base. Observe que as strings de busca constituem apenas de dois termos e não existe filtro relacionado ao período de publicação. Ainda assim, como pode ser visto na Tabela 4, os resultados obtidos compreendem publicações dos últimos três anos, considerando a escrita deste trabalho.

Tabela 3 – Bases de dados e strings de busca.

Base de Dados	**String de Busca**
ACM	[Full Text: "graph database"] AND [[Full Text: chatbot] OR [Full Text: chatterbot]]
Compendex	(chatbot OR chatterbot) AND (graph database)
IEEE	((graph database) AND (chatbot OR chatterbot))
Science Direct	(“chatbot” OR “chatterbot”) AND "graph database"
Scopus	(TITLE-ABS-KEY("chatbot") AND ("graph database")) AND (LIMIT-TO(SUBJAREA , "COMP"))
Web of Science	TS=("graph database"AND chatbot)

Fonte: O Autor.

Após a fase de busca, fez-se uma leitura dos títulos e resumos das publicações resultantes. Isso permitiu descartar tanto as ocorrências de artigos duplicados, como os artigos cujo tema não era relevante para este trabalho. Para este último caso, considerou-se os critérios de inclusão (I) e exclusão (E) a seguir:

I1. O artigo apresenta técnicas de como criar uma base de conhecimento a partir de um banco orientado a grafos a fim de aprimorar uma conversa com um *chatbot*.

[1] Portal Periódicos CAPES <http://www.periodicos.capes.gov.br/>
[2] ACM Digital Library <https://dl-acm-org.ez20.periodicos.capes.gov.br/>
[3] Compendex <https://www-engineeringvillage-com.ez20.periodicos.capes.gov.br/>
[4] IEEE Xplore <https://ieeexplore-ieee-org.ez20.periodicos.capes.gov.br/Xplore/home.jsp>
[5] Science Direct <https://www-sciencedirect.ez20.periodicos.capes.gov.br/>
[6] Scopus <https://www-scopus.ez20.periodicos.capes.gov.br/>
[7] Web of Science <http://apps-webofknowledge.ez20.periodicos.capes.gov.br/>

Tabela 4 – Quantidade de publicações encontradas por base de dados.

Base de Dados	**Quantidade de Publicações**	**Período**
ACM	5 publicações	2018 à 2020
Compendex	8 publicações	2017 à 2020
IEEE	2 publicações	2019
Science Direct	4 publicações	2019 à 2020
Scopus	6 publicações	2017 à 2019

Fonte: O Autor.

I2. O artigo aborda a integração de bancos orientados a grafos com aplicações como assistentes virtuais.

E3. O artigo foca no desenvolvimento de *frameworks*, não relacionados a *chatbots* , que farão uso do banco de dados orientados a grafos.

E4. O artigo não detalha como ocorre a integração entre o *chatbot* e o banco de dados orientados a grafos.

Enfim, o último passo da revisão sistemática compreendeu a leitura completa de 6 artigos que não puderam ser avaliados apenas com a leitura dos títulos e resumos. Destes, apenas 2 abordavam ferramentas e técnicas úteis para este trabalho. Os artigos e seus respectivos autores podem ser visualizados na Tabela 5. Os próximos parágrafos sumarizam o objetivo de cada um desses trabalhos, revelando os métodos e tecnologias utilizados e apontando ideias que podem ser agregadas ao escopo deste trabalho.

Tabela 5 – Artigos escolhidos ao fim da revisão sistemática.

Título	**Autor(es)**
HHH: An Online Medical Chatbot System based on Knowledge Graph and Hierarchical Bi-Directional Attention	Bao, Ni e Liu
Speak to your Software Visualization - Exploring Component-based Software Architectures in Augmented Reality with a Conversational Interface	Seipel et al.

Fonte: O Autor.

O estudo de Bao, Ni e Liu (2020) propõe o desenvolvimento de um *chatbot* e de um sistema online de perguntas e respostas. Este conjunto, nomeado HHH pelos autores, busca facilitar o acesso a informações pertinentes aos cuidados com a saúde. O HHH possui dois módulos: o HBAM (Hierarchical BiLSTM Attention Model), modelo neural que lida com a compreensão do texto, e um grafo de conhecimento, que gerencia os dados médicos. O primeiro se trata de um módulo exposto pelos próprios autores, enquanto que o último foi desenvolvido a

partir do banco orientado a grafos Neo4j [8]. Com a finalidade de selecionar as respostas adequadas do banco de dados, os autores definiram um processo que consiste de cinco passos: 1. Coletar a entrada do usuário; 2. Extrair as entidades da entrada; 3. Extrair as intenções do usuário; 4. Acessar o banco para selecionar a resposta de acordo com as entidades e as intenções; 5. Retornar a resposta.

O trabalho desenvolvido por Seipel et al. (2019) explora a possibilidade de usar diferentes métodos de interação (gestos, fala...) em programas baseados em Realidade Aumentada (RA). O autor realizou os experimentos com o dispositivo HoloLens da Microsoft de forma que o software de RA foi integrado a componentes de conversação, derivados do PLN. A arquitetura adotada pelo autor dispõe de cinco componentes que seguem o padrão Model-View-Controller (MVC):

a) Dispositivo: corresponde à interface do usuário;

b) Monitor Contextual: monitora as ações do usuário (gestos e fala);

c) Repositório: lida com a persistência dos dados;

d) Controlador: considerada a peça central do sistema, uma vez que regula o repositório e o dispositivo com base nas ações do usuário, estabelecendo o fluxo do diálogo em seguida;

e) Serviços de Comunicação: recebem a entrada do usuário, extraem as entidades e intenções, consultam o banco utilizando estes elementos, e retornam os resultados para o controlador.

Um detalhe discutido por Seipel et al. (2019) é que sua arquitetura utiliza um serviço de geração automática de texto (GAT), referido no artigo como *Natural Language Generation* (NLG), para construir sentenças em linguagem natural a partir dos resultados obtidos com as consultas feitas ao banco. Assim como no trabalho anterior, o autor adota o Neo4j como o banco de dados, mas a compreensão do texto é feita através da biblioteca Rasa NLU.

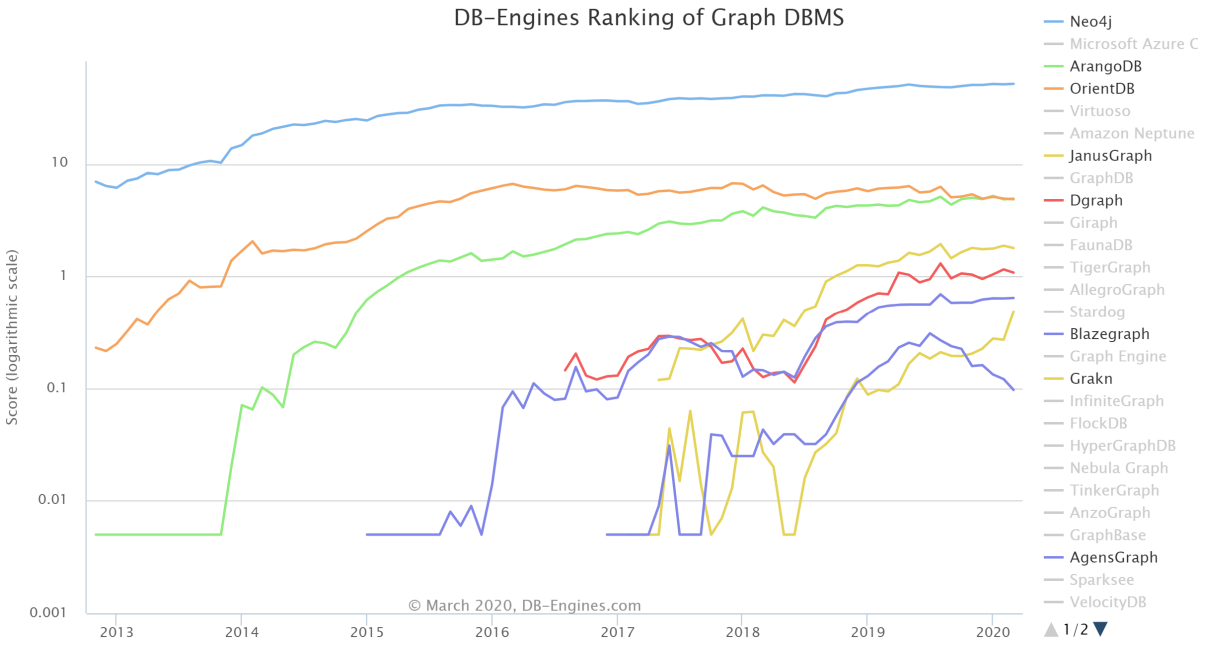
3.2 Pesquisa de Mercado

Os trabalhos selecionados na revisão sistemática apresentam um modelo de arquitetura semelhante em que existe um componente para persistir os dados e outro que faz a compreensão textual. Para o presente trabalho, não é necessário fazer um levantamento das tecnologias que podem exercer o papel deste último componente. Afinal, ele já foi definido e justificado por Silva (2019) e Santos (2019b) em seus respectivos estudos que originaram o *chatbot* Fê. Por outro lado, a pesquisa de mercado sobre os bancos orientados a grafos existentes é fundamental, uma vez que não houve um estudo aprofundado quanto a isso nos trabalhos de Bao, Ni e Liu (2020) e Seipel et al. (2019), ou nos trabalhos relacionados à Fê.

[8] Neo4j <https://neo4j.com/>

O primeiro passo da pesquisa de mercado compreendeu na seleção dos bancos de dados orientados a grafos mais populares de acordo com o portal DB-Engines [9], que se trata de uma iniciativa da companhia austríaca Solid IT com o propósito de coletar e apresentar informações relacionadas a sistemas de gerenciamento de banco de dados. A figura 6 apresenta o ranking dos bancos orientados a grafos open source mais populares até Março de 2020.

Figura 6 – Bancos orientados a grafos open source mais populares



Fonte: (DB-ENGINES, 2020)

Segundo o DB-Engines, a popularidade dos bancos apresentados no gráfico é mensurada segundo os seguintes parâmetros:

- Número de menções do sistema nas plataformas de pesquisa Google [10], Bing[11] e Yandex[12];
- Grau de interesse sobre o sistema de acordo com dados do Google Trends[13];
- Frequência com que ocorrem discussões sobre o sistema, sendo que este indicador consiste do número de questões e de usuários interessados em sites de perguntas e respostas bem conhecidos, como o Stack Overflow[14] e o DBA Stack Exchange[15];

9 DB-Engines <https://db-engines.com/>
10 Google <https://www.google.com/>
11 Bing <https://www.bing.com/>
12 Yandex <https://yandex.com/>
13 Google Trends <https://trends.google.com/trends/>
14 Stack Overflow <https://stackoverflow.com/>
15 DBA Stack Exchange <https://dba.stackexchange.com/>

- Número de oferta de empregos em que o sistema é mencionado nos sites de empregos Indeed[16] e Simply Hired[17];
- Número de perfis em redes sociais corporativas em que o sistema é mencionado. Nesse caso, considera-se os sites LinkedIn[18] e Upwork[19];
- Relevância nas redes sociais. No Twitter[20], por exemplo, conta-se o número de tweets em que o sistema é mencionado.

Inicialmente, o gráfico listava 12 opções como bancos open source orientados a grafos, mas após visualizar a página oficial destas ferramentas, 4 delas foram desconsideradas. Estas são mencionadas no tópico 3.2.1, enquanto que os tópicos 3.2.2 a 3.2.9 apresentam uma análise mais aprofundada dos 8 bancos que aparecem no gráfico, em ordem. Esta análise auxiliou na escolha do banco que mais se adequa ao propósito deste trabalho e levou à conclusão apresentada no tópico 3.2.10.

3.2.1 Ferramentas desconsideradas

Dentre as ferramentas que foram apresentadas inicialmente pelo DB-Engines, observou-se que o FlockDB [21] foi descontinuado em 2017. O Nebula Graph [22] entrou no mercado em 2019 e ainda há pouca informação sobre ele para considerá-lo na pesquisa de mercado. Quanto às demais ferramentas:

Apache TinkerPop [23]

Não se trata de um banco de dados, mas de um framework que define a linguagem de consulta Gremlin, que é a base da maioria dos bancos orientados a grafos. Portanto, desenvolvedores podem criar seus próprios bancos a partir deste framework. TinkerGraph é uma versão mais leve do TinkerPop de modo que, se um desenvolvedor fornece um banco compatível com o TinkerPop, um usuário pode realizar operações sobre este banco com Gremlin da mesma forma que o faria utilizando a linguagem do próprio banco. Exemplos de sistemas que utilizam o TinkerPop são Neo4j, Grakn, JanusGraph e OrientDB.

HyperGraphDB [24]

O site não possui nenhum anúncio de descontinuação do projeto, mas o respectivo repositório não é atualizado há dois anos da data de publicação deste trabalho. Além disso, a última versão deste banco (HyperGraph 1.3) foi lançado em 2015, o que leva a supor que o projeto não recebe mais atualizações recorrentes.

16 Indeed <https://www.indeed.com/>
17 Simply Hired <https://www.simplyhired.com/>
18 LinkedIn <https://www.linkedin.com/>
19 Upwork <https://www.upwork.com/>
20 Twitter <https://twitter.com/>
21 FlockDB <https://github.com/twitter-archive/flockdb>
22 Nebula Graph <https://nebula-graph.io/>

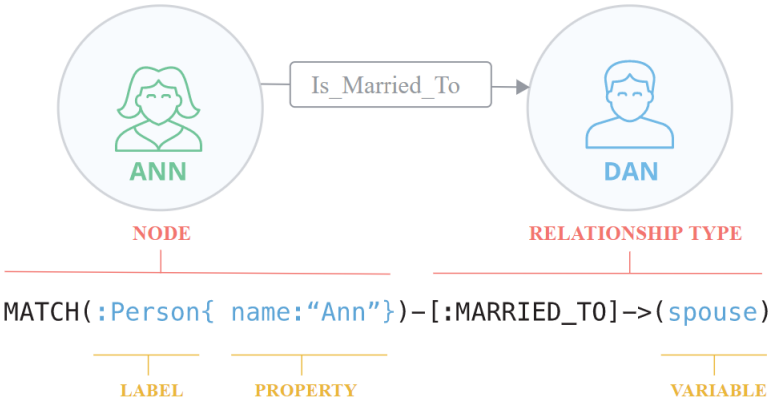
3.2.2 Neo4j

Neo4j é um banco implementado em Java e considerado o banco orientado a grafos mais popular mundialmente. Ele foi criado em 2007 e, atualmente, possui três versões: a Community Edition, que é a versão open source, a Enterprise Edition, que é qualificada para empresas que prezam pelos critérios de disponibilidade e escalabilidade, e a versão denominada Neo4j Aura, que propõe a ideia de banco de dados orientado a grafos como serviço, permitindo a execução do banco em ambientes Cloud.

Alguns cenários de aplicação do Neo4j são inteligência artificial, detecção de fraude, recomendações em tempo real e redes sociais. A linguagem de consulta, denominada Cypher, é simples e fácil de aprender. Com ela, é possível descrever nós, relações e propriedades de forma que fica claro para o usuário o que a consulta realizada significa e como os elementos estão representados no grafo (ver Figura 7). Outras vantagens do Neo4j são:

- A existência de drivers para várias linguagens de programação, como Java, JavaScript, Python, entre outras;
- O fato de que ele segue as propriedades ACID, que garantem a integridade das informações armazenadas;
- É flexível, ou seja, não há a necessidade de seguir um modelo pré-definido e pode-se adicionar ou remover propriedades quando requisitado;
- Possui uma ferramenta de visualização gráfica que torna a tarefa de elaborar consultas mais prática, além de servir de ferramenta para análise e interpretação dos dados.

Figura 7 – Exemplo de uma consulta usando Cypher.



Fonte: https://neo4j.com/cypher-graph-query-language

Apesar de a escalabilidade ser um forte deste banco, esta característica está presente apenas na versão Enterprise Edition. A Community Edition tem uma limitação quanto ao número

de nós no grafo, não permite o particionamento horizontal dos dados (*sharding*), além de outras restrições. Cabe destacar também que ela só pode ser usada dentro de organizações ou dispositivos pessoais. Caso o usuário tenha interesse em tenha a intenção de aplicá-la em um negócio ou para propósitos educacionais, ele deve pagar pela licença da versão Enterprise.

3.2.3 ArangoDB

ArangoDB [25] foi criado em 2012 e possui duas versões: Community Edition e Enterprise Edition. A primeira é open source e a segunda possui recursos extras, como escalabilidade horizontal, replicação de transações e segurança adicional. Alguns casos de uso do ArangoDB são Internet das Coisas, recomendação, criptografia e aprendizado de máquina.

Este banco é caracterizado como multi-modelo por armazenar e gerenciar vários tipos de dados (documentos, grafos ou chave-valor). Isso permite o armazenamento poliglota sem que o usuário se preocupe em ter que lidar com várias tecnologias diferentes. O acesso aos dados, independente do tipo, ocorre a partir da linguagem de consulta AQL (do inglês, ArangoDB Query Language). Ela possui uma estrutura parecida com a SQL (ver Figura 8), por isso um usuário com experiência nesta linguagem não deve ter dificuldade em aprender AQL.

Figura 8 – Comparação entre as linguagens SQL e AQL.

SQL:

```
INSERT INTO users (name, gender)
  VALUES ("John Doe", "m");
```

```
SELECT *
  FROM users;
```

AQL:

```
INSERT { name: "John Doe", gender: "m" }
  INTO users
```

```
FOR user IN users
  RETURN user
```

Fonte: https://www.arangodb.com/why-arangodb/sql-aql-comparison

Outras vantagens do ArangoDB são:

- A existência de drivers para várias linguagens de programação, incluindo Java, JavaScript e Python;
- Possui uma interface web em que é possível visualizar os grafos e realizar consultas;
- Possui uma boa documentação;
- Segue as propriedades ACID;
- É flexível, ou seja, não é necessário definir os atributos de um documento de antemão. Diz-se documento, pois é dessa forma que os vértices e arestas são definidos neste banco.

[25] ArangoDB <https://www.arangodb.com/>

3.2.4 OrientDB

OrientDB [26] é um banco implementado em Java, lançado em 2011 e também é considerado multi-modelo. Ele possui duas versões: Community Edition e Enterprise Edition. A diferença é que a primeira não inclui os recursos de escalabilidade horizontal, tolerância a falhas, clusterização e replicação. Os casos de uso compreendem detecção de fraude, recomendação e análise forense.

Este banco é multi-modelo e permite o armazenamento de dados do tipo documento, chave-valor e grafo. Com relação a linguagem de consulta, o OrientDB decidiu por extender o poder do SQL de modo que esta agregasse os conceitos de grafos. O modelo do OrientDB define os vértices e arestas através de classes genéricas denominadas V e E, respectivamente. O Código 1 exemplifica o uso do SQL do OrientDB para a criação de um esquema modelado em um grafo.

Código 1 – Criando um esquema e realizando consultas com o SQL do OrientDB

```
1  // Criando o esquema do grafo
2  CREATE CLASS Pessoa EXTENDS V
3  CREATE CLASS Carro EXTENDS V
4  CREATE CLASS Possui EXTENDS E
5
6  // Criando dois vertices e uma aresta
7  CREATE VERTEX Pessoa SET name = 'Ana'
8  CREATE VERTEX Carro SET name = 'Ferrari'
9  CREATE EDGE Possui FROM (SELECT FROM Pessoa) TO (SELECT FROM Carro)
10
11 // Consultando qual carro pertence a Ana
12 SELECT name FROM (SELECT EXPAND( OUT('Possui') )) FROM Person WHERE
       name = 'Ana')
```

Outros pontos que devem ser considerados em relação ao OrientDB são:

- Possui drivers para várias linguagens de programação, como Java, JavaScript, Python e PHP;
- Possui uma boa documentação e uma comunidade ativa;
- Existe uma interface web para visualizar e interagir com os dados do banco;
- Ele permite modelar o banco a partir de um esquema pré-definido, mas isso não é um requisito.

3.2.5 JanusGraph

JanusGraph [27] é um banco open source que foi lançado em 2017. Ele surgiu da bifurcação do repositório do TitanDB, um banco orientado a grafos que foi adquirido pela empresa DataStax.

[26] OrientDB <https://orientdb.org/>
[27] JanusGraph <https://janusgraph.org/>

O JanusGraph pode ser integrado a projetos de terceiros e, por isso, diz-se que ele é customizável. Ele pode utilizar o Apache Cassandra ou o HBase como back-end para armazenar os dados, e o Elastic Search ou Apache Solr para realizar pesquisa de texto completo.

Como mencionado na seção 3.2.1, o JanusGraph é um dos bancos que tem como base o Apache TinkerPop. Já que este oferece suporte nativo para a linguagem Gremlin, ela acabou sendo oficialmente utilizada no JanusGraph. Os benefícios em utilizar este banco incluem o fato de que ele é escalável, possui suporte nativo a modelos de grafo integrados ao TinkerPop e ele está sob a licença Apache.

3.2.6 Dgraph

Dgraph [28] foi lançado em 2017 e possui duas versões: a open source, sob licença Apache, e a Enterprise. Esta última inclui os recursos de backup, listas de controle de acesso e criptografia dos dados no banco. No repositório do Dgraph, existe uma tabela (ver Figura 9) que apresenta os recursos do banco e faz uma comparação com os bancos Neo4j e JanusGraph. A partir dela, verifica-se que há vantagens do Dgraph em relação ao Neo4j Community Edition quanto ao particionamento horizontal e replicação dos dados. Já em relação ao JanusGraph, percebe-se que o Dgraph se destaca por possuir suporte nativo para realizar consultas que envolvem pesquisa de texto completo, expressões regulares ou busca por geolocalização.

A linguagem de consulta adotada pelo Dgraph, denominada "GraphQL+-", foi inspirada na linguagem GraphQL criada pelo Facebook. No GraphQL+-, as consultas são compostas de blocos aninhados, sendo que a raiz seleciona os nós em que os filtros serão aplicados. No exemplo da figura 10, a raíz possui o nome da função que caracteriza a consulta (*people*), que serve apenas para identificar os resultados associados a ela na resposta, e os filtros que serão aplicados aos nós (*has(name)*). Os filtros são compostos de funções e, no caso desse exemplo, a função *has* visa buscar os nós que tem a o atributo *name*. Os atributos *name* e *age* no corpo da consulta indicam os campos que devem ser retornados no resultado. Para auxiliar na visualização dos resultados, o Dgraph disponibiliza uma interface web que é configurada durante a instalação do banco.

Com relação ao desempenho do banco, existe uma ferramenta chamada Jepsen [29] cujo propósito é testar a performance de sistemas distribuídos a fim de verificar se eles violam alguma das propriedades de consistência que afirmam satisfazer. Para executar os testes, no entanto, é necessário que os interessados contatem os responsáveis pelo Jepsen e contratem o serviço. Os autores do Dgraph fizeram essa solicitação durante a versão 1.0.2 do banco e, apesar de que 23 problemas foram reconhecidos, todos foram corrigidos até a versão atual (2.0). Um relatório com a lista dos problemas corrigidos está disponível no site do Jepsen [30], mas os responsáveis pelo Dgraph também mantiveram a lista visível no repositório para ciência da comunidade.

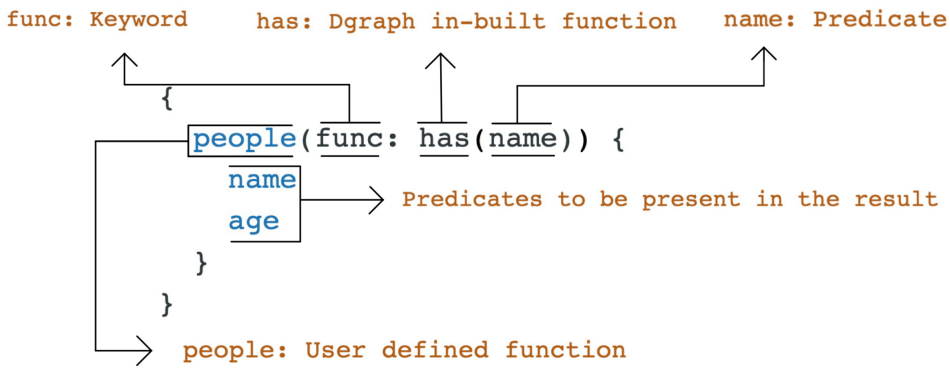
[28] Dgraph <https://dgraph.io/>
[29] Jepsen <https://jepsen.io/>
[30] Relato dos testes feitos no Dgraph <http://jepsen.io/analyses/dgraph-1-0-2>

Figura 9 – Comparação entre os bancos Dgraph, Neo4j e JanusGraph.

Features	Dgraph	Neo4j	Janus Graph
Architecture	Sharded and Distributed	Single server (+ replicas in enterprise)	Layer on top of other distributed DBs
Replication	Consistent	None in community edition (only available in enterprise)	Via underlying DB
Data movement for shard rebalancing	Automatic	Not applicable (all data lies on each server)	Via underlying DB
Language	GraphQL inspired	Cypher, Gremlin	Gremlin
Protocols	Grpc / HTTP + JSON / RDF	Bolt + Cypher	Websocket / HTTP
Transactions	Distributed ACID transactions	Single server ACID transactions	Not typically ACID
Full Text Search	Native support	Native support	Via External Indexing System
Regular Expressions	Native support	Native support	Via External Indexing System
Geo Search	Native support	External support only	Via External Indexing System
License	Apache 2.0	GPL v3	Apache 2.0

Fonte: https://github.com/dgraph-io/dgraph

Figura 10 – Estrutura da linguagem GraphQL+-



Fonte: https://docs.dgraph.io/master/tutorial-1/

3.2.7 Blazegraph

Blazegraph [31] é um banco open source, escrito em Java, escalável e de alta performance que está no mercado desde 2006. Os grafos são representados a partir do modelo Resource Description Framework (RDF), que fornece uma forma padrão de descrever e manipular os dados. Os vértices, suas propriedades e as arestas de um grafo são definidos a partir das chamadas

[31] Blazegraph <https://blazegraph.com/>

instruções RDF.

Um estudo feito por Lissandrini, Brugnara e Velegrakis (2018) propôs avaliar a performance de seis bancos de dados, incluindo o Blazegraph, quando expostos a uma grande quantidade de dados. Essa avaliação considerou operações que envolvem carregar os dados no banco, percorrer os dados no grafo, e as operações básicas de inserção, leitura, atualização e remoção. Como resultado, o Blazegraph foi o banco com o pior rendimento, sendo a demora na execução das consultas o problema principal.

3.2.8 Grakn

O Grakn [32] foi lançado em 2016 com duas versões: Grakn Core e Grakn KGMS (do inglês, *Knowledge Graph Management System*). A principal diferença entre as duas versões é que a primeira é open source, não foi projetada para escalar e não possui um recurso de controle de acesso ou autenticação de usuário. Enquanto isso, a segunda versão inclui recursos como autenticação de usuários, backup e recuperação, ferramentas para migrar dados, suporte profissional e ferramentas para monitorar a performance do banco.

Os casos de uso do Grakn incluem aprendizado de máquina, mineração de texto, biomedicina, descoberta de medicamentos e medicina de precisão. A linguagem de consulta utilizada por este banco é o Graql, que representa um domínio de forma similar ao modelo Entidade-Relacionamento (ER), ou seja, indicando entidades, relacionamentos, atributos e os tipos destes atributos. O Graql se diferencia de linguagens como a Cypher, por exemplo, por evidenciar a semântica dos dados e possuir um aspecto mais ontológico. O Cógido 2 exemplifica uma consulta utilizando as linguagens Cypher e Graql.

Código 2 – Coparação entre Cypher e Graql

```
1 // Cypher
2 MATCH (c:City { name: "Aracaju" })
3 RETURN c
4
5 // Graql
6 match $c isa city, has name "Aracaju";
7 get $c;
```

Além de definir entidades, relacionamentos e atributos, é possível definir regras (*rules*) ao elaborar um esquema, atribuir papéis (*roles*) às entidades e aplicar o conceito do tipo hierarquia (*type hierarchies*). Na Figura 11a, observa-se que há duas entidades definidas no esquema, *person* e *company*, cujos papéis são *employee* e *employer*, respectivamente. Esses papéis são utilizados na declaraçao da relação *employment*, garantindo que as entidades envolvidas nesta relação devem ter esses papéis, especificamente, para que ela exista. Por fim, a Figura 11b apresenta o conceito de hierarquia em que objetos podem herdar o tipo, atributos e papéis de seu ancestral.

[32] Grakn.AI <https://grakn.ai/>

Figura 11 – Representação das propriedades *roles* e *hierarchies* no Grakn.

```
# Entity-Relationship

define

person sub entity,
  has name,
  plays employee;

company sub entity,
  has name,
  plays employer;

employment sub relation,
  relates employee,
  relates employer;

name sub attribute,
  datatype string;

commit
```

(a) Aplicação de *roles* no Grakn.

```
# Type Hierarchies

define

person sub entity,
  has first-name,
  has last-name;

student sub person;
undergrad sub student;
postgrad sub student;

teacher sub person;
supervisor sub teacher;
professor sub teacher;

commit
```

(b) Aplicação de *hierarchies* no Grakn.

Fonte: Página inicial do Grakn.

Assim, as vantagens do Grakn são:

- Existência de drivers para Python, Java e Node.js;
- Possui boa documentação e uma comunidade ativa;
- Possui alto poder semântico devido à sua linguagem Graql;
- Possui uma ferramenta de visualização gráfica denominada Grakn Workbase;
- É um hipergrafo, ou seja, permite representar relacionamentos que englobam mais de duas entidades.

3.2.9 AgensGraph

O AgensGraph [33] foi lançado em 2016 pela empresa Bitnine e possui duas versões: Enterprise Edition e Community Edition. A diferença está no fato de que a primeira versão oferece ferramentas de monitoramento, otimização de memória, alta disponibilidade e suporte profissional contínuo. Alguns de casos de uso do banco AgensGraph envolvem recomendação, análise e gerenciamento de falhas de dispositivos em uma rede, plataformas da área de saúde que lidam com o acompanhamento de pacientes e técnicas para identificar ameaças cibernéticas.

[33] AgensGraph <https://bitnine.net/agensgraph/>

Este banco é caracterizado como multi-modelo, permitindo o gerenciamento de vários tipos de dados (relacionais, documentos e do tipo chave-valor). A compatibilidade do AgensGraph com dados relacionais deriva da integração deste com o PostgreSQL e, assim, o usuário pode utilizar tanto SQL como Cypher para acessar dados que estão organizados em tabelas e gráficos, respectivamente. Há casos, inclusive, em que é admissível combinar as duas linguagens em uma única consulta. Outras vantagens do AgensGraph são:

- Há drivers para Pyhton, Java, Node.js e GoLang, sendo que o suporte para estes dois últimos não é oficial;
- Segue as propriedades ACID;
- Possui uma ferramenta de visualização intuitiva chamada AgensBrowser, que permite visualizar os grafos, interagir com os dados e realizar consultas utilizando Cypher e SQL.

3.2.10 Considerações Finais

Para escolher um banco dentre os apresentados nos tópicos anteriores, considerou-se as características definidas na Tabela 6 como fatores determinantes para a escolha do banco que melhor se encaixa no propósito deste trabalho, que é integrar um banco de dados orientado a grafos a um *chatbot*. A característica C2, em específico, foi considerada pelo fato de o *chatbot* Fê ter sido implementado em Python e, consequentemente, o banco será configurado neste ambiente.

Tabela 6 – Características consideradas na pesquisa de mercado.

Sigla	**Característica**
C1	Linguagem com alto poder semântico
C2	Suporte para linguagem Python
C3	Escalonamento horizontal (*sharding*)
C4	Boa documentação e comunidade ativa
C5	Os dados podem ser armazenados sem estrutura prévia (schema-free)

Fonte: O Autor.

A Tabela 7 apresenta as características que cada banco satisfaz. O banco Blazegraph não foi considerado nessa etapa por conta de seu mau desempenho apresentado no trabalho de Lissandrini, Brugnara e Velegrakis (2018), ponto discutido no tópico 3.2.7. Os bancos Neo4J, Dgraph e Grakn alcançaram os melhores resultados e, assim, foi necessário considerar critérios mais específicos do projeto:

- Representação dos dados:

  Diferentemente do Neo4J e do GraphQL, o Grakn é um banco com a estrutura de um hipergrafo, ou seja, os relacionamentos não se limitam apenas a um par de entidades,

mas a um conjunto arbitrário de entidades. Isso favorece a representação de dados mais complexos e identificação de relacionamentos entre entidades dispersas.

- Capacidade de inferência:

  Não existe uma ferramenta ou método embutidos no Neo4J ou Dgraph que permitam a inferência de dados. Eles utilizam sistemas como GraphScale e Apollo Federation, respectivamente, para alcançar este objetivo. O Grakn, por outro lado, permite a criação de papéis em que é possível manter a coerência entre entidades e relações, ou até mesmo entre relações em si.

- Semântica da linguagem:

  Em comparação a linguagem GraphQL+- e a Cypher, vale ressaltar que o Grakn segue uma estrutura mais ontológica no que se refere à interpretação da linguagem Graql, além de que ela é tanto uma linguagem de manipulação de dados (DML) como de definição de dados (DDL). A Cypher tem estas duas últimas propriedades, com uma estrutura que lembra o SQL. Já o GraphQL+- tem como referência o GraphQL, que é uma linguagem simples com a estrutura de uma árvore, mas que não representa consultas complexas tão bem quanto as concorrentes.

Por fim, as três últimas análises levaram à conclusão de que o Grakn oferece métodos adicionais que tendem a facilitar a interpretação do domínio (regras, papéis, hierarquias e inferência), e que se encaixa muito bem com uma base de conhecimento para a assistente virtual Fê. Logo, o Grakn foi o banco escolhido para dar seguimento ao presente trabalho.

Tabela 7 – Comparação entre os bancos selecionados na pesquisa de mercado.

Banco	**C1**	**C2**	**C3**	**C4**	**C5**
Neo4j Community Edition	X	X		X	X
ArangoDB Community Edition		X		X	X
OrientDB Community Edition		X		X	X
JanusGraph					X
Dgraph		X	X	X	X
Grakn Core	X	X	X	X	
AgensGraph Community Edition	X	X			X

Fonte: O Autor.

4

Desenvolvimento

Este capítulo descreve os passos seguidos durante o aprimoramento da assistente virtual Fê. Eles consistem da modelagem do banco de dados orientado a grafos, geração de um esquema com a linguagem Graql (linguagem oficial do Grakn), integração do banco à Fê e análise dos resultados.

4.1 Criação de um modelo

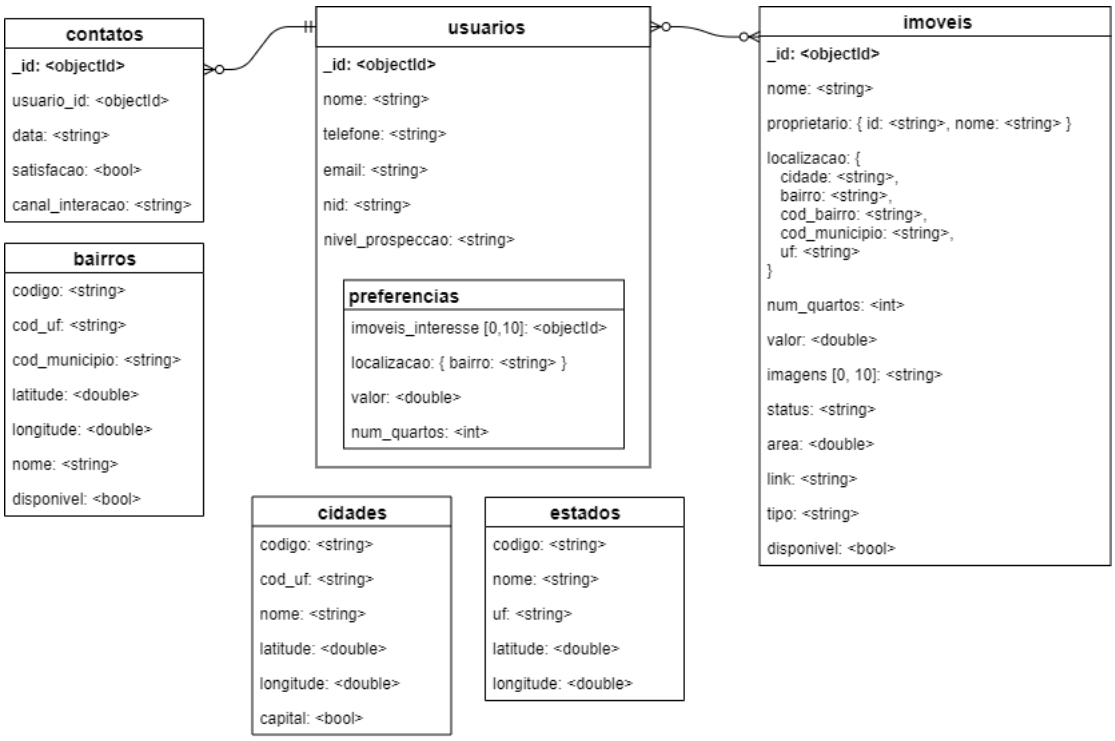
4.1.1 Interpretação do modelo orientado a documentos

Para melhor apresentar as mudanças ocorridas no *chatbot* com a mudança do modelo do banco, este tópico tem como objetivo expor como a Fê foi modelada com o MongoDB antes deste trabalho, justificando algumas decisões tomadas que visavam facilitar a busca e análise de informações no contexto da Fê. A figura 12 representa o esquema criado para o MongoDB.

O objetivo da assistente Fê é recomendar imóveis levando em conta as preferências do usuário, sendo estas representadas pelo número de quartos, localização do imóvel e o valor que o interessado pretende investir. Sendo assim, a coleção de usuários armazena os dados pessoais e suas preferências. A coleção de contatos armazena a data de cada contato entre o usuário e a Fê, o canal pelo qual o usuário fez a comunicação (ex. web ou whatsapp) e a satisfação do usuário finalizado aquele contato. Por fim, a coleção de imóveis armazena as características do imóvel e os dados do proprietário e da localização.

Considerando este modelo, nota-se que algumas decisões foram tomadas sobre os campos "preferência", "proprietário" e "localização". Por exemplo, a relação entre usuários e preferências é de um para muitos, e uma estratégia alternativa ao aninhamento de documentos é a referenciação. Ou seja, também seria possível criar uma coleção à parte para as preferências e referenciar o usuário nesta coleção. No entanto, aninhar as preferências à coleção de usuários é a opção ideal

Figura 12 – Domínio de imóveis representado no modelo do MongoDB.



Fonte: O Autor

uma vez que usuários costumam ter poucas preferências e mantê-los em um único documento evita a necessidade de executar mais de uma consulta para recuperar as informações mistas.

A disposição dos campos “proprietários” e “localização” também contribui para otimizar as consultas feitas sobre os dados. O módulo de recomendação seleciona os imóveis cujas características se assemelham às preferências citadas na conversa com a Fê. Ainda que se tenha um documento com várias informações do imóvel, a recomendação não precisa executar múltiplas consultas, além das operações realizadas no próprio algoritmo de recomendação, a fim de associar os imóveis ao proprietário e à localização. Em contrapartida, estes campos podem se repetir em vários documentos uma vez que uma construtora, por exemplo, pode ter centenas de imóveis na plataforma, bem como vários imóveis podem se situar no mesmo bairro.

4.1.2 Criação de um modelo para o Grakn

Como observado no tópico 3.2.8, o Grakn organiza seu modelo a partir de entidades, atributos e relações. A chave para modelar um conjunto de dados em Grakn é criar um esquema que se assemelhe a concepção dos dados no mundo real em termos de coisas e as interações entre elas. A referência para a criação do modelo da Fê foi o exemplo visto em (GRAKN, 2020), em que é apresentado os passos para definir um esquema cujos conjuntos de dados representam as chamadas telefônicas entre uma empresa e seus clientes.

O primeiro passo na criação do modelo em grafos foi definir os conjuntos de dados

de modo que ficasse claro quais informações são importantes para o fluxo da conversa entre a assistente virtual e o usuário, e quais permitem recuperar respostas para perguntas como:

- Quanto vale um imóvel no bairro Farolândia?
- Qual o grau de interesse no Monticello?
- Qual o bairro mais procurado em Aracaju?

O próximo passo para a construção do grafo de conhecimento foi determinar os atributos, as entidades e as relações entre estas. Os dois primeiros foram deduzidos do esquema do modelo orientado a documentos, mas as relações surgiram da ontologia do domínio mencionada no trabalho de (SANTOS, 2019a), com alterações para integrar as regras de negócio atuais (ver figura 13). Este diagrama utiliza as mesmas notações de setas do esquema apresentado na figura 14 gerado no Grakn Workbase, mas algumas relações foram abstraídas para melhor visualização.

A Fê tem acesso aos registros de imóveis, seus respectivos proprietários e prováveis compradores. Como uma pessoa pode, simultaneamente vender um imóvel e mostrar interesse em um outro imóvel, definiu-se duas relações entre as entidades "Pessoa" e "Imóvel". A relação "vende" interliga uma pessoa com o papel de vendedor ao imóvel, enquanto que "interesseEm" interliga este a uma pessoa com o papel de possível comprador.

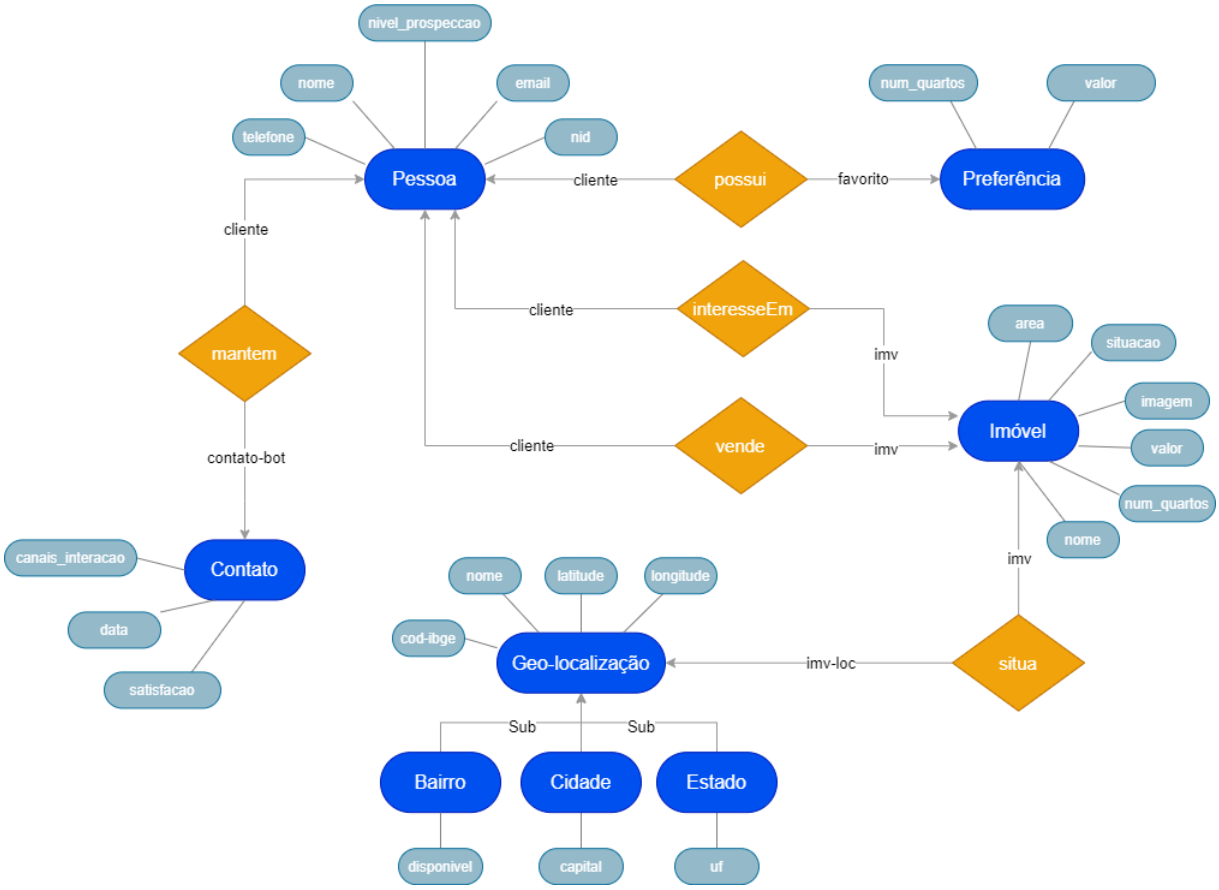
Outra informação importante para o banco é como foi feito o contato entre o usuário e a assistente. Para isso, criou-se a entidade "Contato" que armazena a data e o canal em que ocorreu cada interação com a assistente, bem como o grau de satisfação do usuário ao final da conversa com a Fê. Além do contato, a entidade "Preferência" também possui uma relação com a entidade "Pessoa", sendo que ela representa as características do imóvel ideal do usuário, que as relata durante a conversa.

Por fim, a entidade "Geo-localização" caracteriza onde o imóvel cadastrado para a venda está situado em termos de bairro, cidade e estado. Estes dados foram obtidos do IBGE e tem a finalidade de facilitar a recomendação dos imóveis com base na localização de preferência do usuário.

4.2 Gerando o esquema com Graql

Uma vez com o modelo representacional do domínio em mãos, o próximo passo foi definir o esquema em Graql (ver Anexo A) cujo grafo de conhecimento é representado pela figura 14. Inicialmente, o esquema define as entidades, os papéis que cada uma pode exercer quando interligadas por uma ou mais relações, e seus atributos. A entidade "Pessoa", por exemplo, exerce o papel de cliente quando interligadas com as entidades "Contato", "Preferência" e "Imóvel" pelas relações "mantem", "possui", "interesseEm" e "vende", respectivamente.

Figura 13 – Diagrama de entidade relacionamento adaptado do domínio da Fê.



Fonte: O Autor

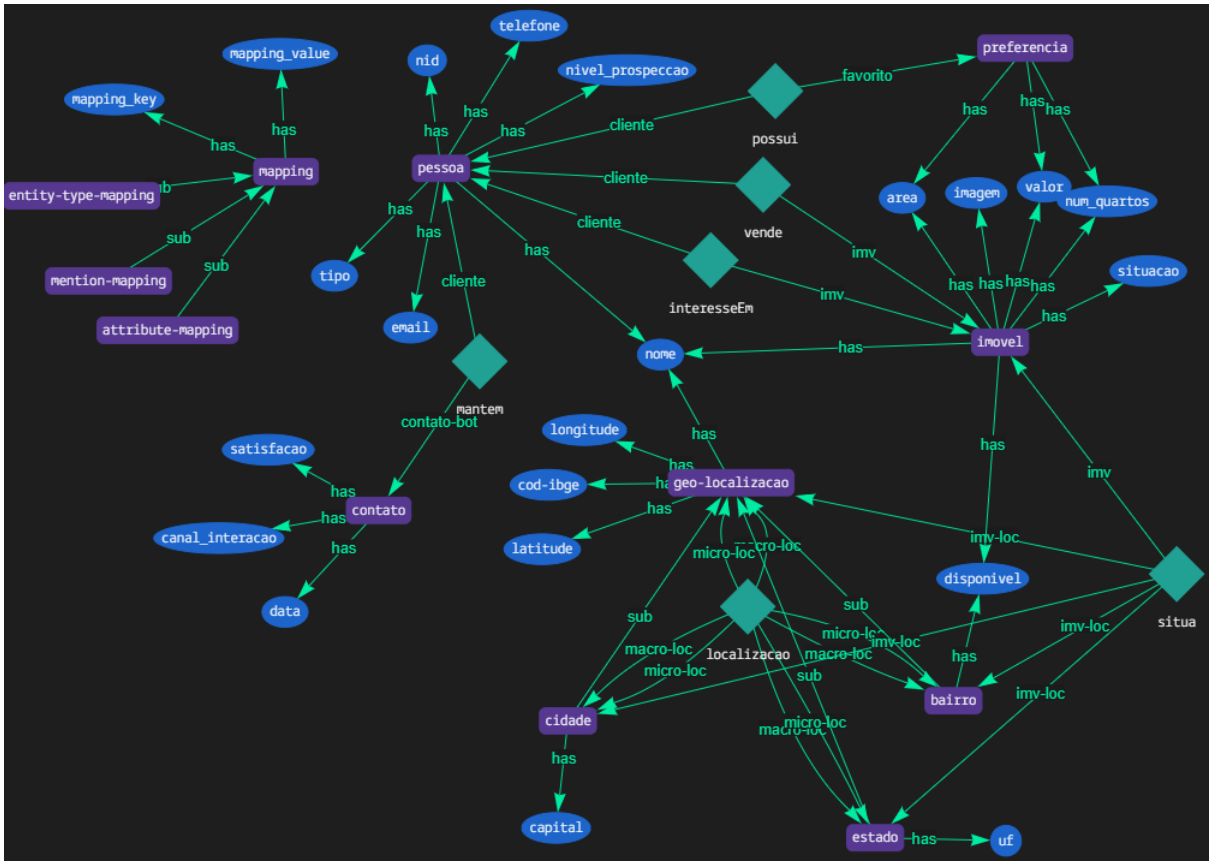
Desta forma, os papéis ajudam na interpretação do grafo de conhecimento e até mesmo na identificação de elementos nas consultas. No código 3, nota-se que é preciso informar os papéis que as entidades interpretam na presença de uma relação “interesseEm”.

Código 3 – Exemplo de consulta utilizando o conceito de *roles* no Grakn

```
// Criando relacao de interesse entre uma pessoa com telefone
// "79912345678" e o imovel de nome "Condominio Agua Azul".
match
  $p isa pessoa, has telefone "79912345678";
  $i isa imovel, has nome "Condominio Agua Azul";
insert $novo-interesse (cliente: $p, imv: $i) isa interesseEm;
```

As três últimas entidades, *mention-mapping*, *entity-type-mapping* e *attribute-mapping*, derivam da entidade *mapping* para herdar seus atributos, *mapping_key* e *mapping_value*. Elas foram criadas para auxiliar em aspectos da conversa entre o usuário e a assistente. Como o próprio nome sugere, elas representam um mapeamento do tipo chave-valor, sendo que as chaves são termos pré-definidos que representam possíveis entradas do usuário e os valores, o significado destes termos para o banco de dados (ver figura 15).

Figura 14 – Grafo de conhecimento representado no Grakn Workbase.



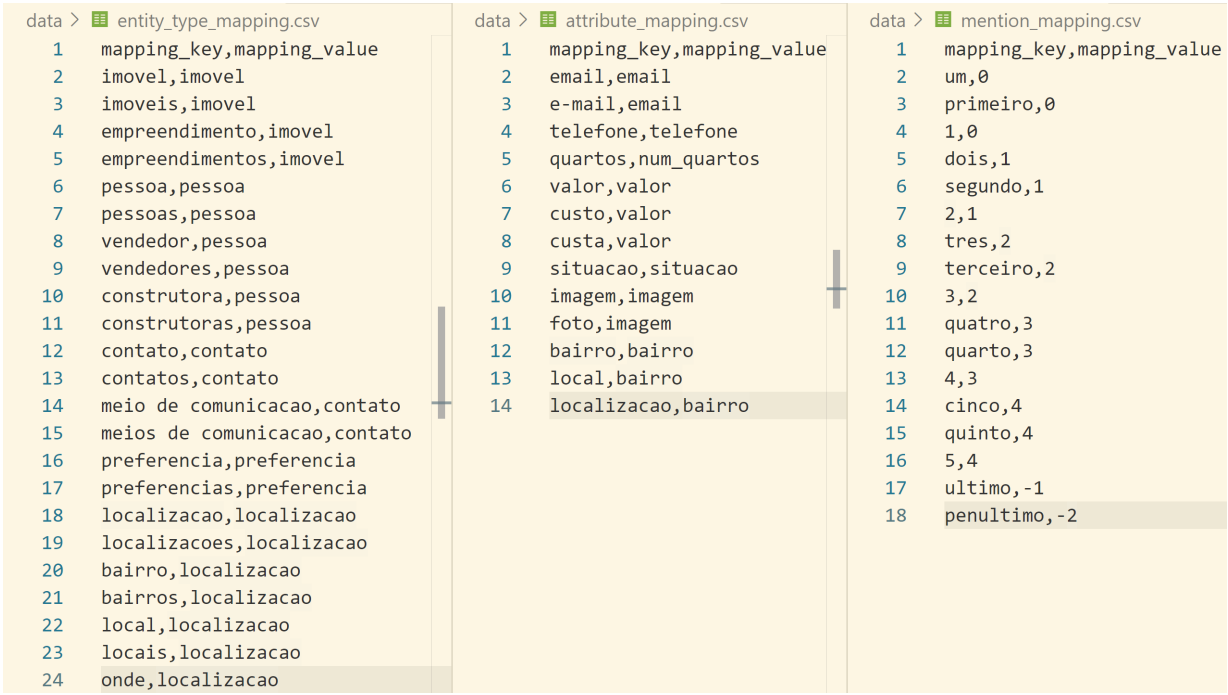
Fonte: O Autor

A título de exemplo, a entidade *attribute-mapping* associa os termos “valor”, “custo” e “custa” ao atributo “valor” da entidade “Imóvel”. Assim, quando o usuário inserir perguntas do tipo “quanto custa”, “qual o valor”, ou “qual o custo”, e o Rasa NLU identificar a entidade *attribute* na intenção, o módulo de ações do Rasa se encarrega de checar com o banco a qual atributo específico do grafo de conhecimento o usuário se refere.

Dando sequência ao esquema, após a definição das entidades, tem-se a declaração dos atributos e seus tipos. Alguns possuem a palavra-chave *regex*, que especifica o padrão de caracteres esperado para aquele atributo, sendo que seu uso neste esquema é como uma alternativa para a criação de um tipo de dado *enum*, já que este não existe na linguagem Graql. Por último, o esquema define as relações que, como uma precondição, conectam dois agentes (entidades, atributos ou outras relações) de acordo com seus papéis.

O conceito de regra (*rules*) é o mecanismo de inferência do Grakn que abstrai e modulariza a lógica de negócio em questão. Com ela, o banco consegue interpretar os dados, deduzir fatos e fornecer respostas automatizadas. A regra *t-localizacao* foi criada para inferir resultados quanto às relações do tipo “localizacao” entre bairros, cidades e estados. Portanto, ao especificar que um bairro B está localizado em uma cidade C e que esta está localizada em um estado E, a existência da regra *t-localizacao* permite induzir que o bairro B está localizado no estado E.

Figura 15 – Da esquerda para direita: entity_type_mapping, attribute_mapping e mention_mapping.



data > entity_type_mapping.csv

```
mapping_key,mapping_value
imovel,imovel
imoveis,imovel
empreendimento,imovel
empreendimentos,imovel
pessoa,pessoa
pessoas,pessoa
vendedor,pessoa
vendedores,pessoa
construtora,pessoa
construtoras,pessoa
contato,contato
contatos,contato
meio de comunicacao,contato
meios de comunicacao,contato
preferencia,preferencia
preferencias,preferencia
localizacao,localizacao
localizacoes,localizacao
bairro,localizacao
bairros,localizacao
local,localizacao
locais,localizacao
onde,localizacao
```

data > attribute_mapping.csv

```
mapping_key,mapping_value
email,email
e-mail,email
telefone,telefone
quartos,num_quartos
valor,valor
custo,valor
custa,valor
situacao,situacao
imagem,imagem
foto,imagem
bairro,bairro
local,bairro
localizacao,bairro
```

data > mention_mapping.csv

```
mapping_key,mapping_value
um,0
primeiro,0
1,0
dois,1
segundo,1
2,1
tres,2
terceiro,2
3,2
quatro,3
quarto,3
4,3
cinco,4
quinto,4
5,4
ultimo,-1
penultimo,-2
```

Fonte: O Autor

4.3 Integração do Grakn com o Rasa

Este tópico apresenta as mudanças feitas sobre o trabalho de (SILVA, 2019), referentes à integração do banco ao *chatbot*, e discute as vantagens que a adição de um grafo de conhecimento trouxe à aplicação. As alterações compreendem a adição de novas *intents* e *actions* ao projeto Rasa.

Antes de aplicar as modificações nos módulos do NLU e de ações do Rasa, fez-se um *script* para inserir dados de usuários, imóveis e os termos que mapeiam atributos, entidades e menções. A tabela 8 apresenta as versões do Rasa e do Grakn utilizadas em paralelo neste projeto.

Tabela 8 – Versões de cada biblioteca python adotada no projeto.

Componente	**Versão**
python	3.7.4
rasa	1.10.0
grakn-client	1.8.1

Fonte: O Autor.

No que se refere ao Rasa NLU, as principais modificações ocorreram na atribuição de novos *slots* e atualização do arquivo que contém dados de treinamento do *chatbot*. Três novas *intents*, *query_attribute*, *query_entities* e *resolve_entity*, foram adicionadas para identificar a categoria da mensagem do usuário. Os itens a seguir descrevem cada categoria.

- intent:query_attribute:

  Corresponde às mensagens cuja consulta ao banco retorna o valor do atributo de uma entidade, ou seja, são perguntas quanto a uma característica específica de uma entidade.

- intent:query_entities:

  Corresponde às mensagens cuja consulta ao banco retorna uma ou mais instâncias de uma entidade dada uma condição, como a relação com outra(s) entidade(s).

- intent:resolve_entity:

  Corresponde às mensagens que indicam menção a alguma entidade já mencionada anteriormente.

O código 4 apresenta algumas sentenças que se encaixam em cada categoria. Recomenda-se adicionar tantos exemplos quanto possível para que, durante o treinamento, o *chatbot* consiga associar os termos entre colchetes aos *slots* correspondentes, indicados entre parênteses.

Código 4 – Exemplos de *intents* para as novas categorias criadas no Rasa

```
## intent:query_attribute
- o [Luzia Residence](imovel) possui quantos [quartos](attribute)
- quantos [quartos](attribute) existem no [Perolas da
   Atalaia](imovel)
- quanto [custa](attribute) o [Marinas Residence](imovel)
- qual o [valor](attribute) do [ultimo](mention) imovel
## intent:query_entities
- quais [imoveis](entity_type) pertencem a [Celi](nome_pessoa)
- em qual [bairro](entity_type) fica o [Marinas
   Residence](nome_imovel)
- [onde](entity_type) fica o [Del Ray](nome_imovel)
- quais os [imoveis](entity_type) disponiveis no bairro
   [Luzia](nome_bairro)
## intent:resolve_entity
- [1](mention)
- [2](mention)
- o [primeiro](mention)
- o [segundo](mention)
- gostei do [terceiro](mention) imovel
- quero mais detalhes sobre o [primeiro](mention) imovel
```

No módulo de ações, há funções (*actions*) para tratar cada tipo de situação (*intent*) com que o *chatbot* se depara. Uma vez que o bot foi treinado e recebe uma mensagem do usuário, ele identifica a qual *intent* aquela mensagem se enquadra e chama a *action* correspondente. Nestas funções, recupera-se os *slots* com seus valores, trata-os se necessário e executa a consulta no banco de dados. Para ilustrar esta etapa, será descrito a seguir como o Rasa trata as mensagens do usuário que estão classificadas como *intent* do tipo *query_entities*.

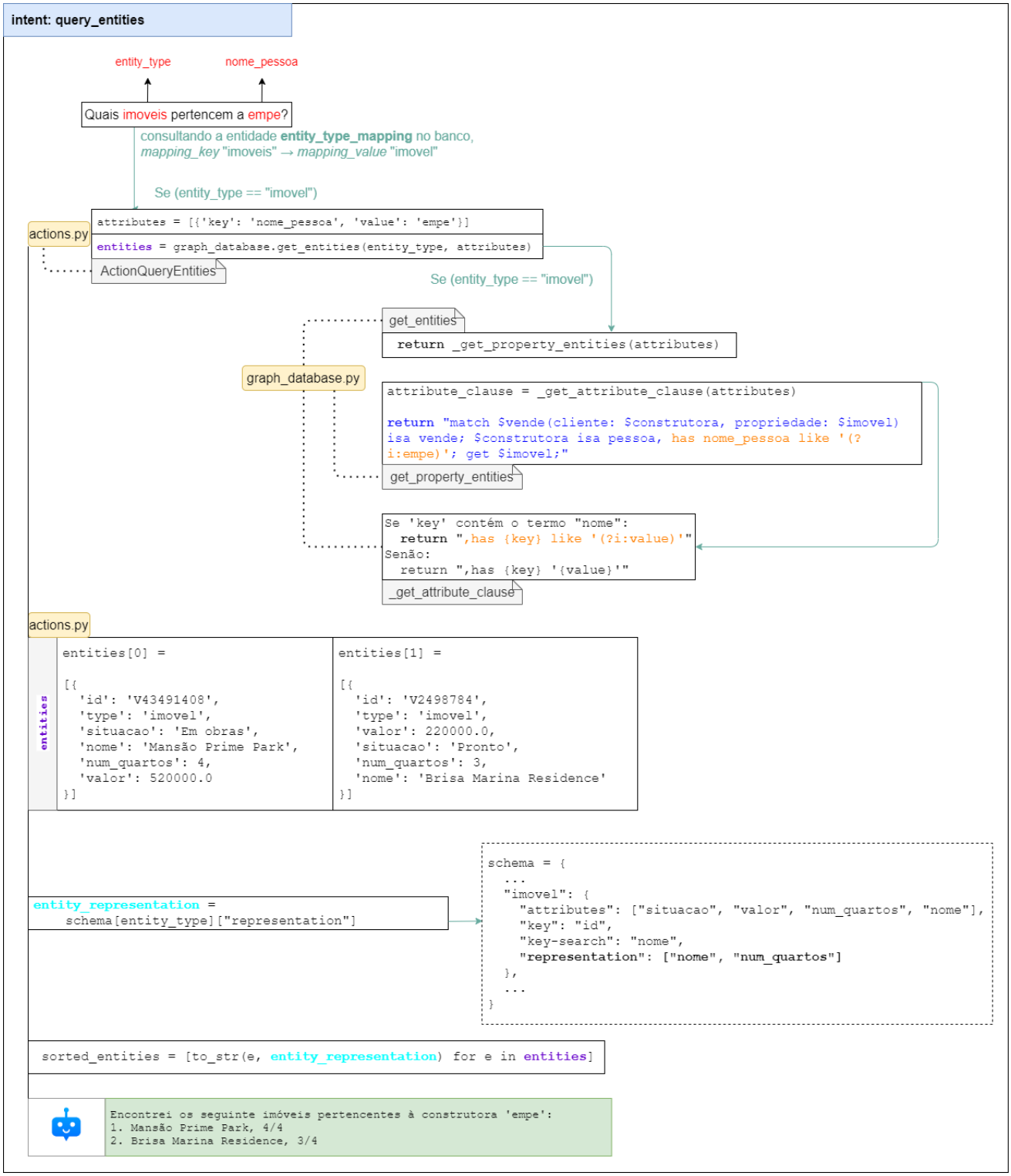
As intents do tipo *query_entities* foram mapeadas no Rasa para reconhecer mensagens que se referem às entidades "Imóvel" e "Geo-localização". O diagrama na figura 16 exibe os passos do *chatbot* desde a identificação dos termos nos *slots* até a recuperação da informação do banco de dados para uma *intent* específica. Nele, os rótulos amarelos identificam os arquivos que contém as definições das *actions* (action.py) e a função auxiliar de consulta ao banco (graph_database.py).

A *action* correspondente à *query_entities* primeiro verifica qual entidade (imóvel ou geo-localização) será recuperada no banco. Para isto, ele recupera o termo do slot *entity_type* e associa a qual entidade ele se refere pelo mapeamento *entity_type_mapping* (figura 15) no banco de dados. Dado que se trata de uma entidade do tipo "Imóvel" e que o próximo *slot* refere-se ao nome de uma pessoa, o Rasa decide que deve recuperar os imóveis que estão à venda por esta pessoa. Esta tomada de decisão é uma regra de negócio imposta no Rasa para separar as mensagens que constituem a mesma *intent*, mas lidam com consultas diferentes do banco de dados.

Quando o Rasa identifica o atributo "nome_pessoa" da mensagem, ele faz a comunicação com o banco, que modifica a consulta correspondente acrescentando os atributos que especificam a pessoa que vende o imóvel, que neste exemplo é a pessoa de nome "empe". Após executar a consulta, o banco retorna como resultado as instâncias (imóveis vendidos pela "empe") na variável *entities*. Elas são apresentadas ao usuário projetando apenas os valores de dois atributos (nome do imóvel e número de quartos). Estes foram definidos em um arquivo, *schema.py*, que contém os atributos considerados mais relevantes para representar a entidade na resposta do *chatbot*.

A lista de entidades, resultante da consulta ao banco, também é armazenada em um *slot* para o caso de o usuário mencionar alguma delas posteriormente na conversa. Se o usuário pergunta "Quanto custa o primeiro imóvel", o Rasa identifica a mensagem como do tipo *query_attribute*, em que o termo "custa" é mapeado para "valor" com o auxílio da entidade *attribute_mapping*, e o termo "primeiro" mapeado para "0" com o auxílio da entidade *mention_mapping*. Com isso, o *chatbot* sabe que o usuário se refere ao primeiro item da lista e procede com a consulta do atributo no banco.

Figura 16 – Diagrama representando como o Rasa interpreta e responde mensagens com intent *query_entities*



Fonte: O Autor

5 Resultados

Este capítulo apresenta os resultados obtidos após a integração do banco orientado a grafos com o *chatbot*. O capítulo foi dividido em dois tópicos que abordam as vantagens do uso do Grakn nos quesitos nível de expressividade (facilidade de interpretar as consultas e quantidade de passos para recuperação do resultado) e aprimoramento lógico do *chatbot* (capacidade de interpretar questões complexas e ambíguas, bem como fazer inferências com o grafo de conhecimento).

5.1 Análise no nível de expressividade

Para avaliar o nível de expressividade, fez-se uma comparação entre o MongoDB e o Graql tendo em vista algumas informações comuns que um usuário pode pedir ao *chatbot*. Para cada possível pergunta, construiu-se uma consulta para cada linguagem, considerando os modelos das figuras 12 e 13, e fez-se um levantamento quanto a semântica.

a Em que cidade está localizado o Green Village? (Código 5)

   Para o modelo orientado a documentos em questão, o nome do município está disponível apenas na coleção “cidades”. Assim, há a necessidade de realizar duas operações: uma para recuperar o documento da coleção “imóvel” dado o seu nome; e outra utilizando o código do município do imóvel para recuperar as informações da cidade desejada. Supondo que, no modelo, o nome do município estivesse acessível diretamente no campo “localizacao” da coleção “imóvel”, seria possível reduzir o número de consultas para um.

   Na consulta com Graql, recupera-se a instância do imóvel a partir do nome, identifica a localização deste com a relação “situa” e, a partir da regra “localizacao” definida no esquema, infere a cidade a que o imóvel pertence.

Código 5 – Grakn vs. MongoDB - Recuperando a cidade em que se encontra um determinado imóvel

```
// Recupera a cidade em que se encontra o imovel Green Village

// Grakn
match
  $i isa imovel, has nome "Green Village";
  (imv: $i, imv-loc: $l) isa situa;
  (micro-loc: $l, macro-loc: $c) isa localizacao;
  $c isa cidade;
get $c;

// MongoDB
var imovel = db.imoveis.findOne({ "nome": "Green Village" })
var cidade = db.cidade.findOne({
    "codigo": imovel.localizacao.cod_municipio
})

// MongoDB (supondo que a colecao "imovel" tivesse o nome
// do municipio dentro do campo "localizacao")
var imovel = db.imoveis.findOne({ "nome": "Green Village" })
var cidade = imovel["localizacao"]["municipio"]
```

b Quais imóveis estão à venda no bairro Luzia em Aracaju? (Código 6)

Considerando que "imóveis" possui o campo "localização" com informações de localização aninhados, a tarefa de recuperar os imóveis localizados em um bairro se tornou mais simples no Mongo. No Graql, foi necessário recuperar os imóveis considerando o bairro de nome "Luzia" e a dedução de que este bairro está localizado na cidade de "Aracaju".

Código 6 – Grakn vs. MongoDB - Recuperando os imóveis que estão à venda em um bairro específico

```
// Recupera os imoveis que estao a venda no bairro Luzia em Aracaju

// Grakn
match
  $i isa imovel;
  (imv: $i, imv-loc: $b) isa situa;
  $b isa bairro, has nome "Luzia";
  (micro-loc: $b, macro-loc: $c) isa localizacao;
  $c isa cidade, has nome "Aracaju";
get $i;

// MongoDB
db.imoveis.find({ "localizacao.bairro": "Luzia",
   "localizacao.cidade": "Aracaju" })
```

c Quanto custa em média um imóvel no bairro Farolândia? (Código 7)

Este exemplo requer o uso de agregação em ambas as linguagens. Aplicar funções de agregação no Graql não adiciona muitos elementos novos da sintaxe à consulta, apenas a palavra-chave, que indica a operação, acompanhada da variável que se submeterá ao cálculo. Neste caso, aplicou-se a função de média *mean* sobre o valor do imóvel. O mesmo vale para os cálculos de soma (*sum*), contagem (*count*), máximo (*max*), mínimo (*min*) e mediana (*median*).

Enquanto isso, no Mongo há uma mudança na estrutura com definições que levam a uma interpretação menos óbvia sobre do que se trata a consulta. Aqui, a agregação é feita por estágios (*match* e *group*), onde o primeiro permite filtrar os documentos e o segundo, agrupar os dados e aplicar operações em campos específicos. Neste exemplo, a primeira etapa recuperou os imóveis do bairro "Farolândia" na cidade de "Aracaju", e a segunda etapa calculou a média do valor destes imóveis e imprimiu o resultado no campo "avgValue". O campo "_id" está como *null* para indicar que não precisa agrupar por um campo específico e, por isso, deve considerar todos os documentos.

Código 7 – Grakn vs. MongoDB - Recuperando o custo médio dos imóveis em um bairro específico

```
// Quanto custa em media os imoveis do bairro Farolandia

// Grakn
match
  $i isa imovel, has valor $valor;
  (imv: $i, imv-loc: $b) isa situa;
  $b isa bairro, has nome "Farolandia";
  (micro-loc: $b, macro-loc: $c) isa localizacao;
  $c isa cidade, has nome "Aracaju";
get $valor; mean $valor;

// MongoDB
db.imoveis.aggregate( [
  { $match: { "localizacao.bairro": "Farolandia",
      "localizacao.cidade": "Aracaju" } },
  { $group: { _id: null, avgValue: { $avg: "$valor" } } }
] );
```

d Quais os bairros mais procurados na cidade de Aracaju? (Código 8)

Antes de aplicar a agregação com a função *count* no Graql, a consulta recupera as instâncias de bairros em "Aracaju" associados a imóveis (pela relação *situa*) que estão ligados a clientes com interesse nesses imóveis (pela relação *interesseEm*). A partir disso, o resultado é agrupado (*group*) pelo nome do bairro acompanhado da contagem de instâncias vinculadas.

Já no Mongo, a estrutura da consulta ficou um pouco mais complexa e houve a necessidade de executar uma pré-consulta para recuperar o código de “Aracaju” da coleção “cidades”. Recuperar os imóveis com clientes interessados requer percorrer os subdocumentos “preferências” de cada usuário para identificar os imóveis pelo campo “imoveis_interesse”, que é um array de IDs.

Para isso, as duas primeiras etapas da agregação utilizam o operador *unwind* que decompõem o documento para cada item do array especificado. Assim, o primeiro decompõe cada “usuário” para cada preferência e o segundo, para cada ID de imóvel presente em “imoveis_interesse”. O operador *lookup*, com função equivalente a de um JOIN, cruza os documentos de “usuários” aos de “imóveis” pelos identificadores “imoveis_interesse” e “_id”.

O quarto estágio seleciona os documentos com código da cidade de “Aracaju” e o último agrupa o resultado por bairros acompanhado da contagem feita com a função *sum* e apresentada pela variável “total”.

Código 8 – Grakn vs. MongoDB - Recuperando os bairros mais procurados em uma cidade

```
// Quais os bairros mais procurados de Aracaju

// Grakn
match
  (imv: $i, imv-loc: $b) isa situa;
  $b isa bairro, has nome $bnome;
  (micro-loc: $b, macro-loc: $c) isa localizacao;
  $c isa cidade, has nome "Aracaju";
  (cliente: $p, imv: $i) isa interesseEm;
get $bnome; group $bnome; count;


// MongoDB
var cidade = db.cidades.findOne({ "nome": "Aracaju" });
db.usuarios.aggregate( [
  { $unwind: "$preferencias" },
  { $unwind: "$preferencias.imoveis_interesse" },
  { $lookup:
      {
        from: "imoveis",
        localField: "imoveis_interesse",
        foreignField: "_id",
        as: "match_interest"
      }
  },
  { $match: { "localizacao.cod_municipio": cidade.codigo } }
  { $group: { _id: "$localizacao.bairro", total: { $sum: 1 } } }
] );
```

e Qual o grau de interesse no Morada Feliz? (Código 9)

Como o grafo de conhecimento possui a relação "interesseEm" entre as entidades "Pessoa" e "Imóvel", basta contar o número de relações deste tipo com a instância de imóvel cujo nome é "Morada Feliz". No caso do Mongo, os imóveis de interesse do usuário são armazenados nos subdocumentos que indicam a preferência do usuário. Após recuperar o documento do imóvel "Morada Feliz" da coleção "imóveis", seu identificador é utilizado na primeira etapa da agregação para filtrar os documentos da coleção "usuários" em que o campo "imoveis_interesse" contém este ID. A segunda etapa soma (*sum*) essas ocorrências e retorna o resultado na variável "total".

Código 9 – Grakn vs. MongoDB - Recuperando o número de interessados em um imóvel

```
// Qual o grau de interesse no Morada Feliz

// Grakn
match
  $i isa imovel, has nome "Morada Feliz";
  $r (cliente: $p, imv: $i) isa interesseEm;
get $r; count;

// MongoDB
var imovel = db.imovel.findOne({ "nome": "Morada Feliz" })
db.usuarios.aggregate( [
  { $match:
    { preferencias:
        { $elemMatch: {imoveis_interesse: $in: [imovel._id] } }
    }
  },
  { $group: { _id: null, total: { $sum: 1 } } }
] );
```

Dados os itens (a) e (b), vale destacar que a decisão de modelagem com o MongoDB pode interferir positivamente ou negativamente na complexidade de uma operação. Por um lado, aninhar documentos entrega melhor performance e garante atomicidade uma vez que permite recuperar todas as informações necessárias de uma vez, bem como realizar operações de atualização e remoção sem a preocupação com a consistência dos dados. Por outro lado, o método de referenciação pode ser vantajoso quando não é possível antecipar o padrão de consulta de dados e um modelo mais normalizado se torna apropriado.

Diferentemente do Mongo, o Graql utiliza parâmetros diferentes na criação de consultas. Todas as operações (leitura, inserção, remoção e agregação) são executadas a partir da cláusula *match*. Com base no vocabulário e conceitos do domínio definidos no esquema, esta cláusula busca as informações no grafo de conhecimento dado um conjunto de instruções que representam componentes do grafo.

Apesar das singularidades de cada linguagem, que se devem principalmente aos diferentes tipos de modelos, os exemplos abordados aqui mostraram que as consultas feitas com o Graql são consistentes em relação ao Mongo, independentemente da operação. Elas têm uma composição semântica análoga à das questões, próxima da linguagem natural, o que reduz as chances de erro e simplifica a interpretação até nos casos mais complexos. Além da questão da consistência, o Graql se beneficia da propriedade de transitividade que vem das inferências descritas como regras no esquema, como pôde ser visto nos exemplos em que havia restrição de localização.

5.2 Análise da interação entre o Rasa e o banco de dados

O resultado final no *chatbot* é o mostrado na figura 17, em que o usuário tem a possibilidade de fazer perguntas dentro do domínio da Fê sem a necessidade de seguir um fluxo sequencial. A combinação do Rasa, que dispõe do processamento de linguagem natural, com o grafo de conhecimento implementado com o Grakn trouxe algumas otimizações na conversação. Termos da entrada do usuário são associados a termos específicos do domínio da Fê, através de mapeamentos, o que permite ao *chatbot* lidar com diálogos mais naturais e diversificados.

Figura 17 – Trecho de conversa com o bot aplicando novas *intents* e *actions*

```
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  ola
Olá
Your input ->  o que voce faz
Estou aqui para te ajudar na busca de um novo lar. Fique à vontade para me perguntar sobre algum imóv
el em particar, caso já tenha algum em mente, mas eu também posso lhe indicar alguns imóveis com base
 nas suas preferências :)
Your input ->  liste os imoveis da empe
Encontrei os seguintes imóveis:
1: Art Life, 3/4
2: Mansão Prime Park, 4/4
3: Monticello Residence, 3/4
Your input ->  quanto custa o segundo imovel
O empreendimento Mansão Prime Park possui moradias que custam em torno de R$1100000.0
Your input ->  onde fica o ravelo
O ravelo fica no bairro JARDINS
Your input ->  quais imoveis estao a venda no bairro Luzia
Encontrei os seguintes imóveis:
1: Porto das Águas, 3/4
2: Pérolas do Luzia, 3/4
3: Pérolas do Luzia, 2/4
Your input ->  qual a situacao do primeiro imovel
As moradias do Porto das Águas tem situação 'Usado'.
```

Fonte: O Autor

Outro ponto positivo além da interpretação é a capacidade do *chatbot* em supor informações graças às propriedades de hierarquia e inferência do Grakn. Este relaciona os bairros, cidades e estados como provenientes da entidade geo-localização, e também os relaciona hierarquicamente no sentido de macrorregiões que contêm microrregiões e vice-versa. Do mesmo modo, relacionamentos complexos podem ser acessados eficientemente pelo banco de dados, como quando há a necessidade de combinar uma entidade pessoa a um imóvel em uma localização

específica. Dada uma consulta que envolve explorar vários níveis do grafo, o Grakn utiliza um mecanismo próprio que otimiza o ato de percorrer os dados no banco.

Em comparação com a versão atual da Fê, cujo fluxo de conversa está representado na figura 18, o banco orientado a grafos forneceu uma base de conhecimento ao *chatbot* concedendo-lhe acesso direto às informações do domínio. Neste exemplo de conversa, o usuário buscava por imóveis localizados no bairro Atalaia com 3 quartos e na faixa de preço de R$350.000,00. O algoritmo de recomendação, mesmo com a existência de imóveis neste bairro, recomendou imóveis localizados em bairros diferentes devido a fatores como o peso das características e os dados passados pelo usuário.

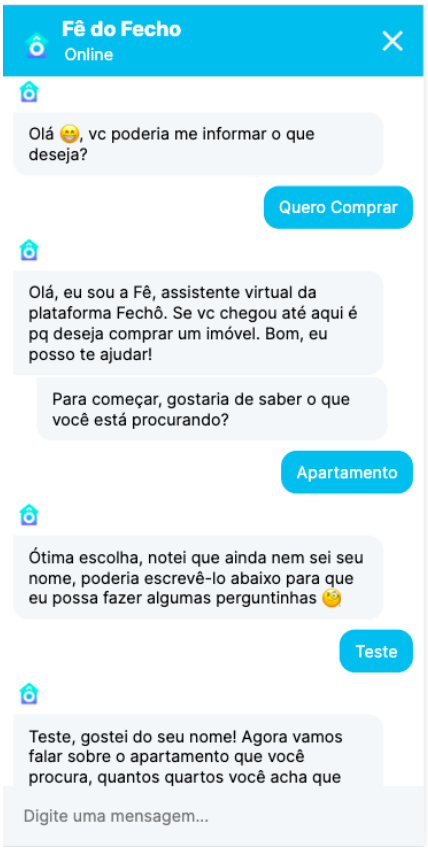
A figura 18d apresenta a situação em que o usuário quer informações dos imóveis vendidos no bairro Atalaia, ainda que não satisfaçam suas preferências. Logo em seguida, o usuário faz uma pergunta específica sobre um imóvel recomendado pelo *chatbot*. Contudo, como este não possui uma base de conhecimento sobre o domínio e está preparado para seguir apenas aquele fluxo de conversa, ele não conseguiu oferecer respostas satisfatórias.

5.3 Considerações Finais

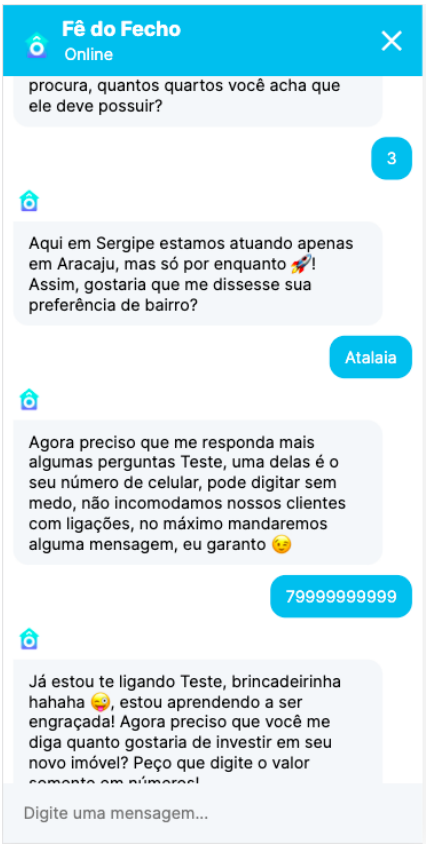
Antes de realizar este trabalho, a Fê era limitada a fazer com que o usuário respondesse a questionários cujas perguntas variavam com o propósito (venda ou compra de um imóvel) do usuário. Quando a intenção era comprar um imóvel, o *chatbot* se comunicava com um sistema de recomendação, cujo algoritmo retornava os imóveis que combinavam com as preferências mencionadas pelo usuário. Da maneira que a assistente foi implementada, a recomendação era um módulo indispensável uma vez que, sem ele, o bot não conseguia passar qualquer outra informação referente ao domínio de imóveis.

Ao integrar o Grakn como uma base de conhecimento para a Fê, ela conseguiu aproveitar o reconhecimento de linguagem natural do Rasa em conjunto com a linguagem de consulta Graql, facilitando a busca de informações do banco, sejam elas complexas ou não. Outro benefício da base de conhecimento é que, por ser uma estrutura fácil de escalar devido às características derivadas dos grafos, ela também garante a escalabilidade do *chatbot* e é capaz de fornecer dados mais ricos em conteúdo.

Figura 18 – Fluxo de conversa da versão atual da Fê.

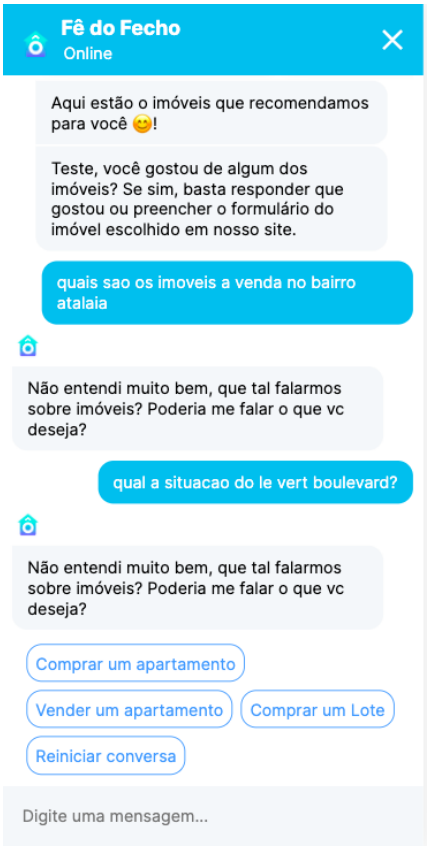


(a) Primeira parte da conversa.



(b) Segunda parte da conversa.

(c) Terceira parte da conversa.



(d) Quarta parte da conversa.

Fonte: https://fe.fecho.app/.

6

Conclusão e Trabalhos Futuros

Terminada a execução do projeto e análise dos resultados, viu-se que o uso de um banco orientado a grafos em conjunto com a assistente Fê aprimorou pontos considerados importantes para um *chatbot*. Não era o bastante que a Fê soubesse o que dizer e quando dizer, considerando que ela deve conhecer as informações do domínio que está inserida. Assim, introduzir uma base de conhecimento com um banco de dados orientado a grafos possibilitou à Fê expandir seu vocabulário. Além disso, a adição ou remoção de conhecimento a essa base ocorre de maneira fluida devido a sua escalabilidade.

Outro ponto positivo encontrado no uso de um banco orientado a grafos na Fê foi a viabilidade de realizar inferências com dados do grafo de conhecimento. Esta característica é importante também no ramo de análise de dados uma vez que, em um cenário com alto volume de dados interligados, a inferência permite identificar informações complexas até para a cognição humana. Em comparação aos demais bancos, o orientado a grafos facilita neste quesito por conta da navegação entre entidades em um grafo ser mais fácil do que agregar atributos de tabelas ou coleções diferentes.

Devido ao fato de bancos orientados a grafos serem a chave para determinar fatos implícitos do conjunto de dados, uma sugestão de trabalho futuro envolve desenvolver um módulo de recomendação para a Fê usando o grafo de conhecimento, explorando o conceito de inferência sobre a base de conhecimento. Além disso, vale também como trabalho futuro expandir o modelo orientado a grafos para considerar outras características do domínio de imóveis como alugueis de temporada, ou a possibilidade de entrar com outros bens (ex. carros, outros imóveis) como parte do pagamento do imóvel.

Referências

BAO, Q.; NI, L.; LIU, J. Hhh: An online medical chatbot system based on knowledge graph and hierarchical bi-directional attention. In: *Proceedings of the Australasian Computer Science Week Multiconference*. [S.l.: s.n.], 2020. p. 1–10. Citado 2 vezes nas páginas 26 e 27.

BARBOSA, J. et al. Introdução ao processamento de linguagem natural usando python. *III Escola Regional de Informatica do Piauí*, v. 1, p. 336–360, 2017. Citado na página 16.

BERGMANN, T. *Set up a knowledge base to encode domain knowledge for Rasa*. 2019. <https://blog.rasa.com/set-up-a-knowledge-base-to-encode-domain-knowledge-for-rasa/>. Acessado em: 12 de nov. de 2019. Citado na página 13.

BHOGAL, J.; CHOKSI, I. Handling big data using nosql. In: IEEE. *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. [S.l.], 2015. p. 393–398. Citado na página 12.

BRANDTZAEG, P. B.; FØLSTAD, A. Why people use chatbots. In: SPRINGER. *International Conference on Internet Science*. [S.l.], 2017. p. 377–392. Citado na página 10.

CAMBRIA, E.; WHITE, B. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, IEEE, v. 9, n. 2, p. 48–57, 2014. Citado 2 vezes nas páginas 16 e 17.

CHOWDHURY, G. G. Natural language processing. *Annual review of information science and technology*, Wiley Online Library, v. 37, n. 1, p. 51–89, 2003. Citado na página 16.

CORDEIRO, A. M. et al. Revisão sistemática: uma revisão narrativa. *Revista do Colégio Brasileiro de Cirurgiões*, SciELO Brasil, v. 34, n. 6, p. 428–431, 2007. Citado na página 24.

COX, G. *Introduction to Graph Databases*. 2017. <https://www.compose.com/articles/introduction-to-graph-databases/>. Acessado em: 27 de fev. 2020. Citado na página 21.

DB-ENGINES. *DB-Engines Ranking - Trend of Graph DBMS Popularity*. 2020. <https://db-engines.com/en/ranking_trend/graph+dbms>. Acessado em: 7 de mar. 2020. Citado na página 28.

GESSERT, F. et al. Nosql database systems: a survey and decision guidance. *Computer Science-Research and Development*, Springer, v. 32, n. 3-4, p. 353–365, 2017. Citado na página 20.

GLEB, B.; VLAD, L. *Capabilities of the Neo4j Graph Database with Real-life Examples*. 2018. <https://rubygarage.org/blog/neo4j-database-guide-with-use-cases>. Acessado em: 27 de fev. 2020. Citado na página 21.

GRAKN. *Defining a Sample Schema*. 2020. <http://docs.grakn.ai/docs/examples/phone-calls-schema>. Acessado em: 19 de nov. 2020. Citado na página 40.

KHAN, R.; DAS, A. *Build Better Chatbots*. [S.l.]: Springer, 2018. Citado na página 10.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: IEEE. *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. [S.l.], 2013. p. 15–19. Citado na página 19.

LISSANDRINI, M.; BRUGNARA, M.; VELEGRAKIS, Y. Beyond macrobenchmarks: microbenchmark-based graph database evaluation. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 12, n. 4, p. 390–403, 2018. Citado 2 vezes nas páginas 35 e 37.

NICKEL, M. et al. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, IEEE, v. 104, n. 1, p. 11–33, 2015. Citado 2 vezes nas páginas 22 e 23.

RICHARDSON, R. Disambiguating databases. *Communications of the ACM*, v. 58, n. 1, p. 54 – 61, 2015. ISSN 00010782. Disponível em: <http://search-ebscohost-com.ez20.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=100097238&lang=pt-br&site=ehost-live>. Citado 2 vezes nas páginas 18 e 19.

SAMPAIO, R. F.; MANCINI, M. C. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. *Brazilian Journal of Physical Therapy*, SciELO Brasil, v. 11, n. 1, p. 83–89, 2007. Citado na página 24.

SANTOS, I. A. d. *Ontologia de domínio para um* chatbot *sobre interesses de compra e venda de imóveis*. Dissertação (B.S. Thesis) — Universidade Federal de Sergipe, 2019. Citado na página 41.

SANTOS, T. C. d. *Projeto de Integração de Chatbot em diferentes interfaces humano-computador*. Dissertação (B.S. Thesis) — Universidade Federal de Sergipe, 2019. Citado 2 vezes nas páginas 10 e 27.

SEGAL, T. *Big Data*. 2019. <https://www.investopedia.com/terms/b/big-data.asp>. Acessado em: 27 de fev. 2020. Citado na página 19.

SEIPEL, P. et al. Speak to your software visualization—exploring component-based software architectures in augmented reality with a conversational interface. In: IEEE. *2019 Working Conference on Software Visualization (VISSOFT)*. [S.l.], 2019. p. 78–82. Citado na página 27.

SILVA, D. d. L. *LIA: um* chatbot *inteligente para o domínio de imóveis*. Dissertação (B.S. Thesis) — Universidade Federal de Sergipe, 2019. Citado 4 vezes nas páginas 10, 11, 27 e 44.

STICHBURY, J. *WTF is a knowledge graph?* 2017. <https://hackernoon.com/wtf-is-a-knowledge-graph-a16603a1a25f>. Acessado em: 19 de fev. 2020. Citado na página 22.

Tang, Y. et al. Graph database based knowledge graph storage and query for power equipment management. In: *2020 12th IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*. [S.l.: s.n.], 2020. p. 1–5. Citado na página 12.

YAN, J. et al. A retrospective of knowledge graphs. *Frontiers of Computer Science*, Springer, v. 12, n. 1, p. 55–74, 2018. Citado 2 vezes nas páginas 21 e 22.

YAO, J. A comparison of different graph database types. 2018. Citado 2 vezes nas páginas 18 e 19.

Zhang, Z. J. Graph databases for knowledge management. *IT Professional*, v. 19, n. 6, p. 26–32, 2017. Citado na página 13.

Anexos

ANEXO A – Esquema usado para criar o banco de dados do Grakn

```
define

# entities

pessoa sub entity,
    plays cliente,
    key nid,
    has telefone,
    has nome,
    has tipo,
    has nivel_prospeccao,
    has email;

contato sub entity,
    plays contato-bot,
    has satisfacao,
    has canal_interacao,
    has data;

preferencia sub entity,
    plays favorito,
    has num_quartos,
    has area,
    has valor;

imovel sub entity,
    plays imv,
    has situacao,
    has disponivel,
```

```
    has imagem,
    has valor,
    has num_quartos,
    has area,
    has nome;

geo-localizacao sub entity,
    abstract,
    plays imv-loc,
    plays micro-loc,
    plays macro-loc,
    key cod-ibge,
    has nome,
    has latitude,
    has longitude;

bairro sub geo-localizacao,
    plays micro-loc,
    has disponivel;
cidade sub geo-localizacao,
    plays micro-loc,
    plays macro-loc,
    has capital;
estado sub geo-localizacao,
    plays macro-loc,
    key uf;

mapping sub entity,
  has mapping_key,
  has mapping_value;

mention-mapping sub mapping;
entity-type-mapping sub mapping;
attribute-mapping sub mapping;

# attribute

mapping_key sub attribute,
  value string;

mapping_value sub attribute,
  value string;
```

```
email sub attribute,
    value string;

nome sub attribute,
    value string;

nivel_prospeccao sub attribute,
    value string,
    regex "^(suspect|lead|prospect)$";

telefone sub attribute,
    value string;

nid sub attribute,
    value string;

tipo sub attribute,
    value string,
    regex "^(proprietario|corretor|imobiliaria|parceiro)$";

satisfacao sub attribute,
    value boolean;

canal_interacao sub attribute,
    value string,
    regex "^(whatsapp|web)$";

data sub attribute,
    value datetime;

num_quartos sub attribute,
    value long;

area sub attribute,
    value double;

valor sub attribute,
    value double;

situacao sub attribute,
    value string,
    regex "^(Pronto|Lançamento|Usado|Em obras)$";
```

```
114 imagem sub attribute,
115     value string;
116
117 cod-ibge sub attribute,
118     value string;
119
120 latitude sub attribute,
121     value double;
122
123 longitude sub attribute,
124     value double;
125
126 disponivel sub attribute,
127     value boolean;
128
129 capital sub attribute,
130     value boolean;
131
132 uf sub attribute,
133     value string;
134
135 # relations
136
137 localizacao sub relation,
138     relates micro-loc,
139     relates macro-loc;
140
141 mantem sub relation,
142     relates cliente,
143     relates contato-bot;
144
145 possui sub relation,
146     relates cliente,
147     relates favorito;
148
149 interesseEm sub relation,
150     relates cliente,
151     relates imv;
152
153 vende sub relation,
154     relates cliente,
155     relates imv;
```

```
156
157 situa sub relation,
158     relates imv,
159     relates imv-loc;
160
161 # rules
162
163 t-localizacao sub rule,
164 when {
165     (micro-loc:$x, macro-loc:$y) isa localizacao;
166     (micro-loc:$y, macro-loc:$z) isa localizacao;
167 }, then {
168     (micro-loc:$x, macro-loc:$z) isa localizacao;
169 };
```