
UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

# Construindo uma API para um Chatbot que efetua Consultas em Bancos de Dados Relacionais

Trabalho de Conclusão de Curso

Iraildo da Costa Carvalho


São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Iraildo da Costa Carvalho

# Construindo uma API para um Chatbot que efetua Consultas em Bancos de Dados Relacionais

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): Prof. Dr. André Britto de Carvalho

São Cristóvão – Sergipe

2022

*Dedico esse trabalho a toda minha família, amigos e professores que me deram todo o suporte necessário.*

# Agradecimentos

A Deus, a família, amigos e professores.

*Tudo posso naquele que me fortalece. Filipenses 4:13*

# Resumo

A informação é algo essencial para a sociedade moderna, especialmente quando ela é obtida de forma rápida e eficaz. E para que isso ocorra é necessário ter uma forma adequada de armazenar e processar tais informações, fornecendo de forma rápida e eficaz quando solicitadas pelo usuário interessado. Os sistemas de banco de dados são componentes essenciais para a sociedade atual, pois eles possibilitam esse gerenciamento das informações criadas nos dispositivos. Porém, para que seja possível obter informações sobre essas bases, geralmente são criados sistemas específicos para cada banco de dados, com interfaces engessadas e comandos bem definidos. Mas se for possível trazer todas essas informações de interesse utilizando a linguagem natural utilizada diariamente pelo ser humano? Isso traria muitas possíbilidade, tais como consultar um catálogo apenas com um comando de voz ou ter acesso a um relatório com apenas uma simples troca de mensagens. Hoje em dia isso é possível com o auxílio de chatbots que podem trazer todas as informações desejadas pelo usuário apenas com simples troca de mensagens com o sistema. Porém, os chatbots são geralmente elaborados com base nos relacionamentos e nas tabelas de cada banco de dados construído. No entanto, existe a possibilidade de criar um chatbot que traga informações independente da base de dados a ser conectada. E isso pode ser criado com auxilio de API's que se comunicam diretamente com a base de dados e fornecem para o chatbots uma interface de alto nível, onde a comunicação entre API e chatbot independe dos relacionamentos e tabelas existente nas bases de dados de interesse. Ciente disso, esse trabalho visa fornecer uma API que faz consultas em qualquer base de dados conectada, utilizando um compilador capaz de traduzir os comandos em álgebra relacional, fornecido pelo chatbot, na linguagem de consulta SQL.

**Palavras-chave**: Banco de Dados, API, Álgebra Relacional, Interface de Linguagem Natural, Chatbot.

# Sumário


**1 Introdução . . . . . 8**
1.1 Motivação . . . . . 9
1.1.1 Objetivos . . . . . 10
1.1.2 Metodologia . . . . . 11

**2 Fundamentação Teórica . . . . . 13**
2.1 Trabalhos Relacionados . . . . . 15

**3 Framework . . . . . 17**
3.1 Arquitetura Geral . . . . . 17
3.1.1 Representação da Consulta . . . . . 17
3.1.1.1 Seleção . . . . . 18
3.1.1.2 Projeção . . . . . 20
3.2 Arquitetura da API . . . . . 20
3.3 Módulos . . . . . 22
3.3.1 Método doisTermos e umTermo . . . . . 22
3.3.2 Método expressãoRelacional . . . . . 23
3.3.3 Método transformaEmGrafos e todosCaminhos . . . . . 23
3.3.4 Compilador . . . . . 24
3.3.5 AllTables . . . . . 27

**4 Aplicação . . . . . 29**
4.1 SGBD's . . . . . 29
4.2 Estudo de Caso . . . . . 29
4.2.1 Caso de teste - método UmTermo . . . . . 30
4.2.2 Caso de teste - método doisTermos . . . . . 31
4.2.3 Caso de teste - método todos os caminhos . . . . . 32
4.2.4 Caso de teste - método expressãoRelacional . . . . . 32
4.2.5 Caso de teste - método todasAsTabelas . . . . . 34

**5 Conclusão . . . . . 37**

**Referências . . . . . 38**

**Apêndices** **40**

**APÊNDICE A GRAMÁTICA E EXPRESSÕES REGULARES DA FERRAMENTA 42**

# 1 Introdução

Nas últimas décadas, é possível identificar que a informação é algo essencial para a sociedade moderna, principalmente quando se trata da velocidade e da facilidade com que ela é obtida. Essas informações são geradas constantemente, sejam por máquinas automatizadas ou sejam por pessoas que se utilizam dos mais diversos dispositivos para troca de mensagens, postagens de fotos, envios de e-mails ou até mesmo armazenamento dos dados de interesse de empresas ou instituições em alguma base. Essas pessoas, hoje em dia, carregam em seus bolsos computadores com um poder de processamento superior ao dos computadores que levaram a humanidade à lua. Tais tecnologias permitem a comunicação entre diferentes tipos de aparelhos e possibilitam que um celular esteja conectado simultaneamente a um veículo e a um satélite. Além disso, com poucos movimentos em seu telefone é possível relembrar o lugar onde esqueceu o seu veículo ou visualizar sua própria posição em tempo real, em algum aplicativo de mapa de sua preferência. Essa quantidade significante de informação que produzimos todos os dias, fez com que Elmasri e Navanthe (2015) escrevessem na introdução de seu livro as seguintes palavras: "Bancos de Dados e Sistemas de Bancos de Dados são componentes essenciais da sociedade moderna".

Todo esse avanço tecnológico forneceu uma base para o aumento das exigências da sociedade para tratar as informações. Conforme declara Vaidya, Ambad e Bhosle (2018), no tempo da indústria 4.0, o objetivo central é atender às necessidades individuais dos clientes e a facilidade com que as informações são obtidas torna-se crucial, pois atende a uma das exigências desses novos clientes. Para os profissionais de computação armazenar e recuperar a informação pode parecer simples e até rotineiro. Porém, ainda resta transpor o obstáculo de acesso para o usuário comum. Sejam cientistas em campo, que estejam catalogando novas espécies de plantas, gerentes que busquem gerar um relatório mensal da empresa com as informações que lhe são mais pertinentes, ou um síndico que deseje saber os gastos de água dos moradores do prédio na última semana. Nenhum desses usuários tem acesso de forma simples, sem muito gasto com sistemas e aplicativos, que tragam para eles relatórios com essas informações.

É muito difícil para uma pessoa que não possui conhecimento aprofundado em computação tenha interesse ou dedique-se ao estudo de linguagens específicas como SQL. Até mesmo para um programador ela torna-se difícil quando não se conhece o domínio que aquela base de dados representa e como ela foi construída. Os profissionais demoram certo tempo entendendo o funcionamento das regras de negócios de determinado domínio analisando os diagramas para poder efetuar consultas mais aprimoradas. Assim, principalmente para o usuário comum, é mais simples apenas utilizar-se de sua linguagem natural para realizar perguntas e obter as respostas de forma compreensível. Deste modo, é necessária a construção de ferramentas que facilitem esse acesso de forma que os usuários comuns e até profissionais da área consigam de uma forma mais simples, utilizando-se da linguagem natural, efetuar consultas de seu interesse.

O mercado de ferramentas que auxiliam nas necessidades especificas de cada indivíduo é bem vasto. Atualmente existem diversas API's - *Application programming interface* (interface de programação de aplicações, em português) que auxiliam nas tarefas diárias de cada pessoa e servem como serviços para a construção de diversas outras ferramentas. Como exemplo, temos a API do Google Maps que além de fornecer a localização de seus usuários em tempo real, ainda serve de base para o funcionamento de vários aplicativos, como WAZE[1](WAZE, 2022), UBER [2](UBER, 2022) e aplicativos de Delivery. Os aplicativos de Delivery, por sua vez, exibem a rota de entrega dos pedidos feitos pelos clientes, tais como UBEREATS(UBEREATS, 2022) e IFOOD(IFOOD, 2022). Somente o UBER, foi responsável por movimentar um segmento profissional inteiro, dando a ele uma nova forma de execução e que passou a ser conhecido como motoristas de aplicativos. Esses motoristas destacam-se pelo uso de aplicativos para gerenciamento de seus serviços, o que os diferenciam dos taxistas que adotam o modelo tradicional de serviços de transporte de passageiros.

Diante disso, esse trabalho consiste em construir uma API que sirva de ponte entre uma aplicação que tenha como parâmetros consultas a base de dados, em álgebra relacional, no formato JSON e aplicações que utilizam a linguagem natural do usuário, facilitando as buscas. Especificamente, esse trabalho será entregue em conjunto com a ferramenta baseada em chatbot que será construida por *Victor Carity Feitosa Silva Carneiro*, e que fornece os comandos em algebra relacional, formando assim uma NLIDB(natural language interfaces to databases).

## 1.1 Motivação

As API's muitas das vezes contam com uma vasta base de dados e fornecem um apoio para a construção de diferentes ferramentas em variados serviços. Na área do comércio, por exemplo, uma das ferramentas que chama atenção é o EASYASK. Essa ferramenta é um motor

[1] Aplicativo de navegação colaborativo que tem como objetivo contornar obstáculos literais e figurativos nas estradas.

[2] A empresa UBER oferece tanto taxis como carros particulares em sua plataforma de mobilidade para passageiros e motoristas.

de pesquisa para eCommerce citado por Agostinho (2003),que possui o poder de processar buscas complexas por produtos, utilizando a linguagem natural através do comando de voz. Hoje em dia, devido a grande concorrência, o comércio busca atender as necessidades específicas de cada cliente. Para isso, são construídos sistemas que efetuam buscas complexas por produtos, mas é necessário que o usuário digite ou escolha diversos parâmetros tais como menor preço, produto mais vendido, marca mais conhecida etc. Porém, se tais buscas fossem feitas por meio de uma conversação e na melhor das hipóteses trouxesse todas as informações por meio de uma única frase, trariam mais comodidade aos usuários e, consequentemente, o fariam utilizar mais vezes aquele determinado serviço. Além disso, existe um interesse crescente na utilização da tecnologia dos chatbots por facilitar a comunicação entre os serviços prestados e o usuário, utilizando a linguagem natural. Estima-se que em 2020, 85% da interação entre os consumidores e as empresas acontecerão sem nenhuma interação humana, utilizando-se como principal função os chatbots, conforme relata a Gartner (2011), não deixando dúvidas do poder facilitador dessa tecnologia para a comunicação natural e autônoma entre o ser humano e a base de dados de interesse.

A comunicação entre o ser humano e a base de dados de interesse pode ser feita da forma mais natural possível, quebrando a estrutura da linguagem humana em uma estrutura menos complexa e que possa ser traduzida para uma linguagem de consulta a um banco de dados, a exemplo do SQL, como já foi proposto no trabalho de Nazareth (2008). Porém, a utilização da linguagem SQL não é suficiente ainda devido a falta de transportabilidade entre domínios e o problema de decisão sobre qual tabela efetuar a consulta quando se usa, por exemplo, o termo "Nome". Em um domínio onde existem tabelas professor, aluno e técnico e que todas elas possuem o atributo "Nome", é necessário verificar sobre qual tabela o usuário tem interesse de pesquisar, exigindo que o usuário conheça bastante da estrutura da base de dados e tornando a consulta complexa. Grande parte dos sistemas comerciais e não comerciais, utilizam os SGBDR's para manipulação de suas bases de dados. Esses SGBDR's tem como linguagem padrão o SQL, que é utilizado para recuperar e realizar a manutenção das informações. Então como tratar esse problema, traduzindo a linguagem natural em uma estrutura menos complexa e que possibilite a efetuar a busca nas bases de dados relacionais?

A proposta de solução desse trabalho é através da ferramenta construída por *Victor Carity Feitosa Silva Carneiroo*, receber uma estrutura menos complexa, a álgebra relacional, transformando-a na linguagem compreendidade por banco de dados relacionais, o SQL, trazendo assim uma independência do domínio e buscando eliminar o problema de decisão sobre qual caminho escolher para efetuar a busca.

### 1.1.1 Objetivos

Este trabalho tem como objetivo propor e construir uma API que forneça a retroalimentação para um chatbot inquisitivo e que permite a consulta em bancos de dados relacionais. A

API vai receber as informações em álgebra relacional, em formato JSON, e irá traduzí-las em linguagem SQL, fará a consulta na base de dados e fornecerá os dados para o chatbot que se encarregará de repassá-los para usuário. A cada termo digitado pelo usuário o chatbot efetuará a pesquisa a essa API, na forma de álgebra relacional, que por sua vez trará todos as tabelas que podem ser de interesse do usuário.

### 1.1.2 Metodologia

A API será construída, contendo como núcleo, o compilador que faz parte da ferramenta denominada EasyRA, desenvolvida por Bilecki e Kalempa (2015). Essa ferramenta é utilizada para traduzir consultas em álgebra relacional para o SQL. Com esse núcleo como base, a ferramenta web, fornecerá 4 principais serviços utilizando a comunicação com os padrões definidos pelo estilo de arquitetura Rest. Tal comunicação se dará através de troca de arquivos no formato JSON e que terá como parâmetros as consultas em Algebra Relacional. O SGBD de interesse será o PostgreSQL e a linguagem de programação será o Java para WEB. A ferramenta será construida seguindo os padrões MVC e com a arquitetura de comunicação REST utilizando arquivos no formato JSON para se comunicar com o chatbot.

Os principais serviços oferecidos por essa ferramenta são: conectar a alguma base de dados de interesse, armazenada no SQGB PostgreSQL (com os parâmetros de nome do banco, login, senha e *schema*), fornecer todas as tabelas dessa base de dado após a conexão, exibir todas as colunas de alguma tabela passada como parâmetro, receber, como parâmetro, alguma consulta em álgebra relacional e retornar o resultado com as informações trazidas da base de dados no formato JSON e por fim percorrer todos as tabelas que tenha alguma ligação, utilizando as chaves estrangeiras, trazendo todos os possíveis caminhos que aquela consulta poderá traçar para efetuar uma junção. Com esses dados retornados, os usuários do Front End da API poderão escolher qual decisão tomar para obter as informações que precisam.

O desenvolvimento dessa ferramenta será possível devido a tabela de informações que cada SGBD possui e que contêm dados sobre os dados, teoricamente conhecido por metadados. Esses metadados fornecem comandos que serão utilizados para percorrer as tabelas através das chaves estrangeiras que podem ser necessárias quando for efetuar alguma consulta que exija junção entre tabelas, por exemplo.

No próximo capítulo, será apresentada a fundamentação teórica, onde são abordados os temas aplicados nesse trabalho de conclusão de curso, com suas devidas referências e os trabalhos relacionados. No Capitulo 3 é apresentada a arquitetura do framework, onde é delhada a arquitetura geral do sistema construído, a API e como é feita a comunicação com o chatbot. Além disso, são apresentados e detalhados os principais métodos da API, responsáveis pelo processamento e consulta das informações. Já no capítulo posterior, da aplicação, são apresentados exemplos do funcionamento com base em algumas consultas feitas em determinada base de dados. Por fim, na conclusão é feita a retomada dos objetivos e os resultados alcançados por esse

trabalho, bem como sugestões de trabalhos futuros com base nesse.

# 2

# Fundamentação Teórica

O propósito desse trabalho de conclusão de curso é construir uma API para retroalimentação de chatbots que efetuam as consultas elaboradas pelo usuário à base de dados relacionais, na forma de linguagem natural. Segundo (IMMAGIC, 2015), API é uma interface que permite a comunicação entre dispositivos ou sistemas. É uma ponte que liga o sistema A ao sistema B. A sua utilização não só diminui o tamanho do software, como também facilita a manutenção e o torna menos suscetível a erros.

O chatbot é um programa de computador construído com o intuito de simular diálogos inteligentes entre o ser humano e a máquina, utilizando a linguagem natural (**??**). O chatbot com o qual a API proposta por esse trabalho vai se comunicar, terá um comportamento inquisitivo, ou seja, através de uma série de perguntas e respostas, serão obtidas informações que tenham sido deixadas de fora da primeira consulta feita pelo usuário. Essas informações são importantes para eliminar o máximo de ambiguidade na consulta. Com base nessas respostas o chatbot irá guiar o usuário sugerindo opções pré-selecionadas, com base nas consultas feitas anteriormente, exibindo novos exemplos de consultas, afunilando ainda mais a comunicação para a direção que o usuário deseja. Esse conjunto de respostas são enviadas pelo chatbot para a API que irá se encarregar de efetuar as consultas em alguma base de dados relacional.

Devido a grande simplicidade e base matemática, o modelo relacional é o preferido na construção dos sistemas desde a década de 70, quando foi introduzido por Ted Codd, da IBM Research ELMASRI; NAVANTHE. Com esse modelo é possível manter registros de base de dados usando o conceito de relação matemática, que se assemelha a um tabela de valores. Para manipular tais base de dados são utilizados SGBD's relacionais, que é o software responsável por todo o acesso aos bancos de dados e tratamento de requisições, conforme afirma Date (2004). O SGBDR disponibiliza uma interface para recuperar e manter as informações contidas na base de dados e essa interface no modelo relacional é a linguagem SQL, que se tornou sua linguagem padrão, conforme relata SILBERSCHATZ; KORTH; SUDARSHAN em um de seus livros.

A tradução direta da linguagem natural do usuário para a linguagem SQL é bastante custosa devido a criação de tabelas intermediárias que sirvam como um dicionário de dados para a consulta ser efetiva. Isso ocorre porque a linguagem SQL contêm diversos comandos e diferentes formas de elaborá-los exigindo uma linguagem intermediária menos complexa e com estrutura mais simples para que a tradução seja eficiente. Ao mesmo tempo, é necessário que essa linguagem, ou estrutura intermediária, seja facilmente traduzida na linguagem SQL, que é a linguagem padrão dos SGBDR's. A linguagem SQL tem como base formal a álgebra relacional e o cálculo relacional. Uma expressão relacional é definida de forma recursiva como uma relação (RAMAKRISHNAN; GEHRKE, 2008). Normalmente, ela é considerada parte integral do modelo relacional e suas operações estão divididas em dois grupos: unárias e binárias (BILECKI; KALEMPA, 2015).

Com esses conceitos em mente, a utilização da álgebra relacional como meio intermediário entre a linguagem natural e a linguagem de consulta padrão, o SQL, justifica-se por ao menos três fatores. Primeiro, ela oferece uma base formal para as operacões no modelo relacional. Segundo, ela é utilizada como base para implementacão e otimizacão de consultas nos módulos de otimizacão e processamento de consultas que são partes integrais dos SGBDR. Terceiro, alguns de seus conceitos são incorporados na base da linguagem de consulta padrão SQL para SGBDR ELMASRI; NAVANTHE. Além do mais, existem alguns trabalhos e ferramentas propostas, tais como o trabalho de (BILECKI; KALEMPA, 2015), que faz a tradução da algebra relacional em linguagem SQL, e a ordem contrária também é possível, e o (ALKHALIFAH, 2014) que constrói uma ferramenta que interpreta os cinco operadores relacionais (seleção, projeção, união, cruzamento e diferenciação), feitos em álgebra relacional, para a linguagem de consulta SQL de forma didática.

Para que seja possível a tradução da álgebra relacional para a linguagem SQL, será utilizado um compilador. Um compilador segundo, AHO Alfred V. et al. (2008), pode ser definido como um programa que recebe uma sequência de códigos em uma linguagem de entrada, denominada fonte, e devolve a mesma sequência de códigos mas em outra liguagem como saída, denominada objeto. Ainda segundo AHO Alfred V. et al. (2008), o processo de compilação pode se dividido em duas partes, análise e sintese. Na análise, o programa fonte é dividido em partes e depois aplicado a estrutura da gramática e gerada uma representação intermédiária do programa fonte. Já a sintese, constrói o programa a partir da representação intermédiária e dos registros na tabela de símbolos. Quanto a análise, ela pode ser léxica, sintática ou semântica. A análise léxica é a primeira etapa de um compilador e ela tem como função transformar sequências de carateres em sequências de caracteres que são chamados de lexemas, ao serem classificados, esses lexemas são transformados em tokens e a sequência de tokens é a entrada recibida na análise sintática DELAMARO (2004). Esses tokens são usados na etapa de análise sintática, que por sua vez, é responsável por utilizar os tokens fornecidos pela análise lexica para formar uma árvore de derivação, onde é mostrada a estrutura gramatical da sequência de tokens AHO Alfred V. et al. (2008). Já a etapa da análise semântica, utiliza as informações

contidas na árvore de derivação e da tabela de símbolos para verificar se os aspactos semânticos do programa estão corretos. DELAMARO (2004). Para a construção desse módulo compilador, foi usada a ferramenta GALS, desenvolvida como trabalho de conclusão de curso na UFSC em 2003. Essa ferramenta gera automaticamente os analisadores léxico, sintático e semântico com base na gramática de entrada GESSER. Assim, no compilador desse projeto, são utilizadas as três etapas, léxica, sintática e semântica para traduzir a linguagem fornecida pelo chatbot, em álgebra relacional, e por sua vez compilar para a linguagem compreendida por bancos de dados relacionais, o SQL.

## 2.1 Trabalhos Relacionados

API'S e diversas outras ferramentas que usam a linguagem natural como meio facilitador para a comunicação entre os usuários e os serviços prestados, já foram propostas e até contruídas em alguns trabalhos, citando como exemplo o ProgramAR (PRATES et al., 2013), que é uma ferramenta que auxilia no ensino da álgebra relacional. Além desse trabalho, temos alguns outros que foram utilizados para o ambiente acadêmico com o mesmo intuito, tais como EnsinAR (PAES, 2004), SIMALG (LAUTERT, 2010), iDFQL (APPEL; TRAINA, 2004). Porém, essas ferramentas foram contruídas com o propósito especifico de apenas ensinar os conceitos e comandos da álgebra relacional, traduzindo-os em SQL. Além disso, elas trazem interfaces bastante padronizadas e que são difícieis de configurar, baseando-se em telas que é necessário colocar informações padrões para que seja possível a sua utilização. Adicionalmente, elas apresentam algumas dificuldades no processo de instalação, sintaxe e com simbologia não adequada. No entanto, os conceitos de tradução da álgebra relacional para a linguagem SQL são utilizados por todas delas, os quais, são objetos de interesse desse trabalho.

Outros trabalhos, que aproximam-se da proposta aqui apresentada, pode ser encontrados em (BOHLE, 2018), que criou um chatbot que possui o objetivo de buscar, analisar e comunicar informações médicas através da linguagem natural. Também existem os trabalhos de SINGH; SOLANKI, que converte a linguagem natural diretamente em SQL. Ele possui uma arquitetura que oferece ao usuário uma interface que aceita linguagem natural textual e realiza as consultas diretamente no banco. Esse tipo de arquitetura é denominado de NQL (Natural Language Query). Esse sistema é semelhante a ferramenta construída por PAPADAKIS; KEFALAS; STILIANAKAKIS, possuindo o mesmo princípio na arquitetura.

A arquitetura feita nesse trabalho está relacionada com as proposta de trabalhos como BILECKI; KALEMPA e o (ALKHALIFAH, 2014), ambos os trabalhos utilizam um compilador para a tradução da expressão relacional em SQL. Segundo (LOUDEN, 2004), compiladores são programas de computador que traduzem de uma linguagem para outra e que recebe como entrada um programa escrito em uma linguagem fonte e o traduz em uma linguagem alvo. Dessa forma, a arquitetura desse projeto também utilizará um compilador, irá efetuar a consulta no SGBD e

retornar as informações para o chatbot. Ao mesmo tempo, para corrigir o problema de indecisão quando um termo fornecido pelo usuário pertencer a diversas tabelas, a API irá se encarregar de consultar o catálogo do sistema SGBD. No catálogo dos sistemas SGBD's estão contidas diversas informações sobre a base de dados, dentre essas estão os metadados. Metadados são usualmente descritos como "dados sobre dados". portanto, são informações adicionais (além da informação espacial e tabular) necessárias para que os dados se tornem úteis. "Por outras palavras, são um conjunto de características sobre os dados que não estão normalmente incluídas nos dados propriamente ditos"(MONTEIRO; LIFSCHITZ; BRAYNER, 2007). A consulta a esses metadados é efetuada, na maioria das vezes, de forma diferente nos SGBD's existentes. Atualmente, vários SGBD's estão em funcionamento em algum sistema, seja ele Oracle, SQL Server, MariaDB ou PostgreSQL. Cada um desses SGBD's possui uma forma específica de acesso aos metadados, de modo que comandos utilizados em um não serve para o outro. Visto essa dificuldade, essse trabalho irá focar no SQGB PostgreSQL.

# 3
# Framework

A proposta geral do trabalho é uma API do tipo NLIDB[1] que, através do uso de um chatbot, receberá consultas em Algebra Relacional e executará tais consultas em alguma base de dados armazenada no SGBD do PostgreSQL. Para que isso seja viável, o funcionamento desta ferramenta irá seguir a arquitetura mostrada na figura 1.

## 3.1 Arquitetura Geral

A arquitetura geral da ferramenta é composta pelo ChatBot e a API de retroalimentação. O chatbot irá ser o *Front End* enquanto a API irá se encarregar de utilizar os dados fornecidos pelo ChatBot e retornar a consulta feita na base de dados, sendo assim o *Back End*. O usuário irá interagir com o chatbot na linguagem natural e essa ferramenta irá processar as informações e repassar para a API no formato de operações que tem como base a álgebra relacional. Essas operações representam consultas que serão feitas na base de dados através da API produzida nesse TCC. Tal representação será detalhada na próxima seção e em seguida será explanada sobre as bases de dados escolhidas para os testes.

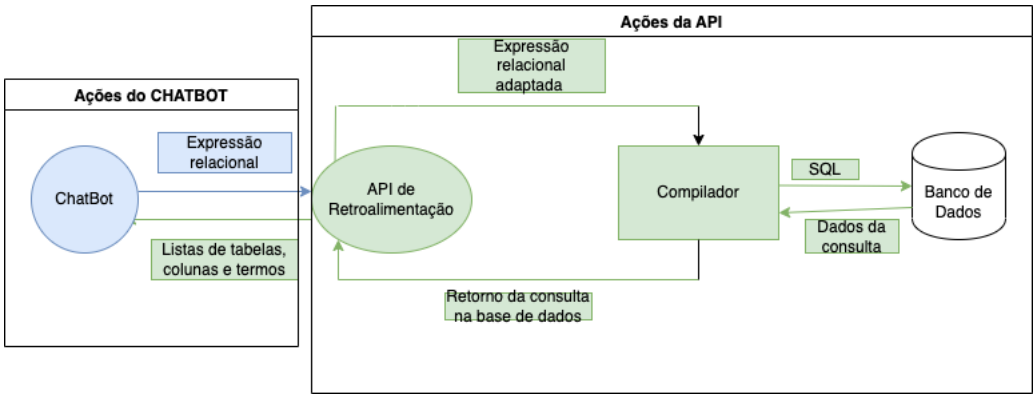
A figura 1 mostra a arquitetura geral do sistema que primeiro o chatbot recebe como entrada as frases em linguagem natural, do usuário, processa essas frases e as transforma elas em álgebra relacional. Depois disso, ele gera uma saída para a API, que processa essa saída e converte ela em SQL utilizando o compilador. Ao final é efetuada a consulta na base de dados e retornadas as informações necessárias sobre as tabelas ou colunas do banco de dados conectado.

### 3.1.1 Representação da Consulta

A ferramenta construída nesse TCC, desenvolvida em linguagem Java, é um serviço web com abordagem Rest e que é composta por alguns módulos que serão descritos nas próximas

[1] Natural language interfaces to databases

Figura 1 – Arquitetura Geral do Projeto




seções. Dentre esses módulos, está o compilador que é o responsável por fazer a tradução da entrada, uma expressão em álgebra relacional, para a linguagem SQL. Essa entrada possui um padrão e representa uma consulta que o usuário deseja realizar em alguma base de dados. Esse padrão será seguido pelo Chatbot para repassar, na forma de álgebra relacional, a consulta digitada pelo usuário em linguagem natural. A seguir é mostrado como é formado esse padrão, para cada operação de álgebra relacional que a ferramenta produzida nesse TCC possui, e que pode ser seguido para efetuar integração com algum outro *Front End*.

As operações nada mais são que consultas utilizadas para selecionar tuplas de uma determinada relação ou a combinação desssas relacionadas a várias relacões com o objetivo de representar uma requisição de recuperação sobre o banco de dados. (TAKAI; ITALIANO I. C. FERREIRA, 2005). Existem dois grupos de operações. O primeiro inclui operações que são baseadas na teoria de conjuntos, tais como interseção, união, produto cartesiano e diferença. Já o segundo grupo é constituído de operações apenas para bases de dados relacionais, entre elas estão projeção, seleção, renomeação junção e divisão. (ELMASRI; NAVANTHE, 2015).

As Seções 3.1.1.1 e 3.1.1.2 descrevem as operações contidas na API desse TCC e que são apresentadas na figura 2 juntamente com todas as outras que constituem o quadro completo de operações da álgrebra relacional.

#### 3.1.1.1 Seleção

A operação de seleção possui a finalidade de consultar dados retornando tuplas que satisfaçam determinadas condições. Essa operação divide a relação em dois conjuntos. O primeiro é o conjunto que segue as condições estabelecidas e são exibidas suas informações no resultado da consulta; o outro conjunto é o que não satisfaz as condições e que por esse motivo não são exibidas suas informações. (ELMASRI; NAVANTHE, 2015).

Essa operação é semelhante a um Select com Where na linguagem SQL e possui a seguinte sintaxe:

$$\sigma_{\text{<condicão de seleção>}}(\mathbf{R})$$

Figura 2 – Operações da Álgebral Relacional

| Operação | Símbolo | Sintaxe |
|---|---|---|
| Seleção | $\sigma$ | $\sigma_{<condicão\ de\ selecão>}(R)$ |
| Projeção | $\pi$ | $\pi_{<lista\ de\ atributos>}(R)$ |
| Atribuição | $\leftarrow$ | $R \leftarrow X$ |
| Renomeação | $\rho$ | $\rho_{(B1,B2,B3,...)}(R)$<br>$\rho_{novonome}(R)$ |
| Divisão | $\div$ | $R \div S$ |
| Junção | $\theta\times$ | $R\,\theta \times S$ |
| Junção Natural | $\bowtie$ | $R \bowtie S$ |
| Junção Externa Esquerda | $\rtimes$ | $R \rtimes S$ |
| Junção Externa Direita | $\ltimes$ | $R \ltimes S$ |
| Junção Externa Completa | $\rtimes\ltimes$ | $R \rtimes\ltimes S$ |
| União | $\cup$ | $R \cup S$ |
| Interseção | $\cap$ | $R \cap S$ |
| Diferença | - | $R - S$ |
| Produto Cartesiano | $\times$ | $R \times S$ |
| Inserção | $\cup$ | $R \leftarrow R \cup E$ |
| Alteração | $\delta$ | $\delta_{(atributo \leftarrow expressão)}(R)$ |
| Exclusão | - | $R \leftarrow R - E$<br>$R \leftarrow \sigma_{<condicão\ de\ selecão>}(R)$ |

Como pode ser visto acima, a sintaxe é formada pelo símbolo grego σ (Sigma) que representa o operador Seleção e logo após um condição de seleção, onde é colocada a expressão booleana que é especificada com base nos atributos da relação (R). A figura 3, retirada do artigo de Bilecki e Kalempa (2015), é apresentada um exemplo do funcionamento do operador de seleção.

Figura 3 – Exemplo de consulta usando Selecao

Considerando a relação convenio (cdConvenio, deRazaoSocial, nuTelefone, deEndereco, cdLabo):

| cdConvenio | deRazaoSocial | nuTelefone | deEndereco | cdLabo |
|---|---|---|---|---|
| 1 | Anil Seguradora | 34556778 | Rua H | 1 |
| 2 | Maria Seguradora | 34545645 | Rua B | 2 |
| 3 | LFB Seguradora | 45324524 | Rua Y | 2 |
| 4 | Mopi Seguradora | 12312312 | Rua a | 1 |

O resultado da operação $\sigma_{cdLabo=1}(CONVENIO)$ é:

| cdConvenio | deRazaoSocial | nuTelefone | deEndereco | cdLabo |
|---|---|---|---|---|
| 1 | Anil Seguradora | 34556778 | Rua H | 1 |
| 4 | Mopi Seguradora | 12312312 | Rua a | 1 |

A consulta SQL equivalente é:
**SELECT** * **FROM** convenio **WHERE** cdLabo = 1

#### 3.1.1.2 Projeção

A operação de projeção seleciona determinadas colunas de uma relação. Se considerarmos a relação como uma tabela, essa operação escolhe certas colunas da tabela e descarta as outras. A operação de projeção é executada, formando uma nova relação contendo apenas os atributos selecionados e com as duplicidades eliminadas. (ELMASRI; NAVANTHE, 2015).

Essa operação é comparável ao comando SELECT DISTINCT, da linguagem SQL, com os atributos especificados antes do FROM e possui a seguinte sintaxe:

$$\pi_{\text{<listas de atributos>}}(\mathbf{R})$$

A projeção é formada pela letra grega $\pi$ (Pi) e uma lista de atributos <listas de atributos> da relação (R). A figura 4 representa um exemplo do funcionamento do operador de projeção.

Figura 4 – Exemplo de consulta usando Projeção

Considerando a relação paciente(cdPaciente, nmPaciente, deEndPaciente, deTelPaciente, dtNasc, cdPlanoSaude):

| cdPaciente | nmPaciente | deEndPaciente | deTelPaciente | dtNasc | cdPlanoSaude |
|---|---|---|---|---|---|
| 2 | Marcos Freitas | Rua B | 32224444 | 22/10/1956 | 1 |
| 3 | Alberta Daltro | Rua I | 32225555 | 01/03/1999 | 1 |
| 4 | José Santos | Rua A | 22222222 | 01/03/1999 | 1 |

O resultado da operação $\pi_{nmPaciente\ ,deEndPaciente}$ (paciente) é:

| nmPaciente | deEndPaciente |
|---|---|
| Marcos Freitas | Rua B |
| Alberta Daltro | Rua I |
| José Santos | Rua A |

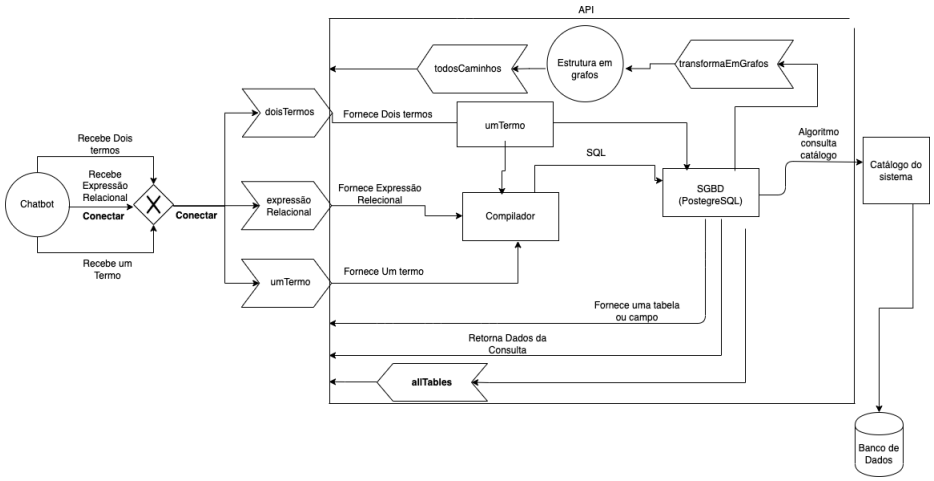
A consulta SQL equivalente é:
**SELECT DISTINCT** nmPaciente, deEndPaciente **FROM** paciente

## 3.2 Arquitetura da API

A consulta aos metadados depende muito do fabricante do SGBD e devido a essa dificuldade, inicialmente, esse trabalho irá limitar-se a consultas efetuadas no SGBD PostgreSQL. Na figura 5 pode ser visto a arquitetura específica da API, composta pelas partes a seguir descritas:

- Método doisTermos - Responsável por trazer todos os caminhos entre os dois termos digitados.
- Método expressãoRelacional - Responsável por efetuar a consulta desejada para o usuário.
- Método umTermo - Responsável por trazer todas as informações sobre aquele termo digitado.

Figura 5 – Arquitetura Específica da API




- Método transformaEmGrafos - Converte as tabelas e seus relacionamentos em uma estrutura de grafos.
- Método todoCaminhos - Responsável por trazer todos os caminhos entre duas tabelas digitadas.
- Estrutura de grafos - A representação em grafo da base de dados conectada.
- Módulo compilador - Processa as informações recebidas no formato de álgebra relacional e a converte na linguagem SQL.
- SGBD - O Sistema Gerenciador de Banco de dados, onde fica armazenado a base de dados.
- Catálogo do Sistema (SGBD) - Possui todas as informações sobre os metadados da base de dados conectada.

Com as informações recebidas pelo chatbot em formato de álgebra relacional, a API irá se encarregar de traduzí-las em SQL e efetuar a consulta na base de dados utilizando a própria tabela de informações do SGBD. Essas informações, que estão contidas no metadados é o que a API precisa para fornecer o conjunto de todos os caminhos, percorridos através das relações entre as tabelas, que contêm dados relacionados com os termos pesquisados pelo usuário. Com essas informações armazenadas, será possível montar uma estrutura em grafos, no qual cada nó é representado pelas tabelas e seus vértices são representados pelas relações entre elas, que utiliza como atributo comum as chaves estrangeiras. Com essa estrutura montada, será possível aplicar algoritmos como o de busca em profundidade, ao passo que o chatbot faz as solicitações. Após encontrada a tabela ou atributo de interesse, essas informações serão repassadas para o chatbot em forma de tabela, o qual irá se encarregar de informar ou formular novas perguntas sobre o que o usuário deseja. Toda essa estrutura aqui descrita pode ser observada na figura 5, a qual traz a arquiterura específica da API.

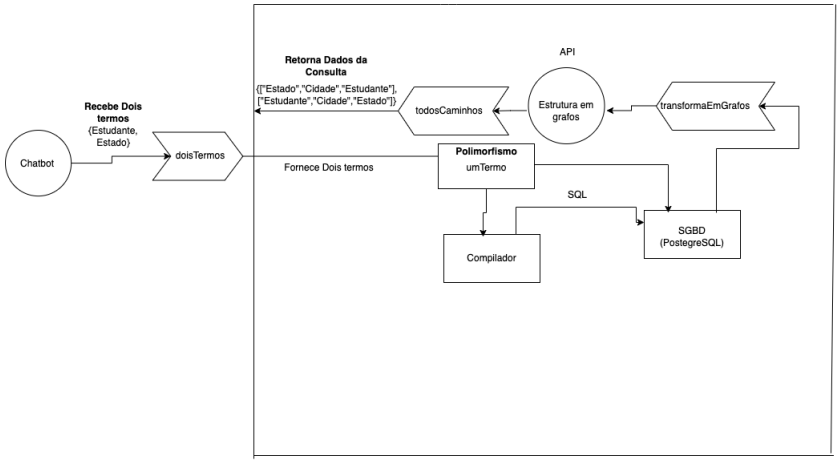
## 3.3 Módulos

Os principais módulos da API criada são os "doisTermos", "expressãoRelacional", "umTermo", "transformaEmGrafos", "todosCaminhos", compilador e por fim "allTables".

### 3.3.1 Método doisTermos e umTermo

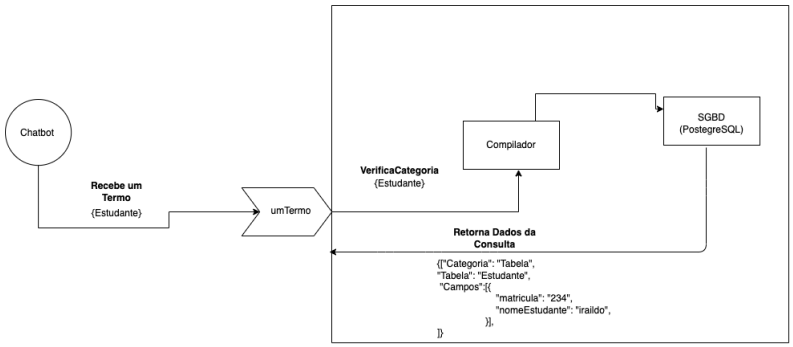
O método doisTermos recebe dois valores fornecidos pelo ChatBot e se encarrega de verificar se esses dois termos representam tabelas na base de dados, através do método umTermo. Caso representem tabelas esses objetos, o doisTermos aciona o método transformaEmGrafos passando como entrada os dois termos digitados. Caso seja contrário, será enviada uma mensagem para a API cliente informando se existe algo relacionado a ele na base de dados. Se existir, são enviados os dados, juntamente com a mensagem, sobre esse termo informando se ele é campo e a qual/quais tabela(s) pertence(m). Caso não exista nenhum termo igual ao digitado é enviado uma mensagem informando isso para a API cliente.O funcionamento desses dois métodos pode ser visualizado na figura 5. Mas a figura 6, mostra o esquema de entradas e saídas do método doisTermos com um exemplo utilizando as tabelas da base de dados de uma estudante.

Figura 6 – Funcionamento do método DoisTermos




O método umTermo se encarrega de verificar as informações sobre aquele termo digitado, dizendo a qual classe pertence, se é tabela ou não e informando todos os locais e todas as informações relacionadas a esse termo, como por exemplo se ele é uma tabela no banco de dados ou se ele é uma coluna e, nesse caso, a qual tabela pertence essa coluna. Além disso, quando verifica que o termo é uma tabela ele já retorna uma lista com a consulta feita por todos os dados pertecentes aquela tabela, semelhante ao comando do SQL: select * from table.

Figura 7 – Funcionamento do método umTermo




Na figura 7, mostra um exemplo do funcionamento do método umTermo.

### 3.3.2 Método expressãoRelacional

O método expressão relacional é responsável por receber uma consulta no formato de álgebra relacional e retornar os dados que essa consulta está solicitando. Para que isso seja possível, o método se encarrega de converter a expressão relacional em SQL, com auxílio do compilador, e assim efetua a consulta no SGBD.

Figura 8 – Funcionamento do método expressaoRelacional

**Efetuar consulta em Expressão Relacional**

```
[GET] tcc/api/banco/{termo}/expressaoRelacional
Input:
{
    algebra: string
}
Output
{
  id: algebra
  consulta: {
            tabela: string,
            campos:[{
                    values: [{
                            tipo: tipo,
                            chave: boolean,
                            idCampo: number,
                            value: string,
                            description: string
                    } ]
            }]
          }
}
```

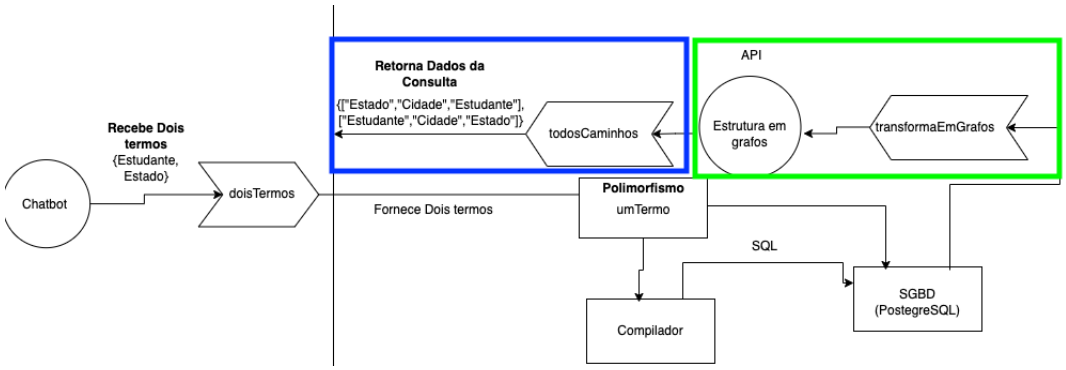
Na figura 8, um json com a entrada e saída usada pelo método expressão relacional, representando seu funcionamento.

### 3.3.3 Método transformaEmGrafos e todosCaminhos

O método transformaEmGrafos está diretamente ligado ao método todosCaminhos pois após passar duas tabelas, a API vai se encarregar de enviá-las para o método transformaEmGrafos ao qual irá transformar as tabelas e seus relacionamentos, respectivamente, em vértices e arestas formando a representação dessa base de dados de interesse em um grafo. Já o método

todosCaminhos irá receber esse grafo e efetuar uma busca em largura para encontrar todos os caminhos entre as duas tabelas informadas pela API cliente.

Figura 9 – Funcionamento dos métodos transformaEmGrafos e todosCaminhos

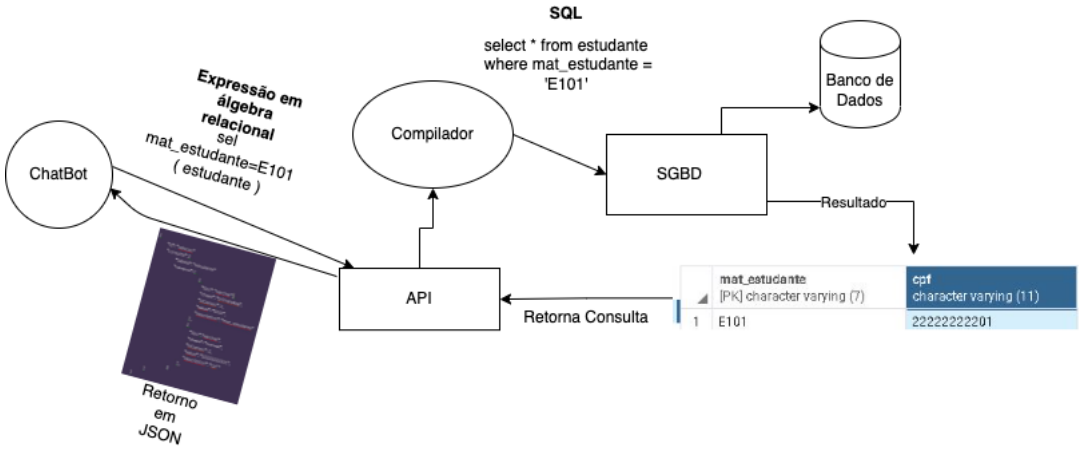



Na figura 9, mostra o esquema de como funciona os métodos transformaEmGrafos e todosCaminhos.

### 3.3.4 Compilador

O módulo compilador é baseado no trabalho de Bilecki e Kalempa (2015), que teve como objetivo criar uma ferramenta de aprendizagem de álgebra relacional e foi aplicada na Universidade do Estado de Santa Catarina. Esse compilador é o módulo mais importante deste trabalho e possui a função principal de converter a álgebra relacional em linguagem SQL e, após essa conversão, efetuar a consulta na base de dados desejada.

Figura 10 – Funcionamento do Compilador




Conforme é mostrado na figura 10, após receber a expressão relacional do ChatBot, o compilador traduz essa expressão para a linguagem SQL. No exemplo é enviado para o compilador o comando de "seleção"de um estudante que tem a matrícula igual a "E101", essa seleção será feita na tabela de estudante.

```
sel mat_estudante=E101 (estudante)
```

A consulta traduzida em SQL é:

```
select * from estudante where mat_estudante='E101'
```

Essa consulta gera o seguinte retorno para o ChatBot, no formato JSON, exibido na 11.

Figura 11 – Retorno da consulta feita no exemplo da Figura 6

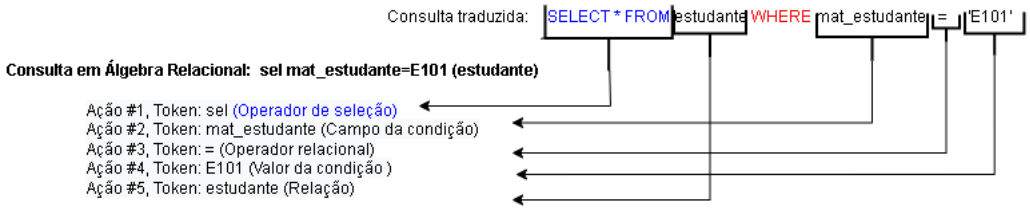

```
{
    "id": "selecao"
    "consulta": {
        "tabela": "estudante"
        "campos": [
                    {
                      "tipo": "varchar",
                      "chave": "primaryKey",
                      "idCampo": 1,
                      "value": "E101",
                      "description": "mat_estudante"
                    },
                    {
                      "tipo":"varchar",
                      "chave": "normal",
                      "idCampo": 2,
                      "value": "22222222201",
                      "description": "cpf "
                    },
                  ]
        }
}
```


O compilador utiliza as três etapas necessárias da análise, que são a léxica, sintática e a semântica para a tradução de um programa tendo como entrada linguagem fonte, que é em álgebra relacional para a linguagem objeto, que é em SQL. A gramática com o conjunto de lexemas ou tokens desse compilador está no Apendice A, na página 39. Considerando a gramática citada e seguinte comando:

```
sel mat_estudante=E101 (estudante)
```

o compilador trata as ações semânticas geradas conforme mostra a 12.

Figura 12 – Ações semânticas do compilador




Depois disso, o resultado executado é enviado como entrada para o método montar da classe de Análise, que analisa as ações que são geradas fazendo o tratamento necessário para traduzir o comando de álgebra relacional para a linguagem SQL. Os principais métodos podem ser vistos na figura 13, onde são mostrados os métodos do compilador que são responsáveis por transformar a sequência de caracteres do comando em sequência de simbolos da gramática.

Figura 13 – Trechos do principal método do Compilador

```
public void retornarConsulta(String txtTrad) {
            if (this.getCon().returnCon() == null)
            String[] str = txtTrad.split("\n");
            lexico = new Lexico();
            sintatico = new Sintatico();
            semantico = new Semantico(this.getContAlge(), con,txtTrad);
            Analise analise = semantico.getAnalise();
            int pE = analise.verificarContagemDeDeterminadoChar("(");
            int pD = analise.verificarContagemDeDeterminadoChar(")");
            lexico.setInput(txtTrad);
            try {
                  sintatico.parse(lexico, semantico);
            } catch (LexicalError e1) {
                  analise.tratarErro(e1);

            } catch (SyntaticError e2) {
                  analise.tratarErro(e2);
            } catch (SemanticError e3) {
                  analise.tratarErro(e3);
            }
            if (!analise.houveErro) {
                  analise.contarNroRelacao();
                  analise.verificarValorCondicao();
                  analise.verificarCampoCondicao();
                  analise.verificarRelacaoF1();
                  analise.validarRelacoes();
                  analise.validacaoContinuacaoProjecao();
                  analise.validarListaProjecao();
                  analise.verificarBinario();
            }
            String consulta = analise.montar();
            Controlador_Principal.RodarQuery executar =
            new  Controlador_Principal.RodarQuery(
                                    Controlador_Principal.this,
                                    consulta,
                                    false);
            Thread thread = new Thread(executar);
            thread.start();

      }
```

Na figura 14, é exibido o método montar, que é responsável por analisar e construir o comando em SQL.

Figura 14 – Trecho do método Montar

```
public String montar()
{
  if (houveErro) {
    return "ERRO";
  }
  String sets;
  if (temDelta) {
    String nome = "";
    if (idDelta.equals("")) {
      nome = relacoes.get(0).relacao;
    } else {
      nome = idDelta;
    }
    String cConsulta = "UPDATE " + nome + " SET ";
    sets = fazListaDeValores2(nome);
    if (sets.equals(""))
    {
      houveErro = true;
      return "";
    }
    String condicao = "";
    if (selecao) {
      condicao = fazCondicao();
    }
    if ((renomeouRel) || (renomeouRel2))
    {
      houveErro = true;
      return "";
    }

    return cConsulta + sets + condicao;
  }
  if ((operacoesSolo) && (!binario.equals("")) && (!selecao) &&
(!projecao)) {
    if (!existeRelacao(nmVar)) {
      houveErro = true;
      return "";
    }
    String rel2 = "";
    for (Analise.Relacao r : relacoes) {
      if (r.acao == 15)
        rel2 = r.relacao;
    }
    if (rel2.trim().equals(""))
    {
      houveErro = true;
      return "";
    }
    if (!existeRelacao(rel2)) {
      houveErro = true;
      return "";
    }
    relacoes.add(new Analise.Relacao(this, 6, nmVar));
    validarLista(campoCondicao, false, false);
    validarLista(campoCondicaoBin, false, false);
    if (houveErro) {
      return "";
    }
    String sql = "SELECT * FROM " + nmVar;
    String sqlN = operacaoBinario(sql);
    return sql + sqlN;
  }
 }
```

### 3.3.5 AllTables

O modulo AllTables é responsável por iniciar a comunicação após o usuário informar os dados do banco que deseja se conectar, conforme mostrado na figura 15.

Figura 15 – Estrutura do JSON o módulo AllTables

```
Obter todas as tabelas do banco
[GET] tcc/api/banco/alltables
input:
{
   nomeBanco: string,
   idBanco: string,
   passwordBanco: string,
}
output:
{
   allTables:[
      {
         nome: string,
         campos: [
              {
                descricaoCampo: string,
                tipoCampo: string,
              }
         ],
         chavePrimaria: string,
      }
   ]

}
```

# 4
# Aplicação

A proposta geral do trabalho é construir uma API que auxilie um chatbot nas consultas em base de dados.

## 4.1 SGBD's

Com as informações recebidas pelo chatbot em forma de álgebra relacional, a API irá se encarregar de traduzí-la em SQL e efetuar a consulta na base de dados utilzando a própria tabela de informações do SGBD. Essas informações, que estão contidas no metadados são o que a API precisa para fornecer o conjunto de todos os caminhos, percorridos através das relações entre as tabelas, que contêm dados relacionados com os termos pesquisados pelo usuário. Com essas informações armazenadas, será possível montar uma estrutura em grafos, no qual cada nó é representado pelas tabelas e seus vértices são representados pelas relações entre elas, que utilizando como atributo comum as chaves estrangeiras. Com essa estrutura montada, será possível aplicar algoritmos como o de busca em profundidade, ao passo que o chatbot faz as solicitações. Após encontrada a tabela ou atributo de interesse, essas informações serão repassadas para o chatbot em forma de tabela, o qual se encarregará informar ou formular novas perguntas sobre o que o usuário deseja. Toda essa estrutura aqui descrita pode ser observada na figura 2, a qual traz a arquiterura específica da API.

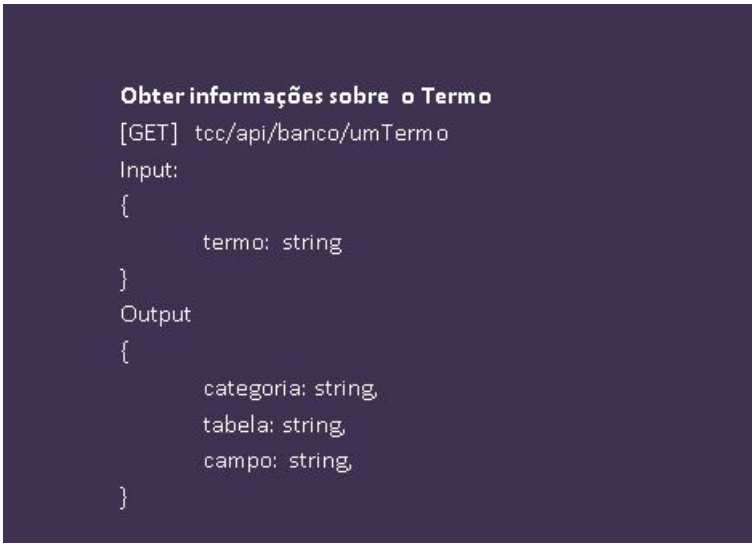
## 4.2 Estudo de Caso

Neste estudo de caso serão apresentados os exemplos das funcionalidades existentes na API e as explicações de cada uma delas. Será usado como exemplo uma base de dados simples, relacionada a login de estudante, composta por 2 tabelas, a quais são: estudante e usuário. Essas tabelas possuem relacionamento com cardinalidade 1:1 e considerando a tabela usuário como uma entidade fraca, dependendo assim, exclusivamente da existência da entidade estudante.

### 4.2.1 Caso de teste - método UmTermo

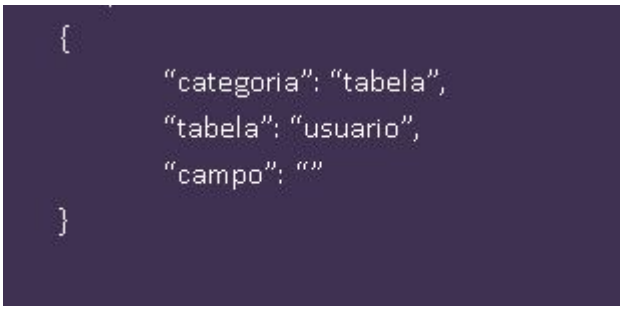
Esse método é responsável por classificar a categoria que o termo digitado pelo usuário pertence. Ele classifica se esse termo é tabela ou campo (coluna da tabela). Essa classificação é importante para saber qual o nível de informação que o usuário deseja no momento dessa consulta. Se for tabela, vai ser retornar valor indicando que é uma tabela se for campo, ele vai retornar valor indicando que é campo e a qual ou as quais tabelas pertence.

Figura 16 – Estrutura do JSON do Método umTermo




Conforme pode ser visualizado na figura 16, a aplicação disponibiliza um serviço que pode ser executado através de uma requisição GET, passando como entrada um termo. Essa requisição gera uma saída com três parâmetros, "categoria", que classifica se é campo ou tabela, "tabela"que é responsável por trazer qual tabela que o termo digitado pertence e "campo", indica o nome do campo ou retorna vazio se o termno digitado for tabela ou inexistente. Na figura 17 é mostrado um exemplo prático, passando como parâmetro do termo "usuário", no qual pertence a base de dados do sistema de login de estudante.

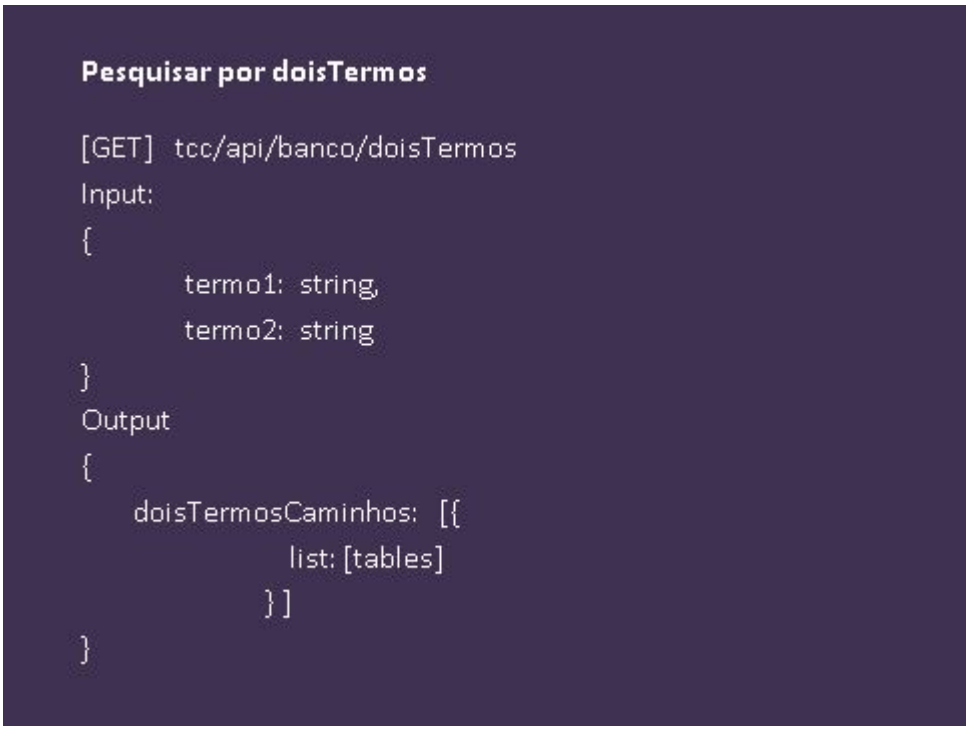
Figura 17 – Response do Método umTermo



### 4.2.2 Caso de teste - método doisTermos

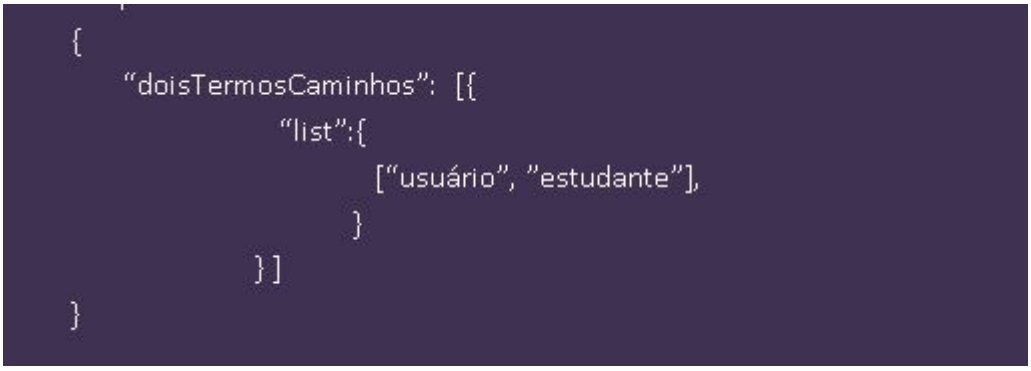
Esse é um dos principais métodos, responsável por trazer as informações do relacionamento existente entre os dois termos digitados. Esse método trabalha em conjunto com os métodos todosOsCaminhos(allPaths) que será abordado na próxima seção e o método umTermo, já exemplificado na seção anterior.

Figura 18 – Estrutura JSON do Método doisTermo




Na figura 18, pode ser visualizada a estrutura JSON do método doisTermos, o qual recebe como parâmetro de requisição GET, dois termos do tipo string. Ele retorna uma lista de vetores contendo em ordem de transição das chaves primárias as tabelas que estão relacionadas. Ele mostra todos os relacionamentos existentes entre as tabelas intermediárias do termo1 e o termo2. Na figura 19 é mostrado o exemplo prático do retorno de uma consulta utilizando as tabelas estudante e usuário, respectivamente.

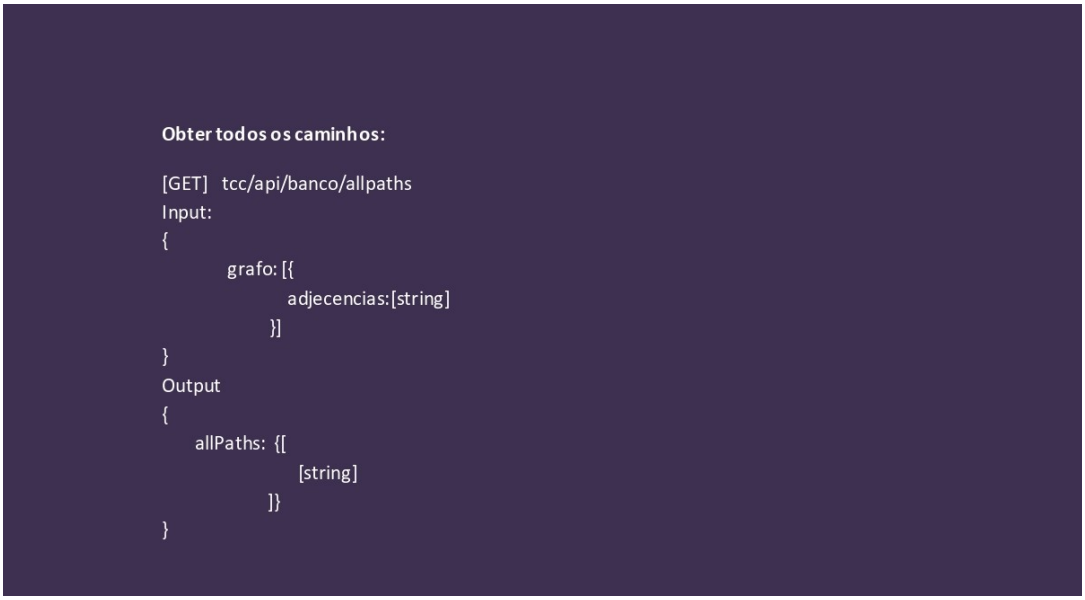
Figura 19 – Response do Método doisTermos



### 4.2.3 Caso de teste - método todos os caminhos

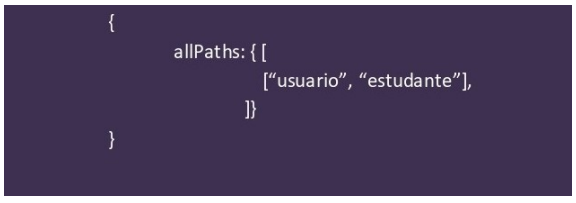
O método todosOsCaminhos(allPaths), conforme mostra a 20, tem como entrada um grafo que é disponibilizado pelo conversão da lista de tabelas existentes na base de dados para um grafo. Esse grafo é construído com base na representação de lista de adjecências. Ao receber tal grafo como entrada, o método allPaths, se encarrega de percorrer todos os caminhos que que ligam o termo1 ao termo2 no grafo. Esses caminhos, são criados através dos relacionamentos existentes nas tabelas intermediárias entre os dois termos digitados, que representam duas tabelas na base de dados. Ele trabalha em conjunto com o métodos doisTermos, que fica responsável por tratar e verificar se os dois termos digitados pelo usuário são tabelas válidas na base de dados, e portanto, já recebe esses dois dados tratados para serem percorridos.

Figura 20 – Representação em JSON do Método Todos os Caminhos




O exemplo prático do método todos os caminhos está na figura 21, na qual é mostrado um exemplo de todos os caminhos que foram percorridos.

Figura 21 – Response do Método Todos os Caminhos

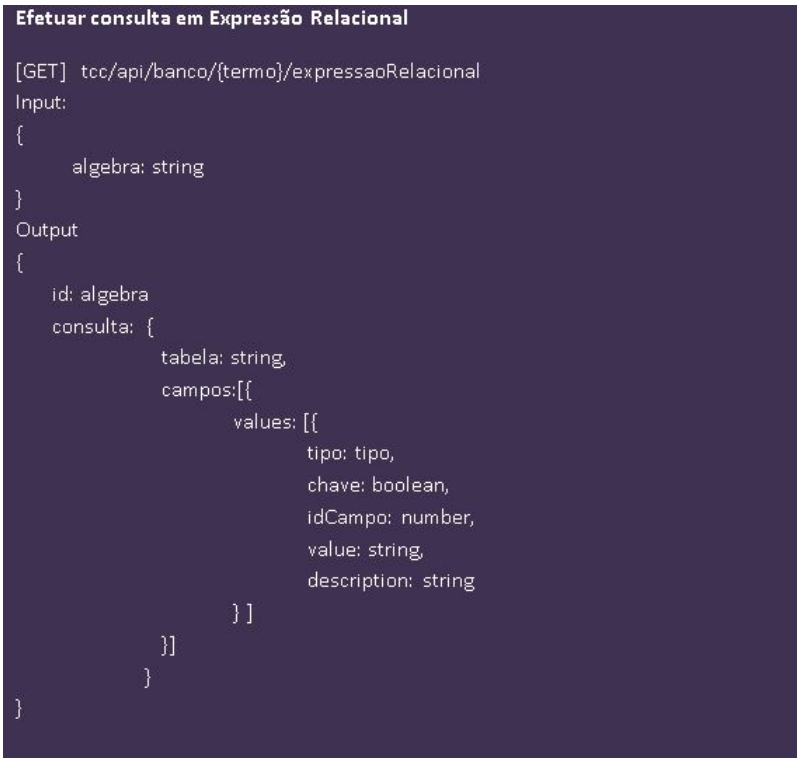



### 4.2.4 Caso de teste - método expressãoRelacional

O principal método da API é o de expressão relacional. Ele recebe uma consulta no formato de álgebra relacional do front end e faz a consulta na base de dados, retornando as informações que o front end solicitou. A saída JSON é uma lista com as informações relacionadas

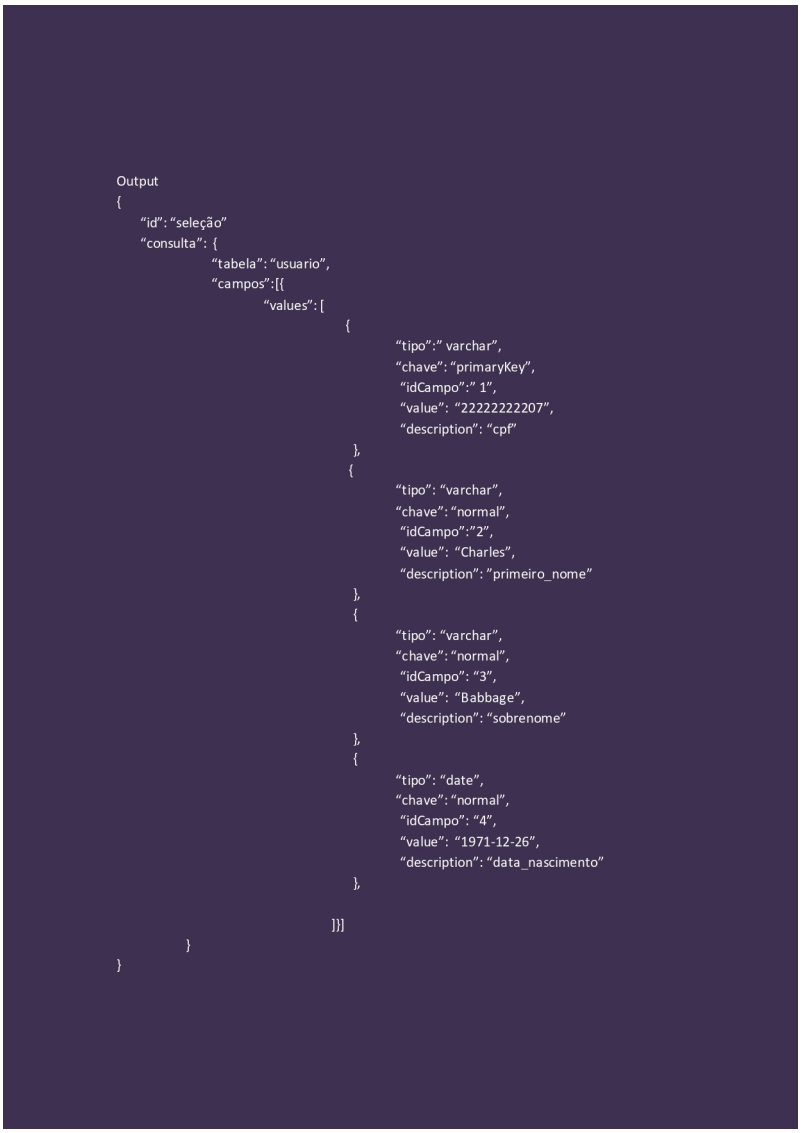
as tabelas, colunas e valores passados como atributos na consulta. A estrutura JSON de entrada e saída, bem como a URL, em que o serviço é disponibilizado pode ser vista na figura 22.

Figura 22 – Representação JSON do Método Expressão Relacional




O primeiro exemplo de álgebra relacional apresentado é o da seleção. Ele filtra os dados da tabela de acordo com um parâmetro informado. Na 23, foi feita a seleção de um dado da tabela "usuário"onde a coluna cpf é igual a 22222222207.

Figura 23 – Response do Método Expressão Relacional




### 4.2.5 Caso de teste - método todasAsTabelas

O método todasAsTabelas (allTables) é disponibilizado através da API, por uma requisição POST, na qual recebe inicialmente os dados do front end necessários para a conexão com o banco de dados desejado. Esse Método é executado logo no início e além de se conectar já fornece todas as tabelas da base de dados para o front end fazer as interações com o usuário. Na figura 24, é mostrado a estrutura JSON desse método.

Figura 24 – Representação JSON do Método Todas as Tabelas

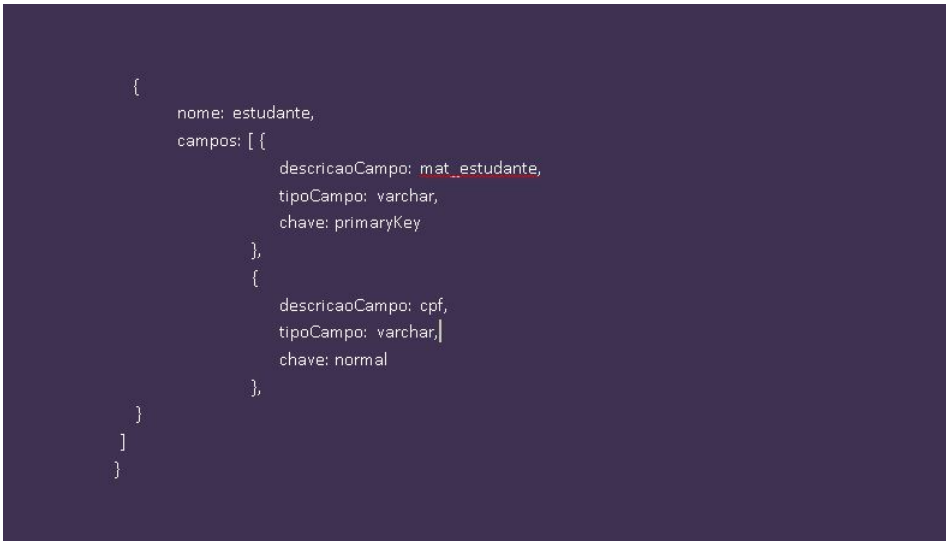

Nesse método, após informados o nomeBanco, idBanco do banco e a senha necessária para conexão, ele salva essa conexão para utilização na interação com o usuário e faz a consulta que retorna todas as tabelas na base de dados informada. Como retorno ele exibe o nome de cada tabela e todos os campos de cada uma.

A lista de campos possui a descricaoCampo, que contem o nome de cada coluna da tabela, o tipoCampo, que contem o tipo dessa coluna e o atributo "chave"que será carregado com o tipo de chave que essa coluna representa, se é chave primária, estrangeira ou normal.

Na figura 25 e a figura 26, são mostrados exemplos práticos do retorno do método allTables passando como parâmetro de entrada os atributos de nome, id e senha da base de dados do sistema de login do estudante. No retorno, é possível ver as tabelas usuário e estudante bem como todas as repectivas colunas de cada uma delas.

Figura 25 – Response do Método Todas as Tabelas


Figura 26 – Response do Método Todas as Tabelas



# 5
# Conclusão

Esse trabalho trouxe como objetivo principal auxiliar um chatbot na busca por informações a qualquer base de dados utilizando a linguagem natural do usuário. Dessa forma foi possível criar uma API que traduz os comandos que são fornecidos por um chatbot, de álgebra relacional para SQL e efetua consulta em qualquer base de dados relacional.

Para isso, foi necessário criar alguns métodos e um compilador, que ficarão disponíveis para a comunicação via http e utilizando o formanto JSON para a comunicação com o chatbot. Esse compilador foi baseado no trabalho feito por Bilecki e Kalempa (2015) e os métodos elaborados foram necessários para efetuar as consultas dos termos e conjuntos de termos relacionados a cada base de dados. Esses termos são fornecidos pelo chatbot, que tem como dados de entrada a linguagem natural do usuário. Após o tratamento necessário, esses termos e conjunto de termos servem como comandos de entrada para a API criada por esse trabalho. Além dessas entradas, consultas no formato de álgebra relacional, também fornecidas pelo chatbot, são convertidas em SQL. Essas consultas em álgebra relacional são montadas pelo chatbot e utilizadas pela API.

Como resultado, foi possível trazer informações sobre qualquer base de dados a ser conectada na API de forma simples e flexível. Fornecida também, uma interface de comunicação entre as bases de dados desejadas e algum chatbot que utilize o protocolo de comunicação HTTP com o formato JSON e que tenha como dados de saída alguma consulta em álgebra relacional. Para futuros trabalhos, fica a possibilidade de incrementar o retorno das consultas de junção e junção natural a API, já que atualmente ele está trazendo apenas consultas simples. Além disso, é possível criar uma portabilidade entre as diferentes produtos de sistemas de gerenciamento de base de dados, onde as conexões sejam feitas não apenas com o SGBD PostgreSQL, mas com todas as tecnologias de SGBD's existentes.

# Referências


AGOSTINHO, C. Interface em linguagem natural para banco de dados: uma abordagem prática. *Universidade Federal de Santa Catarina - UFSC*, p. 104, Agost 2003. Citado na página 10.

AHO ALFRED V., L. et al. *Compiladores: Princípios, técnicas e ferramentas*. [S.l.]: Pearson Universidades, 2008. ISBN 8588639246. Citado na página 14.

ALKHALIFAH, D. d. V. T. Relational algebra interpreter. *International Conference on Advanced Information and Communication Technology for Education (ICAICTE 2014)*, Atlantis Press, p. 6, 2014. Citado 2 vezes nas páginas 14 e 15.

APPEL, A. P.; TRAINA, J. C. idfql - uma ferramenta de apoio ao processo de ensino-aprendizagem da álgebra relacional baseado no construcionismo. *Instituto de Ciências Matemáticas e de Computação-USP. São Carlos, SP*, 2004. Citado na página 15.

BILECKI; KALEMPA. Easyra uma ferramenta para tradução de consultas em álgebra relacional para sql. *Universidade do Estado de Santa Catarina (UDESC) - São Bento do Sul - SC - Brasil*, p. 10, April 2015. Citado 6 vezes nas páginas 11, 14, 15, 19, 24 e 37.

BOHLE, S. "plutchik": artificial intelligence chatbot for searching ncbi databases. *Journal of the Medical Library Association*, v. 106, n. 4, p. 501–503, October 2018. Citado na página 15.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier, 2004. ISBN 8535212736. Citado na página 13.

DELAMARO, M. E. *Como Construir um Compilador - Utilizando Ferramentas Java*. [S.l.]: Novatec, 2004. ISBN 8575220551. Citado 2 vezes nas páginas 14 e 15.

ELMASRI, R.; NAVANTHE, S. B. *Fundamentals of Database Systems*. [S.l.]: Pearson, 2015. ISBN 0133970779. Citado 5 vezes nas páginas 8, 13, 14, 18 e 20.

GARTNER. Gartner customer 360 summit 2011. *Gartner Customer 2011*, p. 9, Agost 2011. Citado na página 10.

GESSER, C. E. Gals: Gerador de analisadores léxicos e sintáticos. *UFSC*, p. 150, 2003. Citado na página 15.

IFOOD. *Como funciona o Uber Eats*. 2022. Disponível em: <https://about.ubereats.com/br/en/?_ga=2.140684978.1625281961.1645747993-980522098.1645747993>. Citado na página 9.

IMMAGIC. *Application programming interface*. 2015. Disponível em: <https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120623A.pdf>. Citado na página 13.

LAUTERT, L. R. Implementac¸ ~ao de um simulador de consultas em ´algebra relacional. *TCC (Graduação) - Curso de Ciência da Computacão, Departamento de Centro de Tecnologia, Universidade Federal de Santa Maria, Santa Maria*, p. 46, 2010. Citado na página 15.

LOUDEN, K. C. *compiladores:princípios e práticas*. [S.l.]: Thomson, 2004. ISBN 8522104220. Citado na página 15.

MONTEIRO, J. M.; LIFSCHITZ, S.; BRAYNER Ângelo. Extraindo metadados de sgbds. *PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIR*, 2007. Citado na página 16.

NAZARETH, L. d. S. Sistema para consultas sobre banco de dados relacional baseado em palavras-chave. *Pontíficia Universidade Católica do Rio de Janeiro*, p. 13, April 2008. Citado na página 10.

PAES, E. L. Ferramenta didática para o ensino de álgebra relacional. *Technical report-Universidade Federal de Santa Catarina - Centro Tecnológico. Departamento de Informática e Estatística.*, 2004. Citado na página 15.

PAPADAKIS, N.; KEFALAS, P.; STILIANAKAKIS, M. A tool for access to relational databases in natural language. *Data & Knowledge Engineering*, v. 38, n. 6, p. 7894–7900, June 2011. Citado na página 15.

PRATES, A. et al. Programar - ferramenta para auxiliar o ensino em algebra ´ relacional. *Universidade Federal de Minas Gerais - UFMG*, v. 13, n. 45, p. 499–510, 2013. Citado na página 15.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de Gerenciamento de Banco de Dados*. [S.l.]: McGraw Hill, 2008. Citado na página 14.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de Banco de Dados*. [S.l.]: Elsevier, 2012. ISBN 8535245359. Citado na página 13.

SINGH, G.; SOLANKI, A. An algorithm to transform natural language into sql queries for relational databases. *Selforganizology*, v. 3, n. 3, p. 100–116, September 2016. Citado na página 15.

TAKAI, O. K.; ITALIANO I. C. FERREIRA, J. Introdução a banco de dados. *DCC-IME-USP*, v. 1, n. 1, p. 124, fevereiro 2005. Citado na página 18.

UBER. *Como usar o app da Uber*. 2022. Disponível em: <https://www.uber.com/br/pt-br/about/how-does-uber-work/>. Citado na página 9.

UBEREATS. *Como funciona o Uber Eats*. 2022. Disponível em: <https://about.ubereats.com/br/en/?_ga=2.140684978.1625281961.1645747993-980522098.1645747993>. Citado na página 9.

VAIDYA; AMBAD; BHOSLE. Industry 4.0 – a glimpse. *Elsevier*, v. 34, n. 20, p. 233–238, February 2018. Citado na página 8.

WAZE. *Como funciona o Uber Eats*. 2022. Disponível em: <https://www.waze.com/about>. Citado na página 9.

# Apêndices

# APÊNDICE A

# GRAMÁTICA E EXPRESSÕES REGULARES DA FERRAMENTA

**Expressões regulares:**

L : [A-Za-z]

D : [0-9]

WS: [\ \t \n \r]

ASIMPLES: “”

UNDERLINE: “_”

PONTO:“.”

D1: [0-2]

D2: [0-1]

H1: [0-5]

H2: [0-3]

AS:“*”

COMMENTOL:#.* \n

COMMENTML:“/*”[∧ “*”]* “*/”n

OUTROTIPO: “*/”n

**Tokens:**

//EXPRESSOES

ID : ({L} | {D})* {UNDERLINE}* ( {L} | {D} )*

literal : {ASIMPLES} ( {L} | {D} ) ( {L} | {D} | “.”| “,”| “:”| “;”| + | − | | “/”| | “(”| “)”| “ ”)* {ASIMPLES}

inteiro : {D} {D}*

float : {D} {D}* “.”{D}*

hora : ( {D1} {D} | “2”{H2} ) “:”{H1}{D} “:”{H1}{D}

data : ( {D1} {D} | “3”{H2} ) “/”( “0”{D} | “1”{D1}) “/”

{D}{D}{D}{D}

campo: ({L} | {D})* {UNDERLINE}* ( {L} | {D} )* “.”({L}|{D})*

{UNDERLINE}* ({L}|{D})* {AS}?

//IGNORAR ESPAÇO EM BRANCO

:{WS}*

//IGNORAR COMENTÁRIO DE UMA LINHA

:![{COMMENTOL}

//IGNORAR COMENTÁRIO DE MÚLTIPLAS LINHAS

:![{COMMENTML}|{OUTROTIPO}

//VARIANTES DA ID

sel = ID : “sel”

proj = ID : “proj”

proj2 = ID : “proj2”

prdcart = ID : “prdcart”

union = ID : “union”

intersect = ID : “intersect”

diff = ID : “diff”

tjoin = ID : “tjoin”

njoin = ID : “njoin”

fojoin = ID : “fojoin”

div = ID : “div”

rjoin = ID : “rjoin”

ljoin = ID : “ljoin”

reb = ID : “reb”

atrib = ID : “atrib”

and = ID : “and”

or = ID : “or”

not = ID : “not”

dif = ID : “dif”

mou = ID:“mou”

meou = ID :“meou”

null = ID: “null”

delta = ID:“delta”

//SIMBOLOS PREDEFINIDOS

“(”

“)”

“,”

“.”

“>”

“,”
“|”
“=”
“|>”
“-”
“{”
“}”

**Não-terminais:**
<COMANDO>
<SELECAO>
<SELECAOOU>
<PROJECAO>
<PROJECAORN>
<RSOLO>
<RSOLO1>
<DIVISAO>
<LISTAINSERCAO>
<TESTE1>
<PROJECAO1>
<RELACAOF1>
<RELACAOF2>
<RELACAOF3>
<RELACAOEP>
<RELACAOPROJ>
<RENOMEARID>
<RENOMEARID2>
<RENOMEARID3>
<CONJUNTOS>
<LIGACAOC>
<BINARIO>
<BINARIOS>
<BINARIOSS>

<OPERACAO>
<TABELA>
<ATR1>
<ATR2>
<ATR3>
<ATR4>
<ATR41>
<ATR5>
<COMANDOSIMPLES>
<PR>
<CONTINUACAO>
<PL>
<MNT>
<CONDICAO>
<CP1> //campo e redireciona pro 2
<CP2> // campo2
<CP3> // parte 3 condicao
<CP4> //parte 4
<CP5> //parte 5
<NNOT>
<CAMPO>
<OPERADOR>
<VALOR>
<LOGICO>
<LISTA_VALORES>
<VALORES2>
<LISTA_VALORESAT>
<VALORES3>
<LISTA_VALORESMNT>
<VALORES4>
<LISTA_VALORESPROJ2>
<VALORES5>
<LISTA_VALORESPROJRN>

<VALORES6>
<LISTA_VALORESDELTA>
<VALORES7>

**Gramática:**

//Comando principal da gramática
<COMANDO> ::= <SELECAO>
| <PROJECAO>
| <PROJECAORN>
| <ATR1>;
//Comandos
<SELECAO> ::= sel #1 <CONDICAO> “(”<RELACAOF1>;
<PROJECAO> ::= proj #10 <LISTA_VALORES> “(”<RELACAOF1>
<RELACAOEP> <CONJUNTOS> <CONTINUACAO>;
<PROJECAO1> ::= proj #21 <LISTA_VALORES>;
<PROJECAORN> ::= reb #34 <RSOLO1>;
<RSOLO1> ::= “(”<LISTA_VALORESPROJRN>“)”“(”<PROJECAO>“)”|
<RSOLO>;
<RSOLO> ::= ID #36 “(”ID #37 “)”;
<RELACAOF1> ::= ID #6 <BINARIO>
| <RENOMEARID> “(”ID #6 “)”<BINARIO>|î;
//Relação proj
<RELACAOPROJ> ::= ID #6
| <RENOMEARID2> “(”ID #6 “)”
|î;
//Relação forma 2
<RELACAOF2> ::= ID #15 “)”
| <RENOMEARID2> “(”ID #15 “)”;
| <SELECAO>
| <PROJECAO> <PR>;
//Relação Forma 3
<RELACAOF3> ::= ID #6 “)”;

```
//Relação Especial Projeção
<RELACAOEP> ::= <SELECAO>“)”|î;
//Continua comando de conjunto
<CONTINUACAO>::= ID #29 |<DIVISAO> proj #31 <LISTA_VALORESPROJ2>
“(”<RELACAOPROJ>|î;
<DIVISAO> ::= div #33 | î;
//Binário
<BINARIO> ::= prdcart #9 <RELACAOF2>
| <CONDICAO> <OPERACAO>
| njoin #9 <RELACAOF2>
| “)”;
<BINARIOSS> ::= prdcart #9 <RELACAOF2>
| njoin #9 <RELACAOF2>
|î;
<OPERACAO> ::= tjoin #9 <RELACAOF2>
| fojoin #9 <RELACAOF2>
| rjoin #9 <RELACAOF2>
| ljoin #9 <RELACAOF2>;
<BINARIOS> ::= prdcart #14
| tjoin #14 <CONDICAO>
| fojoin #14 <CONDICAO>
| rjoin #14 <CONDICAO>
| ljoin #14 <CONDICAO>
| njoin #14
| div #33
| î;
//Somente conjunto
<CONJUNTOS> ::= union #22
| diff #23
| intersect #24
| î;
//Renomear ID
<RENOMEARID> ::= reb ID #7| î;
```

//Renomear ID2
<RENOMEARID2> ::= reb ID #16| î;
//Seleção ou ID
<SELECAOOU> ::= <SELECAO> | ID #43;
//Atribuição e manutencao
<ATR1> ::= delta #42 “{“<LISTA_VALORESDELTA> “}”“(”<SELECAOOU> “)”
|ID #17 <BINARIOSS><CONJUNTOS><CONTINUACAO><ATR2> <ATR3>;
<ATR2> ::= #25 “(”<LISTA_VALORESAT> “)”| î;
<ATR3> ::= atrib #39 <ATR4> <ATR41>;
<ATR4> ::= sel #20 <CONDICAO> “(”<RELACAOF1>
| <PROJECAO1> “(”<RELACAOF1><RELACAOEP>
|<PROJECAORN>
|î;
<ATR41> ::= ID #18 <CONJUNTOS> <BINARIOS> <MNT> <ATR5> | î;
<ATR5> ::= ID #19 | “)”| î;
<LISTAINSERCAO> ::= “(”<LISTA_VALORESMNT> “)”<TESTE1>;
<TESTE1> ::= “,”#26 <LISTAINSERCAO> | î;
//Manutenção
<MNT> ::= <SELECAO>
| proj #28 <LISTA_VALORESMNT> “(”<RELACAOF1> <RELACAOEP>
<CONJUNTOS> <CONTINUACAO>
| “”<LISTAINSERCAO> “”
|î;
//OPERADOR CONDIÇÃO NOT
<NNOT> ::= not #38| î;
//Condicao 5 fases
<CONDICAO> ::= <NNOT><PL> <CP1>;
<CP1> ::= <CAMPO>#2 <CP2>;
<CP2> ::= <OPERADOR> #3 <CP3>;
<CP3> ::= <VALOR> #4 <PR> <CP4>;
<CP4>::= <LOGICO> #5 <CP5>
| î;
<CP5> ::= <NNOT><PL> <CP1>;

<PR> ::= “)”;

<PL> ::= “(”;

//Campo ou é so um ID ou campo

<CAMPO> ::= ID | campo;

//Tipo lógico

<LOGICO> ::= and

| or

| not;

//Tipo de valor que pode ocorrer

<VALOR> ::= campo

| ID

|literal

| inteiro

| hora

| data

| float

| null;

//Operadores

<OPERADOR> ::= “=”

| “<>”

| “<”

| mou

| meou

| “>”

| dif;

//Lista de valores no formato valor1,valor2,valor3

<LISTA_VALORES> ::= <CAMPO> #12 <VALORES2>;

<VALORES2> ::= “,”<LISTA_VALORES> | î; //epsilon significa vazio, ou exceção a esta regra

<LISTA_VALORESAT> ::= <CAMPO> #26 <VALORES3>;

<VALORES3> ::= “,”<LISTA_VALORESAT> | î; //epsilon significa vazio, ou exceção a esta regra

<LISTA_VALORESMNT> ::= <VALOR> #26 <VALORES4>;

<VALORES4> ::= “,”<LISTA_VALORESMNT> | î; //epsilon significa vazio, ou exceção a esta regra

<LISTA_VALORESPROJ2> ::= <CAMPO> #32 <VALORES5>;

<VALORES5> ::= “,”<LISTA_VALORESPROJ2> | î; //epsilon significa vazio, ou exceção a esta regra

<LISTA_VALORESPROJRN> ::= <CAMPO> #35<VALORES6>;

<VALORES6> ::= “,”<LISTA_VALORESPROJRN> | î; //epsilon significa vazio, ou exceção a esta regra

<LISTA_VALORESDELTA> ::= <CAMPO> #40 atrib <VALOR> #41 <VALORES7>;

<VALORES7> ::= “,”<LISTA_VALORESDELTA> | î; //epsilon significa vazio, ou exceção a esta regra