



UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
DEPARTAMENTO DE COMPUTAÇÃO

# **Chatbot para Realização de Consultas em Bancos de Dados Relacionais**

Trabalho de Conclusão de Curso

Victor Carity Feitosa Silva Carneiro



São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
DEPARTAMENTO DE COMPUTAÇÃO

Victor Carity Feitosa Silva Carneiro

## **Chatbot para Realização de 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): André Britto de Carvalho

São Cristóvão – Sergipe

2022

*A minha família,  
aos meus amigos,  
aos meus professores.*

*Obrigado pelo amor, suporte e compreensão.*

*"Velho é o homem quando nasce, e jovem,  
jovem para sempre depois disso."  
- Livro de Taliesin*



# Resumo

Bancos de dados relacionais, no presente, são a forma mais comum de armazenamento de dados, no entanto a recuperação desses dados requer conhecimento técnico de ferramentas como o SQL. Esse conhecimento técnico, grande parte das vezes, não é de interesse do usuário final. Neste trabalho construímos um *chatbot* para a realização de consultas em bancos de dados relacionais, por meio da linguagem natural textual. O *chatbot* é restrito a: consultar os dados de uma tabela, selecionar um campo de uma tabela, filtra o dado selecionado de uma tabela, e relacionar uma tabela com outra tabela. Para que tudo funcione de forma direcionada o *chatbot* teve de adotar um comportamento inquisitivo, onde perguntas são feitas ao usuário com o intuito de guiá-lo as soluções desejadas. Para que a construção do *chatbot* fosse possível, pesquisamos na literatura trabalhos semelhantes, obtendo destes o conhecimento para solidificar a fundamentação teórica necessária para a construção da solução aqui proposta. Também separamos e listamos os materiais necessários para construção e execução da ferramenta. Para validarmos a execução da interface, foram separadas consultas semelhantes em linguagem natural textual e SQL e comparadas ao resultado do *chatbot*.

**Palavras-chave:** Banco de Dados, Modelo Relacional, Interface de Linguagem Natural, Interface, Chatbot, Bot, Processamento de Linguagem Natural.

# Lista de ilustrações

Figura 1 – Análise Semântica para SQL . . . . .	16
Figura 2 – Análise Semântica para Expressão Relacional . . . . .	16
Figura 3 – Tabela alunos para Exemplos de Expressões Relacionais . . . . .	18
Figura 4 – Exemplo da Operação Unária Relacional SELECT . . . . .	18
Figura 5 – Exemplo da Operação Unária Relacional PROJECT . . . . .	19
Figura 6 – Exemplo da árvore de consulta gerada pelo NaLIR . . . . .	20
Figura 7 – Diagrama de Sequência do Funcionamento da Interface . . . . .	22
Figura 8 – Demonstração da Inteface Finalizada . . . . .	23
Figura 9 – Visão Geral do Projeto . . . . .	24
Figura 10 – Interface de Linguagem Natural . . . . .	24
Figura 11 – Fluxo de Estados do Chatbot . . . . .	25
Figura 12 – Classe Principal da Aplicação . . . . .	28
Figura 13 – Código do Express inicializando a aplicação . . . . .	29
Figura 14 – Router que identifica a reposta do index.html na na rota '/' . . . . .	29
Figura 15 – Script to index.html da rota '/' . . . . .	31
Figura 16 – Código que transforma a linguagem natural em tokens POS . . . . .	32
Figura 17 – Código do Estado Inicial . . . . .	34
Figura 18 – Código do Estado Inicial de Análise da Mensagem . . . . .	35
Figura 19 – Código do Estado de Escolha de uma Tabela . . . . .	36
Figura 20 – Código do estado Análise da Mensagem da Escolha de uma Tabela . . . . .	37
Figura 21 – Etapas da validação . . . . .	40
Figura 22 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada . . . . .	41
Figura 23 – NLIDB sendo utilizada para obter os dados da tabela endereco SELECT * FROM endereco . . . . .	41
Figura 24 – log da NLIDB sendo utilizada para obter os dados da tabela endereco SELECT * FROM endereco . . . . .	42
Figura 25 – Resultado da Expressão Relacional . . . . .	42
Figura 26 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada . . . . .	43
Figura 27 – NLIDB sendo utilizada para obter os dados da tabela estudante SELECT * FROM estudante . . . . .	43
Figura 28 – Resultado da Expressão Relacional . . . . .	44

Figura 29 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada . . . . .	44
Figura 30 – NLIDB sendo utilizada para expressão equivalente a <code>SELECT cep FROM endereco</code> . . . . .	45
Figura 31 – Resultado da Expressão Relacional . . . . .	46
Figura 32 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada . . . . .	46
Figura 33 – NLIDB sendo utilizada para expressão equivalente a <code>SELECT * FROM endereco WHERE cep = 25123321</code> . . . . .	47
Figura 34 – Resultado da Expressão Relacional . . . . .	47
Figura 35 – Tabela utilizando o filtro para <code>SELECT * from estudante WHERE mc != 5</code> .	48
Figura 36 – NLIDB sendo utilizada para expressão equivalente a <code>SELECT * from estudante WHERE mc != 5</code> . . . . .	48
Figura 37 – Tabela utilizando o JOIN . . . . .	49
Figura 38 – NLIDB sendo utilizada para expressão equivalente a um JOIN . . . . .	49

# Lista de abreviaturas e siglas

API	Application Programming Interface
DBMS	Database Management System
E-R	Entity-Relationship
IT	Information Technology
NLIDB	Natural Language Interfaces to Query Databases
NLP	Natural Language Processing
NQL	Natural Language Query
POS	Part-of-speech
RDBMS	Relational Database Management Systems
SP	Semantic Parsing
SQL	Structured Query Language
TBL	Transformation Based Learning
WPT	wink-pos-tagger
UML	Unified Modeling Language

# Sumário

<b>1</b>	<b>Introdução</b>	<b>11</b>
1.1	Motivação	12
1.2	Objetivo Geral	12
1.3	Metodologia	13
1.3.1	Métodos	13
1.3.2	Materiais	13
1.4	Organização do Trabalho	14
<b>2</b>	<b>Fundamentação Teórica</b>	<b>15</b>
2.1	Processamento de Linguagem Natural	15
2.1.1	PLN e SQL	15
2.1.2	Análise Semântica	16
2.1.2.1	Semantic Parsing para SQL	17
2.1.2.2	Semantic Parsing para Expressão Relacional	17
2.2	Chatbot	17
2.3	Álgebra Relacional	18
2.3.1	Operação Unária Relacional SELECT	18
2.3.2	Operação Unária Relacional PROJECT	19
2.4	Trabalhos Relacionados	19
2.4.1	Spider	19
2.4.2	SODA	19
2.4.3	NaLIR	20
2.4.4	ATHENA++	20
2.5	Considerações	21
<b>3</b>	<b>Projeto</b>	<b>22</b>
3.1	Visão Geral	24
3.1.1	Arquitetura da NLI	24
3.1.2	Fluxo de Estados do Chatbot	25
3.2	Construção do Protótipo	25
3.2.1	Testes, Cobertura e Qualidade de Código	27
3.2.2	NodeJS	27
3.2.3	Express	29
3.2.4	SocketIO	30
3.2.4.1	View	30
3.2.5	wink-post-tagger	32

3.2.6	Axios	32
3.2.7	Heroku	32
3.2.8	Objeto Expressão Relacional	33
3.2.9	Estados	33
3.2.9.1	Estado Começo	34
3.2.9.2	Estado Escolha uma tabela	36
3.2.10	Estados de Confirmação	37
3.2.11	Estados de Escolha de Filtros	38
3.2.12	Consultar uma Tabela do Banco de Dados	38
3.2.13	Filtrar Resultados de uma Tabela do Banco de Dados	38
3.2.14	Relacionar os Dados da Consulta com Outra Tabela do Banco de Dados	39
3.2.15	Estado de Fim	39
<b>4</b>	<b>Validação</b>	<b>40</b>
4.1	Selecionar uma Tabela	41
4.2	Selecionar um campo de uma tabela	44
4.3	Filtrar os dados de uma tabela	46
4.4	Relacionar uma tabela com outra tabela	48
<b>5</b>	<b>Conclusão</b>	<b>50</b>
	<b>Apêndices</b>	<b>52</b>
	<b>APÊNDICE A Instruções para Executar a Aplicação Localmente</b>	<b>53</b>
	<b>Referências</b>	<b>54</b>

# 1

## Introdução

Neste trabalho de conclusão de curso, abordamos sistemas computadorizados de bancos de dados. Esses sistemas são programas de computador de propósito geral que permitem usuários criarem e manterem banco de dados. Chamados de SGBD (*sistemas de gerenciamento de bancos de dados*). [Elmasri e Navanthe \(2015\)](#) escreveram na introdução de seu livro as seguintes palavras: "Bancos de Dados e Sistemas de Bancos de Dados são um componente essencial da sociedade moderna". Mas, não basta somente armazenar essa quantidade massiva de informação diariamente, é preciso recuperá-la e dar significado a ela.

O modelo de dados relacionais, que no presente é o modelo mais utilizado, define a estruturação de bancos de dados relacionais. Esse modelo tem sua base na álgebra relacional, é dele que temos a divisão dos dados em tabelas, onde cada tabela é um conjunto de tuplas organizada por seus tipos de dados que são colunas e as tuplas que são as linhas. Essas tabelas são representadas por uma coleção de relacionamentos ([ELMASRI; NAVANTHE, 2015](#)), responsáveis pelo armazenamento da informação.

No presente, a forma mais comum de recuperarmos essas informações é utilizando SQL (*standard query language*), em português: *linguagem de consulta estruturada*. O SQL é uma linguagem computacional projetada com o objetivo de gerenciar os dados presentes em um banco de dados relacional. Para profissionais de computação armazenar e recuperar a informação pode parecer simples. Porém, ainda é um obstáculo ao usuário comum. Usuários que podem ser desde cientistas em campo que estejam catalogando novas espécies de plantas a um síndico que deseje saber os gastos de água dos moradores do prédio na última semana. Nenhum desses usuários, tem acesso, de forma simples à essas informações.

É difícil que uma pessoa sem conhecimentos computacionais, tenha interesse ou dedique-se ao estudo de linguagens específicas como SQL. Sendo comum em aplicações a construção de telas com formulários de pesquisa para todas as consultas necessárias. O processamento de linguagem natural, ou seja, o desenvolvimento de algoritmos para análise, geração e aquisição

de conhecimento linguístico (CUNNINGHAM, 1999), permite que o computador processe a linguagem natural do usuário. Logo, por que não utilizarmos do PLN para que o usuário consiga realizar consultas em bancos de dados relacionais?

*Chatbots* inquisitivos, conseguem realizar perguntas que direcionam o usuário a possíveis respostas, respostas essas que são oferecidas como entrada em linguagem natural. Na literatura podemos encontrar estudos (RESHMI; BALAKRISHNAN, 2016) onde os autores apresentam a implementação de um *chatbot* inquisitivo para consulta de base de dados. Ou um *chatbot* que funciona por meio de inteligência artificial (BOHLE, 2018). Em outro estudo realizado por YU et al., os pesquisadores fizeram uso do SP (*semantic parsing*), em português *análise semântica*. Uma técnica de processamento de linguagem natural a qual é utilizada para mapear o significado da linguagem natural para a linguagem de consulta estruturada. E ainda podemos observar outros estudos, que utilizam algoritmo para transformar a linguagem natural em linguagem de consulta (SINGH; SOLANKI, 2016) ou que apresentem ferramentas para acessar os dados do banco pela linguagem natural (PAPADAKIS; KEFALAS; STILIANAKAKIS, 2011).

## 1.1 Motivação

*Chatbots* e linguagens de programação que aproximam-se da linguagem natural já foram propostos em outros trabalhos, mas de forma específica (BOHLE, 2018). Por meio do processamento de linguagem natural ou NLP (*Natural Language Processing*) (JOHN; YORICK, 2019) é possível construir um robô que opere na estrutura de comunicação humana transcrevendo-a em uma linguagem que pode ser compreendida por todos os bancos de dados relacionais, a álgebra relacional.

Um dos caminhos para essa transcrição é através do processo de tradução das consultas em linguagem natural para uma linguagem intermediária, que em seguida é interpretada para consultas SQL. Para isso, pode ser construído um sistema separado em dois módulos, onde um dos módulos é o responsável pelo PLN para a linguagem intermediária e outro responsável pela interpretação da linguagem intermediária para SQL.

## 1.2 Objetivo Geral

O propósito geral deste trabalho de conclusão de curso é a construção de um *chatbot* inquisitivo, que permita a consulta em bancos de dados relacionais, e funcionará a partir do uso de linguagem humana em forma textual, sendo responsável pelo processamento da linguagem natural. Analisando-a, separando-a e transformando-a em uma expressão que representa a álgebra relacional.

## 1.3 Metodologia

Nesta sessão descrevemos os métodos e materiais utilizados para a construção da solução.

### 1.3.1 Métodos

A construção do *chatbot*, será feita a partir de artigos que dissertam sobre métodos e arquiteturas para a construção do bot, como (RESHMI; BALAKRISHNAN, 2016), e artigos a respeito da quebra da linguagem, e como transformar essa análise ortográfica em expressões de álgebra relacional que possam ser reconhecidas pelo computador como (FRANK; CHUN-LING, 2008). No momento, a construção é restrita para algumas formas de consultas: selecionar uma tabela, selecionar um campo numa tabela, filtrar os dados de uma tabela e relacionar uma tabela com outra tabela.

Implementando esta solução estaremos oferecendo uma comunicação entre o usuário comum e o SGBD, uma nova forma de consulta que envolve bancos de dados relacionais. Essa ponte entre usuário comum e a máquina é importante, pois retira a necessidade de um profissional de computação para a realização de tarefas rotineiras como obter um conjunto de informações para geração de relatório. Possibilitando ao usuário comum mais conforto em seu dia a dia, e permitindo ao profissional de tecnologia da informação (TI) disponibilidade para outras tarefas.

O *chatbot* tem um comportamento inquisitivo, o qual é utilizado para a construção do objeto de expressão na álgebra relacional. Esse objeto permitirá: consultar uma tabela de um banco de dados, filtrar os resultados de uma tabela de um banco de dados e relacionar uma tabela com outra em um banco de dados. Faz se necessário também, que o mesmo comunique-se com uma API que possa ler e executar o objeto de expressão relacional. Uma API assim pode ser encontrada no Trabalho de Conclusão de Curso de *Iraildo da Costa Carvalho* (CARVALHO, 2022).

Uma vez que o *chatbot* tenha sido construído, validaremos a interface observando se a expressão objeto relacional obtida é semelhante ao resultado esperado na mesma consulta realizada em SQL e em linguagem natural. Essa validação será realizada para os tipos de consulta restritos ao *chatbot* mencionados anteriormente.

### 1.3.2 Materiais

Para o desenvolvimento da aplicação é necessário utilizarmos alguns materiais já existentes como o NodeJs, possibilitando o acesso a soluções que não são o objetivo deste trabalho, mas oferecem a base para o resultado desejado. Como abordado anteriormente, o objetivo do trabalho é desenvolver um chatbot para consultar bancos de dados relacionais. Para isso o foco do desenvolvimento é dirigido as maneiras como a linguagem natural é analisada e estruturada para que possa ser compreendida pelo computador, evitando que grandes quantidades de tempo

de desenvolvimento sejam alocado para soluções já existentes como a execução de APIs *web*. Os materiais são:

1. **NodeJS** - biblioteca em tempo de execução assíncrona projetada para a construção de aplicações web escaláveis.
2. **Express** - conjunto de funcionalidades para o NodeJS que tem como objetivo facilitar a construção da aplicação web.
3. **SocketIO** - biblioteca que permite comunicação entre cliente servidor, construída em cima do protocolo de websockets.
4. **Página Html** - interface visual para o usuário final realizar a comunicação com a interface.
5. **wink-post-tagger** - biblioteca responsável pelo processamento da linguagem natural textual na aplicação.
6. **axios** - cliente http para o NodeJS para realização de requisições na API.
7. **Heroku** - solução em nuvem para disponibilização do *chatbot* de maneira remota.

## 1.4 Organização do Trabalho

Este trabalho foi organizado em diferente capítulos com a finalidade de apresentar de forma sucinta:

1. **Introdução** - Onde é apresentado o conteúdo deste documento.
2. **Fundamentação Teórica** - Capítulo que apresenta a definição de termos técnicos, conceitos e conhecimentos necessários para a tese.
3. **Projeto** - Capítulo que mostra uma implementação de um *chatbot* inquisitivo que constrói objetos de expressão relacional.
4. **Estudos de Caso** - Apresentação dos resultados do *chatbot* na construção do objeto de expressão relacional.
5. **Conclusão** - O documento é encerrado, acompanhado de considerações e observações finais.

# 2

## Fundamentação Teórica

Neste capítulo são apresentados alguns dos termos técnicos, conceitos e conhecimentos necessários para construção da NLIDB.

### 2.1 Processamento de Linguagem Natural

Processamento de linguagem natural, engenharia de linguagem natural ou linguística computacional significam a mesma coisa, processamento textual (JOHN; YORICK, 2019). Para que computadores executem instruções é preciso que essas instruções estejam em sua própria linguagem de máquina. Assim, para que o computador possa compreender a linguagem natural é preciso transcrever a mesma em um conjunto de instruções que possa ser compreendida pela máquina.

Uma vez que o computador consiga processar a entrada provida pelo usuário, também é necessário dar significado ao input oferecido, esse significado pode ter diferentes formas, como uma expressão lógica ou em uma consulta SQL. Uma maneira para que a significação ocorra é utilizando de uma técnica de processamento de linguagem natural chamada de SP(*semantic parsing*) ou análise semântica (YU et al., 2018).

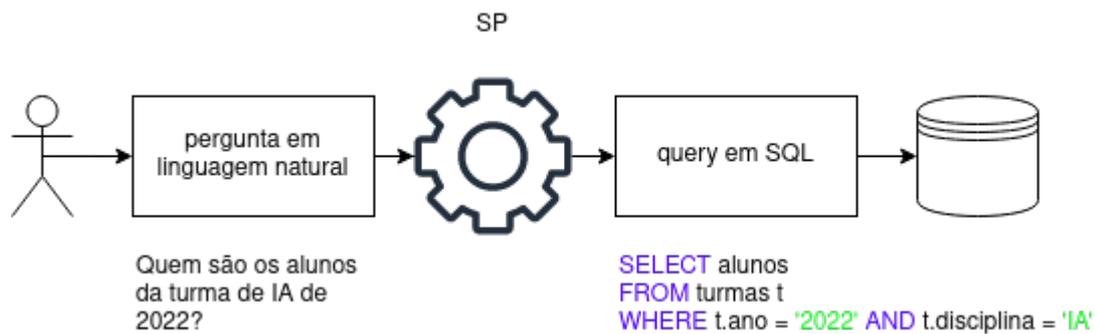
#### 2.1.1 PLN e SQL

Por meio do processamento de linguagem natural, podemos oferecer ao computador uma entrada na linguagem natural e obter como resultado uma saída SQL. Este processo abordado é em estudos como o de SINGH; SOLANKI, YU et al. ou LI; JAGADISH, onde também são citadas diferentes abordagens para a execução deste processamento, como: inteligência artificial ou linguística.

### 2.1.2 Análise Semântica

A análise semântica é o processo de extração do texto que mapeia os enunciados presentes para representações semânticas (KAMATH; DAS, 2018). Representações essas que podem ser compreendidas pelo computador, e costumam ser executadas em um ambiente ou contexto próprio com a finalidade de gerar uma saída desejada. Um exemplo é o mapeamento direto para uma consulta SQL executável, semelhante ao que é apresentado no trabalho de YU et al. pode ser visto na Figura 1.

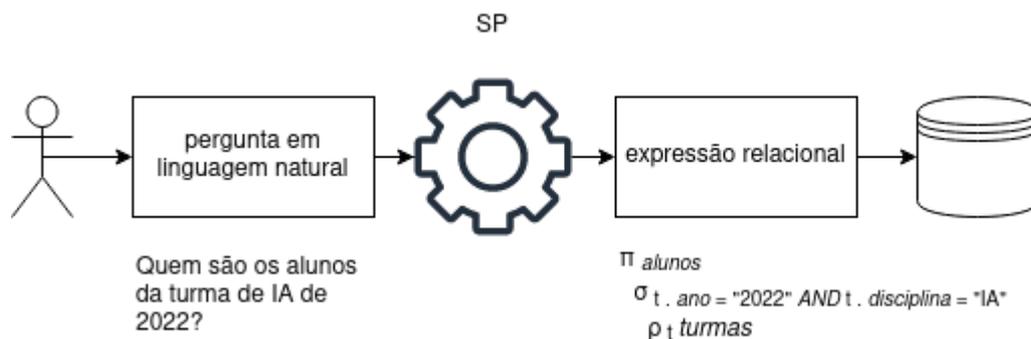
Figura 1 – Análise Semântica para SQL



Fonte: Autores (2022).

Uma outra possibilidade é utilizar a significação dos enunciados para uma linguagem intermediária, a partir da qual podemos organizar a representação semântica para atender a uma necessidade específica. No contexto desse trabalho a linguagem intermediária desejada é a expressão álgebra relacional. Assim, para que cheguemos a esse resultado, utilizamos uma variação do *semantic parsing* onde a expressão relacional é obtida por meio de entradas textuais de linguagem natural extraídas por meio de um *chatbot*, nossa NLI (*natural language interface*).

Figura 2 – Análise Semântica para Expressão Relacional



Fonte: Autores (2022).

### 2.1.2.1 Semantic Parsing para SQL

Compreendendo que a análise semântica é um processo de significação da linguagem natural para uma linguagem intermediária alvo compreendida pelo computador. Podemos observar uma maneira de realizar um mapeamento de linguagem natural para SQL por *SP* analisando o trabalho realizado por [LI; JAGADISH](#).

### 2.1.2.2 Semantic Parsing para Expressão Relacional

De maneira análoga a que foi abordada a sessão anterior podemos, mas sem um mapeamento direto. Aqui faremos uso do *SP* para extrair palavras chaves que vão permitir a construção da expressão relacional, o processo intermediário é descrito com mais detalhes na próxima sessão.

## 2.2 Chatbot

O *chatbot* é uma interface para interações humano computador por meio da linguagem natural, considerado uma interface clássica para interações deste tipo segundo [RESHMI; BALAKRISHNAN](#), ela é responsável por analisar o input textual oferecido em linguagem natural pelo usuário. É uma ferramenta essencial para o desenvolvimento da NLIDB deste projeto, para realizar a análise, o *chatbot* desta tese faz uso de NLP por meio de uma biblioteca construída com base no TBL ou *Transformation Based Learning* ([ERIC, 1995](#)).

Como supracitado, o diálogo em linguagem natural com o *chatbot* é textual no idioma inglês. O usuário, por sua vez, é guiado pelo *chatbot* no que é chamado de comportamento inquisitivo ([RESHMI; BALAKRISHNAN, 2016](#)). Isto é, o *chatbot* tem como output perguntas direcionadas ao usuário, que necessitam de confirmação. As perguntas têm o objetivo de extrair informações específicas na elaboração da expressão relacional. As informações são importantes, também, para garantir que não haja ambiguidade nas consultas.

Durante o diálogo com o programa, as informações extraídas dos inputs textuais do usuário são transformadas para uma classificação POS (*Part-of-speech*) onde aplicamos uma ideia análoga a apresentada por [NAZANIN et al.](#) Ou seja, com a classificação POS extraímos *substantivos*, para tabelas ou campos, e os *adjetivos* para operações lógicas. Essas tabelas, campos e operações constituem um objeto que também é a representação de uma expressão em álgebra relacional.

Uma vez que temos uma expressão em álgebra relacional, podemos enviá-la a API, obtendo resultados e sugestões que são modificamos conforme o input do usuário. Este resultado da API externa nós é retornado na forma de tabelas. As tabelas retornadas pela API, devem indicar a necessidade de informações que ficaram ausentes na consulta, e as relações do resultado da consulta atual. Assim, o bot será capaz de ir para um estado que tenham perguntas para

preencher as informações necessárias e oferecer sugestões em sua interface.

## 2.3 Álgebra Relacional

A álgebra relacional é um conjunto básico de operações para o modelo relacional, estas operações permitem a um usuário especificar consultas como expressões de álgebra relacional. O resultado da consulta é uma nova relação, que pode ser utilizada usando operações da mesma álgebra (ELMASRI; NAVANTHE, 2015). Por tratar-se do conjunto base de operações do modelo relacional, podemos utilizar uma expressão relacional em qualquer banco de dados que siga o modelo relacional.

Utilizaremos a Figura 3, uma tabela de alunos, com as colunas: nome, matricula, media e ano\_de\_entrada, para ilustrar e explicar nas sub-sessões subsequentes algumas das operações básicas da álgebra relacional.

Figura 3 – Tabela alunos para Exemplos de Expressões Relacionais

Table alunos

nome	matricula	media	ano_de_entrada
Fred	COD2020011	7.5	2020
Velma	COD2021034	9.8	2021
Norville	COD2022003	5.3	2022

Fonte: Autores (2022).

### 2.3.1 Operação Unária Relacional SELECT

A operação SELECT é utilizada para escolher um subconjunto de tuplas de uma relação que satisfaça a condição de seleção (ELMASRI; NAVANTHE, 2015). A operação de SELECT da expressão relacional é diferente da operação SELECT do SQL, pois aqui ela funciona como um filtro ou restrição. Observe que na ilustração 4 referente ao SELECT escolhemos os dados da tabela alunos que possuem *media* > 7.

Figura 4 – Exemplo da Operação Unária Relacional SELECT

$\sigma_{media > 7} \text{ alunos}$

nome	matricula	media	ano_de_entrada
Fred	COD2020011	7.5	2020
Velma	COD2021034	9.8	2021

Fonte: Autores (2022).

### 2.3.2 Operação Unária Relacional PROJECT

Conforme explicado por [ELMASRI; NAVANTHE](#), a operação PROJECT seleciona colunas específicas de uma tabela e descarta as demais. No exemplo de PROJECT é definido que queremos apenas a coluna *nome* da tabela *alunos*.

Figura 5 – Exemplo da Operação Unária Relacional PROJECT

$\pi_{\text{nome}} \text{ alunos}$

nome
Fred
Velma
Norville

Fonte: Autores (2022).

## 2.4 Trabalhos Relacionados

Nesta sessão observamos alguns trabalhos já realizados que possuem relação com o objetivo desta tese ou onde as áreas de conhecimento sobrepõem-se. Em seguida, é discursado a respeito dos mesmos, seus objetivos, resultados ou diferenças.

### 2.4.1 Spider

O *Spider* foi projetado como uma solução de análise semântica que converte texto para SQL utilizando conjuntos de dados de grande escala ([YU et al., 2018](#)). A solução utiliza de aprendizagem e testes que verificam múltiplas consultas SQL e diferentes bases de dados. Esse uso de múltiplas bases de dados é, também, o diferencial do *Spider* em relação a outras soluções que costumam ser treinadas e testadas em apenas uma base de dados.

Uma vez que o processo faz uso de suas múltiplas base de dados para treinamento e testes, o modelo consegue generalizar respostas, tanto para novas consultas SQL, como para novas bases de dados. Por depender de diversas bases de dados o *Spider* acaba limitado a necessidade de uma grande base de conhecimento prévio, o que pode dificultar sua aplicação em interfaces com pequenas bases de dados.

### 2.4.2 SODA

*SODA* é uma sigla para *search over data warehouse* que em português quer dizer busca sobre um depósito de dados ([KAMATH; DAS, 2012](#)). Esta solução permite que usuários realizem buscas de maneira análoga a utilizar um buscador como o Google ou Bing, utilizando palavras chaves da "busca" para gerar consultas SQL.

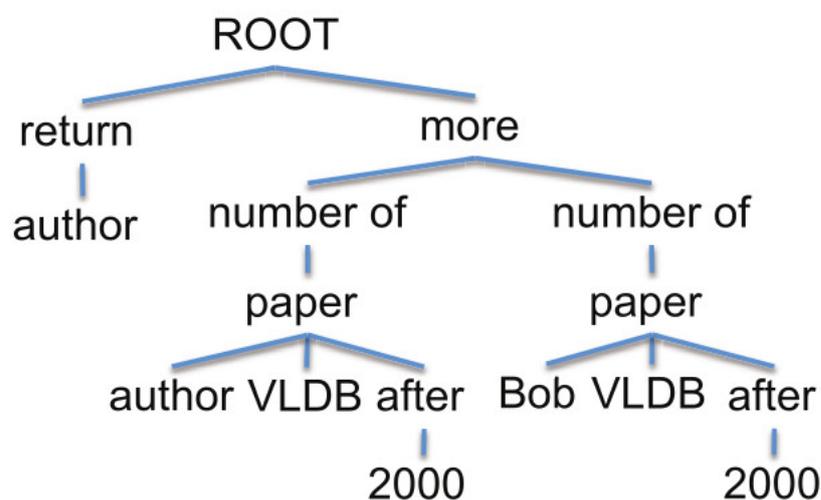
O diferencial desta solução é a maneira como as consultas SQL são geradas, portanto ao invés de fazer uso de processamento de linguagem natural, a transcrição da busca para o SQL acontece por um algoritmo de casamento de padrões. Os metadados ficam armazenados em um grafo que permitem ao *SODA* computar e priorizar uma lista de consultas SQL executáveis que provavelmente atendem ao usuário.

### 2.4.3 NaLIR

*NaLIR*, é uma interface genérica interativa de linguagem natural para realização de consultas em bancos de dados, e é isso que seu nome também significa *natural language interface for relational databases* (LI; JAGADISH, 2016). Similar ao *SODA*, *NaLIR* também toma como base a ideia de campos de busca como forma de limitar a interação do usuário com a interface, mas por meio de *semantic parsing*.

A solução utiliza o *SP* com um comunicador iterativo que espera a confirmação do usuário validando a sua consulta e ajustando a consulta até que o usuário fique satisfeito. Uma vez que a consulta é finalizada, durante o processo é gerado uma árvore que representa a consulta, a qual é encaminhada a um tradutor que realiza a conversão para SQL.

Figura 6 – Exemplo da árvore de consulta gerada pelo NaLIR



Fonte: (LI; JAGADISH, 2016)

### 2.4.4 ATHENA++

O *ATHENA++* é um sistema de ponta à ponta focado em solucionar consultas complexas que envolvem a inteligência do negócio (*BI*) por meio da linguagem natural (SEN et al., 2020) traduzindo-a em consultas SQL aninhadas. Para efetivar esse processo *ATHENA++* combina padrões de linguística com raciocínio utilizando ontologia para detecção e geração de consultas SQL aninhadas.

## 2.5 Considerações

Estudando a base teórica apresentada neste capítulo, oferecemos e reforçamos o conhecimento necessário para a construção de um *chatbot* para consulta de bandos de dados relacionais. Ao analisarmos os trabalhos apresentados na sessão anterior, podemos concluir que existem diferentes formas de se implementar uma *NLIDB*.

O que diferencia uma interface de outra é a maneira como processamos a linguagem natural e como ocorre a interação do usuário com a interface. É possível concluir, também, que a motivação dos trabalhos é semelhante: permitir o acesso do usuário comum as bases de dados sem que exista a necessidade de conhecimentos técnicos prévios.

Ainda que existam soluções testadas e avaliadas, não há um consenso ou convenção sobre qual apresenta a melhor usabilidade para o usuário, nem sobre a melhor solução, o estudo feito por [SEN et al.](#) por exemplo, não apresenta uma taxa de conversão de 100% para as consultas ainda que utilize o *Spider* como *benchmark* e consiga resultados de performance melhor que trabalhos anteriores.

O sistema desenvolvido neste trabalho segue outro caminho para construção de uma *NLIDB*. A interface tem a forma de um *chatbot* com comportamento inquisitivo, que é em parte semelhante ao comunicador iterativo apresentado por [LI](#); [JAGADISH](#). Contudo, ele também serve como guia para a elaboração da consulta do usuário, que apesar de estar limitada a consultas mais simples, deve representar com precisão o desejo do usuário. É importante destacar que o processamento textual difere, por fazer uso do *SP* com foco em obter os *tokens* POS ao invés de ter como objetivo uma consulta SQL, afinal o objetivo é ter uma expressão relacional.

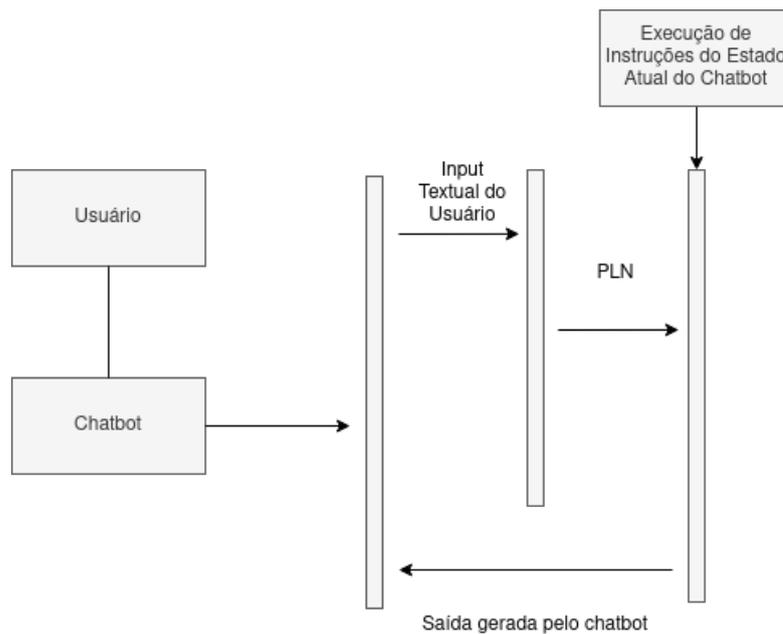
# 3

## Projeto

A proposta deste trabalho é mostrar como realizar a construção de uma NLIDB que funciona por meio de um *chatbot* com comportamento inquisitivo. Toda a comunicação é textual em inglês. A interface irá funcionar via web utilizando os métodos de requisição do HTTP. Para que a solução seja factível, seguiremos os diagramas projetados apresentados neste capítulo.

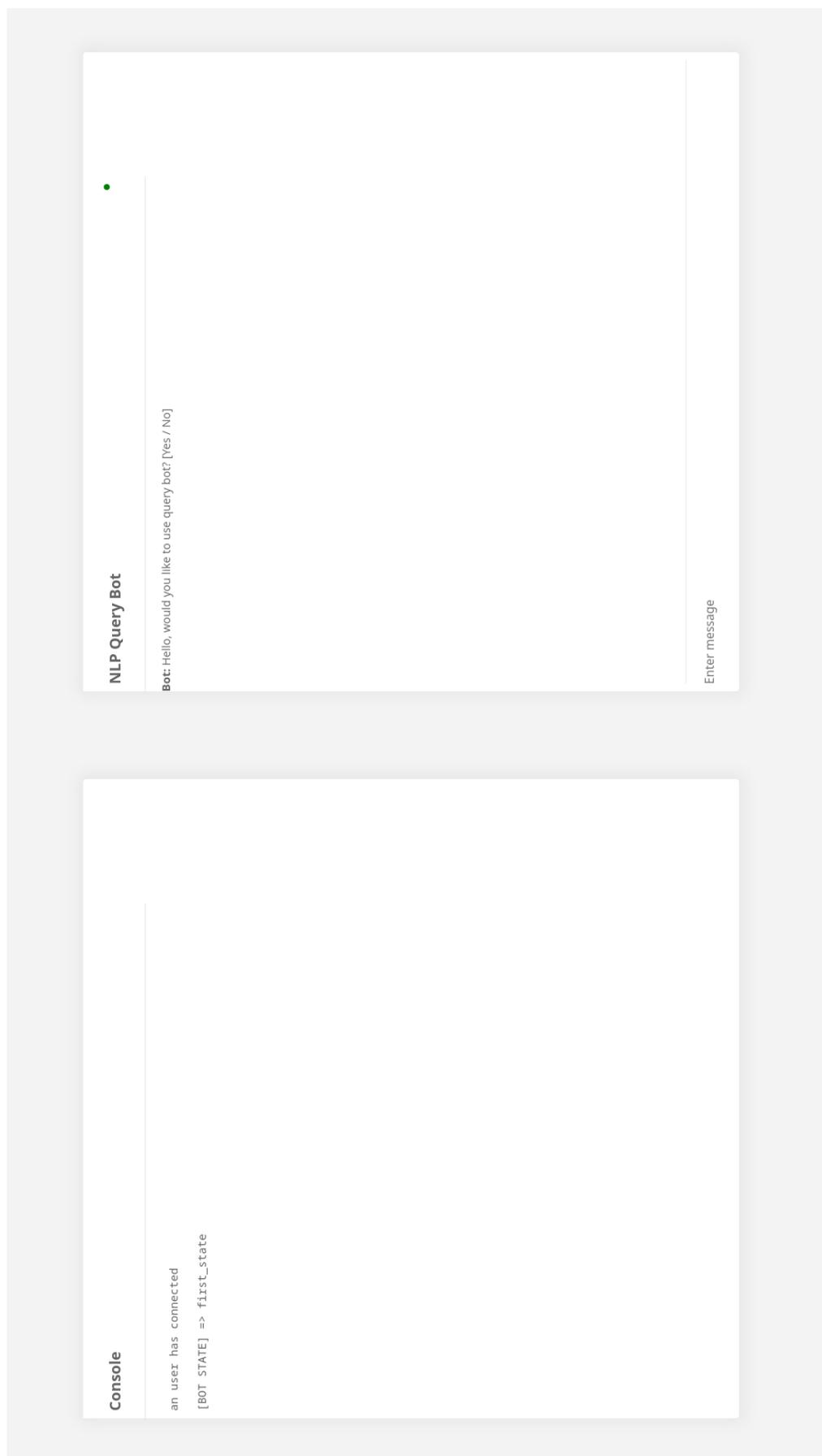
A Figura 7 demonstra como o usuário irá comunicar-se com o *chatbot*. A Figura 8 ilustra como ficou o protótipo da interface finalizada, onde do lado esquerdo mostramos um *log* com finalidade de estudo para saber o que ocorre internamente na aplicação, e do lado direito a NLIDB textual com a qual o usuário interage para realizar as consultas.

Figura 7 – Diagrama de Sequência do Funcionamento da Interface



Fonte: Autores (2022).

Figura 8 – Demonstração da Interface Finalizada

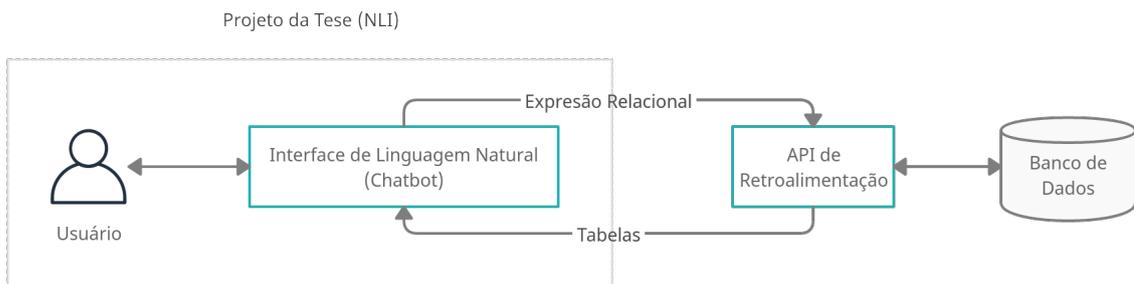


Fonte: Autores (2022).

### 3.1 Visão Geral

Na visão geral do projeto, é destacado que a interface é apenas uma de duas partes, e esclarecido qual o problema que ela se propõe a solucionar. Ou seja, o projeto tem a finalidade de construir a NLIDB responsável pelo preparo da expressão relacional e a mesma é utilizada unicamente pelo usuário comum.

Figura 9 – Visão Geral do Projeto



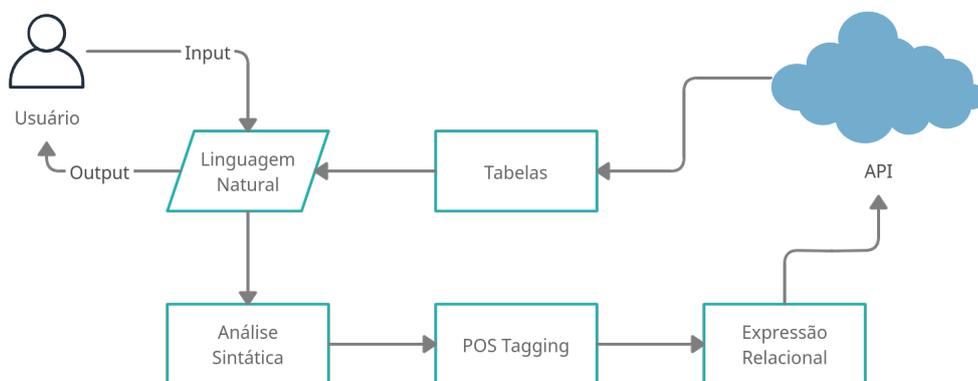
Fonte: Autores (2022).

#### 3.1.1 Arquitetura da NLI

Na arquitetura é apresentada uma abstração de como ocorre o processo de tradução da linguagem natural para a expressão relacional, e de como tratamos as respostas recebidas da API. Todo input feito pelo usuário passa pelo *semantic parsing* o processo onde extraímos os tokens POS que são utilizados na elaboração da expressão relacional.

A expressão relacional é então enviada para uma API que a executa e retorna como respostas tabelas. As tabelas indicam possíveis novas consultas ou opções de filtro que são oferecidas ao usuário pelo *chatbot*, limitando suas escolhas para o domínio das soluções possíveis.

Figura 10 – Interface de Linguagem Natural

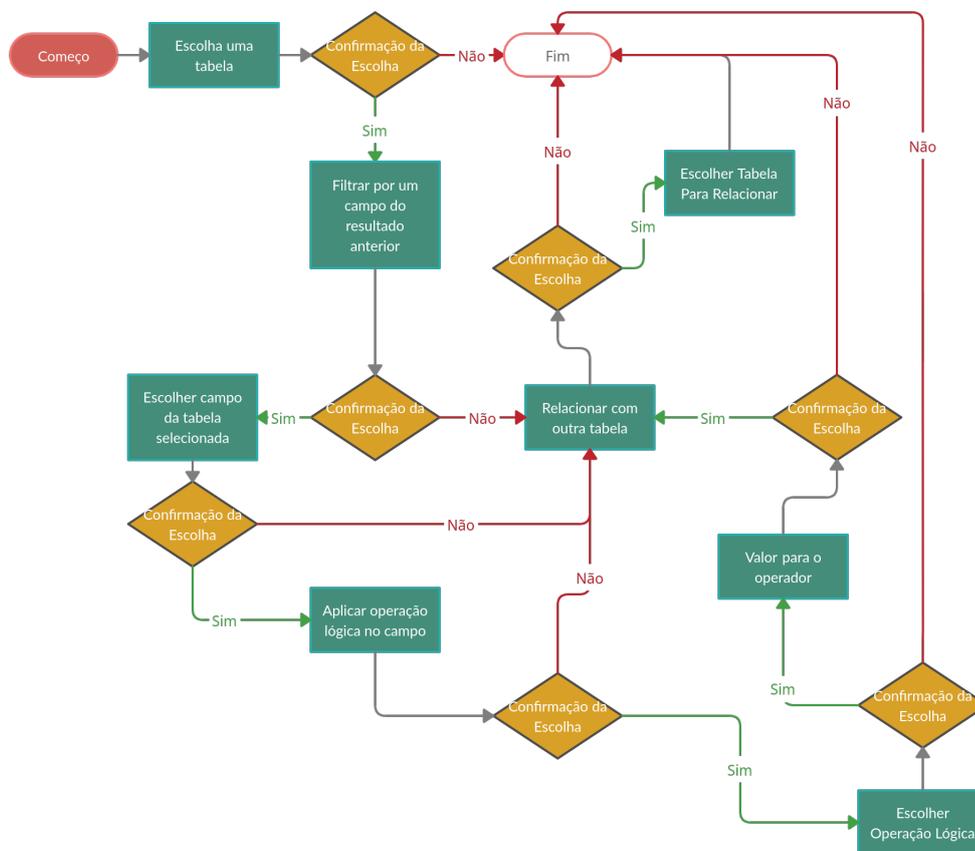


Fonte: Autores (2022).

### 3.1.2 Fluxo de Estados do Chatbot

A Figura 11 que demonstra o fluxo dos estados do *chatbot*, esquematiza como o *chatbot* deve comportar-se para manter o seu padrão inquisitivo e para que consiga chegar ao objetivo da interface, que é construir uma expressão relacional. Qualquer input de linguagem natural que não consiga ser processado é considerado um erro na elaboração da expressão relacional e encerra a aplicação de forma imediata. Nas seções a seguir onde explicamos como é realizada a construção do protótipo, também é descrito em detalhes o que acontece em cada um dos estados e como foi feita sua implementação

Figura 11 – Fluxo de Estados do Chatbot



Fonte: Autores (2022).

## 3.2 Construção do Protótipo

Uma vez que os desenhos do comportamento de tradução da linguagem natural e o fluxograma de estados do *chatbot* foram elaborados pode se dar início à implementação do protótipo. Essa ferramenta de processamento de linguagem natural irá se comunicar com uma API via web, então escolhemos construí-la utilizando a framework *Express* disponível por meio do runtime *NodeJS*.

Por ter a forma de um *chatbot* foi preciso oferecer ao usuário uma forma de enviar e receber mensagens. Uma maneira de oferecer um chat com troca de mensagens em tempo real é através de websockets (MELNIKOV; FETTE, 2011). Para que os websockets funcionem no protótipo, foi utilizada uma biblioteca chamada SocketIO.

As mensagens que são trocadas com o *chatbot* precisam ser processadas pelo *SP*. Para realizar a análise semântica nas mensagens enviadas pelo usuário, foi utilizado o pacote *wink-pos-tagger* responsável pelo processamento de linguagem natural da aplicação. Como o usuário precisa de um feedback visual e enviar suas mensagens de alguma maneira, fora criada uma página HTML com CSS utilizando apenas de um form que ao ser submetido envia a mensagem do usuário pelo websocket.

Para que a interface consiga comunicar-se com a API é preciso que ela realize requisições *HTTP*, a maneira como a aplicação pode realizar estes pedidos web através do navegador é utilizando o *Axios*, um cliente *HTTP*. Com finalidade de oferecer um protótipo funcional, a *NLIDB* precisou ser hospedada em alguma plataforma na nuvem. A plataforma escolhida foi Heroku.

O desenvolvimento do protótipo foi feito com controle de versionamento pelo *Bitbucket*, onde foram separadas as funcionalidades em branches e ordem de prioridade. A aplicação é escrita em Javascript fazendo uso do Typescript. O JavaScript facilita também a manipulação do *Javascript Object Notation (JSON...)* (2017) que é o meio como é representada nossa expressão relacional para a API externa.

Por tratar-se de um protótipo, a interface não foi preparada para fazer requisições na API. Ainda que tenha sido destacado o uso do *Axios*. Todas as respostas de tabelas e campos são falsas, mas existem para mostrar o aspecto essencial da interface que é a construção da expressão relacional. Portanto, uma vez que exista uma API poderemos fazer a comunicação apenas adicionando as requisições na interface.

O protótipo está disponível para acesso em: <https://nlp-query-bot.herokuapp.com/> e foi desenvolvido seguindo essas etapas:

- **Inicialização** - Onde é inicializado o projeto em NodeJS e o controle de versionamento.
- **API Funcional** - Adição do Express a aplicação, para realizar requisições web e obter respostas.
- **Chat** - Construção visual do chat como uma página HTML e análise semântica das mensagens do usuário.
- **Construção da expressão relacional** - onde foi adicionado o comportamento inquisito como estados no *chatbot*.

Nas subseções seguinte é descrito em detalhes o runtime, as bibliotecas, pacotes e plataforma que tornam o protótipo funcional, oferecendo exemplos de uso das ferramentas.

### 3.2.1 Testes, Cobertura e Qualidade de Código

Não foi utilizada nenhuma metodologia de desenvolvimento baseada em testes, durante a construção do protótipo. Também não adotamos nenhuma ferramenta de testagem, cobertura ou supervisão de qualidade de código durante a execução do trabalho.

### 3.2.2 NodeJS

O NodeJS é uma biblioteca em tempo de execução assíncrona projetada para a construção de aplicações web escaláveis (NODEJS..., 2022) e pode ser instalado conforme as instruções no site ou pelo seu gerenciador de pacotes do linux. O NodeJS vem acompanhado do *NPM* que significa gerenciador de pacotes node. O *NPM* é a ferramenta que será utilizada para adicionar e gerenciar as bibliotecas e pacotes da aplicação.

Figura 12 – Classe Principal da Aplicação

```
1 import express from 'express';
2 import { createServer, Server } from 'http';
3 import { Server as ioServer, Socket } from 'socket.io';
4 import router from './routes/router';
5 import navigateThroughBotStates from './functions/navigateThroughBotStates';
6 import Message from './types/Message';
7 import BotData from './types/BotData';
8 import selectBotStateMessageAnalysis from './functions/messageAnalysis';
9
10 class App {
11   public app: express.Application;
12   public server: Server;
13   private io: ioServer;
14   public state: any;
15   public last_reiceved_data: any;
16   public table: any;
17   public field_to_filter: any;
18   public logic_operator: any;
19   public logic_operation_value: any;
20   public table_to_relate: any;
21   public botData: BotData;
22
23   constructor() {
24     this.app = express();
25     this.server = createServer(this.app);
26     this.io = new ioServer(this.server);
27     this.routes();
28     this.listen();
29     this.state;
30     this.last_reiceved_data;
31     this.botData = {
32       state: this.state,
33       last_reiceved_data: this.last_reiceved_data,
34       table: this.table,
35       field_to_filter: this.field_to_filter,
36       logic_operator: this.logic_operator,
37       logic_operation_value: this.logic_operation_value,
38       table_to_relate: this.table_to_relate,
39     }
40   }
41
42   routes() {
43     this.app.use(router);
44   }
45
46   private listen(): void {
47     this.io.on('connection', (socket: Socket) => {
48       this.botData = {
49         state: 'first_state',
50         last_reiceved_data: [],
51         table: '',
52         field_to_filter: '',
53         logic_operator: '',
54         logic_operation_value: '',
55         table_to_relate: '',
56       }
57
58       navigateThroughBotStates(this.botData, socket);
59
60       socket.on('chat message', async (message: Message) => {
61         const messageJSON = JSON.stringify(message);
62
63         socket.emit('console message', `received message => ${messageJSON}`);
64
65         socket.emit('chat message', message);
66
67         this.botData = selectBotStateMessageAnalysis(this.botData, socket, message.text);
68         this.botData = navigateThroughBotStates(this.botData, socket);
69       });
70     });
71   }
72 }
73
74 export default new App();
75
76
```

Fonte: Autores (2022).

### 3.2.3 Express

Para utilizar o *Express* é preciso instalar a framework por meio do *NPM*. Instruções para a instalação podem ser encontradas no site da framework ([EXPRESS...](#), 2022). No guia *getting started*. A diferença entre o *Express* e o *NodeJS* é que por se tratar de uma framework o *Express* irá prover um conjunto de funcionalidades para o *NodeJS* que tem como objetivo facilitar a construção da aplicação web.

Figura 13 – Código do Express inicializando a aplicação

```
1 import app from './app';
2
3 const PORT: string | number = process.env.PORT || '8000';
4
5 app.server.listen(PORT, () => {
6   console.log(`>[server]: hosting @${PORT}`);
7 });
```

Fonte: Autores (2022).

Figura 14 – Router que identifica a resposta do index.html na rota '/'

```
1 import express from 'express';
2 import path from 'path';
3
4 const router = express.Router();
5
6 router.use(express.static(path.join(__dirname, '../', 'views')));
7
8 router.route('/').get((_req, res, _next) => {
9   res.render('index.html');
10 });
11
12 export default router;
13
```

Fonte: Autores (2022).

Observe na Figura 13, como é simples inicializar uma aplicação web pelo *Express*. Tudo que precisamos fazer é importar o *app* e pedir para que escute em uma porta definida na variável de ambiente do projeto. *App* é a classe principal do nosso código e é nela onde iremos inicializar as bibliotecas e pacotes da aplicação. Observe a imagem 12, que mostra o código principal da aplicação. Nele definimos os atributos e realizamos as chamadas necessárias para que a aplicação funcione.

Na linha 24, da Figura 12, o `express` é instanciado e é ele que é chamado na linha 5 do código de inicialização da aplicação. Quando chamamos a aplicação no navegador a linha 43 é onde definimos as rotas, e a resposta que a aplicação deve devolver é o arquivo `index.html` na rota `'/'` que é a nossa view que atua como NLIDB, ilustrado na imagem 14.

### 3.2.4 SocketIO

O *SocketIO* é uma biblioteca que permite comunicação entre cliente servidor, construída em cima do protocolo de websockets (SOCKETIO..., 2022), é por meio dela que realizaremos os envios de mensagem que o usuário submete na página html e também por onde irá receber as mensagens do *chatbot* nessa mesma página. Realizamos a emissão da mensagem inicial pelo primeiro estado que será explicado mais adiante. Para emitir mensagens utilizamos `socket.emit` como na linha 65 da da Figura 12, e para receber mensagens utilizamos `socket.on`.

#### 3.2.4.1 View

Página HTML que é retornada na rota `'/'`. O Arquivo é um formulário simples, mas para que funcione como uma interface de conversa em chat com o *chatbot* precisamos utilizar alguns scripts. O script tanto verifica a conexão do websocket, a bolinha verde indica que há uma conexão ativa enquanto que a bolinha vermelha indica que a conexão foi encerrada. Também é responsável por emitir as mensagens recebidas do bot e de capturar as mensagens do usuário depois que o mesmo digitar algum texto e apertar enter como pode ser visto nas linhas 24-32 da Figura 15.

Figura 15 – Script to index.html da rota '/'

```
1 import $ from 'jquery';
2 import io from 'socket.io-client';
3 import Message from '../types/Message';
4
5 $(() => {
6   const socket = io();
7
8   const status = $('.connection-status');
9
10  status.addClass('connected');
11
12  socket.on('open', () => {
13    status.addClass('connected');
14  });
15
16  socket.on('error', () => {
17    status.addClass('connected');
18  });
19
20  socket.on('disconnect', () => {
21    status.removeClass('connected');
22  });
23
24  $('#message').on('keypress', e => {
25    if (e.code === 'Enter') {
26      socket.emit('chat message', {
27        name: 'User: ',
28        text: $('#message').val(),
29      });
30      $('#message').val('');
31    }
32  });
33
34  socket.on('chat message', (message: Message) => {
35    const data = `

<b> ${message.name} </b> ${message.text}</p>`;
36
37    $('#messages').append(data);
38
39    $('#messages').scrollTop($('#messages')[0].scrollHeight);
40  });
41
42  socket.on('console message', (message: JSON) => {
43    const data = `

<code> ${message}</code></p>`;
44
45    $('#console').append(data);
46
47    $('#console').scrollTop($('#console')[0].scrollHeight);
48  });
49 });
50


```

Fonte: Autores (2022).

### 3.2.5 wink-post-tagger

O `wink-post-tagger` é a biblioteca responsável pelo processamento da linguagem natural textual na aplicação. A ferramenta faz parte de uma família de ferramentas que tem como foco análise estatística, processamento de linguagem natural e aprendizado de máquina (WINK..., 2022). É do nosso interesse apenas a capacidade do `WPT` de aplicar análise semântica e extrair a classificação POS TAGGER do texto em inglês (Figura 16).

Depois de obter o array de tokens classificados da mensagem, a NLIDB irá agir conforme o estado inquisitivo atual do bot, que determina como os tokens serão utilizados. Esse processo é explicado na sessão de estados

Figura 16 – Código que transforma a linguagem natural em tokens POS

```
1 import PosTagger, { PosTaggedToken } from 'wink-pos-tagger';
2
3 function naturalToTokens(message: string): PosTaggedToken[] {
4   const tagger = new PosTagger();
5
6   return tagger.tagSentence(message);
7 }
8
9 export default naturalToTokens;
10
```

Fonte: Autores (2022).

### 3.2.6 Axios

Apesar de não estar implementado, o `Axios` precisa ser mencionado pois é a maneira como a NLIDB irá comunicar-se com a API que serve de acesso para a base de dados. O `Axios` é um Cliente HTTP que pode ser chamado do navegador (AXIOS..., 2022).

### 3.2.7 Heroku

O `Heroku` é plataforma que hospedada a aplicação na nuvem. Oferece serviços gratuitos que são simples de utilizar em razão do seu CLI, que permite um *deploy* simplificado da aplicação sem que o desenvolvedor se preocupe imediatamente com questões de infraestrutura (HEROKU..., 2022). Instruções para utilização do `Heroku` estão disponíveis no próprio site da plataforma.

### 3.2.8 Objeto Expressão Relacional

Observe que na imagem 12 da classe principal do projeto temos atributo que representa um objeto chamado de `botData`. Esse objeto é o nosso objeto que representa a expressão relacional e é passado de estado em estado no *chatbot* inquisitivo. Esse objeto também é quem define qual o próximo estado será escolhido pelo *chatbot*. Seus atributos são:

- **state** - atributo para uso interno do *chatbot*, utilizado para saber qual o estado atual.
- **last\_received\_data** - memória temporária que armazena os últimos dados recebidos como resposta da API
- **table** - tabela escolhida pelo usuário para consulta.
- **field\_to\_filter** - campo da tabela para filtrar a consulta
- **logic\_operator** - operador lógico para filtrar a consulta
- **logic\_operation\_value** - valor para ser utilizado como comparação com o operador lógico
- **table\_to\_relate** - tabela para relacionar com a consulta.

### 3.2.9 Estados

A seguir é analisado e explicado os estados do *chatbot* inquisitivo. O método *navigateThroughBotStates* chamado na linha 58(12) é o responsável por navegar entre os estados do *chatbot*.

Observe que sempre que o usuário digita uma mensagem precisamos passar a mensagem por um dos estados de análise pelo método *selectBotStateMessageAnalysis* inicialmente chamado na linha 67 da classe principal(12). Para analisar a confirmação do estado começo conforme mostrado pela mensagem do *chatbot* na próxima sub subseção.

Figura 17 – Código do Estado Inicial

```
1 import { Socket } from 'socket.io';
2 import BotData from '../types/BotData';
3
4 function executeFirstState(botData: BotData, socket: Socket): void {
5   socket.emit('console message', 'an user has connected');
6
7   socket.emit('console message', `[BOT STATE] => ${botData.state}`);
8
9   socket.emit('chat message', {
10     name: 'Bot: ',
11     text: 'Hello, would you like to use query bot? [Yes / No]',
12   });
13 }
14
15 export default executeFirstState;
16
```

Fonte: Autores (2022).

### 3.2.9.1 Estado Começo

Estado inicial do *chatbot*, automaticamente inicializado sempre que a aplicação é aberta no navegador. O *chatbot* inicializa a conversa na interface com o usuário, perguntando se o usuário deseja utilizá-la e entra em um estado de confirmação onde só aceita sim(*yes*) ou não(*no*) como resposta (imagem 17).

- **Caso Afirmativo** - Segue para o estado da escolha de tabelas.
- **Caso Negativo** - Encerra a aplicação e exibe a mensagem para reiniciar a aplicação caso deseje utilizar novamente.

Figura 18 – Código do Estado Inicial de Análise da Mensagem

```
1 import { Socket } from 'socket.io';
2 import BotData from '../types/BotData';
3 import naturalToTokens from '../functions/naturalToTokens';
4
5 function firstUserConfirmation(socket: Socket, message: string): BotData {
6   let nextState: string;
7
8   const tokens = naturalToTokens(message);
9
10  const tokensJSON = JSON.stringify(tokens);
11
12  socket.emit('console message', `message analysis tokens => ${tokensJSON}`);
13
14  const tags = ['UH', 'DT'];
15
16  socket.emit('console message', `message analysis allowed POS Tags => ${tags}`);
17
18  const filteredTokens = tokens.map(token => {
19    if (tags.indexOf(token.pos) !== -1) {
20      return token.value;
21    }
22  });
23
24  const result = filteredTokens.filter(token => token !== null);
25
26  socket.emit('console message', `message analysis result => ${result}`);
27
28  if (result[0] === 'Yes' || result[0] === 'yes') {
29    nextState = 'tables';
30  } else {
31    nextState = 'end';
32  }
33
34  return {
35    state: nextState,
36    last_reiceved_data: [],
37    table: '',
38    field_to_filter: '',
39    logic_operator: '',
40    logic_operation_value: '',
41    table_to_relate: '',
42  }
43 }
44
45 export default firstUserConfirmation;
46
```

Fonte: Autores (2022).

Observe como a abstração de um estado do *chatbot* inquisitivo é dividida em duas etapas. A primeira que atua como emissora da mensagem, e a segunda que analisa a mensagem em linguagem natural fornecida pelo usuário. Essa mensagem é classificada utilizando o *WPT* como mencionado anteriormente.

Uma vez obtido o array de tokens que é o resultado da função *naturalToTokens*, separamos as palavras pela sua classificação, como procuramos por determinantes ou interjeições já que queremos confirmações definimos quais tags pos é desejada no estado na linha 14(Figura 18), como mostrada pela imagem que mostra o código do estado inicial da análise da imagem e em seguida filtramos os tokens como mostrado da linha 18-22(Figura 18). Caso algum dos tokens seja igual a uma das palavras desejadas o *chatbot* segue seu fluxo alterando a propriedade que indica qual o próximo estado que deve ser visitado. Caso o *chatbot* não reconheça os tokens

ele simplesmente encerra a consulta e pede para que o usuário reinicie a interface e comece novamente.

Esse é um comportamento análogo em todos os estados, o que os diferencia são os tokens que são verificados e irão variar de estado para estado.

### 3.2.9.2 Estado Escolha uma tabela

Estado onde o *chatbot* apresenta ao usuário todas as tabelas disponíveis para consulta no banco de dados, e permite apenas a escolha de uma tabela. O usuário oferece como entrada seu input textual em linguagem natural, o qual passa pelo *SP* que transcreve a entrada em tokens com classificação POS, desses token são separados os substantivos, e somente o primeiro substantivo que for igual ao nome de uma das tabelas oferecidas é escolhido para consulta. Em seguida o usuário é levado ao estado de confirmação da escolha de uma tabela.

- **Caso um dos tokens tenha o nome de uma tabela** - Seleciona a tabela na expressão relacional e segue para o próximo estado.

Figura 19 – Código do Estado de Escolha de uma Tabela

```
1 import { Socket } from 'socket.io';
2 import BotData from '../types/BotData';
3 import getMockedResponse from '../functions/getMockedResponse';
4
5 function executeTablesState(botData: BotData, socket: Socket): BotData {
6   const response = getMockedResponse();
7
8   socket.emit('console message', `[BOT STATE] => ${botData.state}`);
9
10  const data: Array<string> = response[botData.state];
11
12  const dataJSON = JSON.stringify(data);
13
14  socket.emit('chat message', {
15    name: 'Bot: ',
16    text: `Please, select one of the [${botData.state}] => ${dataJSON}`,
17  });
18
19  botData.last_reiceved_data = data;
20
21  return botData;
22 }
23
24 export default executeTablesState;
25
```

Fonte: Autores (2022).

Figura 20 – Código do estado Análise da Mensagem da Escolha de uma Tabela

```
1 import { Socket } from 'socket.io';
2 import BotData from '../types/BotData';
3 import naturalToTokens from '../functions/naturalToTokens';
4 import compareWithLastReicevedData from '../functions/compareWithLastStateResponse';
5
6 function tables(socket: Socket, message: string, botData: BotData): BotData {
7   let nextState: string;
8
9   const tokens = naturalToTokens(message);
10
11   const tokensJSON = JSON.stringify(tokens);
12
13   socket.emit('console message', `message analysis tokens => ${tokensJSON}`);
14
15   const nounTags = ['NN', 'NNS', 'NP', 'NNP'];
16
17   socket.emit('console message', `message analysis allowed POS Tags => ${nounTags}`);
18
19   const nounTokens = tokens.map(token => {
20     if (nounTags.indexOf(token.pos) !== -1) {
21       return token.value;
22     }
23   });
24
25   const result = nounTokens.filter(token => token !== null);
26
27   const intersection = compareWithLastReicevedData(botData.last_reiceved_data, result);
28
29   if (!intersection[0]) {
30     botData.state = 'end'
31
32     return botData
33   }
34
35   socket.emit('console message', `message analysis result => ${intersection[0]}`);
36
37   nextState = 'tables_user_confirmation'
38
39   botData.state = nextState;
40
41   botData.last_reiceved_data = intersection[0];
42
43   return botData;
44 }
45
46 export default tables;
47
```

Fonte: Autores (2022).

Observe no código da linha 15-22(Figura 20), como agora separamos somente os substantivos e comparamos os tokens com os valores fornecidos como resultado da tabela. Os outros estados de seleção de campos, filtros e junção funcionam de forma análoga a esse.

### 3.2.10 Estados de Confirmação

Esses estados atuam como ponte entre os inputs do usuário e outputs da interface, e servem somente para garantir que as escolhas feitas até o momento por meio do diálogo estejam corretas com o que o usuário escreveu e deseja.

Funcionam de forma semelhante ao estado inicial, com a diferença do próximo estado para onde o *chatbot* é enviado, pois varia conforme o estado de confirmação, a imagem com o

fluxograma de todos os estados ilustra o estado de destino após os estados de confirmação..

### 3.2.11 Estados de Escolha de Filtros

Referentes aos estados: de escolha de Campo, operador lógico, escolha do operador Lógico, escolha de tabela para relacionamento. Esses estados oferecem ao usuário a opção de elaborar consultas um pouco mais complexas para obter dados mais específicos. Apesar de diferentes, esses estados funcionam de forma análoga ao estado de confirmação de tabela, o que os diferencia são os valores que são comparados com os tokens: campos de uma tabela para o estado de campo, opções de filtro para um estado de filtro, ou opções de relacionamento para um estado de junção.

Qualquer ação tomada em um desses estados leva para o próximo estado de confirmação, que por consequência guia o usuário para o próximo estado de escolha no fluxograma.

### 3.2.12 Consultar uma Tabela do Banco de Dados

Dado um banco de dados relacional qualquer, a interface deve ser capaz de selecionar uma das tabelas listadas desse banco e ter como resposta todos os dados daquela tabela.

Para que isso aconteça é necessário extrair do input textual do usuário em linguagem natural o que esperamos da resposta no estado de escolher uma tabela do *chatbot*. Uma vez que os tokens são extraídos do input do usuário é realizada uma tradução para expressão relacional. Essa tradução ocorre se existir alguma interseção entre as tabelas oferecidas como escolha e o token de substantivo presente no input do usuário. A interface sempre considera somente o primeiro substantivo que pertence a interseção nesta etapa.

### 3.2.13 Filtrar Resultados de uma Tabela do Banco de Dados

Uma vez obtido o resultado da consulta de uma tabela o usuário deve poder filtrar esse resultado com base operações de comparação.

Nessa etapa o *chatbot* inicialmente opera semelhante a consulta de uma tabela como descrito na sub sessão anterior, porém com a distinção de escolher um dos campos da tabela escolhida no estado anterior. A seguir o output oferece a opção de utilizarmos operações lógicas no campo escolhido, o processo de tradução é semelhante, mas olhando para os advérbios na linguagem natural e assim adicionando a operação escolhida na expressão. Caso o usuário confirme a escolha oferecemos a ele um campo aberto para o valor da operação permitindo qualquer string a ser adicionada na expressão relacional.

### **3.2.14 Relacionar os Dados da Consulta com Outra Tabela do Banco de Dados**

Obtendo o resultado da consulta atual de uma tabela. O usuário poderá escolher relacionar os dados da tabela atual escolhida com os dados de outras tabelas. A primeira tabela escolhida poderá se relacionar somente com as tabelas que a resposta da API determinar. Nesse estado não foi adicionado nenhum estado de confirmação seguinte, pois ambas as escolhas levam para o estado final da interface, diferindo somente na resposta obtida.

### **3.2.15 Estado de Fim**

Estado que encerra o fluxo do *chatbot* inquisitivo. Qualquer ação que não possa ser compreendida, ou qualquer escolha que leve o bot a esse estado, encerra o *chatbot*. Para que a interface seja utilizada novamente a mesma precisa ser reinicializada.

# 4

## Validação

Neste capítulo são abordados os estudos de caso efetuados com o protótipo do *chatbot* para realização de consultas em bancos de dados relacionais. O estudo foi feito olhando para os três tipos de consulta permitidos pela interface: consultar uma tabela, filtrar resultados de uma tabela e relacionar uma tabela com outra tabela.

Para validar a precisão das consultas em expressão relacional, foram escolhidas cinco consultas feitas em SQL que serão comparadas com sua forma equivalente em linguagem natural e seu resultado obtido pela NLIDB como expressão relacional nas próximas seções deste capítulo. As consultas englobam todas as funcionalidades da NLIDB ilustrando o seu funcionamento e suas limitações.

É importante lembrar que o protótipo não possui uma comunicação real com a API, pois a API não foi disponibilizada, todos os dados parciais mostrados nas ilustrações são *mocks* de valores em um objeto, que tem a finalidade de ilustrar a formação da expressão relacional.

Figura 21 – Etapas da validação



Fonte: Autores (2022).

## 4.1 Selecionar uma Tabela

Espera-se que seja possível criar a expressão relacional da tabela endereço selecionada pela interface. É o equivalente da consulta *SELECT \* FROM endereço* em SQL. Em seguida temos imagens comparando o resultado esperado da expressão relacional e ilustrando o comportamento dela no protótipo da NLIDB, onde destacamos em verde a entrada do usuário utilizando linguagem natural para selecionar a tabela endereço, e como este input é classificado e organizado.

Figura 22 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada

Linguagem	Pergunta
Linguagem Natural	show me table endereço
SQL	SELECT * FROM endereço
Expressão Relacional	[TABLE] => endereço [FIELD] => [OPERATOR] => [OPERATOR VALUE] => [TABLE TO RELATE] =>

Fonte: Autores (2022).

Figura 23 – NLIDB sendo utilizada para obter os dados da tabela endereço *SELECT \* FROM endereço*

```

Bot: Hello, would you like to use query bot? [Yes / No]
User: yes
Bot: Please, select one of the [tables] => ["endereço","estudante"]
User: show me table endereço
Bot: Did you select endereço ? [Yes / No]
User: yes
Bot: Selected [endereço] => [{"idEndereco":1,"cep":"49100000","rua":"Rua A","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"},{"idEndereco":2,"cep":"49100000","rua":"Rua B","bairro":"Centro","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":3,"cep":"49100000","rua":"Rua C","bairro":"Centro","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":4,"cep":"49100000","rua":"Rua A","bairro":"Salgado Filho","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":5,"cep":"49100000","rua":"Rua A","bairro":"Atalaia","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":6,"cep":"49100000","rua":"Rua ABC","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":7,"cep":"49100000","rua":"Rua C","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":8,"cep":"49100000","rua":"Rua A","bairro":"Centro","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":9,"cep":"49100000","rua":"Rua A","bairro":"Cirurgia","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":10,"cep":"49100000","rua":"Rua A1","bairro":"Bairro América","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":11,"cep":"49100000","rua":"Rua A","bairro":"Siqueira Campos","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":12,"cep":"49100000","rua":"Rua A","bairro":"Getúlio Vargas","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}]
    
```

Fonte: Autores (2022).

Figura 24 – log da NLIDB sendo utilizada para obter os dados da tabela endereco  
SELECT \* FROM endereco

```

received message => {"name":"User: ","text":"show me table endereco"}

message analysis tokens =>
[{"value":"show","tag":"word","normal":"show","pos":"VB","lemma":"show"},
{"value":"me","tag":"word","normal":"me","pos":"PRP"},
{"value":"table","tag":"word","normal":"table","pos":"NN","lemma":"table"},
{"value":"endereco","tag":"word","normal":"endereco","pos":"NNP","lemma":"endereco"}]

message analysis allowed POS Tags => NN,NNS,NP,NNP

message analysis result => endereco

[BOT STATE] => tables_user_confirmation

received message => {"name":"User: ","text":"yes"}

message analysis tokens => [{"value":"yes","tag":"word","normal":"yes","pos":"UH"}]

message analysis allowed POS Tags => UH,DT

message analysis result => yes

[BOT STATE] => table

received message => {"name":"User: ","text":"no"}

message analysis tokens => [{"value":"no","tag":"word","normal":"no","pos":"DT"}]

message analysis allowed POS Tags => UH,DT

message analysis result => no

[BOT STATE] => relate_table_state

received message => {"name":"User: ","text":"no"}
-----

```

Fonte: Autores (2022).

Observe destacado em verde na duas figuras anteriores como é feita a classificação POS utilizada para extrair a entrada em linguagem natural do input oferecido pelo usuário.

Figura 25 – Resultado da Expressão Relacional

```

[BOT STATE] => end
[TABLE] => endereco
[FIELD] =>
[OPERATOR] =>
[OPERATOR VALUE] =>
[TABLE TO RELATE] =>

```

Fonte: Autores (2022).

Comparando o resultado esperado na tabela da imagem 22 com o obtido da interface na imagem 25, observamos que os resultados foram dentro do esperado, o bot state é apenas um atributo interno de controle do *chatbot* e não tem implicação no resultado.

De forma análoga deve ser possível criar a expressão relacional da tabela estudante selecionada pela interface. É o equivalente da consulta *SELECT \* FROM estudante* em SQL. O que de fato acontece se olharmos as figuras 26 e compararmos o resultado do protótipo em 28.

Figura 26 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada

Linguagem	Pergunta
Linguagem Natural	show me estudante
SQL	SELECT * FROM estudante
Expressão Relacional	[TABLE] => estudante [FIELD] => [OPERATOR] => [OPERATOR VALUE] => [TABLE TO RELATE] =>

Fonte: Autores (2022).

Figura 27 – NLIDB sendo utilizada para obter os dados da tabela estudante *SELECT \* FROM estudante*

```
User: yes
Bot: Please, select one of the [tables] => ["endereco","estudante"]
User: show estudante
Bot: Did you select estudante ? [Yes / No]
User: yes
Bot: Selected [estudante] => [{"matriculaEstudante":"E101","cpf":"2222222201","mc":7},
{"matriculaEstudante":"E102","cpf":"2222222202","mc":8.3},
{"matriculaEstudante":"E103","cpf":"2222222203","mc":6.7},{matriculaEstudante":"E104","cpf":"2222222204","mc":0},
{"matriculaEstudante":"E105","cpf":"2222222205","mc":9},{matriculaEstudante":"E106","cpf":"2222222206","mc":7.7},
{"matriculaEstudante":"E107","cpf":"2222222207","mc":5.5},
{"matriculaEstudante":"E108","cpf":"2222222208","mc":6.5},{matriculaEstudante":"E109","cpf":"2222222209","mc":6},
{"matriculaEstudante":"E110","cpf":"2222222210","mc":2.1},
{"matriculaEstudante":"E111","cpf":"2222222211","mc":3.3},
{"matriculaEstudante":"E112","cpf":"2222222212","mc":4.5},
{"matriculaEstudante":"E113","cpf":"2222222213","mc":8.1}]
Bot: Would you like to select a field ? [Yes / No]
User: no
Bot: Would you like to relate the result with another table ? [Yes / No]
User: no
Bot: Please, refresh to use again.
```

Fonte: Autores (2022).

Observe como a imagem 27 tem semelhança com a Figura 23, a diferença esta nas tabelas selecionadas, e nos dados obtidos como resposta. Mas o comportamento inquisitivo do *chatbot* permanece constante.

Figura 28 – Resultado da Expressão Relacional

```
[BOT STATE] => end
[TABLE] => estudante
[FIELD] =>
[OPERATOR] =>
[OPERATOR VALUE] =>
[TABLE TO RELATE] =>
```

Fonte: Autores (2022).

Ao manter a estrutura e mudar somente as tabelas, observamos como é redundante repetir uma consulta apenas alterando seus campos. Esse exemplo foi utilizado com finalidade de esclarecer o porquê do estudo de caso evitar repetições e priorizar a demonstração das funcionalidades da NLIDB.

Comparando o resultado esperado na tabela da imagem 26 com o obtido da interface na imagem 28, observamos que os resultados foram dentro do esperado, o bot state é apenas um atributo interno de controle do *chatbot* e não tem implicação no resultado.

## 4.2 Selecionar um campo de uma tabela

Nesta seção criamos uma expressão relacional que seleciona os dados de uma tabela, o próximo passo é conferir se é possível selecionar um campo específico dessa tabela com base na consultas *SELECT cep FROM endereco*.

Figura 29 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada

Linguagem	Pergunta
Linguagem Natural	show me cep of endereco
SQL	SELECT cep FROM endereco
Expressão Relacional	[TABLE] => endereco [FIELD] => cep [OPERATOR] => [OPERATOR VALUE] => [TABLE TO RELATE] =>

Fonte: Autores (2022).

Figura 30 – NLIDB sendo utilizada para expressão equivalente a SELECT cep FROM endereco

**Bot:** Hello, would you like to use query bot? [Yes / No]

**User:** yes

**Bot:** Please, select one of the [tables] => ["endereco","estudante"]

**User:** endereco

**Bot:** Did you select endereco ? [Yes / No]

**User:** yes

**Bot:** Selected [endereco] => [{"idEndereco":1,"cep":"49100000","rua":"Rua A","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":2,"cep":"49100000","rua":"Rua B","bairro":"Centro","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":3,"cep":"49100000","rua":"Rua C","bairro":"Centro","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":4,"cep":"49100000","rua":"Rua A","bairro":"Salgado Filho","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":5,"cep":"49100000","rua":"Rua A","bairro":"Atalaia","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":6,"cep":"49100000","rua":"Rua ABC","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":7,"cep":"49100000","rua":"Rua C","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":8,"cep":"49100000","rua":"Rua A","bairro":"Centro","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":9,"cep":"49100000","rua":"Rua A","bairro":"Cirurgia","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":10,"cep":"49100000","rua":"Rua A1","bairro":"Bairro América","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":11,"cep":"49100000","rua":"Rua A","bairro":"Siqueira Campos","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":12,"cep":"49100000","rua":"Rua A","bairro":"Getúlio Vargas","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}]

Fonte: Autores (2022).

```
{ "idEndereco":7,"cep":"49100000","rua":"Rua C","bairro":"Jd. Rosa Elze","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":8,"cep":"49100000","rua":"Rua A","bairro":"Centro","cidade":"São Cristóvão","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":9,"cep":"49100000","rua":"Rua A","bairro":"Cirurgia","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":10,"cep":"49100000","rua":"Rua A1","bairro":"Bairro América","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":11,"cep":"49100000","rua":"Rua A","bairro":"Siqueira Campos","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}, {"idEndereco":12,"cep":"49100000","rua":"Rua A","bairro":"Getúlio Vargas","cidade":"Aracaju","estado":"Sergipe","pais":"Brasil"}]
```

**Bot:** Would you like to select a field ? [Yes / No]

**User:** yes

**Bot:** Please, select one of the fields: [idEndereco,cep,rua,bairro,cidade,estado,pais]

**User:** cep

**Bot:** Did you select cep field? [Yes / No]

**User:** yes

**Bot:** Would you like to apply a logic operator ? [Yes / No]

**User:** no

**Bot:** Would you like to relate the result with another table ? [Yes / No]

**User:** no

**Bot:** Please, refresh to use again.

Fonte: Autores (2022).

Ficou destacado em verde na imagem anterior o que acontece quando o usuário deseja uma consulta mais elaborada. Naturalmente a conversa com o *chatbot* se estende para que a interface consiga realizar mais perguntas e alterar seu resultado.

Figura 31 – Resultado da Expressão Relacional

```
[BOT STATE] => end
[TABLE] => endereco
[FIELD] => cep
[OPERATOR] =>
[OPERATOR VALUE] =>
[TABLE TO RELATE] =>
```

Fonte: Autores (2022).

Observe como o protótipo mostra ser possível a formação da expressão, essa seleção de campo é fundamental para que se possa aplicar filtros nas tabelas por meio da interface em linguagem natural.

Comparando o resultado esperado na tabela da imagem 29 com o obtido da interface na imagem 31, observamos que os resultados foram dentro do esperado, o bot state é apenas um atributo interno de controle do *chatbot* e não tem implicação no resultado.

### 4.3 Filtrar os dados de uma tabela

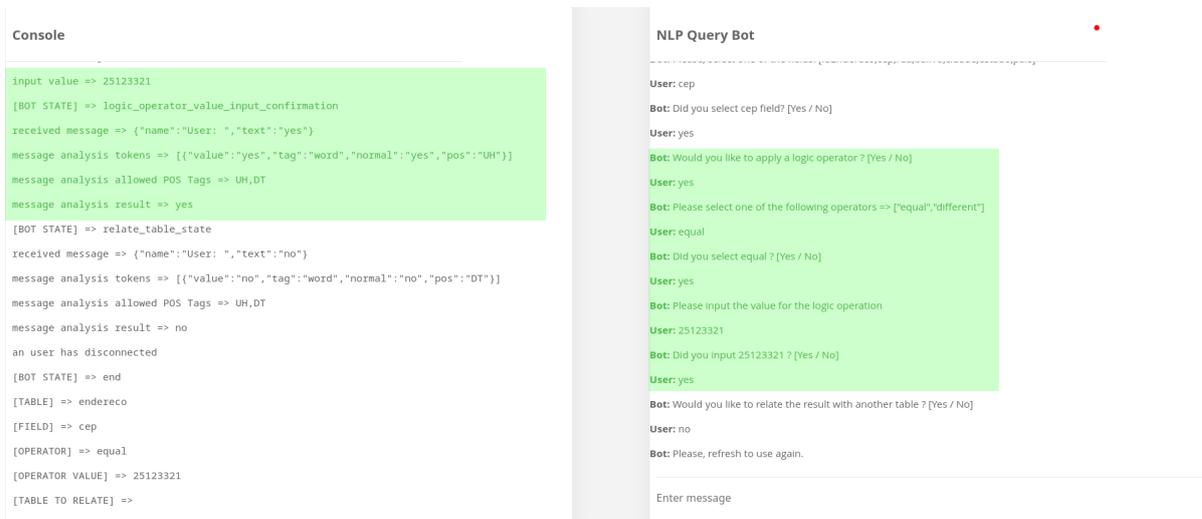
Nesta seção é observado o funcionamento da NLIDB gerando uma expressões relacionais que filtram seus resultados aplicando operações lógicas, a de igualdade e a de diferença. Para que seja possível a utilização do filtro, é preciso selecionar um dos campos da tabela. O comportamento do protótipo é ilustrado e comparado nas próximas imagens e é o equivalente das consultas *SELECT \* FROM endereco WHERE cep = 25123321* e *SELECT \* from estudante WHERE mc != 5*.

Figura 32 – Tabela ilustrando o que foi utilizado em linguagem natural, qual o objetivo em SQL e qual a expressão relacional esperada

Linguagem	Pergunta
Linguagem Natural	show me estudante where mc is different of 5
SQL	SELECT * FROM estudante WHERE mc != 5
Expressão Relacional	[TABLE] => estudant [FIELD] => mc [OPERATOR] => different [OPERATOR VALUE] => 5 [TABLE TO RELATE] =>

Fonte: Autores (2022).

Figura 33 – NLIDB sendo utilizada para expressão equivalente a `SELECT * FROM endereco WHERE cep = 25123321`



Fonte: Autores (2022).

Destacamos em verde na Figura anterior a extensão do diálogo do *chatbot*, de forma que possamos mostrar como o comportamento inquisitivo continua, guiando o usuário com finalidade de elaborar uma consulta mais complexa.

Figura 34 – Resultado da Expressão Relacional

```

[BOT STATE] => end
[TABLE] => endereco
[FIELD] => cep
[OPERATOR] => equal
[OPERATOR VALUE] => 25123321
[TABLE TO RELATE] =>
  
```

Fonte: Autores (2022).

Comparando o resultado esperado na tabela da imagem 32 com o obtido da interface na imagem 34, observamos que os resultados foram dentro do esperado, o bot state é apenas um atributo interno de controle do *chatbot* e não tem implicação no resultado.

Nas duas figuras a seguir, executamos o mesmo padrão, alterando somente as tabelas, campos e operadores lógicos.

Figura 35 – Tabela utilizando o filtro para SELECT \* from estudante WHERE mc != 5

Linguagem	Pergunta
Linguagem Natural	show me estudante where media is different of 5
SQL	SELECT * FROM estudante WHERE mc != 5
Expressão Relacional	[TABLE] => estudante [FIELD] => [OPERATOR] => different [OPERATOR VALUE] => 5 [TABLE TO RELATE] =>

Fonte: Autores (2022).

Figura 36 – NLIDB sendo utilizada para expressão equivalente a SELECT \* from estudante WHERE mc != 5

The figure consists of two side-by-side screenshots. The left screenshot is a terminal window titled 'Console' showing the bot's internal state and the user's input. The right screenshot is a chat interface titled 'NLP Query Bot' showing a conversation between a user and a bot.

**Console:**

```

input value => 5
[BOT STATE] => logic_operator_value_input_confirmation
received message => {"name": "User", "text": "yes"}
message analysis tokens => [{"value": "yes", "tag": "word", "normal": "yes", "pos": "UH"}]
message analysis allowed POS Tags => UH,DT
message analysis result => yes
[BOT STATE] => relate_table_state
received message => {"name": "User", "text": "no"}
message analysis tokens => [{"value": "no", "tag": "word", "normal": "no", "pos": "DT"}]
message analysis allowed POS Tags => UH,DT
message analysis result => no
an user has disconnected
[BOT STATE] => end
[TABLE] => estudante
[FIELD] => mc
[OPERATOR] => different
[OPERATOR VALUE] => 5
[TABLE TO RELATE] =>

```

**NLP Query Bot:**

```

User: mc
Bot: Did you select mc field? [Yes / No]
User: yes
Bot: Would you like to apply a logic operator? [Yes / No]
User: yes
Bot: Please select one of the following operators => ["equal", "different"]
User: different
Bot: Did you select different? [Yes / No]
User: yes
Bot: Please input the value for the logic operation
User: 5
Bot: Did you input 5? [Yes / No]
User: yes
Bot: Would you like to relate the result with another table? [Yes / No]
User: no
Bot: Please, refresh to use again.
Enter message

```

Fonte: Autores (2022).

Observe que segundo as imagens anteriores o comportamento da interface não mudou. Ela se comportou analogamente ao que foi proposto na Figura 32 quando tentamos realizar a consulta apontada na Figura 35. A resposta novamente foi dentro do que é esperado e pode ser vista no final do input no console em 36.

## 4.4 Relacionar uma tabela com outra tabela

Aqui é abordado a junção de uma tabela a outra, é uma consulta mais complexa, mas que pode ser efetivada para junções simples. Como exemplo utilizamos a seguinte *query* como estudo *SELECT \* FROM estudante JOIN estudante\_disciplinas ON estudante.id = estudante\_disciplinas.estudante\_id JOIN disciplinas ON estudante\_disciplinas.disciplinas\_id = disciplinas.id*. E conseguimos gerar uma relação entre as tabelas na expressão regular conforme mostrado a seguir.

Figura 37 – Tabela utilizando o JOIN

Linguagem	Pergunta
Linguagem Natural	show me estudante and the disciplina they are taking
SQL	SELECT * from estudante JOIN estudante_disciplinas ON estudante.id = estudante_disciplinas.estudante_id JOIN disciplinas ON estudante_disciplinas.disciplinas_id = disciplinas.id
Expressão Relacional	[TABLE] => estudante [FIELD] => [OPERATOR] => [OPERATOR VALUE] => [TABLE TO RELATE] => disciplinas

Fonte: Autores (2022).

Novamente destacamos em verde na imagem a seguir a extensão da conversa com o *chatbot*, ilustrando seu comportamento inquisitivo no que se refere relacionar a tabela inicial com outra tabela. Também foi destacado o resultado ao final do log.

Figura 38 – NLIDB sendo utilizada para expressão equivalente a um JOIN

Fonte: Autores (2022).

Por fim confirmamos como observado ao final do log na imagem 38 que o resultado foi novamente dentro do esperado estabelecido na Figura 37.

# 5

## Conclusão

O objetivo deste trabalho foi elaboramos e construímos uma NLIDB. Sendo o mesmo impulsionado pela tentativa de oferecer uma maneira para que o usuário realize consultas em bancos de dados relacionais sem a necessidade do aprendizado de linguagens técnicas como o SQL. Para que a esta construção fosse possível, realizamos um levantamento de estudos similares que mostraram a elaboração ou construção de outras NLIDBs, interfaces com o mesmo objetivo mas que operavam de maneiras diferentes.

A NLIDB apresentada neste trabalho, funcionou de forma inquisitiva, direcionando o usuário através de perguntas e utilizando apenas do NLP para o POS TAGGING. Os estudos de caso indicaram respostas dentro do que era esperado na consulta equivalente em SQL, o que pode indicar um caminho nesta maneira de elaboração da expressão relacional por meio do processamento da linguagem natural. Ainda que a significação da linguagem natural para expressões relacionais funcione apenas para os casos mais simples, ela permite que o usuário realize consultas em bancos de dados relacionais sem precisar saber SQL.

Como trabalhos futuros, uma vez que o modulo desenvolvido no trabalho de outro aluno (CARVALHO, 2022), referente a API seja disponibilizado. A interface será retroalimenta com dados mais verossímeis. Esse alimentação de dados pode requerer ajustes na maneira como o JSON que representa expressão relacional é estruturado, ou como a NLIDB espera a resposta da API, ou ainda do fluxo inquisitivo do *chatbot*.

Ainda existem possibilidades de melhoria que podem tomar a forma de outros trabalhos, com o objetivo de tornar esta solução mais amigável. A NLIDB oferece possibilidade de melhorias em alguns aspectos, os que mais chamam atenção imediata são: a quantidade de campos escolhidos durante a seleção das colunas de uma tabela, e a adição dos demais operadores lógicos como  $<$ ,  $>$ ,  $<=$ ,  $>=$ . Em seguida, quando expressões mais complexas forem possíveis, também é interessante expandir a relação das tabelas para mais que duas. A usabilidade também pode ser melhorada, uma vez que o chat não formata as respostas e ainda as apresenta de uma

maneira que não é muito clara e amigável para o usuário final. No momento o a NLIDB funciona apenas com a língua inglesa, expandir para outros idiomas também seria uma forma interessante de aumentar a usabilidade da ferramenta.

# **Apêndices**

# APÊNDICE

# A

## Instruções para Executar a Aplicação Localmente

Para executar o chatbot localmente em sua máquina, para estudo, avaliação ou desenvolvimento siga as instruções:

1. **Obtenha o código fonte** - Para obter o código fonte da aplicação realize uma solicitação por email para [victorcarity@gmail.com](mailto:victorcarity@gmail.com) ou [andre@dcomp.ufs.br](mailto:andre@dcomp.ufs.br), ou verifique se o código foi disponibilizado no repositório do Departamento de Computação.
2. **Instale o NodeJS e o NPM** - É preciso ter instalado em sua máquina o NodeJS e o NPM antes de seguir os próximos passos. As distribuições mais populares do linux disponibilizam o NodeJS em seus gerenciadores de pacote. Para mais informações acesse o cite do NodeJS ([NODEJS...](#), 2022).
3. **Instale as dependências** - Execute o comando `npm install` pelo terminal, na raiz do diretório onde esta o código fonte. Esse comando irá instalar automaticamente as bibliotecas e pacotes necessários para a execução do projeto.
4. **Execute o processo de transpilação do TypeScript** - Execute o comando `npm run build` pelo terminal, na raiz do diretório onde esta o código fonte. Essa etapa é necessária para que o código seja transpilado para javascript, antes de ser executado.
5. **Execute a aplicação** - Execute o comando `npm run start` pelo terminal, na raiz do diretório onde esta o código fonte.
6. **Acesse a aplicação** - Para acessar a aplicação abra um navegador e digite a seguinte *url* `http://localhost:8000/`.

# Referências

- AXIOS website. 2022. <<https://axios-http.com/docs/intro>>. Accessed: 2022-06-13. Citado na página 32.
- 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 12.
- CARVALHO, I. d. C. Construindo uma api para um chatbot que efetua consultas em bancos de dados relacionais. 3 2022. Citado 2 vezes nas páginas 13 e 50.
- CUNNINGHAM, H. A definition and short history of language engineering. *Natural Language Engineering*, Cambridge University Press, v. 5, n. 1, p. 1–16, 1999. Citado na página 12.
- ELMASRI, R.; NAVANTHE, S. B. *Fundamentals of Database Systems*. [S.l.]: Pearson, 2015. ISBN 0133970779. Citado 3 vezes nas páginas 11, 18 e 19.
- ERIC, B. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, v. 21, n. 4, p. 543–565, 1995. Citado na página 17.
- EXPRESS website. 2022. <<https://expressjs.com/>>. Accessed: 2022-06-13. Citado na página 29.
- FRANK, S. C. T.; CHUN-LING, C. Enriching the class diagram concepts to capture natural language semantics for database access. *Data & Knowledge Engineering*, v. 67, n. 1, p. 1–29, October 2008. Citado na página 13.
- HEROKU website. 2022. <<https://www.heroku.com/>>. Accessed: 2022-06-13. Citado na página 32.
- JOHN, T.; YORICK, W. Anniversary article: Then and now: 25 years of progress in natural language engineering. *Natural Language Engineering*, v. 25, n. 3, p. 405–418, May 2019. Citado 2 vezes nas páginas 12 e 15.
- JSON Data Interchange Syntax. Geneva, CH-1204, 2017. Citado na página 26.
- KAMATH, A.; DAS, R. A survey on semantic parsing. *Proceedings of the VLDB Endowment*, v. 5, n. 10, p. 932–943, 6 2012. Citado na página 19.
- KAMATH, A.; DAS, R. A survey on semantic parsing. *CoRR*, abs/1812.00978, 2018. Citado na página 16.
- LI, F.; JAGADISH, H. V. Nalir: an interactive natural language interface for querying relational databases. *SIGMOD '14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, p. 709–712, 6 2014. Citado na página 15.
- LI, F.; JAGADISH, H. V. Nalir: an interactive natural language interface for querying relational databases. *ACM SIGMOD Record*, v. 45, n. 1, p. 6–13, 3 2016. Citado 3 vezes nas páginas 17, 20 e 21.

MELNIKOV, A.; FETTE, I. *The WebSocket Protocol*. RFC Editor, 2011. RFC 6455. (Request for Comments, 6455). Disponível em: <<https://www.rfc-editor.org/info/rfc6455>>. Citado na página 26.

MIGUEL, L.; ANTONIO, F. How to make a natural language interface to query databases accessible to everyone: An example. *Computer Standards & Interfaces*, v. 35, n. 5, p. 470–481, September 2013. Nenhuma citação no texto.

NAZANIN, F. et al. Keyword extraction: Issues and methods. *Natural Language Engineering*, v. 26, n. 3, p. 259–291, May 2020. Citado na página 17.

NODEJS website. 2022. <<https://nodejs.org/>>. Accessed: 2022-06-13. Citado 2 vezes nas páginas 27 e 53.

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 12.

RESHMI, S.; BALAKRISHNAN, K. Implementation of an inquisitive chatbot for database supported knowledge bases. *Sādhanā*, v. 41, n. 10, p. 1173–1178, October 2016. Citado 3 vezes nas páginas 12, 13 e 17.

SEN, J. et al. Athena++: natural language querying for complex nested sql queries. *Proc. VLDB Endow.*, v. 13, n. 12, p. 2747–2759, 8 2020. Citado 2 vezes nas páginas 20 e 21.

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 2 vezes nas páginas 12 e 15.

SOCKETIO website. 2022. <<https://socket.io/docs/v4/>>. Accessed: 2022-06-13. Citado na página 30.

WINK website. 2022. <<https://winkjs.org/>>. Accessed: 2022-06-13. Citado na página 32.

YU, T. et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv, p. 3911–3921, September 2018. Citado 4 vezes nas páginas 12, 15, 16 e 19.