INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE CAMPUS ARACATI
COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALEX LIMA BEZERRA

INTERFACE PARA BANCO DE DADOS RELACIONAL PARA CONVERSÃO DE CONSULTAS EM LINGUAGEM NATURAL NO IDIOMA PORTUGUÊS PARA COMANDOS EM SQL.

ARACATI-CE
2017

ALEX LIMA BEZERRA

INTERFACE PARA BANCO DE DADOS RELACIONAL PARA CONVERSÃO DE CONSULTAS EM LINGUAGEM NATURAL NO IDIOMA PORTUGUÊS PARA COMANDOS EM SQL.

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador (a): Prof. Msc. Francisca Raquel de Vasconcelos Silveira

Aracati-CE
2017

Dados Internacionais de Catalogação na Publicação
Instituto Federal do Ceará - IFCE
Sistema de Bibliotecas - SIBI
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

B574i Bezerra, Alex Lima.
INTERFACE PARA BANCO DE DADOS RELACIONAL PARA CONVERSÃO DE CONSULTAS EM LINGUAGEM NATURAL NO IDIOMA PORTUGUÊS PARA COMANDOS EM SQL / Alex Lima Bezerra. - 2017.
65 f. : il.

Trabalho de Conclusão de Curso (graduação) - Instituto Federal do Ceará, Bacharelado em Ciência da Computação, Campus Aracati, 2017.
Orientação: Profa. Ma. Francisca Raquel de Vasconcelos Silveira.

1. Banco de dados. 2. Processamento de Linguagem Natural. 3. SQL.. I. Titulo.

ALEX LIMA BEZERRA

INTERFACE PARA BANCO DE DADOS RELACIONAL PARA CONVERSÃO DE CONSULTAS EM LINGUAGEM NATURAL NO IDIOMA PORTUGUÊS PARA COMANDOS EM SQL.

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Aprovada em 11 / 10 / 2017

BANCA EXAMINADORA

Fco. Raquel de V. Silveira

Prof. Msc. Francisca Raquel de Vasconcelos Silveira (Orientadora)
Instituto Federal do Ceará - IFCE

Fábio José Gomes de Sousa

Prof. Msc. Fábio José Gomes de Sousa
Instituto Federal do Ceará - IFCE

Paulo

Prof. Msc. Paulo Alberto Melo Barbosa
Instituto Federal do Ceará - IFCE

DEDICATÓRIA

Aos meus pais.
Aos mestres.

AGRADECIMENTOS

Agradeço a Deus pela proteção e força para superar os desafios encontrados nessa caminhada.

Agradeço ao Instituto Federal de Educação, Ciência e Tecnologia do Ceará, campus Aracati pelo ensino, pelos professores e funcionários.

Agradeço à Professora Msc. Francisca Raquel de V. Silveira, por ter deixado que eu fosse o seu orientando e o seu bolsista, pelas correções, pela paciência. Muito obrigado.

Agradeço a minha família pelo apoio.

Agradeço ao Programa Institucional de Bolsas de Iniciação Científica (PIBIC) pelo financiamento da pesquisa.

E, finalmente, a todos colegas e amigos que passaram e ficaram nessa caminhada que durou quatro anos, principalmente Marcos Antônio e Matheus Gurgel.

RESUMO

A quantidade de informações que vem aumentando e que são armazenadas em um meio digital, tal como, em um banco de dados, permite que o conhecimento, presente nas informações, possa ser usado para tomar decisões que mudem as estratégias de um negócio. Então, o acesso aos dados é importante e uma das maneiras de interagir com as informações presentes em um banco de dados é através de uma linguagem de consulta, como: SQL, que possibilita extrair os dados armazenados nos bancos de dados. Contudo, como é uma linguagem técnica, é necessário aprendê-la. Logo, os usuários leigos, que desejam ter o acesso aos dados, podem encontrar dificuldades inicialmente. Além disso, a estrutura do banco de dados, como: as tabelas, os atributos e outras estruturas devem ser conhecidos. Portanto, criar meios que possibilitem o acesso dos bancos de dados por usuários leigos é importante, pois eles terão o conhecimento presente nas informações. Nesse contexto, o processamento de linguagem natural visa entender textos em linguagem natural e uma, das suas aplicações, é para o acesso aos bancos de dados. Assim, possibilitando entender uma consulta aos dados a partir de uma pergunta. Este trabalho tem o objetivo de conceber uma abordagem e implementá-la para que os usuários sem conhecimentos técnicos na área de banco de dados possam acessá-lo. A abordagem deste trabalho utiliza o processamento de linguagem natural na primeira fase e depois são criadas estratégias para relacionar o que foi produzido na etapa anterior com a linguagem SQL. Dessa forma, o usuário leigo pode acessar os dados de uma maneira mais fácil.

Palavras-chave: Banco de dados. Processamento de Linguagem Natural. SQL.

ABSTRACT

The amount of information that is increasing and that are stored in a digital medium, such as in a database, allows the knowledge, present in the information, can be used to make decisions that change their business strategies. Then, access to data is important and one of the ways to interact with the information present in a database through a query language, such as: SQL, which makes it possible to extract the data stored in the databases. However, as it is a technical language, it is necessary to learn it. Soon, lay users, who wish to have access to the data, you may encounter difficulties initially. In addition, the structure of the database, such as: tables, attributes and other structures must be known. Therefore, create means that allow the access of the databases for lay users is important because they will have the knowledge contained in the information. In this context, the natural language processing aims to understand natural language texts and one of its applications, for the access to databases. Thus, allowing to understand a data query from a question. This work has the objective to develop an approach and implement it so that users without technical knowledge in the area of database can access it. The approach of this work uses natural language processing in the first phase and then are created strategies to relate what was produced in the previous step with the SQL language. In this way, the layman user can access the data from an easier way.

Keywords: Database. Natural Language Processing. SQL.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
atri	atributo
BD	Banco de dados
CPF	Cadastro de Pessoas Físicas
dv	dado
IDE	*Integrated Development Environment*
ILNDB	Interface em Linguagem Natural para Banco de Dados
ILNDBs	Interfaces em Linguagem Natural para Banco de Dados
JSP	*Java Server Pages*
LN	Linguagem Natural
num	número
OWL	*Web Ontology Language*
PNL	Processamento de Linguagem Natural
PROLOG	Programação em Lógica
POS Tagger	*Part-Of-Speech Tagger*
SGBD	Sistema Gerenciador de Banco de Dado
SQL	*Structured Query Language* (Linguagem de Consulta estrutura)
tb	tabela
UML	Linguagem de Modelagem Unificada
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Com o crescente número de informações disponíveis e o valor dos dados como forma de conhecimento, logo, é inegável que o acesso a eles é fundamental. O usuário que deseja interagir com os dados, necessita de ferramentas que consigam obter a leitura e também gerir um conjunto grande dos mesmos que estão armazenados. Sem estas ferramentas o valor presente nos dados perde o seu propósito (RAMAKRISHNAN; GEHRKE, 2008).

Um banco de dados é um componente importante para os sistemas de informática, por sua capacidade de armazenamento dos dados, velocidade e otimização. Segundo (RAMAKRISHNAN; GEHRKE, 2008), "um banco de dados é uma coleção de dados que, tipicamente, descreve as atividades de uma ou mais organizações relacionadas". Para (DATE, 2004), o banco de dados é uma coleção de dados persistentes, usados pelos sistemas de aplicação de uma determinada empresa. O primeiro conceito mostra o conhecimento que os dados possuem, pois descrevem as atividades de uma organização e no segundo é descrito a sua forma prática de armazenar os dados.

Os dados armazenados permitem que os gerentes possam tomar decisões que tragam benefícios para uma organização. Alguns exemplos de informações que tem base nos dados (IMPACTA, 2017): o rendimento dos funcionários, a qualidade dos atendimentos, impactos das ações de *marketing*, as preferências dos clientes etc. Além disso, o próprio controle da organização tanto interno quanto externo, otimizado esta tarefa.

Os tipos de usuários que interagem com um banco de dados são: usuários especialistas (administradores de banco de dados), usuários experientes (desenvolvedores de software ou utilizam uma linguagem de consulta para acessá-lo), desenvolvedores de aplicativos (usam uma interface pré-definida dos programas) e usuários leigos (profissionais que necessitam ter acesso dos dados, mas não possuem conhecimentos da linguagem de consulta e sequer sabem como o banco foi estruturado) (SOUZA; CAMPOS; SANTOS, 2006). Logo, os usuários considerados leigos perdem os benefícios que o acesso aos dados trazem, pois deixam de ter os conhecimentos presentes nos mesmos.

O acesso aos dados pode ser feito por uma linguagem técnica e a mais utilizada é a linguagem de consulta estruturada (*Structured Query Language* - SQL). Então, antes de poder visualizar os dados, é necessário aprender uma linguagem de consulta, conforme a sintaxe e a lógica das consultas que trazem os dados. Outro

conhecimento necessário para a tarefa de visualizar os dados é a estrutura do próprio banco de dados, pois ele possui estruturas que devem ser conhecidas e como essas estão interligadas. Essas estruturas são utilizadas na linguagem de consulta, então sem essas informações é impossível criar um comando em SQL para ter os dados (RAMAKRISHNAN; GEHRKE, 2008).

Conforme Silva e Lima (2013), para criar uma consulta que exibe os dados é necessário ter conhecimento sobre como o banco de dados foi desenvolvido, ou seja, saber a estrutura, os nomes utilizados nos esquemas, as tabelas, os atributos e também os relacionamentos entre as mesmas. Ainda mais, o conhecimento da própria linguagem de consulta, a sua sintaxe, os termos que a língua reconhece e a limitação da sua semântica. Portanto, aprender tudo o que foi dito é um fator muito importante para os dias de hoje em que pessoas lidam com os dados em meio digital.

Como é necessário ter conhecimentos a respeito da linguagem de consulta e a estrutura do banco, os usuários que desconhecem essa área podem criar uma resistência em aprender, ou mesmo não querem saber os detalhes do esquema de um banco de dados para consultar os dados (SOUZA; CAMPOS; SANTOS, 2006). Por isso, se torna necessário desenvolver outros meios que beneficiem esses usuários e levar o que é mais significativo em um banco de dados, ou seja, a informação e o conhecimento. Assim sendo, a criação de instrumentos que retiram esses obstáculos que são especificados, tende a trazer várias melhorias para os mesmos. Em outras palavras, o acesso é feito de um modo mais simples.

O modo usual para visualizar os dados em um banco de dados é através da construção de uma consulta que possibilita retornar os dados. Ela pode ser feita em várias linguagens de consultas. Para este trabalho, o foco é a SQL, devido a sua ampla utilização. Em uma consulta em SQL simples temos: `Select * from tbProfessor`. `Select * from` são palavras e símbolo reservados da SQL, enquanto que o termo `tbProfessor` faz parte da estrutura do banco de dados. O significado desse comando é retornar todos os dados presentes na tabela `tbProfessor`. Esse comando é considerado simples, mas o mesmo pode ter mais variações que o torna complexo. É possível especificar os atributos que serão mostrados, ordenar os dados de forma alfabética ou não, pode ser colocado uma cláusula para que os dados não apareçam duplicados, dentre outras especificações. Isso tudo pode ser utilizado de forma individual ou em conjunto. `Select distinct nome from tbProfessor order by nome` é um exemplo que utiliza as variações em um único comando.

Esses são comandos básicos das consultas que podem ser desenvolvidas e existem vários outros comandos bem mais complexos do que simplesmente retornar os dados de uma tabela, por exemplo, relacionar as tabelas, trazer os dados agru-

pados, informação sobre quantidade que envolve média, soma, maiores valores ou menores, dentre outros. Portanto, um usuário leigo pode ter dificuldades, pois é preciso conhecimento para o acesso de um banco de dados.

Criar um meio que facilite a interação dos usuários sem conhecimentos técnicos para a área de banco de dados é um fator importante para os mesmos. Não somente isso, mas também pode ocorrer um ganho de tempo na indústria do software. Logo, é preciso elaborar ferramentas que tornem a comunicação com um sistema, que armazena conhecimento, de modo mais natural. Uma possibilidade para isso é utilizar o que o usuário emprega no seu dia a dia para a comunicação. Ou seja, o usuário pode criar uma consulta em linguagem natural (NANTES, 2008) e uma ferramenta lidar com todos os detalhes que são necessários para gerar um comando em SQL. Em outras palavras, a pessoa escreve uma consulta como: mostre os professores cadastrados; em seguida, a ferramenta converte a mesma para `Select * from tbProfessor`, e, finalmente, é exibido o resultado da consulta para quem solicitou.

O Processamento de Linguagem Natural (PLN) é um ramo da inteligência artificial que estuda meios de interpretar e desenvolver textos em linguagem natural, como: o português, inglês etc. (SILVA; LIMA, 2013), e pode ser usado para diversas aplicações: tradução de um texto de uma língua para outra, corretores ortográficos, sistemas de extração de informação, sumarização, sistema de perguntas e respostas, reconhecimento de voz (BRAGA et al., 2008), além de manipulação de banco de dados (SILVA et al., 2007).

Enfim, além de tentar entender o que o usuário digitou, visto que o processo de compreender uma linguagem natural não é uma tarefa trivial, pois envolve a ambiguidade, onde uma palavra pode ter vários significados. Uma frase pode apresentar mais de um sentido e muitos outros fatores ligados à língua (NANTES, 2008). Ainda mais, é necessário desenvolver meios que relacionam o que foi escrito por uma pessoa com um comando em SQL.

Esse trabalho gera uma interface que utiliza a linguagem natural no idioma português e converte a frase do usuário em uma consulta em SQL. De forma geral, além do processamento da sentença informada pelo usuário, temos que lidar com a sintaxe da SQL e com a estrutura do banco de dados para realizar a conversão de uma consulta em linguagem natural para um comando em SQL. Com isso, os usuários leigos passam a ter, de um modo mais prático, o acesso aos dados do bancos de dados. Essa abordagem foi inspirada no artigo (LI; JAGADISH, 2014) que desenvolve uma interface para o idioma inglês.

1.1 Objetivo

Este trabalho tem como objetivo o desenvolvimento de uma interface em linguagem natural para banco de dados relacional, capaz de converter consultas na língua portuguesa para comandos em SQL. O comando convertido é aplicado ao banco de dados e é obtido um retorno para quem solicitou, sem que para isso, o usuário tenha conhecimento técnico, permitindo que os mesmos possam ter acesso às informações presentes em um banco de dados relacional. As consultas podem envolver as cláusulas: *SELECT, FROM, WHERE, GROUP BY, HAVING* e *ORDER BY*.

1.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma: no capítulo 2 é apresentada a fundamentação para o assunto abordado, mostrando banco de dados, a teoria dos grafos, o processamento de linguagem natural, as arquiteturas de interface em linguagem natural e trabalhos relacionados a mesma temática e os principais desafios para essa área; no capítulo 3 é detalhada a interface em linguagem natural para banco de dados desenvolvida neste trabalho; no capítulo 4 são expostos os resultados neste trabalho; e, por fim, a conclusão e os trabalhos futuros são apresentados no capítulo 5.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem o objetivo de trazer informações de modo que fundamente o trabalho. Ou seja, os conceitos necessários para banco de dados, desde a estrutura até a linguagem em SQL. É detalhado também o processamento de linguagem natural, mostrando os diversos analisadores e os seus componentes. De modo mais específico para a criação de uma interface responsável pela interação homem e máquina, são apresentadas as arquiteturas de interface em linguagem natural, presentes na literatura. O conceito de grafo e algumas características também são apresentados. E, finalmente, são apresentados os trabalhos relacionados a temática abordada neste trabalho.

2.1 Banco de dados

Para Meira (2013), um banco de dados é uma coleção que reúne vários dados que estão relacionados. Quando o termo dado é utilizado nesse conceito, assume o papel de unidade da informação que de forma isolada não tem significado, como, 123.467.789-00, mas dentro de um contexto, tal qual, o CPF do aluno é 123.467.789-00, passa a ter um sentido. Como também, os dados só podem ser úteis se estão armazenados, assim, permitindo as consultas.

O Sistema de Gerenciamento de Banco de Dados (SGBD) é um programa para gerenciar os dados, consiste em uma reunião de vários programas feitos para a definição, construção e manipulação de base de dados. Alguns exemplos de SGBD são: Microsoft SQL Server, MySQL, PostgreSQL e outros. As características desse programa são: Controle de Redundâncias para não conter dados duplicados; Compartilhamento dos dados para permitir a concorrência ao acesso dos dados e garantindo que eles não sejam corrompidos; Controle de Acesso para controlar e definir o acesso que os usuários podem ter, como: leitura, atualização etc.; e outras propriedades (MEIRA, 2013).

Os bancos de dados podem fazer parte de diversas aplicações. Alguns exemplos são: um programa de controle de estoque de matérias de uma loja, aplicações bancárias, sistema para segurança pública e várias outras que necessitam de dados armazenados (MEIRA, 2013).

Segundo Elmasri e Navathe (2011), um modelo de dados é uma coleção de conceitos que especifica a estrutura de um banco de dado. A estrutura consiste nos tipos que os dados podem ser, nos relacionamentos e nas restrições impostas sobre

os mesmos. Além disso, os modelos contêm um conjunto de procedimentos que possibilita recuperar e atualizar um banco de dados. Os autores classificam o modelo de dados a partir dos conceitos que especificam a estrutura do banco de dados, tais como: Modelos de dados de alto nível ou conceituais contendo conceitos que são mais próximos dos usuários no quesito de percepção aos dados; Modelos de dados de baixo nível ou físico possui conceitos que detalham como os dados estão guardados no computador, ou seja, em disco rígido. E, é voltado para especialistas; Modelos de dados representativo apresenta conceitos que são mais fáceis de entender pelo usuário e é mais próximo de como os dados estão organizados e armazenados. São utilizados nos SGBDs comerciais. O modelo relacional e os mais antigos, o modelo de rede e o hierárquico, fazem parte dessa classificação.

Em Elmasri e Navathe (2011), o modelo entidade-relacionamento é o modelo de dados de alto nível muito usado para construir projetos conceituais voltados para aplicação de banco de dados. Esse modelo retrata os dados através de entidades, atributos e relacionamentos.

Conforme Date (2004), uma entidade representa algo do mundo real e com características bem definidas, sobre o qual devemos registrar as suas informações. Para uma empresa de material de construção é interessante guardar informações sobre os seus clientes, fornecedores dos materiais, funcionários, vendas, materiais, dentre outros. As entidades são: clientes, fornecedores, funcionários, vendas, materiais, etc. formam as entidades básicas que a empresa em questão deve armazenar (DATE, 2004).

Além disso, cada entidade é descrita através de um conjunto de atributos. A escolha dos atributos expressa o nível de detalhes sobre o qual devemos armazenar (RAMAKRISHNAN; GEHRKE, 2008). Em outras palavras, cliente pode ter atributos como: nome, telefone, endereço, dentre outros. Como existe uma coleção de entidades, por exemplo, clientes, é necessário ter um atributo que identifica um entre os demais, o qual os autores chamam de chave primária.

E, finalmente, os relacionamentos que ocorrem entre as entidades. Quando duas ou mais entidades são associadas, temos uma relação entre as mesmas (RAMAKRISHNAN; GEHRKE, 2008). Um exemplo utilizando a empresa material de construção, quando um cliente compra algum material é possível ter o relacionamento entre cliente e venda e, do mesmo modo, venda com material. A venda é o conjunto de relacionamentos e deve possuir os dados das entidades envolvidas, cliente e material.

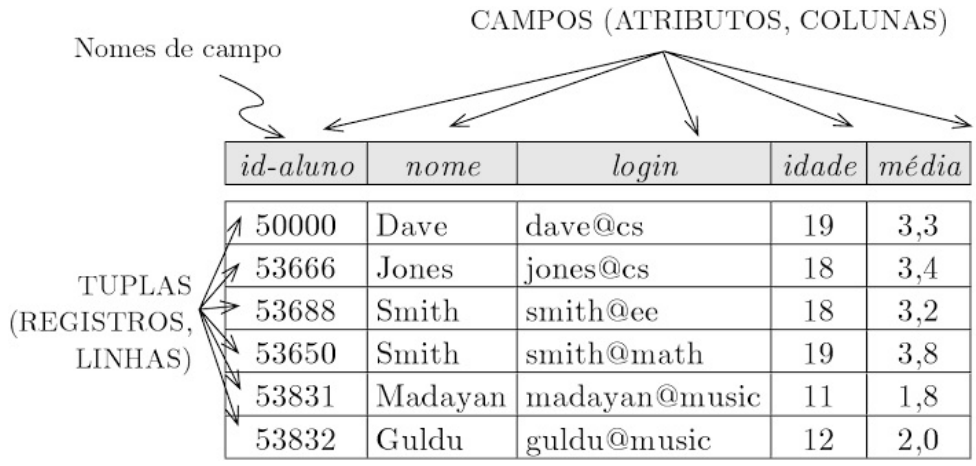
O modelo entidade-relacionamento é convertido para um esquema de banco de dados relacional. Para isso, são feitos mapeamentos das entidades que originam as tabelas com os seus atributos e os relacionamentos entre as mesmas. No modelo relacional são usadas as chaves primárias e estrangeiras para desenvolver os

relacionamentos. Assim, as chaves mantêm uma ligação entre as tabelas envolvidas (ELMASRI; NAVATHE, 2011).

Conforme Ramakrishnan e Gehrke (2008), o modelo de dados relacional é simples, formando assim, um banco de dados que possui uma coleção de uma ou mais relações, em que cada relação é uma tabela com linhas e colunas. Para os autores, esse modelo facilita o entendimento do banco de dados por usuários iniciantes e favorece as consultas aos dados com o uso de uma linguagem de alto nível.

De modo geral, um banco de dados relacional é formado de um conjunto de relações que apresentam nomes diferentes (RAMAKRISHNAN; GEHRKE, 2008). Segundo (RAMAKRISHNAN; GEHRKE, 2008), uma relação é composta por um esquema e uma instância da relação. Em que o esquema da relação torna singular um nome para a mesma, o nome de cada campo (colunas ou atributos) e o domínio de cada campo. O domínio são os valores aceitos pelos campos. E uma instância da relação é uma coleção de tuplas ou registros, onde cada um, presente na coleção, tem a mesma quantidade de atributos que existem no seu esquema. A instância da relação pode ser considerada uma tupla na tabela, em que as tuplas são as linhas e em todas tem o mesmo número de atributos, como é possível ver na figura 1 instância da relação Aluno. As consultas em linguagem natural convertidas nesse trabalho são aplicadas em um banco de dados relacional.

Figura 1 – Instância da relação Aluno.



id-aluno	*nome*	*login*	*idade*	*média*
50000	Dave	dave@cs	19	3,3
53666	Jones	jones@cs	18	3,4
53688	Smith	smith@ee	18	3,2
53650	Smith	smith@math	19	3,8
53831	Madayan	madayan@music	11	1,8
53832	Guldu	guldu@music	12	2,0

Fonte: (RAMAKRISHNAN; GEHRKE, 2008), pág.50.

De acordo com Ramakrishnan e Gehrke (2008), uma consulta de banco de dados relacional ou simplesmente consulta, consiste em uma pergunta sobre os dados e um retorno em resposta na forma de tabela. Como exemplo, podemos querer encontrar todos os alunos da cidade do Aracati ou todos os alunos que fazem o curso de Hotelaria. Para (RAMAKRISHNAN; GEHRKE, 2008), uma linguagem de consulta é uma linguagem especializada para escrever consultas. A SQL (*Structured Query Lan-*

guage) ou linguagem de consulta estruturada (*Structured Query Language*) é a mais utilizada comercialmente para as consultas (RAMAKRISHNAN; GEHRKE, 2008).

Um exemplo de consulta em SQL que mostra como as tabelas de um banco podem ser consultadas:

```
SELECT * FROM Aluno WHERE cidade = 'Aracati'
```

O símbolo asterisco `*` mostra todos os atributos do esquema da relação. O nome do esquema fica depois do termo `FROM`. Na cláusula `WHERE` temos a condição da `cidade` que restringe o retorno. O comando, presente no exemplo, possibilita exibir as instâncias que tenham o atributo `cidade` igual ao valor `Aracati`. Por exemplo, essa consulta tem o resultado mostrado na figura 2.

Figura 2 – Resultado da consulta.

id	nome	telefone	cidade
1	Ernesto Cesário	(88)8823-2398	Aracati
2	Angélica de Sousa	(88)9214-2343	Aracati
3	Rebeca Costa	(88)9982-2398	Aracati
4	Érica Andrade	(88)9974-1253	Aracati
5	Carolina de Sousa	(85)9823-2343	Aracati
7	Pedro de Alcântara	(88)9645-9843	Aracati

Fonte: Elaborado pelo autor.

A linguagem SQL não é utilizada apenas para consultas, mas, também, para construção do banco de dados, das tabelas, dos relacionamentos, dentre outros. Além disso, é possível ter ações para inserir os dados, atualizá-los e apagá-los (PUGA; FRANÇA; GOYA, 2013). Neste trabalho o foco é as consultas, logo não são empregadas as outras operações de manipulação de dados (inserção, alteração e exclusão).

Uma consulta é uma investigação feita no banco de dados com o objetivo de recuperar as linhas (tuplas, registros) que seguem certas condições (PUGA; FRANÇA; GOYA, 2013). Dependendo da consulta mais recursos, da SQL, são necessários. Para o nível de recursos as consultas podem ser definidas, segundo os autores, como:

- Consultas simples.
- Consultas baseadas em condições simples ou compostas.
- Consultas que recuperam dados de diferentes tabelas (junções).
- Agregação e agrupamento.
- Consultas com resultado de outras consultas.

Consulta simples: obtém todas as linhas e colunas de uma tabela e a indicação das colunas (atributos) que devem ser mostradas (PUGA; FRANÇA; GOYA, 2013). A sua sintaxe é:

```
SELECT {* | lista de colunas}
FROM nome da tabela
```

As operações aritméticas: adição, subtração, multiplicação e divisão podem ser feitas nas colunas com números ou em qualquer cláusula do SQL, exceto no FROM (PUGA; FRANÇA; GOYA, 2013). Além disso, é possível renomear um atributo ou expressões complexas, assim, promovendo um melhor entendimento do retorno das consultas. A sintaxe para renomear tem a presença do `as` e o novo nome:

```
SELECT {atributo | expressão [as] novo nome}
FROM nome da tabela
```

Os resultados das consultas aos dados podem ser ordenados, nesse caso é utilizada a cláusula `ORDER BY` e um atributo ou vários atributos. No comando SQL, essa cláusula fica na parte final, e a ordenação é ascendente, que é o padrão, ou o oposto da anterior, ou seja, descendente e para isso, é utilizado o atributo e o termo da SQL (`DESC`) (COSTA, 2006).

Consultas baseadas em condições: para isso, é utilizada a cláusula `WHERE`. O seu objetivo é restringir as linhas pesquisadas. Através de uma expressão lógica que pode envolver: os operadores de comparação (`=, >, <, >=, <=, <>`), os operadores lógicos (`AND`, `OR`, `NOT`) e operações da SQL: `IS (NOT) NULL`, `IS (NOT) LIKE`, `IN` e outros são usados na condição. O resultado de uma expressão é um valor verdade ou falso. Os operadores lógicos `AND` e `OR` são usados quando a restrição possui mais de uma expressão lógica (COSTA, 2006). A sintaxe básica é:

```
SELECT {* | atributos}
FROM {nome da tabela}
WHERE {condição}
```

Consultas que recuperam dados de diferentes tabelas: quando um resultado de uma consulta necessita de mais de uma tabela para criar uma resposta. Existem vários modos de relacionar as tabelas (PUGA; FRANÇA; GOYA, 2013), neste trabalho é usada uma comparação entre a chave primária e chave estrangeira das tabelas que são relacionadas. A sintaxe utilizada é:

```
tabelaA.atributo = tabelaB.atributo
```

, onde `tabelaA.atributo` representa o nome do atributo na `tabelaA` que é chave primária e `tabelaB.atributo` o nome do atributo na `tabelaB` que é chave

estrangeira, correspondendo a chave primária na `tabelaA`. A comparação de igualdade desses dois atributos é apresentada na cláusula `WHERE`.

Agregação e agrupamento: as funções agregadas possibilitam a partir de um conjunto de valores o retorno de um único valor[1]. Segundo (ELMASRI; NAVATHE, 2005), essas funções podem ficar na cláusula `SELECT` ou `HAVING` que é descrita nos próximos parágrafos. Para (COSTA, 2006), SQL possui várias funções agregadas, como:

- `COUNT()`: retorna a quantidade de linhas.
- `MAX()`: retorna o maior valor de uma coluna.
- `MIN()`: retorna o menor valor de uma coluna.
- `SUM()`: retorna a soma dos valores de uma coluna.
- `AVG()`: retorna a média entre os valores de uma coluna.

Conforme ELMASRI e NAVATHE (2005), algumas vezes, devem ser utilizadas as funções agregadas para subgrupos que existem em uma tabela. Além disso, os subgrupos são formados a partir de um atributo em comum, como exemplo, obter a quantidade de alunos que moram em cada cidade, ou seja, o atributo cidade define os grupos. A cláusula `GROUP BY` tem o objetivo de agrupar as linhas de uma tabela, formando grupos que tenham um ou mais atributos iguais. Sua sintaxe básica é:

```
SELECT {atributo | função(atributo)}
FROM {nome da tabela}
GROUP BY {atributo}
```

Segundo ELMASRI e NAVATHE (2005), a cláusula `HAVING` possibilita restringir os grupos formados. Logo, os resultados que são mostrados devem obedecer à condição encontrada na cláusula `HAVING`. Além disso, a posição dessa palavra reservada é apresentada após a cláusula `GROUP BY`.

Consultas com resultado de outras consultas: quando uma consulta necessita ter valores que inicialmente são desconhecidos, é necessário fazer uma subconsulta (PUGA; FRANÇA; GOYA, 2013). Sendo assim, a subconsulta é uma consulta que é utilizada por outra. Neste trabalho é utilizada subconsulta com a seguinte sintaxe:

```
SELECT {* | atributos}
FROM {nome da tabela}
```

[1] https://technet.microsoft.com/pt-br/library/ms173454(v=sql.105).aspx

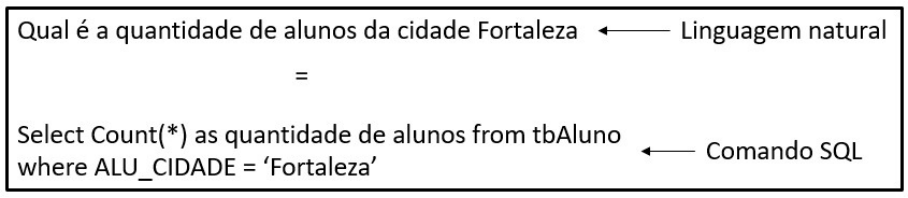
```
WHERE {atributo = (SELECT{atributo}FROM{nome da tabela})}
```

2.2 Processamento de Linguagem Natural

A linguagem natural é aquela utilizada no dia a dia dos seres humanos para compartilharem informações, ou seja, para a comunicação. A partir da mesma as pessoas de vários lugares do mundo podem expressar as suas vontades, o que pensam. Ela tende a evoluir com o passar das gerações, com isso desenvolver regras para as mesmas é uma tarefa difícil (BIRD; KLEIN; LOPER, 2009).

Para Agosti et al. (2003), o conceito de processamento de linguagem natural está voltado para o entendimento de textos. Essa é uma subárea da Inteligência Artificial que objetiva desenvolver modelos da língua que podem ser usados em um computador, para a interação homem e máquina, para a validação das teorias geradas pelos linguistas, dentre outras finalidades. O PNL pode ser usado para aplicações como: tradução, corretores de escrita e concordância, sistemas de extração de informação, sumarização, sistema de perguntas e respostas, reconhecimento de voz, acesso a dados de um banco de dados, análise de sentimento etc. O acesso de um banco de dados geralmente é feito através de uma linguagem de consulta. Na figura 3 temos um exemplo de pergunta feita em linguagem natural e seu correspondente em SQL.

Figura 3 – Exemplo de consulta.



Fonte: Elaborado pelo autor.

Para entender uma linguagem natural e de maneira que possa ser usada por um computador são necessários vários componentes como é demostrado por (AGOSTI et al., 2003): análise morfológica, análise sintática, análise semântica, análise de discurso e análise pragmática. Todas essas análises representam etapas da análise da língua e cada uma produz informações que serão utilizadas para os outros analisadores. Em outras palavras, a análise morfológica fornece um resultado e esse será a entrada do próximo, que é a análise sintática, e, do mesmo modo, segue para os demais.

Para as fases do processamento são necessárias estruturas que colaboram com o processo de entendimento da língua como um todo, sendo: léxico ou dicionário,

gramática, modelo do domínio e modelo do usuário.

Léxico ou dicionário: é uma estrutura de dados que comporta os itens lexicais (palavras) e suas informações. Para (AGOSTI et al., 2003), os sistemas que envolvem o PNL podem ter esse componente. Além disso, as entradas dessa estrutura permite palavras formadas de uma única palavra (por exemplo, casa) ou por uma composição, ou seja, formada por mais de uma palavra (por exemplo, lua de mel). Como também, as informações estão relacionadas com a categoria gramatical, o gênero, o número, o tempo para verbos e outras. O número de informações escolhido para compor este componente depende da aplicação.

Neste trabalho, são utilizadas as classes gramaticais para analisar a frase do usuário, constituindo uma parte importante para o desenvolvimento das consultas. Sem as mesmas seria ainda mais complexo converter as consultas de banco de dados.

Gramática: consiste em regras que aceitam as frases que produzem um sentido, logo, são desconsideradas as que não seguem as normas. Para (AGOSTI et al., 2003), é uma estrutura que define, através de regras, quais são as cadeias de sentenças válidas em uma língua. A análise é feita nas classes gramaticais, pois é impossível ter todas as sentenças válidas de uma linguagem natural.

Modelo do Domínio: é um local que armazena vários conhecimentos a respeito do domínio da aplicação. Esse será utilizado pelo sistema para que os termos da sentença tenham uma relação correta na forma semântica, retirando, assim, a ambiguidade lexical. Para determinar as figuras de estilo ou figuras retóricas particulares, esse processo é feito na análise do discurso (AGOSTI et al., 2003).

Modelo do Usuário: responsável por armazenar as informações do usuário do sistema para que a comunicação entre o sistema e o seu utilizador possa ser feita da melhor forma possível. Com esta base a aplicação pode responder de maneira mais adequada às solicitações dos usuários, desde as palavras utilizadas, o nível linguístico e outros (AGOSTI et al., 2003).

O processamento de linguagem natural possui vários analisadores que utilizam as estruturas mencionadas, tais como:

Analisador Léxico: O objetivo deste analisador é encontrar os termos de uma frase, contendo uma ou mais palavras. Para achar as unidades de sentidos de um texto, necessita que as mesmas tenham limites na sua digitação, tais como: pontuação ou espaço em branco. Com todas as palavras encontradas, é o momento de identificar as classes gramaticais das mesmas, este processo é conhecido como etiquetação (AGOSTI et al., 2003). Por exemplo, dada a seguinte sentença: A mulher olha o mar. Tem como etiquetação: A/artigo definido, mulher/substantivo, olha/verbo,

o/artigo definido, mar/substantivo. Segundo (SILVA et al., 2007), este componente pode ser simples, dependendo da estrutura do léxico e dos atributos exigidos pela aplicação.

Analisador Sintático: Para distinguir uma frase adequada de uma inadequada é necessário ver se seus componentes estão na ordem correta para, então, analisar. A sintaxe de uma língua são várias regras que permitem reconhecer a estrutura da frase e a função de cada componente presente na mesma. Este analisador utiliza as informações geradas pelo analisador léxico e o resultado gerado nesse é armazenado em uma estrutura de árvore, criando, assim, uma árvore de derivação, exibindo as ligações entre as palavras da frase (PINTO; PINHEIRO; PEREIRA, 2011). Em (AGOSTI et al., 2003), o autor utiliza a frase “João viu Maria”, como exemplo para mostrar o uso deste analisador. Maria e o verbo viu constituem um conjunto denominado sintagma verbal que tem como núcleo o verbo, que associado a palavra João que é o núcleo do sintagma nominal. Este analisador deve ter, na sua gramática, informações sobre as sentenças válidas, por exemplo, quando uma frase contém um sintagma nominal, e em seguida ocorre um sintagma verbal, formando assim, uma sentença válida.

Análise Semântica: Analisa as frases sintaticamente corretas (PINTO; PINHEIRO; PEREIRA, 2011). Ela consiste em analisar o sentido das palavras na sentença (AGOSTI et al., 2003). No entanto, este analisador passa por um impasse, ambiguidade das palavras (AGOSTI et al., 2003) (PINTO; PINHEIRO; PEREIRA, 2011). Segundo (AGOSTI et al., 2003), como não existe uma correspondência direta entre sintaxe e semântica, uma mesma estrutura sintática pode dar origem a diferentes representações semânticas. Por isso, o módulo de domínio é utilizado para corrigir as interpretações de termos ambíguos (SILVA et al., 2007).

Análise de Discurso: O sentido de uma frase depende muitas vezes das outras frases mencionadas. Um outro exemplo por (AGOSTI et al., 2003), é o caso da palavra "aquilo" na sentença "Pedro não conseguiu aquilo" o seu significado necessita da frase anterior, e a palavra "Pedro" pode trazer sentido para a palavra da próxima frase no texto, como: "Ele sempre quis". O “Ele” equivale à palavra Pedro.

Análise Pragmática: O entendimento de uma frase pode depender do contexto em que ela está inserida. Ou seja, na sentença "Você sabe que horas são?", esta pode ser interpretada de duas maneira: uma pergunta sobre as horas e a outra é que alguém está atrasado (SILVA et al., 2007).

Desafios do processamento de linguagem natural.

Como citado por (AGOSTI et al., 2003), existem vários desafios no desenvolvimento de uma interface para banco de dados ou para outros programas que utilizam

a linguagem natural. Esses desafios podem ser resumidos em ambiguidades, como:

Ambiguidade léxica: o entendimento de uma palavra pode ser complexo, pois pode ter mais de um sentido, mas sendo resolvido com o uso do contexto em que esta palavra está inserida (AGOSTI et al., 2003). Como exemplo a palavra banco, na frase Lívia tentou achar um banco. O banco pode ser: local onde as pessoas depositam o dinheiro ou um lugar para sentar e descansar;

Ambiguidade sintática: pode ocorrer quando uma frase no processo de análise sintática, tem como resultado mais de uma estrutura válida para a sentença. O exemplo encontrado no trabalho (AGOSTI et al., 2003), "O menino viu o homem de binóculos.", a sentença pode ser entendida de duas maneiras, a primeira o menino estava de binóculos ou como uma segunda interpretação de que era o homem que estava com o equipamento;

Ambiguidade semântica: quando a frase em análise tem como resultado mais de um sentido para a mesma, seguindo a ambiguidade sintática. Quando a análise sintática produz mais de uma árvore para uma frase e estas são válidas e as mesmas apresentam análises semânticas válidas (AGOSTI et al., 2003). Este exemplo é utilizado pelo autor, "Pedro viu Maria passeando", esta frase tem dois sentidos em que Pedro estava passeando e viu Maria, ou Maria estava passeando e Pedro a viu;

Ambiguidade anafórica: acontece quando uma anáfora pronominal pode estar ligada a duas ou mais palavras anteriores e diferentes. O exemplo citado por (AGOSTI et al., 2003), "o ladrão entrou na casa do prefeito e tirou toda a sua roupa", na sentença a palavra "sua" pode estar ligada com ladrão ou prefeito.

2.3 Interface em Linguagem Natural para Banco de Dados-ILNDB

Uma interface tem o objetivo de converter a comunicação entre as entidades. Logo, assegurando entendimento da mensagem para cada entidade envolvida (AGOSTI et al., 2003). Assim, uma Interface que utilizada a linguagem natural para o acesso a um banco de dados admite que os usuários possam interagir com um banco de dados utilizando a linguagem natural, por exemplo: o português. Eles poderão ter acesso aos dados sem ter conhecimentos referentes ao banco, podendo escrever um comando e a interface converte em uma linguagem de consulta.

Para (AGOSTI et al., 2003) uma ILNDB possui quatro tipos de arquitetura como: Sistemas baseados em Comparação de Padrões, Sistemas baseados em Sintaxe, Sistemas baseados em Gramática Semântica, Sistemas baseados em Representação Intermediária. Cada arquitetura será descrita:

Sistemas baseados em Comparação de Padrões esta arquitetura foi a pri-

meira utilizada para algumas interfaces em linguagem natural. Não tem a presença do analisador sintático, logo, não possui uma gramática. Por isso, a análise da sentença do usuário era feita utilizando um conjunto de padrões ou palavras-chave. Além disso, o seu uso é para sistema com pouca interação e também, a sua implementação é mais fácil. A desvantagem do uso desta arquitetura é notada com a sua forma simples de análise da frase, tendo resposta não condizente a pergunta do usuário (ANDROUTSOPOULOS; RITCHIE; THANISCH, 1995).

Para demonstrar esta arquitetura (AGOSTI et al., 2003) coloca o exemplo de uma tabela com o nome produto e seus atributos Cod_Produto, Nome_Produto, Peso _Produto e Preço_Produto. Para entender uma frase feita pelo usuário é utilizado um padrão, como:

- Padrão: ..."preço | quanto custa".... "produto"... <Nome do Produto>
- Ação: exibir Preço_Produto onde Nome_Produto = Nome do Produto

O padrão especifica que na frase do usuário tenha a palavra preço ou quanto custa, seguido do termo produto e no final um nome de produto. Quando o usuário insere uma consulta com este padrão, o retorno é uma tupla que contém o preço do produto (AGOSTI et al., 2003).

Segundo (AGOSTI et al., 2003) esta arquitetura foi utilizada no sistema ELIZA que simula uma conversa humano. O programa foi modelado para ter a função de um terapeuta. Essa abordagem é utilizada hoje com variações permitindo fazer uma análise mais profunda com melhores respostas para o usuário.

Neste trabalho é utilizada essa arquitetura, mas o usuário não é obrigado a escrever os mesmos termos do padrão. Para isso, usamos uma base com os sinônimos para os nomes das tabelas e atributos. Como também, uma medida de similaridade para os termos do banco de dados que podem ser diferentes das palavras originais como: nome e no banco PRO_NOME. Logo, o usuário não terá de seguir um padrão tão rígido e nem o desenvolvedor precisará fazer muitos padrões.

Sistemas Baseados em Sintaxe a frase do usuário tem uma árvore produzida no analisador sintático. Além disso, a gramática possui estruturas sintáticas das perguntas do usuário com relação ao banco de dados. Essa árvore é mapeada para um comando em uma linguagem de consulta. Para isso, utiliza regras que correspondem a estrutura gerada da árvore sintática. A desvantagem existe na dificuldade de desenvolver estas regras que podem ser mapeadas e o seu uso é para um banco de dados específico (ANDROUTSOPOULOS; RITCHIE; THANISCH, 1995).

Sistemas baseados em Gramática Semântica é produzida uma árvore com os elementos semânticos que restringem as consultas do usuário. A gramática semân-

tica não corresponde aos elementos sintáticos como verbos, nomes, sintagmas. Com isso, erros que podem ocorrer com a utilização da análise sintática deixam de existir. Contudo, é necessário construir modelos individuais para cada sistema. Enquanto o Sistema Baseado em Sintaxe usa os elementos sintáticos e podem ser reaproveitados para outros sistemas (ANDROUTSOPOULOS; RITCHIE; THANISCH, 1995).

Sistemas Baseados em Representação Intermediária a frase do usuário passa para uma linguagem intermediária, que é independente do banco de dados. Essa é convertida para uma linguagem de consulta. Também, para formar a linguagem intermediária são usadas regras semânticas. Esse sistema é modular e os módulos que têm destaques são: à parte linguística e à interface com banco de dados. O Baseado em Representação Intermediária favorecem a transportabilidade da interface em diferentes níveis (AGOSTI et al., 2003).

Conforme Agosti et al. (2003), uma capacidade que todas as ILNBDs devem possuir é a transportabilidade entre os domínios de banco de dados, ou seja, elas devem ser criadas de modo genérico. Com isso, segundo (AGOSTI et al., 2003), não é necessário ficar modificando os recursos de processamento de linguagem natural ou alterando o mapeamento para novas estruturas de dados e criar um outro léxico para um determinado domínio. Assim, uma interface que consegue acessar a um banco de uma loja, pode extrair informações de um banco, por exemplo, em um sistema acadêmico.

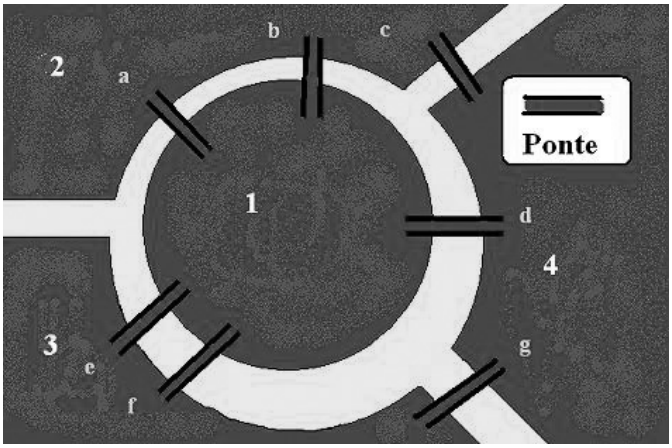
Esse trabalho tem recursos que são genéricos, como a análise léxica, ela não é restrita a um domínio específico e do mesmo modo, os sinônimos, ou seja, podem ser empregados para mais de um domínio. No entanto, a base dos sinônimos não contém todas as palavras, por exemplo, um sinônimo de aluno é discente e de professor é docente, esses não existem na base que armazena os sinônimos. Logo, para obter a transportabilidade é necessária uma base que contemple mais palavras. Mas, mesmo assim, a base em questão possui palavras que pertencem a mais de um domínio, como: estudante, funcionário, clínico, médico, fornecedor, produto, depósito, dentre outras.

2.4 Grafo

Segundo AGUIAR (2009), a teoria dos Grafos surgiu da necessidade de encontrar uma solução para um problema. Este é com relação à realização de um passeio completo nas setes pontes da cidade Königsberg figura 4, onde não deve repetilas. Leonhard Euler solucionou esse problema. Ele provou que é impossível passar por todas, uma única vez. Para isso, modelou cada massa de terra com um ponto e as pontes são equivalentes às linhas que ligam esses pontos. Para que fosse possível

passar pelas pontes sem repetir não poderia haver mais de duas massas de terra com um número ímpar de pontes. Na figura 4 as quatro massas (1, 2, 3, 4) de terra possuem um número ímpar de pontes. A maneira que Leonhard lidou com o problema, vendo como um sistema de pontos e linhas conectadas nos mesmos, passou a ser conhecido por Grafo.

Figura 4 – Esquema de pontes.

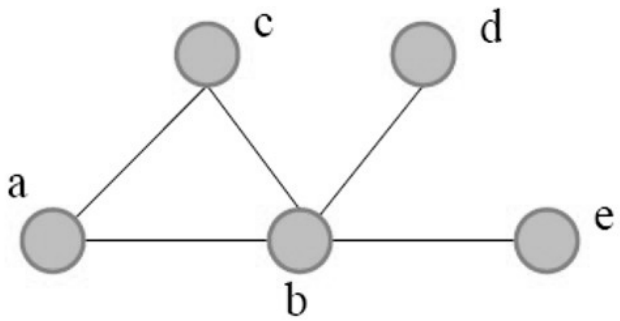


Fonte: Wikipédia-Sete pontes de Königsberg.

Um grafo definido de modo formal é um par de (V, A), onde V é um conjunto finito e não vazio. Os elementos desse conjunto recebem o nome de vértice ou nó. E cada A é uma par de V. O A é conhecido por arcos ou arestas. Então um grafo é um conjunto de vértices ligados entre si por arestas e estas não podem ser paralelas (AGUIAR, 2009).

AGUIAR (2009) coloca um exemplo de grafo, seja G o grafo formado pelos conjuntos V={a, b, c, d, e} e A = {(a,b), (b,c), (a,c), (b,d), (b,e)}, a sua representação é mostrada na Figura 5.

Figura 5 – Grafo G.



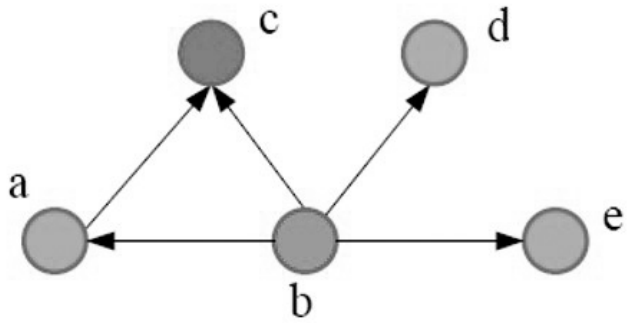
Fonte: (AGUIAR, 2009), pág. 34.

De acordo com AGUIAR (2009), a cardinalidade do conjunto dos vértices determina a ordem (n) e do conjunto das arestas o tamanho do grafo (m). O grau (k) é o número de linhas ligadas a um vértice. Com isso, o tamanho e a ordem do grafo da figura 5 é 5, e o grau do vértice a é k = 2. Além disso, o grafo presente na figura 5 é classificado como um grafo sem laço, pois não tem qualquer conexão de um vértice com ele mesmo. Ou seja, nem um nó possui uma linha que sai dele e volta para o mesmo. Como também, este grafo é considerado simples, pois não tem arestas paralelas ligadas aos vértices e os mesmos não contêm a presença de laços.

Grafo não-direcionado

Para AGUIAR (2009) este grafo pode ser chamado de não-orientado ou não-direcionado quando não existe a presença de uma direção das linhas que ligam os vértices do grafo. Ou seja, tanto faz a direção de a para b ou de b para a, sendo assim, iguais. Caso as linhas tenham uma direção de um vértice para um outro, chama-se direcionado ou orientado ou dígrafo. O Exemplo do não-orientado está na figura 5 e do grafo direcionado fica na figura 6.

Figura 6 – Grafo orientado.



Fonte: (AGUIAR, 2009), pág. 35.

Em um grafo considerado dígrafo temos as terminologias de vértices fontes e sumidouros. O vértice fonte é quando não possui nenhuma linha apontando para o mesmo e aponta para um número maior ou igual a 1 vértices. Já o sumidouro é o oposto, onde ele não aponta para nenhum outro e as quantidade de linhas apontadas para o mesmo é maior ou igual a 1 (AGUIAR, 2009). A Figura 6, o vértice b é um fonte e c um sumidouro. (AGUIAR, 2009) aborda que existem muitos tipos de grafos como: ponderado, desconexo, sem arestas paralelas e sem laço etc.

Representação computacional de um grafo

Em Fernandes (2015) a forma de representar um grafo em código pode ser: lista de adjacência, lista de incidência, matriz de adjacência e incidência. Estes são descritos a seguir:

- A lista de adjacência tem os seus vértices armazenados como registros e cada vértice possui uma lista com os adjacentes armazenados (FERNANDES, 2015).
- A lista de incidência nesta tanto os vértices, quanto as arestas possuem registros e estes são armazenados. Cada vértice armazena as suas arestas incidentes e do mesmo modo, cada aresta armazena os seus vértices incidentes (FERNANDES, 2015).
- A matriz de adjacência utiliza uma matriz com duas dimensões, onde as linhas são os vértices de origem e as linhas representam os vértices de destino (FERNANDES, 2015). Na matriz o elemento da posição (2,3), 2 é um determinado vértice e 3 é outro, se eles têm uma aresta essa posição recebe 1, caso contrário fica 0.
- A matriz de incidência é uma matriz bidimensional do tipo booleano. As suas linhas são os vértices, enquanto as arestas são representadas pelas colunas. As entradas desta tabela indica se as arestas incidem com alguma linha que são os vértices (FERNANDES, 2015).

Esse texto descreveu um pouco da teoria dos grafos e como eles são representados na implementação. No entanto, existem outras informações que não são o foco deste trabalho. Neste é construído um grafo não-direcionado com as tabelas do banco de dados e assim, possibilita criar os relacionamentos entre as tabelas. Esse processo é baseado no trabalho (LI; JAGADISH, 2014). Além disso, a representação empregada para o grafo no desenvolvimento foi a lista de adjacência.

2.5 Trabalhos Relacionados

O trabalho apresentado em (AGOSTI et al., 2003) está organizado inicialmente por um estudo no processamento de linguagem natural. Ele explica as técnicas necessárias para a retirada das informações presentes em consultas. Fala sobre os vários analisadores que tratam as frases. No trabalho ele fala sobre e desenvolve uma Interface de Linguagem Natural para banco de dados onde o acesso é feito através da internet. A interface analisa a estrutura da frase tentando encontrar palavras-chave que podem ser utilizadas para os comandos SQL. As ferramentas utilizadas são: Linguagem de Modelagem Unificada (UML), Java Server Pages (JSP), Java Beans, PROLOG, JSpell. Ele necessita que um usuário com conhecimento no banco de dados configure uma base de sinônimos para os nomes das tabelas e atributos. Possibilitando assim, a correspondência com as palavras que podem ser colocadas na consulta. O mesmo testou o programa antes de disponibilizar, mostra algumas

questões feitas e os comandos gerados automaticamente para a SQL pela aplicação e as que o programa não conseguiu obter respostas. Utiliza um algoritmo que analisa as palavras individualmente e também em grupos. As entidades, atributos e relacionamentos encontrados são guardados em vetores, esses serão empregados no processo de montagem dos comandos SQL. O trabalho (AGOSTI et al., 2003) não reconhecer necessidade de subconsultas. O nosso sistema de interface não necessita que um usuário faça a configuração de um dicionário de sinônimos e é possível fazer subconsulta.

Em (PINTO; PINHEIRO; PEREIRA, 2011), os autores fazem um relato sobre o processamento da linguagem natural, recuperação de informação, inteligência artificial e XML. Para o desenvolvimento da interface deles, tiveram que encontrar os padrões (a sequência das classes gramaticais) das frases interrogativas. Realizaram diversas análises com a finalidade de fixar as regras das prováveis consultas dos usuários aos dados. Utilizaram um banco de dados de Sistema de Controle de Biblioteca e um banco para Gestão de Centros de Custos. Para identificar os nomes dos atributos e das tabelas, eles usam um arquivo XML, com os sinônimos. É apresentado ao usuário todas as tabelas e atributos presentes no banco de dados, para ele determinar os sinônimos.

Para formar a consulta, eles têm que conhecer o banco de dados e atribuir manualmente os mapeamentos. No exemplo abordado: “Qual o livro emprestado possui multa?”, que possui o padrão de classes gramaticais “PRIT ARTD SUB VERB VERB SUB PONT”. O início dessa consulta em SQL fica “SELECT LIVRO.TITULO”. Logo, eles fizeram a correspondência da palavra livro para LIVRO.TITULO do banco de dados. Além disso, o arquivo XML é empregado no processo de tradução da frase em português para o SQL. Para gerar uma consulta eles usam os padrões encontrados, nas sequências que as palavras (classes gramaticais) aparecem na frase interrogativa e depois fazem associações com as cláusulas do select, from e where do SQL. Não implementaram, funções agregadas, cláusulas group by, order by, having e distinct. O nosso trabalho utiliza uma base de dados para os sinônimos, não necessitando que um usuário faça esta configuração. E não associamos nenhuma palavra do usuário com relação ao esquema do banco de dados de forma manual, ou seja, este processo é feito de modo automático. Além disso, são utilizadas três funções agregadas (`COUNT, MAX e MIN`) e as cláusulas indicadas.

Em (LI; JAGADISH, 2014), os autores utilizam o analisador de dependência da Stanford, para ter as dependências presentes entre as palavras na consulta em linguagem natural. As ligações encontradas servem para uma melhor escolha quando mais de uma opção existir no mapeamento das palavras; criaram uma abordagem que irá modificar a sentença original para produzir uma árvore na perspectiva do banco de

dados e inserindo palavras para a consulta ficar mais completa. Eles utilizam o WordNet como base para descobrir os sinônimos dos nomes das tabelas e atributos, com isso os usuários não precisam conhecer as estruturas do banco de dados e fazem uso da raiz quadrada de Jaccard index para avaliar a semelhança entre palavras. Além disso, usam um grafo do esquema para fazer o caminho de junção. A abordagem principal neste trabalho é a "Query tree" uma estrutura intermediária que será modificada, com inserção de palavras, algumas destas pertencentes a linguagem SQL e em outro momento incluindo palavras já mencionadas na frase, desde que tenha uma correspondência entre os esquemas, para ter êxito na geração nos comandos em SQL. Este trabalho foi criado na língua inglesa, e a interface desenvolvida aceita consulta em inglês. O nosso trabalho é inspirado na abordagem deste, mas utilizamos para o português. O nosso não usa a "Query tree", e sim, os termos e classes gramaticais presentes na consulta do usuário no desenvolvimento dos comandos em SQL.

Marins et al. (2015) faz uso da ontologia de domínio para o enriquecimento semântico. Na fase inicial do processamento é encontrado as palavras-chave que contêm termos que representam conceitos e sinônimos. Utiliza ontologias de domínio para interpretar os termos da consulta, com isso, traz novos termos ligados as palavras-chave encontradas e dispõe de outras possibilidades de interpretação semântica da consulta inicial. Para interpretação e enriquecimento da consulta são utilizadas as atividades de análise dos termos da ontologia e agregação semântica. Para achar as palavras-chave que indicam funções agregadas, agrupamento e de ordenação utiliza um conjunto de palavras reservadas. Desenvolveu um conjunto de ferramentas que trabalha com a linguagem OWL que lida com ontologia. As consultas criadas estão no idioma inglês. A aplicação não usa um banco de dados padrão, a partir da consulta é pesquisado na *web* um banco de dados que tenha as palavras-chave ou os termos relacionados. A abordagem encontra os bancos de dados disponíveis e que tenham o mesmo sentido do que foi encontrado na consulta, e traduz a consulta original em interpretações expressas em SQL, que são submetidas aos bancos de dados encontrados. O nosso trabalho utiliza as consultas em português.

3 PROPOSTA

Para que os usuários possam acessar os dados presentes em um banco de dados relacional, eles necessitam conhecer uma linguagem de consulta de dados. Além disso, como o banco de dados foi estruturado, desde os nomes da estrutura, até os relacionamentos das tabelas. Este trabalho vem apresentar o desenvolvimento de uma interface que utiliza a linguagem natural, como meio de interação, para conversão de uma sentença informada pelo usuário em linguagem natural para uma consulta em SQL. Retirando assim, a necessidade do usuário ter conhecimentos prévios. As consultas produzidas podem ser formadas com as cláusulas: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` e `ORDER BY`. Assim, a interface desenvolvida permite que os usuários sem conhecimento da estrutura do banco de dados e da linguagem, visualizem as informações que existem em um banco de dados.

Neste capítulo é definida como a abordagem foi desenvolvida para este trabalho. Inicialmente são exibidos os procedimentos que permitem ao usuário informar a sentença de consulta aos dados em linguagem natural. Na sequência, temos arquitetura geral da ferramenta e as suas etapas. Em adição, é exibido o banco de dados utilizado para a implementação. Como também, os passos que os usuários terão de fazer para ter a sua consulta convertida. E, finalmente, as consultas que este trabalho consegue criar.

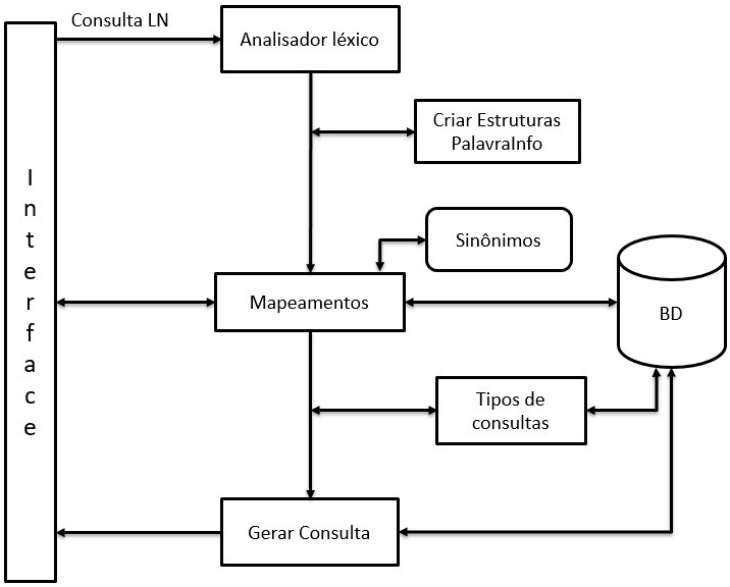
3.1 Visão geral da ferramenta

3.1.1 Arquitetura geral

Na figura 7 temos a arquitetura geral da ferramenta. Como é possível observar, cada etapa possui uma sequência. A entrada para todo o processo é uma consulta em linguagem natural informada pelo usuário na interface. Dada a sentença, é realizado o processamento de linguagem natural, utilizando o analisador léxico. Após isso, é criada uma estrutura com o resultado da etapa anterior (Criar Estruturas PalavraInfo). No próximo processo ocorrem os mapeamentos, visando comparar os termos da consulta com a linguagem SQL e a estrutura do banco de dados. Além disso, algumas vezes o usuário terá que escolher o correspondente de uma palavra que teve mais de um mapeamento. O processo antecedente utiliza os sinônimos. A saída produzida pela etapa anterior é utilizada para os tipos de consultas, neste componente é feito o direcionamento para um tipo que produz uma consulta. Na última etapa (GerarConsulta), a consulta convertida em SQL é enviada para o banco de dados e o

resultado é mostrado para o usuário. As próximas subseções possuem mais detalhes para cada etapa.

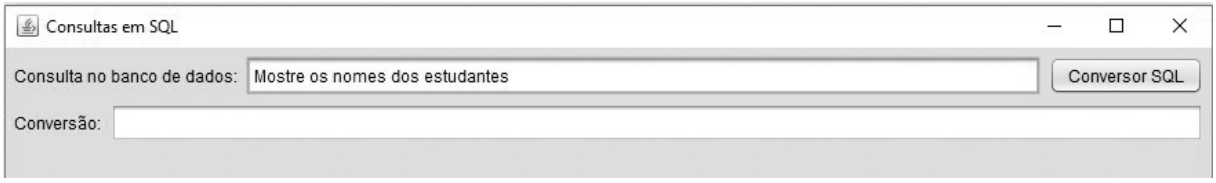
Figura 7 – Arquitetura geral da ferramenta.



Fonte: Elaborado pelo autor.

3.2 Consulta em linguagem natural

Figura 8 – Tela da consulta.



Fonte: Elaborado pelo autor.

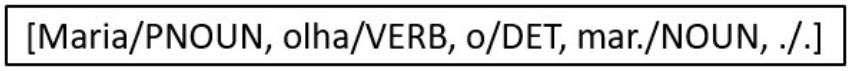
Na figura 8 temos a interface que faz a comunicação com o usuário. No campo específico para inserir uma frase, o usuário informa a sua consulta aos dados em linguagem natural. Após a seleção do botão de conversão SQL, é desencadeada uma sequência de processos para o tratamento da frase e no final a conversão para a linguagem SQL, desde o processamento de linguagem natural, até o último processo que é a exibição do resultado da consulta. No momento do mapeamento, o usuário poderá interagir para escolher certas informações que o sistema considerou idênticas. E, finalmente, mostra-se a consulta em SQL produzida e o resultado da mesma.

3.3 Analisador léxico

Esta é a primeira etapa em que a frase passa. O usuário insere no campo especificado da interface e pressiona o botão conversor SQL. Em seguida, a consulta é enviada para um método que recebe um parâmetro do tipo *String* (consulta do usuário). No método, a sentença tem seus termos separados e recebem as suas etiquetas gramaticais. A biblioteca utilizada para realizar essa ação foi a *Stanford CoreNLP*[1], dela é usada a tokenização e *Part-Of-Speech Tagger* (POS Tagger) (TOUTANOVA et al., 2003). Esta API não possui o português como língua padrão, mas os seus algoritmos permitem utilizar outras línguas (GROUP, 2017). O modelo utilizado neste trabalho foi treinado com o algoritmo *Cyclic Dependency Network* (TOUTANOVA et al., 2003) disponível no *Stanford CoreNLP*, cujo modelo foi desenvolvido por (ERICK, 2017)[2]. A análise léxica da frase é feita utilizando o modelo em português do Brasil e com *Stanford CoreNLP*.

Para essa análise temos: a tokenização com a função de separar as palavras e os sinais de pontuação e *POS Tagger* para etiquetar os termos da sentença. A etiqueta é com relação à classe gramatical. Na figura 9 representa a saída produzida pelo sistema para a sentença “Maria olha o mar.”

Figura 9 – Análise Léxica.



Fonte: Elaborado pelo autor.

A figura 9 é o resultado do *Stanford CoreNLP* com o modelo treinado em português. As palavras e o sinal de pontuação da sentença são separados e etiquetados com suas respectivas classes gramaticais. O termo PNOUN é a etiqueta para substantivos próprios; VERB para verbos presentes na sentença; DET equivale ao artigo; os substantivos recebem a etiqueta NOUN, enquanto que as pontuações têm a etiqueta ‘.’.

3.4 Estrutura palavraInfo

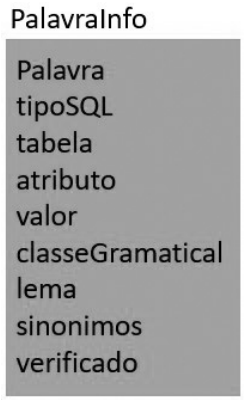
Considerando o exemplo apresentado na figura 9, o analisador léxico separa a sentença "Maria olha o mar." em cinco partes: `Maria/PNOUN`, `olha/VERB`, `o/DET`,

[1] disponível: https://stanfordnlp.github.io/CoreNLP/

[2] O modelo para POS Tagger em português do Brasil é encontrado no link: https://docs.google.com/uc?export=download&id=0B9AWPiAx5i1lb3ZLWVVCTkJBSTQ

`mar/NOUN` e `./..` O próximo processo divide cada parte da saída do analisador léxico em duas estruturas: (i) palavra (parte anterior ao símbolo / (barra)) e (ii) classe gramatical (parte posterior ao símbolo / (barra)). Nesse momento, é criada uma estrutura, aqui denominada "PalavraInfo", para cada uma das palavras identificadas pelo analisador léxico, com as informações que foram separadas e com atributos que representam. De forma geral, os atributos contidos na estrutura PalavraInfo armazenam as informações obtidas antes e durante o processo de mapeamento. Logo, os atributos palavra e classeGramatical são os primeiros a serem preenchidos com o resultado da análise léxica. Os outros serão preenchidos durante o mapeamento, como: tipoSQL, o nome da tabela, nome do atributo, o nome do valor, lema, a lista de sinônimos e verificado.

Figura 10 – Estrutura para as palavras.



Fonte: Elaborado pelo autor.

Especificamente, cada atributo apresentado em PalavraInfo, conforme figura 10, é detalhado a seguir:

- **Palavra** armazena um termo da consulta em linguagem natural. E, todos atributos posteriores estão relacionados a esse.
- **tipoSQL** armazena o papel da palavra em relação à consulta SQL, por exemplo, um termo considerado uma tabela, recebe a classificação tb. Os papéis podem ser: select, ordem, infExib, igual, maior, menor, conte, grupo, tb, atri e dv. O papel select é para uma palavra que tenha significado da cláusula `SELECT`, ordem para um termo que indique ordenação, infExib é para um termo que tenha o sentido de mostrar todos os atributos de uma tabela, igual para uma palavra que denote o sinal de igualdade, maior é para uma palavra que indique o maior e do mesmo modo para o menor, conte para palavra que traga a ideia da função agregada `COUNT`, grupo para termo que signifique agrupamento, atri para os atributos e dv para valores mapeados.

- **tabela** armazena o nome da estrutura existente no banco de dados que representa uma tabela. Os nomes das estruturas do banco de dados podem ser diferentes das palavras presentes na consulta em linguagem natural informada pelo usuário (LI; JAGADISH, 2014). Por exemplo, na frase "Mostre os cursos cadastrados", a palavra cursos se refere a tabela que armazena os dados dos cursos, representada por tbCurso no banco de dados. Logo, tbCurso deve ser armazenado nesse atributo.

- **atributo** inclui o nome do atributo como existe na estrutura do banco. Tendo como exemplo, na consulta "Mostre os nomes dos professores", o termo nomes é considerado um atributo contido na tabela professor, na qual a sua escrita equivale ao PRO_NOME na estrutura da tabela.

- **valor** salva o nome que representa um dado. Por exemplo, para o termo Ricardo que representa o nome de um estudante, ou seja, durante o mapeamento é encontrado um dado que contém esse termo e esse dado deve ser armazenado neste campo.

- **lema** armazena o lema da palavra presente na sentença do usuário. Para a palavra alunas, como exemplo, é guardado o termo aluno, ou seja, na sua forma canônica, no masculino e singular (LUCCA; NUNES, 2002).

- **sinônimos** armazena os sinônimos da palavra e que são obtidos na API JWNBr WordNet.Br (OLIVEIRA; ROMAN, 2013).

- **verificado** representa um controle de quando uma palavra recebe informações. Este atributo é iniciado como zero e quando é obtido um correspondente recebe um.

Cada termo da frase com sua classe gramatical é transformado na estrutura PalavraInfo, conforme apresentado na figura 10, com diversas informações a serem inseridas no decorrer dos analisadores. As primeiras informações são: a Palavra e a classe gramatical, ou seja, a frase “Maria olha o mar.” tem como resultado a figura 11.

3.5 Mapeamento

Esta etapa consiste em analisar cada palavra presente na consulta informada pelo usurário e armazenando as informações na estrutura apresentada na figura 10. Cada palavra é verificada para encontrar seus correspondentes e, ao final, construir a consulta em SQL. Inicialmente é analisado se cada palavra corresponde a: (i) um termo da sintaxe de uma consulta em SQL; (ii) uma tabela no banco de dados; (iii)

um atributo de uma tabela no banco de dados; ou (iv) um valor armazenado em um atributo de uma tabela no banco de dados.

Figura 11 – Estrutura com as palavras.

Palavra	Maria	olha	o	mar	.
tipoSQL					
tabela					
atributo					
valor					
classeGramatical	PNOUN	VERB	DET	NOUN	.
lema					
sinonimos					
verificado	0	0	0	0	0

Fonte: Elaborado pelo autor.

3.5.1 Palavras reservadas

Para os termos da sintaxe de uma consulta em SQL, temos as palavras reservadas. Cada palavra da linguagem SQL utilizada neste trabalho pode ser expressa pelo usuário por termos sinônimos que são mostrados a seguir:

- `SELECT` = {"retorne", "liste", "mostre", "apresente", "traga", "exiba", "quais","qual"};
- Campos da tabela (*) = {"dados", "informações"};
- Igual = {"igual", "mesmo","mesma", "idêntico", "semelhante", "identico"};
- Maior = {"maior", "maiores", "superior", "superiores", "depois", "após", "apos", "mais", "maximo", "maximos"};
- Menor = {"menor","menores", "inferior","inferiores", "antes", "anterior"};
- Função `COUNT`= {"conte", "quantidade", "numero","número", "numeros","números"};
- `GROUP BY` = {"cada", "por"} e
- `ORDER BY` = {"ordenado", "ordem", "ordenados", "ordenada", "ordenadas"};

Cada palavra da frase é verificada se representa uma palavra reservada, correspondendo a estrutura da sintaxe de uma consulta em SQL. Quando um termo corresponde a algum dos tipos entre chaves acima recebe o valor 1 no campo verificado e uma etiqueta (papel) no TipoSQL referente à relação com SQL, tais como:

- A palavra que teve um correspondente ao `SELECT` recebe a etiqueta **select**;

- `ORBER BY` recebe **'ordem'**;
- Dados (*) recebe **'infExib'**;
- igual recebe a sua etiqueta **'igual'**;
- Maior recebe **'maior'**;
- Menor recebe **'menor'**;
- Função `COUNT` fica **'conte'**;
- `GROUP BY` a sua etiqueta **'grupo'**

3.5.2 Tabelas

Para comparar as tabelas, são obtidas todas as tabelas pertencentes ao banco de dados. Para isso, usamos a consulta:

```
SELECT name
FROM sys.objects
WHERE type_desc = 'USER_TABLE'
```

O resultado dessa consulta é armazenado em um vetor. Nesse processo, são comparadas as palavras que não pertencem as seguintes classes gramaticais: artigo, preposição, pronome e verbo.

Para conferir se uma palavra informada na sentença do usuário corresponde a uma tabela é utilizada a medida de similaridade *Jaccard index* (DEBATTY, 2017). Quando duas palavras são comparadas, a medida retorna um valor entre zero e um. O número um significa diferente e zero exatamente igual. Para fazer a comparação, pré-processamos cada palavra, convertendo todas as letras para minúsculo, retirando os acentos, o caractere *underline* é trocado por um espaço em branco e o prefixo tb é removido.

A medida de similaridade é utilizada, pois no banco de dados as tabelas podem ter a grafia diferente da usual (LI; JAGADISH, 2014), por exemplo: a palavra professor pode no banco possuir sua escrita, tal como: tbProfessor, Professores, com isso, não é feita a comparação de igual para igual. O número escolhido que signifique que uma palavra é igual à outra foi o valor abaixo de 0.4. Este número foi escolhido com base em análises empíricas. Por exemplo, para a frase: mostre as disciplinas. O termo disciplinas é comparado com vários nomes de tabelas que foram tratados e as palavras que têm valores menores que um, para este caso são: tbDisciplina e tbOfertaDisciplina.

Tabela 1 – Resultado da Medida Jaccard index.

Termo da Frase	Palavra original do BD	Palavra tratada do BD	Termo da Frase e Palavra tratada
disciplinas	tbDisciplina	disciplina	0.11111111111111116
disciplinas	tbOfertaDisciplina	ofertadisciplina	0.4666666666666667
alunos	tbAluno	aluno	0.25

Fonte: Elaborado pelo autor.

A tabela 1 exibe o resultado da medida *Jaccard index* para a comparação dos termos da frase com estruturas do banco de dados. O valor escolhido é 0.4, pois possibilita que as duas palavras (tbDisciplina e tbOfertaDisciplina) sejam consideradas diferentes para esta implementação e o usuário, assim, não teria que escolher entre as duas. Se isso não fosse feito o usuário deveria selecionar uma das duas palavras encontradas. Além disso, o valor não poderia permanecer próximo de 0.1, pois como é observado na tabela 1 as palavras comparadas (alunos e aluno) obtêm na medida de similaridade 0.25, sendo assim, o número que represente que estes termos são iguais nesse trabalho foi 0.4.

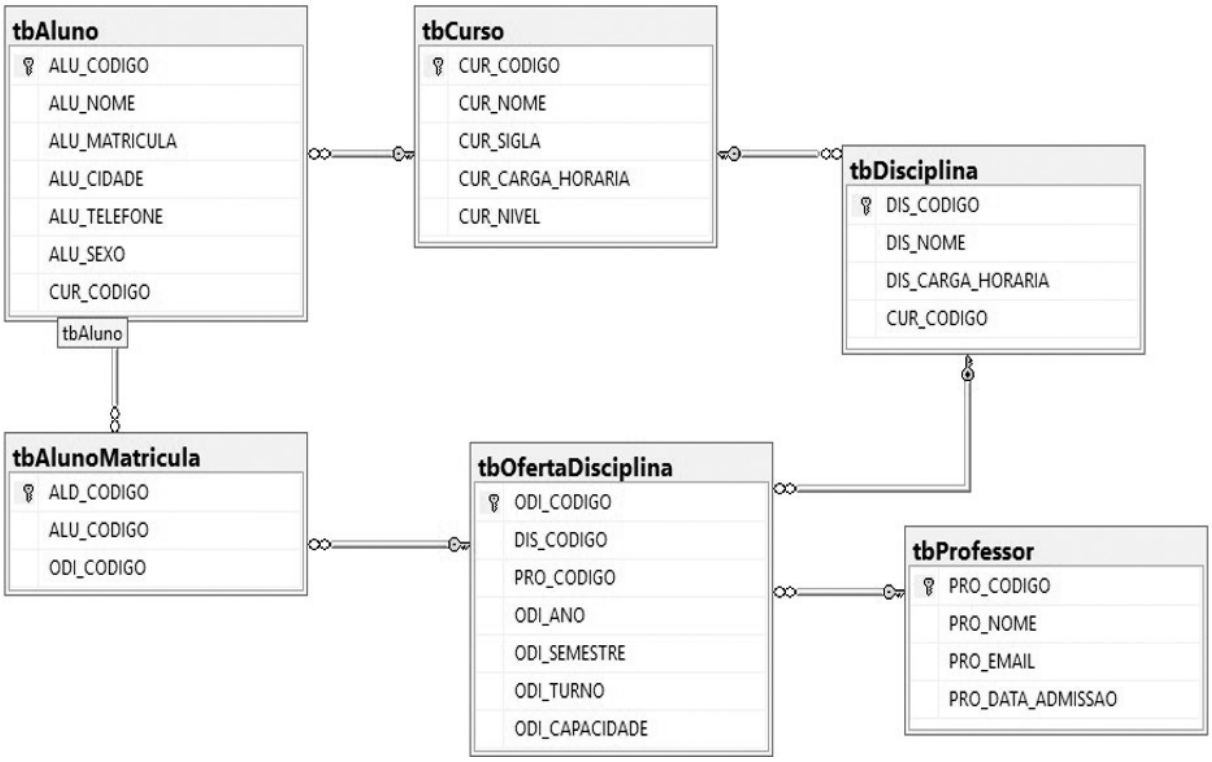
Segundo Li e Jagadish (2014), os usuários no momento da elaboração de uma consulta podem escrever os termos e estes serem diferentes dos nomes encontrados na estrutura do banco de dados, mas com o mesmo significado. Para exemplificar, a sentença “mostre os nomes dos estudantes da cidade do Aracati”, esta consulta possui a palavra estudantes, mas o banco de dados, conforme figura 12, não tem a presença desse termo. Logo, os sinônimos são usados para esses casos.

Para utilizar os sinônimos presentes na API JWNBr WordNet.Br (OLIVEIRA; ROMAN, 2013) as palavras devem estar na sua forma canônica[3]. Essa API retorna uma lista de sinônimos de uma determinada palavra. A lista é armazenada no campo Sinonimo (figura 10). A comparação de cada sinônimo das palavras da sentença do usuário com a estrutura do banco de dados é realizada com a medida de similaridade *Jaccard*.

As palavras consideradas tabelas, no processo de mapeamento, recebem a etiqueta “tb” no campo tipoSQL (figura 10) e no campo tabela é armazenado o nome mapeado da estrutura do banco. Assim, o campo verificado, das mesmas, é inserido o número um, com isso, essas palavras não são utilizadas nos outros mapeamentos. Um caso que pode ocorrer, é quando um mapeamento encontra mais de um nome existente na estrutura do banco de dados, relacionado com o termo pesquisado, logo o usuário deve selecionar a melhor opção para a sua consulta.

[3] O processo de trazer a forma canônica da palavra é feito com a biblioteca Java OpenNLP.

Figura 12 – Banco de dados-Sistema Acadêmico.



Fonte: Elaborado pelo autor.

3.5.3 Atributos

Os atributos das tabelas do banco de dados, obtidas no processo descrito na seção anterior, também são armazenados em um vetor. A consulta que retorna os nomes dos atributos é dada por:

```
SELECT COLUMN_NAME, DATA_TYPE
FROM information_schema.columns
WHERE table_name = 'tabela'
```

O processo de comparação é similar ao realizado para as tabelas. Além disso, como mencionado anteriormente, as palavras que obtêm um correspondente tem em campo verificado (figura 10) o número um, portanto, não são processadas nesse ponto. Como também, as palavras das classes: artigo, preposição, verbo, pronome. Dessa forma, todos os termos que são diferentes das condições apontadas podem ser mapeados.

Quando o usuário escreve a frase, mostre a carga horária das disciplinas, na qual "carga horária" deve ser considerada o atributo DIS_CARGA_HORARIA da tabela disciplina. No entanto, o processo realizado na análise léxica separa as palavras e cada uma será verificada na medida de similaridade. Ou seja, os termos carga e horária são conferidos de forma individual e isso pode acarretar um valor maior no retorno da medida de similaridade, sendo assim, considerados diferentes do atributo

presente na estrutura do banco de dados (DIS_CARGA_HORARIA). Para contornar isso, é utilizado a abordagem n-grama que consistem em formar grupos de palavras em uma sequência organizada de n palavra (MARTHA; BARRA; CAMPOS, 2004). Por exemplo: o unigrama tem um termo (carga), digrama é formado de dois termos (carga horária), para três termos é formado o trigrama (horária da disciplina) até n-palavras. Logo, com a união dos termos (carga horária) é obtido um valor menor na medida em relação quando eles estão separados.

Para este trabalho é feito a união de duas palavras(bigrama) e de três palavras(trigrama). Esse processo de união dos termos é feito com base no valor produzido na medida de similaridade. Em adição, para melhorar o valor de retorno, é feito uma limpeza nas palavras da estrutura do banco, como: removido o *underline* e substituído por um espaço em branco e deixá-las com caracteres minúsculos, como também, para o termo da consulta é retirado os acentos e todas as letras da palavra passam a ser minúsculas. Assim, aprimorando o valor produzido pela medida de similaridade.

Primeiro são unidas duas palavras e o valor gerado pela medida de similaridade é armazenado, da mesma forma, com três palavras. Os valores produzidos nas comparações são examinados e escolhido o menor número e abaixo de 0.4. Se o menor valor veio de duas palavras, a segunda é concatenada com a primeira e na sua classe gramatical é colocada a palavra “removido”. Do mesmo modo, com três, as duas últimas são unidas com a primeira e essas recebem “removido”.

Os termos correspondidos como atributos recebem a etiqueta “atri” no tipoSQL (figura 10). Além disso, o campo atributo (figura 10) é preenchido com o termo mapeado (estrutura do Banco) e o campo tabela inclui a nome da tabela do termo mapeado. Em outras palavras, o termo "carga horária" é considerado um atributo e está contido na tabela disciplina, logo, na sua estrutura PalavraInf (carga horária) possui os nomes da estrutura do banco de dados que representam uma tabela e um atributo.

Quando uma consulta em linguagem natural envolve mais de uma tabela e possui um atributo, pode ocorrer que esse atributo no mapeamento tenha mais de um termo correspondente, nas estruturas das tabelas. Assim, o usuário deve escolher o mapeamento. Por exemplo: mostre os nomes dos alunos do curso Hotelaria, onde nomes é mapeado para ALU_NOME e CUR_NOME.

3.5.4 Dados

A última comparação é realizada com os dados contidos nos atributos das tabelas do banco de dados. As palavras que restaram na consulta, diferentes de artigo, preposição, pronome e verbo, são enviadas com o comando `like '%palavra`

`da frase%'` em todos os atributos pertencentes às tabelas encontradas na frase. O *`like`* tem o objetivo de trazer os dados que contém a palavra da frase na sua escrita. Em outras palavras, se o termo que restou for 'Maria', é colocado no comando:

```
SELECT atributo
FROM tabela
WHERE atributo like '%Maria%'
```

Utilizando a mesma abordagem presente no atributo, como: a união e a comparação é possível mapear os dados. No entanto, um fator que fez aumentar o valor da medida de similaridade para 0.68 está relacionado com os tipos de dados armazenados que podem conter mais de três termos, tal qual, Bacharelado em Ciência da computação. Logo, quando um usuário desenvolve uma consulta: mostre os dados do curso Ciência da Computação e mesmo usado o trigrama para unir (Ciência da Computação), no entanto, o dado possui mais termos e, assim sendo, considerados diferentes, portanto, o aumento da medida possibilita mapear esse tipo de dado. Para essa situação que necessita de mais de três palavras, o valor que é mapeado precisa ficar abaixo de 0.68. A etiqueta para estas palavras é "dv" e é armazenada no tipoSQL da (figura 10). Assim como, o valor (figura 10) é preenchido com o termo mapeado, atributo obtém o atributo do termo mapeado, o campo tabela recebe a tabela do termo mapeado e verificado ganha o valor 1.

3.6 Tipos de consultas

Este trabalho pode desenvolver seis tipos de consultas em linguagem natural. Para isso, é preciso que nas consultas tenham determinadas condições, tais como:

1. Se contém tabela(s): é possível fazer os comandos mais básicos da SQL envolvendo uma tabela ou com duas tabelas necessitando do caminho de junção que nesse trabalho é considerado complexo, é possível ordenar o resultado, para isso, a consulta deve conter um termo para ordenação, como também, o usuário pode especificar os atributos que devem ser apresentados;

2. Se contém tabela(s) e valor (dv): é possível fazer comandos com restrições; com caminho de junção nesse caso tem que ter duas tabelas; subconsulta, sem a presença de sinal de igualdade na consulta do usuário, que nesse trabalho é considerada complexa; é possível especificar a ordem(Alfabética) para exibição do resultado, como também, o usuário pode indicar os atributos;

3. Se contém tabela(s), valor (dv) e sinal de igual (mesmo, igual): desenvolve comandos com restrições, pode ordenar os resultados de forma alfabética, o usuá-

rio pode escolher os atributos que são exibidos e de modo complexo, com: caminho de junção para duas tabelas e subconsulta com sinal igual;

4. Se contém tabela(s), função `COUNT`: é criado o comando da SQL envolvendo a função agregada `COUNT`, pode ter a presença de restrição e quando existir duas tabelas é construído o caminho de junção;

5. Se contém tabela(s), um atributo, função `COUNT` e `GROUP BY`: são feitos comandos com agrupamento e é possível ordenar pelo atributo. Caso houver a presença de um número e um sinal de operação é utilizada `HAVING`. Além disso, com duas tabelas na consulta do usuário é desenvolvido o caminho de junção e

6. Se contém tabela(s), um atributo e função agregada `MAX ou MIN`: são desenvolvidos comandos com essas funções, com também, a subconsulta envolvendo `MAX e MIN`, e criar o caminho de junção quando na consulta houver duas tabelas.

Cada tipo necessita de determinados parâmetros, ou seja, é necessário que na consulta do usuário tenha a presença dos mesmos. Para isso, são utilizadas listas que armazenam as posições do que foi mapeado. Logo, cada possível mapeamento que este trabalho realiza possui uma lista. No total, existem treze listas, a saber, select, tabela, atributo, valor, count, igual, menor, maior, grupo, infExib, nome próprio (encontrado no analisador léxico), ordem e num que é obtido no analisador léxico. A posição é em relação à sequência dos termos na frase, em outras palavras, mostre os alunos cadastrados, nesta sentença a palavra alunos é referente à tabela aluno e se encontra na posição três.

Para preencher as listas mencionadas acima é feito um processo que percorre da esquerda para a direta o tipoSQL de cada termo mapeado e assim, armazenando a posição do mesmo em sua lista. Além disso, como cada lista se destina a um tipo de mapeamento é possível saber se na consulta tem os parâmetros para selecionar um determinado tipo de consulta, ou seja, ocorre um direcionamento. Então, uma consulta que contém uma tabela, vai ter na lista tabela um elemento presente na mesma e sabendo dessa informação é direcionado para o tipo de consulta 1.

Uma primeira análise é sobre o nome próprio que foi identificado pelo *Pos Tagger* (analisador léxico). Ele é verificado se obteve um correspondente no banco de dados, caso não tenha um, essa consulta é encerrada, pois, um nome próprio denota uma restrição ao que o usuário deseja, logo, não tem a necessidade de criar uma consulta. Por exemplo, a consulta retorne os dados do aluno Miguel Mariano, como Miguel Mariano não obtive um correspondente e o usuário quer os dados do mesmo e não de outros estudantes, portanto, não tem porque criar um comando da SQL, pois

não vai retornar nada. Nesse caso, uma mensagem é exibida, tal qual: nome próprio sem correspondente e pede para reformular a frase.

O desenvolvimento de um comando da SQL, como abordado anteriormente, dependerá do que foi encontrado na consulta em linguagem natural. Logo, as listas são essenciais para esse propósito, pois através dos seus métodos são feitos os direcionamentos para escolher um tipo de consulta. Para este trabalho o usuário poderá no máximo incluir na sua consulta duas tabelas.

Nessa parte os tipos de consultas são resumidos em dois grupos. O primeiro grupo são as consultas que consideramos simples, pois elas não precisam de outros processos para a sua formação. O outro grupo são as consultas que consideramos complexas, necessitando de processos: para duas tabelas é necessário formar o caminho de junção e para as subconsultas são necessárias comparações baseadas no trabalho de (LI; JAGADISH, 2014).

3.6.1 Gerar consulta simples

Todas as consultas desenvolvidas nesse trabalho precisam de uma palavra que denote a cláusula `SELECT`, pois ela caracteriza que a consulta do usuário se refere, em termos gerais, uma consulta de dados.

Uma consulta simples corresponde a seguinte sintaxe: `SELECT * FROM{nome da tabela}`. Essa estrutura é gerada quando a frase do usuário possui apenas uma tabela de consulta. Para construir o comando SQL é feita a substituição do `{nome da tabela}` pela tabela encontrada, obtido através do campo tabela da figura 10, assim, a consulta é gerada como uma consulta simples. Desta forma, é necessária na consulta do usuário a existência de palavras que denotem a cláusula `SELECT` e uma tabela do banco de dados. Caso não seja identificada a denotação para a cláusula `SELECT` e para uma tabela, não é produzido nenhum comando em SQL.

Se na lista atributo houver elemento(s) e uma tabela, o asterisco “*” é trocado pelo(s) atributo(s). Um exemplo de consulta: mostre os nomes dos estudantes. Na pesquisa dessa consulta é achado um atributo correspondente ao termo "nomes" que equivale na estrutura do banco de dados figura 12 o nome ALU_NOME e "estudantes" é tbAluno. Este último foi encontrado por causa dos sinônimos. A consulta produzida é `SELECT ALU_NOME FROM tbAluno`.

Uma variante do comando apresentado é `SELECT * FROM{nome da tabela} ORDER BY {nome do atributo}`. A cláusula `ORDER BY` (palavra reservada) tem função de ordenar o resultado de uma consulta a partir de um ou mais atributos. O desenvolvimento dessa consulta é feito da seguinte forma: na posição armazenada

para `ORDER BY` são percorridos os atributos depois do termo que signifique uma ordenação e estes atributos substituem a expressão `nome do atributo`. Um exemplo de comando produzido para uma consulta nesse estilo é "mostre os professores ordenados pelo nome", cuja consulta em SQL é `SELECT * FROM tbProfessor ORDER BY PRO_NOME`.

As restrições aos dados são realizadas conforme o comando `SELECT * FROM {nome tabela} WHERE {atributodv = valordv}`. Quando há um termo que possui correspondente com um dado no banco de dados, esse representará o elemento com o maior número de informações da figura 10, possuindo uma tabela, um atributo a que pertence e o valor mapeado, e com esses dados é possível construir o comando SQL. Portanto, para formar a parte da restrição na cláusula `WHERE` nesse trabalho é necessário um atributo, um sinal de igualdade e a restrição (dado), essas informações vêm do “dv”, ou seja, o atributo está no campo atributo do “dv” (dado) e a restrição fica no campo valor, com isso, é desenvolvido a consulta. Por exemplo, para a consulta "Exiba a cidade do aluno Luís Miguel", temos em SQL `SELECT ALU_CIDADE FROM tbAluno WHERE ALU_NOME = 'Luís Miguel'`, em que `ALU_NOME` vem do campo atributo do dv e "Luís Miguel" é valor do campo dv.

Todos os tipos de consulta mencionados acima podem ser combinados. Por exemplo: mostre os nomes e os telefones dos estudantes de Aracati ordenados pelos nomes. Logo, tendo como consulta `SELECT distinct tbAluno.ALU_NOME, tbAluno.ALU_TELEFONE FROM tbAluno WHERE tbAluno.ALU_CIDADE = 'Aracati' ORDER BY tbAluno.ALU_NOME`.

O desenvolvimento do código é feito por partes. Na primeira é conferida a existência de um termo para ordem e se existir é criada a cláusula `ORDER BY` e os atributos são separados por vírgula. Na restrição, quando um termo for considerado `dv` é construída a cláusula `WHERE` seguido do `dv`. As tabelas identificadas no processo ficam na cláusula `FROM`. E, finalmente, o(s) atributo(s) ou `*` é inserido depois da cláusula `SELECT`.

3.6.1.1 Presença de Função agregada e agrupamento

As funções agregadas que este trabalho desenvolve são: `COUNT`, `MAX` e `MIN`. Quando em uma sentença há uma palavra que represente a função `COUNT` e uma tabela, como nesta consulta "qual é a quantidade de estudantes?", onde o termo quantidade é uma palavra reservada e aluno é identificado como uma tabela, é desenvolvido o comando `SELECT COUNT(*) FROM tbAluno`.

Outro exemplo, "qual a quantidade de alunos de Aracati". Esta sentença tem a presença de uma restrição e o comando em SQL produzido é `SELECT COUNT(*)`

`FROM tbAluno WHERE tbAluno.ALU_CIDADE = 'Aracati'`, adicionando a cláusula `WHERE tbAluno.ALU_CIDADE = 'Aracati'` que representa a restrição.

Para a cláusula `GROUP BY` nesse trabalho, a sintaxe da SQL necessita ter um atributo, a presença do `COUNT` e uma palavra reservada que represente o agrupamento. Quando uma consulta do usuário tem essa formação é feito o comando `SELECT {atributo}, COUNT(*) as {alias} FROM {tabela} GROUP BY{ atributo}`. A consulta "Qual a quantidade de alunos por cidade" corresponde a seguinte consulta em SQL `SELECT tbAluno.ALU_CIDADE, COUNT(*) as alunos FROM tbAluno GROUP BY tbAluno.ALU_CIDADE`. Quando na frase tem a mais um número e um sinal de operação, a saber, maior, menor ou igual é incluída a cláusula HAVING seguida de `COUNT(*) {sinal de operação} {número}` ao final da consulta acima.

`SELECT MIN(tbDisciplina.DIS_CARGA_HORARIA)FROM tbDisciplina` é um exemplo que mostra uma consulta feita com a presença da função agregada `MIN`. Para que este comando seja formado, o usuário deve colocar na sua sentença um atributo e uma palavra reservada para menor. O mesmo vale para a função `MAX` que retorna o maior valor. O padrão é `SELECT {MIN | MAX}({atributo}) FROM {tabela}`. A consulta "Qual a menor carga horária das disciplinas" tem a sua conversão em SQL no início deste parágrafo.

3.6.2 Consultas complexas

Com o processamento da consulta informada pelo usuário, são consideradas como consultas complexas no escopo deste trabalho aquelas que: (i) apresentam duas tabelas; ou (ii) com a presença de subconsulta. De forma geral, para as consultas que apresentam duas tabelas se faz necessário identificar uma condição de junção entre elas. Quanto as subconsultas, neste trabalho são desenvolvidas com funções `MAX` e `MIN`. Por exemplo, na sentença "Exiba as disciplinas com menor carga horária", a informação de menor carga horária deve ser encontrada antes de desenvolver o restante da consulta. Como também, a subconsulta presente na frase "Traga os alunos da cidade do René Alan", em que é necessário saber primeiro a cidade do René Alan para produzir a consulta final.

Figura 13 – Análise léxica subconsulta Min.

Fonte: Elaborado pelo autor.

Para a consulta "Retorne as disciplinas com menor carga horária", a sua aná-

lise léxica é apresentada na figura 13. Esse exemplo necessita de uma subconsulta para obter a menor carga horária entre as disciplinas. Assim, é necessário encontrar primeiro o menor valor e depois identificar qual a disciplina que possui a menor carga horária. A consulta presente neste parágrafo tem os elementos para criar o comando em SQL na versão simples da subseção 3.6.1.1. Uma maneira de criar a subconsulta é identificar o termo anterior à palavra menor, para esse exemplo é `com/ADP`, conforme a figura 13, ou seja, uma preposição. Uma preposição representa uma dependência entre os termos (GALLI, 2017). Desta forma, a conversão fica:

```
SELECT DISTINCT * FROM tbDisciplina
WHERE tbDisciplina.DIS_CARGA_HORARIA = (SELECT
MIN(tbDisciplina.DIS_CARGA_HORARIA) FROM tbDisciplina)
```

A mesma conversão é feita para a função agregada `MAX`. A consulta que possui uma palavra que denote esse emprego terá de maneira equivalente à abordagem que foi mostrada no parágrafo anterior.

Além disso, quando ocorre uma consulta envolvendo duas tabelas é necessário criar o caminho de junção. Neste trabalho não são consideradas as consultas com duas tabelas sem haver junção entre elas, como "Mostre os alunos e os professores", isto é, uma consulta para mostrar os alunos e outra para mostrar os professores. Se isso ocorrer, a ferramenta tenta unir as tabelas, podendo gerar resultados insatisfatórios.

Para formar um caminho que realiza a junção entre as tabelas, foi desenvolvido um algoritmo que apresenta as tabelas presentes em um banco de dados em um grafo, essa abordagem é baseada no trabalho (LI; JAGADISH, 2014). O grafo é iniciado na etapa tipos de consultas da figura 7. A partir da consulta elaborada por (AGUIAR, 2017):

```
SELECT Name as ChaveEstrangeira,
OBJECT_NAME(Referenced_Object_ID) AS TabelaPai,
OBJECT_NAME(Parent_Object_ID) AS TabelaFilha
FROM SYS.FOREIGN_KEYS.
```

O comando anterior retorna as tabelas pais e filhas. Este é aplicado no banco de dados da figura 12 e resultando na tabela 2.

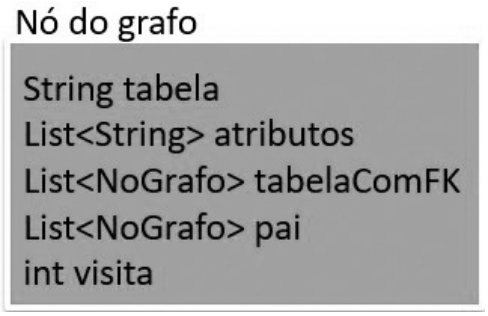
Com uma lista que armazena objetos do tipo figura 14. Onde as informações presentes são: nome da tabela e as listas (atributos, tabelaComFK, pai). Na qual, a lista atributos salva a chave primária e a(s) estrangeira(s) da tabela. O tabelaComFK, que é um campo, armazena as tabelas filhas(adjacente) da tabela em questão e o campo pai armazena a(s) tabela(s) pai(s)(adjacente). O visita é um dado utilizado para não repetir o caminho formado no grafo.

Tabela 2 – Dependências entre as tabelas

	ChaveEstrangeira	TabelaPai	TabelaFilha
1	FK__tbDiscipl__CUR_C__25869641	tbCurso	tbDisciplina
2	FK__tbAluno__CUR_COD__2B3F6F97	tbCurso	tbAluno
3	FK__tbOfertaD__DIS_C__2E1BDC42	tbDisciplina	tbOfertaDisciplina
4	FK__tbOfertaD__PRO_C__2F10007B	tbProfessor	tbOfertaDisciplina
5	FK__tbAlunoMa__ALU_C__31EC6D26	tbAluno	tbAlunoMatricula
6	FK__tbAlunoMa__ODI_C__32E0915F	tbOfertaDisciplina	tbAlunoMatricula

Fonte: Elaborado pelo autor.

Figura 14 – Vértice do grafo.



Fonte: Elaborado pelo autor.

De forma detalhada, para cada tabela do banco de dados é desenvolvida uma estrutura presente na figura 14, ou seja, os vértices do grafo. Assim, permitindo criar o caminho de junção entre duas tabelas posteriormente. A sequência para construir essa estrutura é mostrada a seguir:

1. Criar para cada tabela uma estrutura da figura 14 que equivale ao vértice do grafo e o primeiro campo a ser preenchido é tabela com o nome da mesma. Essa estrutura é armazenada em uma lista denominada banco de dados;
2. Preencher a lista atributos da figura 14, para isso, é utilizado o comado SQL:

   `SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE TABLE_NAME = 'nome da tabela'`, esse comando obtém todas as chaves (Primaria e Estrangeira(s)) contidas na tabela.
3. Para as listas (tabelaComFK, pai) é utilizada o resultado da tabela 2, para inserir essas informações são realizados esses procedimentos:

   a) Para cada linha da tabela 2 que possui elementos presentes nas colunas (TabelaPai e TabelaFilha) é procurado na lista (banco de dados) esses dois elementos;

   b) Encontrados os dois elementos é feito um processo de inserção nas listas internas, como: para o primeiro vértice (TabelaPai) é armazenado na lista

interna tabelaComFK o segundo vértice (TabelaFilha). Para o segundo vértice (TabelaFilha) é guardado na lista pai o primeiro vértice (TabelaPai). Por exemplo, para linha 1 da tabela 2 que possui os elementos: tbCurso e tbDisciplina, o processo encontra tbCurso e tbDisciplina na lista do banco de dados, com isso, é inserido na lista tabelaComFK do tbCurso o vértices tbDisciplina e na lista pai do vértice tbDisciplina é preenchida com tbCurso. Este processo é feito com todas as linhas da tabela 2.

Desse modo, é criado o grafo neste trabalho.

Na figura 12 podemos identificar as ligações entre as tabelas. Então, o algoritmo faz as ligações igualmente e, em adição, é possível percorrer para formar um caminho de junção. Com a lista de nós do grafo é possível chegar em cada tabela. Do mesmo modo, de uma tabela pode-se chegar em outras que estão nas listas internas (adjacentes) de cada tabela. A partir da tabela `tbAluno` é possível chegar à tabela `tbCurso`, pois na lista interna de `tbAluno` tem a tabela pai, neste caso é `tbCurso`, e do mesmo modo, a partir de `tbCurso` é possível chegar em `tbAluno`, pois possui a tabela filha `tbAluno`.

Quando o usuário informa uma consulta envolvendo duas tabelas, é encontrado o caminho que uni as mesmas. Para isso, utiliza o algoritmo de busca em largura no grafo. Esse algoritmo percorre o grafo a partir do vértice a (tabela1) até alcançar o vértice b (tabela2). Assim, formando uma rota que é armazenada e será utilizada para criar o caminho reverso entre b e a. Com isso, é encontrado o caminho entre as duas tabelas.

Os nós que fazem parte do caminho encontrado são comparados de dois em dois. Os atributos em comum são colocados em uma *string* para formar o caminho de junção. Exemplo: `tbCurso` com o atributo `CUR_CODIGO` e `tbAluno` com uma chave estrangeira `CUR_CODIGO`, tendo em comum `CUR_CODIGO`. Forma o caminho `tbCurso.CUR_CODIGO = tbAluno.CUR_CODIGO`. Este caminho fica no comando em SQL após a cláusula `WHERE`.

Inspirado no trabalho (LI; JAGADISH, 2014) que possui subconsultas, criamos três tipos dessas, tais como: "Mostre os alunos da cidade igual à da Angélica de Sousa", "Mostre os alunos da cidade do Ernesto Ribeiro" e “quais são as disciplinas com menor carga horária”. Onde as duas primeiras são parecidas, mas um tem a palavra igual (palavra reservada) e a outra não. E a última envolver função agregada. (LI; JAGADISH, 2014) necessita de uma estrutura em árvore para representar a frase do usuário e modificá-la para produzir o comando SQL. O nosso trabalho usa a própria frase do usuário para fazer as consultas em SQL.

Para desenvolver a consulta "Mostre os alunos da mesma cidade da Angé-

lica de Sousa", é preciso fazer algumas verificações, como: (i) se existe a ordem (`atri`, `ADP` e `dv`); (ii) o nome do atributo com o campo atributo do `dv`, neste caso `ALU_CIDADE` e `ALU_NOME`, são diferentes; e (iii) a última comparação se a tabela dos mesmos são iguais, logo, é possível trazer o mesmo atributo (`atri`) com a restrição (`dv`). Essas comparações são baseadas no trabalho (LI; JAGADISH, 2014). Assim, é produzido:

```
SELECT * FROM tbAluno
WHERE tbAluno.ALU_CIDADE = (SELECT
TOP(1) tbAluno.ALU_CIDADE FROM tbAluno
WHERE tbAluno.ALU_NOME = 'Angélica de Sousa')
```

Para a consulta "Mostre os alunos da cidade igual à da Angélica de Sousa" é realizado um processo de percorrer os termos a partir da palavra igual, indo no sentido da esquerda até encontrar um `atri` e no sentido da direita para encontrar um `dv`. Além disso, é identificado se as tabelas contidas em `atri e dv` são iguais e é feita uma comparação se o atributo do `dv` é diferente de `atri`, significando que o valor apresentado na consulta do usuário não se refere ao atributo informado, sugerindo a existência de uma subconsulta, pois considera-se que o atributo do valor está omitido. Com isso é produzido o seguinte comando em SQL:

```
SELECT * FROM tbAluno
WHERE tbAluno.ALU_CIDADE = (SELECT
TOP(1) tbAluno.ALU_CIDADE FROM tbAluno
WHERE tbAluno.ALU_NOME = 'Angélica de Sousa')
```

A frase "Mostre os alunos da cidade do Ernesto Cesário" tem o mesmo funcionamento das subconsultas apresentadas anteriormente, ou seja, tem a sequência (`atri`, `ADP` e `dv`), os campos atributos das palavras mapeadas como `atri` e `dv` que devem ser diferentes e com as tabelas iguais, contudo, o usuário não é obrigado a colocar um termo que denote um sinal de operação (mesma, igual) na sua consulta. A seguinte consulta em SQL é produzida:

```
SELECT * FROM tbAluno
WHERE ALU_CIDADE = (SELECT
TOP(1) ALU_CIDADE FROM tbAluno
WHERE ALU_NOME = 'Ernesto Cesário'
```

Para essa consulta é usada a classe gramatical `ADP` presente no termo "do", que é uma preposição. Como as preposições ligam termos, é verificado se existe uma entre as palavras. Para o nosso propósito na construção de subconsultas é impossível criá-las sem a presença dessa classe gramatical. Por isso, é uma parte importante, a partir dela é possível ter outra informação, como: a dependência entre os termos.

4 RESULTADOS

Este capítulo mostra os testes feitos na ferramenta que foi implementada. Serão utilizadas diversas consultas em linguagem natural na interface. As consultas possuem, por exemplo: o retorno de todos os dados contidos em uma tabela; indicação dos atributos, assim, não é exibido todos os campos; restrições; ordenação dos resultados; agrupamentos a partir de um atributo; funções agregadas: `COUNT`, `MAX`, `MIN`; sinônimos das palavras pesquisadas; subconsultas e a presença de duas tabelas. O resultado obtido na conversão das mesmas é apresentado. O banco de dados utilizado é o da figura 12. Assim, expondo o potencial da proposta retratada no capítulo anterior.

Para esse trabalho foram usadas as IDEs (*Integrated Development Environment*) Eclipse Mars.1 e NetBeans IDE 8.2, ambas permitem o uso da linguagem de programação Java. Como também, o SGBD (Sistema Gerenciador de Banco de Dados) Microsoft SQL Server Express 2012, onde foi criado o banco de dados utilizado para os testes.

4.1 Simples

Essas consultas seguem o primeiro padrão mencionado na abordagem. As sentenças possuem a presença de uma palavra de retorno e uma tabela mapeada. As palavras: “Retorne”, “Mostre”, “Quais” e “Traga” são consideradas equivalentes da cláusula `SELECT` da SQL. Já os termos: “professores”, “alunos”, “cursos” e “disciplinas” foram mapeados para as tabelas do banco de dados da figura 12. Portanto, a interface utiliza o que foi encontrado na frase, assim, produz os comandos em SQL.

Consulta 1: Retorne os professores

Conversão: `SELECT DISTINCT * FROM tbProfessor`

Consulta 2: Mostre os alunos cadastrados

Conversão: `SELECT DISTINCT * FROM tbAluno`

Consulta 3: Quais são os cursos?

Conversão: `SELECT DISTINCT * FROM tbCurso`

Consulta 4: Traga todas as disciplinas

Conversão: `SELECT DISTINCT * FROM tbDisciplina`

4.2 Sinônimos

As consultas em linguagem natural presentes nesta seção não produzem comandos em SQL de forma direta, pois os termos não correspondem com nenhum nome da figura 12, mas os seus sinônimos permitem essa relação. Ou seja, a palavra estudante e mestre não existem na estrutura do banco de dados da figura 12. No entanto, os seus sinônimos aluno e professor estão presentes na estrutura do banco de dados.

Consulta 1: Liste os estudantes presentes no banco de dados

Conversão: `SELECT DISTINCT * FROM tbAluno`

Consulta 2: Mostre os mestres deste instituto

Conversão: `SELECT DISTINCT * FROM tbProfessor`

4.3 Escolhendo os atributos

Esta etapa utiliza os atributos na consulta, assim, permite que o usuário escolha os campos de cada tabela. Na consulta os termos que serão mapeados para atributos recebem a etiqueta `atri` que permite perceber que o usuário quer exibir determinadas caraterísticas de uma tabela.

Consulta 1: Retorne os nomes dos professores

Conversão: `SELECT DISTINCT tbProfessor.PRO_NOME FROM tbProfessor`

Consulta 2: Exiba os nomes e as cidades dos alunos

Conversão: `SELECT DISTINCT tbAluno.ALU_NOME,tbAluno.ALU_CIDADE FROM tbAluno`

Consulta 3: Traga os cursos, eu quero só os nomes e o nível

Conversão: `SELECT DISTINCT tbCurso.CUR_NOME,tbCurso.CUR_NIVEL FROM tbCurso`

Consulta 4: Retorne os nomes e as cargas horárias das disciplinas

Conversão: `SELECT DISTINCT tbDisciplina.DIS_NOME, tbDisciplina.DIS_CARGA_HORARIA FORM tbDisciplina`

4.4 Restrição

Para testar as restrições são utilizadas as sentenças com termos que possuem relação com os dados. Sendo assim, o mapeamento é feito verificando os dados armazenados. As palavras da consulta que receberam um correspondente referente aos dados fica com a etiqueta `dv`. Esses termos são os elementos que têm mais informações e ficam depois da cláusula `WHERE`. A ordem depois dessa cláusula fica: nome do atributo relativo ao `dv`, em seguida o sinal de igualdade e no final o termo que obteve um correspondente.

Consulta 1: Retorne os dados do professor Raimundo Maciel

Conversão: `SELECT DISTINCT * FROM tbProfessor`

`WHERE tbProfessor.PRO_NOME = 'Raimundo Maciel'`

Consulta 2: Retorne os dados do professor com o nome Raimundo Maciel

Conversão: `SELECT DISTINCT * FROM tbProfessor`

`WHERE tbProfessor.PRO_NOME = 'Raimundo Maciel'`

Consulta 3: Traga os alunos do Aracati

Conversão: `SELECT DISTINCT * FROM tbAluno`

`WHERE tbAluno.ALU_CIDADE = 'Aracati'`

Consulta 4: Mostre os alunos da cidade do Fortim

Conversão: `SELECT DISTINCT * FROM tbAluno`

`WHERE tbAluno.ALU_CIDADE = 'Fortim'`

Consulta 5: Quais são os dados do curso ciência da computação?

Conversão: `SELECT DISTINCT * FROM tbCurso`

`WHERE tbCurso.CUR_NOME = 'Bacharelado em Ciência da Computação'`

Consulta 6: Retorne a carga horária da disciplina Alimentos e bebidas

Conversão: `SELECT DISTINCT tbDisciplina.DIS_CARGA_HORARIA FROM tbDisciplina`

`WHERE tbDisciplina.DIS_NOME = 'Alimentos e Bebidas'`

4.5 Ordem

Este ordena os dados a partir de um atributo. A interface necessita da presença de uma palavra de ordenação para que possa fazer esses comandos. O resultado produzido por esses comandos em SQL apresentam os dados de uma tabela na ordem alfabética. Além disso, para encontrar o atributo que faça a ordenação é percorrido na sequência a palavra descoberta de ordem até o final da frase. O termo achado que é um `atri` não fica na cláusula `SELECT`, sendo assim é removido da lista de atributos.

Consulta 1: Mostre as cidades dos alunos ordenados pela cidade

Conversão: `SELECT DISTINCT tbAluno.ALU_CIDADE FROM tbAluno`

`ORDER BY tbAluno.ALU_CIDADE`

Consulta 2: Retorne os professores ordenados pelos nomes

Conversão: `SELECT DISTINCT * FROM tbProfessor`

`ORDER BY tbProfessor.PRO_NOME`

Consulta 3: Retorne os cursos ordenados pela sigla

Conversão: `SELECT DISTINCT * FROM tbCurso`

`ORDER BY tbCurso.CUR_SIGLA`

Consulta 4: Traga as disciplinas ordenadas pelos nomes

Conversão: `SELECT DISTINCT * FROM tbDisciplina`

`ORDER BY tbDisciplina.DIS_NOME`

4.6 Funções agregadas, GROUP BY e HAVING

Nesta seção é mostrada o uso das funções agregadas, tais como: `COUNT` que retorna o número de linhas, `MAX` que exibe o maior valor de um determinado atributo, e `MIN` que tem o objetivo oposto do anterior, trazendo o menor valor. Logo, as frases precisam ter elementos que correspondem as funções mencionadas. Além das funções agregadas são realizados experimentos com a cláusula `GROUP BY`, que significa agrupar os elementos a partir de um atributo em comum. E, por último, a cláusula `HAVING` que representa uma restrição do grupo formado, necessitando de uma restrição para o agrupamento.

Consulta 1: Qual é a quantidade de estudantes?

Conversão: `SELECT COUNT(*) FROM tbAluno`

Consulta 2: Qual é a quantidade de estudantes do Aracati?

Conversão: `SELECT COUNT(*) FROM tbAluno`

`WHERE tbAluno.ALU_CIDADE = 'Aracati'`

Consulta 3: Qual é o número de professores?

Conversão: `SELECT COUNT(*) FROM tbProfessor`

Consulta 4: Qual é o número de professores com o email raimundo@gmail.-com?

Conversão: `SELECT COUNT(*) From tbProfessor`

`WHERE tbProfessor.PRO_EMAIL = 'raimundo@gmail.com'`

Consulta 5: Qual é a maior carga horária das disciplinas?

Conversão: `SELECT MAX(tbDisciplina.DIS_CARGA_HORARIA) FROM tbDisciplina`

Consulta 6: Qual é a menor carga horária das disciplinas?

Conversão: `SELECT MIN(tbDisciplina.DIS_CARGA_HORARIA) FROM tbDisciplina`

Consulta 7: Qual é a menor carga horária dos cursos?

Conversão: `SELECT MIN(tbCurso.CUR_CARGA_HORARIA) FROM tbCur-so`

Consulta 8: Qual é a quantidade de alunos por cidade?

Conversão: `SELECT DISTINCT tbAluno.ALU_CIDADE, COUNT(*) as alunos FROM tbAluno`

`GROUP BY tbAluno.ALU_CIDADE`

Consulta 9: Qual é a quantidade de estudantes por cidade?

Conversão: `SELECT DISTINCT tbAluno.ALU_CIDADE, COUNT(*) as estudantes FROM tbAluno`

`GROUP BY tbAluno.ALU_CIDADE`

Consulta 10: Qual é a quantidade de estudantes por cidade maior que 10?

Conversão: `SELECT DISTINCT tbAluno.ALU_CIDADE, COUNT(*) as estudantes FROM tbAluno`

`GROUP BY tbAluno.ALU_CIDADE HAVING COUNT(*) > 10`

Consulta 11: Qual é a quantidade de alunos por cidade menor que 10?

Conversão: `SELECT DISTINCT tbAluno.ALU_CIDADE, COUNT(*) as`

```
alunos FROM tbAluno
      GROUP BY tbAluno.ALU_CIDADE HAVING COUNT(*) < 10
```

4.7 Subconsulta

As consultas em linguagem natural nesta seção possuem subconsultas. Essas precisam que uma consulta seja construída antes para depois desenvolver o comando em SQL final. Para este trabalho, considera-se que as subconsultas são notadas com o uso de termos que signifique igual ou com a utilização da classe gramatical preposição. Para as subconsultas relacionadas com as funções agregadas `MAX` e `MIN` é observado se antes da palavra que representa uma função tem uma preposição (`ADP`).

Consulta 1: Retorne as disciplinas com menor carga horária

Conversão: `SELECT DISTINCT * FROM tbDisciplina`

Conversão: `WHERE tbDisciplina.DIS_CARGA_HORARIA = (SELECT MIN(tbDisciplina.DIS_CARGA_HORARIA) FROM tbDisciplina)`

Consulta 2: Retorne as disciplinas com maior carga horária

Conversão: `SELECT DISTINCT * FROM tbDisciplina`

`WHERE tbDisciplina.DIS_CARGA_HORARIA = (SELECT MAX(tbDisciplina.DIS_CARGA_HORARIA) FROM tbDisciplina)`

No caso da subconsulta com a presença de uma palavra com o sentido de igual e este termo permanece antes do atributo encontrado é conferida a sequência `atri+ADP+dv`. Se a sequência existir são comparadas as tabelas dos elementos `atri` e `dv` e tem que ser a mesma. Além disso, os nomes dos atributos devem ser diferentes. A outra subconsulta com a palavra com o significado da anterior, mas que fica entre `atri` e `dv` tem o processo que percorre para a esquerda até encontrar um atributo e para a direita achando uma tabela, com isso, todas as outras comparações são feitas: os nomes dos atributos e as tabelas. Para a subconsulta sem um termo "igual" é feita do mesmo modo, primeiro se contém a sequência `atri+ADP+dv`, depois os atributos são diferentes e por último as tabelas devem ser iguais.

Se os atributos são iguais não há necessidade de subconsulta. Se a condição anterior for diferente e as tabelas iguais é feita a subconsulta. Como as tabelas são iguais, elas compartilham os mesmos atributos. Logo, é possível fazer uma subconsulta com a instância especificada e ter todos os seus atributos, mas o escolhido é o `atri` da frase. Esse é colocado na consulta principal.

Consulta 3: Mostre os alunos da mesma cidade da Angélica de Sousa

Conversão: `SELECT DISTINCT * FROM tbAluno WHERE tbAluno.ALU_CIDADE = (SELECT TOP(1) tbAluno.ALU_CIDADE FROM tbAluno`

`WHERE tbAluno.ALU_NOME = 'Angélica de Sousa')`

Consulta 4: Retorne os alunos da cidade igual do Carlos Eduardo

Conversão: `SELECT DISTINCT * FROM tbAluno`

`WHERE tbAluno.ALU_CIDADE = (SELECT TOP(1) tbAluno.ALU_CIDADE FROM tbAluno WHERE tbAluno.ALU_NOME = 'Carlos Eduardo')`

Consulta 5: Retorne os alunos da cidade do Ernesto Cesário

Conversão: `SELECT DISTINCT * FROM tbAluno`

Conversão: `WHERE ALU_CIDADE = (SELECT TOP(1) ALU_CIDADE FROM tbAluno WHERE ALU_NOME = 'Ernesto Cesário')`

Consulta 6: Retorne os alunos da cidade do Ernesto Cesário ordenado pelo nome

Conversão: `SELECT DISTINCT * FROM tbAluno`

Conversão: `WHERE ALU_CIDADE = (SELECT TOP(1) ALU_CIDADE FROM tbAluno WHERE ALU_NOME = 'Ernesto Cesário')ORDER BY tbAluno.ALU_NOME`

Consulta 7: Retorne os nomes dos alunos da cidade do Ernesto Cesário

Conversão: `SELECT DISTINCT tbAluno.ALU_NOME FROM tbAluno`

`WHERE ALU_CIDADE = (SELECT TOP(1) ALU_CIDADE FROM tbAluno WHERE ALU_NOME = 'Ernesto Cesário')`

4.8 Envolvendo duas tabelas

Este processo necessita do algoritmo descrito na seção das consultas complexas, pois forma o caminho de junção entre as tabelas. Nestas consultas o usuário pode especificar os atributos, ordenar por um atributo e, também, utilizar a função para trazer a quantidade de linhas da tabela.

Consulta 1: Retorne os nomes das disciplinas do professor Raimundo Maciel

Conversão: `SELECT DISTINCT tbDisciplina.DIS_NOME FROM tbProfessor, tbOfertaDisciplina, tbDisciplina`

`WHERE tbProfessor.PRO_NOME = 'Raimundo Maciel' AND tbProfessor.PRO_CODIGO = tbOfertaDisciplina.PRO_CODIGO AND tbOfertaDisciplina.DIS_CODIGO = tbDisciplina.DIS_CODIGO`

Consulta 2: Retorne os nomes dos estudantes do curso ciência da computação

Conversão: `SELECT DISTINCT tbAluno.ALU_NOME FROM tbCurso, tbAluno`

`WHERE tbCurso.CUR_NOME = 'Bacharelado em Ciência da Computação' AND tbCurso.CUR_CODIGO = tbAluno.CUR_CODIGO`

Consulta 3: Retorne os nomes dos estudantes do curso ciência da computação ordenado pelo nome

Conversão: `SELECT DISTINCT tbAluno.ALU_NOME FROM tbCurso, tbAluno`

`WHERE tbCurso.CUR_NOME = 'Bacharelado em Ciência da Computação' AND tbCurso.CUR_CODIGO = tbAluno.CUR_CODIGO`

`ORDER BY tbAluno.ALU_NOME`

Consulta 4: Qual é a quantidade de alunos por nomes dos cursos?

Conversão: `SELECT DISTINCT tbCurso.CUR_NOME, COUNT(*) as alunos FROM tbCurso, tbAluno`

`WHERE tbCurso.CUR_CODIGO = tbAluno.CUR_CODIGO`

`GROUP BY tbCurso.CUR_NOME`

As consultas anteriores sempre necessitam que o usuário selecione a tabela que pertence o termo nome. Uma vez que, os termos pesquisados possuem palavras idênticas em ambas as tabelas.

Consulta 5: Qual é a quantidade de disciplinas do professor Raimundo Maciel?

Conversão: `SELECT COUNT(*) FROM tbProfessor, tbOfertaDisciplina, tbDisciplina`

`WHERE tbProfessor.PRO_NOME = 'Raimundo Maciel' AND tbProfessor.PRO_CODIGO = tbOfertaDisciplina.PRO_CODIGO AND tbOfertaDisciplina.DIS_CODIGO = tbDisciplina.DIS_CODIGO`

5 CONCLUSÃO

A linguagem de consulta estrutura (SQL) permite obter os dados armazenados nos bancos de dados. No entanto, ela é uma linguagem técnica, é necessário aprendê-la. Além disso, para formar um comando da mesma é preciso conhecer a estrutura do banco de dados. Logo, os usuários leigos, que pretendem ter o acesso aos dados, podem encontrar dificuldades. Para contornar essa situação é proposto a criação de uma interface que lide com a linguagem SQL e a estrutura do banco de dados. Este trabalho é inspirado na abordagem presente no (LI; JAGADISH, 2014) que possui uma interface para o idioma inglês.

Nesse trabalho foi desenvolvida uma Interface em Linguagem Natural para Banco de Dados relacional em português e testada com várias consultas em linguagem natural, com: presença de sentenças que significam retorno das informações das tabelas; a utilização de restrições; subconsultas; agrupamento; ordenação do resultado da tabela; funções agregadas, tais como: Count, Max e Min; sinônimos; envolvendo até duas tabelas e o usuário pode escolher os atributos que devem ser mostrados. E, finalmente, é obtido o comando em SQL das consultas. A interface em linguagem natural permite que um usuário possa fazer uma pergunta em português e convertendo-a em um comando do SQL. Esse comando é aplicado no banco de dados e obtém um resultado em forma de tabela para o usuário.

Além disso, diferentemente dos Sistemas baseados em Comparação de Padrões não é preciso criar padrões para cada frase, ou seja, especificar individualmente as sentenças que o sistema aceita. Como é observado, neste trabalho, um tipo pode ser usado para gerar mais de uma consulta. Por exemplo, o padrão da seção Consulta simples, permite desenvolver quatro consultas diferentes e foi criado um padrão. Dessa forma, basta trocar os nomes das tabelas na posição do padrão, assim, não tem a necessidade de ter quatro padrões.

A utilização do processamento de linguagem natural, mesmo que tenha sido só o analisador léxico, possibilitou o desenvolvimento das subconsultas a partir da classe gramatical preposição presente nas frases analisadas. E, o uso dos sinônimos permitiu que o usuário não digite os mesmos nomes encontrados na estrutura do banco. Como também, API WordNet.BR possui palavras que pertencem a vários domínios, logo, pode ser utilizada em outros bancos de dados.

Portanto, os usuários leigos podem fazer uma consulta de dados utilizando o seu idioma, nesse caso é o português, e uma conversão é feita, assim, retirando a obrigação de conhecer: uma linguagem técnica, como os nomes das estruturas

estão no banco e os relacionamentos encontrados no mesmo. Logo, a interface em linguagem natural proposta nesse trabalho teve o seu objetivo alcançado que é de acesso aos dados de modo mais simples.

Como trabalhos futuros serão a inclusão das demais funções agregadas, utilizar outros recursos do próprio processamento de linguagem natural, como, análise sintática, análise semântica etc. Como também, fazer testes com outros domínios de bancos de dados e com usuários leigos. Assim, possibilitarão uma melhor avaliação em forma quantificada, com isso, permitirá utilizar medidas presentes nas literaturas.

REFERÊNCIAS

AGOSTI, C. et al. Interface em linguagem natural para banco de dados: uma abordagem prática. Florianópolis, SC, 2003. Citado 8 vezes nas páginas 23, 24, 25, 26, 27, 28, 31 e 32.

AGUIAR, G. M. *Mapeando dependências entre tabelas.* 2017. https://gustavomaiaaguiar.wordpress.com/2008/09/12/mapeando-dependencias-entre-tabelas/. Acesso em: 22.05.2017. Citado na página 49.

AGUIAR, M. Redes de palavras em textos escritos: Uma análise da linguagem verbal utilizando redes complexas. *Programa de pós-graduaçao em fısca, Universidade Federal da Bahia, Salvador*, 2009. Citado 3 vezes nas páginas 28, 29 e 30.

ANDROUTSOPOULOS, I.; RITCHIE, G. D.; THANISCH, P. Natural language interfaces to databases–an introduction. *Natural language engineering*, Cambridge University Press, v. 1, n. 1, p. 29–81, 1995. Citado 2 vezes nas páginas 27 e 28.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural language processing with Python: analyzing text with the natural language toolkit*. [S.l.]: "O'Reilly Media, Inc.", 2009. Citado na página 23.

BRAGA, D. F. M. M. d. et al. Algoritmos de processamento da linguagem natural para sistemas de conversao texto-fala em português. 2008. Citado na página 15.

COSTA, R. D. C. *SQL Guia Prático - 2a edição*. [S.l.]: BRASPORT, 2006. https://books.google.com.br/books?id=3Lxv-q6-S3MC. ISBN 9788574522951. Citado 2 vezes nas páginas 21 e 22.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004. Citado 2 vezes nas páginas 13 e 18.

DEBATTY, T. *Jaccard index*. 2017. https://github.com/tdebatty/java-string-similarity/blob/master/src/main/java/info/debatty/java/stringsimilarity/Jaccard.java. Acesso em: 06.08.2017. Citado na página 40.

ELMASRI, R.; NAVATHE, S. *Sistemas de Banco de Dados–Fundamentos e aplicações, tradução da 6a. ed.[por] Daniel Vieira*. [S.l.]: São Paulo, Pearson Addison Wesley, 2011. Citado 3 vezes nas páginas 17, 18 e 19.

ELMASRI, R.; NAVATHE, S. B. *Sistema de Banco de Dados. Revisor técnico Luíz Ricardo de Figueiredo*. [S.l.]: São Paulo: Pearson Addison Wesley, 2005. Citado na página 22.

ERICK. *Treinando modelos do parser de Stanford*. 2017. http://linguaecomputador.blogspot.com.br/2016/03/treinando-modelos-do-parser-destanford.html. Acesso em: 23.07.2017. Citado na página 36.

FERNANDES, F. da S. Banco de dados vs mapreduce em algoritmos para grafos. 2015. Citado 2 vezes nas páginas 30 e 31.

GALLI, G. *Preposições conceitos.* 2017. http://www.lpeu.com.br/q/ruse. Acesso em: 03.09.2017. Citado na página 49.

GROUP, T. S. N. L. P. *The Stanford Parser: A statistical parser*. 2017. https://nlp.stanford.edu/software/lex-parser.shtml. Acesso em: 23.07.2017. Citado na página 36.

IMPACTA, B. *Entenda a importância de um banco de dados em uma organização*. 2017. http://www.impacta.com.br/blog/2017/01/30/entenda-a-importancia-de-um-banco-de-dados-em-uma-organizacao/. Acesso em: 23.08.2017. Citado na página 13.

LI, F.; JAGADISH, H. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 8, n. 1, p. 73–84, 2014. Citado 11 vezes nas páginas 15, 31, 32, 38, 40, 41, 46, 49, 51, 52 e 61.

LUCCA, J. D.; NUNES, M. d. G. V. Lematização versus stemming. *USP, UFSCar, UNESP, São Carlos, São Paulo*, 2002. Citado na página 38.

MARINS, W. F. et al. Ontologias de domínio na interpretação de consultas a bancos de dados relacionais. Universidade Federal de Goiás, 2015. Citado na página 33.

MARTHA, A. S.; BARRA, P. S. C.; CAMPOS, C. J. R. de. Recuperação de informações em textos livres de prontuários do paciente. In: *CBIS-IX Congresso Brasileiro de Informática em Saúde*. [S.l.: s.n.], 2004. Citado na página 43.

MEIRA, M. R. *Banco de Dados*. [S.l.]: IFBA–Campus Ilhéus, 2013. http://regilan.com.br/wp-content/uploads/2013/10/Apostila-Banco-de-Dados.pdf. Citado na página 17.

NANTES, L. M. Desenvolvimento de um sistema baseado em linguagem natural para consultas em banco de dados na web. 2008. Citado na página 15.

OLIVEIRA, V. M.; ROMAN, N. T. Jwn-br–uma api java para a wordnet. br. In: *9th Brazilian Symposium in Information and Human Language Technology*. [S.l.: s.n.], 2013. p. 153–157. Citado 2 vezes nas páginas 38 e 41.

PINTO, B. P.; PINHEIRO, D.; PEREIRA, D. A. Recuperação de dados em banco de dados por meio da linguagem natural. *e-xacta*, v. 3, n. 2, 2011. Citado 2 vezes nas páginas 25 e 32.

PUGA, S.; FRANÇA, E.; GOYA, M. *Banco de Dados: Implementação em SQL, PL\SQL e Oracle 11g*. [S.l.]: São Paulo: Pearson Education-Br, 2013. Citado 3 vezes nas páginas 20, 21 e 22.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de banco de dados-3*. [S.l.]: AMGH Editora, 2008. Citado 5 vezes nas páginas 13, 14, 18, 19 e 20.

SILVA, B. C. D. da et al. Introdução ao processamento das línguas naturais e algumas aplicações. *Série de Relatórios do Núcleo Interinstitucional de Lingüística Computacional*, v. 3, 2007. Citado 2 vezes nas páginas 15 e 25.

SILVA, R. R.; LIMA, S. M. B. *Consultas em Bancos de Dados Utilizando Linguagem Natural*. 2013. Citado 2 vezes nas páginas 14 e 15.

SOUZA, S. de; CAMPOS, E. de; SANTOS, A. D. Uma ferramenta para a definição de consultas baseada em entidades e papéis. *IEEE Latin America Transactions*, v. 4, 2006. Citado 2 vezes nas páginas 13 e 14.

TOUTANOVA, K. et al. Feature-rich part-of-speech tagging with a cyclic dependency network. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. [S.l.], 2003. p. 173–180. Citado na página 36.