UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Marcos Andrei Dranka

SIMDBS - SISTEMA AUXILIAR PARA COMPARAÇÃO ESTRUTURAL DE DIFERENTES BANCOS DE DADOS NO PROCESSO DE INTEGRAÇÃO DE DADOS.

Florianópolis, Santa Catarina – Brasil
2023

Marcos Andrei Dranka

SIMDBS - SISTEMA AUXILIAR PARA COMPARAÇÃO ESTRUTURAL DE DIFERENTES BANCOS DE DADOS NO PROCESSO DE INTEGRAÇÃO DE DADOS.

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Profª. Carina Friedrich Dorneles, Drª.

Florianópolis, Santa Catarina – Brasil
2023

Notas legais:

Não há garantia para qualquer parte do software documentado. Os autores tomaram cuidado na preparação desta tese, mas não fazem nenhuma garantia expressa ou implícita de qualquer tipo e não assumem qualquer responsabilidade por erros ou omissões. Não se assume qualquer responsabilidade por danos incidentais ou consequentes em conexão ou decorrentes do uso das informações ou programas aqui contidos.

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.
Arquivo compilado às 22:40h do dia 11 de julho de 2023.

Marcos Andrei Dranka
SimDBS - Sistema auxiliar para comparação estrutural de diferentes bancos de dados no processo de integração de dados. / Marcos Andrei Dranka; Orientador(a), Profª. Carina Friedrich Dorneles, Drª. – Florianópolis, Santa Catarina – Brasil, 04 de julho de 2023.
83 p.

Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico, Programa de Graduação em Ciências da Computação.

Inclui referências

1. Banco de dados, 2. Integração de dados, 3. Usabilidade, 4. Similaridade de dados, I. Profª. Carina Friedrich Dorneles, Drª. II. Programa de Graduação em Ciências da Computação III. SimDBS - Sistema auxiliar para comparação estrutural de diferentes bancos de dados no processo de integração de dados.

Marcos Andrei Dranka

SIMDBS - SISTEMA AUXILIAR PARA COMPARAÇÃO ESTRUTURAL DE DIFERENTES BANCOS DE DADOS NO PROCESSO DE INTEGRAÇÃO DE DADOS.

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, Santa Catarina – Brasil, 04 de julho de 2023.

Prof. Renato Cislaghi, Dr.
Coordenador(a) de Projetos

Banca Examinadora:

Profª. Carina Friedrich Dorneles, Drª.
Orientador(a)
Universidade Federal de Santa Catarina – UFSC

Profº. Renato Fileto, Dr.
Universidade Federal de Santa Catarina – UFSC

Profº. Ronaldo dos Santos Mello, Dr.
Universidade Federal de Santa Catarina – UFSC

AGRADECIMENTOS

Agradeço a Deus por toda a generosidade que sempre teve comigo, mesmo com meus inúmeros defeitos e falhas, e à Nossa Senhora, que sempre intercede por mim. Agradeço à minha esposa, Luana, que sempre me incentivou muito mesmo quando eu achava que não ia dar certo (e não é que ela estava certa?). Agradeço por escolher passar o resto da vida ao meu lado e assumir todas as responsabilidades que vieram junto. Agradeço aos meus pais, Orti e Cirlene, que em todos os momentos sempre me apoiaram e incentivaram em todos os desafios por mais de 20 anos. Agora vocês podem descansar, pois minha esposa está cumprindo bem esse papel. Agradeço também à minha filha, Beatriz, por esperar o papai terminar de escrever o TCC para nascer. Foi uma das épocas mais cansativas, mas vejo que vale a pena a cada vez que estou contigo nos braços. Agradeço também à professora Carina, por orientar o trabalho ao longo de um ano, sempre atenciosa e indicando as melhores soluções. Aos professores Ronaldo e Fileto, que participaram da banca e à todos os professores do INE, que participaram de minha formação nos últimos anos. Por fim, mas não menos importante, agradeço à todos os amigos que conviveram comigo e sempre me apoiaram e ajudaram nas mais variadas circunstâncias. Ao Robson e ao Gabriel, os melhores colegas de trabalho da UFSC. Ao José e à Camila, por todos os bons conselhos e amizade. Aos amigos do curso, principalmente o Alan, que sempre está à disposição para ajudar. E aos amigos que moram longe, mas que fizeram e ainda fazem parte da minha vida, em especial o Paulo, o Jakson, o Wilson e o Juliano (*in memorian*).

RESUMO

O avanço tecnológico que vem ocorrendo nos últimos anos fez com que recursos computacionais mais robustos fossem adotados cada vez mais para auxiliar em diversos tipos de tarefas cotidianas. Neste contexto, as empresas constantemente executam melhorias, substituindo equipamentos e sistemas informatizados já obsoletos por outros mais recentes e eficazes para oferecer um atendimento mais rápido e eficiente aos clientes. Porém, os dados gerados no sistema obsoleto não podem ser descartados, mas precisam de um tratamento adequado para serem integrados ao novo software: essa tarefa toma um tempo considerável na análise das tabelas, pois a modelagem de dados desenvolvida para o sistema antigo pode ser muito diferente da modelagem atual. As diferenças podem ocorrer porque novas informações são necessárias, as regras de negócio mudaram, novas leis exigem alterações, etc. Desta forma, o objetivo deste trabalho é desenvolver um sistema que obtenha as informações sobre as tabelas de duas bases de dados e as mostre em uma única tela para agilizar a comparação. Ele recebe as informações de acesso aos bancos de dados e retorna uma tabela mostrando as colunas com maior similaridade entre as duas tabelas comparadas, além de informações sobre cada coluna e o percentual de similaridade entre as colunas das duas tabelas. Para isto, foram utilizados JavaScript, HTML, CSS e PostgreSQL, além de algumas bibliotecas disponíveis através do NPM.

Palavras-chaves: Banco de dados. Integração de dados. Usabilidade. Similaridade de dados.

ABSTRACT

The technological advancement that has been taking place in the last years has made it possible more robust computational resources to be adopted in order to assist in various types of everyday tasks. In this context, companies are constantly making improvements, replacing obsolete equipment and computerized systems with more recent and effective ones to offer faster and more efficient service to their customers. However, the data generated in the obsolete system cannot be discarded, but it needs an appropriate treatment to be integrated into the new software: this task takes substantial time in the tables analysis, since the data modeling developed for the old system can be very different from the current model. These differences might occur because new information is needed, business rules have changed, new laws require changes, and so on. Thus, the objective of this work is to develop a system that obtains information about the tables of two databases and shows them on a single screen to quicken the comparison. It collects the information to access the databases and delivers a table that shows the columns with the greatest similarity between the two compared tables, in addition to information about each column and the percentage of similarity between the columns of those two tables. For this purpose, JavaScript, HTML, CSS and PostgreSQL were used, along with some libraries available on NPM.

Keywords: Database. Integration. Usability. Data Similarity.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Program Interface
BD	Banco de dados
CSS	Cascading Style Sheets
DBA	DataBase Administrator
HTML	HyperText Markup Language
HTTP	HyperText Tranfer Protocol
JSON	JavaScript Object Notation
NPM	Node Package Manager
SGBD	Sistema Gerenciador de Banco de dados
SimDBS	Similarity DataBase Structure analyzer
SUS	System Usability Scale

SUMÁRIO

1 INTRODUÇÃO

O grande volume de dados produzido hoje em dia nos mais variados setores do mercado junto com as rápidas mudanças que ocorrem no mercado de tecnologia, trazem grandes desafios na maneira como os dados são armazenados e gerenciados. Em diversos casos, uma base de dados pode trazer informações insuficientes para uma análise estratégica, sendo necessário buscar informações complementares em outras bases. Outro desafio, é quando um sistema legado é substituído por um mais recente e completo. O novo *software* precisa importar os dados do anterior, que são de extrema importância, pois trazem toda a informação que a empresa obteve ao longo dos anos. Para esses casos, a análise da estrutura onde os dados estão armazenados é imprescindível para entender a melhor forma de trabalhar com esses dados.

Cada base de dados pode utilizar diferentes estruturas para armazenar dados semelhantes, pois, em um mesmo domínio, diferentes administradores de bancos de dados (DBAs) podem definir estruturas bem diferentes para construir sua base de dados, mesmo trabalhando com informações semelhantes. Para citar um exemplo, os sistemas utilizados por secretarias da saúde de diversos municípios podem ser bem diferentes. Embora os dados busquem informar a mesma coisa, a forma de armazenamento pode diferir muito, não havendo um padrão. Quando os dados precisam ser enviados ao Ministério da Saúde, há a necessidade de padronizar os dados, para que não seja gerada uma confusão na chegada de dados nos sistemas do Ministério. A análise da estrutura dos bancos de dados auxilia muito no trabalho de quem precisa manipular esses dados para padronizá-los.

A análise desses dados é uma tarefa exaustiva e que requer muita atenção. A tarefa de obter os metadados manualmente, bem como criar planilhas ou modelos para analisar custam um tempo valioso do DBA, tornando a análise dos dados mais lenta e trabalhosa. Há vários metadados disponíveis, sendo alguns importantíssimos e outros irrelevantes para uma análise de similaridade (MONTEIRO; LIFSCHITZ; BRAYNER, 2007). Uma ferramenta que permita obter e visualizar os dados rapidamente, poupa tempo e esforço do DBA, que poderá dar mais ênfase na integração de dados, obtendo melhores resultados.

Há algumas aplicações que buscam facilitar a visualização da estrutura de dados, porém de forma manual ou com poucas informações. O MySQL oferece uma forma de visualizar os metadados, mas com poucas informações e de forma manual (DIGITAL, 2023). O Multiple DBData View (ROSA, 2022) fornece informações mais completas, porém a navegação é um pouco trabalhosa, sendo necessário alternar entre as abas, dificultando uma análise rápida.

Portanto, este trabalho busca resolver alguns dos problemas citados anteriormente, utilizando o SGBD PostgreSQL, por ser gratuito e com um número considerável de

usuários. o SimDBS tem o objetivo de acessar dois bancos de dados simultaneamente e compará-los, trazendo o resultado da comparação lado a lado, de forma intuitiva e de fácil leitura.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver um *software* que auxilie no processo de integração de dados, fornecendo informações sobre a similaridade entre a estrutura dos bancos de dados informados. Para isso, o SimDBS irá receber as informações de acesso aos BDs, fazer a análise comparativa entre as estruturas dos mesmos e mostrar uma tabela comparativa com as informações lado a lado, de forma clara e de simples leitura.

1.1.2 Objetivos específicos

Objetivo 1: Elaborar o projeto de interface, comparar e escolher o algoritmo de similaridade a ser utilizado, definir a linguagem de desenvolvimento e as ferramentas que serão utilizadas.

Objetivo 2: Implementar o código responsável pela comparação, utilizando o algoritmo de similaridade mais adequado e automatizando o processo de comparação, através de agrupamento de colunas mais semelhantes.

Objetivo 3: Desenvolver uma interface fácil de usar, com informações visuais bem organizadas e detalhadas, permitindo ao usuário visualizar os dados rapidamente.

Objetivo 4: Avaliar o resultado produzido pelo *software*, com o uso de bancos de dados de testes e verificar se a saída gerada produz resultados satisfatórios.

Objetivo 5: Avaliar a usabilidade do *software*, através de testes com alguns usuários.

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho possui mais três capítulos, sendo o Capítulo 2 dedicado à análise de alguns trabalhos relacionados, utilizados como fonte de informações para este trabalho. O Capítulo 3 descreve a fundamentação Teórica, e explicita o funcionamento de alguns algoritmos considerados para calcular a similaridade de Strings. Já no Capítulo 4, demonstra-se os testes com algumas strings usando os algoritmos, buscando escolher o que produz o melhor resultado para o objetivo do trabalho. Além disso, são definidos os critérios de semelhança, quais metadados serão utilizados na comparação e o peso de cada metadado na porcentagem de similaridade. O Capítulo 5

mostra o desenvolvimento do código, a linguagem e os recursos utilizados, bem como as classes e funções desenvolvidas, o projeto da interface e o desenvolvimento do protótipo que será utilizado nos testes. Nos capítulos 6 e 7, são realizados os testes de qualidade e de usabilidade, respectivamente. Nos testes de qualidade, busca-se avaliar se os resultados produzidos pelo SimDBS são satisfatórios, utilizando para isso alguns bancos de dados projetados para testes e verificando se os resultados produzidos são condizentes com o objetivo do trabalho. No teste de usabilidade, a finalidade é verificar se a interface do programa cumpre os requisitos de facilidade de uso e interface amigável ao usuário. Por fim, o Capítulo 8 apresenta as considerações finais e sugestões de trabalhos a serem elaborados posteriormente, tendo por base os resultados alcançados ao final deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ALGORITMOS DE AVALIAÇÃO DE SIMILARIDADE

O trabalho aborda conceitos como banco de dados PostgreSQL, similaridade de dados e estruturas de bancos de dados. O principal conceito tratado neste trabalho é a similaridade entre os dados.

2.1.1 Similaridade de dados

Sendo o principal conceito do trabalho, a similaridade de dados busca comparar dados e retornar um valor que representa a semelhança entre os dados (geralmente um valor entre 0 e 1, a depender do algoritmo). Alguns algoritmos foram considerados, buscando um que se adequasse ao tipo de comparação necessário ao trabalho.
Com o objetivo de mostrar os metadados de forma organizada para facilitar o entendimento das estruturas dos bancos, o sistema utiliza um algoritmo de comparação entre as colunas de duas tabelas previamente selecionadas de dois bancos de dados diferentes, mas de mesmo domínio de aplicação. A análise de comparação através de algoritmos envolve muitos fatores, que podem gerar resultados inconsistentes dependendo do tipo de metadado fornecido. Deve-se levar em conta que a comparação entre frases e entre palavras individuais podem fornecer resultados bem diferentes. Uma frase pode ser reescrita usando outras palavras, causando uma análise errônea por alguns algoritmos. Para o SimDBS, foi considerado o contexto de sua aplicação, permitindo uma escolha de algoritmo mais abrangente.
Por se tratar de comparação entre tabelas de um banco de dados, o uso de algoritmos de avaliação de similaridade de strings se mostrou muito útil, pois, em geral, o nome das colunas de uma tabela são palavras curtas, que especificam bem que tipo de informação deve ser inserido naquela coluna. Qualquer DBA experiente utilizará palavras que busquem eliminar qualquer ambigüidade, para alcançar maior consistência das informações armazenadas. Os termos usados tendem a ser muito semelhantes para a mesma informação, sendo um algoritmo de similaridade de string uma escolha óbvia para essa comparação.
Há diversos algoritmos de similaridade de strings, cada um com suas particularidades. Para verificar qual fornece um resultado apropriado para o objetivo do trabalho, foi feito um teste com 4 algoritmos de similaridade. Cada um deles utiliza uma técnica diferente para gerar um taxa de similaridade. Os algoritmos considerados para esse teste são:

Dice's Coefficient: No campo da recuperação da informação, o coeficiente pode ser visto como o dobro da informação compartilhada, referente à soma das cardinalidades. O coeficiente também pode ser usado como uma medida de similaridade entre strings. Dadas duas strings x e y, podemos calcular o coeficiente da seguin-

te forma:

$s = \frac{2n_t}{n_x+n_y}$

onde n_t é o número de dígrafos (compostos de dois caracteres consecutivos) comuns às duas strings, n x é o número de dígrafos em x e n y o número de dígrafos em y . Por exemplo, para calcular a semelhança entre:

`nighte - nacht`,

calculamos os dígrafos de cada palavra:

`ni, ig, gh, ht`
`na, ac, ch, ht`

Cada conjunto possui quatro elementos, e sua interseção se reduz a um elemento ht. Com a fórmula fornecida acima, obtemos

$s = \frac{2\times1}{4+4} = 0,25.$

Jaro-Winkler: A métrica de distância Jaro-Winkler é a medida da similaridade entre duas strings. A métrica Jaro distance estabelece que dadas duas strings s_1, s_2, sua distância d_j é:

$d_j = \frac{m}{3a} + \frac{m}{3b} + \frac{m-t}{3m}$

onde:

m é o número de correlações entre caracteres; a e b são os tamanhos de s_1 e s_2, respectivamente; t é o número de transposições.

Levenshtein: A distância Levenshtein ou distância de edição entre duas "strings"(duas sequências de caracteres) é dada pelo número mínimo de operações necessárias para transformar uma string na outra. Entendemos por "operações"a inserção, deleção ou substituição de um caractere.

String Cosine Similarity: Separa a string em palavras ou sílabas e, para cada conjunto, verifica quantas ocorrências há em cada string. São gerados dois vetores (contendo 0's e 1's), um para cada String, e esses valores são aplicados no cálculo do cosseno, retornando a taxa de similaridade. Tende a gerar valores maiores, pois, diferente do Dice's Coefficient, leva em conta todas as combinações em qualquer posição, aumentando a similaridade. Para duas frases que usem as mesmas palavras, por exemplo, a similaridade será alta, mesmo que as frases tenham significados completamente diferentes.

2.2 METADADOS EM SGBDS

A comparação será entre informações obtidas dos bancos de dados. Essas informações ficam armazenadas no SGBD e são chamadas metadados. Ao criar um banco

de dados em qualquer SGBD, é criado junto uma ou mais tabelas com as informações sobre aquele banco, trazendo informações sobre as tabelas criadas, os tipos de dados armazenados, entre outras informações importantes. Os metadados usados neste trabalho são do catálogo do PostgreSQL (MONTEIRO; LIFSCHITZ; BRAYNER, 2007), das tabelas `pg_catalog` e `information_schema`:

`table_name`: Lista as tabelas disponíveis no banco de dados.

`column_name`: Mostra as colunas de determinada tabela.

`constraint_name`: Nome da relação entre duas tabelas.

`data_type`: Tipo de dados que a coluna armazena.

`character_maximum_length`: Tamanho da String.

`is_nullable`: Indica se o campo pode estar vazio ou não.

`is_updatable`: Indica se a informação pode ser alterada ou não.

As consultas realizadas encontram-se no código-fonte, como apêndice ao final deste documento.

2.3 INTEGRAÇÃO DE DADOS

A integração de dados é uma área que busca unir informações que estão dispersas ou armazenadas em diferentes formas. Um dos exemplos mais comuns é a necessidade de substituir um software antigo por um mais completo, mas preservar os dados que já foram produzidos. O grande volume de dados do software antigo precisa ser adicionado ao novo software, mas de forma organizada, permitindo que os dados possam ser usados no novo sistema. Para isso, é necessário buscar uma forma de integrar esses dados, fazendo um tratamento para permitir que sejam armazenados em uma mesma base de dados. A integração de dados tem várias utilidades, principalmente nos dias atuais, com a enorme quantidade de dados produzidos diariamente. Dados redundantes tendem a ser cada vez mais comuns, e a integração de dados pode auxiliar a unir essas informações, eliminando o que for duplicado. Ao realizar uma pesquisa, em qualquer área do conhecimento, pode ser necessário reunir informações de diferentes bancos de dados, e a integração entre os dados pode tornar o trabalho muito mais simples, com informações padronizadas, facilitando o uso de softwares para estatística, entre tantos outros.

2.4 USABILIDADE DA INTERFACE

O conceito de usabilidade tem ganhado um bom espaço nos últimos anos. Quando os computadores pessoais surgiram, as empresas da área começaram a pensar

em maneiras de facilitar o uso dos computadores, criando interfaces mais bonitas e fáceis de utilizar. O conceito de usabilidade busca identificar maneiras de projetar uma interface intuitiva, permitindo que o usuário mais leigo tenha uma curva de aprendizado bem pequena ao utilizar o sistema. Vários fatores podem tornar o sistema mais fácil de usar, como a posição que cada botão fica na tela, ou qual menu esconde determinada função. Atualmente, com o grande número de sites e aplicativos existentes, alguns padrões se consolidaram, e não é mais uma opção escolher entre várias formas. O ícone de menu em praticamente qualquer site ou aplicativo é o ícone 'sanduíche' (três barras horizontais). A posição do botão 'fechar janela' na maioria dos sistemas operacionais é no canto superior direito, ainda que alguns sistemas utilizem o lado oposto. Nesses casos, a melhor solução é utilizar o padrão já consolidado, pois o usuário já espera encontrar essa opção na mesma posição que em outros sistemas. O objetivo deste trabalho é ser útil e fácil de utilizar, logo, é importante que a interface tenha uma boa usabilidade, sendo intuitiva e compreensível para usuários iniciantes.

3 TRABALHOS RELACIONADOS

Aqui são apresentados trabalhos que serviram de base para a elaboração do presente trabalho. Durante o desenvolvimento, vários conceitos destes trabalhos foram utilizados, e buscou-se melhorar soluções já existentes, permitindo obter resultados ainda melhores na análise dos dados.

3.1 STRING METRICS AND WORD SIMILARITY APPLIED TO INFORMATION RETRIEVAL

Este trabalho analisa diferentes algoritmos de similaridade de dados, demonstrando com experimentos a eficiência e os resultados dos algoritmos abordados (CHEN, 2012). O trabalho mostra um panorama geral sobre a similaridade de informações, descreve o funcionamento dos algoritmos e os testa em diferentes cenários. Entre os algoritmos abordados, há a distância de Levenshtein, a distância de Hamming, a similaridade por cosseno e a similaridade de Dice. Entre os algoritmos, alguns apresentam resultados melhores em comparações entre textos, outros em comparações com outros tipos de dados, como imagens e outros tipos de objetos. Os resultados apresentados forneceram auxílio para traçar o desenvolvimento do presente trabalho, fornecendo informações sobre os algoritmos e norteando os testes realizados posteriormente para determinar o algoritmo que melhor se enquadraria no objetivo do SimDBS.

3.2 EXTRAINDO METADADOS DE SGBDS

Os bancos de dados armazenam várias informações sobre as tabelas que os compõe. Essas informações, chamadas metadados, permitem analisar a estrutura das tabelas, entendendo como os dados são tratados e armazenados (MONTEIRO; LIFSCHITZ; BRAYNER, 2007). Nesse trabalho, o objetivo é analisar os metadados dos SGBDs mais utilizados, buscando determinar quais são os metadados mais úteis para a análise de variadas métricas. Conforme a visão do autor, os metadados são importantes pois:

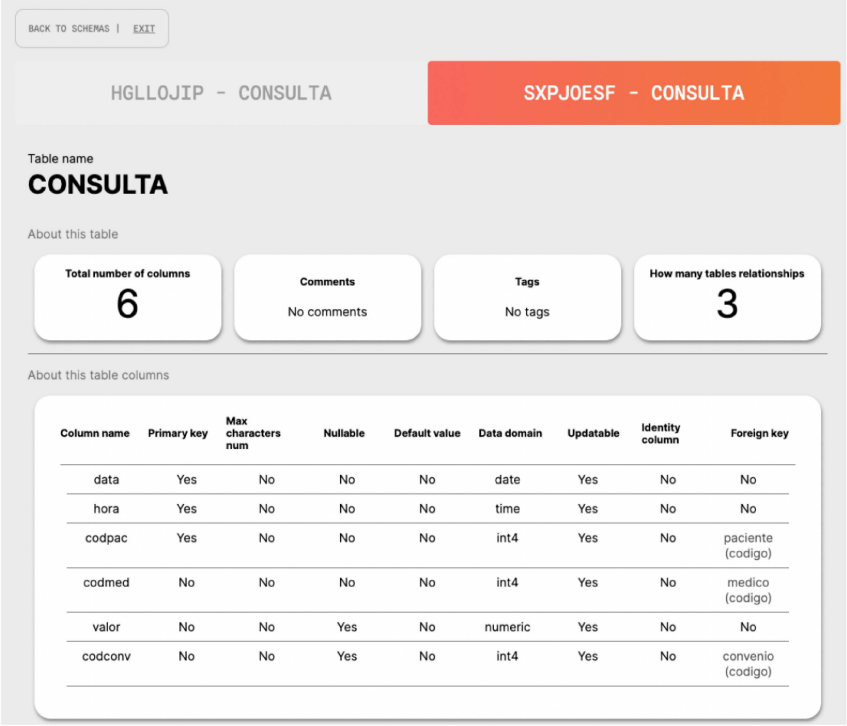
- Fornecem contexto aos dados armazenados
- Descobrir os tipos de dados, os relacionamentos entre eles e quando foram criados ou alterados.
- Possibilitam checagem de tipo, validação dos dados, formatação, entre outras.
- Possibilitar análises das tabelas e do desempenho do banco de dados.

Serão utilizados alguns dos metadados citados pelo trabalho, buscando fornecer contexto sobre as tabelas, para que a análise de similaridade alcance melhores resultados.

3.3 INTERFACE PARA VISUALIZAÇÃO DE DADOS BRUTOS VINDOS DE DIFERENTES FONTES

Deste trabalho veio a idéia inicial para o desenvolvimento do SimDBS. Nele, a autora desenvolve um sistema com o objetivo de facilitar a leitura dos metadados, tornando mais simples a comparação entre bancos de dados diferentes (ROSA, 2022). O sistema compara duas bases de dados diferentes, fornecendo uma lista de tabelas de cada banco, que podem ser selecionadas e comparadas. Um dos objetivos da autora, foi fornecer uma interface bem elaborada e fácil de usar, permitindo que qualquer pessoa que queira comparar bancos de dados heterogêneos tenham facilidade no processo. Dessa forma, o usuário apenas fornece as informações de acesso aos bancos de dados, e o sistema faz a leitura de metadados importantes, exibindo-os aos usuários através de cards e objetos 3D. O usuário pode interagir com a interface, selecionando as tabelas e mostrando os metadados e algumas estatísticas sobre essa tabela. O foco do trabalho foi a visualização das informações, para que o usuário possa poupar tempo na tarefa de obter os metadados organizados, porém, sem processar tais dados, deixando essa tarefa para o usuário executar manualmente. A interface do DBDataView é mostrada na Figura 1.

Figura 1 – Interface do DBDataView



Fonte: Autora

3.4 TABELA COMPARATIVA

Na Tabela 1 é feita uma comparação entre os diferentes trabalhos relacionados, com a contribuição dos mesmos para o desenvolvimento do SimDBS

Tabela 1 – Tabela comparativa entre os trabalhos

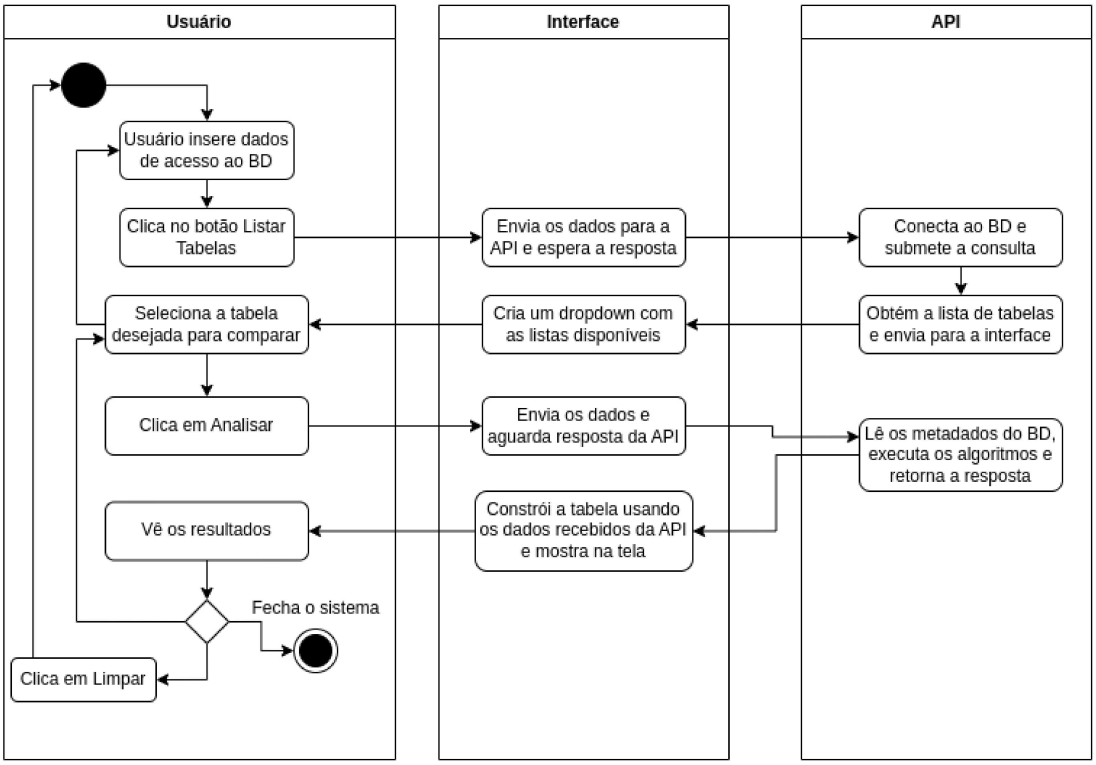
Trabalho	**Objetivo**	**Contribuição para este trabalho**	**Diferenciais com o SimDBS**
(CHEN, 2012)	Testar diferentes algoritmos de similaridade, e representar graficamente o desempenho de cada algoritmo.	Fornecer algoritmos e análise prévia do desempenho dos mesmos, auxiliando na escolha do algoritmo mais adequado.	Apenas testa alguns algoritmos de similaridade em diferentes cenários, fornecendo informações importantes para quem precisa escolher um dos algoritmos.
(MONTEIRO; LIFSCHITZ; BRAYNER, 2007)	Verificar os metadados mais importantes no entendimento da estrutura de um banco de dados, buscando determinar quais os metadados importantes para calcular certas métricas.	Fornecer uma lista de metadados do PostgreSQL importantes para o cálculo de similaridade.	Faz um mapa dos metadados disponibilizados pelo PostgreSQL, mas não utiliza os metadados para comparar informações.
(ROSA, 2022)	Desenvolver um software que obtenha os metadados do BD e os mostre de forma organizada, auxiliando na comparação entre diferentes tabelas de bancos de dados.	Trabalho usado como inspiração, no qual o objetivo é auxiliar na comparação de bancos de dados. O SimDBS, entretanto, busca utilizar os metadados para efetuar cálculos de similaridade e mostrar ao usuário os dados já tratados, automatizando a comparação que antes ainda era manual e trabalhosa.	Tem muitas semelhanças, porém mostra o resultado em duas abas diferentes, sendo necessário alternar entre elas, e não calcula a similaridade como o SimDBS, apenas mostra as informações para o usuário.
SimDBS	Desenvolver um sistema que auxilie na comparação da estrutura entre dois bancos de dados, automatizando o processo de comparação entre tabelas de forma que o usuário possa visualizar a comparação em uma única tela organizada, com informações visuais que tornem a compreensão dos dados mais rápida	-	-

4 SIMDBS

O SimDBS (Sistema auxiliar para comparação de estruturas de bancos de dados por similaridade para integração de dados) tem como objetivo simplificar a análise para integração de dados. Um dos objetivos da construção do sistema foi a usabilidade, permitindo que qualquer usuário com pouco conhecimento possa utilizar o sistema.

A Figura 2 mostra o fluxo seguido pelo usuário. Primeiro, o usuário insere as informações de acesso aos bancos de dados (host, usuário, senha e nome do banco). Após, clica no botão `Listar tabelas`, fazendo com que a sistema requisite à API as tabelas disponíveis. A API irá acessar o banco de dados com as informações fornecidas e retornará uma lista de tabelas disponíveis naquele banco de dados. As tabelas serão mostradas em uma lista no formato `dropdown`, para que o usuário possa escolher a tabela que deseja comparar. O processo se repete com o segundo banco de dados que será uttilizado na comparação. Após escolher as tabelas que serão utilizadas, basta clicar no botão `Analisar`, e o sistema irá requisitar à API o resultado da comparação. A API, novamente, acessa os bancos de dados e busca os metadados das tabelas informadas. O cálculo de similaridade entre as propriedades é efetuado e o resultado é retornado para o *front-end*. O SimDBS constrói a tabela utilizando as informações que recebeu da API. O usuário pode, então, escolher outras tabelas para comparar, ou mesmo outros bancos de dados, e prosseguir com as análises.

Figura 2 – Fluxo de uso



Fonte: Própria

4.1 ESCOLHA DO ALGORITMO

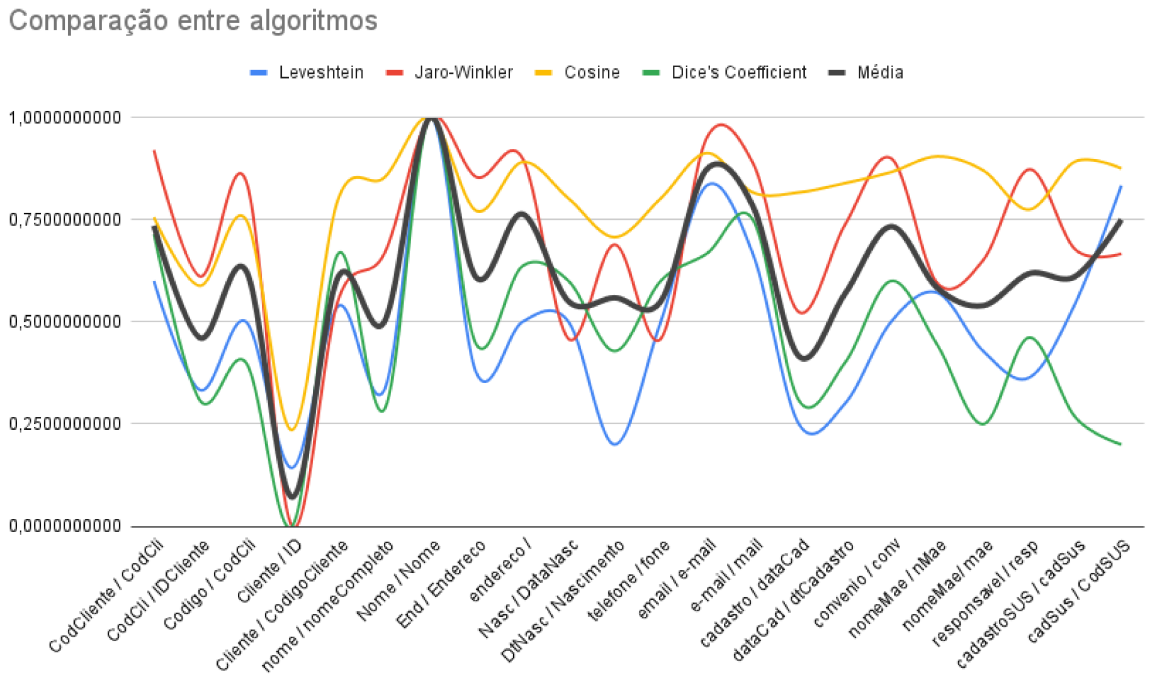
Os algoritmos foram testados utilizando nomes de colunas possíveis em uma tabela de clientes. Os resultados das comparações são mostrados na Figura 3, e o gráfico comparativo na Figura 4.

Figura 3 – Tabela comparativa dos algoritmos

	Levenshtein	Jaro-Winkler	Cosine	Dice's Coefficient
CodCliente / CodCli	0.6	0.92	0.7559289460184544	0.7142857142857143
CodCli / IDCliente	0.3333333333333333	0.611111111111111	0.588348405414552	0.3076923076923077
Codigo / CodCli	0.5	0.8444444444444443	0.7499999999999999	0.4
Cliente / ID	0.14285714285714285	0	0.2357022603955158	0
Cliente / CodigoCliente	0.5384615384615384	0.553113553113553	0.8001322641986387	0.6666666666666666
nome / nomeCompleto	0.3333333333333333	0.6666666666666666	0.8528028654224417	0.2857142857142857
Nome / Nome	1	1	1	1
End / Endereco	0.375	0.8541666666666666	0.7715167498104596	0.4444444444444444
endereco / enderecoCompleto	0.5	0.9	0.8908708063747479	0.6363636363636364
Nasc / DataNasc	0.5	0.4583333333333333	0.8017837257372732	0.6
DtNasc / Nascimento	0.2	0.6888888888888888	0.7071067811865476	0.42857142857142855
telefone / fone	0.5	0.4583333333333333	0.8017837257372732	0.6
email / e-mail	0.8333333333333334	0.9500000000000001	0.9128709291752769	0.6666666666666666
e-mail / mail	0.6666666666666666	0.8888888888888888	0.8164965809277261	0.75
cadastro / dataCad	0.25	0.5238095238095238	0.8164965809277259	0.3076923076923077
dataCad / dtCadastro	0.3	0.7371428571428572	0.8391463916782737	0.4
convenio / conv	0.5	0.9	0.8660254037844387	0.6
nomeMae / nMae	0.5714285714285714	0.5952380952380952	0.9045340337332909	0.4444444444444444
nomeMae/ mae	0.42857142857142855	0.6507936507936508	0.8703882797784892	0.25
responsavel / resp	0.36363636363636365	0.8727272727272728	0.7745966692414834	0.46153846153846156
cadastroSUS / cadSus	0.5454545454545454	0.6767676767676768	0.8922178162191937	0.26666666666666666
cadSus / CodSUS	0.8333333333333334	0.6666666666666666	0.8749999999999998	0.2

Fonte: Testes realizados pelo autor

Figura 4 – Gráfico comparativo



Fonte: O autor

De acordo com os testes feitos,podemos observar que alguns algoritmos, como o de Jaro-Winkler e Cosine geram um valor mais alto do que o esperado para alguns casos. Já os algoritmos de Levenshtein e Dice's Coefficient produzem valores melhores para os fins de comparação dos nomes. Ambos geram valores próximos, e os dois tiveram bons resultados em casos específicos. No gráfico, vemos que a variação destes dois algoritmos se mantém mais próxima da média, enquanto que aqueles algoritmos geram alguns picos acima ou abaixo da média. Tanto o algoritmo de Levenshtein quanto o Dice's Coefficient poderiam ser usados, com resultados semelhantes. Diante desse fato, a escolha foi pelo **Dice's Coefficient**, pelo fato de que em alguns casos específicos, os resultados produzidos foram mais satisfatórios.

4.2 SIMILARIDADE ENTRE TABELAS E ATRIBUTOS

Uma das características principais do SimDBS é a comparação entre as tabelas por similaridade. Para isso, foi necessário buscar um algoritmo que obtivesse valores condizentes, e que se comportasse bem em comparações das diversas colunas, sem apresentar anomalias em casos isolados.

As colunas das tabelas apresentam várias informações para comparação, sendo que as informações mais significativas são o nome da coluna e o tipo de dados. Com essas duas informações, podemos determinar se os dados armazenados naquela coluna tem alguma semelhança, ou essa semelhança pode ser descartada. Levando

em conta a análise de algumas tabelas, observa-se que há colunas que se parecem muito, independente da tabela, por se tratar de informações padronizadas, como datas, número de documentos, nomes de estados, etc. Há também, campos bem diversos, que dependem exclusivamente do domínio da aplicação, e que podem variar bastante. Em geral, um DBA experiente utilizará nomes que evitem ambigüidade, o que produz uma certa padronização nos nomes dos atributos.

Com base nisso, determinou-se que o nome da coluna e o tipo de dados irão representar 85% da similaridade entre os campos, e os demais 25% serão definidos por informações que possuem menor relevância. A definição das porcentagens foi feita avaliando quais metadados têm maior relevância ao informar qual a informação está salvos naquela coluna do banco. A forma que um banco de dados vai armazenar uma determinada informação pode variar de acordo com o domínio da aplicação, ou mesmo critérios subjetivos do DBA que projetou o banco de dados. Independente desses critérios, o principal metadado que irá identificar a informação armazenada é o nome da coluna, por carregar todo o significado semântico da informação. Os metadados utilizados nas consultas estão descritos no capítulo Fundamentação teórica, seção 2.2. Os pesos dos metadados para o cálculo de similaridade foram definidos de forma intuitiva, em um trabalho futuro pode-se automatizar utilizando algum algoritmo de machine learning. Foram definidos assim:

- Nome do Atributo: 60%
- Tipo de dados (e tamanho, para strings): 25%
- Primary Key - 5%
- Nullable - 2%
- Updatable - 2%
- Foreign Key - 4%
- Restrição - 2%

4.2.1 Nome do Atributo

Para o nome do atributo, a principal métrica de similaridade é a semelhança entre o nome do campo. Isso determina 60% da similaridade do campo. O nome é a principal identificação sobre a informação que está sendo armazenada naquela coluna. Para calcular a similaridade, foi utilizado o algoritmo Dice's Coefficient, selecionando uma das colunas da tabela 1 e comparando com todas as colunas da tabela 2, buscando encontrar a mais similar. Então, a coluna da tabela 2 com maior similaridade no nome é associada à coluna previamente escolhida da tabela 1, e todos os demais atributos daquela coluna são comparados para formar a porcentagem de similaridade. Cabe

ressaltar que o programa buscará a coluna com maior semelhança, e em alguns casos a semelhança poderá ser baixa. Mesmo com semelhança baixa, sendo a maior semelhança entre dois campos, eles serão associados, sendo indicado que a semelhança é baixa e que provavelmente os campos não armazenam o mesmo tipo de informação.

4.2.2 Tipos de dados

Para o tipo de dados, foram definidos valores fixos para cada caso, de acordo com a similaridade dos tipos de dados. Os dados numéricos, por exemplo, são semelhantes entre si, porém há diferença entre um dado do tipo *integer* e um dado do tipo *double*. Ambos são números, mas na conversão de um *double* para *integer*, perde-se a exatidão do valor. Para os vários tipos de dados, foram definidos os valores abaixo, após uma análise comparativa entre eles.

Números: smallint/(integer or serial): **0.5**

smallint/(bigint or decimal or numeric or real or double precision or bigserial): **0.1**

integer/(smallint or bigint or decimal or numeric or real or double or serial or bigserial): **0.22**

outras comparações: **0.15**

Texto: A similaridade, para os tipos de dados *char* e *varchar*, levará em conta o tamanho máximo, dividindo o tamanho da menor string pelo tamanho da maior. No caso de um campo *estado*, por exemplo, a diferença entre SC e Santa Catarina será grande ($2/14 \approx 0,142$ similar), pois o número máximo de caracteres será bem diferente. *Char* e *Varchar* terão similaridade de **0.1**, independente do número de caracteres, pois a obrigatoriedade de um tamanho fixo pode inviabilizar uma possível união entre os dados.

Datas: Caso a formatação seja exatamente igual, o score de similaridade será **1**, a comparação entre *date/time* será **0**, pois data e hora são informações diferentes. Informações com e sem *timezone* terão diferença de **0.22**.

Boolean: Não há variação neste tipo de dados, portanto sempre serão iguais, e a similaridade será **1**.

Demais casos: Outros casos terão score de similaridade 0, por se tratar de valores que podem variar bastante, podem ter usos diferentes com o mesmo tipo, e são utilizados com freqüência muito baixa.

4.2.3 Demais metadados

Os demais metadados fornecem informações relevantes para a informação, porém não determinam a informação. Na integração é importante considerá-los, pois são

informações relevantes para a validação dos dados, e por isso entrarão no cálculo de similaridade. Terão um peso bem baixo em relação aos atributos vistos anteriormente, contribuindo com 15% da taxa de similaridade. Para Primary Key, Nullable, Updatable, Foreign Key e Restrição, caso a coluna tenha o mesmo valor nas duas tabelas, será atribuída a porcentagem de similaridade descrita anteriormente. Do contrário, a similaridade será zero. Dois campos similares em que um deles seja Nullable e o outro não, terá similaridade 0. Se forem iguais, terão similaridade 1. Esse valor de similaridade será multiplicado pelo peso do atributo Nullable fica $1 \times 0,02 = 0,02$, que é adicionado à soma geral da similaridade. Todos os valores serão somados e integrarão um *score* de similaridade, que varia entre 0 e 1. Esse valor, posteriormente, será convertido em porcentagem.

O **objetivo 1** é alcançado neste capítulo, com o projeto do sistema elaborado, permitindo o início do desenvolvimento.

5 DESENVOLVIMENTO

5.1 LINGUAGEM E RECURSOS

Para o processamento dos dados que ocorre no servidor, foi utilizado Node.js, com todo o código desenvolvido em JavaScript. Também foram utilizadas as bibliotecas *pg*, para acesso aos bancos de dados PostgreSQL, e *string-similarity*, onde é implementado o algoritmo Dice's Coefficient, para a comparação de strings. Os bancos de dados de testes foram criados no PostgreSQL, e armazenados em um servidor Microsoft Azure. A interface foi desenvolvida utilizando HTML e CSS, com comandos desenvolvidos em JavaScript. Neste capítulo, o **objetivo 2** é alcançado.

5.1.1 Classes e Funções

O software é composto por classes e funções. As classes buscam agrupar os dados para facilitar a manipulação através das funções. Cada função executa um conjunto de operações, retornando os dados após efetuar o processamento.

Classe `Attribute:` Possui atributos sobre uma coluna de uma tabela do banco de dados. Cada Attribute corresponde à uma coluna, com informações como o Nome, Tipo de dados, tamanho se for String, etc.

Classe `Result:` Uma classe semelhante à Attribute, porém com os dados das duas colunas mais semelhantes, junto com o percentual de semelhança entre elas. Usada para facilitar a formatação de saída.

Função `getKeys:` Função que acessa o banco de dados e obtém informações sobre as chaves primárias. São armazenadas para posterior comparação.

Função `getData:` Função responsável por acessar o banco de dados e obter as informações sobre as colunas da tabela. Nome da coluna, tipo de dados, tamanho da String, entre outras informações, são obtidas e salvas utilizando a classe Result.

Função `buildTable:` Função que constrói a tabela de resultados. Seleciona uma coluna da primeira tabela e procura a melhor correspondência na segunda tabela. Os dados são adicionados à um Array de objetos da classe Result.

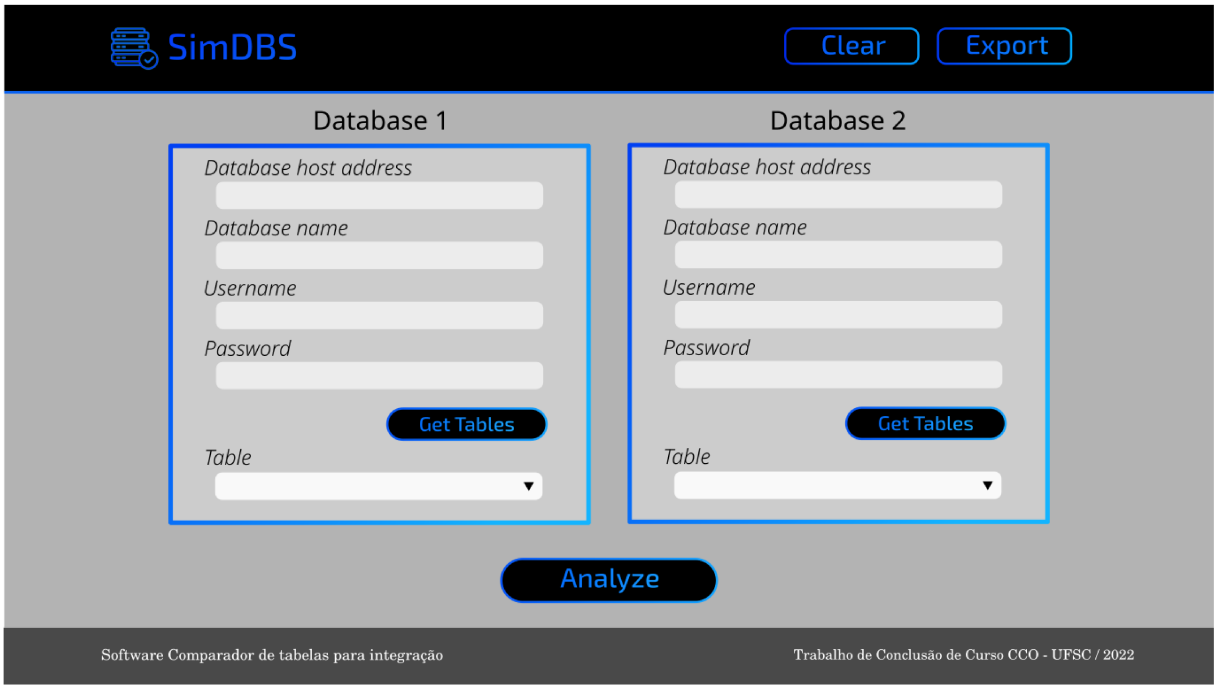
Função `typeSimilarity:` Função que compara o tipo de dados entre duas colunas. Caso sejam do mesmo tipo, o grau de similaridade é igual a 1, caso sejam diferentes, há valores fixos de similaridade entre alguns tipos semelhantes, conforme descrito na seção 4.2.2. Para tipos de dados incompatíveis, o grau de similaridade é 0.

Função `simPercentCalc`: Calcula a porcentagem de similaridade de acordo com o peso de cada atributo já determinado anteriormente. Após calcular o valor, retorna `attribSim`, que contém o resultado para ser agrupado com os demais e enviado para o cliente.

5.1.2 Projeto de interface

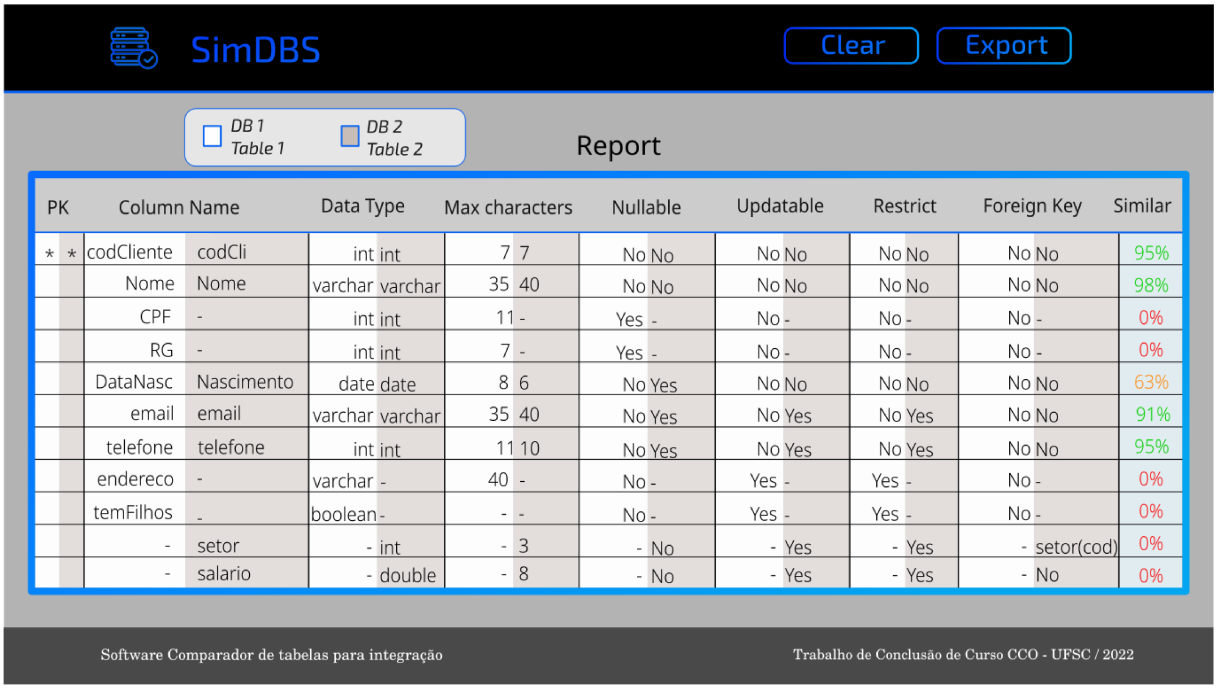
A interface com o usuário deve ser simples e objetiva, tornando fácil e intuitiva a utilização do software. A Figura 5 é o esboço da tela inicial do sistema, onde o usuário insere as informações para acesso ao banco de dados. Quando o usuário solicitar a comparação, o sistema irá processar as informações para retornar o resultado, como na Figura 6, onde há um esboço da saída. O objetivo 3 é atingido nesta seção.

Figura 5 – Tela inicial, onde o usuário insere as informações



Fonte: Própria

Figura 6 – Tela de resultados

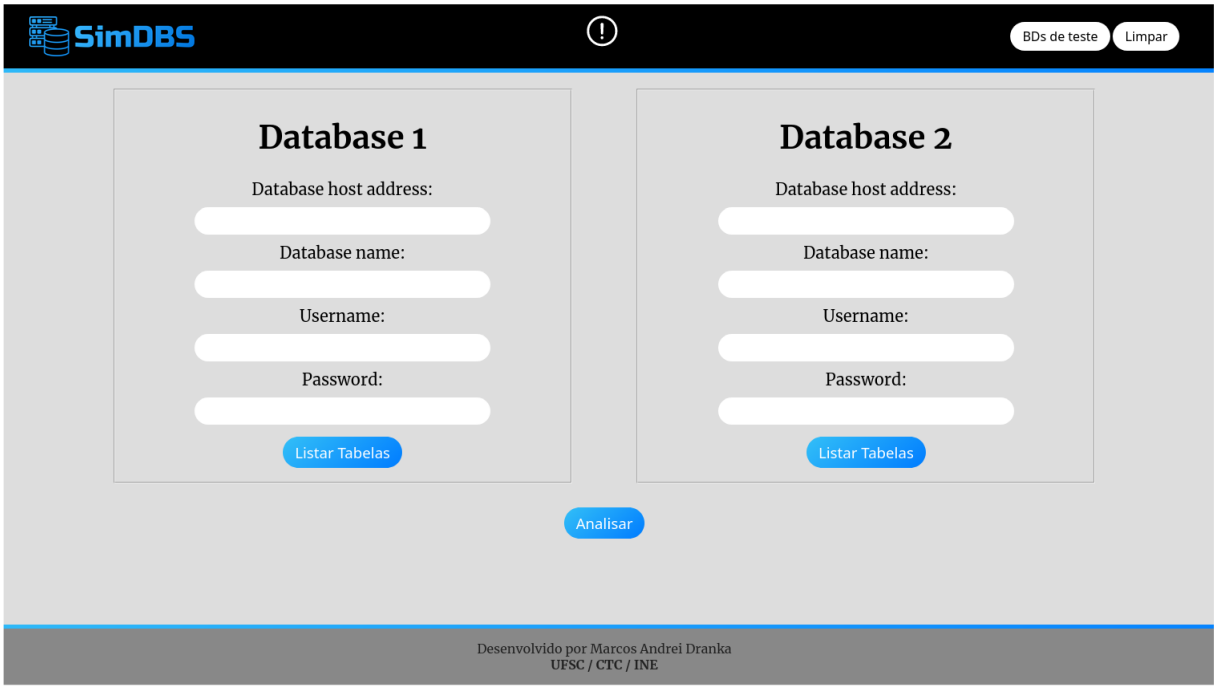


Fonte: Própria

5.1.3 Protótipo

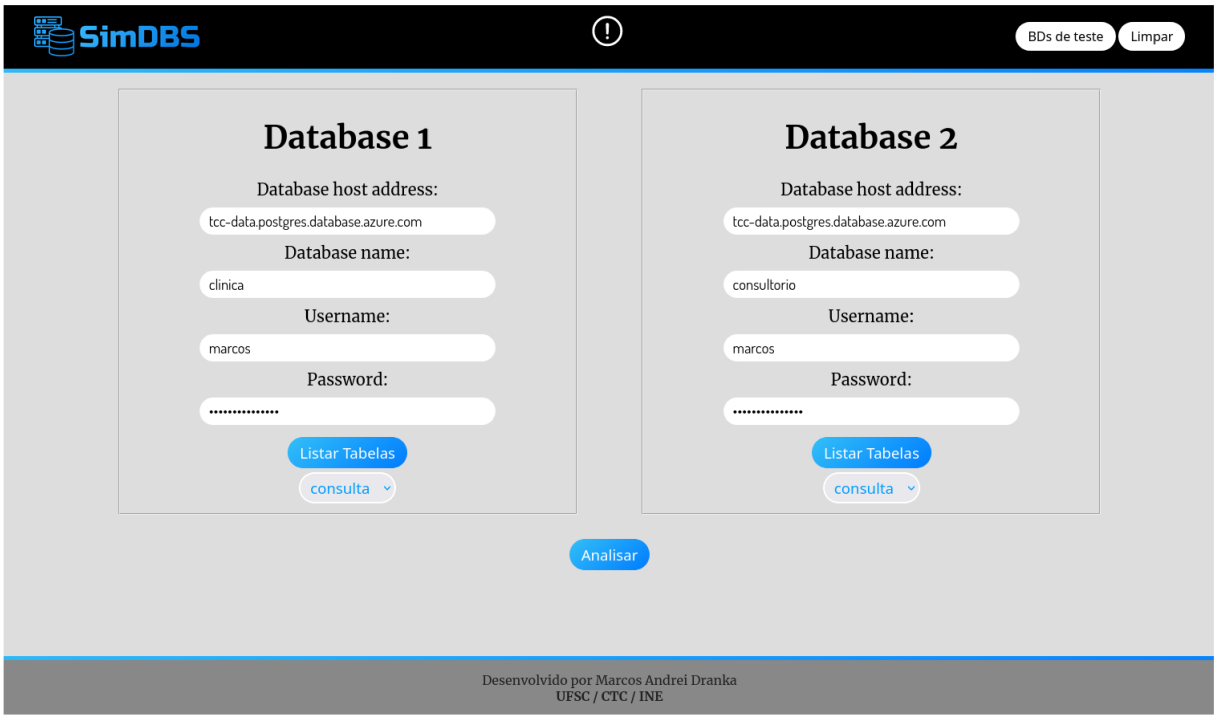
O protótipo foi desenvolvido buscando atingir os objetivos propostos no início do trabalho. Após o desenvolvimento do *back-end*, a interface foi desenvolvida utilizando HTML, CSS e JavaScript. Buscando simplificar ao máximo a utilização do software, mesmo para usuários iniciantes, algumas mudanças foram feitas na interface do programa. O software é executado em uma única tela, facilitando a análise, já que as informações inseridas continuam visíveis na tela junto com a análise feita pelo programa. Na Figura 7 é mostrado o visual atualizado, mantendo muitas características do esboço visto anteriormente, com algumas melhorias na escolha das cores e organização da tela.

Figura 7 – Tela inicial, mostrada ao acessar o SimDBS



Fonte: Própria

Figura 8 – Após preencher os dados de acesso e listar as tabelas disponíveis



Fonte: Própria

Após inserir as informações de acesso ao Banco de dados e clicar no botão Listar Tabelas, o SimDBS irá buscar as tabelas disponíveis no banco de dados e mostrar em um *dropdown*. A primeira tabela já estará selecionada, sendo possível alterar para

fazer os testes desejados. Na Figura 8 é possível ver que em ambos os bancos a tabela selecionada é a tabela consulta. Após selecionar as duas tabelas, é possível executar a análise, clicando no botão Analisar. O resultado é mostrado em uma tabela, conforme a Figura 9, onde os dados são formatados utilizando as cores verde, laranja e vermelho, indicando o grau de semelhança.

Figura 9 – Resultado formatado, permitindo uma visualização mais rápida

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	crm	smallint	integer	0	0	NO	YES	NO	NO	YES	YES	NO	NO	82.5%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	especialidade	varchar	varchar	20	20	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtcontrato	datacontrato	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	82.0%
telefone	telefone	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Própria

5.1.3.1 Definições de desenvolvimento do Protótipo

Durante o desenvolvimento do protótipo, vários testes foram executados para verificar se os objetivos estavam sendo alcançados. De acordo com os resultados dos testes, algumas definições foram adotadas, buscando uma interface limpa e fácil de entender, para que a compreensão dos dados pelo usuário seja rápida. Dentre as definições adotadas, as mais importantes estão descritas abaixo:

Tela única: Com o objetivo de facilitar a usabilidade, optou-se pelo desenvolvimento em tela única, já que o usuário irá comparar várias tabelas. Dessa forma, após a comparação de duas tabelas, o usuário pode selecionar novas tabelas para comparação sem precisar esperar o carregamento de uma nova página, bastando selecionar as tabelas e clicar em analisar.

Critérios de similaridade: Alguns critérios foram utilizados para fornecer uma tabela comparativa com maior precisão, agrupando os dados de forma automática. O agrupamento é feito escolhendo cada uma das colunas da primeira tabela

selecionada e comparando com todas as colunas da segunda tabela. As colunas com maior semelhança entre os nomes são agrupadas se a semelhança for maior que 30%, utilizando o algoritmo *Dice's Coefficient*. O nome equivale a 60% da similaridade total da coluna, fazendo com que, na prática, os dados sejam agrupados apenas se a similaridade for maior que 18% entre as colunas. As colunas que não atingirem esse percentual mínimo, não são agrupadas, e aparecem listadas na tabela sem nenhuma associação. Quando duas colunas obtêm a maior semelhança e estão acima da porcentagem mínima, elas são associadas e comparadas. Caso a semelhança, representada na tabela pela coluna *simScore*, seja menor ou igual a 60%, a linha toda terá o texto em vermelho, indicando que aquelas colunas tem semelhança muito baixa. Já para taxas de similaridade maiores que 60% e menores ou iguais a 85%, a cor do texto será laranja, informando que os campos possuem uma similaridade considerável, mas não é garantido que as informações salvas nas duas colunas armazenam o mesmo tipo de informação. As colunas com similaridade maior que 85%, por sua vez, são mostradas na cor verde, indicando a alta semelhança entre as colunas. Visualmente, é possível saber se a semelhança entre as tabelas comparadas é alta ou baixa instantaneamente, bastando observar qual é a cor predominante.

Efeito *hover* nas linhas: Quando se está trabalhando com tabelas com muitas colunas, a comparação irá apresentar uma tabela consideravelmente grande como resultado. Para facilitar a visualização dos dados, ao passar o cursor do mouse sobre uma linha, toda ela terá o *background* alterado, permitindo visualizar aquela linha destacada entre as demais.

As definições listadas acima cumprem o **objetivo 3**.

6 EXPERIMENTOS INICIAIS

6.1 OBTENDO O SOFTWARE ATRAVÉS DO GITHUB

O software está disponível no GitHub, acessando `github.com/mdranka/dbsc`. Para usá-lo, basta seguir as instruções descritas na página do GitHub, ou acessando o arquivo README.md que está no repositório. Há no software os dados de acesso de dois bancos de dados de teste. Caso não estejam mais disponíveis online, há o código SQL utilizado para a criação dos mesmos. Para usar os bancos de dados já inclusos, basta utilizar o botão *Bancos de teste*, que eles serão preenchidos automaticamente.

6.2 TESTES COM O SOFTWARE

Foram efetuados testes com o objetivo de verificar se o software conseguia acessar os bancos de dados, obter e organizar as informações e mostrar os resultados da análise. Ao cumprir tais requisitos, o **objetivo 4** deste trabalho é alcançado.

Para os testes, foram utilizados alguns bancos de dados criados criados especificamente para isso, ilustrando diferentes cenários e avaliando se o resultado foi satisfatório. A tabela de resultados mostra as informações lado a lado, tornando rápida a observação dos valores comparados. Cada linha representa uma coluna da tabela 1 associada com uma coluna da tabela 2, com os detalhes da comparação e o percentual de similaridade. Caso não seja encontrada uma coluna similar o suficiente para ser agrupada na mesma linha, a coluna aparecerá sozinha na linha, para as duas tabelas comparadas. As colunas `name1` e `name2` correspondem, respectivamente, ao nome da coluna da tabela 1 e o nome da coluna da tabela 2. De forma semelhante, as colunas `type1` e `type2` indicam o tipo de dados na tabela 1 e tabela 2. `size1` e `size2` indicam o tamanho da String, caso o tipo seja string. Caso contrário, mostra `0`. `pk1` e `pk2` indicam se o campo é chave primária. `nullable1` e `nullable2` indicam se o campo pode estar vazio. `updatable1` e `updatable2` indicam se é permitido alterações. `fk1` e `fk2` indicam se o campo é chave estrangeira. Por fim, a coluna `simScore` mostra a porcentagem de similaridade entre as colunas das duas tabelas. Na Figura 10 ocorre a comparação entre a tabela `medico` do banco de dados `clinica` e a tabela `medico` do banco `consultorio`. São duas tabelas semelhantes, e espera-se que haja semelhança entre as colunas das duas tabelas. Podemos observar que os campos que há linhas grifadas em verde (5), linhas em laranja (3) e linhas em vermelho (3). Podemos rapidamente notar que os campos `codmedico` e `bairro` da primeira tabela não possuem correspondente na segunda tabela, da mesma forma que não há correspondente para a coluna `cpf` da segunda tabela. Algumas colunas, em laranja, possuem leves diferenças, indicando a necessidade de tratar os dados na integração. Já as colunas em verde possuem grande semelhança, indicando que são praticamente idênticas, dispensando tratamento

Figura 10 – Testes com o software (medico/medico)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	crm	smallint	integer	0	0	NO	YES	NO	NO	YES	YES	NO	NO	82.5%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	especialidade	varchar	varchar	20	20	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtcontrato	datacontrato	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	82.0%
telefone	telefone	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

Figura 11 – Testes com o software(medico/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	---	smallint	---	0	---	NO	---	NO	---	YES	---	NO	---	0.0%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
dtcontrato	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
telefone	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	idpaciente	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	rg	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	datanasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	convenio	---	varchar	---	15	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

especial em algus casos, como no campo `endereco`, por exemplo, que possui *simScore* de 100%. A Figura 11 representa outro cenário, onde as tabelas são diferentes, mas possuem colunas que armazenam informações semelhantes. Podemos observar que algumas colunas são mostradas em verde, uma delas em laranja e um número maior delas em vermelho. Isso já era esperado, por se tratarem de duas tabelas diferentes, a tabela `medico` e a tabela `paciente`. No terceiro cenário, comparamos uma outra tabela medico com ela mesma. Duas tabelas iguais devem gerar o resultado de 100% de similaridade, e é exatamente isso que acontece. Na Figura 12 observamos todas as linhas em verde e com `simScore` igual a 100%. Um cenário alternativo é mostrado da Figura 13 onde, embora as tabelas tenham o mesmo nome, as colunas são bem diferentes. Apenas uma coluna aparece em verde e uma em laranja, enquanto todas as outras estão em vermelho. Também podemos observar outra diferença nesta comparação em relação às anteriores, que é a associação de duas colunas diferentes, `cod_pessoa` e `nomepess`. Embora estejam associadas, são mostradas em vermelho, indicando que ambas dificilmente armazenam a mesma informação, por terem similaridade muito baixa, de apenas 30.5%. Com estes testes, criamos vários possíveis cenários de com-

Figura 12 – Testes com o software (medico)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
crm	crm	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	100.0%
nome	nome	varchar	varchar	30	30	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
idade	idade	integer	integer	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
codcidade	codcidade	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%
codesp	codesp	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%
numeroa	numeroa	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%

Fonte: Testes realizados pelo autor

Figura 13 – Testes com o software (pessoa/pessoa)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
cod_pessoa	nomepess	integer	varchar	0	40	YES	NO	NO	YES	YES	YES	NO	NO	30.5%
sexo	sexo	character	character	2	1	NO	NO	YES	YES	YES	YES	NO	NO	87.5%
nome	---	varchar	---	50	---	NO	---	YES	---	YES	---	NO	---	0.0%
sobrenome	---	varchar	---	50	---	NO	---	YES	---	YES	---	NO	---	0.0%
data_de_nascimento	datanasc	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	70.0%
codigo_local_nasc	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
data_de_falecimento	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codigo_local_falec	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
codigo_uniao_pais	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
---	cic	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	conjugecic	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	0.0%

Fonte: Testes realizados pelo autor

Figura 14 – Testes com o software(consulta/consulta)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
crm	---	integer	---	0	---	YES	---	NO	---	YES	---	YES	---	0.0%
rg	---	integer	---	0	---	YES	---	NO	---	YES	---	YES	---	0.0%
data	data	date	date	0	0	YES	YES	NO	NO	YES	YES	NO	NO	100.0%
hora	hora	integer	time without time zone	0	0	NO	YES	YES	NO	YES	YES	NO	NO	68.0%
coddoenca	codpac	integer	integer	0	0	NO	YES	YES	NO	YES	YES	YES	YES	51.5%
---	codmed	---	integer	---	0	---	NO	---	NO	---	YES	---	YES	0.0%
---	valor	---	numeric	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	codconv	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	0.0%

Fonte: Testes realizados pelo autor

paração, observando que os resultados fornecidos pelo SimDBS são satisfatórios em relação ao que se esperava. Nas Figuras 14, 15, 16, 17, 18 e 19 são mostrados outros testes realizados com outros bancos de dados, também com resultados satisfatórios. Com o resultado destes testes, o objetivo 4 foi cumprido.

Figura 15 – Testes com o software (consulta/consulta)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codcons	codconsulta	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	85.0%
dtcons	datconsulta	date	date	0	0	NO	NO	NO	NO	YES	YES	NO	NO	72.0%
horacons	horaconsulta	time without time zone	time without time zone	0	0	NO	NO	YES	YES	YES	YES	NO	NO	86.7%
codmed	crmmedico	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	67.7%
codpac	codpaciente	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	80.0%
sala	nrsala	character	character	6	6	NO	NO	YES	YES	YES	YES	NO	NO	85.0%
valorcons	valorconsulta	money	money	0	0	NO	NO	YES	YES	YES	YES	NO	NO	88.0%

Fonte: Testes realizados pelo autor

Figura 16 – Testes com o software(consulta/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codcons	convenio	integer	varchar	0	15	YES	NO	NO	YES	YES	YES	NO	NO	26.5%
dtcons	---	date	---	0	---	NO	---	NO	---	YES	---	NO	---	0.0%
horacons	---	time without time zone	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codmed	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
codpac	idpaciente	integer	integer	0	0	NO	YES	YES	NO	YES	YES	YES	NO	54.7%
sala	---	character	---	6	---	NO	---	YES	---	YES	---	NO	---	0.0%
valorcons	---	money	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	nome	---	varchar	---	45	---	NO	---	NO	---	YES	---	NO	0.0%
---	rg	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	endereco	---	varchar	---	80	---	NO	---	YES	---	YES	---	NO	0.0%
---	cidade	---	varchar	---	25	---	NO	---	YES	---	YES	---	NO	0.0%
---	estado	---	character	---	2	---	NO	---	YES	---	YES	---	NO	0.0%
---	datanasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

Figura 17 – Testes com o software (funcionario/professor)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
rg	rg	integer	integer	0	0	YES	NO	NO	NO	YES	YES	NO	NO	95.0%
nome	nome	varchar	varchar	30	40	NO	NO	YES	YES	YES	YES	NO	NO	93.8%
idade	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codcidade	codcid	integer	integer	0	0	NO	NO	YES	NO	YES	YES	YES	YES	84.2%
salario	---	double precision	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	codigo	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	sexo	---	character	---	1	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

Figura 18 – Testes com o software(paciente/funcionario)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codpac	codcidade	integer	integer	0	0	YES	NO	NO	YES	YES	YES	NO	YES	47.5%
nomepac	nome	varchar	varchar	40	30	NO	NO	NO	YES	YES	YES	NO	NO	71.8%
rg	rg	integer	integer	0	0	NO	YES	YES	NO	YES	YES	NO	NO	93.0%
cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
endereco	---	varchar	---	80	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	idade	varchar	integer	30	0	NO	NO	YES	YES	YES	YES	NO	NO	68.3%
estado	---	character	---	2	---	NO	---	YES	---	YES	---	NO	---	0.0%
dtnasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
convenio	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	salario	---	double precision	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

Figura 19 – Testes com o software (paciente/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codpac	idpaciente	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	65.7%
nomepac	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
rg	rg	integer	integer	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
cpf	cpf	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtnasc	datanasc	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	70.0%
convenio	convenio	varchar	varchar	20	15	NO	NO	YES	YES	YES	YES	NO	NO	93.8%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Fonte: Testes realizados pelo autor

7 TESTES DE USABILIDADE

7.1 ESCALA SUS

Testar a usabilidade não é uma tarefa simples, pois cada usuário tem um nível de conhecimento diferente e a facilidade de uso acaba se tornando muito subjetiva. Foi utilizada a escala SUS , desenvolvida por John Brooke, pois é baseada na experiência de uso, captando a experiência subjetiva do usuário. O teste foi realizado com 3 pessoas de áreas que podem vir a utilizar o sistema. As pessoas tabalham nas áreas de Big Data, Front-end e Full Stack. Por serem pessoas da área da programação, é de se esperar uma maior facilidade no uso da ferramenta. O teste cumpre o **objetivo 5** do trabalho.

7.1.1 Realização do teste

O objetivo do teste é verificar a facilidade de uso do sistema e coletar sugestões de melhoria no projeto. Para atingir o objetivo, buscou-se fornecer apenas as informações essenciais, como a explicação sobre o que é o SimDBS e qual o problema que ele pretende resolver. Se fosse necessário uma longa explicação antes do teste, isso já é um indício de que a usabilidade deixa a desejar. No próprio sistema há um *card* com algumas explicações básicas sobre o uso, para que qualquer usuário possa entender rapidamente o funcionamento. Antes do teste, as únicas informações passadas aos usuários que iriam testar foram:

> O SimDBS tem por objetivo auxiliar no processo de Integração de dados, comparando a estrutura de duas tabelas e informando as diferenças e semelhanças entre elas. (Por ser um protótipo, recomenda-se o uso apenas de bancos de dados criados exclusivamente para testes. Não é necessário criar um novo BD para testes, há dois BDs disponíveis no próprio sistema). O objetivo do teste é avaliar a usabilidade. As únicas informações fornecidas são as que estão no próprio sistema. O sistema foi projetado para ser intuitivo, de forma que uma pessoa com conhecimento básico em bancos de dados seja capaz de utilizá-lo. O sistema foi projetado para uso em computador. Em dispositivos móveis a página poderá ficar desconfigurada. Sugere-se a utilização de um computador.

Após os testes, os usuários responderam ao questionário SUS (BARROS, 2022), informando, para cada pergunta, um valor entre 1 e 5, com 1 representando 'Discordo totalmente' e 5 representando 'Concordo totalmente'. Os critérios que o SUS ajuda a avaliar são:

- **Eficácia:** se as pessoas podem realmente completar suas tarefas e atingir seus

objetivos.

- **Eficiência:** a medida em que eles gastam recursos para alcançar seus objetivos.
- **Satisfação:** o nível de conforto que eles experimentam para alcançar esses objetivos.

Foram disponibilizadas 10 perguntas, das quais as de número ímpar questionam um aspecto positivo, no qual quanto maior a pontuação, melhor, e as perguntas de número par questionam um aspecto negativo, onde pontuações menores são melhores. Essa ordem foi pensada para diminuir o viés ao responder perguntas sempre no mesmo tom. As questões feitas sofreram pequenas alterações para se adequarem melhor à este trabalho, sendo enumeradas abaixo:

1. Eu acho que gostaria de usar esse sistema com frequência.
2. Eu acho o sistema desnecessariamente complexo.
3. Eu achei o sistema fácil de usar.
4. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema.
5. Eu acho que as funções do sistema estão bem integradas.
6. Eu acho que o sistema apresenta muita inconsistência.
7. Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente.
8. Eu achei o sistema atrapalhado de usar.
9. Eu senti confiança ao usar o sistema.
10. Eu precisei aprender várias coisas novas antes de conseguir usar o sistema.

Após coletar as respostas, foi calculada a média da pontuação de cada usuário (valores entre 0 e 100) e obtido o valor **92.5**. Na escala SUS, considera-se aceitável valores acima de 68. Como o valor está bem acima, significa que o sistema apresenta uma excelente usabilidade. É importante salientar que os usuários que participaram do teste já têm conhecimentos básicos sobre bancos de dados, e um usuário comum talvez possa ter dificuldades. O teste foi feito com pessoas que pertencem ao público-alvo da solução, e para esse público, o resultado dos testes foi positivo.

8 CONCLUSÃO

8.1 CONCLUSÃO SOBRE O TRABALHO ATUAL

Com a demanda cada vez maior de sistemas mais robustos para gerenciamento de informações, a integração de dados torna-se imprescindível. Com o desenvolvimento do SimDBS, espera-se contribuir na tomada de decisão por parte dos Administradores de Bancos de Dados no processo de integração de dados. A comparação entre tabelas correlatas em bancos de dados diferentes é uma das etapas mais importantes para verificar quais as necessidades de ajuste nos dados para integrá-los, ou até mesmo para verificar se uma integração entre as tabelas é viável. Através dos testes, verificamos que o processo de análise é muito simples, bastando informar os dados de acesso ao banco e selecionar as tabelas a serem comparadas. Isso poupa um tempo considerável do DBA, que não precisa mais obter os dados manualmente para comparar, bastando inserir os dados e clicar em um botão. O processo que levaria um longo tempo, agora pode ser executado em cerca de um ou dois minutos.

8.2 ATIVIDADES FUTURAS

Ao longo do desenvolvimento, surgiram novas idéias que poderiam ser integradas ao SimDBS, tornando-o mais completo e ainda mais efetivo, mas que não puderam ser elaborados neste momento. Durante os testes de usabilidade, alguns usuários sugeriram algumas melhorias também, onde algumas foram aplicadas, e outras, mais trabalhosas, se tornaram idéias para novos trabalhos. Algumas sugestões de trabalhos que podem partir deste como base são:

- Automatizar ainda mais o SimDBS, fazendo a análise completa do banco de dados, buscando tabelas correlatas em bancos de dados diferentes e mostrando as comparações de todas as tabelas.
- Expandir o back-end, permitindo o processamento de dados de outros BDs além do PostgreSQL.
- Permitir a entrada de outros tipos de dados para representar o banco, como os comando SQL de criação ou uma entrada JSON, à escolha do usuário.
- Um *Dashboard* mostrando os principais pontos da comparação e informações importantes que auxiliem o DBA na tomada de decisão mais rápida.
- Criar uma comparação mais robusta, buscando considerar mais metadados na comparação.
- Automatizar a definição de pesos dos metadados.

REFERÊNCIAS BIBLIOGRÁFICAS

BARROS, Myrela. **Guia atualizado de como utilizar a escala SUS (System Usability Scale) no seu produto**. 2022. Disponível em: <https://brasil.uxdesign.cc/guia-atualizado-de-como-utilizar-a-escala-sus-system-usability-scale-no-seu-produto-ab773f29c522>. Acesso em: 26 mai. 2023. Citado na p. 37.

CHEN, Hao. **String metrics and word similarity applied to information retrieval**. 2012. Diss. (Mestrado) – Itä-Suomen yliopisto. Citado nas pp. 18, 20.

DIGITAL, Instituto Metrópole. **Visualização da estrutura de uma tabela**. Disponível em: <https://materialpublic.imd.ufrn.br/curso/disciplina/3/73/10/2>. Acesso em: 2 jul. 2023. Citado na p. 11.

GOMAA, Wael H; FAHMY, Aly A *et al.* A survey of text similarity approaches. **international journal of Computer Applications**, Citeseer, v. 68, n. 13, p. 13–18, 2013.

MONTEIRO, José Maria; LIFSCHITZ, Sérgio; BRAYNER, Ângelo. **Extraindo Metadados de SGBDs**. 2007. UNIFOR. Citado nas pp. 11, 16, 18, 20.

ROSA, Ana Caroline Ignácio da. **Interface para visualização de dados brutos vindos de diferentes fontes**. 2022. Universidade Federal de Santa Catarina. Disponível em: <https://repositorio.ufsc.br/handle/123456789/238001>. Acesso em: 22 abr. 2023. Citado nas pp. 11, 19, 20.

Apêndices

APÊNDICE A – ARTIGO SOBRE O SIMDBS NO FORMATO SBC

SimDBS: Sistema auxiliar para comparação estrutural de diferentes bancos de dados no processo de integração de dados.

Marcos Andrei Dranka[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

marcos.dranka@grad.ufsc.br

Abstract. *The large volume of data generated nowadays requires new ways of handling them. A research that uses heterogeneous data obtained from different sources, for example, needs to integrate the data to allow better results. The unification of systems is also an area that needs to integrate data, and for that, it is necessary to identify the similarities and differences in the structure of the Databases that store this information. The objective of this work is to produce software that automates the task of comparing different databases, saving time for the database administrator and allowing a more consistent result in data integration.*

Resumo. *O grande volume de dados gerado nos dias atuais necessita novas formas de tratá-los. Uma pesquisa que utiliza dados heterogêneos obtidos de diferentes fontes, por exemplo, precisam integrar os dados para permitir melhores resultados. A unificação de sistemas também é uma área que precisa integrar dados, e para isso, é necessário identificar as semelhanças e diferenças na estrutura dos Bancos de dados que armazenam essas informações. O objetivo deste trabalho é produzir um software que automatize a tarefa de comparar diferentes bancos de dados, poupando tempo do administrador de banco de dados e permitindo um resultado mais consistente na integração de dados.*

1. Introdução

O grande volume de dados produzido hoje em dia nos mais variados setores do mercado junto com as rápidas mudanças que ocorrem no mercado de tecnologia, trazem grandes desafios na maneira como os dados são armazenados e gerenciados. Em diversos casos, uma base de dados pode trazer informações insuficientes para uma análise estratégica, sendo necessário buscar informações complementares em outras bases. Outro desafio, é quando um sistema legado é substituído por um mais recente e completo. O novo *software* precisa importar os dados do anterior, que são de extrema importância, pois trazem toda a informação que a empresa obteve ao longo dos anos. Para esses casos, a análise da estrutura onde os dados estão armazenados é imprescindível para entender a melhor forma de trabalhar com esses dados.

Cada base de dados pode utilizar diferentes estruturas para armazenar dados semelhantes, pois, em um mesmo domínio, diferentes administradores de bancos de dados (DBAs) podem definir estruturas bem diferentes para construir sua base de dados, mesmo trabalhando com informações semelhantes. Para citar um exemplo, os sistemas utilizados

por secretarias da saúde de diversos municípios podem ser bem diferentes. Embora os dados busquem informar a mesma coisa, a forma de armazenamento pode diferir muito, não havendo um padrão. Quando os dados precisam ser enviados ao Ministério da Saúde, há a necessidade de padronizar os dados, para que não seja gerada uma confusão na chegada de dados nos sistemas do Ministério. A análise da estrutura dos bancos de dados auxilia muito no trabalho de quem precisa manipular esses dados para padronizá-los.

A análise desses dados é uma tarefa exaustiva e que requer muita atenção. A tarefa de obter os metadados manualmente, bem como criar planilhas ou modelos para analisar custam um tempo valioso do DBA, tornando a análise dos dados mais lenta e trabalhosa. Há vários metadados disponíveis, sendo alguns importantíssimos e outros irrelevantes para uma análise de similaridade. Uma ferramenta que permita obter e visualizar os dados rapidamente, poupa tempo e esforço do DBA, que poderá dar mais ênfase na integração de dados, obtendo melhores resultados.

Há algumas aplicações que buscam facilitar a visualização da estrutura de dados, porém de forma manual ou com poucas informações. O MySQL oferece uma forma de visualizar os metadados, mas com poucas informações e de forma manual. O Multiple DBData View fornece informações mais completas, porém a navegação é um pouco trabalhosa, sendo necessário alternar entre as abas, dificultando uma análise rápida.

Portanto, este trabalho busca resolver alguns dos problemas citados anteriormente, utilizando o SGBD PostgreSQL, por ser gratuito e com um número considerável de usuários. o SimDBS tem o objetivo de acessar dois bancos de dados simultaneamente e compará-los, trazendo o resultado da comparação lado a lado, de forma intuitiva e de fácil leitura.

1.1. Objetivo geral

O objetivo geral deste trabalho é desenvolver um *software* que auxilie no processo de integração de dados, fornecendo informações sobre a similaridade entre a estrutura dos bancos de dados informados. Para isso, o SimDBS irá receber as informações de acesso aos BDs, fazer a análise comparativa entre as estruturas dos mesmos e mostrar uma tabela comparativa com as informações lado a lado, de forma clara e de simples leitura.

1.2. Objetivos específicos

Objetivo 1: Elaborar o projeto de interface, comparar e escolher o algoritmo de similaridade a ser utilizado, definir a linguagem de desenvolvimento e as ferramentas que serão utilizadas.

Objetivo 2: Implementar o código responsável pela comparação, utilizando o algoritmo de similaridade mais adequado e automatizando o processo de comparação, através de agrupamento de colunas mais semelhantes.

Objetivo 3: Desenvolver uma interface fácil de usar, com informações visuais bem organizadas e detalhadas, permitindo ao usuário visualizar os dados rapidamente.

Objetivo 4: Avaliar o resultado produzido pelo *software*, com o uso de bancos de dados de testes e verificar se a saída gerada produz resultados satisfatórios.

Objetivo 5: Avaliar a usabilidade do *software*, através de testes com alguns usuários.

2. Algoritmos de avaliação de similaridade

O trabalho aborda conceitos como banco de dados PostgreSQL, similaridade de dados e estruturas de bancos de dados. O principal conceito tratado neste trabalho é a similaridade entre os dados.

2.1. Similaridade de dados

Sendo o principal conceito do trabalho, a similaridade de dados busca comparar dados e retornar um valor que representa a semelhança entre os dados (geralmente um valor entre 0 e 1, a depender do algoritmo). Para este trabalho, o algoritmo utilizado foi o Dice's Coefficient.

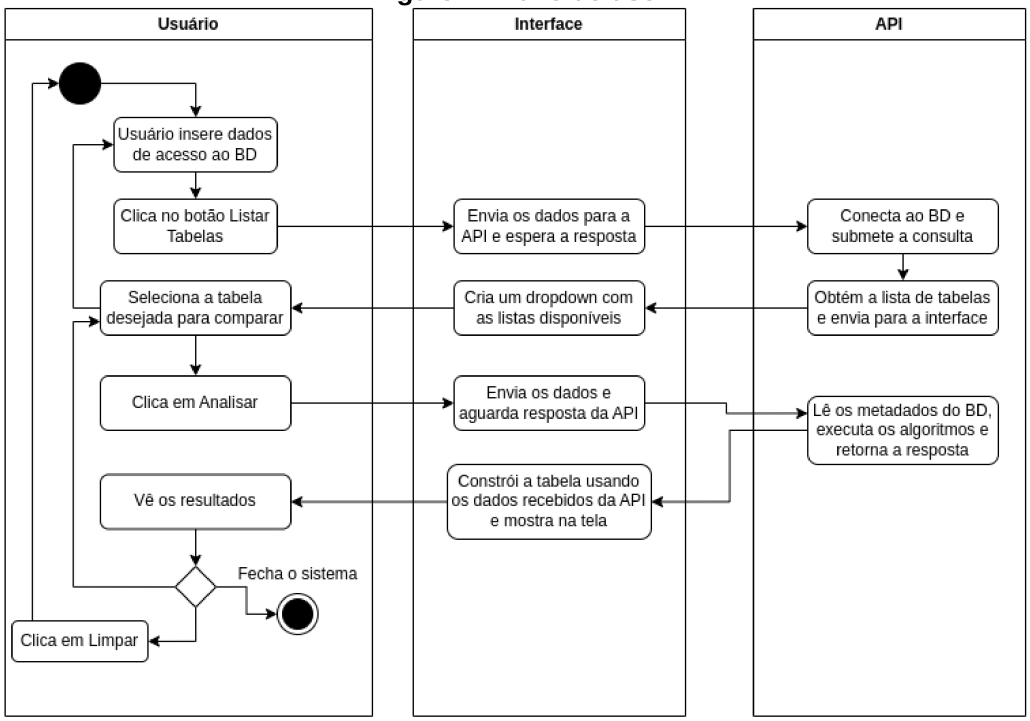
Dice's Coefficient: No campo da recuperação da informação, o coeficiente pode ser visto como o dobro da informação compartilhada, referente à soma das cardinalidades. O coeficiente também pode ser usado como uma medida de similaridade entre strings. Dadas duas strings x e y, podemos calcular o coeficiente da seguinte forma:
$s = \frac{2n_t}{n_x+n_y}$
onde n_t é o número de dígrafos (compostos de dois caracteres consecutivos) comuns às duas strings, n x é o número de dígrafos em x e n y o número de dígrafos em y . Por exemplo, para calcular a semelhança entre:
`nighte - nacht`,
calculamos os dígrafos de cada palavra:
`ni, ig, gh, ht`
`na, ac, ch, ht`
Cada conjunto possui quatro elementos, e sua interseção se reduz a um elemento ht. Com a fórmula fornecida acima, obtemos
$s = \frac{2\times1}{4+4} = 0,25$.

3. SimDBS

O SimDBS (Sistema auxiliar para comparação de estruturas de bancos de dados por similaridade para integração de dados) tem como objetivo simplificar a análise para integração de dados. Um dos objetivos da construção do sistema foi a usabilidade, permitindo que qualquer usuário com pouco conhecimento possa utilizar o sistema.

A Figura 1 mostra o fluxo seguido pelo usuário. Primeiro, o usuário insere as informações de acesso aos bancos de dados (host, usuário, senha e nome do banco). Após, clica no botão `Listar tabelas`, fazendo com que a sistema requisite à API as tabelas disponíveis. A API irá acessar o banco de dados com as informações fornecidas e retornará uma lista de tabelas disponíveis naquele banco de dados. As tabelas serão mostradas em uma lista no formato `dropdown`, para que o usuário possa escolher a tabela que deseja comparar. O processo se repete com o segundo banco de dados que será uttilizado na comparação. Após escolher as tabelas que serão utilizadas, basta clicar no botão `Analisar`, e o sistema irá requisitar à API o resultado da comparação. A API, novamente, acessa os bancos de dados e busca os metadados das tabelas informadas. O cálculo de similaridade entre as propriedades é efetuado e o resultado é retornado para o *front-end*. O SimDBS constrói a tabela utilizando as informações que recebeu da API. O usuário pode, então, escolher outras tabelas para comparar, ou mesmo outros bancos de dados, e prosseguir com as análises.

Figure 1. Fluxo de uso



3.1. Similaridade entre tabelas e atributos

Uma das características principais do SimDBS é a comparação entre as tabelas por similaridade. Para isso, foi necessário buscar um algoritmo que obtivesse valores condizentes, e que se comportasse bem em comparações das diversas colunas, sem apresentar anomalias em casos isolados.

As colunas das tabelas apresentam várias informações para comparação, sendo que as informações mais significativas são o nome da coluna e o tipo de dados. Com essas duas informações, podemos determinar se os dados armazenados naquela coluna tem alguma semelhança, ou essa semelhança pode ser descartada. Levando em conta a análise de algumas tabelas, observa-se que há colunas que se parecem muito, independente da tabela, por se tratar de informações padronizadas, como datas, número de documentos, nomes de estados, etc. Há também, campos bem diversos, que dependem exclusivamente do domínio da aplicação, e que podem variar bastante. Em geral, um DBA experiente utilizará nomes que evitem ambigüidade, o que produz uma certa padronização nos nomes dos atributos.

Com base nisso, determinou-se que o nome da coluna e o tipo de dados irão representar 85% da similaridade entre os campos, e os demais 25% serão definidos por informações que possuem menor relevância. A definição das porcentagens foi feita avaliando quais metadados têm maior relevância ao informar qual a informação está salvos naquela coluna do banco. A forma que um banco de dados vai armazenar uma determinada informação pode variar de acordo com o domínio da aplicação, ou mesmo critérios subjetivos do DBA que projetou o banco de dados. Independente desses critérios, o principal metadado que irá identificar a informação armazenada é o nome da coluna, por carregar todo o significado semântico da informação. Os pesos dos metadados para o

cálculo de similaridade foram definidos de forma intuitiva, em um trabalho futuro pode-se automatizar utilizando algum algoritmo de machine learning. Foram definidos assim:

- Nome do Atributo: 60%
- Tipo de dados (e tamanho, para strings): 25%
- Primary Key - 5%
- Nullable - 2%
- Updatable - 2%
- Foreign Key - 4%
- Restrição - 2%

3.1.1. Nome do Atributo

Para o nome do atributo, a principal métrica de similaridade é a semelhança entre o nome do campo. Isso determina 60% da similaridade do campo. O nome é a principal identificação sobre a informação que está sendo armazenada naquela coluna. Para calcular a similaridade, foi utilizado o algoritmo Dice's Coefficient, selecionando uma das colunas da tabela 1 e comparando com todas as colunas da tabela 2, buscando encontrar a mais similar. Então, a coluna da tabela 2 com maior similaridade no nome é associada à coluna previamente escolhida da tabela 1, e todos os demais atributos daquela coluna são comparados para formar a porcentagem de similaridade. Cabe ressaltar que o programa buscará a coluna com maior semelhança, e em alguns casos a semelhança poderá ser baixa. Mesmo com semelhança baixa, sendo a maior semelhança entre dois campos, eles serão associados, sendo indicado que a semelhança é baixa e que provavelmente os campos não armazenam o mesmo tipo de informação.

3.1.2. Tipos de dados

Para o tipo de dados, foram definidos valores fixos para cada caso, de acordo com a similaridade dos tipos de dados. Os dados numéricos, por exemplo, são semelhantes entre si, porém há diferença entre um dado do tipo *integer* e um dado do tipo *double*. Ambos são números, mas na conversão de um *double* para *integer*, perde-se a exatidão do valor. Para os vários tipos de dados, foram definidos os valores abaixo, após uma análise comparativa entre eles.

Números: smallint/(integer or serial): **0.5**
smallint/(bigint or decimal or numeric or real or double precision or bigserial): **0.1**
integer/(smallint or bigint or decimal or numeric or real or double or serial or bigserial): **0.22**
outras comparações: **0.15**

Texto: A similaridade, para os tipos de dados *char* e *varchar*, levará em conta o tamanho máximo, dividindo o tamanho da menor string pelo tamanho da maior. No caso de um campo *estado*, por exemplo, a diferença entre SC e Santa Catarina será grande ($2/14 \approx 0,142$ similar), pois o número máximo de caracteres será bem diferente. *Char* e *Varchar* terão similaridade de **0.1**, independente do número de caracteres, pois a obrigatoriedade de um tamanho fixo pode inviabilizar uma possível união entre os dados.

Datas: Caso a formatação seja exatamente igual, o score de similaridade será **1**, a comparação entre *date/time* será **0**, pois data e hora são informações diferentes. Informações com e sem *timezone* terão diferença de **0.22**.

Boolean: Não há variação neste tipo de dados, portanto sempre serão iguais, e a similaridade será **1**.

Demais casos: Outros casos terão score de similaridade 0, por se tratar de valores que podem variar bastante, podem ter usos diferentes com o mesmo tipo, e são utilizados com freqüência muito baixa.

3.1.3. Demais metadados

Os demais metadados fornecem informações relevantes para a informação, porém não determinam a informação. Na integração é importante considerá-los, pois são informações relevantes para a validação dos dados, e por isso entrarão no cálculo de similaridade. Terão um peso bem baixo em relação aos atributos vistos anteriormente, contribuindo com 15% da taxa de similaridade. Para Primary Key, Nullable, Updatable, Foreign Key e Restrição, caso a coluna tenha o mesmo valor nas duas tabelas, será atribuída a porcentagem de similaridade descrita anteriormente. Do contrário, a similaridade será zero. Dois campos similares em que um deles seja Nullable e o outro não, terá similaridade 0. Se forem iguais, terão similaridade 1. Esse valor de similaridade será multiplicado pelo peso do atributo Nullable fica $1 \times 0,02 = 0,02$, que é adicionado à soma geral da similaridade. Todos os valores serão somados e integrarão um *score* de similaridade, que varia entre 0 e 1. Esse valor, posteriormente, será convertido em porcentagem.

3.2. Protótipo

O protótipo foi desenvolvido buscando atingir os objetivos propostos no início do trabalho. Após o desenvolvimento do *back-end*, a interface foi desenvolvida utilizando HTML, CSS e JavaScript. Buscando simplificar ao máximo a utilização do software, mesmo para usuários iniciantes, algumas mudanças foram feitas na interface do programa. O software é executado em uma única tela, facilitando a análise, já que as informações inseridas continuam visíveis na tela junto com a análise feita pelo programa. Na Figura 2 é mostrado o visual atualizado, mantendo muitas características do esboço visto anteriormente, com algumas melhorias na escolha das cores e organização da tela.

Após inserir as informações de acesso ao Banco de dados e clicar no botão Listar Tabelas, o SimDBS irá buscar as tabelas disponíveis no banco de dados e mostrar em um *dropdown*. A primeira tabela já estará selecionada, sendo possível alterar para fazer os testes desejados. Na Figura 3 é possível ver que em ambos os bancos a tabela selecionada é a tabela consulta. Após selecionar as duas tabelas, é possível executar a análise, clicando no botão Analisar. O resultado é mostrado em uma tabela, conforme a Figura 4, onde os dados são formatados utilizando as cores verde, laranja e vermelho, indicando o grau de semelhança.

3.2.1. Definições de desenvolvimento do Protótipo

Durante o desenvolvimento do protótipo, vários testes foram executados para verificar se os objetivos estavam sendo alcançados. De acordo com os resultados dos testes, algumas

Figure 2. Tela inicial, mostrada ao acessar o SimDBS

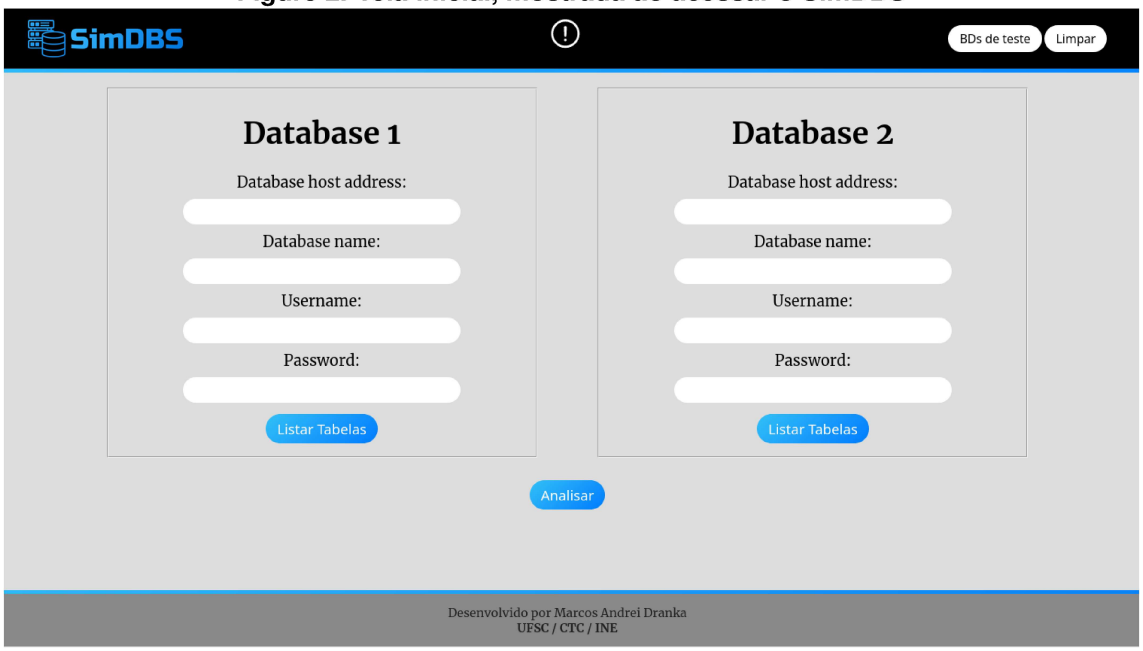


Figure 3. Após preencher os dados de acesso e listar as tabelas disponíveis

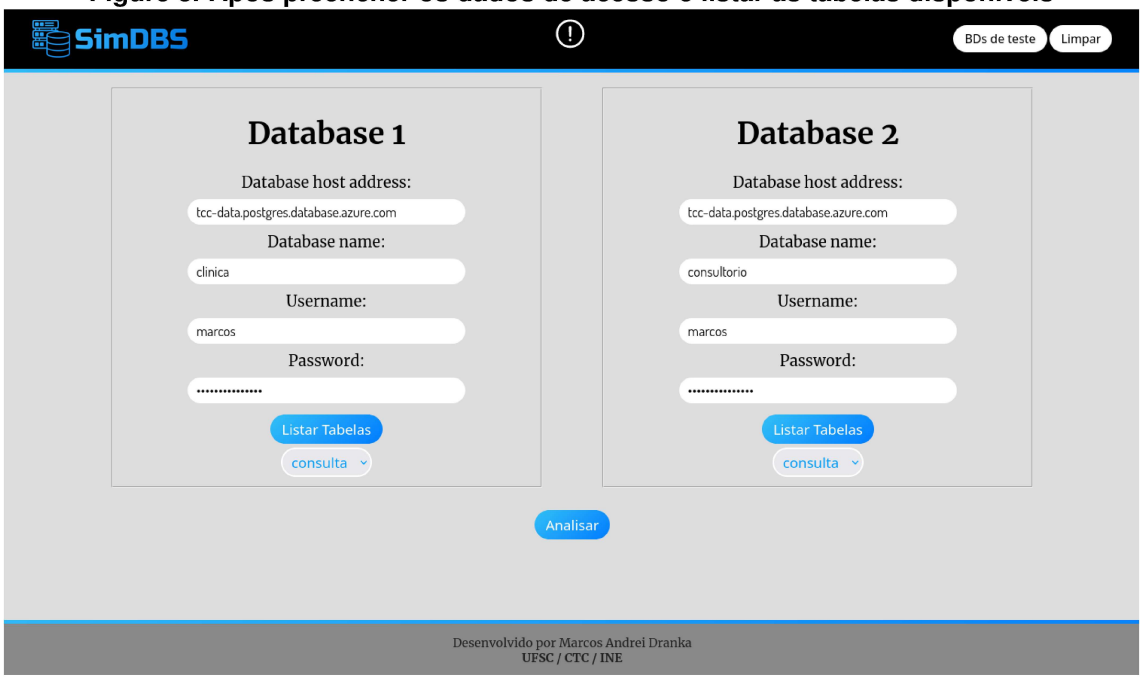


Figure 4. Resultado formatado, permitindo uma visualização mais rápida

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	crm	smallint	integer	0	0	NO	YES	NO	NO	YES	YES	NO	NO	82.5%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	especialidade	varchar	varchar	20	20	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtcontrato	datacontrato	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	82.0%
telefone	telefone	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

definições foram adotadas, buscando uma interface limpa e fácil de entender, para que a compreensão dos dados pelo usuário seja rápida. Dentre as definições adotadas, as mais importantes estão descritas abaixo:

Tela única: Com o objetivo de facilitar a usabilidade, optou-se pelo desenvolvimento em tela única, já que o usuário irá comparar várias tabelas. Dessa forma, após a comparação de duas tabelas, o usuário pode selecionar novas tabelas para comparação sem precisar esperar o carregamento de uma nova página, bastando selecionar as tabelas e clicar em analisar.

Critérios de similaridade: Alguns critérios foram utilizados para fornecer uma tabela comparativa com maior precisão, agrupando os dados de forma automática. O agrupamento é feito escolhendo cada uma das colunas da primeira tabela selecionada e comparando com todas as colunas da segunda tabela. As colunas com maior semelhança entre os nomes são agrupadas se a semelhança for maior que 30%, utilizando o algoritmo *Dice's Coefficient*. O nome equivale a 60% da similaridade total da coluna, fazendo com que, na prática, os dados sejam agrupados apenas se a similaridade for maior que 18% entre as colunas. As colunas que não atingirem esse percentual mínimo, não são agrupadas, e aparecem listadas na tabela sem nenhuma associação. Quando duas colunas obtêm a maior semelhança e estão acima da porcentagem mínima, elas são associadas e comparadas. Caso a semelhança, representada na tabela pela coluna *simScore*, seja menor ou igual a 60%, a linha toda terá o texto em vermelho, indicando que aquelas colunas tem semelhança muito baixa. Já para taxas de similaridade maiores que 60% e menores ou iguais a 85%, a cor do texto será laranja, informando que os campos possuem uma similaridade considerável, mas não é garantido que as informações salvas nas duas colunas armazenam o mesmo tipo de informação. As colunas com similaridade maior que 85%, por sua vez, são mostradas na cor verde, indicando a

alta semelhança entre as colunas. Visualmente, é possível saber se a semelhança entre as tabelas comparadas é alta ou baixa instantaneamente, bastando observar qual é a cor predominante.

Efeito *hover* nas linhas: Quando se está trabalhando com tabelas com muitas colunas, a comparação irá apresentar uma tabela consideravelmente grande como resultado. Para facilitar a visualização dos dados, ao passar o cursor do mouse sobre uma linha, toda ela terá o *background* alterado, permitindo visualizar aquela linha destacada entre as demais.

4. Obtendo o software através do GitHub

O software está disponível no GitHub, acessando `github.com/mdranka/dbsc`. Para usá-lo, basta seguir as instruções descritas na página do GitHub, ou acessando o arquivo README.md que está no repositório. Há no software os dados de acesso de dois bancos de dados de teste. Caso não estejam mais disponíveis online, há o código SQL utilizado para a criação dos mesmos. Para usar os bancos de dados já inclusos, basta utilizar o botão *Bancos de teste*, que eles serão preenchidos automaticamente.

5. Testes com o software

Foram efetuados testes com o objetivo de verificar se o software conseguia acessar os bancos de dados, obter e organizar as informações e mostrar os resultados da análise. Ao cumprir tais requisitos, o **objetivo 4** deste trabalho é alcançado.

Para os testes, foram utilizados alguns bancos de dados criados criados especificamente para isso, ilustrando diferentes cenários e avaliando se o resultado foi satisfatório. A tabela de resultados mostra as informações lado a lado, tornando rápida a observação dos valores comparados. Cada linha representa uma coluna da tabela 1 associada com uma coluna da tabela 2, com os detalhes da comparação e o percentual de similaridade. Caso não seja encontrada uma coluna similar o suficiente para ser agrupada na mesma linha, a coluna aparecerá sozinha na linha, para as duas tabelas comparadas. As colunas `name1` e `name2` correspondem, respectivamente, ao nome da coluna da tabela 1 e o nome da coluna da tabela 2. De forma semelhante, as colunas `type1` e `type2` indicam o tipo de dados na tabela 1 e tabela 2. `size1` e `size2` indicam o tamanho da String, caso o tipo seja string. Caso contrário, mostra `0`. `pk1` e `pk2` indicam se o campo é chave primária. `nullable1` e `nullable2` indicam se o campo pode estar vazio. `updatable1` e `updatable2` indicam se é permitido alterações. `fk1` e `fk2` indicam se o campo é chave estrangeira. Por fim, a coluna `simScore` mostra a porcentagem de similaridade entre as colunas das duas tabelas. Na Figura 5 ocorre a comparação entre a tabela `medico` do banco de dados `clinica` e a tabela `medico` do banco `consultorio`. São duas tabelas semelhantes, e espera-se que haja semelhança entre as colunas das duas tabelas. Podemos observar que os campos que há linhas grifadas em verde (5), linhas em laranja (3) e linhas em vermelho (3). Podemos rapidamente notar que os campos `codmedico` e `bairro` da primeira tabela não possuem correspondente na segunda tabela, da mesma forma que não há correspondente para a coluna `cpf` da segunda tabela. Algumas colunas, em laranja, possuem leves diferenças, indicando a necessidade de tratar os dados na integração. Já as colunas em verde possuem grande semelhança, indicando que são praticamente idênticas, dispensando tratamento especial em algus casos, como no campo `endereco`, por exemplo, que possui *simScore* de 100%.

Figure 5. Testes com o software (medico/medico)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	crm	smallint	integer	0	0	NO	YES	NO	NO	YES	YES	NO	NO	82.5%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	especialidade	varchar	varchar	20	20	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtcontrato	datacontrato	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	82.0%
telefone	telefone	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Figure 6. Testes com o software(medico/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codmedico	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	---	0.0%
crm	---	smallint	---	0	---	NO	---	NO	---	YES	---	NO	---	0.0%
nomemed	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
bairro	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
especialidade	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
dtcontrato	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
telefone	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	idpaciente	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	rg	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	datanasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	convenio	---	varchar	---	15	---	NO	---	YES	---	YES	---	NO	0.0%

A Figura 6 representa outro cenário, onde as tabelas são diferentes, mas possuem colunas que armazenam informações semelhantes. Podemos observar que algumas colunas são mostradas em verde, uma delas em laranja e um número maior delas em vermelho. Isso já era esperado, por se tratarem de duas tabelas diferentes, a tabela `medico` e a tabela `paciente`. No terceiro cenário, comparamos uma outra tabela medico com ela mesma. Duas tabelas iguais devem gerar o resultado de 100% de similaridade, e é exatamente isso que acontece. Na Figura 7 observamos todas as linhas em verde e com `simScore` igual a 100%. Um cenário alternativo é mostrado da Figura 8 onde, embora as tabelas tenham o mesmo nome, as colunas são bem diferentes. Apenas uma coluna aparece em verde e uma em laranja, enquanto todas as outras estão em vermelho. Também podemos observar outra diferença nesta comparação em relação às anteriores, que é a associação de duas colunas diferentes, `cod_pessoa` e `nomepess`. Embora estejam associadas, são mostradas em vermelho, indicando que ambas dificilmente armazenam a mesma informação, por terem similaridade muito baixa, de apenas 30.5%. Com estes testes, criamos vários possíveis

Figure 7. Testes com o software (medico)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
crm	crm	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	100.0%
nome	nome	varchar	varchar	30	30	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
idade	idade	integer	integer	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
codcidade	codcidade	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%
codesp	codesp	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%
numeroa	numeroa	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	100.0%

Figure 8. Testes com o software (pessoa/pessoa)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
cod_pessoa	nomepess	integer	varchar	0	40	YES	NO	NO	YES	YES	YES	NO	NO	30.5%
sexo	sexo	character	character	2	1	NO	NO	YES	YES	YES	YES	NO	NO	87.5%
nome	---	varchar	---	50	---	NO	---	YES	---	YES	---	NO	---	0.0%
sobrenome	---	varchar	---	50	---	NO	---	YES	---	YES	---	NO	---	0.0%
data_de_nascimento	datanasc	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	70.0%
codigo_local_nasc	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
data_de_falecimento	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codigo_local_falec	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
codigo_uniao_pais	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
---	cic	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	conjugecic	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	0.0%

Figure 9. Testes com o software(consulta/consulta)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
crm	---	integer	---	0	---	YES	---	NO	---	YES	---	YES	---	0.0%
rg	---	integer	---	0	---	YES	---	NO	---	YES	---	YES	---	0.0%
data	data	date	date	0	0	YES	YES	NO	NO	YES	YES	NO	NO	100.0%
hora	hora	integer	time without time zone	0	0	NO	YES	YES	NO	YES	YES	NO	NO	68.0%
coddoenca	codpac	integer	integer	0	0	NO	YES	YES	NO	YES	YES	YES	YES	51.5%
---	codmed	---	integer	---	0	---	NO	---	NO	---	YES	---	YES	0.0%
---	valor	---	numeric	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	codconv	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	0.0%

cenários de comparação, observando que os resultados fornecidos pelo SimDBS são satisfatórios em relação ao que se esperava. Nas Figuras 9, 10, 11, 12, 13 e 14 são mostrados outros testes realizados com outros bancos de dados, também com resultados satisfatórios. Com o resultado destes testes, o objetivo 4 foi cumprido.

6. Conclusão sobre o trabalho atual

Com a demanda cada vez maior de sistemas mais robustos para gerenciamento de informações, a integração de dados torna-se imprescindível. Com o desenvolvimento do SimDBS, espera-se contribuir na tomada de decisão por parte dos Administradores de Bancos de Dados no processo de integração de dados. A comparação entre tabelas correlatas em bancos de dados diferentes é uma das etapas mais importantes para verificar quais as necessidades de ajuste nos dados para integrá-los, ou até mesmo para verificar se uma integração entre as tabelas é viável. Através dos testes, verificamos que o processo de análise é muito simples, bastando informar os dados de acesso ao banco e selecionar as tabelas a serem comparadas. Isso poupa um tempo considerável do DBA, que não precisa mais obter os dados manualmente para comparar, bastando inserir os dados e clicar em um botão. O processo que levaria um longo tempo, agora pode ser executado em cerca de um ou dois minutos.

Figure 10. Testes com o software (consulta/consulta)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codcons	codconsulta	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	85.0%
dtcons	datconsulta	date	date	0	0	NO	NO	NO	NO	YES	YES	NO	NO	72.0%
horacons	horaconsulta	time without time zone	time without time zone	0	0	NO	NO	YES	YES	YES	YES	NO	NO	86.7%
codmed	crmmedico	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	67.7%
codpac	codpaciente	integer	integer	0	0	NO	NO	YES	YES	YES	YES	YES	YES	80.0%
sala	nrsala	character	character	6	6	NO	NO	YES	YES	YES	YES	NO	NO	85.0%
valorcons	valorconsulta	money	money	0	0	NO	NO	YES	YES	YES	YES	NO	NO	88.0%

Figure 11. Testes com o software(consulta/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codcons	convenio	integer	varchar	0	15	YES	NO	NO	YES	YES	YES	NO	NO	26.5%
dtcons	---	date	---	0	---	NO	---	NO	---	YES	---	NO	---	0.0%
horacons	---	time without time zone	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codmed	---	integer	---	0	---	NO	---	YES	---	YES	---	YES	---	0.0%
codpac	idpaciente	integer	integer	0	0	NO	YES	YES	NO	YES	YES	YES	NO	54.7%
sala	---	character	---	6	---	NO	---	YES	---	YES	---	NO	---	0.0%
valorcons	---	money	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	nome	---	varchar	---	45	---	NO	---	NO	---	YES	---	NO	0.0%
---	rg	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	endereco	---	varchar	---	80	---	NO	---	YES	---	YES	---	NO	0.0%
---	cidade	---	varchar	---	25	---	NO	---	YES	---	YES	---	NO	0.0%
---	estado	---	character	---	2	---	NO	---	YES	---	YES	---	NO	0.0%
---	datanasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	0.0%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Figure 12. Testes com o software (funcionario/professor)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
rg	rg	integer	integer	0	0	YES	NO	NO	NO	YES	YES	NO	NO	95.0%
nome	nome	varchar	varchar	30	40	NO	NO	YES	YES	YES	YES	NO	NO	93.8%
idade	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
codcidade	codcid	integer	integer	0	0	NO	NO	YES	NO	YES	YES	YES	YES	84.2%
salario	---	double precision	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	codigo	---	integer	---	0	---	YES	---	NO	---	YES	---	NO	0.0%
---	sexo	---	character	---	1	---	NO	---	YES	---	YES	---	NO	0.0%

Figure 13. Testes com o software(paciente/funcionario)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codpac	codcidade	integer	integer	0	0	YES	NO	NO	YES	YES	YES	NO	YES	47.5%
nomepac	nome	varchar	varchar	40	30	NO	NO	NO	YES	YES	YES	NO	NO	71.8%
rg	rg	integer	integer	0	0	NO	YES	YES	NO	YES	YES	NO	NO	93.0%
cpf	---	bigint	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
endereco	---	varchar	---	80	---	NO	---	YES	---	YES	---	NO	---	0.0%
cidade	idade	varchar	integer	30	0	NO	NO	YES	YES	YES	YES	NO	NO	68.3%
estado	---	character	---	2	---	NO	---	YES	---	YES	---	NO	---	0.0%
dtnasc	---	date	---	0	---	NO	---	YES	---	YES	---	NO	---	0.0%
convenio	---	varchar	---	20	---	NO	---	YES	---	YES	---	NO	---	0.0%
---	salario	---	double precision	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

Figure 14. Testes com o software (paciente/paciente)

Name1	Name2	Type1	Type2	Size1	Size2	PK1	PK2	Nullable1	Nullable2	Updatable1	Updatable2	FK1	FK2	simScore
codpac	idpaciente	integer	integer	0	0	YES	YES	NO	NO	YES	YES	NO	NO	65.7%
nomepac	nome	varchar	varchar	40	45	NO	NO	NO	NO	YES	YES	NO	NO	77.2%
rg	rg	integer	integer	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
cpf	cpf	bigint	bigint	0	0	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
endereco	endereco	varchar	varchar	80	80	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
cidade	cidade	varchar	varchar	30	25	NO	NO	YES	YES	YES	YES	NO	NO	95.8%
estado	estado	character	character	2	2	NO	NO	YES	YES	YES	YES	NO	NO	100.0%
dtnasc	datanasc	date	date	0	0	NO	NO	YES	YES	YES	YES	NO	NO	70.0%
convenio	convenio	varchar	varchar	20	15	NO	NO	YES	YES	YES	YES	NO	NO	93.8%
---	cadsus	---	integer	---	0	---	NO	---	YES	---	YES	---	NO	0.0%

7. Atividades futuras

Ao longo do desenvolvimento, surgiram novas idéias que poderiam ser integradas ao SimDBS, tornando-o mais completo e ainda mais efetivo, mas que não puderam ser elaborados neste momento. Durante os testes de usabilidade, alguns usuários sugeriram algumas melhorias também, onde algumas foram aplicadas, e outras, mais trabalhosas, se tornaram idéias para novos trabalhos. Algumas sugestões de trabalhos que podem partir deste como base são:

- Automatizar ainda mais o SimDBS, fazendo a análise completa do banco de dados, buscando tabelas correlatas em bancos de dados diferentes e mostrando as comparações de todas as tabelas.
- Expandir o back-end, permitindo o processamento de dados de outros BDs além do PostgreSQL.
- Permitir a entrada de outros tipos de dados para representar o banco, como os comando SQL de criação ou uma entrada JSON, à escolha do usuário.
- Um *Dashboard* mostrando os principais pontos da comparação e informações importantes que auxiliem o DBA na tomada de decisão mais rápida.
- Criar uma comparação mais robusta, buscando considerar mais metadados na comparação.
- Automatizar a definição de pesos dos metadados.

8. Bibliografia

References

[Chen 2012] Chen, H. (2012). String metrics and word similarity applied to information retrieval. Master's thesis, Itä-Suomen yliopisto.

[Digital] Digital, I. M. Visualização da estrutura de uma tabela.

[Monteiro et al. 2007] Monteiro, J. M., Lifschitz, S., and Brayner, (2007). Extraindo metadados de sgbds.

[Rosa 2022] Rosa, A. C. I. d. (2022). Interface para visualização de dados brutos vindos de diferentes fontes.

Anexos

ANEXO A – CÓDIGO-FONTE DESENVOLVIDO

A.1 CÓDIGO FONTE DO SIMDBS

```
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width,
           initial-scale=1.0">

        <link rel="stylesheet" href="./css/style.css">
        <link rel="shortcut icon" href="./favicon.ico"
           type="image/x-icon" />
        <script src="./js/front.js"></script>
        <script
           src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.
        <script type="module"
           src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm.js
        <script nomodule
           src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js"></
        <title>SimDBS - Similarity DataBase Structure
           analyzer</title>
</head>

<body>
    <script>
        $(document).ready(function() {
            const host1 = "tcc-data.postgres.database.azure.com";
            const dbname1 = "clinica";
            const dbuser1 = "marcos";
            const dbpass1 = "JPsiqKsGTcvmW4w";
            const table1 = "medico"
            const host2 = "tcc-data.postgres.database.azure.com";
            const dbname2 = "consultorio";
            const dbuser2 = "marcos";
            const dbpass2 = "JPsiqKsGTcvmW4w";
            const table2 = "medico"

            $("#autofill").click(function(){
```

```
33              $("#host1").val(host1);
34              $("#dbname1").val(dbname1);
35              $("#dbuser1").val(dbuser1);
36              $("#dbpass1").val(dbpass1);
37              $("#host2").val(host2);
38              $("#dbname2").val(dbname2);
39              $("#dbuser2").val(dbuser2);
40              $("#dbpass2").val(dbpass2);
41              $("#table1").val(table1);
42              $("#table2").val(table2);
43          });
44      });
45      </script>
46      <div id="root">
47
48          <div class="header">
49              <img id="logo" src="./img/logo.png" />
50              <div class="btn btn-primary tooltip"><ion-icon
                    name="alert-circle-outline"></ion-icon>
51                  <div class="bottom">
52                      <h3>Como utilizar</h3>
53                      <p>Preencha os dados de acesso ao banco de dados.
54                          Em seguida, clique em listar tabelas, para
                                listar as tabelas disponiveis no BD para
                                comparar.
55                          Selecione a tabela que deseja utilizar para a
                                comparacao.
56                          Apos preencher os dados dos dois BDs, clique
                                em Analyze.
57                          O resultado sera mostrado abaixo do botao
                                (pode levar alguns segundos dependendo da
                                conexao com o banco de dados.)</p>
58                          <p>Clicando no botao <strong>BDs de
                                teste</strong> ao lado, os dados serao
                                preenchidos com o acesso a um banco de
                                testes.</p>
59                          <p>O botao <strong>Limpar</strong> limpa o
                                formulario e recarrega a pagina.</p>
60                      <i></i>
61                  </div>
62              </div> <!-- Ajuda -->
63
```

```
64              <div>
65                  <button id="autofill">BDs de teste</button>
66                  <button id="clear"
                        onclick="location.reload(true)">Limpar</button>
67              </div>
68          </div>
69          <div class="line"></div>
70
71          <div class="content">
72              <!-- form 1 -->
73          <div class="insertdb">
74              <div class="db">
75                  <form class="form1">
76                      <fieldset>
77                          <h1>Database 1</h1>
78                          <label for="host1">Database host
                                address:</label><br />
79                          <input class="input-data" type="text"
                                id="host1" /><br />
80
81                          <label for="dbname1">Database
                                name:</label><br />
82                          <input class="input-data" type="text"
                                id="dbname1" /><br />
83
84                          <label for="dbuser1">Username:</label><br />
85                          <input class="input-data" type="text"
                                id="dbuser1" /><br />
86
87                          <label for="dbpass1">Password:</label><br />
88                          <input class="input-data" type="password"
                                id="dbpass1" /><br />
89
90                          <button class="botao" id="bt1" type="button"
                                onclick="getTables('tab01')">Listar
                                Tabelas</button><br />
91
92                          <div id="list-tab01"></div>
93
94                          <!-- <label for="table1">Table:</label><br />
95                          <input class="input-data" type="text"
                                id="table1" /><br />-->
```

```
96
97                      </fieldset>
98                  </form>
99              </div>
100         </div>
101         <!-- form 2 -->
102         <div class="insertdb">
103             <div class="db">
104                 <form class="form2">
105                     <fieldset>
106                         <h1>Database 2</h1>
107                         <label for="host2">Database host
                                address:</label><br />
108                         <input class="input-data" type="text"
                                id="host2" /><br />
109
110                         <label for="dbname2">Database
                                name:</label><br />
111                         <input class="input-data" type="text"
                                id="dbname2" /><br />
112
113                         <label for="dbuser2">Username:</label><br />
114                         <input class="input-data" type="text"
                                id="dbuser2" /><br />
115
116                         <label for="dbpass2">Password:</label><br />
117                         <input class="input-data" type="password"
                                id="dbpass2" /><br />
118
119                         <button class="botao" id="bt2" type="button"
                                onclick="getTables('tab02')">Listar
                                Tabelas</button><br />
120
121                         <div id="list-tab02"></div>
122
123                         <!--<label for="table2">Table:</label><br />
124                         <input class="input-data" type="text"
                                id="table2" /><br /> -->
125
126                     </fieldset>
127                 </form>
128             </div>
```

```
129         </div>
130
131
132     </div><!-- content -->
133     <div class="analyzeBut"><button class="botao" type="button"
            onclick="analyzeData()"> Analisar </button></div>
134     <div id="loading"></div>
135
136     <div id="result"></div>
137
138 </div><!-- root -->
139 <div class="line"></div>
140 <div id="footer">
141     <p>Desenvolvido por Marcos Andrei Dranka</p>
142     <p><strong>UFSC / CTC / INE</strong></p>
143 </div>
144
145 </body>
146
147 </html>
```

```
1  @import
       url('https://fonts.googleapis.com/css2?family=Dosis&family=Kanit&family=Merriwe
2  body {
3  margin: 0 auto;
4  background-color: #ddd;
5  font-family: Merriweather, -apple-system, BlinkMacSystemFont, 'Segoe
       UI', 'Roboto', 'Oxygen',
6    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
7    sans-serif;
8  -webkit-font-smoothing: antialiased;
9  -moz-osx-font-smoothing: grayscale;
10 scroll-behavior: smooth;
11 }
12
13 .root {
14     margin: 0;
15     padding: 0;
16     border: 0;
17   }
18
19 .header {
20   height: 80px;
21   background-color: #000/*#282c34*/;
22   width: 100%;
```

```
23    display: flex;
24    align-items: center;
25    justify-content: space-around;
26  }
27
28  #logo{
29    height: 50px;
30    width: auto;
31  }
32
33  .header.div{
34    color: white;
35  }
36
37  #autofill, #clear {
38    font-size: 12pt;
39    background-color: #fff;
40    border-radius: 18px;
41    padding: 7px 15px;
42    border: none;
43    cursor: pointer;
44  }
45
46  .line {
47    background-image: linear-gradient(320deg, #007cff, #33c0f7);
48    margin: 0;
49    padding: 0;
50    width: 100%;
51    height: 5px;
52  }
53
54  .input-data{
55    font-size: 14pt;
56    font-family: Dosis, sans-serif;
57    width: 18vw;
58    background-color: #FFF;
59    border-radius: 20px;
60    border: none;
61    padding: 5px 12px;
62  }
63
64  .content {
65    width: 100%;
66    display: inline-flex;
67    justify-content: center;
68  }
69
70  .insertdb{
```

```
71    padding: 20px 2vw;
72    width: 30vw;
73  }
74
75  .db{
76    text-align: center;
77    font-size: 15pt;
78  }
79
80  .form.fieldset > label, input {
81    margin: 10px;
82  }
83
84  #tab01, #tab02{
85    border: 2px solid #fff;
86    color: #009dff;
87    font-size: 14pt;
88    padding: 5px 12px;
89    border-radius: 20px;
90  }
91
92  /* Animacao de carregamento */
93  #loading {
94    width: 2rem;
95    height: 2rem;
96    border: 5px solid #fff;
97    border-top: 5px solid #25baff;
98    border-radius: 100%;
99    margin: auto;
100   visibility: hidden;
101   animation: spin 1s infinite linear;
102 }
103 #loading.display {
104   visibility: visible;
105 }
106 @keyframes spin {
107   from {
108       transform: rotate(0deg);
109   }
110   to {
111       transform: rotate(360deg);
112   }
113 }
114
115
116 #result {
117   padding-top: 15px;
118   display: grid;
```

```
119   justify-content: center;
120   align-items: center;
121   padding-bottom: 40px;
122 }
123
124 #label {
125   display: flex;
126   width: auto;
127   justify-content: flex-end;
128 }
129
130 #label > p {
131   font-weight: 900;
132   padding: 0 15px;
133 }
134
135 #table {
136   border: 8px solid;
137   border-image: linear-gradient(320deg, #007cff, #33c0f7) 1;
138 }
139
140 .resTable{
141   table-layout: auto;
142   max-width: 95vw;
143   border: none;
144   border-collapse: collapse;
145   font-size: 12.5pt;
146   background-color: #eee;
147 }
148
149 .tableHead{
150   border: 1px solid #000;
151   padding: 4px 10px;
152   font-weight: 900;
153 }
154
155 .red{
156   color: rgb(145, 0, 0)
157 }
158
159 .orange{
160   color: rgb(235, 94, 0)
161 }
162
163 .green{
164   color: rgb(0, 104, 0)
165 }
166
```

```
167
168 .par {
169   text-align: left;
170 }
171
172
173 .t_line:hover{
174   background-color: #bbb;
175 }
176
177 th {
178   text-align: right;
179 }
180
181 td {
182   border: 1px solid #666;
183   text-align: right;
184   padding: 3px 7px;
185 }
186
187 #legenda {
188   border: 2px solid #007cff;
189 }
190
191 .analyzeBut {
192   display: flex;
193   justify-content: center;
194   align-items: center;
195   padding: 5px 12px;
196 }
197
198 .botao{
199   font-size: 14pt;
200   border-radius: 20px;
201   background-image: linear-gradient(320deg, #007cff, #33c0f7);
202   padding: 7px 15px;
203   color: #FFF;
204   margin: 5px 0;
205   border: none;
206   cursor: pointer;
207 }
208
209
210 #footer{
211   font-size: 12pt;
212   color: #222;
213   background-color: #888;
214   align-items: center;
```

```
215     text-align: center;
216     width: 100%;
217     position: relative;
218     bottom: 0;
219     margin: 0;
220     padding: 15px 0;
221 }
222 
223 #footer > p{
224     margin: 0;
225 }
226 
227 /* Codigo para a tooltip */
228 .tooltip {
229     display:inline-block;
230     position:relative;
231     text-align:left;
232     color:#FFF;
233     font-size: 35pt;
234 }
235 
236 .tooltip .bottom {
237     min-width: 350px;
238     top:40px;
239     left:50%;
240     transform:translate(-50%, 0);
241     padding:10px 20px;
242     color:#000000;
243     background-color:#cacaca;
244     font-weight:normal;
245     font-size:13px;
246     border-radius:8px;
247     position:absolute;
248     z-index:99999999;
249     box-sizing:border-box;
250     box-shadow:0 1px 8px rgba(0, 0, 0, 0.5);
251     visibility:hidden; opacity:0; transition:opacity 0.8s;
252 }
253 
254 .tooltip:hover .bottom {
255     visibility:visible; opacity:1;
256 }
257 
258 .tooltip .bottom i {
259     position:absolute;
260     bottom:100%;
261     left:50%;
262     margin-left:-12px;
```

```
263   width:24px;
264   height:12px;
265   overflow:hidden;
266 }
267
268 .tooltip .bottom i::after {
269   content:'';
270   position:absolute;
271   width:12px;
272   height:12px;
273   left:50%;
274   transform:translate(-50%,50%) rotate(45deg);
275   background-color:#cacaca;
276   box-shadow:0 1px 8px rgba(0,0,0,0.5);
277 }
```

```
1  const getTables = async (tab) => {
2      let dbdata;
3      if (tab === 'tab01'){
4          dbdata = {
5              "host1": document.getElementById('host1').value,
6              "user1": document.getElementById('dbuser1').value,
7              "pass1": document.getElementById('dbpass1').value,
8              "db1": document.getElementById('dbname1').value,
9          }
10     } else if(tab === 'tab02'){
11         dbdata = {
12             "host2": document.getElementById('host2').value,
13             "user2": document.getElementById('dbuser2').value,
14             "pass2": document.getElementById('dbpass2').value,
15             "db2": document.getElementById('dbname2').value
16         };
17     };
18
19     const options = {
20         method: 'POST',
21         headers: {'Content-Type': 'application/json'},
22         body: `${JSON.stringify(dbdata)}`
23       };
24
25       fetch(`http://localhost:3001/${tab}`, options)
26         .then(response => response.json())
27         .then(response => listTables(response, tab))
28         .catch(err => console.error(err));
29 }
30
31 const listTables = async(res, tab) => {
```

```
32      let tables = res;
33      document.getElementById(`list-${tab}`).innerHTML = '';
34      let selector = `<select id="${tab}">`;
35      for (let i=0; i < tables.length; i++){
36          selector += `<option
                value="${tables[i].tabela}">${tables[i].tabela}</option>`;
37      }
38      document.getElementById(`list-${tab}`).innerHTML += selector +
            `</select>`
39  }
40
41  const analyzeData = async () => {
42      document.getElementById('result').innerHTML = '';
43      displayLoading();
44      let dbdata = {
45          "host1": document.getElementById('host1').value,
46          "user1": document.getElementById('dbuser1').value,
47          "pass1": document.getElementById('dbpass1').value,
48          "db1": document.getElementById('dbname1').value,
49          "table1": document.getElementById('tab01').value,
50
51          "host2": document.getElementById('host2').value,
52          "user2": document.getElementById('dbuser2').value,
53          "pass2": document.getElementById('dbpass2').value,
54          "db2": document.getElementById('dbname2').value,
55          "table2": document.getElementById('tab02').value
56      };
57
58      const options = {
59          method: 'POST',
60          headers: {'Content-Type': 'application/json'},
61          body: `${JSON.stringify(dbdata)}`
62        };
63        response = '';
64        await fetch(`http://localhost:3001/result`, options)
65          .then(response => response.json())
66          .then(response => listResult(response))
67          .catch(err => console.error(err));
68      hideLoading();
69  }
70
71  // Animacao de carregamento
72  function displayLoading() {
73      document.querySelector("#loading").classList.add("display");
74      // to stop loading after some time
75      setTimeout(() => {
76          document.querySelector("#loading").classList.remove("display");
77      }, 5000);
```

```
78  }
79
80  // hiding loading
81  function hideLoading() {
82      document.querySelector("#loading").classList.remove("display");
83  }
84
85  const listResult = async(resTable) => {
86      let result = resTable;
87      const label = `<div id="label">`+
88                      `<p class="green">&gt;85%</p>`+
89                      `<p class="orange">&gt;60% e &le;85%</p>`+
90                      `<p class="red">&le;60%</p>`+
91                      `</div>`;
92      let finalTable = label +`<div id="table">` +
93      `<table class="resTable">`+
94      `<thead>`+
95      `<tr>`+
96      `<th class="tableHead">Name1</th>`+
97      `<th class="tableHead par">Name2</th>`+
98      `<th class="tableHead">Type1</th>`+
99      `<th class="tableHead par">Type2</th>`+
100     `<th class="tableHead">Size1</th>`+
101     `<th class="tableHead par">Size2</th>`+
102     `<th class="tableHead">PK1</th>`+
103     `<th class="tableHead par">PK2</th>`+
104     `<th class="tableHead">Nullable1</th>`+
105     `<th class="tableHead par">Nullable2</th>`+
106     `<th class="tableHead">Updatable1</th>`+
107     `<th class="tableHead par">Updatable2</th>`+
108     `<th class="tableHead">FK1</th>`+
109     `<th class="tableHead par">FK2</th>`+
110     `<th class="tableHead">simScore</th>`+
111     `</tr>`+
112     `</thead>`+
113     `<tbody>`;
114     for (let i=0; i < result.length; i++){
115         let line;
116         if(result[i].simScore <= 60) {
117             line = `<tr class="t_line red"><td>${result[i].name1}</td>`;
118         } else if (result[i].simScore > 60 && result[i].simScore <= 85)
                {
119             line = `<tr class="t_line
                    orange"><td>${result[i].name1}</td>`;
120         } else if (result[i].simScore > 85) {
121             line = `<tr class="t_line
                    green"><td>${result[i].name1}</td>`;
122         };
```

```
123         line += `<td class="par">${result[i].name2}</td>` +
124                 `<td>${result[i].type1}</td>` +
125                 `<td class="par">${result[i].type2}</td>` +
126                 `<td>${result[i].size1}</td>` +
127                 `<td class="par">${result[i].size2}</td>` +
128                 `<td>${result[i].pk1}</td>` +
129                 `<td class="par">${result[i].pk2}</td>` +
130                 `<td>${result[i].nullable1}</td>` +
131                 `<td class="par">${result[i].nullable2}</td>` +
132                 `<td>${result[i].updatable1}</td>` +
133                 `<td class="par">${result[i].updatable2}</td>` +
134                 `<td>${result[i].fk1}</td>` +
135                 `<td class="par">${result[i].fk2}</td>` +
136                 `<td>${result[i].simScore.toFixed(1)}%</td></tr>`;
137         finalTable += line;
138     }
139     finalTable += `</tbody></table></div>`;
140     document.getElementById(`result`).innerHTML = finalTable;
141     document.querySelector('#footer').scrollIntoView();
142 }
```

```
1  const express = require('express');
2  const compare = require('./compare');
3
4  const router = express.Router();
5
6
7  router.post('/tab01', compare.getTable1);
8  router.post('/tab02', compare.getTable2);
9  router.post('/result', compare.analyze);
10
11 router.use((error, req, res, next) => {
12     // Bad Request Error
13     res.status(501)
14     res.json("Dados para acesso ao BD incorretos! Corrija")
15   })
16
17
18 module.exports = router;
```

```
1  const express = require('express');
2  const cors = require('cors');
3  const router = require('./router');
4
5  const app = express();
6
7  app.use(express.json());
```

```
app.use(cors());
app.use(router);


const PORT = 3001;
app.listen(PORT, () => console.log(`Servidor executando na porta ${PORT}`));
```

```
const sim = require('string-similarity');
const { Pool } = require('pg');




let con_bd1, con_bd2;
function pool1(data){
    con_bd1 = new Pool ({
        host: `${data.host1}`, //'db.bvaqcsjdajjffqekutvg.supabase.co',
        database: `${data.db1}`, //'clinica',
        user: `${data.user1}`, //'postgres',
        password: `${data.pass1}`, //'JPsiqKsGTcvmW4w',
        max: 20,
        idleTimeoutMillis: 2000,
        connectionTimeoutMillis: 15000,
    });
    //con_bd1.connect()
}

function pool2(data){
    con_bd2 = new Pool ({
        host: `${data.host2}`, //'db.bvaqcsjdajjffqekutvg.supabase.co',
        database: `${data.db2}`, //'consultorio',
        user: `${data.user2}`, //'postgres',
        password: `${data.pass2}`, //'JPsiqKsGTcvmW4w',
        max: 20,
        idleTimeoutMillis: 2000,
        connectionTimeoutMillis: 15000,
    });
    //con_bd2.connect();
}


```

```
34 let getTable1 = (async(req, res) => {
35     pool1(req.body);
36     const client = await con_bd1.connect();
37     const { rows } = await client.query(`SELECT tablename AS tabela
          FROM pg_catalog.pg_tables
38         WHERE schemaname NOT IN ('pg_catalog', 'information_schema',
              'pg_toast')
39         ORDER BY tablename`);
40     await client.release();
41     return res.status(200).send(rows);
42 });
43
44 let getTable2 = (async(req, res) => {
45     pool2(req.body);
46     const client = await con_bd2.connect();
47     const { rows } = await client.query(`SELECT tablename AS tabela
          FROM pg_catalog.pg_tables
48         WHERE schemaname NOT IN ('pg_catalog', 'information_schema',
              'pg_toast')
49         ORDER BY tablename`);
50     await client.release();
51     return res.status(200).send(rows);
52 });
53
54
55 let analyze = (async(req, res) => {
56
57     // Atributos tabela 01
58     let cn1 = []; // nome da coluna
59     let t1 = []; // tipo de dados
60     let s1 = []; // Tamanho da string, caso haja
61     let pk1 = []; // chave primaria
62     let n1 = []; // Nullable
63     let up1 = []; // Updatable
64     let fk1 = []; // Chave estrangeira
65     let r1 = []; // Restrict
66     let pkeys1 = []; // chaves primarias
67     let fkeys1 = []; // chaves estrangeiras
68
69     // Atributos tabela 02
70     let cn2 = []; // nome da coluna
71     let t2 = []; // tipo de dados
```

```
72      let s2 = []; // Tamanho da string, caso haja
73      let pk2 = []; // chave primaria
74      let n2 = []; // Nullable
75      let up2 = []; // Updatable
76      let fk2 = []; // Chave estrangeira
77      let r2 = []; // Restrict
78      let pkeys2 = []; // chaves primarias
79      let fkeys2 = []; // chaves estrangeiras
80
81      pool1(req.body);
82      pool2(req.body);
83
84      // Busca as colunas que sao chaves primarias e secundarias em
           todas as tabelas do banco e salva para comparacao
85      await getKeys(con_bd1, 'PRIMARY KEY', pkeys1);
86      await getKeys(con_bd1, 'FOREIGN KEY', fkeys1);
87      await getKeys(con_bd2, 'PRIMARY KEY', pkeys2);
88      await getKeys(con_bd2, 'FOREIGN KEY', fkeys2);
89
90      await getData(con_bd1, req.body.table1, cn1, pkeys1, pk1,
           fkeys1, fk1, t1, s1, n1, up1);
91      await getData(con_bd2, req.body.table2, cn2, pkeys2, pk2,
           fkeys2, fk2, t2, s2, n2, up2);
92
93
94      // cria lista de atributos da primeira tabela
95      let tab01 = [];
96      for (let i = 0; i < cn1.length; i++) {
97          tab01.push(new Attribute(cn1[i], t1[i], s1[i], pk1[i],
               n1[i], up1[i], fk1[i], 'NO', 'NO'))
98      }
99      // cria lista de atributos da segunda tabela
100     let tab02 = [];
101     for (let i = 0; i < cn2.length; i++) {
102         tab02.push(new Attribute(cn2[i], t2[i], s2[i], pk2[i],
               n2[i], up2[i], fk2[i], 'NO', 'NO'))
103     }
104     // Executa as funcoes e retorna a tabela com o resultado
105     const result = await simPercentCalc(buildTable(tab01, tab02));
106     con_bd1.end();
107     con_bd2.end();
108     return res.status(200).send(result);
```

```
109 });
110
111
112
113 class Attribute {
114     constructor(_name, _type, _size, _pk, _nullable, _updatable,
        _fk, _restrict){
115         this.name = _name;              // Nome da coluna
116         this.type = _type;              // Tipo de dados
117         this.size = _size;              // Tamanho da String (se for
            string)
118         this.pk = _pk;                  // E chave primaria?
119         this.nullable = _nullable;      // Pode estar vazio?
120         this.updatable = _updatable;    // Pode ser alterado?
121         this.fk = _fk;                  // E chave estrangeira?
122         this.restrict = _restrict;      // Possui restricao de
            preenchimento?
123         this.match = false              // Durante a analise,
            informa se a coluna ja foi associada com uma coluna da
            outra tabela
124     }
125 }
126
127 class Result {
128     constructor(_name1, _name2, _type1, _type2, _size1, _size2,
        _pk1, _pk2, _nullable1, _nullable2, _updatable1, _updatable2,
        _fk1, _fk2, _restrict1, _restrict2) {
129         this.name1 = _name1;
130         this.name2 = _name2;
131         this.type1 = _type1;
132         this.type2 = _type2;
133         this.size1 = _size1;
134         this.size2 = _size2;
135         this.pk1 = _pk1;
136         this.pk2 = _pk2;
137         this.nullable1 = _nullable1;
138         this.nullable2 = _nullable2;
139         this.updatable1 = _updatable1;
140         this.updatable2 = _updatable2;
141         this.fk1 = _fk1;
142         this.fk2 = _fk2;
143         this.restrict1 = _restrict1;
```

```
144         this.restrict2 = _restrict2;
145         this.simScore = 0
146     }
147 }
148
149 async function getKeys(conn_bd, tipo, keys){
150     let res = await conn_bd.query(`SELECT t.table_name as tabela,
            k.column_name as coluna
151                                     FROM
                                            information_schema.table_constraints
                                            t
152                                     JOIN
                                            information_schema.key_column_usage
                                            k
153                                     USING(constraint_name,table_schema,table_n
154                                     WHERE
                                            t.constraint_type='${tipo}'`);
155     for (let i = 0; i < res.rowCount; i++){keys.push({'tabela':
            res.rows[i].tabela, 'coluna': res.rows[i].coluna})};
156 }
157
158 // funcao que busca os dados no bd e salva para comparacao
159 async function getData(conn_bd, table, cn, pkeys, pk, fkeys, fk, t,
    s, n, up){
160         // nome das colunas e se e chave primaria ou estrangeira
161         res = await conn_bd.query(`SELECT column_name FROM
                information_schema.columns WHERE table_name = '${table}'
                ORDER BY ordinal_position ASC`);
162         for (let i = 0; i < res.rowCount; i++){
163             cn.push(res.rows[i].column_name)
164             for (let j = 0; j < pkeys.length; j++){
165                 if (table === pkeys[j].tabela &&
                        res.rows[i].column_name === pkeys[j].coluna){
166                     pk[i] = 'YES'
167                     break;
168                 } else {
169                     pk[i] = 'NO'
170                 }
171             }
172             for (let j = 0; j < fkeys.length; j++){
173                 if (table === fkeys[j].tabela &&
                        res.rows[i].column_name === fkeys[j].coluna){
```

```
174                     fk[i] = 'YES'
175                     break;
176                 } else {
177                     fk[i] = 'NO'
178                 }
179             }
180         };
181         // tipo de dados bd1
182         res = await conn_bd.query(`SELECT data_type FROM
               information_schema.columns WHERE table_name = '${table}'
               ORDER BY ordinal_position ASC`);
183         for (let i = 0; i < res.rowCount; i++){
184             if (res.rows[i].data_type === 'character varying') {
185                 t.push('varchar')
186             } else {
187                 t.push(res.rows[i].data_type)
188             }
189         };
190         // tamanho string bd1
191         res = await conn_bd.query(`SELECT character_maximum_length
               FROM information_schema.columns WHERE table_name =
               '${table}' ORDER BY ordinal_position ASC`);
192         for (let i = 0; i < res.rowCount; i++){
193             if (res.rows[i].character_maximum_length === null){
194                 s.push(0);
195             } else {
196                 s.push(parseInt(res.rows[i].character_maximum_length));
197             }
198         };
199
200         // nullable bd1
201         res = await conn_bd.query(`SELECT is_nullable FROM
               information_schema.columns WHERE table_name = '${table}'
               ORDER BY ordinal_position ASC`);
202         for (let i = 0; i < res.rowCount;
               i++){n.push(res.rows[i].is_nullable)};
203         // updatable bd1
204         res = await conn_bd.query(`SELECT is_updatable FROM
               information_schema.columns WHERE table_name = '${table}'
               ORDER BY ordinal_position ASC`);
205         for (let i = 0; i < res.rowCount;
               i++){up.push(res.rows[i].is_updatable)};
```

```
206 }
207
208 function buildTable(tab01, tab02) { // Constroi tabela comparativa
       dos atributos
209     let attribSim = []; // Lista com os dados a serem analisados
210     let bmi; // Indice da string mais similar
211     let simindex; // valor de similaridade
212     for (let i = 0; i < tab01.length; i++) {
213         let n2 = []; // lista de nomes de atributo da segunda tabela
214         for (let j = 0 ; j < tab02.length; j++){
215             n2.push(tab02[j].name);
216         }
217         bm = sim.findBestMatch(tab01[i].name, n2);
218         bmi = bm.bestMatchIndex;
219         simindex = sim.compareTwoStrings(tab01[i].name,
               tab02[bmi].name);
220         if (!tab02[bmi].match && simindex > 0.3){ // se nao foi
               listado ainda
221             attribSim.push(new Result(tab01[i].name,
                   tab02[bmi].name, tab01[i].type, tab02[bmi].type,
                   tab01[i].size, tab02[bmi].size,
222                 tab01[i].pk, tab02[bmi].pk, tab01[i].nullable,
                       tab02[bmi].nullable,
223                 tab01[i].updatable, tab02[bmi].updatable,
                       tab01[i].fk, tab02[bmi].fk,
224                 tab01[i].restrict, tab02[bmi].restrict));
225                 tab01[i].match = true; // marca como listado
226                 tab02[bmi].match = true;
227         } else {
228             attribSim.push(new Result(tab01[i].name, '---',
                   tab01[i].type, '---',
229                 tab01[i].size, '---', tab01[i].pk, '---',
                       tab01[i].nullable, '---',
230                 tab01[i].updatable, '---', tab01[i].fk, '---',
231                 tab01[i].restrict, '---'));
232                 tab01[i].match = true; // marca como listado
233         }
234     }
235     for (let i = 0; i < tab02.length; i++) {
236         if (!tab02[i].match) {
237             attribSim.push(new Result('---', tab02[i].name, '---',
                   tab02[i].type,
```

```
238                 '---', tab02[i].size, '---', tab02[i].pk, '---',
                        tab02[i].nullable,
239                 '---', tab02[i].updatable, '---', tab02[i].fk,
240                 '---', tab02[i].restrict));
241                 tab02[i].match = true; // marca como listado
242         }
243     }
244 return attribSim;
245 }
246
247 // Calcula semelhanca entre os tipos de dados e atributos
248 function typeSimilarity(t1, t2, s1 = 0, s2 = 0) { // t1, t2: tipo
       dado 1 e 2, s1, s2: tamanho para char ou varchar.
249     let score = 0;
250     if (t1 === t2 && s1 === s2){
251         score = 1;
252     } else {
253         switch (t1){
254             case 'smallint':
255                 if(t2 === ('integer' || 'serial')){
256                     score = 0.5;
257                 } else if (t2 === ('bigint' || 'decimal' ||
                        'numeric' || 'real' || 'double precision' ||
                        'bigserial')){
258                     score = 0.1;
259                 } break;
260             case 'integer':
261                 if(t2 === ('smallint' || 'bigint' || 'decimal' ||
                        'numeric' || 'real' || 'double precision' ||
                        'serial' || 'bigserial')){
262                     score = 0.22;
263                 } break;
264             case 'bigint':
265             case 'decimal':
266             case 'numeric':
267             case 'real':
268             case 'double precision':
269             case 'serial':
270             case 'bigserial':
271                 if(t2 === ('smallint' || 'integer' || 'bigint' ||
                        'decimal' || 'numeric' || 'real' || 'double
                        precision' || 'serial' || 'bigserial')){
```

```
272                     score = 0.15;
273                 } break;
274             case 'character':
275                 if(t2 === 'character'){
276                     score = Math.min(s1, s2) / Math.max(s1, s2);
277                 } else if (t2 === 'character varying') {
278                     score = 0.1;
279                 } break;
280             case 'varchar':
281                 if(t2 === 'varchar'){
282                     score = Math.min(s1, s2) / Math.max(s1, s2);
283                 } else if (t2 === 'character') {
284                     score = 0.1;
285                 } break;
286             case 'timestamp with time zone':
287                 if(t2 === 'timestamp without time zone'){
288                     score = 0.7
289                 } else if (t2 === 'time without time zone'){
290                     score = 0.22;
291                 } else if (t2 === 'time with time zone'){
292                     score = 0.35;
293                 } break;
294             case 'timestamp without time zone':
295                 if(t2 === 'timestamp with time zone'){
296                     score = 0.7;
297                 } else if(t2 === 'time without time zone'){
298                     score = 0.35;
299                 } else if (t2 === 'time with time zone'){
300                     score = 0.22;
301                 } break;
302             case 'time without time zone':
303                 if(t2 === 'timestamp with time zone'){
304                     score = 0.22;
305                 } else if(t2 === 'timestamp without time zone'){
306                     score = 0.35;
307                 } else if (t2 === 'time with time zone'){
308                     score = 0.7;
309                 } break;
310             case 'time with time zone':
311                 if(t2 === 'timestamp with time zone'){
312                     score = 0.35;
313                 } else if(t2 === 'timestamp without time zone'){
```

```
314                     score = 0.22;
315                 } else if (t2 === 'time without time zone'){
316                     score = 0.7;
317                 } break;
318         }
319     }
320     return score;
321 }
322
323
324 // Calculo do percentual de similaridade
325 function simPercentCalc(attribSim) {
326     let similarity = 0;
327     for (let i = 0; i < attribSim.length; i++) {
328         // Calcula similaridade do nome
329         similarity += (0.6 *
            (sim.compareTwoStrings(attribSim[i].name1,
            attribSim[i].name2)));
330         //calcula similaridade do tipo de dado
331         similarity += (0.25 * (typeSimilarity(attribSim[i].type1,
            attribSim[i].type2, attribSim[i].size1,
            attribSim[i].size2)));
332         // primary key
333         similarity += (0.05 * (attribSim[i].pk1 === attribSim[i].pk2
            ? 1 : 0));
334         // nullable
335         similarity += (0.02 * (attribSim[i].nullable1 ===
            attribSim[i].nullable2 ? 1 : 0));
336         // updatable
337         similarity += (0.02 * (attribSim[i].updatable1 ===
            attribSim[i].updatable2 ? 1 : 0));
338         // foreign key
339         similarity += (0.04 * (attribSim[i].fk1 === attribSim[i].fk2
            ? 1 : 0));
340         // restrict
341         similarity += (0.02 * (attribSim[i].restrict1 ===
            attribSim[i].restrict2 ? 1 : 0));
342         // Formatacao da saida para porcentagem
343         attribSim[i].simScore = (100 * similarity);
344         similarity = 0;
345     }
346
```

```
347     return attribSim;
348 }
349
350 module.exports = {
351     analyze,
352     getKeys,
353     getData,
354     simPercentCalc,
355     typeSimilarity,
356     buildTable,
357     Attribute,
358     Result,
359     getTable1,
360     getTable2
361 }
```

A.2 CÓDIGO SQL PARA CRIAÇÃO DAS TABELAS DE TESTE PADRÃO USADAS NO SIMDBS

```
1
2     CREATE TABLE paciente(
3     codPac SERIAL NOT NULL PRIMARY KEY,
4     nomePac varchar(40) NOT NULL,
5     rg int,
6     cpf bigint,
7     endereco varchar(80),
8     cidade varchar(30),
9     estado char(2),
10    dtNasc date,
11    convenio varchar(20)
12 );
13
14 CREATE TABLE medico(
15    codMedico SERIAL NOT NULL PRIMARY KEY,
16    crm smallint NOT NULL UNIQUE,
17    nomeMed varchar(40) NOT NULL,
18    endereco varchar(80),
19    bairro varchar(20),
20    cidade varchar(30),
21    estado char(2),
```

```
22      especialidade varchar(20),
23      dtContrato date,
24      telefone bigint
25  );
26
27  CREATE TABLE consulta(
28      codCons SERIAL NOT NULL PRIMARY KEY,
29      dtCons date NOT NULL,
30      horaCons time,
31      codMed int,
32      codPac int,
33      sala char(6),
34      valorCons money,
35      CONSTRAINT FK_medico FOREIGN KEY(codMed)
36                      REFERENCES medico(codMedico),
37      CONSTRAINT FK_paciente FOREIGN KEY(codPac)
38                      REFERENCES paciente(codPac)
39  );
```

```
1
2       CREATE TABLE paciente(
3       idPaciente SERIAL NOT NULL PRIMARY KEY,
4       nome varchar(45) NOT NULL,
5       rg int,
6       cpf bigint UNIQUE,
7       endereco varchar(80),
8       cidade varchar(25),
9       estado char(2),
10      dataNasc date,
11      cadSUS int UNIQUE,
12      convenio varchar(15)
13  );
14
15  CREATE TABLE medico(
16      crm int NOT NULL PRIMARY KEY,
17      nome varchar(45) NOT NULL,
18      endereco varchar(80),
19      cidade varchar(25),
20      estado char(2),
21      especialidade varchar(20),
```

```
22      dataContrato date,
23      telefone bigint,
24      cpf bigint UNIQUE
25  );
26
27  CREATE TABLE consulta(
28      codConsulta SERIAL NOT NULL PRIMARY KEY,
29      datConsulta date NOT NULL,
30      horaConsulta time,
31      codPaciente int,
32      crmMedico int,
33      nrSala char(6),
34      valorConsulta money,
35      CONSTRAINT FK_medico FOREIGN KEY(crmMedico)
36                      REFERENCES medico(crm),
37      CONSTRAINT FK_paciente FOREIGN KEY(codPaciente)
38                      REFERENCES paciente(idPaciente)
39  );
```