UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CAIO JOSÉ DOS SANTOS ARRUDA

SCHEMACHECK:
CORRETOR AUTOMÁTICO DE ESQUEMAS SQL

CAMPINA GRANDE - PB

2023

CAIO JOSÉ DOS SANTOS ARRUDA

SCHEMACHECK:
CORRETOR AUTOMÁTICO DE ESQUEMAS SQL

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Claudio Elízio Calazans Campelo.

CAMPINA GRANDE - PB

2023

CAIO JOSÉ DOS SANTOS ARRUDA

SCHEMACHECK:
CORRETOR AUTOMÁTICO DE ESQUEMAS SQL

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Claudio Elízio Calazans Campelo

Orientador – UASC/CEEI/UFCG

Professor Dr. Carlos Wilson Dantas de Almeida

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 14 de fevereiro de 2023.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)[1]

Fix and provide feedback on created SQL schemas by students of database courses can be a process long and arduous for teachers. This is due to the complexity schemas, which can be written in different ways, even with a roadmap, which contains the functional requirements on what should be implemented, requiring the teacher to have enough attention during schema correction to identify possible problems or errors in the student's scheme. Furthermore, the criteria for correcting schemes are usually quite specific, requiring the teacher to describe exactly what the student made a mistake in and what could have done better. Thus, it may be difficult for Database course teachers keep pace of delivery of the students' schemes and, consequently - to guarantee that each student receives feedback regarding the scheme created, No delays. In this work, a line application was developed command that aims to be a tool to help correct of the schemes created by the students, based on correction criteria to be defined by the teacher. The application is expected help database course professors with corrections of SQL schemas, making correction more automated and less manual.

[1] Caso seu artigo esteja em inglês, coloque aqui o resumo em português; caso esteja o artigo em português, coloque aqui o resumo em inglês.

SchemaCheck: Corretor automático de Esquemas SQL

Trabalho de Conclusão de Curso

Caio José dos Santos Arruda (Aluno), Cláudio Campelo (Orientador)
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba - Brasil

RESUMO

Corrigir e fornecer feedbacks a respeito dos esquemas SQL criados pelos alunos de cursos de banco de dados pode ser um processo longo e árduo para os professores. Isso se deve à complexidade dos esquemas, que podem ser escritos de diversas formas, mesmo com um roteiro, que contém os requisitos funcionais sobre o que deve ser implementado, exigindo que o professor tenha bastante atenção durante a correção do esquema para identificar possíveis problemas ou erros no esquema do aluno. Além disso, os critérios de correção dos esquemas costumam ser bastante específicos, exigindo que o professor descreva exatamente no que o aluno errou e o que poderia ter feito de melhor. Dessa forma, pode ser difícil para os professores de cursos de Banco de Dados acompanharem o ritmo de entrega dos esquemas dos alunos e, consequentemente - garantir que cada aluno receba um feedback a respeito do esquema criado, sem atrasos. Neste trabalho foi desenvolvido uma aplicação de linha de comando que visa ser uma ferramenta para auxiliar a correção dos esquemas criados pelos alunos, com base em critérios de correção à serem definidos pelo professor. Espera-se que a aplicação auxilie os professores de cursos de Banco de Dados nas correções de esquemas SQL, tornando-a correção mais automatizada e menos manual.

PALAVRAS-CHAVE

Schemas, Sql, Banco de Dados

Repositório

https://github.com/arrudacaio/schema-check

1 INTRODUÇÃO

Uma das habilidades fundamentais para qualquer pessoa que deseja trabalhar com Banco de dados é a criação de esquemas SQL, pois permite que voce construa um banco de dados do zero e os organize de forma lógica e eficiente para atender os requisitos solicitados.

Ao criar um esquema, será necessário definir as tabelas e colunas que serão incluídas no banco de dados, bem como as possíveis relações entre essas tabelas. Tudo isso é feito através de instruções SQL, como *CREATE TABLE* e *ALTER TABLE*. Além disso, é de extrema importância adicionar restrições de chave primária e estrangeira afim de garantir a integridade dos dados presentes no banco. O processo de criação de esquemas requer planejamento cuidadoso e com muita atenção aos detalhes do problema que se pretende resolver.

Durante o processo de ensino de banco de dados relacionais em cursos de tecnologia, é comum que os professores forneçam atividades de criação de esquemas SQL, a partir de requisitos funcionais, afim de capacitar o aluno na construção de bancos de dados relacionais. Entretanto é bastante comum que os professores enfrentem desafios relacionados à gestão de tempo e esforço para corrigir os esquemas dos alunos e fornecer feedback em tempo hábil, uma vez que é necessário avaliar, com bastante critério e atenção, o conhecimento e a habilidade de cada aluno na hora de construir o respectivo esquema, afim de garantir que o aluno tenha construído o esquema de forma lógica e eficiente.

Atualmente, há diversas ferramentas disponíveis para auxiliar no ensino de programação, como CodeAcademy[1], FreeCodeCamp[2] e Blockly[3]. Estas ferramentas são amplamente utilizadas por professores para ensinar aos seus alunos como programar. No entanto, quando se trata de ensinar banco de dados, os recursos disponíveis são escassos. Isso é preocupante, pois o conhecimento em banco de dados é fundamental para quem deseja trabalhar com aplicações de larga escala.

Felizmente, existem algumas ferramentas interessantes que podem ser utilizadas para ensinar banco de dados. Por exemplo, a ferramenta SQL-Tutor[2] que fornece exercícios interativos e feedbacks para ajudar os estudantes a aprender SQL. Outra ferramenta útil é a viSQLizer[3], que oferece uma interface gráfica para visualizar e detalhar consultas SQL, ajudando os alunos a desenvolver um modelo mental para compreender as consultas criadas. Além disso, existe o SQLCheck[1] que é uma ferramenta para ser aplicada em uma consulta SQL que irá identificar anti-padrões presentes na consulta e fornecerá um feedback a respeito do que foi encontrado, afim de garantir a eficiência da consulta. Embora essas ferramentas possam ser úteis, ainda há uma necessidade maior de recursos para ensinar banco de dados de forma eficiente. É importante que mais ferramentas sejam desenvolvidas para atender à crescente demanda no aprendizado de banco de dados.

O restante do artigo está estruturado da seguinte maneira. Na Seção 2, descrevemos a solução encontrada para resolver o problema descrito. Na Seção 3, apresentamos a arquitetura da aplicação e o fluxo da informação na ferramenta. Na Seção 4, discutimos a respeito de um estudo de caso para entendermos como a aplicação pode ser utilizada e suas possibilidades. Na Seção 5, discutimos sobre os possíveis trabalhos futuros e as limitações da solução encontrada.

[1] https://www.codecademy.com/
[2] https://www.freecodecamp.org/
[3] https://developers.google.com/blockly/

2 PROBLEMA E SOLUÇÃO

2.1 Esquemas SQL

Os esquemas SQL são uma importante ferramenta para auxiliar a gerenciar e organizar banco de dados. Eles te permitem a criação de tabelas e colunas, bem como a definição das relações existentes entre as tabelas do banco de dados. Além disso, também são utilizados para especificar os tipos de dados permitidos em cada coluna, afim de garantir a consistência e integridade dos dados armazenados. É através dos esquemas que podemos estabelecer regras de integridade que deverão ser seguidas para garantir a qualidade dos dados.

É importante destacar que os esquemas são amplamente utilizados nas empresas, pois é uma funcionalidade fundamental para o armazenamento e gerenciamento de grande quantidades de dados, possibilitando a análise e tomada de decisões baseadas em informações confiáveis e atualizadas. Portanto, o conhecimento a respeito de esquemas SQL é de fundamental importância para qualquer pessoa que trabalhe com banco de dados ou que precise manipular uma grande quantidade de dados.

2.2 Complexidade da Correção dos Esquemas

Durante o ensino de bancos de dados em cursos de computação, é comum que os professores atribuam aos seus alunos atividades de criação de esquemas SQL para que eles possam aprender a construir um banco de dados do zero, a partir de requisitos funcionais. No entanto, a correção dessas atividades pode ser um processo complexo e trabalhoso para os professores, pois cada aluno pode seguir uma estratégia diferente para implementar o seu esquema. Isso requer que os professores façam uma análise cuidadosa de cada esquema para garantir que ele atenda aos requisitos especificados na atividade. Além disso, pode ser necessário que os professores apliquem os esquemas em um sistema de gerenciamento de banco de dados para validar se eles estão corretos e não apresentam erros estruturais. Esse processo pode ser bastante árduo e custoso para os professores, pois exige muito tempo para realizar todas as validações de forma manual e com o cuidado necessário.

2.3 SchemaCheck

O SchemaCheck é uma ferramenta de linha de comando que possibilita a correção automatizada de um esquema SQL por um professor, baseado em um esquema gabarito que ele deve criar. O esquema gabarito é flexível e permite a validação de pontos similares, que talvez não sejam exatamente iguais. Isso é feito através de um comentário no esquema utilizado antes de cada instrução SQL, que define possíveis nomes para tabelas, chaves primárias ou estrangeiras. Para utilizar o SchemaCheck, o professor precisa fornecer o esquema gabarito e o esquema do aluno, e a aplicação é capaz de comparar os dois e identificar possíveis falhas na implementação do esquema do aluno. Além disso, o SchemaCheck é muito útil para garantir a qualidade do trabalho dos alunos e promover a aprendizagem de forma mais eficiente, fornecendo feedbacks sobre o esquema criado pelo aluno.

3 ARQUITETURA DA SOLUÇÃO

Esta seção fornece detalhes sobre a arquitetura e os fundamentos teóricos utilizados na criação da aplicação de *command-line* (CLI) para um melhor entendimento do projeto. Inicialmente, são fornecidos detalhes sobre as tecnologias utilizadas no projeto e, em seguida, descrevemos como funciona o fluxo da informação da aplicação. Após isto, é descrito como pode-se realizar a configuração de um esquema gabarito, afim de torná-lo mais flexível, para ajudar a encontrar possíveis similaridades com o esquema do aluno. Depois, é descrito como funciona o Algoritmo de Distância de Levenshtein[4], que utilizamos para realizar comparações de strings. E por fim dedicamos uma seção a explicar a consulta SQL fundamental para obtermos os metadados dos esquemas aplicados no banco de dados, é a partir dessa consulta que podemos realizar as devidas comparações em busca de semelhanças entre os esquemas.

3.1 Componentes do Sistema

O SchemaCheck é uma aplicação de *command-line* (CLI) implementada com tecnologias como Python[4], Docker[5], Postgres[6] e Makefile[7]. A lógica de negócio do sistema foi construída inteiramente em Python, a escolha se deu devido a a linguagem Python ter um ótimo ecossistema para lidar com banco de dados, além de ser simples e eficiente para a construção de scripts, propiciando facilidades no desenvolvimento do projeto. O Postgres foi escolhido como o Sistema de Gerenciamento de Banco de Dados por fornecer a infraestrutura necessária para aplicar os schemas SQL, além de permitir a coleta de metadados sobre o banco de dados. O Makefile foi utilizado para facilitar a execução de scripts da aplicação e torná-la mais fácil de usar. O Docker foi utilizado para facilitar a criação do ambiente da aplicação de forma isolada e para facilitar a execução em máquinas de terceiros. Na Figura 1, é possível ver como a arquitetura do projeto está organizada e como seus componentes se comunicam.

Primeiramente, para entendermos o fluxo da informação presente na Figura 1, o SchemaCheck exige que o usuário insira o esquema de correção na raíz do diretório da aplicação, com o nome *"correction.sql"* . Além disso, também é necessário inserir o esquema que deve ser corrigido, que deve estar com o nome *"lesson.sql"*, também na raíz do projeto. Dessa forma a aplicação saberá qual é o esquema base de correção e qual o esquema que precisa ser avaliado. Após ter os esquemas no lugar correto, a aplicação irá aplicá-los, de forma isolada, no banco de dados Postgres. Em seguida, ela fará uma consulta no banco de dados para obter informações sobre os metadados presentes nos esquemas. Com base nessas informações, o SchemaCheck utiliza o Algoritmo de Distância de Levenshtein[4] para realizar comparações entre os metadados dos dois esquemas e capturar possíveis similaridades. Ao final de sua execução, a aplicação fornece um relatório sobre o que foi encontrado e quaisquer possíveis erros presentes do esquema do aluno.

3.2 Configurações do Esquema Gabarito

Para criarmos um esquema gabarito flexível, precisamos adicionar uma configuração antes da instrução SQL para identificar possíveis

[4]https://www.python.org/
[5]https://www.docker.com/
[6]https://www.postgresql.org/
[7]https://www.gnu.org/software/make/

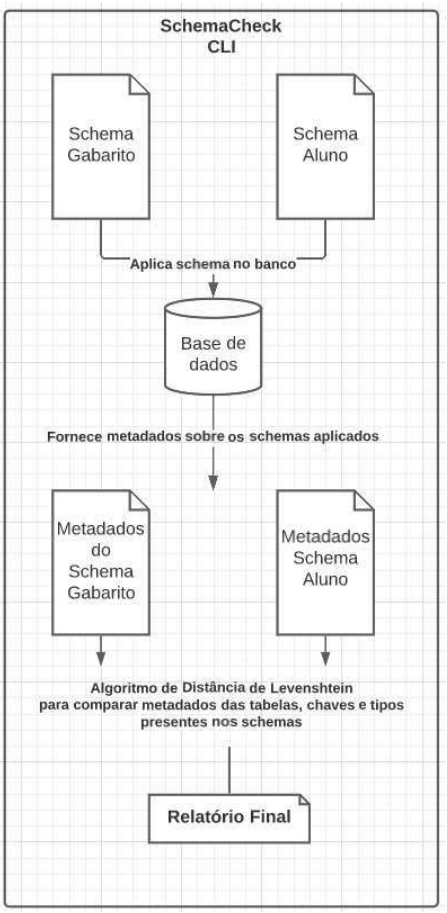

Figura 1: Arquitetura do SchemaCheck

semelhanças com o esquema do aluno. Para isso basta adicionar um bloco de comentário no arquivo SQL que representa o esquema base de correção. No SQL, o bloco de comentário inicia-se com a combinação de dois caracteceres */** e encerra-se com a mesma combinação, porém invertida ***/** . Dentro do bloco de comentário podemos adicionar os possíveis nomes da tabela que está sendo criada na instrução SQL, basta realizar a seguinte indicação *tableNames=['nome1', 'nome2']* e dentro dos colchetes descrevemos os nomes possíveis da tabela, dentro das aspas e separado por vírgula. O mesmo pode ser feito para definir os possíveis nomes e tipos das chaves primárias e estrangeiras (*pkNames*, *fkNames*, *pkTypes* e *fkTypes*) e também é possível especificarmos que a tabela pode ter sido construída com chaves compostas, basta utilizar a opção *pkHasCompose=True*. Toda essa configuração será adicionada aos metadados gerados pelo banco de dados, juntamente com os dados gerados pela aplicação do esquema base (gabarito), para a correção do esquema do aluno utilizando o Algoritmo de Distância de Levenshtein[4]. Abaixo temos um exemplo de como deve ser escrito as configurações do esquema gabarito.

```
[Exemplo de configuração para o esquema gabarito]
/*
tableNames=['dono_do_carro', 'owner', 'driver']
pkNames=['id', 'placa', 'rg']
pkTypes=['char','integer', 'bigint']
pkHasCompose=False
fkNames=['carro','moto']
fkTypes=['varchar', 'char']
*/
CREATE TABLE motorista
(
  cpf TEXT NOT NULL,
  PRIMARY KEY(cpf)
);
```

3.3 Comparação de Strings

O SchemaCheck compara os metadados do schema gabarito com o schema do aluno, para validar se existe semelhança entre os metadados. Para isso, faz uso do algoritmo de distância de Levenshtein[4] para medir a similaridade entre as strings (ou seja, as sequências de caracteres). Este algoritmo conta o número mínimo de operações (inserção, exclusão ou substituição) necessárias para transformar uma string em outra e é comumente utilizado em aplicações de processamento de linguagem natural. Na aplicação, foi estabelecido que se a distância entre duas strings for maior ou igual a 0,5 (considerando o intervalo de 0 a 1), essas strings podem ser consideradas similares.

3.4 Consulta aos metadados dos esquemas

Para identificarmos as semelhanças entre os esquemas, que foram aplicados em diferentes bases de dados, precisamos acessar os metadados de cada um deles. Isso inclui informações como nomes de tabelas, chaves e tipos das chaves. Com essas informações, podemos realizar uma comparação dos metadados de cada esquema, utilizando o algoritmo de distância de Levenshtein[4]. Desta forma, podemos encontrar similaridades entre os esquemas. Para isso foi utilizado a seguinte consulta SQL:

```
SELECT tco.constraint_type,
       kcu.table_name,
       tco.constraint_name,
       kcu.ordinal_position as position,
       kcu.column_name as key_column,
       ico.data_type as type
FROM information_schema.table_constraints tco
JOIN information_schema.key_column_usage kcu
     ON kcu.constraint_name = tco.constraint_name
     AND kcu.constraint_schema = tco.constraint_schema
     AND kcu.constraint_name = tco.constraint_name
INNER JOIN information_schema.columns ico
ON kcu.column_name = ico.column_name
WHERE
    (
      tco.constraint_type = 'PRIMARY KEY'
      or tco.constraint_type = 'FOREIGN KEY'
    )
AND kcu.table_schema = 'lesson'
ORDER BY kcu.table_schema,
         kcu.table_name,
         POSITION;
```

A *consulta SQL*, descrita na seção 3.4, seleciona informações sobre chaves primárias e estrangeiras em uma determinada base de dados. Ela faz isso unindo as tabelas *table constraints, key column usage* e *columns* do esquema, filtrando apenas as entradas cujo tipo de restrição seja PRIMARY KEY ou FOREIGN KEY. Depois, será retornado as informações como o tipo de restrição, o nome da tabela, o nome da restrição, a posição da chave na tabela, o nome da coluna chave e o tipo de dados da coluna. A consulta também irá ordenar os resultados pelo esquema, nome da tabela e posição da chave.

4 ESTUDO DE CASO

Esta seção apresenta um estudo de caso, que tem como objetivo demonstrar as possibilidades de utilização da aplicação *SchemaCheck*.

4.1 Cenário do Estudo de Caso

Vamos imaginar o seguinte cenário: O professor de um curso de banco de dados forneceu uma atividade de criação de esquemas SQL aos seus alunos, onde o principal objetivo é que o aluno crie um banco de dados do zero, para armazenar informações a respeito de uma locadora de carros. Para atender aos requisitos funcionais da atividade, é esperado que o aluno entregue ao professor um arquivo SQL,que contenha o esquema utilizado para a criação da base de dados. Com o esquema do aluno em mãos, o professor está apto para corrigir a atividade do aluno utilizando o *SchemaCheck*.

4.2 Utilizando o SchemaCheck

Para corrigir a atividade do aluno com o SchemaCheck, o professor deverá clonar o repositório da aplicação[1] e executar o comando *make build* para preparar e deixar a ferramenta pronta pra uso. Neste momento, o professor deverá inserir o arquivo do esquema do aluno na raíz do diretório da ferramenta, e deixá-lo com o nome *"lesson.sql"*. Em seguida está um exemplo de conteúdo de um esquema criado por um aluno:

```
[Esquema do Aluno]
CREATE TABLE motor
(
  id text not null,
  primary key(id)
);

CREATE TABLE car
(
  id text not null,
  placa char(7) not null,
  modelo varchar(20) not null,
  dono_do_carro text not null,
  primary key(placa),
  foreign key (dono_do_carro) references motor(id)
);
```

Feito isto, o professor deve criar um arquivo, também na raíz do projeto, mas com o nome *"correction.sql"* que será o arquivo que o professor deve descrever o seu esquema gabarito de correção. Em seguida está um exemplo de conteúdo contido no esquema gabarito do professor:

```
[Esquema do Professor]
/*
tableNames=['uber', 'dono_do_carro', 'owner', 'driver']
pkNames=['id', 'placa', 'rg']
pkTypes=['char','integer', 'bigint']
pkHasCompose=False
*/
CREATE TABLE motorista
(
  cpf TEXT NOT NULL,
  PRIMARY KEY(cpf)
);

/*
tableNames=['car', 'auto', 'vehicle', 'automovel']
pkNames=['chassi']
fkNames=['owner']
fkTypes=['varchar varying']
*/
CREATE TABLE carro
(
  id TEXT NOT NULL,
  placa CHAR(7) NOT NULL,
  modelo VARCHAR(20) NOT NULL,
  dono TEXT NOT NULL,
  PRIMARY KEY(placa),
  FOREIGN KEY(dono) REFERENCES motorista(cpf)
);
```

E uma vez que o esquema do aluno e o esquema gabarito estão presentes na raíz do projeto, é possível rodarmos a aplicação para realizar as validações, mas antes disso vamos entender a configuração utilizada pelo professor no esquema gabarito. Para entendermos, vamos utilizar a primeira instrução sql que o professor utilizou no esquema, neste caso foi o *CREATE TABLE motorista*, que vai criar uma tabela no banco de dados com nome *"motorista"*, no entanto foi especificado que a tabela *"motorista"* pode ter outros nomes no esquema do aluno, como fica explícito no campo *tableNames* no bloco de comentário que representa a configuração do esquema, fornecendo uma lista de possíveis nomes que podem estar representados no esquema do aluno, como *['car','auto', 'vehicle', 'automovel']*. Com essa lista de nomes possíveis para a tabela *"motorista"*, a aplicação irá realizar comparações com o esquema do aluno afim de encontrar tabelas com nomes similares aos da lista passada, utilizando o Algoritmo de Distância de Levenshtein[4]. O mesmo será feito para os nomes e tipos das chaves primárias e estrangeiras de uma tabela, com os campos *pkNames, fkNames, pkTypes, fkTypes*. Ao rodarmos a aplicação em um terminal, utilizando o comando *make start*, será gerado um relatório a respeito do que foi encontrado no esquema do aluno, como mostra a Figura 2. Portanto podemos analisar, com base na Figura 2 que foi encontrada uma tabela, de nome *motor* que provavelmente é similar com alguma tabela descrita no esquema de correção e a partir dessa similaridade de nome foi realizada outras validações. O relatório também descreve que na tabela *motor*, o nome da chave primária e o seu tipo estão provavelmente corretos e de acordo com o esperado. No entanto, é provável que os campos relacionados a chave composta e a chave estrangeira não estejam

de acordo com o esquema gabarito, necessitando uma validação manual por parte do professor. A mesma análise foi feita para a tabela *car*, que provavelmente está presente no esquema gabarito, em uma tabela com nome similar e nesse caso provavelmente o tipo da chave primária e a chave primária composta não estão de acordo com o esquema gabarito.

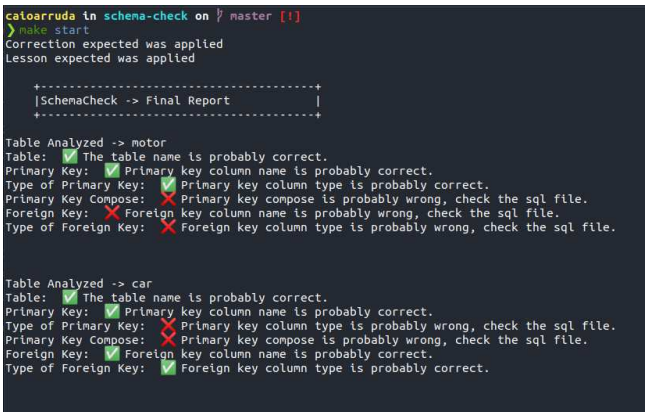

Figura 2: Relatório Gerado pelo SchemaCheck

5 DISCUSSÃO E TRABALHOS FUTUROS

Esta seção discute as limitações da aplicação desenvolvida e aponta para trabalhos futuros.

Este trabalho apresenta uma ferramenta útil para professores de cursos de banco de dados relacionais, pois permite um processo de correção automatizado e com poucas intervenções do professor. No entanto, existem algumas limitações na ferramenta que podem ser resolvidas para ampliar o alcance das correções. Uma dessas limitações é que a ferramenta depende fortemente do Algoritmo de distância de Levenshtein[4] para encontrar similaridades nos esquemas, o que pode levar à correção enviesada, já que só procura similaridades e não valida necessariamente as estruturas utilizadas para criar o esquema. Pois é possível que o aluno tenha definido o esquema corretamente, mas tenha nomeado as estruturas de forma incorreta, e dessa forma a aplicação não iria conseguir validar corretamente as estruturas. Além disso, o trabalho atual se limita a bancos de dados relacionais, com foco principal no SGBD Postgres. Um possível trabalho futuro seria expandir a aplicação para permitir a correção de esquemas de bancos de dados, seja relacionais ou não relacionais, como MongoDB[8], CassandraDB[9], entre outros.

O repositório da aplicação está hospedado na plataforma de código aberto, GitHub[10], onde qualquer pessoa pode contribuir para o projeto e trabalhar na sua evolução como uma ferramenta de código aberto. Além disso, qualquer pessoa pode sugerir novas funcionalidades ou possíveis melhorias.

[8]https://www.mongodb.com
[9]https://cassandra.apache.org/
[10]https://github.com/arrudacaio/schema-check

6 AGRADECIMENTOS

Gostaria de agradecer ao corpo docente do curso de Ciência da Computação da UFCG, pela passagem de conhecimento durante toda a graduação e pelos ensinamentos que levarei para a vida.

Aos meus pais, por todo o esforço que fizeram por mim e por todo o ambiente que me proveram para que eu sempre tivesse o foco nos meus estudos.

Aos meus amigos da graduação, que me ajudaram nos momentos mais difíceis do curso.

Ao professor Cláudio Campelo por ter sido um ótimo orientador e ter sido essencial na construção deste documento.

E por fim agradeço ao Eslen Delanogare, por me permitir vislumbrar todo o potencial que existia adormecido em mim.

O curso de ciência da computação na UFCG era o meu sonho e ele se concretizou. Nessa reta final posso dizer que este curso mudou minha vida, obrigado.

REFERÊNCIAS

[1] John Doe. 2012. SQLCheck: A Tool for Identifying Anti-patterns in SQL Queries. *ACM Transactions on Database Systems* 37, 4 (2012), 26–44.
[2] Jens Lechtenbörger and Jan Mendling. 2008. SQL-Tutor: A Tool for Teaching SQL. *International Journal of Information Systems Education* 7, 3 (2008), 43–56.
[3] J. Lehmann, P. Mayr, and J. Mendling. 2011. viSQLizer: Using visualization for learning SQL. *International Journal of Technology Enhanced Learning* 3, 4 (2011), 318–329.
[4] Wikipedia. 2022. Levenshtein distance. https://en.wikipedia.org/wiki/Levenshtein_distance