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

HIGOR SANTOS DE BRITO DANTAS

DBS:
FERRAMENTA PARA AUXILIAR NA EVOLUÇÃO DO MODELO DE DADOS COM INTEGRAÇÃO CONTÍNUA

CAMPINA GRANDE - PB

2022

HIGOR SANTOS DE BRITO DANTAS

DBS:
FERRAMENTA PARA AUXILIAR NA EVOLUÇÃO DO MODELO DE DADOS COM INTEGRAÇÃO CONTÍNUA

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. Adalberto Cajueiro de Farias.

CAMPINA GRANDE - PB

2022

HIGOR SANTOS DE BRITO DANTAS

DBS:
FERRAMENTA PARA AUXILIAR NA EVOLUÇÃO DO MODELO DE DADOS COM INTEGRAÇÃO CONTÍNUA

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. Adalberto Cajueiro de Farias
Orientador – UASC/CEEI/UFCG

Professora Dr. Jorge César Abrantes de Figueiredo
Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 06 de abril de 2022.

CAMPINA GRANDE - PB

ABSTRACT

The use of agile methodologies in software development brings several benefits, such as when you want to carry out the evolution of the same. However, the evolution of a software, in many cases, requires changes in the database, making management difficult. While this type of evolution is aided by tools, the necessary changes are created and analyzed manually. Therefore, developer negligence or the need to perform more complex operations end up making this activity susceptible to execution errors, and even database integrity. Soon, throughout this article, a tool will be developed that allows the management of changes in the database in a semi-automatic way. Where the product of this project will generate scripts to perform the desired changes in the state of the database. Therefore, it is expected that the evolution of software will become simpler and more efficient, with a reduction in errors related to the necessary changes in the databases.

DbS: Ferramenta para auxiliar na evolução do modelo de dados com Integração Contínua

Higor Santos de Brito Dantas
higor.dantas@ccc.ufcg.edu.br
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande

Adalberto Cajueiro de Farias
adalberto@computacao.ufcg.edu.br
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande

RESUMO

A utilização de metodologias ágeis no desenvolvimento de softwares traz diversos benefícios, como quando se deseja realizar a evolução do mesmo. Entretanto, a evolução de um sistema, em muitos casos, necessita de mudanças na base de dados dificultando o gerenciamento. Embora esse tipo de evolução seja auxiliada por ferramentas, as mudanças necessárias são criadas e analisadas manualmente. Portanto, uma negligência do desenvolvedor ou a necessidade de realizar operações mais complexas acabam tornando essa atividade passível a erros de execução, e até de integridade do banco de dados. Logo, ao longo deste trabalho será feito o desenvolvimento de uma ferramenta que permita o gerenciamento de modificações no banco de dados de maneira semiautomática. Onde o produto desse projeto irá gerar scripts para a realização das mudanças desejadas no estado do banco de dados. Sendo assim, é esperado que a evolução dos softwares se tornem mais simples e eficientes, com uma redução nos erros relacionados às alterações necessárias nos bancos de dados.

PALAVRAS-CHAVE

Banco de dados, metodologias ágeis, evolução, modelo de dados, PL/SQL.

REPOSITÓRIO

https://github.com/HigorSnt/dbs

1. INTRODUÇÃO

1.1 Contextualização

A manutenção de softwares, seja ela adaptativa, corretiva ou preventiva, é algo imprevisível e, principalmente, os modelos lineares não trazem a liberdade ideal para a realização dessas atividades. Isso porque, são métodos ideais para softwares com escopo fechado, algo bastante raro atualmente, onde as empresas costumam realizar adequações de acordo com feedbacks de clientes.

Desde que o manifesto ágil foi lançado, em 2001, o desenvolvimento de softwares começou a ser intensamente aplicado nesse novo contexto [11]. Com essa nova forma de entender e realizar o processo de desenvolvimento, manter e evoluir um software se tornou algo mais simples e fácil, se comparado com modelos antigos. Seguindo essa linha, Martin Fowler [22] fala que as metodologias ágeis possuem a premissa de serem adaptativas ao invés de serem predeterminantes, logo, são ideais para sistemas que não possuem um escopo fechado e podem sofrer modificações ao longo da sua vida útil.

Em grandes softwares, é comum irem surgindo novas necessidades ao longo do ciclo de vida, ou problemas inesperados, como bugs. Com as metodologias mais antigas, realizar manutenção adaptativa, por exemplo, é antagônico aos seus princípios, já que são métodos com ciclos bem definidos, onde uma fase só pode ser iniciada caso a anterior esteja totalmente concluída. Então, essas metodologias prezam por um software cujo cliente interessado sabe exatamente como o produto deve funcionar. Já as metodologias ágeis, são baseadas em iterações curtas, tornando um processo bem mais simples e adaptativo. Portanto, as metodologias ágeis surgiram com a volatilidade dos requisitos de software, como é dito por Florêncio e Santos [4].

Embora as metodologias ágeis tenham contribuído para melhorar o desenvolvimento de softwares, elas trazem consigo novos problemas, como as modificações em esquemas de banco de dados. Tais modificações são bastante frequentes, a ponto de serem executadas pelo menos 1 vez a cada semana [9] em empresas de grande porte. Isso pode significar instabilidades na base de dados com certa frequência, caso essas mudanças de estado no banco sejam feitas de maneira inadequada. Então, uma nova feature pode significar uma nova coluna, a criação ou deleção de tabelas, ou em modificações muito mais complexas a ponto de ser necessário o auxílio de DBAs (*Database Administrator*). E, para auxiliar surgiram muitas ferramentas e práticas, mas cada uma delas possui seus prós e contras.

Recentemente, o uso de *Continuous Integration* (CI) e *Continuous Deployment* (CD) em conjunto com técnicas de migrações tem se consolidado no processo de desenvolvimento. A prática da integração contínua (CI) dá ao desenvolvedor uma segurança de que as modificações realizadas não afetaram o funcionamento do sistema, isso graças à realização de testes automatizados. Já a implantação contínua (CD), permite que as modificações feitas cheguem aos clientes de forma automática após a passagem por uma série de processos pré-determinados, chamado *pipeline*. Por fim, a técnica de migrações aplica um versionamento ao esquema da aplicação, ou seja, cada modificação realizada será uma nova versão do banco de dados que fica geralmente registrado em uma tabela gerada pela ferramenta, contendo dados como hash do arquivo, autor, data, entre outros dados importantes para garantir a unicidade e integridade de cada arquivo.

As técnicas de CI e CD em conjunto com ferramentas específicas para a realização de migrações de banco de dados, permitem que o trabalho manual e a quantidade de problemas sejam reduzidos. Isso porque as migrações permitem quebrar

comandos SQL em pequenos pedaços, facilitando a manutenção e identificação de erros, sendo esses comandos executados por ferramentas que realizam o gerenciamento dessas alterações.

A utilização desses processos é extremamente útil, já que é comum a realização de testes de modo a garantir a integridade da aplicação após as alterações, mas ainda é passível de erros, por exemplo, um dos principais erros que ainda podem ocorrer é em relação à criação das migrações do banco de dados, já que é uma atividade feita manualmente. Daí, surge o objetivo deste trabalho: criar uma ferramenta que constrói arquivos de migração focados no SGBD Oracle, não sendo mais necessário o desenvolvedor gerar comandos SQL manualmente, apenas executá-lo, o que pode ser evitado caso utilize as técnicas CI e CD citadas.

1.2 Problema

A utilização de migrações para alterar o estado de um banco de dados vem sendo uma boa prática cuja adoção vem crescendo, assim como as ferramentas criadas para dar suporte a mesma. Porém, para realizar alterações em bancos de dados é necessário bastante cuidado e atenção, já que qualquer erro de digitação ou operação incorreta pode gerar perdas de dados, indisponibilidade ou até mesmo em falhas de segurança, dessa forma deixar que qualquer desenvolvedor possa realizar tais modificações traz consigo os riscos citados. Embora existam soluções possíveis, que serão discutidas abaixo, algumas trazem um custo associado contribuindo para um risco desnecessário.

1.2.1 Apenas DBAs realizam operações do tipo

Uma solução para problemas do tipo, é fazer com que essas operações sejam realizadas por pessoas especializadas. Entretanto, equipes pequenas nem sempre terão pessoas com foco apenas em operações nos bancos de dados, até pelo custo operacional de manter um funcionário com esse objetivo, cuja demanda não é constante. Além disso, ainda é mantido o problema de erros humanos na criação dos scripts de migração.

1.2.2 Cada desenvolvedor possuir uma instância do banco de dados

Outra solução, inclusive citada por Fowler [5], é fazer com que cada desenvolvedor possua uma instância da base de dados em sua máquina, ou até mesmo em máquinas na nuvem. Dessa forma, torna-se possível realizar testes antes de realizar a execução do script no banco de dados principal e reduzir a chance da ocorrência de erros. Entretanto, acaba sendo custoso a gerência de tantas instâncias de testes e a manutenção das instâncias dos bancos em nuvem.

1.2.3 Utilizar softwares de gerenciamento de bancos de dados

Existem muitos softwares, como o pgAdmin para PostgreSQL, Toad para Oracle, MySQL Workbench para MySQL, e outros aplicativos que dão suporte a mais de um SGBD, fornecendo aos usuários uma facilidade no gerenciamento dos esquemas com interfaces virtuais e atalhos para realizar certas operações. Esses aplicativos costumam auxiliar bastante e reduzir problemas por dar opções de operações automatizadas. Contudo, algumas destas ferramentas limitam o uso gratuito e oferecem opções mais amplas com preço alto, além de ainda não resolver totalmente o problema destacado. Então, o uso dessa prática pode trazer custo financeiro e ainda pode gerar problemas com erros manuais.

1.2.4 DbS: A ferramenta desenvolvida

Existem ainda muitas outras soluções possíveis, mas boa parte terão problemas semelhantes aos destacados anteriormente. Dessa maneira, a solução para o problema destacado é automatizar a geração de scripts, minimizando erros manuais. Portanto, o DbS (*Database Synchronizer*) surge com o objetivo de ser o núcleo de uma aplicação cujo frontend é quem interage diretamente com o desenvolvedor. É a partir dessa interação que a ferramenta proposta irá receber os dados necessários para realizar a geração do script PL/SQL a ser executado na instância que se deseja alterar o estado. Com isso, fica a cargo do desenvolvedor apenas a execução do mesmo. Entretanto, não é foco da ferramenta a geração de scripts de *rollback*, ou seja, a geração de scripts com comandos cujo objetivo é desfazer as alterações geradas.

1.3 Motivação

A ideia desta ferramenta surgiu a partir de uma demanda dentro de um projeto de Pesquisa e Desenvolvimento desenvolvido pela Universidade Federal de Campina Grande (UFCG) e a Fundação Parque Tecnológico da Paraíba (PaqTcPB) em parceria com a Dell, onde orientador e orientando estavam inseridos. Porém, com a evolução da ideia por parte da empresa parceira, essa ferramenta acabou se tornando desnecessária dentro do novo planejamento. Contudo, a ideia da ferramenta era bem desafiante e com funcionalidades interessantes fazendo com que a ideia de utilizá-la como tema deste trabalho fosse mantida.

2. ARQUITETURA DA FERRAMENTA

O DbS é uma biblioteca JavaScript escrita utilizando o TypeScript, que é apenas um superconjunto da linguagem com recursos adicionais, como tipagem [20]. A ferramenta desenvolvida tem como função principal a geração de scripts a partir da diferença entre os dois JSONs recebidos como parâmetros. Destarte, a biblioteca possui dois módulos com funcionalidades distintas e claras: um módulo responsável pela geração da diferença entre os JSONs passados como argumentos e outro módulo cuja responsabilidade é a geração dos scripts, este se comunica com aquele para que seja dado início à geração dos scripts.

2.1 Visão geral da ferramenta

O DbS surge para auxiliar em um fluxo na evolução de modelos de dados, cujo representação pode ser observada na Figura 1.

Figura 1. Fluxo da utilização da ferramenta durante o processo de desenvolvimento, com destaque para o papel da ferramenta a ser desenvolvida neste trabalho.

Inicialmente, um desenvolvedor realiza a edição ou a criação de um modelo, após o mesmo finalizar todas as alterações desejadas é preciso realizar a criação dos scripts SQL que irão atualizar os bancos de dados com a nova modelagem, nesse passo

é que a ferramenta desenvolvida atua, realizando a comparação entre o modelo antigo e o modificado. Por fim, o desenvolvedor apenas realiza a execução dos scripts resultantes do passo anterior.

2.2 Tecnologias utilizadas

O projeto foi desenvolvido utilizando TypeScript, como já citado. Essa definição foi feita baseando-se na ideia de que a mesma seria utilizada no projeto de pesquisa que estava sendo desenvolvido em parceria com a Dell. Então, como a ideia seria utilizar no front-end da aplicação construída em Angular [1], que utiliza TypeScript, o ideal seria utilizar a mesma tecnologia para facilitar a compatibilidade.

Outra tecnologia empregada foi o SQL, mais especificamente a extensão da linguagem o PL/SQL, desenvolvido e utilizado pelos bancos de dados da Oracle. Essa foi mais uma decisão de tecnologia com influência da empresa envolvida no desenvolvimento da ideia, já que os bancos de dados utilizados pela mesma costumam ser da Oracle.

Além disso, foram utilizadas algumas bibliotecas para realizar análise da qualidade de código e de formatação, como o ESLint [3] e o Prettier [17]. E ainda a biblioteca *sql-formatter* [18] para realizar a formatação dos comandos SQL gerados.

Por último, foi utilizado *jest* [6], que é um framework JavaScript bastante poderoso, com foco na construção de testes e com a capacidade de entregar dados relacionados à cobertura dos testes, um dos principais motivos pela escolha na utilização do mesmo.

2.3 Componentes do Sistema

O sistema é constituído por duas partes: a primeira, nomeada de *script-generation*, é o *kernel* da ferramenta já que é quem recebe os objetos JSON e realiza todo o processamento necessário para a geração dos scripts. A segunda parte, nomeada de *json-diff*, é responsável por analisar os objetos passados no início do processo. Portanto, há uma agregação entre as partes a fim de compor a cadeia de execução.

Na Figura 2, é possível observar, além do descrito anteriormente, que o *script-generation* possui um submódulo contendo alguns dados importantes relacionados às linguagens cuja ferramenta dá suporte e outro especializado na geração dos comandos para cada objeto do banco de dados.

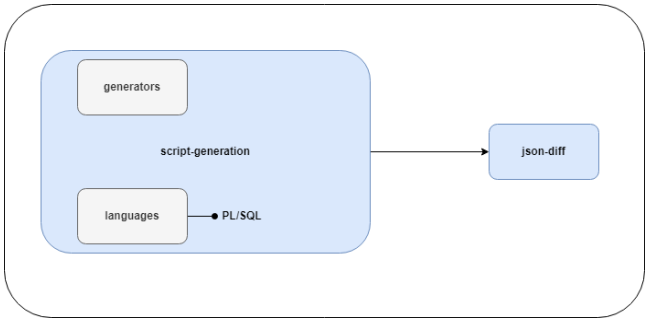


Figura 2. Representação da organização da ferramenta

2.3.1 Script-generation

Então, o objetivo central da ferramenta é a geração de scripts para que seja mais ágil a geração de atualizações dos modelos de dados. Dessa forma, é no módulo *script-generation* que se localizam todas as regras e informações necessárias para que esse objetivo seja cumprido.

Por ser uma biblioteca, o DbS deixa exposto apenas uma função de maneira *default* para que seja utilizada pelos projetos dependentes. Essa função é a porta de entrada para a geração do novo modelo de dados, recebendo algumas informações como parâmetros.

A primeira informação recebida pela função é o tipo de objeto do banco de dados que se deseja modificar ou criar, além disso, para evitar que seja recebido nomes inválidos ou não suportados é exposto um tipo com todas as opções que a ferramenta consegue interpretar. Atualmente, é dado suporte para as seguintes estruturas: *functions*, *packages*, *views*, *triggers*, *procedures*, *sequences* e *tables*.

Como segundo parâmetro, a função recebe o nome da sintaxe variante do SQL que é desejado como resultado final, atualmente, é suportado apenas o PL/SQL. Por fim, a função recebe um objeto representando o modelo desejado e outro objeto representado o modelo a ser alterado.

O primeiro passo a ser executado é passar os dados para a função responsável por tratar especificamente da estrutura do banco de dados recebida na entrada. Independentemente da estrutura a ser modificada, os passos seguintes são similares. É realizada a execução do *json-diff* que retorna todas as diferenças entre os objetos oriundos da entrada. Em seguida, é verificada cada diferença para identificar se é uma inserção ou deleção. Contudo, apesar do retorno da diferença entre os objetos indicar se um dado existe em ambos mas não são iguais, o *script-generation* ainda não é capaz de realizar operações de alteração entre os modelos, apenas criações e deleções.

Em seguida, o processamento é direcionado ao gerador específico da estrutura em questão do banco de dados, presente no submódulo *generators*. Dentro deste submódulo é feita a geração de cada comando utilizando dados fornecidos dentro do submódulo *languages*, que contém as palavras reservadas e templates de comandos específicos do SGDB selecionado no início do processo.

2.3.2 Json-diff

Durante o processo de geração dos comandos um passo necessário é a comparação entre os objetos de entrada a fim de identificar o que é preciso ser criado, alterado ou deletado para que os modelos se tornem equivalentes. Este importante papel é desempenhado pelo módulo *json-diff*.

O funcionamento deste pedaço do DbS é bem simples, já que contém apenas uma função cujos parâmetros são dois *arrays* de objetos que se deseja obter a diferenciação. Inicialmente, esses arrays são transformados em mapas contendo como chave a propriedade *name* e como valor o objeto convertido em texto. A propriedade *name* foi a selecionada para identificar cada JSON, pois, é uma propriedade comum a todos os objetos esperados, além de indicar o nome da estrutura no banco de dados, logo, ela deve ser necessariamente única por definição do próprio SQL.

Essa transformação em mapas acaba reduzindo a complexidade do algoritmo, já que busca em mapas são constantes com complexidade O(1).

Dessa forma, se faz necessário apenas verificar se no mapa contendo os objetos do modelo a ser alterado contém cada uma das chaves do mapa contendo os objetos do modelo fonte. Dessa forma, caso contenha e os valores não sejam iguais isso indica que houve alguma alteração. Em contrapartida, caso não

contenha é uma indicação que no modelo fonte esse objeto foi criado e que deverá ser criado no novo modelo.

Em seguida, é feita a operação contrária, ou seja, é verificado se o mapa contendo os objetos do modelo a ser alterado possui alguma chave cujo o mapa do modelo fonte não contenha, indicando a necessidade de uma remoção dessa estrutura do banco de dados.

Assim sendo, o resultado final do processamento feito nesse módulo é uma coleção de objetos contendo o valor da estrutura analisada e a indicação se o mesmo foi adicionado, deletado ou alterado, representado pela Figura 3.

```
export interface Diff {
  value: any;
  added?: boolean;
  changed?: boolean;
  removed?: boolean;
}
```

Figura 3. Modelo do retorno das diferenças encontradas pelo *json-diff*.

2.4 Fluxo de execução

Com o já exposto anteriormente, é possível imaginar o fluxo de execução da ferramenta. Porém, na Figura 4 este fluxo está melhor descrito em um gráfico.

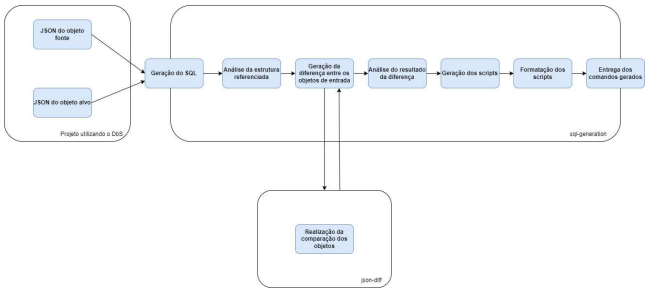


Figura 4. Fluxo de execução do DbS

Inicialmente, o projeto que estiver utilizando o DbS como dependência necessita passar alguns parâmetros entre eles duas coleções de objetos, o primeiro indicando o modelo de dados que se deseja como resultado final e o segundo representando o modelo de dados que se deseja alterar. Além dessas duas coleções, é recebido como parâmetro de entrada o tipo de estrutura em banco de dados que essas coleções representam e a sintaxe SQL do SGBD.

Em seguida, é feita a avaliação de que tipo de estrutura os dados repassados representam e ocorre a execução da função específica para a estrutura. Dentro dessa função especializada, ocorre uma comunicação com o módulo *json-diff* a fim de obter as diferenças entre os dois modelos.

O *json-diff* realiza a análise das diferenças existentes entre as duas entradas e responde a requisição do *sql-generation* com uma coleção de objetos, cujo formato está representado na Figura 3, indicando as diferenças existentes.

No passo seguinte, o módulo *sql-generation* realiza a análise de cada diferença e gera o comando necessário para a sincronização entre os modelos. Ademais, antes de entregar ao projeto cliente dependente do DbS os comandos de sincronização, cada um destes passa por uma formatação com o auxílio da biblioteca *sql-formatter* [18], que é uma dependência interna da ferramenta desenvolvida.

Na sequência, os comandos já estão gerados e formatados, sendo entregues ao cliente.

3. AVALIAÇÃO

Após a finalização da fase de desenvolvimento, deu-se início a fase de avaliação da ferramenta. Com o objetivo de verificar se a ferramenta estava gerando resultados de acordo ao esperado, foram gerados 3 cenários básicos para cada estrutura do banco de dados que a mesma dá suporte e feito uma análise da sintaxe dos comandos gerados com o auxílio da ferramenta Live SQL [15]. Para cada um dos cenários, foram gerados arquivos JSON com o formato do modelo esperado pela ferramenta.

3.1 Cenários

3.1.1 Cenário 1

O primeiro cenário de teste construído consiste em ter ao menos uma estrutura cadastrada no modelo fonte e nenhuma estrutura presente no modelo a ser alterado. Dessa forma, é esperado que o resultado para este cenário seja uma string com comandos de criação.

3.1.2 Cenário 2

No segundo cenário, o modelo fonte e o modelo que se deseja alterar contêm estruturas presentes no JSON representando cada uma delas. Entretanto, o objetivo é que o modelo a ser alterado passe a ser igual ao modelo fonte. Então, todas as estruturas presentes no modelo a ser alterado e que não estão presentes no modelo fonte devem ser deletadas, e criadas as estruturas que estão neste.

3.1.3 Cenário 3

O terceiro cenário de teste é oposta ao mostrado no cenário 1, ou seja, o modelo fonte não irá conter nenhuma estrutura referente cadastrada, enquanto o modelo que necessita de alterações sim. Portanto, todas as estruturas que estão presentes neste necessitam de remoção, sendo esperado uma string apenas com comandos de deleção das estruturas.

3.2 Testes realizados

3.2.1 Function

Para todos os testes realizados foram criadas algumas funções com certas características a fim de realizar testes com uma diversidade de dados. Na Tabela 1, estão descritos cada uma delas.

Nome	Característica
get_address	Contém 1 configuração de permissão, e recebe 1 parâmetro.

is_palindrome	Contém 1 parâmetro e nenhuma configuração de permissão.
fsum *fsub* *fdiv*	Contém 2 parâmetros, 2 configurações de permissões e está contida dentro de um *schema* chamado *math_functions*.

Tabela 1. Nomes e características das *functions* criadas para os testes

No primeiro cenário, o modelo fonte contém as funções *get_address*, *is_palindrome* e *fsum*. Enquanto no segundo cenário, o modelo fonte possui a definição das funções *get_address*, *is_palindrome*, *fsum* e *fdiv* e o modelo a ser modificado contém *get_address*, *fsum* e *fsub*. Por fim, no terceiro cenário o modelo que necessita ser alterado possui as funções *get_address*, *is_palindrome* e *fsum*.

3.2.2 Package

Para os testes relacionados à *package*, foram pensados dois modelos básicos que foram adaptados entre os cenários para realizar os testes.

Em relação ao primeiro cenário, o modelo fonte foi pensado contendo os packages *employee*, contendo uma configuração de permissão (*grant*), e *address*, sem nenhum *grant* e inserido à um schema. No segundo cenário, o modelo fonte contém o *package employee* e o modelo alvo de modificações possui o *package address*, nesse cenário, ambos com configurações de permissão. Já no terceiro cenário o modelo a ser modificado contém a mesma configuração do modelo alvo no primeiro cenário de teste.

3.2.3 Procedure

Semelhante ao realizado nos testes de *packages*, para *procedure* foram pensados dois modelos básicos e realizado adaptações entre os cenários para aumentar os casos cobertos.

No primeiro cenário, foi gerado uma *procedure* bem básica nomeada de *hello_world* e outra com nome *insert_user* contendo 2 parâmetros, 1 grant e inserido a um schema. Em seguida, no segundo cenário, o modelo alvo possui a *procedure hello_world* com 1 *grant* e no modelo a ser modificado com o procedimento *insert_user*, com as mesmas características já expostas. Por último, no terceiro cenário o modelo que necessita de alterações irá conter os mesmos *procedures* descritos no primeiro cenário para o modelo alvo.

3.2.4 Sequence

Para realizar testes sobre sequências foram criados alguns modelos, cujas características estão descritas na Tabela 2.

Nome	Características
id_seq	Não está inserido em nenhum *schema*, valor inicial é 1 e é incrementado com passo 1, possui como valor mínimo 1 e máximo 10000 e cache com tamanho 5. Para este caso, não existe geração de *grant*.
engineer_id_seq	Está inserido no schema company, com valor inicial igual a 800 e é decrementado com passo -1, não possui valor mínimo e possui valor máximo igual a 10000, sem realizar o armazenamento de cache. Para este caso, há uma configuração de segurança.

Tabela 2. Nomes e características das *sequences* criadas para os testes

Para o primeiro cenário de teste, o modelo fonte foi planejado contendo os dois objetos descritos na Tabela 2. Já no segundo cenário de teste, o modelo fonte contém o objeto *engineer_id_seq* e o modelo alvo de alterações contém a *sequence id_seq*. Por fim, no último cenário, o modelo a ser modificado contém os mesmos objetos que o definido para o primeiro cenário de teste no modelo fonte.

3.2.5 Table

A fim de realizar os testes direcionados à criação e deleção de tabelas, foram pensadas duas tabelas cujas características estão descritas na Tabela 3.

Nome	Características
employee	Esta tabela está inserida em um *schema*, contendo 3 propriedades, onde cada uma possui um comentário, 2 *constraints* (uma de chave primária e outra de chave estrangeira) e 1 index e 1 *grant*.
author	Esta tabela não possui nenhum *schema*, contém 3 propriedades, sem comentários e index, 1 *grant*.

Tabela 3. Nomes e características das *tables* criadas para os testes.

No primeiro e terceiro cenário de teste, ambas as tabelas são utilizadas, sendo neste utilizado para representar o modelo que necessita de alterações, enquanto naquele utilizado no modelo fonte. Já no segundo cenário de teste, o modelo fonte contém a tabela *employee* e o modelo alvo das alterações contém a tabela *author*.

3.2.6 Trigger

Nos testes de *trigger*, foram gerados dois modelos *user_updated_at_column* e *check_age*. Ambos os modelos possuem características semelhantes sendo a principal diferença que o modelo *check_age* está contido em um esquema e ele não pode ser substituído caso já exista algum *trigger* com mesmo nome.

Quanto aos testes, nos cenários 1 e 3 as entradas foram semelhantes aos casos já citados anteriormente, o modelo fonte contendo ambos os objetos no primeiro cenário, enquanto o

modelo alvo das alterações contendo ambos os objetos no terceiro cenário, indicando a necessidade de remoção dos mesmos. Já no segundo cenário, o modelo fonte contém o *trigger check_age* e o modelo a ser alterado contém o *trigger user_updated_at_column*, o que torna como resultado esperado a remoção do mesmo e a criação da estrutura presente no modelo alvo.

3.2.7 *View*

Nos testes focados em averiguar a validade das *views* foram criadas duas estruturas e seus detalhes estão descritos na Tabela 4.

Nome	Características
inventory	Não possui nenhum *schema* definido, realiza a seleção dos valores de 2 colunas presentes em uma tabela *products* e possui 1 configuração de segurança.
supplier_orders	Está inserida à um *schema*, seleciona os valores de 3 colunas, sendo necessário a realização de uma junção de tabelas e possui 1 configuração de segurança.

Tabela 4. Nomes e características das *views* criadas para os testes.

Seguindo o padrão descrito nos testes anteriores, nos cenários de testes 1 e 3 o modelo fonte contém ambas as estruturas da Tabela 4 no primeiro, e o modelo alvo das alterações ambas estruturas no 3º cenário. Já no segundo cenário, o modelo fonte contém a estrutura *inventory* e o modelo alvo contém a estrutura *supplier_orders*.

4. RESULTADOS

Com os cenários e os testes idealizados explicados, se faz necessário realizar uma análise dos resultados obtidos. Ademais, todos os testes criados podem ser visualizados no diretório *tests* do repositório da ferramenta. Dentro do mesmo, há o subdiretório *files* contendo cada um dos arquivos utilizados como entrada (*source.json* e *target.json*) e o resultado esperado, além de conter o *script* gerado como resultado do processo de geração dos comandos de sincronização dentro do diretório *out*.

Foram criados 21 testes, gerando uma cobertura de 99.79% das linhas de código da ferramenta e 86.45% de cobertura dos desvios que o código pode sofrer, como a presença de trechos condicionais. Ademais, houve a execução de 97.43% das funções criadas, o que é mostrado pela Figura 4.

Figura 4. Dados gerais da cobertura dos testes aplicados na ferramenta.

No módulo *json-diff*, foi obtido uma cobertura de 93.5% das linhas e quase 89% dos desvios que o código realiza, esse valor é satisfatório. Ademais, é importante destacar que ao analisar quais os casos cujo testes não cobriram, é possível observar que foram apenas nas linhas referentes à identificação de uma estrutura que possui o mesmo nome nos dois modelos, mas que não são exatamente iguais. E isso ocorre em virtude de a ferramenta não cobrir casos de mudanças em estruturas.

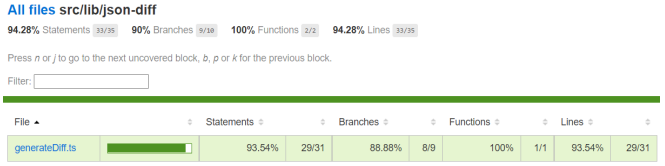


Figura 5. Dados de cobertura do módulo *json-diff*

Figura 6. Identificação dos trechos não cobertos pelos testes no módulo *json-diff*

Em relação ao módulo *generators*, foi obtido a cobertura de 100% das linhas e funções executadas, porém, uma taxa aproximada de 82.6% de cobertura nos desvios condicionais que os códigos possuem. Apesar de ser uma taxa adequada, como mostram os dados da Figura 8, é possível perceber que os testes sobre, principalmente, *triggers* e *packages* poderiam ter sido mais explorados e cobrir mais situações.

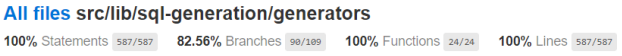
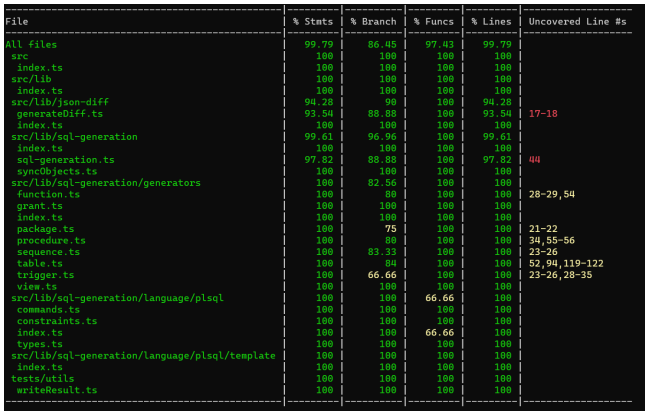


Figura 7. Cobertura geral dos testes realizados no módulo *generators*

Figura 8. Dados de cobertura sobre os geradores de comandos de cada estrutura suportada pela DbS

Por fim, todos os dados de cobertura de cada arquivo presente no código da ferramenta gerados pelo *jest* podem ser visualizados na Figura 9.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	99.79	86.45	97.43	99.79	
src	100	100	100	100	
index.ts	100	100	100	100	
src/lib	100	100	100	100	
index.ts	100	100	100	100	
src/lib/json-diff	94.28	90	100	94.28	
generateDiff.ts	93.54	88.88	100	93.54	17-18
index.ts	100	100	100	100	
src/lib/sql-generation	99.61	96.96	100	99.61	
index.ts	100	100	100	100	
sql-generation.ts	97.82	88.88	100	97.82	44
syncObjects.ts	100	100	100	100	
src/lib/sql-generation/generators	100	82.56	100	100	
function.ts	100	80	100	100	28-29,54
grant.ts	100	100	100	100	
index.ts	100	100	100	100	
package.ts	100	75	100	100	21-22
procedure.ts	100	80	100	100	34,55-56
sequence.ts	100	83.33	100	100	23-26
table.ts	100	84	100	100	52,94,119-122
trigger.ts	100	66.66	100	100	23-26,28-35
view.ts	100	100	100	100	
src/lib/sql-generation/language/plsql	100	100	66.66	100	
commands.ts	100	100	100	100	
constraints.ts	100	100	100	100	
index.ts	100	100	66.66	100	
types.ts	100	100	100	100	
src/lib/sql-generation/language/plsql/template	100	100	100	100	
index.ts	100	100	100	100	
tests/utils	100	100	100	100	
writeResult.ts	100	100	100	100	

Figura 9. Dados obtidos dos testes sobre todos os arquivos da ferramenta.

Apesar da cobertura de testes ser uma boa métrica que é utilizada com frequência por grandes empresas para realizar análises da qualidade dos mesmos, essa técnica não dá uma garantia total da corretude. Dessa forma, pensando em analisar se a sintaxe gerada dos scripts estava válida foi feito o uso da ferramenta disponibilizada pela Oracle, o Live SQL, para realizar essa verificação.

Entretanto, a ferramenta Live SQL possui algumas limitações nos comandos que podem ser executados, até por questões de seguranças do sistema da Oracle. Dessa forma, alguns comandos precisaram de adaptações para serem validados, por exemplo, não é possível a criação de usuários e *schemas* havendo a necessidade de remover o nome do *schema* em alguns comandos, assim como a remoção dos comandos de criação de *grants*.

Todos os comandos testados foram bem sucedidos, por exemplo, na Figura 10 é possível observar que foram criadas as duas tabelas que estão destacadas em azul e necessitou de realizar a criação da tabela *department* que se relaciona com a tabela *employee*. Enquanto que na Figura 11 está mostrando que as funções conseguiram ser criadas e na Figura 12 a criação de uma *view* realizando o join de duas tabelas que foram criadas para esse teste.

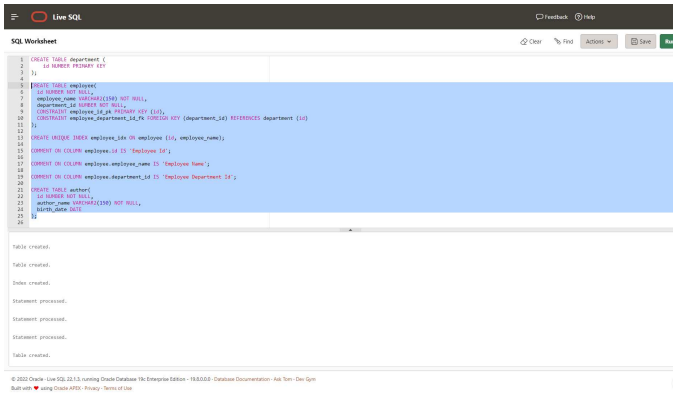


Figura 10. Validação do comando de criação de tabelas

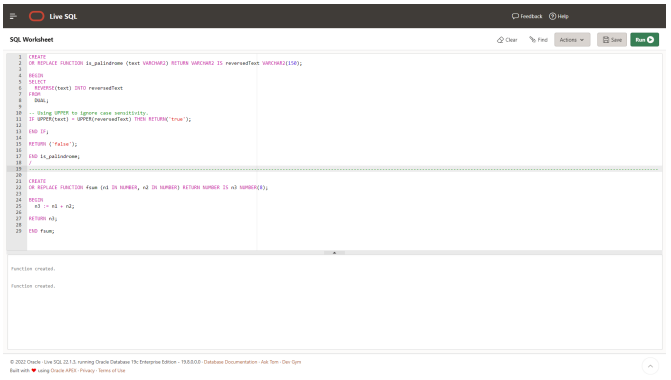


Figura 11. Validação do comando de criação de funções

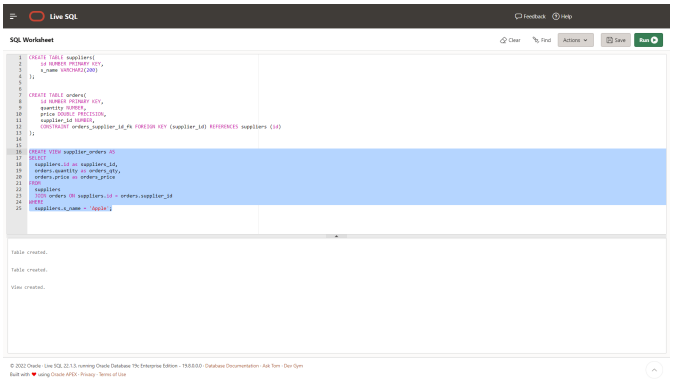


Figura 12. Validação do comando da criação de uma *view* com junção de tabelas

Portanto, a ferramenta se mostrou promissora e capaz de gerar comandos que consigam iniciar uma sincronização de bancos de dados.

5. EXPERIÊNCIA DE DESENVOLVIMENTO

5.1 Processo de desenvolvimento

O início do período de desenvolvimento se deu por meio de estudo de bibliotecas que fazem comparações de JSONs, a fim de entender as técnicas utilizadas pelas mesmas. Após essa análise, foi possível imaginar uma forma de gerar as diferenças entre os modelos de uma maneira ótima para a execução dos próximos passos.

O período de desenvolvimento durou cerca de 2 meses. Em seguida, foram realizadas as validações, algumas refatorações e correções que surgiam durante as validações com duração de 3 semanas.

5.2 Principais desafios

Para a realização dos geradores de scripts foi necessário a realização de muitos estudos e avaliações dos comandos de cada estrutura, principalmente, porque não era de conhecimento a sintaxe do PL/SQL. Então, foi necessário a realização de muitas consultas à documentação oficial da Oracle e de sites que dão suporte.

Além disso, foi necessário analisar quais seriam os modelos dos comandos a serem suportados na versão inicial da ferramenta, já que as possibilidades são muitas e, em alguns

casos, bem complexas. Dessa forma, foi pensado em suportar apenas os comandos mais simples e mais comumente utilizados.

6. CONCLUSÃO

6.1 Limitações

Atualmente, a ferramenta não realiza nenhum tipo de validação sobre os valores recebidos e isso pode acabar gerando erros durante a execução ou em scripts inválidos. Por exemplo, não faz sentido tentar criar uma tabela sem a indicação de um nome para o objeto. Além disso, nem todas as estruturas de um banco de dados estão suportadas, como *types*, *synonyms*, *roles*, *materialized views* etc. Então, essa escolha ocorre em virtude de se tratar de uma PoC e necessitar de uma avaliação que a ideia funcionaria dentro do esperado.

Ademais, mesmo sobre as estruturas que a ferramenta desenvolvida dá suporte nem todas as sintaxes estão cobertas. Por exemplo, se for necessário a realização de uma *view* cujo comando de geração necessite de junções de tabelas aninhadas não será possível. Por fim, um outro ponto que pode ser limitador é o desempenho, já que não foi realizado nenhum teste em busca de analisar o comportamento da ferramenta sobre uma base de dados grande.

6.2 Trabalhos futuros

Já que o objeto deste trabalho é uma prova sobre um conceito, muitas de suas funcionalidades são iniciais, então a possibilidade de trabalhos futuros é bem ampla. Entre elas podemos destacar a adaptação para permitir novos SGBDs, como PostgreSQL e MySQL, hoje a ferramenta possui uma configuração bem inicial para dar suporte à múltiplos SGBDs e uma possibilidade para dar continuidade à essa funcionalidade seria adicionar as palavras chaves e templates dos comandos de cada objeto específico do SGBD e achar uma forma de generalizar esses dados de cada sistema de gerenciamento de banco de dados em uma interface para ser possível propagar o objeto com as sintaxes e não ser necessário realizar importações específicas em cada arquivo que necessite utilizar.

Outros pontos seriam adaptar os templates dos comandos e os modelos de cada objeto para ampliar as possibilidades de comandos que podem ser gerados resolvendo, por exemplo, a limitação citada referente às *views* sobre junções de tabelas aninhadas; criar uma interface gráfica para facilitar a geração dos JSONs que serão utilizados como entrada; adaptar o código para passar a observar também as mudanças entre objetos e não apenas criação ou deleção; passar a aceitar comandos do tipo DML, podendo gerar comandos de seleção, inserção entre outros; e ainda, gerar relatórios e diagramas sobre um banco de dados a partir de um novo módulo com essa responsabilidade cuja entrada seria um JSON representando o banco de dados por completo.

7. AGRADECIMENTOS

Primeiramente agradeço à Deus, por ter me acompanhado durante todo o percurso até aqui. Além disso, agradeço aos meus pais que tanto lutaram para que chegasse até onde cheguei e aos meus amigos que aguentaram as minhas reclamações, que me deram forças para suportar todas as adversidades que foram enfrentadas e que me ajudaram de alguma forma a construir esse trabalho. Além disso, dedico à lembrança de Henry Nóbrega que foi um amigo e companheiro, principalmente durante a monitoria de Linguagem de Programação 2. Por fim, agradeço aos professores que mesmo diante de uma pandemia, onde foi necessário se reinventar para continuar a passar conhecimento, se mantiveram fortes e se adaptaram à situação; e em especial o professor Adalberto ao qual me presenteou com diversas oportunidades. Entre elas, a de participar de projetos de pesquisas e de sua orientação no desenvolvimento deste trabalho.

8. REFERÊNCIAS

[1] Angular. Retrieved February 2022 from https://angular.io/.

[2] Continuous integration vs. continuous delivery vs. continuous deployment. Retrieved September 2021 from https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[3] ESLint. Retrieved February 2022 from https://eslint.org/.

[4] Evolução dos bancos de dados no ambiente de desenvolvimento de projetos ágeis. Retrieved August 2021 from http://revistatis.dc.ufscar.br/index.php/revista/article/download/295/95.

[5] Evolutionary Database Design. Retrieved August 2021 from https://martinfowler.com/articles/evodb.html.

[6] Jest. Retrieved February 2021 from https://jestjs.io/.

[7] jsdiff. Retrieved December 2021 from https://github.com/kpdecker/jsdiff.

[8] json-diff. Retrieved December 2021 from https://github.com/andreyvit/json-diff.

[9] LESSQL: Dealing with database changes in Continuous Deployment. Retrieved August 2021 from https://ieeexplore.ieee.org/document/9054796.

[10] L'approche CI/CD en DevOps : pourquoi la mettre en place dans votre entreprise ?. Retrieved September 2021 from https://www.retengr.com/2020/12/18/approche-ci-cd-devops-definition-avantages/.

[11] Metodologias ágeis para gerenciamento de projetos de inovação e pesquisa e desenvolvimento. Retrieved September 2021 from https://www15.fgv.br/network/tcchandler.axd?TCCID=5498.

[12] Metodologias ágeis para o desenvolvimento de softwares. Retrieved August 2021 from https://periodicos.ufca.edu.br/ojs/index.php/cienciasustentabilidade/article/view/314/308.

[13] Migrations: o porque e como usar. Retrieved September 2021 from https://juniorb2s.medium.com/migrations-o-porque-e-como-usar-12d98c6d9269.

[14] Oracle Database Documentation. Retrieved December 2021 from https://docs.oracle.com/en/database/oracle/oracle-database/index.html.

[15] Oracle Live SQL. Retrieved March 2022 from https://livesql.oracle.com/.

[16] Oracle Tutorial. Retrieved December 2021 from https://www.oracletutorial.com/.

[17] Prettier. Retrieved February 2021 from https://prettier.io/.

[18] sql-formatter. Retrieved January 2022 from https://github.com/zeroturnaround/sql-formatter.

[19] Tech on the Net. Retrieved December 2021 from https://www.techonthenet.com/oracle/index.php.

[20] Typescript. Retrieved February 2022 from https://www.typescriptlang.org/.

[21] Tutorialspoint. Retrieved December 2021 from https://www.tutorialspoint.com/plsql/index.htm.

[22] UML Essencial: Um breve guia para a linguagem-padrão de modelagem de objetos; 3ª Edição. Retrieved September 2021.