



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

ANARCO QUARESMA ZEFERINO NASCIMENTO

**FERRAMENTA PARA VISUALIZAÇÃO DAS DEPÊNDENCIAS
ENTRE OBJETOS EM UM BANCO DE DADOS E SIMULAÇÃO DE
ALTERAÇÕES NOS OBJETOS**

CAMPINA GRANDE - PB

2021

ANARCO QUARESMA ZEFERINO NASCIMENTO

**FERRAMENTA PARA VISUALIZAÇÃO DAS DEPÊNDENCIAS
ENTRE OBJETOS EM UM BANCO DE DADOS E SIMULAÇÃO DE
ALTERAÇÕES NOS OBJETOS**

**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 Carlos Eduardo Santos Pires

CAMPINA GRANDE - PB

2021



N244f Nascimento, Anarco Quaresma Zeferino.
Ferramenta para visualização das dependências entre objetos em um banco de dados e simulação de alterações nos objetos. / Anarco Quaresma Zeferino Nascimento. - 2021.

10 f.

Orientador: Prof. Dr. Carlos Eduardo Santos Pires.

Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Sistemas de banco de dados. 2. Manutenção de banco de dados. 3. Operação DROP. 4. Oracle Autonomus Data Warehouse. 5. Sistema gerenciador de banco de dados. 6. Objetos em banco de dados. I. Pires, Carlos Eduardo Santos. II. Título.

CDU:004.65(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

ANARCO QUARESMA ZEFERINO NASCIMENTO

**FERRAMENTA PARA VISUALIZAÇÃO DAS DEPÊNDENCIAS
ENTRE OBJETOS EM UM BANCO DE DADOS E SIMULAÇÃO
DE ALTERAÇÕES NOS OBJETOS**

**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.(a.) Carlos Eduardo Santos Pires
Orientador – UASC/CEEI/UFCG**

**Professora Dr.(a.) João Arthur Brunet
Examinador – UASC/CEEI/UFCG**

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

Trabalho aprovado em: 25 de maio de 2021.

CAMPINA GRANDE - PB

ABSTRACT

In the context of database systems, both the development and the maintenance are process that dictates changings in its models. These changings have consequences in the entire system. The physic, logic and conceptual models owns versions that vary as the system evolves. The way as the development team or database administrators (DBA's) visualizes the system, more specifically the relationships between the objects of the system, have direct effects in the management of the system. Therefore, it was developed a web application that allows the visualization of those relationships and shows to the user the objects that would have any side effect if any other object of the system changes. For this work's scope, it was only considered side effects from the DROP operation over any object of the database management system Oracle Autonomous Data Warehouse, from Oracle Cloud service. This tool show itself fundamentally useful in the database systems maintenance.

FERRAMENTA PARA VISUALIZAÇÃO DAS DEPÊNDENCIAS ENTRE OBJETOS EM UM BANCO DE DADOS E SIMULAÇÃO DE ALTERAÇÕES NOS OBJETOS

Anarco Nascimento

Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
anarco.nascimento@ccc.ufcg.edu.br

Orientador: Carlos Eduardo Santos Pires, PHD

Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
cesp@dsc.ufcg.edu.br

RESUMO

No contexto de sistemas de banco de dados, tanto o desenvolvimento quanto a manutenção são processos que demandam alterações nos seus modelos. Essas alterações causam consequências em todo o sistema. Os modelos lógico, conceitual e físico possuem versões que variam à medida que o sistema evolui. O modo como a equipe de desenvolvimento ou profissionais de manutenção de banco de dados, os DBAs, visualizam o sistema, mais especificamente as relações entre os objetos desse sistema, tem efeitos diretos na forma em que o sistema será gerenciado por eles. Sendo assim, desenvolvemos uma aplicação web que permita a visualização dessas relações e exiba ao usuário os objetos que sofreriam algum efeito colateral caso algum outro objeto do sistema for alterado. Para o escopo deste trabalho, consideraremos apenas as alterações causadas pela operação DROP sobre algum objeto do sistema gerenciador de banco de dados Oracle Autonomous Data Warehouse, do serviço Oracle. Essa ferramenta se mostra fundamentalmente útil na manutenção de sistemas de banco de dados.

Palavras-chave

Sistemas de banco de dados, esquema, grafo, dependência, DROP.

Repositório

<https://github.com/anarcoqzn/graphndrop>

1. INTRODUÇÃO

Em sua definição, banco de dados é uma coleção de dados relacionados [3]. Cada banco de dados representa uma parte do mundo real chamada, armazenando informações sobre os objetos e relações deste minimundo [3]. A compreensão de como essas relações se constroem é de grande relevância na construção e manutenção de um sistema de banco de dados. Durante todo o processo de desenvolvimento de um sistema desse tipo, alterações em seu *design* são muito comuns, podendo ocorrer em qualquer fase do processo, ou seja, na fase de construção do seu modelo conceitual, do modelo lógico e, no pior caso, quando o sistema já está em ambiente de produção. Em todos os casos, visualizar de

maneira concreta o resultado dessas mudanças antes de executá-las diretamente no sistema, torna o processo de desenvolvimento do sistema mais seguro, auxiliando a manter a consistência dos dados, dos objetos e de suas relações em todas as versões do sistema.

Para Sistemas Gerenciadores de Banco de Dados (SGBDs) diferentes, a definição e construção dos esquemas também é diferente. No caso do SGBD Oracle abordado neste trabalho, um esquema de banco de dados é uma coleção de estruturas de dados lógica. Um usuário do banco de dados possui um esquema de banco de dados, que tem o mesmo nome que o nome do usuário. Objetos do esquema são estruturas criadas pelos usuários que fazem referência direta aos dados no banco de dados [5]. Os objetos são diversos e suas relações são tão complexas quanto aquelas existentes no minimundo que o banco de dados representa.

Sabe-se que visualizar os objetos de um sistema facilita tanto o processo de desenvolvimento quanto a manutenção desse sistema. No entanto, construir ferramentas que forneçam esse tipo de serviço não é trivial. No contexto de banco de dados, os grandes *players* desta área fornecem para seus usuários um conjunto de ferramentas que permitem a visualização e manutenção dos seus esquemas de banco de dados, dando suporte com editores de código e de consultas, visualização de uma árvore de dados, com detalhamento de cada objeto, entre outros. No momento atual, a Oracle está desenvolvendo uma ferramenta para seus usuários chamada GraphDatabase [2] prometendo prover visualização e análises de um banco de dados autônomo. Porém, dificilmente será uma aplicação de código aberto e software livre.

Dessa forma, este trabalho relata o desenvolvimento da ferramenta GraphNDropOracle com código aberto que permita visualizar as relações de dependência entre os objetos em um esquema de banco de dados Oracle através de um grafo direcionado, bem como visualizar os efeitos caso algum objeto seja alterado. A operação considerada neste trabalho foi o DROP sobre um objeto. Para este fim, a aplicação executa várias consultas ao dicionário de dados do sistema e analisa as *views* (visões) necessárias e deduz esses efeitos, exibindo-os ao usuário.

2. SOLUÇÃO

Para atingir os objetivos citados anteriormente, foi criada a ferramenta GraphNDropOracle. Nas seções a seguir, é apresentada a descrição, arquitetura e o processo de desenvolvimento desta aplicação.

2.1 DESCRIÇÃO

O GraphNDropOracle é uma aplicação *web* com código aberto para visualização das dependências de um banco de dados *Autonomous Data Warehouse* (ADW) – um serviço disponibilizado pela Oracle – e dos efeitos colaterais da execução da operação DROP sobre algum objeto do sistema de banco de dados selecionado. Esse tipo de banco de dados foi escolhido principalmente pela sua facilidade de conexão externa. O ADW disponibiliza um conjunto de passos simplificando o processo de conexão com um banco de dados a partir de uma aplicação externa. Ao fazer o download da *Wallet* [4] do esquema do usuário, pode-se acessar o banco de dados fazendo uso das *Connect Strings* que são *strings* de acesso ao esquema que indica onde será o host que executa as instruções direcionadas ao banco de dados.

Com o fim de simplificar o processo de desenvolvimento, considerou-se como relações entre objetos de um esquema de banco de dados as relações descritas na visão *USER_DEPENDENCIES* e as restrições de chave estrangeira. Com isso, o GraphNDropOracle fornece as seguintes funcionalidades:

- Cadastrar esquemas: No ADW, cada usuário possui um esquema. O usuário pode conectar seu esquema à aplicação fornecendo a *Connect String*, o nome de usuário e a senha;
- CRUD em conexões à esquemas: Conjunto de operações (*Create, Read, Update e Delete*) criar, ler, atualizar e remover conexões à esquemas existentes.
- Selecionar esquema: Selecionar um esquema da lista de esquemas para operar;
- Selecionar tipo de relação: O usuário pode escolher se deseja visualizar o grafo representando as relações descritas na visão *USER_DEPENDENCIES* ou as relações de restrição de chave estrangeira;
- Selecionar um objeto: Cada nó do grafo representa um objeto do esquema criado pelo usuário. Ao selecionar, o usuário pode visualizar detalhes deste objeto;
- Visualizar os efeitos da operação DROP: Após selecionar o objeto, o usuário pode visualizar os objetos que sofreriam algum efeito colateral caso a operação DROP for executada;
- Visualizar detalhes dos objetos dependentes: Após executar a simulação da operação DROP, o usuário pode visualizar informações dos objetos dependentes.

2.2 ARQUITETURA

Por se tratar de uma ferramenta que acessa um banco de dados do usuário, esta foi dividida em *back-end* e *front-end*, uma responsável pela lógica de configuração de conexão e consultas ao esquema e a outra pela lógica de exibição das informações e interações com o usuário na tela.

2.2.1 Back-end

Para o *back-end* foi escolhido o *framework* NodeJS com o *framework* Express para criação de um servidor HTTP em conjunto com a biblioteca *node-oracledb* para conectar e acessar o banco de dados ADW. NodeJS é popularmente conhecido por facilitar o processo de criação de APIs REST, simplificando todo o processo de desenvolvimento. Para realizar as consultas SQL, foi utilizada a versão 19c da Oracle.

A estrutura do código foi dividida em duas seções principais: configuração do banco de dados e consultas. Na seção de configurações, é onde se realiza o CRUD sobre as conexões cadastradas pelo usuário. Na seção de consultas, são feitas várias consultas ao dicionário de dados do esquema para resgatar os dados requisitados pelo *front-end*. A partir de consultas às visões *USER_CONS_COLUMNS* e *USER_CONSTRAINTS* descritas na documentação da linguagem [1] pode-se inferir as relações de restrição de chave estrangeira; e a partir de consultas à visão *USER_DEPENDENCIES* pode-se deduzir outros tipos de relações no esquema de banco de dados selecionado.

A aplicação fornece, portanto, uma API REST para se comunicar com o *front-end*. Os principais *endpoints* são listados a seguir no formato:

{Método HTTP} /nome:

- {POST} /connections – Cria uma nova conexão à um esquema;
- {PATCH} /connections – Seleciona uma conexão da lista de conexões;
- {GET}/userDependencies – Retorna os dados da visão *USER_DEPENDENCIES*;
- {GET} /tables/dependencies – Retorna todas as relações de chave estrangeira;

2.2.2 Front-end

No *front-end* foi escolhido utilizar o *framework* ReactJS para construção das telas, o Axios para executar requisições à API criada no *back-end* e o *framework* Redux para gerenciamento dos estados da aplicação durante sua execução. A escolha do ReactJS se dá principalmente pela simplificação do processo de desenvolvimento por ser um *framework* escrito em JavaScript – a mesma linguagem em que o NodeJS é escrito – evitando complexidades por se usar linguagens diferentes em um mesmo projeto, e pela sua característica inerente de componentização do sistema, trazendo maior organização e reuso de código. Para estilização das telas, foi escolhido trabalhar com a biblioteca *styled-components* disponível no repositório de bibliotecas do Node (npm); por ser necessário apenas de CSS para estilização

utilizando esse framework, elimina-se a dependência de várias outras bibliotecas ao utilizar componentes estilizados prontos. Para construção e exibição do grafo direcionado que representa as relações descritas previamente neste artigo, utilizou-se a biblioteca react-d3-graph, pela sua simplicidade no uso e não ser paga, também disponível no repositório npm.

A estrutura do código no *front-end* foi dividida em duas seções principais: *Services* e *Components*. Na seção de *Services* está todo o código necessário para comunicação com o *back-end* e gerenciamento do estado da aplicação durante sua execução. Na seção de componentes está o código de todos os componentes utilizados na aplicação, cada um com seu devido código CSS.

2.2.3 Tela principal

Na figura 1 está a apresentação da tela principal com a delimitação das áreas. Logo após, está a descrição de cada área, identificando-as pelo número entre colchetes.

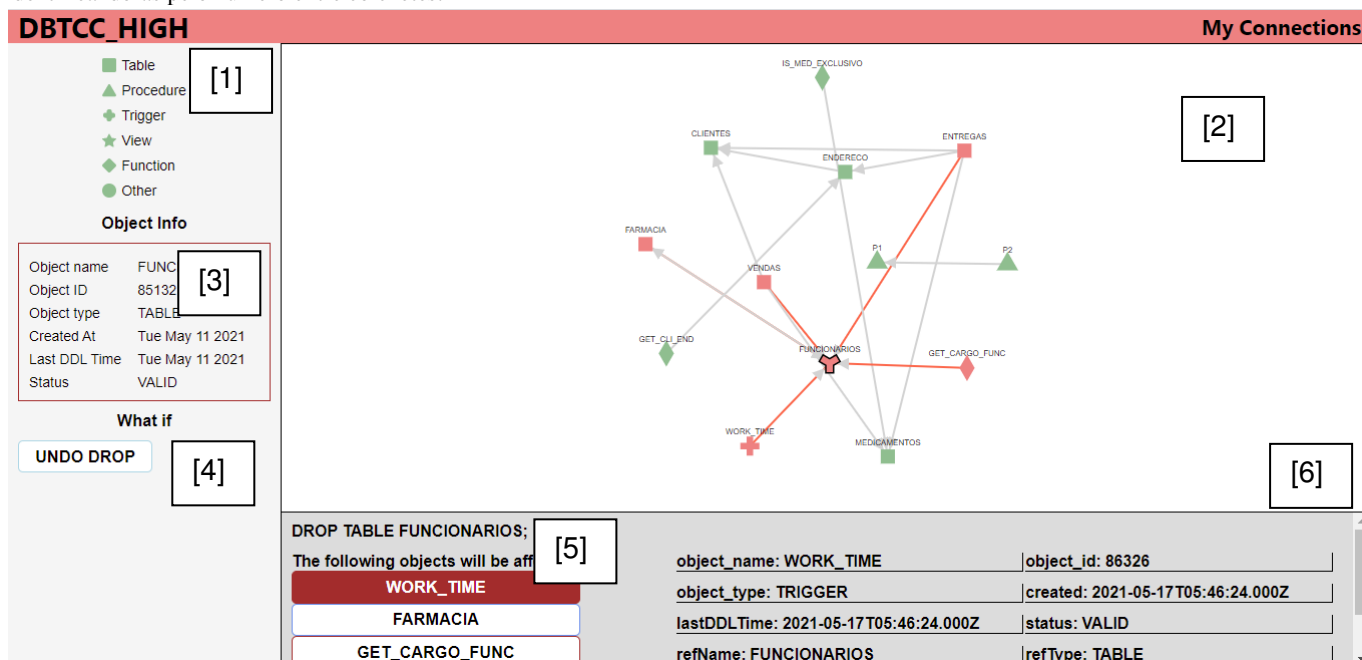


Figura 1 Tela principal da aplicação

Na área definida pelo número 1, é exibida uma legenda com as possíveis figuras que um nó pode assumir dependendo do tipo de objeto que representa.

Na área delimitada pelo número 2, é mostrado o grafo de dependência entre os objetos. Este grafo é interativo, podendo-se fazer zoom-in e zoom-out, bem como dispor os nós em qualquer posição dentro da área delimitada. Cada nó representa um objeto armazenado no esquema do usuário e a direção da aresta indica a dependência.

Na área delimitada pelo número 3, estão as informações de um objeto do grafo quando um nó é selecionado. Estas informações são extraídas da visão *USER_OBJECTS*.

Na área delimitada pelo número 4, está o botão com a opção de visualizar os efeitos da operação *DROP* sobre o objeto selecionado.

Na área delimitada pelo número 5, é possível visualizar os objetos que dependem diretamente do objeto sobre o qual a operação *DROP* foi simulada.

Finalmente, na área delimitada pelo número 6, estão as informações detalhadas do objeto selecionado da lista de objetos dependentes. Estas informações foram extraídas da visão *USER_OBJECTS*.

O que o GraphNDropOracle exibe serve como suporte para desenvolvedores e DBAs em todo o tempo de vida do sistema de banco de dados, pois ao mostrar as dependências do sistema, pode-se inferir as consequências que a operação *DROP* causará ao banco de dados em geral. Analisando essas consequências, pode-se ter uma ideia do que fazer para evitar maiores danos.

2.2.4 Lógica da aplicação

Para executar testes sobre a ferramenta e exibir os códigos e grafos nas figuras 1,2 ,3 e 4 foi criado um esquema exemplo no serviço Cloud Oracle. O leitor pode ter acesso aos comandos de definição do esquema no repositório do projeto.

Como dito anteriormente, são feitas várias consultas ao dicionário de dados do esquema selecionado, extraindo das visões necessárias as informações que descrevem as relações de dependência entre os objetos. Ao consultar a visão *USER_DEPENDENCIES*, é retornado um objeto JSON, com as informações consideradas relevantes para a construção do grafo e para a análise da operação *DROP* sobre este objeto.

Como descrito na referência ao banco de dados da Oracle, a visão `USER_DEPENDENCIES` descreve as dependências entre `procedures`, `packages`, `functions`, `package bodies`, e, `triggers` possuídos pelo usuário atual, incluindo dependências sobre `views` criadas sem um link a um banco de dados. Suas colunas são as mesmas que aquelas em `ALL_DEPENDENCIES` [1]. Portanto, esta visão não traz informações sobre as relações de restrição de chave estrangeira.

O exemplo mostrado na figura 2 retrata o caso de o banco de dados possuir um objeto do tipo `PROCEDURE` chamado `P2`, que dependa de outro objeto do mesmo tipo chamado `P1`.

```
{
  "name": "P2",
  "type": "PROCEDURE",
  "refOwner": "ANARCO",
  "refName": "P1",
  "refType": "PROCEDURE",
  "refLinkName": null,
  "schemaID": 147,
  "depType": "HARD"
}
```

Figura 2 Consulta a `USER_DEPENDENCIES` com `P2` dependendo de `P1` ambos do tipo `PROCEDURE`

Neste caso, será construído e exibido para o usuário um grafo direcionado com a aresta partindo do nó representando `P2` e chegando no nó representando `P1`, como mostrado na figura 3.

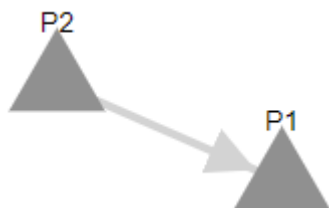


Figura 3 Grafo representando dependência de `P2` com `P1`

Ao consultar as visões `USER_CONS_COLUMNS` e `USER_CONSTRAINTS` pode-se deduzir quais tabelas se relacionam entre si através da restrição de chave estrangeira e em quais colunas das tabelas a restrição é aplicada. Neste caso, também é retornado um objeto JSON com os dados necessários para montagem do grafo de dependência e para a análise, como mostrado na figura 4.

```
{
  "constraint_name": "ENDERECO_CPF_CLIENTE_FKEY",
  "from": {
    "table": "ENDERECO",
    "column": "CPF_CLIENTE"
  },
  "to": {
    "table": "CLIENTES",
    "column": "CPF"
  }
}
```

Figura 4 Objeto JSON representando a relação de chave estrangeira

Com essas informações, é possível construir um grafo direcionado e interativo que representa as relações de dependência entre os objetos em um banco de dados.

No grafo, cada tipo de objeto é mapeado para um nó com uma forma específica, para diferenciá-los na visualização. Porém, como em um mesmo banco de dados existem vários tipos diferentes de objetos, apenas aqueles considerados principais foram selecionados para terem uma forma específica. Outros objetos não considerados principais tem uma forma padrão. Na tabela 1 estão listados as possíveis formas e os objetos os quais elas representam.








	Tabela
	Visão
	Função
	Procedimento
	Trigger
	Outros
	Objeto sob DROP

Tabela 1 Mapeamento da figura do nó com o tipo de objeto

No carregamento inicial da ferramenta, são feitas as consultas ao dicionário de dados e retornado para o *front-end* os objetos JSON que representam as dependências mencionadas anteriormente. Com esses dados, é plotado o grafo de dependência do esquema, mapeando cada objeto para o seu respectivo nó considerando o seu tipo. Caso o status do objeto no banco de dados seja `VALID` (válido), o nó que o representa fica da cor verde; caso o status seja `INVALID` (inválido), o nó que o representa fica da cor vermelha. Caso o status não seja nenhuma dessas opções, o nó fica cinza.

2.3 A EXECUÇÃO DA OPERAÇÃO

Ao selecionar um objeto do grafo, o usuário pode simular os efeitos no grafo e, conseqüentemente, no banco de dados que o grafo representa caso esse objeto seja removido do esquema com a execução da operação `DROP`.

Executando essa operação, a figura do nó selecionado muda para a figura que indica um nó sob `DROP`, como descrito na tabela 1; e tanto a cor do objeto selecionado como a cor dos objetos que dependem diretamente do objeto selecionado fica vermelha, mostrando que os nós serão invalidados caso a operação seja efetuada. Até que a operação `UNDO DROP` seja feita, retornando o grafo para o estado original, não é possível selecionar outro

objeto para fazer a simulação, ou seja, a ferramenta, atualmente, permite simular a operação apenas sobre um objeto por vez.

A figura 5 mostra o diagrama de sequência que descreve o funcionamento geral da aplicação. Ao reduzir as consultas ao banco de dados, reduziu também o tempo de carregamento dos objetos. Porém, ao fazer isso, caso o usuário altere o banco de dados durante o uso da aplicação, será necessário recarregar a página, retornando ao estado inicial.

Inicialmente, definiu-se quais tecnologias deveriam ser usadas neste projeto. A escolha se deu principalmente pela experiência em trabalhos semelhantes anteriores. Em seguida, foi necessário um estudo sobre a documentação da linguagem SQL do Oracle e sobre as referências ao banco de dados para selecionar as visões que possuem os dados necessários. Logo depois, se pensou nos requisitos e funcionalidades do *back-end* e em como montar os objetos a serem retornados e quais *endpoints* deveriam existir na API REST. Em seguida, planejou-se um cronograma para implementação do código e escrita deste artigo em paralelo.

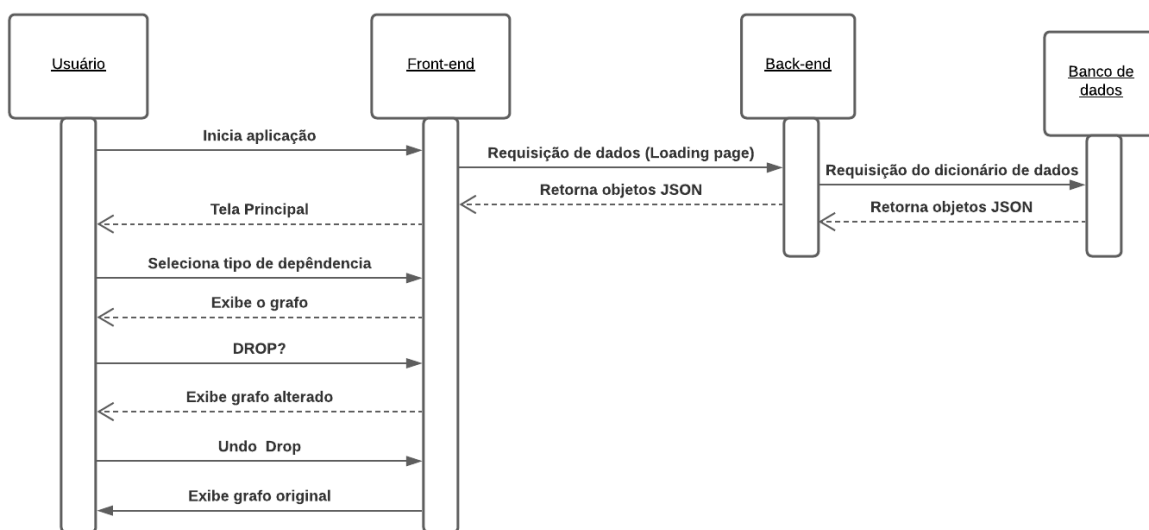


Figura 5 Diagrama de sequência da aplicação

2.2.5 A escolha da operação

Em um sistema de banco de dados comum, existem diversos objetos e cada um com um conjunto de operações específicas. Como pode ser visto na seção 3.5, planeja-se implementar algumas destas operações. Porém, para o escopo desta aplicação, resolveu-se reduzir a uma operação. Ao analisar os objetos do banco de dados Oracle, foi notado que a operação DROP é uma operação comum a todos os objetos Oracle, portanto ela foi a escolhida para realizar a análise, resultando em uma generalização no processo de análise e simplificação no processo de desenvolvimento.

3. EXPERIÊNCIA

Com este trabalho, adquiriu-se experiência ao ser introduzido um novo tipo de SGBD e novas tecnologias de desenvolvimento de aplicações web, como o react-d3-graph e node-oracledb, aumentando o leque de possibilidades para futuros trabalhos.

Englobando novas tecnologias, também pôde-se aprimorar o processo de desenvolvimento de software ao separar e organizar as seções do código, trazendo mais reuso e legibilidade.

3.1 PROCESSO DE DESENVOLVIMENTO

A partir disso, foi necessário encontrar alguma ferramenta que auxiliasse a plotagem de grafos em ReactJS. Ao encontrar a biblioteca react-d3-graph e executar a primeira montagem do grafo de dependência utilizando esta ferramenta (figura 4), o processo de construção do grafo foi facilitado. Em seguida, executou-se um estudo mais aprofundado da documentação para definir a operação que iria ser analisada pelo sistema, chegando à operação DROP.

Tendo o primeiro grafo montado, o próximo passo foi definir as figuras para cada tipo de objeto, como o grafo deveria simular a execução da operação DROP e as seções da aplicação demonstradas na figura 1.

Com isso em mãos, foi possível concluir o desenvolvimento da aplicação nessa primeira instância, exibindo tanto o grafo de dependência quanto o resultado da análise.

3.2 DESAFIOS

Manter a comunicação e sincronização entre os componentes do sistema através dos estados se mostrou mais desafiador do que se esperava. A definição, implementação e retorno das consultas ao banco de dados também se mostrou uma barreira difícil, principalmente pelo fato de o SQL da Oracle ser uma nova tecnologia que foi aprendida durante o processo de

desenvolvimento. Estas dificuldades foram suplantadas a partir de análise e melhoria do código escrito.

Uma dificuldade que talvez seja inerente a qualquer tipo de sistema web é a implementação de uma interface gráfica simples, porém útil ao usuário, se mostrou uma dificuldade neste projeto, sendo necessária a reescrita de código por várias vezes para atingir o objetivo esperado.

Pelo fato de a aplicação abordar apenas uma operação, não se demonstra ainda com potencial para ser colocada em ambiente de produção, estando ainda em fase de teste da operação já desenvolvida e planejamento e implementação de outras.

O grande desafio que determinou o maior atraso no cronograma foi a dificuldade em encontrar uma maneira de conectar a aplicação ao banco de dados. Ao descobrir a biblioteca node-oracledb, essa barreira foi rapidamente superada.

3.3 LIMITAÇÕES

O fato de ser necessária abrir uma conexão com o banco de dados para então executar as consultas SQL e logo em seguida fechar essa conexão torna o carregamento dos dados necessário lento. Essa limitação é um pouco menos sentida pelo usuário ao fazer o carregamento antes de direcioná-lo para a tela principal (figura 1). Porém, caso o usuário atualize o banco de dados durante a execução da ferramenta, será necessário recarregar os dados transferindo a aplicação para o seu estado inicial.

Uma outra limitação é a pequena faixa de possibilidades de mapeamento do tipo do objeto para uma figura específica em um nó do grafo. Também não há um filtro para determinar a quantidade de objetos ou os tipos de objetos que se deseja visualizar no grafo de dependência. Ambas limitações podem tornar a visualização das dependências mais confusa a depender da quantidade de objetos no banco de dados.

3.4 LIÇÕES APRENDIDAS

Como existem diversas tecnologias para construção de sistemas web, torna-se totalmente necessário o desenvolvedor se manter ativo no estudo e prática tanto de novas tecnologias como de tecnologias que já possua experiência, para facilitar o processo de aprendizado caso uma nova tecnologia se apresente.

3.5 TRABALHOS FUTUROS

As possibilidades de agregar valor para o GraphNDropOracle são várias. Algumas delas já em vistas de implementação é a adição novas operações para objetos mais específicos, como por exemplo ALTER TABLE, ou DROP COLUMN, exibir ao usuário todas as análises referentes à estas operações e coloca-la em ambiente de produção para suprir as necessidades dos usuários.

Também se pensa na possibilidade de a aplicação abarcar outros Sistemas Gerenciadores de Banco de Dados (SGBDs) como

PostgreSQL e MariaDB, dando visibilidade da aplicação para outros usuários.

Atualmente, a aplicação não faz nenhuma alteração no banco de dados original. No futuro, pode-se evoluir para uma aplicação de gerenciamento completa, executando comando DDL e monitorando usuários, esquemas e objetos no geral.

REFERÊNCIAS

- [1] DatabaseReference.
<https://docs.oracle.com/en/database/oracle/oracle-database/19/refm/>
- [2] Graph Database Oracle.
<https://www.oracle.com/database/graph/>
- [3] Elmasri, Ramez e Navathe, Shamkant B; Sistemas de banco de dados; tradução de Daniel Vieira; 6ª ed. – São Paulo: Pearson Addison Wesley, 2011
- [4] Oracle Cloud Infrastructure Information, Connecting to an Autonomous Database
<https://docs.oracle.com/en-us/iaas/Content/Database/Tasks/adbconnecting.htm>
- [5] Oracle Database Concepts.
<https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/introduction-to-oracle-database.html#GUID-35C20601-E266-486E-987B-7F355DB10DD4>