Plataforma de ensino de Algebra Relacional utilizando programação em blocos com infraestrutura em Serverless

Thales Atheniel Farias de Godoi

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Uberlândia
2024

Thales Atheniel Farias de Godoi

Plataforma de ensino de Algebra Relacional utilizando programação em blocos com infraestrutura em Serverless

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Humberto Luiz Razente

Uberlândia
2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade de Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4144 - http://www.portal.facom.ufu.br/ facom@ufu.br

ATA DE DEFESA - GRADUAÇÃO

Curso de Graduação em:	Bacharelado em Sistemas de Informação				
Defesa de:	GSI040 - Trabalho de Conclusão de Curso 2				
Data:	26/04/2024	Hora de início:	16:10	Hora de encerramento:	17:40
Matrícula do Discente:	11511BSI232				
Nome do Discente:	Thales Atheniel Farias de Godoi				
Título do Trabalho:	Plataforma de ensino de álgebra relacional utilizando programação em blocos com infraestrutura em serverless				
A carga horária curricular foi cumprida integralmente?	(X) Sim ( ) Não				

Reuniu-se no Anfiteatro/Sala 1B132, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Curso de Graduação em Sistemas de Informação, assim composta: Professores: Prof. Rafael Dias Araújo - FACOM/UFU; Prof. Felipe Alves da Louza - FEELT/UFU; e Humberto Luiz Razente - FACOM/UFU, orientador(a) do(a) candidato(a).

Iniciando os trabalhos, o(a) presidente da mesa, Prof. Humberto Luiz Razente, apresentou a Comissão Examinadora e o candidato(a), agradeceu a presença do público, e concedeu ao discente a palavra, para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do curso.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

(X) Aprovado Nota [65]

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Humberto Luiz Razente**, **Professor(a) do Magistério Superior**, em 28/04/2024, às 07:42, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rafael Dias Araújo**, **Professor(a) do Magistério Superior**, em 29/04/2024, às 09:46, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Felipe Alves da Louza**, **Professor(a) do Magistério Superior**, em 30/04/2024, às 08:31, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5373953** e o código CRC **9E1ED432**.

Referência: Processo nº 23117.029626/2024-71 SEI nº 5373953

Resumo

Este trabalho apresenta o desenvolvimento inicial de uma plataforma destinada ao ensino de álgebra relacional através da programação em blocos. A principal motivação para este projeto é a dificuldade enfrentada por iniciantes na compreensão de conceitos fundamentais de álgebra relacional, essenciais para o estudo de bancos de dados. A solução proposta visa tornar o aprendizado mais acessível e intuitivo, utilizando uma abordagem lúdica e interativa.

A plataforma foi construída com uma infraestrutura completamente serverless, englobando tanto o *frontend* quanto o *backend*. No *frontend*, foram empregadas as tecnologias React, Blockly e Cloudflare Pages. No *backend*, utilizamos Node.js e Cloudflare Workers. O armazenamento de dados foi implementado utilizando o Cloudflare D1.

Os principais resultados indicam que a plataforma é capaz de suportar uma demanda razoável de requisições diárias, demonstrando uma alta escalabilidade e baixo custo operacional. A abordagem da programação em blocos adotada facilita o entendimento dos conceitos de álgebra relacional, tornando o aprendizado mais próximo.

Palavras-chave: Algebra Relacional, Javascript, Node, React, Serverless.

Lista de símbolos

σ	Seleção (Letra grega Sigma)
π	Projeção (Letra grega Pi)
ρ	Renomeação (Letra grega Rho)
$\cup$	União
$\cap$	Interseção
$-$	Diferença
$\times$	Produto Cartesiano
$\bowtie$	Junção (join)
$\div$	Divisão

Lista de ilustrações

Figura 1 – Dashboard dbSnap 25
Figura 2 – Dashboard DB-Learn 26
Figura 3 – Dashboard Algebra Blocks 43
Figura 4 – Biblioteca de Blocos 44
Figura 5 – Playground. 45
Figura 6 – Visualização de Dados 46
Figura 7 – Resultado do teste de carga K6 48
Figura 8 – Painel da CloudFlare 49

Lista de tabelas

Tabela 1 – Quantidade de requisições gratuitas por dia usando Serverless da Cloud-Flare 23
Tabela 2 – Custo mensal para absorver 1000 usuários 23

Sumário

1 INTRODUÇÃO 15
1.1 Motivação 15
1.2 Justificativa 15
1.3 Objetivos 16

2 REVISÃO BIBLIOGRÁFICA 19
2.1 Algebra Relacional 19
2.1.1 Seleção 20
2.1.2 Projeção 20
2.1.3 Renomeação 21
2.1.4 União 21
2.1.5 Intersecção 21
2.1.6 Diferença 21
2.1.7 Produto Cartesiano 22
2.1.8 Junção 22
2.1.9 Divisão 22
2.2 Serverless 23
2.3 Programação em Blocos 24
2.4 Trabalhos Relacionados 25
2.4.1 dbSnap 25
2.4.2 DB-Learn 26

3 DESENVOLVIMENTO 27
3.1 Infraestrutura 27
3.1.1 Ambiente de Desenvolvimento 27
3.1.2 Ambiente de Produção 28
3.2 Tecnologias utilizadas no Frontend 29
3.2.1 React 29

3.2.2 Blockly . . . 29
3.2.3 Cloudflare Pages . . . 29
3.3 Tecnologias utilizadas no Backend . . . 29
3.3.1 Node.js . . . 29
3.3.2 Cloudflare Workers . . . 29
3.4 Tecnologias utilizadas no Banco de Dados . . . 30
3.4.1 Cloudflare D1 . . . 30
3.4.2 PostgreSQL . . . 30
3.5 Implementação do Frontend . . . 31
3.5.1 Arquivo `app.tsx` . . . 31
3.5.2 Arquivo `getDefaultToolbox.ts` . . . 36
3.6 Implementação do Backend . . . 41
3.6.1 Arquivo `worker.ts` . . . 41

4 RESULTADOS . . . 43
4.1 Plataforma . . . 43
4.1.1 Dashboard Inicial . . . 43
4.1.2 Biblioteca de Blocos . . . 44
4.1.3 Playground de Álgebra Relacional . . . 45
4.1.4 Visualização de Dados e Resultados . . . 46
4.1.5 Funcionalidades Adicionais . . . 47
4.2 Teste de Carga . . . 48
4.2.1 Resultado do teste de carga . . . 48

5 CONSIDERAÇÕES FINAIS . . . 51
5.1 Limitações do Trabalho . . . 52
5.2 Trabalhos Futuros . . . 52
5.3 Considerações Finais . . . 53

REFERÊNCIAS . . . 55

Capítulo 1

Introdução

Este capítulo apresenta os seguintes tópicos: motivação, justificativa e objetivos na construção de uma nova plataforma para ensino de Algebra Relacional utilizando programação em blocos com infraestrutura em Serverless.

1.1 Motivação

A principal motivação é revitalizar o dbSnap (SILVA; CHON, 2015; SILVA; LOZA; RAZENTE, 2022a; SILVA; LOZA; RAZENTE, 2022b), uma plataforma de programação em blocos para álgebra relacional, através da criação de uma nova plataforma. O maior objetivo é torná-la escalável e mais amigável, incorporando novas tecnologias e uma nova interface.

Ao fundir os princípios fundamentais da álgebra relacional com a abordagem lúdica programação em blocos, pretende-se oferecer uma experiência de aprendizagem mais acessível e envolvente para os usuários. Destaca-se que essa abordagem não apenas facilitará a compreensão dos conceitos teóricos densos da álgebra relacional, mas também poderá aumentar a retenção do conhecimento com aplicações práticas.

Dessa forma, a principal meta do presente projeto é criar uma nova base para reengenharia do dbSnap, implementando um novo sistema de *backend* e banco de dados relacional que processem às consultas em uma arquitetura serverless escalável.

1.2 Justificativa

O desenvolvimento desta plataforma de ensino de álgebra relacional baseada em programação em blocos é motivado por diversas razões, que poderiam ser aplicadas em várias outras áreas de estudo, principalmente dentro da computação.

Em primeiro lugar, observa-se uma oportunidade de facilitar o ensino da álgebra relacional, que é frequentemente utilizada como introdução ao estudo de banco de dados e

linguagem de consultas. É comum que os estudantes comecem a explorar temas mais complexos sem terem uma compreensão sólida dos fundamentos da álgebra relacional, o que pode resultar em dificuldades significativas ao avançar para disciplinas mais avançadas em banco de dados.

Além disso, a programação em blocos tem se mostrado uma abordagem eficaz no ensino de programação para crianças e jovens, oferecendo uma maneira intuitiva e acessível de aprender os fundamentos da lógica de programação, instruções explícitas sobre design centrado no usuário são necessárias, mas podem ser mais eficazes quando o usuário atinge a idade de 10 ou 11 anos (HANSEN et al., 2016). No entanto, essa metodologia é raramente aplicada no contexto do ensino da álgebra relacional ou em outros assuntos da computação, apesar de seu potencial para simplificar conceitos complexos e tornar o aprendizado mais envolvente e interativo.

Portanto, a justificativa para a criação desta plataforma é originada na necessidade de fornecer fundamentos sólidos aos estudantes de computação no aprendizado da álgebra relacional e temas mais avançados de banco de dados, utilizando uma metodologia que tem se mostrado promissora.

1.3 Objetivos

O principal objetivo deste projeto foi desenvolver uma plataforma funcional, intuitiva e escalável para o ensino de álgebra relacional utilizando programação em blocos e serverless. Para alcançar esse objetivo, os seguintes objetivos específicos foram perseguidos:

1. **Desenvolver uma interface amigável:** Criar uma interface de usuário intuitiva e fácil de usar, que permita aos usuários interagir de forma direta com os conceitos da álgebra relacional por meio de blocos de programação.

2. **Implementar funcionalidades essenciais:** Incorporar os principais conceitos e operadores da álgebra relacional, como seleção, projeção, união, interseção, diferença, produto cartesiano e junção, de forma clara e didática na plataforma.

3. **Proporcionar feedback imediato:** Integrar um sistema de feedback que forneça retornos imediatos aos usuários, permitindo-lhes entender rapidamente o impacto de suas ações e corrigir eventuais erros durante as consultas.

4. **Adotar uma abordagem escalável:** Utilizar uma infraestrutura serverless para garantir que a plataforma possa lidar com inúmeros usuários simultâneos e que possa ser facilmente escalada conforme necessário, e o mais importante, mantendo um custo de operação baixo.

Ao alcançar esses objetivos, espera-se que a plataforma desenvolvida não apenas ajude os usuários a compreender os conceitos da álgebra relacional de forma mais eficaz, mas

também sirva como um modelo para o uso de programação em blocos no ensino de outros tópicos da computação.

Capítulo 2

Revisão Bibliográfica

2.1 Algebra Relacional

A álgebra relacional (CODD, 1970) foi proposta em 1970, antes mesmo do desenvolvimento da linguagem SQL. Foi criada por Edgard F. Codd, um dos grandes matemáticos britânicos da época. É uma derivação da lógica de primeira ordem e da álgebra de conjuntos, possui os operadores: Seleção, Projeção, União, Interseção, Diferença, Produto Cartesiano e Junção usados para criar relacionamento entre tabelas.

A álgebra relacional desempenha um papel fundamental ao fornecer uma base teórica sólida para o funcionamento dos bancos de dados relacionais. Em especial, ela serve como alicerce para a linguagem de consulta SQL, amplamente utilizada na gestão e manipulação de dados em sistemas de banco de dados relacionais. É amplamente utilizada para o ensino de banco de dados e linguagens de consulta, por apresentar conceitos importantes de relacionamento entre tabelas, comumente aplicado nas fases iniciais destes cursos

As características matemáticas deste tema estão fundamentadas especialmente na teoria de conjuntos. O conjunto básico de operações empregadas no modelo relacional é denominado como álgebra relacional.

O ilustríssimo Shamkant B. Navathe destaca três razões para reconhecermos a importância dessas operações: primeiro, ela oferece um alicerce formal para as operações do modelo relacional; segundo, e talvez mais importante, ela é usada como base para a implementação e otimização de consultas nos módulos de otimização e processamento de consulta, que são partes integrais dos sistemas de gerenciamento de banco de dados relacional (SGBDRs); terceiro, seus conceitos são incorporados na linguagem de consulta padrão SQL para SGBDRs (NAVATHE, 2010).

Por meio dessas operações, um indivíduo pode definir as solicitações básicas de recuperação como expressões da álgebra relacional. O resultado de uma recuperação ou consulta é uma nova relação.

Ainda que a álgebra estabeleça um conjunto de operações para o modelo relacional, o cálculo relacional proporciona uma linguagem declarativa de nível superior para especificar

consultas relacionais. Geralmente, a álgebra relacional é vista como uma importante parcela componente do modelo de dados relacional. Pode-se dividir as operações em duas categorias: uma que engloba operações da teoria de conjuntos matemáticos; e outra composta por operações desenvolvidas especificamente para bancos de dados relacionais, denominadas operações relacionais.

Existem duas principais formas de avaliar um conjunto específico de operações, são elas: Avaliação Materializada: nesta abordagem, cada operação da álgebra é materializada em uma relação temporária, caso haja necessidade, e posteriormente utilizada como entrada para a próxima operação. Essa abordagem é considerada a situação usual padrão no processamento de consultas. Avaliação em Pipeline: Em apenas um único passo, uma sequência de operações algébricas é realizada, em que cada tupla gerada por uma operação é encaminhada para a próxima operação. Dessa forma, todas as tuplas percorrem um canal (pipe) de operações, e apenas o resultado final do pipeline é materializado, caso haja necessidade. Essa abordagem impede a materialização de todo e qualquer resultado intermediário durante o processamento de uma consulta, e é um método que provê uma forma eficiente do uso de memória principal. Existem seis operações consideradas essenciais da álgebra relacional, elas são divididas em unárias (operações de projeção, seleção e rename), quando se aplicam a apenas uma relação e binárias (união, interseção, subtração, produto cartesiano, junção e divisão), quando se cria uma relação partindo de duas outras relações, dessa maneira, tem-se a união de todas as tuplas dessa nova relação

2.1.1 Seleção

$$\sigma_{\text{idade}>18}(\text{usuários})$$

Essa operação, representada pelo operador σ (sigma), recebe como entrada, uma tabela ou relação. A finalidade dela é selecionar um conjunto de tuplas que corresponda a um predicado (lista de condições) designado nos valores dos atributos, ou seja, obtém-se, por meio de uma relação, um conjunto de linhas que apresentam certas limitações.

2.1.2 Projeção

$$\pi_{\text{nome, sobrenome}}(\text{usuários})$$

Essa operação, representada pelo operador π (pi), cria um subconjunto contendo apenas as colunas indicadas com base na condição escolhida, ela acaba por excluir algumas colunas do resultado, o que por consequência reduz a quantidade de dados a serem analisados.

2.1.3 Renomeação

$$\rho_{\text{NovoNome}(A_1,A_2,...,A_n)}(\text{Operação})$$

Essa operação, representada pelo operador ρ (rô), é amplamente empregada em situações que envolvem expressões relacionais extensas. Através dela, torna-se possível alterar o nome de uma agregação para referenciá-la na mesma consulta. Esse recurso é útil quando se faz necessário acessar mais de uma tupla da mesma tabela de entrada para cada tupla gerada no resultado, além de dar mais sintaxe às operações.

2.1.4 União

$$R \cup S$$

Essa operação, representada pelo operador $\cup$ (união), é responsável por combinar as linhas das tabelas verticalmente. No entanto, essa operação só pode ser executada se as tabelas forem compatíveis para união, ou seja, se tiverem a mesma estrutura com mesmo número de colunas e com o domínio de cada atributo idêntico, resultando em um conjunto com linhas distintas.

2.1.5 Intersecção

$$R \cap S$$

Essa operação, representada pelo operador $\cap$ (intersecção), resulta em uma relação a partir de duas outras relações compatíveis, estabelecendo a interseção entre elas, resultando em uma nova relação com os registros em comum.

2.1.6 Diferença

$$R - S$$

Essa operação, representada pelo operador $-$ (diferença), é aplicada em estruturas com relações compatíveis, resultando em uma nova relação contendo as tuplas que pertençam à primeira relação e ausentes na segunda.

2.1.7 Produto Cartesiano

$$R \times S$$

Essa operação, representada pelo operador $\times$ (produto cartesiano), possibilita a combinação de informações de duas relações distintas, resultando em uma combinação de todas as tuplas entre as duas relações de entrada, a nova relação possui todos os atributos que compõem cada uma das tabelas.

2.1.8 Junção

$$R \bowtie_{A=B} S$$

Essa operação, representada pelo operador $\bowtie$ (junção), é uma ligação entre duas tabelas R e S, em que são combinadas com base em um ou mais campos em comum (A=B), sendo possível combinar as operações de seleção e produto cartesiano. Segundo Navathe há duas variações de junção: a equijunção, em que compara uma ou mais colunas especificadas para verificar a igualdade; junção Natural, que compara colunas com nomes idênticos, dispensando a necessidade de especificá-las.

2.1.9 Divisão

$$R \div S$$

Essa operação, representada pelo símbolo $\div$ (divisão), resultará basicamente em elementos da primeira tabela que estejam relacionados com todos os elementos da segunda tabela.

2.2 Serverless

Serverless (REDHAT, 2022) é uma forma de infraestrutura escalável e com um custo muito baixo, pois as máquinas não precisam ficar acionadas o tempo todo. Para o caso da nova plataforma de Algebra Relacional esse cenário é ideal, já que será possível atender às requisições dos usuários sem precisar manter máquinas ativas 100% do tempo, principalmente por não ser necessário fornecer suporte a longas transações atômicas neste ambiente de ensino.

Como modo de promover a plataforma, essa forma de hospedagem oferece centenas de milhares de requisições gratuitas, como apresentado nas Tabelas 1 e 2.

[1]

Serviço	**Requisições gratuitas por dia**
CloudFlare Pages	Ilimitadas
CloudFlare Workers	100,000
CloudFlare D1	5 milhões de leituras / 100,000 escritas

Tabela 1 – Quantidade de requisições gratuitas por dia usando Serverless da CloudFlare

(Fonte: o autor)

[2]

Serviço	**Custo mensal para absorver 1000 usuários**
Serverless	Gratuito
VPS	R$ 116,90
Cloud	R$ 474,93

Tabela 2 – Custo mensal para absorver 1000 usuários

(Fonte: o autor)

[1] Mais informações sobre a plataforma Serverless da Cloudflare podem ser encontradas em: <https://developers.cloudflare.com/workers/platform/pricing/> (Tabela 1)

[2] Mais informações sobre o comparativo de preços entre VPS e Cloud, foi usado como referência o plano VPS 08: <https://dokehost.com.br/> (Tabela 2)

2.3 Programação em Blocos

A programação em blocos é uma abordagem de ensino de programação que utiliza interfaces gráficas, onde os usuários podem montar programas juntando blocos de comandos, sem a necessidade de escrever código tradicional. Essa metodologia, inspirada em linguagens visuais como Scratch (MIT, 2007) e Blockly (GOOGLE, 2012).

Ao contrário dos métodos tradicionais de ensino de programação, que envolvem muitas vezes a escrita de código complexo desde o início, a programação em blocos oferece uma abordagem mais acessível e intuitiva. Os usuários podem criar programas manipulando blocos de comandos com funções específicas, como movimento, aparência, som e controle, de forma visual e interativa. Essa abordagem tem se mostrado vantajosa no ensino de programação por ter uma abordagem mais prática, de fácil compreensão e por ter um feedback imediato.

Neil Fraser, autor do artigo "Ten Things We've Learned from Blockly", observa que, para novos usuários, os blocos mais difíceis são os condicionais e os loops. Ele comenta a importância da escolha das cores e, na hora de construir o aplicativo de blocos, cita como é importante a diferenciação dos blocos na hora de conceber os grupos lógicos (FRASER, 2015).

Erik Pasternak, Rachel Fenichel e Andrew N. Marshall, autores do artigo "Tips for Creating a Block Language with Blockly", discutem a importância de considerar o público-alvo ao criar uma linguagem de blocos. Eles enfatizam que as linguagens de blocos para estudantes do ensino médio que controlam um robô serão muito diferentes das linguagens para um profissional de TI que configura um roteador. Destacam que é essencial considerar o que os usuários devem obter ao usar o aplicativo, a idade e o nível de leitura do público-alvo, entre outros fatores (PASTERNAK; FENICHEL; MARSHALL, 2017). Portanto, a programação em blocos oferece uma maneira acessível de se ensinar programação, por isto foi escolhida como pilar deste projeto para ensino de Algebra Relacional.

2.4 Trabalhos Relacionados

2.4.1 dbSnap

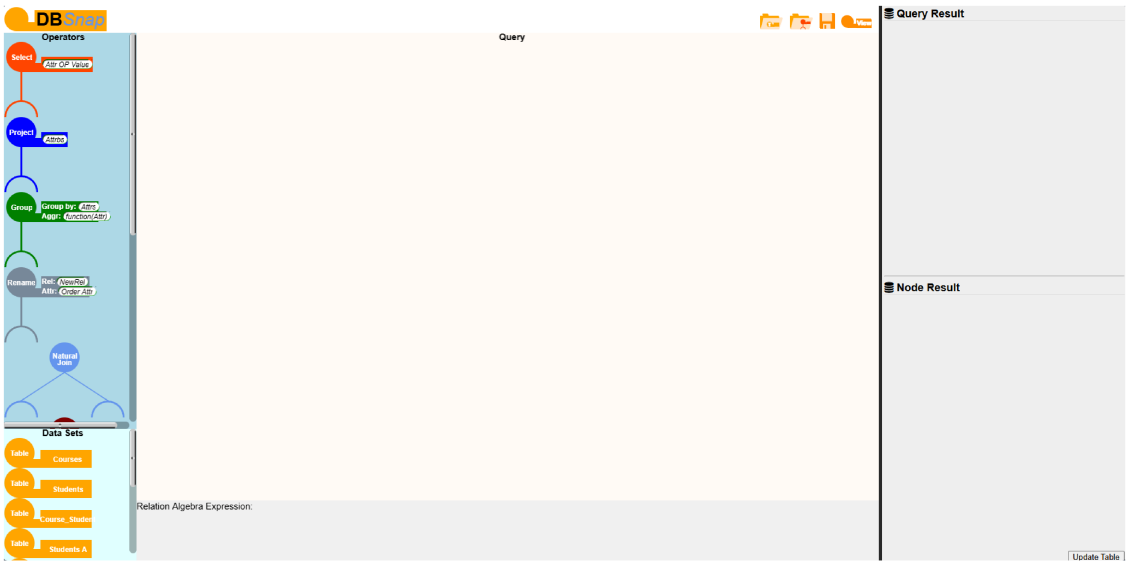

Figura 1 – `Dashboard dbSnap`

(Fonte: o autor)

O presente trabalho tem como inspiração a plataforma dbSnap (SILVA; CHON, 2015; SILVA; LOZA; RAZENTE, 2022a; SILVA; LOZA; RAZENTE, 2022b). Trata-se de uma importante ferramenta online que permite a criação de consultas de banco de dados, especialmente aquelas relacionadas à álgebra relacional, através do encaixe de blocos.

Suas principais características são: método de consulta de banco de dados baseado em blocos de uso intuitivo e de fácil compreensão; resultados de consulta atualizados de forma dinâmica, conforme a consulta é modificada; acessível em qualquer dispositivo que possua um navegador de internet e incorporação de conjuntos de dados personalizados.

O dbSnap oferece um suporte completo para a construção intuitiva de consultas em banco de dados em forma de árvores, uma abordagem reconhecida como altamente eficaz para o ensino desse conceito. Além disso, um de seus principais atributos é a sua dinamicidade, pois exibe os resultados da consulta à medida que esta é elaborada, permitindo que o usuário analise os resultados intermediários de qualquer nó de consulta a qualquer momento. Amplamente disponível de forma pública, o dbSnap tem como maior objetivo provocar uma mudança significativa na forma de aprendizado de base de dados, semelhante ao dos sistemas precursores pautados em blocos no aprendizado da programação tradicional.

Uma das principais diferenças entre as plataformas é o fato de DBSap possuir uma interface rústica sem integração com SGBD, e com processamento em memória. Já a

plataforma do presente projeto possui uma interface mais dinâmica e atual, possuindo uma integração com SGBD e *backend*.

2.4.2 DB-Learn

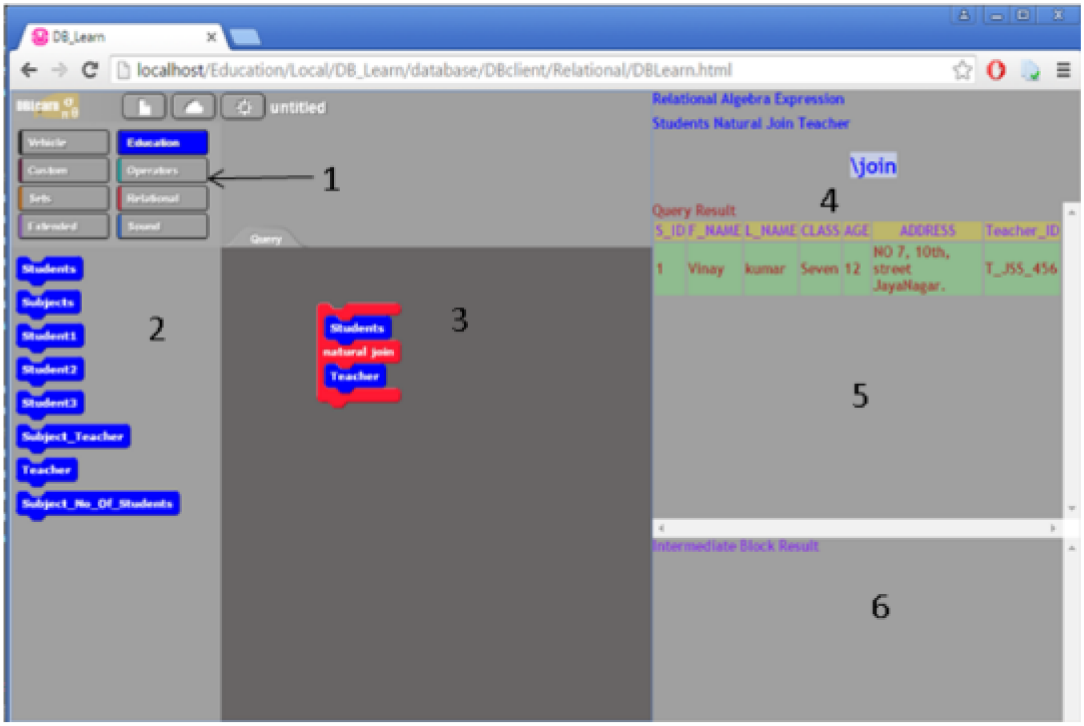

Figura 2 – `Dashboard DB-Learn`

(Fonte: VINAYAKUMAR; SOMAN; MENON, 2018b)

Outro trabalho interessante que segue uma abordagem semelhante é o DB-Learn (VINAYAKUMAR; SOMAN; MENON, 2018b). Este projeto de programação em blocos foi criado para aprender os fundamentos da álgebra relacional. Com o DB-Learn, os usuários podem simplesmente arrastar e soltar blocos para criar consultas de álgebra relacional de maneira intuitiva. A estrutura do DB-Learn é construída com base no CT-Blocks (VINAYAKUMAR; SOMAN; MENON, 2018a), uma linguagem de programação baseada em nuvem e blocos que reforça o pensamento computacional em todos os tipos de usuários.

Uma funcionalidade interessante do DB-Learn é a capacidade de adicionar som às atividades do projeto. Isso torna o processo de aprendizagem mais interativo, permitindo que os usuários incorporem efeitos sonoros ao executar scripts. Isso pode enriquecer a experiência do projeto e tornar o aprendizado mais envolvente para os iniciantes.

Capítulo 3

Desenvolvimento

Neste capítulo, serão apresentadas as tecnologias utilizadas na implementação da plataforma de ensino de Álgebra Relacional utilizando programação em blocos em infraestrutura serverless.

3.1 Infraestrutura

3.1.1 Ambiente de Desenvolvimento

Docker(OPENSOURCE, 2024) é uma plataforma de código aberto que permite a criação, o gerenciamento e a execução de aplicativos em contêineres. Os contêineres Docker são unidades leves e portáteis de software que encapsulam todo o ambiente necessário para executar um aplicativo, incluindo o código, as bibliotecas e as dependências. Eles oferecem consistência entre ambientes de desenvolvimento e produção, simplificando o processo de implantação e garantindo que os aplicativos funcionem da mesma forma em qualquer ambiente.

Docker Compose é uma ferramenta que permite definir e gerenciar aplicativos Docker multicontêineres. Com o Docker Compose, é possível descrever todo o ambiente de desenvolvimento em um arquivo YAML, incluindo serviços, redes e volumes necessários para a execução do aplicativo. Ele simplifica o processo de configuração e inicialização de aplicativos Docker complexos, garantindo consistência e reprodutibilidade em diferentes ambientes.

Para o ambiente de desenvolvimento, foi adotada uma abordagem baseada em contêineres usando Docker. Isso permitiu uma configuração simplificada e consistente do ambiente de desenvolvimento em diferentes máquinas. O Docker Compose foi configurado para subir três serviços principais:

- Aplicação React(WALKE, 2013): O *frontend* da plataforma foi desenvolvido utilizando React. Com o Docker Compose, foi possível configurar um ambiente de desenvolvimento local para a aplicação web.

- PostgreSQL(STONEBRAKER, 2022): Para armazenamento de dados, foi utilizado o PostgreSQL, que é um sistema de gerenciamento de banco de dados relacional de código aberto. Este serviço foi configurado para persistir os dados da aplicação durante o desenvolvimento.

- Backend com Node.js(DAHL, 2009): O *backend* da plataforma foi implementado usando Node.js. O Docker Compose permitiu a configuração de um ambiente local para o *backend*, facilitando o desenvolvimento e depuração.

Essa abordagem de contêineres proporcionou um ambiente de desenvolvimento isolado e simples de configurar, garantindo consistência entre os ambientes de diferentes desenvolvedores.

3.1.2 Ambiente de Produção

Serverless é um modelo de computação em nuvem que permite aos desenvolvedores construir e executar aplicativos sem se preocupar com a infraestrutura subjacente. Nele, os provedores de nuvem são responsáveis por provisionar, escalar e gerenciar os recursos necessários para executar os aplicativos, como servidores e ambientes de execução. Os desenvolvedores podem se concentrar exclusivamente no desenvolvimento de código, enquanto os provedores de nuvem lidam com a complexidade operacional e garantem alta disponibilidade e escalabilidade automática.

Para o ambiente de produção, foram adotadas tecnologias baseadas em serviços em serverless para garantir escalabilidade, disponibilidade e baixa latência:

- Cloudflare Pages(CLOUDFLARE, 2022b): A aplicação React foi hospedada e servida usando o Cloudflare Pages, uma plataforma de hospedagem de sites estáticos que oferece implantação contínua direta a partir de repositórios Git.

- Banco de Dados Serverless Cloudflare D1(CLOUDFLARE, 2022a): Para armazenamento de dados de forma escalável e gerenciável, foi utilizado o Cloudflare D1, que é um serviço de banco de dados serverless oferecido pela Cloudflare.

- Cloudflare Workers(CLOUDFLARE, 2022c): Para implementar a lógica de negócios do *backend*, bem como a comunicação com o banco de dados, foram utilizados os Cloudflare Workers, que são funções serverless executadas na borda da rede da Cloudflare.

Essa arquitetura baseada em serviços de nuvem permitiu uma infraestrutura escalável, com capacidade de lidar com um número razoável de usuários simultâneos de forma eficiente e econômica. Além disso, a presença de serviços serverless reduziu a complexidade operacional e eliminou a necessidade de gerenciar infraestrutura de servidor tradicional.

3.2 Tecnologias utilizadas no Frontend

3.2.1 React

React (WALKE, 2013) é uma biblioteca JavaScript de código aberto mantida pelo Facebook, utilizada para construir interfaces de usuário (UI) interativas. Ele oferece uma abordagem baseada em componentes, permitindo o desenvolvimento modular e reutilizável de interfaces de usuário complexas. Sua popularidade e vasta comunidade de desenvolvedores contribuem para sua robustez e constante evolução.

3.2.2 Blockly

Blockly(GOOGLE, 2012) é uma biblioteca de código aberto desenvolvida pelo Google para a criação de editores visuais de programação em blocos. Ela é amplamente utilizada em ambientes educacionais para ensinar programação de forma visual e intuitiva. A integração do Blockly no *frontend* da plataforma proporciona uma experiência de programação acessível e interativa para os usuários.

3.2.3 Cloudflare Pages

Cloudflare Pages(CLOUDFLARE, 2022b) é uma plataforma de hospedagem de sites estáticos oferecida pela Cloudflare. Ela permite hospedar e servir sites diretamente de repositórios Git, oferecendo implantação contínua e rápida entrega de conteúdo estático aos usuários finais. A integração do Cloudflare Pages no ambiente de produção da plataforma garante uma experiência de usuário confiável e de alto desempenho.

3.3 Tecnologias utilizadas no Backend

3.3.1 Node.js

Node.js (DAHL, 2009) é um ambiente de execução JavaScript assíncrono e baseado em eventos, projetado para criar aplicativos de rede escaláveis. Ele permite que os desenvolvedores usem JavaScript tanto no lado do usuário quanto no servidor, unificando o desenvolvimento de aplicativos web. Sua natureza assíncrona o torna ideal para operações de entrada/saída intensivas, como comunicação de rede e acesso a banco de dados.

3.3.2 Cloudflare Workers

Cloudflare Workers(CLOUDFLARE, 2022c) é uma plataforma serverless oferecida pela Cloudflare, que permite aos desenvolvedores executar funções JavaScript na borda da rede da Cloudflare. Essas funções são executadas em locais próximos aos usuários finais,

garantindo baixa latência e alta disponibilidade. A integração dos Cloudflare Workers no *backend* da plataforma facilita a implementação de lógica de negócios e a comunicação com o banco de dados de forma eficiente e escalável.

3.4 Tecnologias utilizadas no Banco de Dados

3.4.1 Cloudflare D1

Cloudflare D1(CLOUDFLARE, 2022a) é um serviço de banco de dados serverless oferecido pela Cloudflare. Ele permite armazenar e consultar dados de forma escalável e gerenciável, sem a necessidade de provisionamento ou configuração de infraestrutura de banco de dados, por meio da linguagem SQL ou noSQL. A integração do Cloudflare D1 na arquitetura da plataforma proporciona um armazenamento eficiente de dados, garantindo alta disponibilidade e baixa latência para os usuários finais.

3.4.2 PostgreSQL

PostgreSQL (STONEBRAKER, 2022) é um sistema de gerenciamento de banco de dados relacional de código aberto amplamente utilizado. Ele oferece recursos avançados de conformidade com padrões, desempenho robusto e extensibilidade, tornando-se uma escolha popular para muitos aplicativos. Com suporte para SQL completo e transações ACID, o PostgreSQL é adequado para cargas de trabalho de todos os tamanhos, desde pequenos projetos até aplicativos empresariais de missão crítica.

3.5 Implementação do Frontend

3.5.1 Arquivo `app.tsx`

Neste arquivo, encontra-se o componente principal da aplicação React, responsável por gerenciar a interface do usuário e a lógica de interação com o Blockly.

3.5.1.1 Componente `App`

O componente `App` é o ponto de entrada da aplicação. Ele define a estrutura da página, incluindo a área de trabalho do Blockly, os botões para executar consultas e limpar resultados, e exibir o SQL gerado e os resultados das consultas.

O componente `App` possui os seguintes estados:

- ❏ `queryResult`: Armazena os resultados da consulta SQL.
- ❏ `TableTitle`: Armazena o título da tabela.
- ❏ `tableData`: Armazena os dados da tabela.
- ❏ `generatedSql`: Armazena o SQL gerado pelo Blockly.
- ❏ `tutorialModal`: Controla a exibição do modal de tutorial.
- ❏ `blurModal`: Controla o blur na tela, quando o modal é acionado.

3.5.1.2 Componentes Modais

```
1  const TutorialModal = ({ show, handleClose, handleVerTutorial }) => (
2    <Modal show={show} onHide={handleClose} backdrop="static"
         keyboard={false} centered>
3      <Modal.Header closeButton>
4        <Modal.Title>Tutorial</Modal.Title>
5      </Modal.Header>
6      <Modal.Body>
7        <p>Deseja fazer o tutorial de como usar o Algebra Blocks?</p>
8      </Modal.Body>
9      <Modal.Footer>
10       <Button variant="danger" onClick={handleClose}>
11         Pular Tutorial
12       </Button>
13       <Button variant="primary" onClick={handleVerTutorial}>
14         Ver Tutorial
15       </Button>
16     </Modal.Footer>
17   </Modal>
18 );
```

Listing 3.1 – Componente TutorialModal

```
const BlurModal = ({ show, handleClose, title, text }) => (
  <Modal show={show} onHide={handleClose} backdrop="static"
     keyboard={false} centered>
    <Modal.Header closeButton>
      <Modal.Title>{title}</Modal.Title>
    </Modal.Header>
    <Modal.Body>
      <p>{text}</p>
    </Modal.Body>
    <Modal.Footer>
      <Button variant="secondary" onClick={handleClose}>
        Fechar
      </Button>
    </Modal.Footer>
  </Modal>
);
```

Listing 3.2 – Componente blurModal

Os componentes `TutorialModal` e `BlurModal` são responsáveis por exibir modais na interface do usuário. O `TutorialModal` da Figura 5 é exibido quando a aplicação é carregada pela primeira vez, oferecendo ao usuário a opção de ver um tutorial. O `BlurModal` da Figura 6 é utilizado para aplicar blur na tela e também apresentar informações relevantes aos usuários.

Função `handleTutorialModalClose()` e `handleBlurModalClose()`

```
  const handleTutorialModalClose = () => setTutorialModal({
     ...tutorialModal, show: false });
  const handleBlurModalClose = () => setBlurModal({ ...blurModal, show:
     false });
```

Listing 3.3 – handleTutorialModalClose() e handleBlurModalClose()

Essas funções da Figura 7 possuem o papel de gerenciar o estado de ativação dos modais de tutorial e de blur.

3.5.1.3 Funções

Função `executeQuery()`

```
const executeQuery = async (query) => {
  try {
    const { data } = await axios.post(process.env.BACKEND_URL, { query
       }, { headers: { 'Content-Type': 'application/json' } });
    return data;
  } catch (error) {
    throw new Error('Ocorreu um erro na execucao da consulta.');
  }
};
```

Listing 3.4 – executeQuery()

Essa função da Figura 8 é responsável por enviar uma consulta SQL para o *backend* e retornar os resultados. Ela utiliza a biblioteca Axios para fazer a requisição HTTP POST para o *backend*. Ela possui uma desestruturação do objeto de resposta e também possui um tratamento de erro com o bloco try-catch.

Função `renderTable()`

```
const renderTable = (data, containerClassName) => {
  if (!data || data.length === 0) {
    return null;
  }

  const headers = Object.keys(data[0]);

  return (
    <div className={`table-container ${containerClassName}`}>
      <table className="table table-bordered table-striped">
        <thead className="thead-dark">
          <tr>
            {headers.map((header, index) => (
              <th key={index}>{header}</th>
            ))}
          </tr>
        </thead>
        <tbody>
          {data.map((row, rowIndex) => (
            <tr key={rowIndex}>
              {headers.map((header, colIndex) => (
                <td key={colIndex}>{row[header]}</td>
              ))}
            </tr>
          ))}
        </tbody>
      </table>
```

```
28      </div>
29    );
30  };
```

Listing 3.5 – renderTable()

Essa função da Figura 9 gera a representação HTML de uma tabela a partir dos dados fornecidos. Ela recebe os dados e uma classe de contêiner como parâmetros e retorna o código HTML correspondente à tabela.

Função `runQuery()`

```
1   const runQuery = async () => {
2     try {
3       const workspace = Blockly.getMainWorkspace();
4
5       if (!validateOperationBlock()) {
6         return;
7       }
8
9       const resultStrings =
            javascriptGenerator.workspaceToCode(workspace);
10
11      const result = resultStrings.split('&');
12      let query = result[0];
13      let tableTitle = result[1];
14
15      const table = criarConsultaPadrao(query);
16
17      const [queryResult, tableData] = await
            Promise.all([executeQuery(query), executeQuery(table)]);
18
19      setGeneratedSql(query);
20      setQueryResult(queryResult);
21      setTableData(tableData);
22      setTableTitle(tableTitle);
23    } catch (e) {
24      setBlurModal({ show: true, title: 'Erro', text: e.message });
25    }
26  };
```

Listing 3.6 – runQuery()

Essa função da Figura 10 é chamada quando o usuário clica no botão "Iniciar Consulta". Ela obtém o código gerado pelo Blockly, executa a consulta SQL correspondente no *backend* e atualiza os estados da aplicação com os resultados.

Função `validateOperationBlock()`

```
1  const validateOperationBlock = () => {
2    const workspace = Blockly.getMainWorkspace();
3    const operationBlocks = workspace.getAllBlocks(false);
4
5    if (operationBlocks.length > 1) {
6      setBlurModal({ show: true, title: 'Erro', text: 'Voce so pode
           adicionar uma operacao por vez.' });
7      return false;
8    }
9
10   return true;
11 };
```

Listing 3.7 – validateOperationBlock()

Essa função da Figura 11 tem o papel de validar que apenas 1 bloco está no workspace da aplicação, esperamos no futuro não ser mais necessária, podendo o usuário adicionar mais blocos no workspace.

Função `criarConsultaPadrao()`

```
1  const criarConsultaPadrao = (sql) => {
2    try {
3      const tabela = extrairTabelaDaSQL(sql);
4      return `SELECT * FROM ${tabela}`;
5    } catch (error) {
6      throw new Error('Nao foi possivel criar a consulta padrao.');
7    }
8  };
```

Listing 3.8 – criarConsultaPadrao()

Essa função da Figura 12 tem o papel de realizar a primeira consulta, executando um SELECT * em todos os dados da tabela para trazer ao usuário uma visão de todas as linhas da tabela selecionada.

Função `extrairTabelaDaSQL()`

```
1  const extrairTabelaDaSQL = (sql) => {
2    const regex = /\bFROM\s+([^\s;]+)/i;
3    const match = sql.match(regex);
4
5    if (match && match[1]) {
6      return match[1];
```

```
7     } else {
8       throw new Error('Nao foi possivel extrair o nome da tabela.');
9     }
10  };
```

Listing 3.9 – extrairTabelaDaSql()

Essa função da Figura 13 tem o papel de extrair o nome da tabela do SQL, é aplicado um Regex para conseguir extrair esta string.

Função `clearResult()`

```
1   const clearResults = () => {
2     setQueryResult([]);
3     setTableTitle('');
4     setTableData([]);
5     setGeneratedSql('');
6   };
```

Listing 3.10 – clearResult()

Essa função da Figura 14 tem o papel de limpar os estados anteriores, permitindo que o usuário possa remover as consultas ativas e executar novas consultas no futuro.

3.5.2 Arquivo `getDefaultToolbox.ts`

Esse arquivo define o conjunto de blocos disponíveis para o Blockly, bem como suas funcionalidades e comportamentos.

3.5.2.1 Blocos SQL

Faz-se necessária a análise dos principais blocos e sua lógica interna:

1. `sql_selection`: Este bloco representa a operação de seleção em uma consulta SQL, onde o usuário pode especificar um atributo, um operador de comparação e um valor. A lógica JavaScript associada valida os campos fornecidos pelo usuário com base nas tabelas permitidas e gera a cláusula WHERE correspondente na consulta SQL.

2. `sql_projection`: Este bloco representa a operação de projeção em uma consulta SQL, onde o usuário pode selecionar um atributo para ser projetado. A lógica associada valida os campos fornecidos e gera a cláusula SELECT correspondente na consulta SQL.

3. `sql_groupBy`: Este bloco representa a operação de agrupamento em uma consulta SQL, onde o usuário pode especificar um atributo para agrupar os resultados. A

lógica associada valida os campos fornecidos e gera a cláusula GROUP BY correspondente na consulta SQL.

4. `sql_rename`: Este bloco representa a operação de renomeação em uma consulta SQL, onde o usuário pode renomear um atributo. A lógica associada valida os campos fornecidos e gera a cláusula AS correspondente na consulta SQL.

5. `sql_naturalJoin`: Este bloco representa a operação de junção natural em uma consulta SQL, onde o usuário pode especificar duas tabelas para serem unidas. A lógica associada gera a cláusula NATURAL JOIN correspondente na consulta SQL.

6. `sql_crossJoin`: Este bloco representa a operação de produto cartesiano em uma consulta SQL, onde o usuário pode especificar duas tabelas para serem unidas. A lógica associada gera a cláusula CROSS JOIN correspondente na consulta SQL.

7. `sql_union`: Este bloco representa a operação de união em uma consulta SQL, onde o usuário pode especificar duas tabelas para serem unidas. A lógica associada gera a cláusula UNION correspondente na consulta SQL.

8. `sql_intersect`: Este bloco representa a operação de interseção em uma consulta SQL, onde o usuário pode especificar duas tabelas para serem unidas. A lógica associada gera a cláusula INTERSECT correspondente na consulta SQL.

9. `sql_difference`: Este bloco representa a operação de diferença em uma consulta SQL, onde o usuário pode especificar duas tabelas para serem unidas. A lógica associada gera a cláusula EXCEPT (ou MINUS) correspondente na consulta SQL.

Cada bloco é responsável por validar os campos fornecidos pelo usuário, garantindo que apenas atributos válidos sejam usados nas consultas SQL. Isso ajuda a evitar erros de sintaxe e consultas inválidas.

Além disso, a função `getDefaultToolbox` retorna a configuração padrão do Blockly, organizando os blocos SQL em categorias para facilitar a utilização e compreensão por parte do usuário.

3.5.2.2 Gerenciamento de Entrada de Dados e Tratamento de Exceção

Um aspecto fundamental da funcionalidade dos blocos é o gerenciamento da entrada de dados fornecida pelo usuário e o tratamento de exceções para lidar com cenários inesperados ou erros de validação.

Validação de Campos

Cada bloco define campos de entrada que permitem ao usuário especificar valores relevantes para a operação SQL. Por exemplo, no bloco `sql_selection`, os campos de

entrada incluem o atributo, o operador de comparação e o valor a ser selecionado na consulta. Antes de gerar o código SQL correspondente, os campos fornecidos pelo usuário são validados para garantir que sejam válidos e adequados para a operação.

Tratamento de Exceções

Se ocorrerem erros durante a validação dos campos ou na geração do código SQL, são lançadas exceções para alertar o usuário sobre o problema e fornecer informações sobre como corrigi-lo. Por exemplo, se o usuário especificar um atributo que não é permitido para a tabela selecionada, uma exceção será lançada indicando o erro e sugerindo os atributos válidos que podem ser usados. O tratamento adequado de exceções é essencial para melhorar a usabilidade e a experiência do usuário, fornecendo feedback claro e útil sobre qualquer problema que possa surgir durante a criação das consultas SQL.

Os blocos definidos incluem operações como seleção, projeção, agrupamento, renomeação, junção natural, junção cruzada, união, interseção e diferença. Cada bloco tem sua própria lógica de geração de SQL e validação de campos.

Função `blocoSql()`

```
Blockly.Blocks['sql_operation'] = {
  init: function () {
    this.appendDummyInput().appendField('OPERACAO');
    this.setInputsInline(true);
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(210);
    this.setTooltip('Bloco base para operacoes SQL');
    this.setHelpUrl('');
  },
  onchange: function () {
    const workspace = this.workspace;

    if (workspace) {
      const blocks = workspace.getAllBlocks();
      const operationBlocks = blocks.filter((block) =>
          block.type.startsWith('sql_') && block.type !==
          'sql_operation');

      isOperationPresent = operationBlocks.length > 0;
    }
  }
};
```

Listing 3.11 – blocoSql()

Essa função da Figura 15 tem o papel de gerar o componente Blockly de cada consulta da Algebra Relacional, neste caso é o bloco base, onde podemos customizar adicionando: labels, dropdowns, cores, etc.

Função `JavaScriptGenerator()`

```
javascriptGenerator['sql_selection'] = function (block) {
  const field1 = block.getFieldValue('FIELD1');
  const field2 = block.getFieldValue('FIELD2');
  const field3 = block.getFieldValue('FIELD3');
  const tableVar = block.getFieldValue('TABLE_VAR');

  if (!tableVar) {
    return '/* Variavel de tabela ausente */';
  }

  const allowedFields = {
    estudantes: ['nome', 'sobrenome', 'id', 'idade'],
    cursos: ['id', 'nome'],
    professores: ['id', 'nome', 'sobrenome'],
    estudante_curso: ['estudante_id', 'curso_id'],
    professor_curso: ['professor_id', 'curso_id'],
    estudantes_A: ['nome', 'sobrenome', 'id', 'idade'],
    estudantes_B: ['nome', 'sobrenome', 'id', 'idade'],
    estudantes_C: ['nome', 'sobrenome', 'id', 'idade']
  };

  if (tableVar in allowedFields) {
    if (!allowedFields[tableVar].includes(field1)) {
      throw new Error(`Campo nao permitido para a tabela ${tableVar}.
         Use apenas ${allowedFields[tableVar].join(', ')}.`);
    }
  } else {
    throw new Error(`Tabela nao permitida: ${tableVar}`);
  }

  const code = `SELECT * FROM '${tableVar}' WHERE ${field1} ${field2}
     '${field3}'`;
  return `${code} & ${tableVar}`;
};
```

Listing 3.12 – JavaScriptGenerator()

Essa função da Figura 16 tem o papel de gerar o código JavaScript gerado por cada bloco, ela possui uma lógica de montagem da string SQL final que será enviada ao *backend* e também um controle de exceção por conta da seleção de tabelas ou atributos não permitidos. Cada bloco possui sua função geradora final.

Função `getDefaultToolbox()`

```
export const getDefaultToolbox = (): Blockly.utils.toolbox.ToolboxInfo
    => {
  return {
    kind: 'categoryToolbox',
    contents: [
      {
        kind: 'category',
        name: 'Operadores',
        colour: '259',
        contents: [
          {
            kind: 'block',
            type: 'sql_selection'
          },
          {
            kind: 'block',
            type: 'sql_projection'
          },
          {
            kind: 'block',
            type: 'sql_groupBy'
          },
          {
            kind: 'block',
            type: 'sql_rename'
          },
          {
            kind: 'block',
            type: 'sql_naturalJoin'
          }
        ]
      }
    ]
  };
};
```

Listing 3.13 – getDefaultToolbox()

A função `getDefaultToolbox` da Figura 17 retorna a configuração padrão do Blockly, que inclui os blocos SQL definidos, organizados em uma estrutura de categorias. Cada bloco é configurado com seus respectivos campos e funcionalidades, permitindo que o usuário crie consultas SQL de forma intuitiva e guiada.

3.6 Implementação do Backend

3.6.1 Arquivo `worker.ts`

Nesse arquivo, encontramos o código responsável por lidar com as requisições HTTP, executar consultas SQL no banco de dados e retornar os resultados para o *frontend*.

3.6.1.1 Função `fetch`

```
async fetch(request, env) {
  const { pathname } = new URL(request.url);

  try {
    const { query } = await request.json();
    const { results } = await env.DB.prepare(query).all();

    return new Response(JSON.stringify(results), {
      status: 200,
      headers: {
        "Content-Type": "application/json",
        ...corsHeaders,
      },
    });
  } catch (error) {
    if (error instanceof SyntaxError) {
      return new Response("Invalid JSON", {
        status: 400,
        headers: corsHeaders,
      });
    } else if (error instanceof Error) {
      return new Response("Database error", {
        status: 500,
        headers: corsHeaders,
      });
    }

    return new Response("Internal Server Error", {
      status: 500,
      headers: corsHeaders,
    });
  }
},
```

Listing 3.14 – fetch()

Esse arquivo da Figura 18 contém todo o código do *backend*. Ele apresenta uma estrutura simples: uma controladora responsável por validar requisições. Caso uma requisição

seja considerada válida, ela é encaminhada para um banco de dados localizado no contexto desta função Serverless. O resultado da consulta é então enviado de volta para o lado do usuário.

Por mais simples que este código seja, a estratégia de infraestrutura por trás dele, garante alta disponibilidade e escalabilidade.

Extração do Caminho da URL

O código extrai o caminho da URL da requisição usando o objeto `URL` do JavaScript.

Cabeçalhos CORS

Define os cabeçalhos CORS para permitir solicitações de origens diferentes.

Tratamento de Solicitações POST para /api/query

Se a requisição for uma solicitação POST para o caminho "/api/query"e o tipo de conteúdo for JSON, o código executa a seguinte lógica: extrai a consulta SQL do corpo da requisição JSON; executa a consulta SQL no banco de dados usando a conexão fornecida no objeto `env`; retorna os resultados da consulta em formato JSON.

Tratamento de Erros

O código inclui tratamento de erros para lidar com exceções durante o processamento da requisição. Se ocorrer um erro de sintaxe JSON, retorna uma resposta com status 400 e uma mensagem de erro "JSON inválido". Se ocorrer um erro no banco de dados, retorna uma resposta com status 500 e uma mensagem de erro "Erro de banco de dados". Se ocorrer outro tipo de erro, retorna uma resposta com status 500 e uma mensagem de erro "Erro interno do servidor".

Tratamento de Requisições Não Correspondidas

Se a requisição não corresponder a nenhum dos caminhos definidos, o código retorna uma resposta com status 404.

Esse código forma a base do *backend*, lidando com as requisições HTTP, executando consultas SQL no banco de dados e retornando os resultados para o *frontend*. Para integrar com o *frontend*, basta fazer solicitações HTTP para os endpoints adequados, como "/api/query", e processar os resultados recebidos.

Capítulo 4

Resultados

4.1 Plataforma

4.1.1 Dashboard Inicial

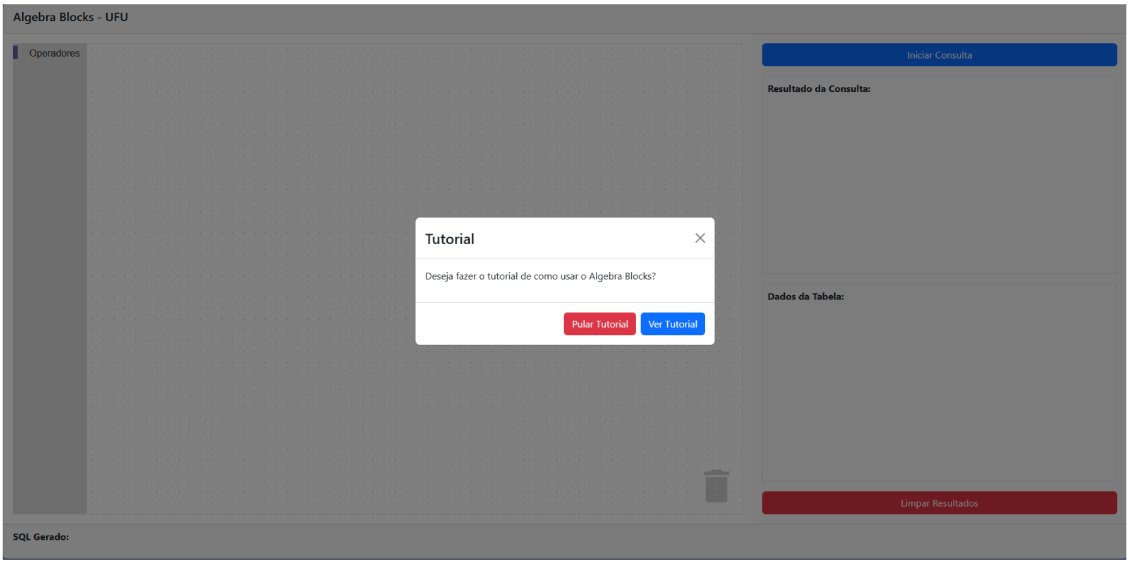

Figura 3 – `Dashboard Algebra Blocks`

(Fonte: o autor)

Ao acessar a plataforma, os usuários são recebidos com um *dashboard* inicial que serve como ponto de entrada para a experiência de usuário. Esse *dashboard* da Figura 3 apresenta uma interface limpa, onde os usuários têm a opção de assistir um tutorial para orientação ou pular diretamente para o *playground* de Álgebra Relacional. A inclusão de um modal oferecendo um vídeo tutorial visa atender às necessidades dos usuários que podem estar menos familiarizados com a plataforma e desejam aprender como utilizá-la de forma eficiente. Este tutorial direciona os usuários para um vídeo no YouTube, fornecendo uma maneira eficaz de aprender sobre os recursos e funcionalidades da plataforma.

4.1.2 Biblioteca de Blocos

Figura 4 – `Biblioteca de Blocos`

(Fonte: o autor)

No canto esquerdo do *dashboard*, os usuários têm acesso à biblioteca de blocos de Álgebra Relacional. Essa biblioteca da Figura 4 é uma coleção organizada de diferentes tipos de blocos que representam operações, como seleção, projeção, união, interseção e outras, comumente usadas na Álgebra Relacional. Cada bloco é visualmente identificável e pode ser arrastado para o *playground* central para a construção de consultas.

4.1.3 Playground de Álgebra Relacional

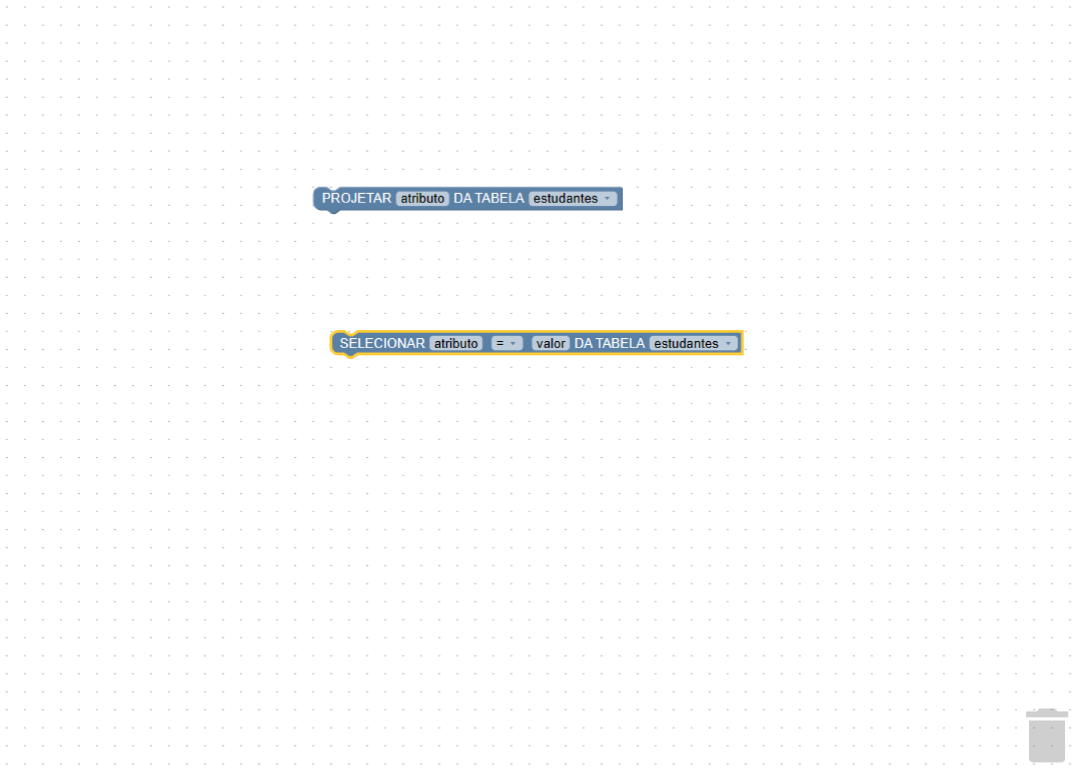

Figura 5 – `Playground`.

(Fonte: o autor)

Localizado no centro da tela do *dashboard*, o *playground* de Álgebra Relacional da Figura 5 é onde os usuários podem interagir diretamente com os blocos da biblioteca e montar suas consultas. Nesse espaço de trabalho intuitivo, os usuários podem arrastar e soltar os blocos selecionados da biblioteca. A interface do *playground* é projetada para ser responsiva e amigável, proporcionando uma experiência de construção de consultas sem atritos. Um ícone de lixeira é disponibilizado para caso o usuário deseje remover um bloco.

4.1.4 Visualização de Dados e Resultados

Figura 6 – `Visualização de Dados`

(Fonte: o autor)

À direita do *playground*, os usuários têm acesso a uma visualização em tempo real dos dados da tabela e dos resultados das consultas. Essa área da interface da Figura 6 é dividida em duas seções: a primeira exibe os dados, enquanto a segunda apresenta os resultados da consulta conforme são gerados. Essa visualização dinâmica permite aos usuários monitorar seus resultados e ajustar suas consultas conforme necessário.

4.1.5 Funcionalidades Adicionais

Além das funcionalidades básicas de construção e execução de consultas, o *playground* de Álgebra Relacional oferece recursos adicionais para melhorar a experiência do usuário. Por exemplo, botões de lixeira estão disponíveis para limpar os blocos do *playground*, permitindo aos usuários começar do zero ou realizar alterações em suas consultas. Além disso, botões dedicados para executar a consulta e limpar os resultados estão localizados abaixo da área de visualização.

4.2 Teste de Carga

O K6 é um framework de código aberto usado para testes de carga, desempenho e estresse. Ele permite simular o comportamento de milhares de usuários acessando uma aplicação simultaneamente, ajudando a identificar e resolver problemas de desempenho antes que afetem os usuários reais.

4.2.1 Resultado do teste de carga

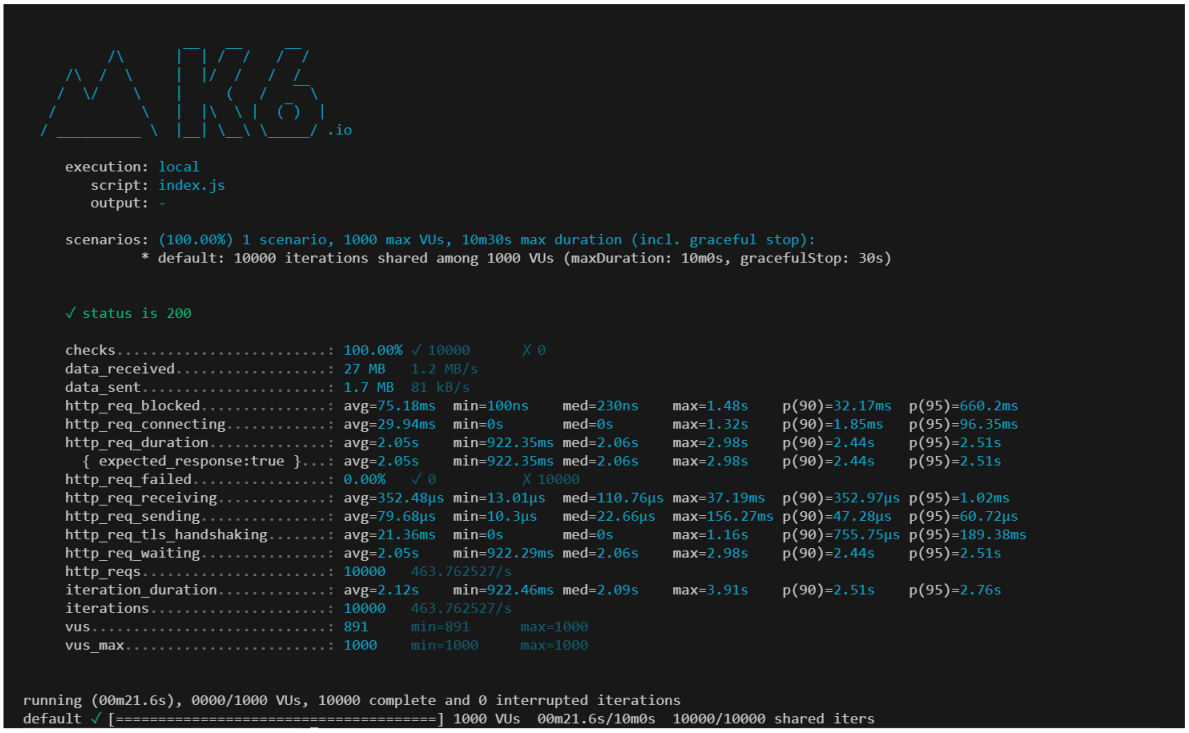

Figura 7 – `Resultado do teste de carga K6`

(Fonte: o autor)

Neste momento, conforme a Figura 7, verifica-se que a escolha de uma infraestrutura Serverless foi razoável, foi realizado um teste de carga no *backend* simulando um cenário de 1000 usuários simultâneos e cada um realizando 10 consultas na plataforma.

- ❏ `Taxa de sucesso das requisições`: 100%.
- ❏ `Taxa de erro das requisições`: 0%.
- ❏ `Quantidade de usuários simultâneos`: 1000.
- ❏ `Tempo médio de resposta da API`: 2.05s.
- ❏ `Duração do teste`: 21.6 segundos.
- ❏ `Quantidade total de requisições`: 10000 requisições.

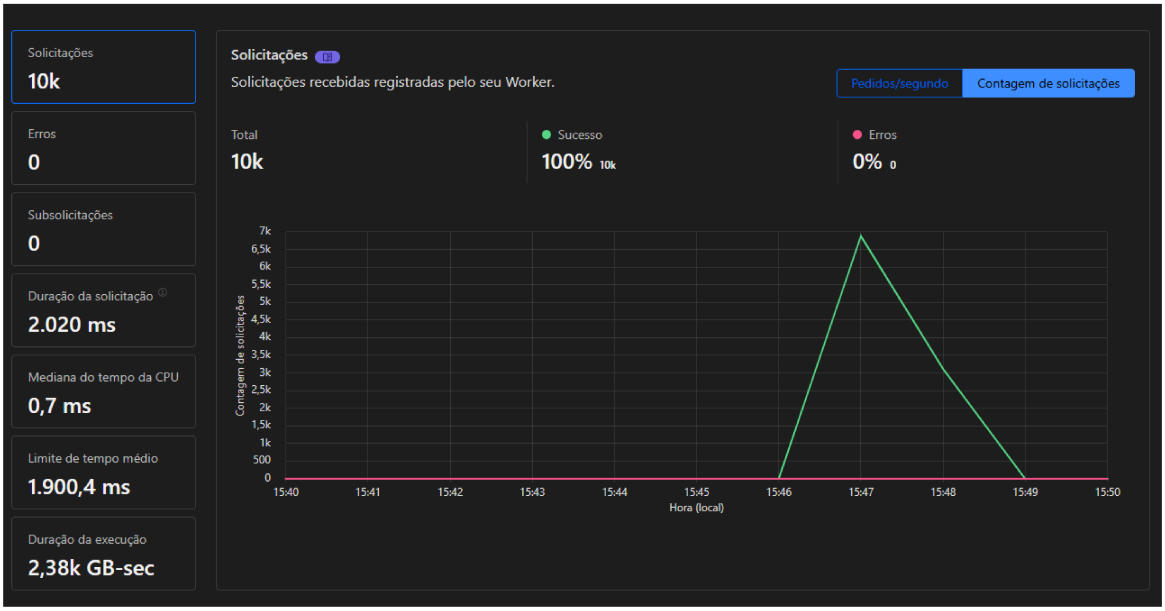

Figura 8 – `Painel da CloudFlare`

(Fonte: o autor)

Os resultados obtidos da Figura 8 durante o teste de carga confirmam a eficácia da infraestrutura Serverless adotada para 1000 usuários simultâneos. A API demonstrou uma capacidade de escalabilidade sob demanda, mantendo uma taxa de sucesso de 100% e uma taxa de erro de 0%. Isso evidencia a capacidade do sistema em lidar com uma quantidade razoável de usuários simultâneos, garantindo uma confiabilidade para os usuários. Além disso, ao manter o tempo médio de resposta da API em um nível aceitável, mesmo sob uma carga importante.

Portanto, os resultados positivos do teste de carga validam que a escolha de uma infraestrutura Serverless foi adequada para o momento do projeto.

Capítulo 5

Considerações Finais

Este trabalho teve como objetivo iniciar o desenvolvimento de uma plataforma para o ensino de álgebra relacional, utilizando a programação em blocos e uma infraestrutura completamente serverless. Através da utilização de tecnologias como React, Blockly, Node.js, Cloudflare Workers e Cloudflare D1, foi possível iniciar a construção de uma plataforma funcional e escalável, tanto no *frontend* quanto no *backend*.

Após a implementação da plataforma, foi possível observar que ela está ativa e funcional, permitindo realizar consultas utilizando blocos de programação de forma intuitiva e interativa. Além disso, a infraestrutura serverless garante que as consultas dos usuários sejam processadas de forma eficiente e confiável.

A escolha do serverless para hospedar a plataforma foi fundamental, pois sua capacidade de absorção de carga e baixo custo são uma ótima solução para uma aplicação educacional como a apresentada neste trabalho. Através do serverless, garantimos uma plataforma acessível, independentemente da demanda, e que os recursos sejam utilizados de forma eficiente.

Por mais que seja possível a utilização de uma aplicação *frontend* com os dados em memória, dar a certeza aos usuários de que suas consultas estão sendo processadas por um *backend* e banco de dados relacional fornece um entendimento mais completo de como uma aplicação real funciona.

5.1 Limitações do Trabalho

Apesar dos avanços alcançados no início da construção da plataforma, algumas limitações continuam presentes, sugerindo áreas para melhorias futuras:

1. **Encadeamento de Operadores:** Atualmente, a plataforma permite a construção de consultas SQL de forma isolada, mas não oferece suporte para encadear operadores, o que limita a complexidade das consultas que os usuários podem realizar. Incorporar essa funcionalidade seria um passo importante para permitir aos usuários explorar conceitos mais avançados de álgebra relacional de forma mais completa.

2. **Experiência do Usuário (UX):** Embora a plataforma ofereça uma interface interativa para a construção de consultas, a experiência do usuário ainda pode ser aprimorada. Isso inclui melhorar a usabilidade dos blocos de programação, fornecer feedback mais claro sobre erros e sugerir melhorias nas consultas. Investir em UX pode aumentar a eficácia da plataforma.

3. **Coleta de Feedback dos Usuários:** Para orientar o desenvolvimento futuro da plataforma, é essencial coletar feedback dos usuários. Isso pode incluir a realização de pesquisas, a análise de dados de uso e a solicitação de feedback direto dos usuários. Entender as necessidades e preferências dos usuários ajudará a direcionar os esforços de desenvolvimento e a garantir que a plataforma atenda às expectativas do público-alvo.

Escopo do Trabalho: É importante ressaltar que este trabalho representa apenas o início da construção da plataforma. Embora tenhamos alcançado uma infraestrutura funcional e algumas consultas básicas funcionando, há muito espaço para expansão e aprimoramento. É importante lemb rar que o desenvolvimento é um processo contínuo e esperamos abordar essas limitações em trabalhos futuros.

5.2 Trabalhos Futuros

Alguns possíveis trabalhos futuros incluem:

- ❑ Implementar a funcionalidade de encadeamento de operadores com vários blocos, permitindo aos usuários explorar conceitos mais avançados de álgebra relacional de forma mais completa.

- ❑ Introduzir recursos de feedback aprimorados para os usuários, fornecendo orientações específicas sobre erros e sugestões de melhorias em suas consultas.

- ❑ Desenvolver módulos adicionais de aprendizado, por exemplo, como geração de gráficos.

- ❑ Registrar logs e eventos das ações dos usuários para no futuro coletar informações relevantes, como, por exemplo, que tipo de consultas os usuários podem ter mais dificuldade.

Essas são apenas algumas ideias para possíveis melhorias e expansões da plataforma. Com o contínuo desenvolvimento e feedback dos usuários, é esperado que a plataforma evolua e se torne uma ferramenta ainda mais eficiente para o ensino de álgebra relacional.

5.3 Considerações Finais

O desenvolvimento da plataforma Algebra Blocks representa um início do que pode ser uma plataforma completa de ensino de bancos de dados. Ao longo deste trabalho, pudemos observar os benefícios do uso de uma infraestrutura serverless para hospedar a plataforma, garantindo sua escalabilidade e disponibilidade, além de manter os custos operacionais baixos. Esperamos que a plataforma Algebra Blocks possa contribuir significativamente para o ensino e aprendizado da álgebra relacional.

Referências

CLOUDFLARE. Cloudflare d1 documentation. In: . [s.n.], 2022. Disponível em: <https://developers.cloudflare.com/d1>.

_____. Cloudflare pages documentation. In: . [s.n.], 2022. Disponível em: <https://developers.cloudflare.com/pages>.

_____. Cloudflare workers documentation. In: . [s.n.], 2022. Disponível em: <https://developers.cloudflare.com/workers>.

CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, ACM, New York, NY, USA, v. 13, n. 6, p. 377–387, jun 1970. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/362384.362685>.

DAHL, R. Node.js — introduction to node.js. In: . [s.n.], 2009. Disponível em: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>.

FRASER, N. Ten things we've learned from blockly. In: **Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)**. USA: IEEE Computer Society, 2015. (BLOCKS AND BEYOND '15), p. 49–50. ISBN 9781467383677. Disponível em: <https://doi.org/10.1109/BLOCKS.2015.7369000>.

GOOGLE. Blockly | google for developers. In: . [s.n.], 2012. Disponível em: <https://developers.google.com/blockly/reference/js/blockly?hl=pt-br>.

HANSEN, A. K. et al. User-centered design in block-based programming: Developmental pedagogical considerations for children. In: **Proceedings of the The 15th International Conference on Interaction Design and Children**. New York, NY, USA: Association for Computing Machinery, 2016. (IDC '16), p. 147–156. ISBN 9781450343138. Disponível em: <https://doi.org/10.1145/2930674.2930699>.

MIT. Scratch - developers. In: . [s.n.], 2007. Disponível em: <https://scratch.mit.edu/developers>.

NAVATHE, S. B. Sistemas de banco de dados 6ª edição. In: . [S.l.]: Pearson Universidades, 2010.

OPENSOURCE. What is docker? In: . [s.n.], 2024. Disponível em: <https://opensource.com/resources/what-docker>.

PASTERNAK, E.; FENICHEL, R.; MARSHALL, A. N. Tips for creating a block language with blockly. In: **2017 IEEE Blocks and Beyond Workshop (B and B)**. [S.l.: s.n.], 2017. p. 21–24.

REDHAT. What is serverless? In: . [s.n.], 2022. Disponível em: <https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless>.

SILVA, Y. N.; CHON, J. Dbsnap: Learning database queries by snapping blocks. In: DECKER, A. et al. (Ed.). **Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE 2015, Kansas City, MO, USA, March 4-7, 2015**. ACM, 2015. p. 179–184. Disponível em: <https://doi.org/10.1145/2676723.2677220>.

SILVA, Y. N.; LOZA, A.; RAZENTE, H. L. Dbsnap 2: New features to construct database queries by snapping blocks. In: BECKER, B. A. et al. (Ed.). **ITiCSE 2022: Innovation and Technology in Computer Science Education, Dublin, Ireland, July 8 - 13, 2022, Volume 2**. ACM, 2022. p. 601–602. Disponível em: <https://doi.org/10.1145/3502717.3532156>.

_____. Dbsnap-eval: Identifying database query construction patterns. In: BECKER, B. A. et al. (Ed.). **ITiCSE 2022: Innovation and Technology in Computer Science Education, Dublin, Ireland, July 8 - 13, 2022, Volume 1**. ACM, 2022. p. 131–137. Disponível em: <https://doi.org/10.1145/3502718.3524822>.

STONEBRAKER, M. Postgresql: Developers. In: . [s.n.], 2022. Disponível em: <https://developers.cloudflare.com/pages>.

VINAYAKUMAR, R.; SOMAN, K.; MENON, P. Ct-blocks: Learning computational thinking by snapping blocks. In: **2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)**. [S.l.: s.n.], 2018. p. 1–7.

_____. Db-learn: Studying relational algebra concepts by snapping blocks. In: **2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)**. [S.l.: s.n.], 2018. p. 1–6.

WALKE, J. Quick start - react. In: . [s.n.], 2013. Disponível em: <https://react.dev/learn>.