SISTEMA DE BANCO DE DADOS PARA GERENCIAMENTO DE UMA CRIADORA DE JOGOS

Pedro Henrique Vieira Pinto
pedro.pinto@ufu.br

Emanoel Rodrigo Borges
emanoel.borges@ufu.br

1 Resumo

O desenvolvimento de jogos virtuais é uma indústria em constante crescimento, demandando sistemas eficientes para gerenciar informações como cadastro de jogos, plataformas, desenvolvedores, promoções e avaliações. Banco de dados relacionais são fundamentais para organizar esses dados, garantindo integridade, desempenho em consultas, e segurança no acesso. Este projeto propõe um modelo completo para uma empresa fictícia de jogos, utilizando MySQL, com recursos avançados como Stored Procedures, Triggers e Views para automatizar processos e validar regras de negócio.

Justificativa: A gestão de dados em planilhas é propensa a erros e pouco escalável para empresas de médio/grande porte.
Público-alvo: Empresas de desenvolvimento de jogos ou lojas de distribuição digital.

Palavras-chave: Banco de dados, jogos virtuais, MySQL, Stored Procedure, Triggers, Views.

2 Objetivos

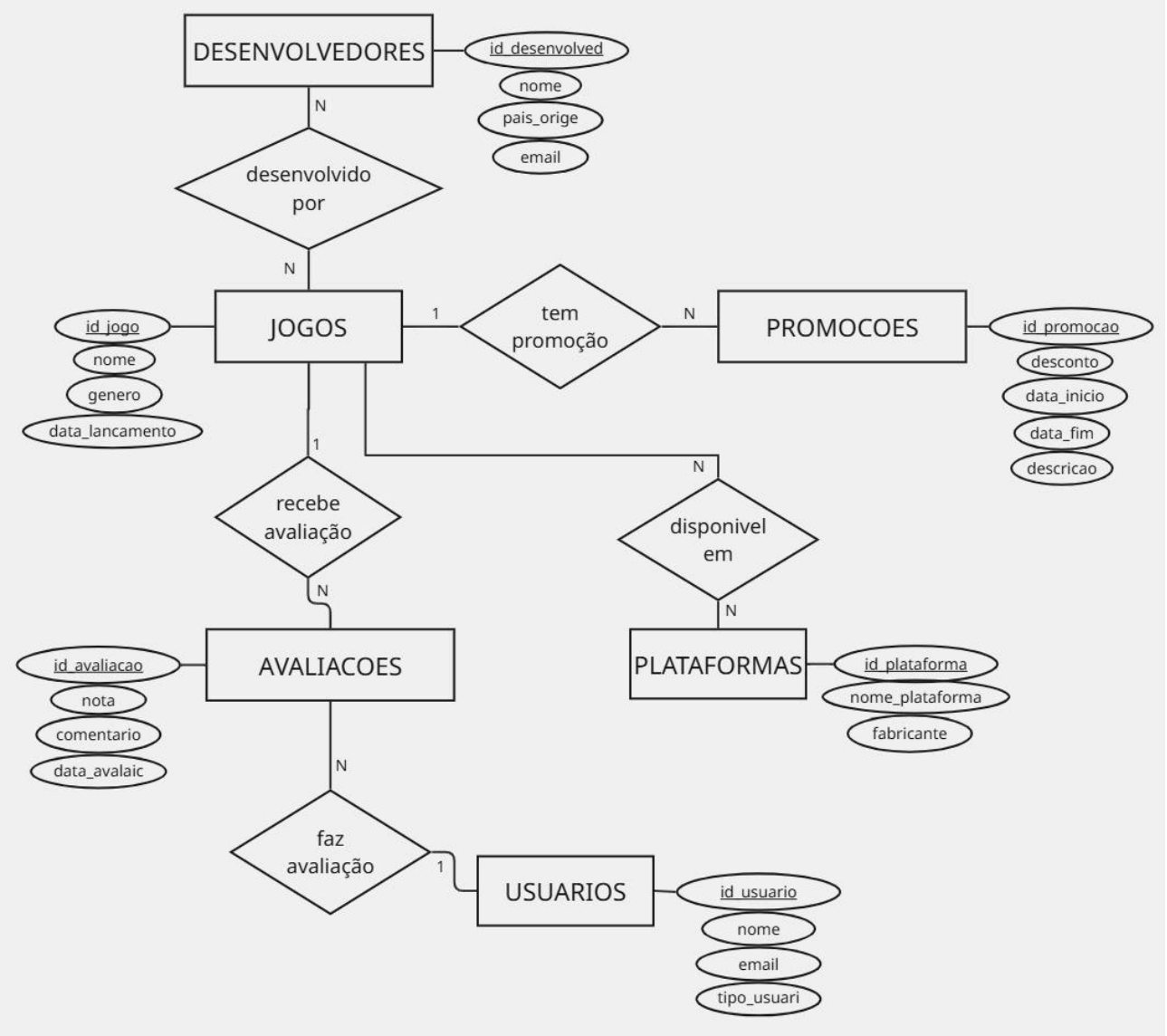
- Criar um banco de dados relacional para centralizar informações de jogos, desenvolvedores, plataformas e usuários.
- Implementar mecanismos de validação para garantir a qualidade dos dados. (Ex.: Notas de validação entre 1 e 10).
- Facilitar operações recorrentes por meio de Stored Procedures. (Ex.: Atualização de preços, criação de promoções).
- Gerar relatórios estratégicos via views. (Ex.: Média de avaliações por jogo, promoções ativas).
- Estabelecer perfis de acessos diferenciados (Admin, dev, avaliador, leitor) para segurança e controle de operações.

3 Destaques do Projeto

- ➔ 8 Tabelas e Relacionamentos
- ➔ 5 Triggers para validação em tempo real.
- ➔ 4 Stored Procedures para automação.
- ➔ 5 Views para relatórios estratégicos.
- ➔ Controle de acesso por perfil (Admin, Dev, etc.)

4 Modelagem

4.1 Diagrama Entidade Relacionamento



O diagrama mostra as principais entidades e relacionamentos do banco de dados da empresa criadora de jogos virtuais, conforme implementado no script SQL.

As entidades principais mostradas no diagrama são:

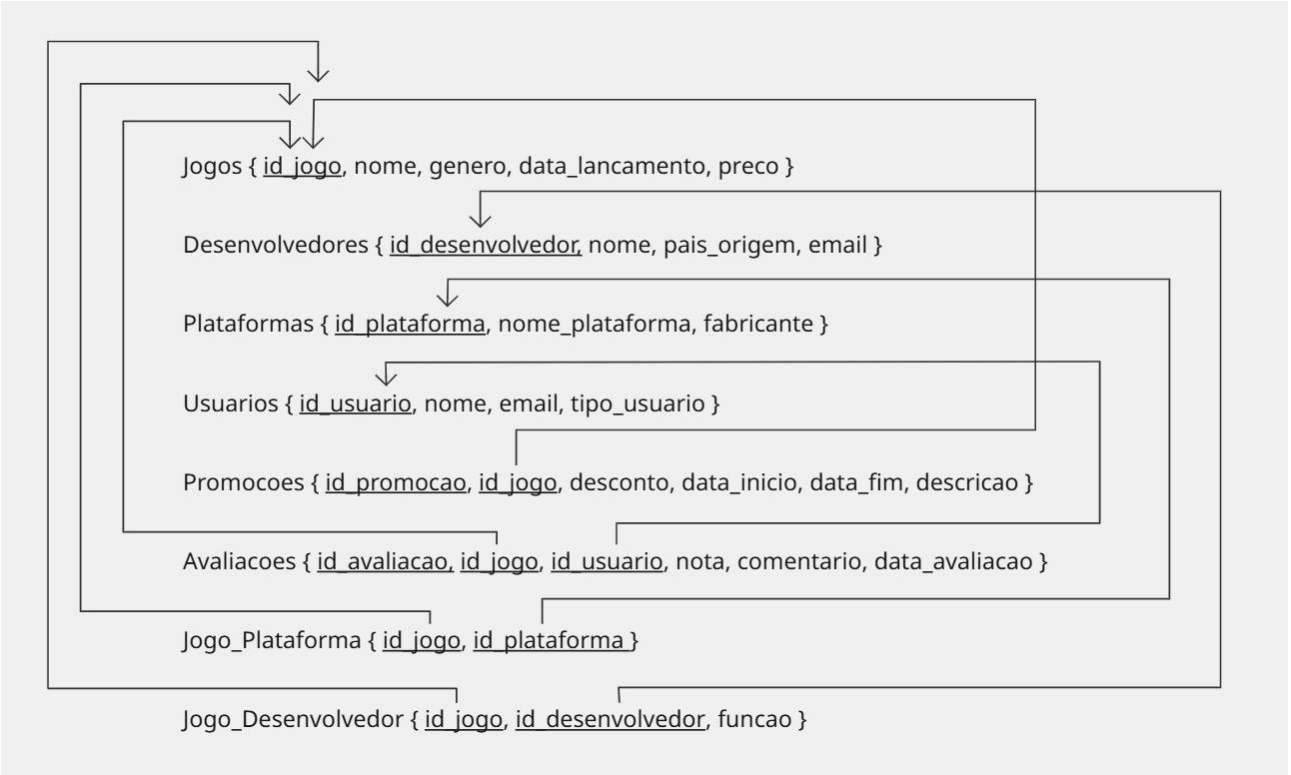
1. **Jogos**
   1.1 Atributos: id_jogo, nome, genero, data_lancamento, preco.
   1.2 Relacionamento:
      1.2.1 Recebe avaliações (1:N com Avaliações).
      1.2.2 Tem promoções (1:N com Promoções).
      1.2.3 Disponível em plataformas (N:N com Plataformas via tabela associativa Jogo_Plataforma).
      1.2.4 Desenvolvido por desenvolvedores (N:N com Desenvolvedores via tabela associativa Jogo_Desenvolvedor).

2. **Desenvolvedores**
   2.1 Atributos: id_desenvolvedor, nome, pais_origem, email.
   2.2 Relacionamento: Desenvolvedor jogos (N:N com Jogos).
3. **Plataformas**
   3.1 Atributos: id_plataforma, nome_plataforma, fabricante.
   3.2 Relacionamento: Hospedagem jogos (N:N com Jogos).
4. **Avaliações**
   4.1 Atributos: id_avaliacoes, nota, comentario, data_avaliacao.
   4.2 Relacionamentos:
      4.2.1 Feitas por usuários (N:1 com Usuarios).
      4.2.2 Referentes a jogos (N:1 com Jogos).
5. **Promoções**
   5.1 Atributos: id_promocao, desconto, data_inicio, data_fim, descricao.
   5.2 Relacionamento: Aplicadas a jogos (N:1 com Jogos).
6. **Usuários**
   6.1 Atributos: id_usuario, nome, email, tipo_usuario.
   6.2 Relacionamento: Fazem avaliações (1:N com Avaliações).

Cardinalidade:

- 1:N - Um jogo pode ter várias avaliações/promoções, mas cada avaliação/promoção pertence a um único jogo.
- N:N - Jogos podem estar em múltiplas plataformas e ter vários desenvolvedores (e vice-versa), resolvido com tabelas associativas.

4.2 Modelo Relacional



O diagrama apresenta as tabelas do banco de dados da empresa criadora de jogos em formato de modelo relacional, listando os atributos de cada entidade e seus relacionamentos, chaves estrangeiras e suas referências.

Observações sobre o diagrama do modelo relacional:

1. **Chaves estrangeiras (FKs):**
   1.1 Promocoes.id_jogo referencia Jogos.id_jogo;
   1.2 Avaliacoes.id_jogo e Avaliacoes.id_usuario referenciam Jogos.id_jogo e Usuarios.id_usuario, respectivamente,
2. **Atributos críticos:**
   1.1 Jogo_Desenvolvedor.funcao: Indica o papel do desenvolvedor no jogo (Ex.: “Programador”, “Designer”).
   1.2 Promocoes.desconto: Validado por CHECK para garantir valores entre 0 e 100%.

5 Desenvolvimento: Detalhes e Decisões

Abaixo são apresentados os principais elementos de banco de dados, com ênfase nas decisões de implementação relacionadas a triggers, stored procedures e funções.

5.1 Triggers: Validação e Automação

Os triggers foram implementados para garantir integridade dos dados e conformidade com regras de negócio.

5.1.1 Validação de Email em Usuários

```
CREATE TRIGGER antes_insert_usuario
BEFORE INSERT ON Usuarios
FOR EACH ROW
BEGIN
    IF NEW.email IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Email é obrigatório!';
    END IF;
END;
```

Impede a inserção de usuários sem email, evitando inconsistência em comunicações e login.

5.1.2 Data Automática em Avaliações

```
CREATE TRIGGER atualizar_data_avaliacao
BEFORE INSERT ON Avaliacoes
FOR EACH ROW
BEGIN
    IF NEW.data_avaliacao IS NULL THEN
        SET NEW.data_avaliacao = CURDATE(); -- Define a data atual se não for
informada
    END IF;
END;
```

Garante que toda avaliação tenha uma data registrada, facilitando filtros por período.

5.1.3 Impedir Exclusão de Administradores

```
CREATE TRIGGER impedir_exclusao_admin
BEFORE DELETE ON Usuarios
FOR EACH ROW
BEGIN
    IF OLD.tipo_usuario = 'admin' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Admins não podem ser
excluídos!';
    END IF;
END;
```

Protege contas críticas contra exclusões acidentais, mantendo a segurança do sistema.

5.2 Stored Procedures: Automação de Processos

As procedures foram criadas para simplificar operações

5.2.1 CriarPromocao: Gestão de Descontos

```
CREATE PROCEDURE CriarPromocao (
    IN jogo_id INT,
    IN desc_promo VARCHAR(255),
    IN desconto_promo DECIMAL(5,2),
    IN inicio_promo DATE,
    IN fim_promo DATE
)
BEGIN
    INSERT INTO Promocoes (id_jogo, descricao, desconto, data_inicio, data_fim)
    VALUES (jogo_id, desc_promo, desconto_promo, inicio_promo, fim_promo);
END;
```

Centraliza a criação de promoções, garantindo que todas as etapas (validação, inserção) sejam executadas em uma única chamada.

5.2.2 AtualizarPreco: Ajuste Seguro de Preços

```
CREATE PROCEDURE AtualizarPreco (
    IN id INT,
    IN novo_preco DECIMAL(10,2)
)
BEGIN
    -- Trigger 'atualizar_preco_negativo' impede valores negativos
    UPDATE Jogos SET preco = novo_preco WHERE id_jogo = id;
END;
```

Combina a atualização com um trigger para evitar preços inválidos, como valores negativos.

5.3 Funções: Cálculos Especializados

Funções foram utilizadas para encapsular lógica de negócio e simplificar consultas.

5.3.1 CalcularPrecoComDesconto: Precificação Dinâmica

```
CREATE FUNCTION CalcularPrecoComDesconto(jogo_id INT) RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE preco_final DECIMAL(10,2);
    SELECT preco * (1 - IFNULL(desconto, 0)/100) INTO preco_final
    FROM Jogos LEFT JOIN Promocoes ON Jogos.id_jogo = Promocoes.id_jogo
```

```
    WHERE Jogos.id_jogo = jogo_id AND CURDATE() BETWEEN data_inicio AND data_fim
    LIMIT 1;
    RETURN IFNULL(preco_final, (SELECT preco FROM Jogos WHERE id_jogo = jogo_id));
END;
```

Calcula o preço promocional em tempo real, considerando promoções ativas. Se não houver desconto, retorna o preço original.

5.3.2 VerificarEmailUnica: Controle de Unicidade

```
CREATE FUNCTION VerificarEmailUnico(email_check VARCHAR(100)) RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE total INT;
    SELECT COUNT(*) INTO total FROM Usuarios WHERE email = email_check;
    RETURN total = 0; -- Retorna TRUE se o email não existir
END;
```

Usada em aplicações frontEnd para validar emails antes de inserir novos usuários, evitando duplicatas.

5.4 Views: Simplificação de Consultas

As views foram projetadas para agilizar relatórios e proteger dados sensíveis.

5.4.1 View_PromocoesAtivas: Promoções em Andamento

```
CREATE VIEW View_PromocoesAtivas AS
SELECT j.nome AS jogo, p.desconto, p.data_inicio, p.data_fim
FROM Promocoes p
JOIN Jogos j ON p.id_jogo = j.id_jogo
WHERE CURDATE() BETWEEN p.data_inicio AND p.data_fim;
```

Filtra apenas promoções vigentes, útil para exibição em vitrines de e-commerce.

5.4.2 View_TopJogos: Ranking por Avaliação

```
CREATE VIEW View_TopJogos AS
SELECT j.nome, AVG(a.nota) AS media_nota
FROM Jogos j
LEFT JOIN Avaliacoes a ON j.id_jogo = a.id_jogo
GROUP BY j.id_jogo
ORDER BY media_nota DESC
LIMIT 10;
```

Ordena jogos por nota média, facilitando a identificação dos mais populares.

6 Conclusão

6.1 Dificuldades Encontradas

- Complexidade dos relacionamentos N:N: A modelagem de tabelas associativas exigiu atenção redobrada para garantir que as chaves estrangeiras e as cardinalidades estivessem corretas.
- Validações em Triggers: Implementar Triggers foi desafiador devido à necessidade de cobrir todos os cenários de erro sem impactar o desempenho.
- Gestão de promoções ativas: A função CalcularPrecoComDesconto precisou lidar casos onde múltiplas promoções poderiam ser aplicadas ao mesmo jogo.

6.2 Limitações do Projeto

- O banco de dados não foi testado com grandes volumes de dados, o que poderia exigir índices adicionais ou particionamentos de tabelas.
- Controle de acesso básico: Os perfil de usuário são estáticos. No caso, seria necessário um sistema de permissões mais granular.
- Falta de histórico: Alterações críticas não são registradas em uma tabela de log, dificultando auditorias.

6.3 Opinião Individual

6.3.1 Pedro Pinto

Tive algumas dificuldades na implementação e adaptação de alguns Triggers, porém, o trabalho me ajudou a reforçar esse conceito e além desse, também o conceito de Stored Procedure. Apesar das limitações, o sistema atende aos requisitos propostos, mostrando como um banco de dados pode ser a base para aplicações complexas. A experiência reforçou a importância de planejar cuidadosamente as relações entre entidades e antecipar casos de uso reais.

6.3.2 Emanoel Borges

No início, tive muita dificuldade com a **sintaxe do SQL**, especialmente em comandos mais complexos como CREATE TRIGGER e STORED PROCEDURE. Erros de vírgula, parênteses esquecidos ou palavras-chave mal escritas (como DECLARE vs DELIMITER) quebravam o código todo e eram difíceis de debugar.
Apesar dos desafios, o projeto me fez evoluir muito em SQL. Agora consigo escrever queries básicas e entendo a importância de praticar constantemente.