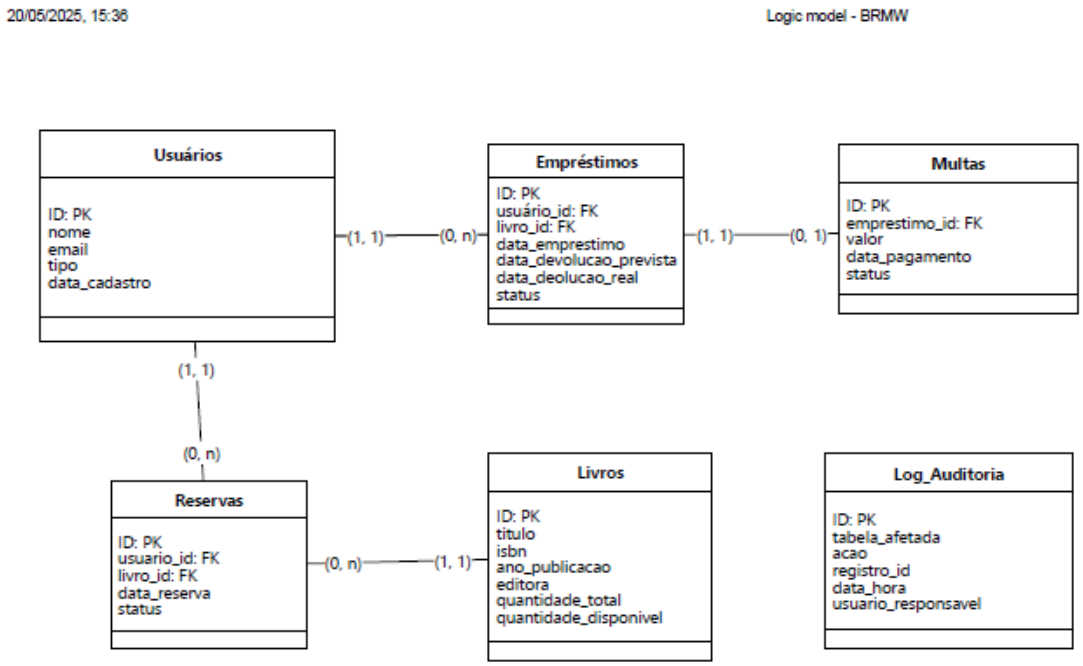
**PROJETO FINAL – BANCO DE DADOS AVANÇADO**

**Maria Eduarda Maia, Lucas Accioly e João Victor Soares.**

## Entregáveis

O nosso grupo ficou responsável por criar um Sistema de Biblioteca Universitária que possui Usuários, um sistema de Empréstimos, Reservas de livros e Multas.

## Modelagem




## Implementações e ScriptsSQL organizados

**---database criada manualmente---**
**---CRIAÇÃO DAS TABELAS---**
**---Criação da Tabela Usuarios---**

```
CREATE TABLE Usuarios (
  id SERIAL PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  tipo VARCHAR(20) CHECK (tipo IN ( 'aluno', 'professor', 'funcionario')) NOT NULL,
  data_cadastro DATE DEFAULT CURRENT_DATE
);
```

**---Criação da Tabela livros---**

```
CREATE TABLE Livros (
  id SERIAL PRIMARY KEY,
  titulo VARCHAR(200) NOT NULL,
  autor VARCHAR(100) NOT NULL,
  isbn VARCHAR(20) UNIQUE,
  ano_publicacao INT,
  editora VARCHAR(100),
  quantidade_total INT NOT NULL,
  quantidade_disponivel INT NOT NULL,
  CHECK (quantidade_disponivel <= quantidade_total)
);
```

**---Criação da Tabela Emprestimos---**

```
CREATE TABLE Emprestimos (
  id SERIAL PRIMARY KEY,
  usuario_id INT NOT NULL,
  livro_id INT NOT NULL,
  data_emprestimo DATE DEFAULT CURRENT_DATE,
  data_devolucao_prevista DATE NOT NULL,
  data_devolucao_real DATE,
  status VARCHAR(20) DEFAULT 'ativo' CHECK (status IN ('ativo', 'finalizado', 'atrasado')),
  FOREIGN KEY (usuario_id) REFERENCES Usuarios(id),
  FOREIGN KEY (livro_id) REFERENCES Livros(id)
);
```

**---Criação da tabela Reservas---**

```
CREATE TABLE Reservas (
   id SERIAL PRIMARY KEY,
   usuario_id INT NOT NULL,
   livro_id INT NOT NULL,
   data_reserva TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   status VARCHAR(20) DEFAULT 'ativo' CHECK (status IN ('ativo', 'cancelado', 'finalizado')),
   FOREIGN KEY (usuario_id) REFERENCES Usuarios(id),
   FOREIGN KEY (livro_id) REFERENCES Livros(id)
);
```

**---Criação da tabela Multas---**

```
CREATE TABLE Multas (
   id SERIAL PRIMARY KEY,
   emprestimo_id INT NOT NULL,
   valor NUMERIC(10, 2) NOT NULL,
   data_pagamento DATE,
   status VARCHAR(20) DEFAULT 'pendente' CHECK (status IN ('pendente', 'pago')),
   FOREIGN KEY (emprestimo_id) REFERENCES Emprestimos(id)
);
```

**---Criação da tabela Log_Auditoria---**

```
CREATE TABLE Log_Auditoria (
   id SERIAL PRIMARY KEY,
   tabela_afetada VARCHAR(50) NOT NULL,
   acao VARCHAR(10) CHECK (acao IN ('INSERT', 'UPDATE', 'DELETE')) NOT NULL,
   registro_id INT NOT NULL,
   data_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   usuario_responsavel VARCHAR(100)
);
```

**---INSERÇÃO DE DADOS---**
**---Inserir usuários---**

```
INSERT INTO Usuarios (nome, email, tipo) VALUES
('João Silva', 'joao@universidade.edu', 'aluno'),
('Maria Souza', 'maria@universidade.edu', 'professor'),
('Carlos Oliveira', 'carlos@universidade.edu', 'funcionario'),
('Ana Pereira', 'ana@universidade.edu', 'aluno');

SELECT * FROM Usuarios;
```

**---Inserir livros---**

*INSERT INTO Livros (titulo, autor, isbn, ano_publicacao, editora, quantidade_total, quantidade_disponivel) VALUES*
*('Banco de Dados Avançados', 'Carlos Heuser', '1234567890123', 2020, 'Bookman', 5, 5),*
*('SQL para Leigos', 'Alan Beaulieu', '9876543210987', 2018, 'Alta Books', 3, 3),*
*('Introdução à Programação', 'Luis Damas', '4567891230456', 2021, 'FCA', 8, 8),*
*('Redes de Computadores', 'Andrew Tanenbaum', '7891234560789', 2019, 'Pearson', 4, 4);*

*SELECT * FROM Livros;*

**---Inserir empréstimos---**

*INSERT INTO Emprestimos (usuario_id, livro_id, data_emprestimo, data_devolucao_prevista, status) VALUES*
*(1, 1, '2025-05-01', '2025-05-15', 'ativo'),*
*(2, 2, '2025-05-03', '2025-05-17', 'ativo'),*
*(3, 3, '2025-04-20', '2025-05-04', 'atrasado'),*
*(4, 4, '2025-05-10', '2025-05-24', 'ativo');*

*SELECT * FROM Emprestimos;*

**---Inserir multas---**

*INSERT INTO Multas (emprestimo_id, valor, status) VALUES*
*(3, 10.50, 'pendente');*

*SELECT * FROM Multas;*

**---Inserir reservas---**

INSERT INTO Reservas (usuario_id, livro_id, status) VALUES
(1, 3, 'ativo'),
(2, 1, 'cancelado');

SELECT * FROM Reservas;

***Obs.: SELECT * FROM***

***Testar se dados foram inseridos corretamente.***

**SUBCONSULTAS**

**---SUBCONSULTAS---**
**---1: Média de livros emprestados por usuário---**

**Obs.: Calcula a média de livros emprestados por usuários usando as funções AVG e COUNT**

```
SELECT AVG(total_emprestimos) AS media_emprestimos
FROM (
   SELECT usuario_id, COUNT(*) AS total_emprestimos
   FROM Emprestimos
   GROUP BY usuario_id
) AS subquery;
```

**---2: Contar empréstimos por livro---**

**Obs.: Conta empréstimos por livros usando a função COUNT**

```
SELECT
   titulo,
   (SELECT COUNT(*) FROM Emprestimos WHERE livro_id = l.id) AS total_emprestimos
FROM Livros l;
```

***---3: Livros nunca emprestados---***

**Obs.: Exibir livros que nunca foram emprestados**

```
SELECT titulo
FROM Livros l
WHERE NOT EXISTS (
   SELECT 1
   FROM Emprestimos e
   WHERE e.livro_id = l.id
);
```

**---4: Usuários com empréstimos ativos---**

**Obs.: Exibir usuários com empréstimos ativos**

```
SELECT nome
FROM Usuarios
WHERE id IN (
   SELECT usuario_id
   FROM Emprestimos
   WHERE status = 'ativo'
);
```

**---5: Livro mais emprestado---**

**Obs.: Calcular livros mais emprestados usando a função COUNT**

```
SELECT titulo, autor
FROM Livros
WHERE id = (
   SELECT livro_id
   FROM Emprestimos
   GROUP BY livro_id
   ORDER BY COUNT(*) DESC
   LIMIT 1
);
```

**Obs.: Foram usados o FROM, SELECT, EXISTS, NOT EXISTS**

**---JOINS AVANÇADOS---**

**---INNER JOIN---**

**Obs.: Exibe usuários e seus empréstimos mostrando os registros que estão nas duas tabelas.**

```
SELECT u.nome, l.titulo, e.data_emprestimo
FROM Usuarios u
INNER JOIN Emprestimos e ON u.id = e.usuario_id
INNER JOIN Livros l ON e.livro_id = l.id;
```

**---LEFT JOIN---**

**Obs.: Exibe todos os usuários, mesmo aqueles que nunca fizeram empréstimos (Quando não houver empréstimos aparece NULL na tabela direita).**

```
SELECT u.nome, e.id AS emprestimo_id
FROM Usuarios u
LEFT JOIN Emprestimos e ON u.id = e.usuario_id;
```

**---RIGHT JOIN---**

**Obs.: Exibe todos os usuários, mesmo aqueles que nunca fizeram empréstimos (Quando não houver empréstimos aparece NULL na tabela esquerda).**

```
SELECT e.id AS emprestimo_id, u.nome
FROM Emprestimos e
RIGHT JOIN Usuarios u ON e.usuario_id = u.id;
```

**---FULL OUTER JOIN---**

**Obs.: Exibe usuários com ou sem empréstimos e vice-versa caso o usuário tenha sido excluído.**

```
SELECT u.nome, e.id AS emprestimo_id
FROM Usuarios u
FULL OUTER JOIN Emprestimos e ON u.id = e.usuario_id;
```

**---JOIN COM SUBCONSULTA COMO TABELA DERIVADA---**

**Obs.: Exibe usuários com mais de um empréstimo usando uma subconsulta como uma tabela temporária.**

```
SELECT u.nome, sub.total
FROM Usuarios u
JOIN (
   SELECT usuario_id, COUNT(*) AS total
   FROM Emprestimos
   GROUP BY usuario_id
   HAVING COUNT(*) > 1
) AS sub ON u.id = sub.usuario_id;
```

**---STORED PROCEDURES---**

**---Procedure com cursar---**

**Obs.: Atualiza automaticamente os empréstimos atrasados para o status "ATRASADO".**

```
CREATE OR REPLACE PROCEDURE atualizar_status_emprestimos()
LANGUAGE plpgsql
AS $$
DECLARE
   emp RECORD;
BEGIN
   FOR emp IN SELECT * FROM Emprestimos WHERE data_devolucao_real IS NULL LOOP
      IF emp.data_devolucao_prevista < CURRENT_DATE THEN
         UPDATE Emprestimos
         SET status = 'atrasado'
         WHERE id = emp.id;
      END IF;
   END LOOP;
END;
$$;
```

```
CALL atualizar_status_emprestimos();
SELECT * FROM Emprestimos;
```

**---Procedure com Parâmetros Opcionais---**

**Obs.: Permite buscar usuários por tipo e/ou nome.**

```
CREATE OR REPLACE PROCEDURE buscar_usuarios(tipo_filtro VARCHAR DEFAULT NULL,
nome_filtro VARCHAR DEFAULT NULL)
LANGUAGE plpgsql
AS $$
BEGIN
   IF tipo_filtro IS NOT NULL AND nome_filtro IS NOT NULL THEN
      RAISE NOTICE 'Tipo e nome:';
      PERFORM * FROM Usuarios WHERE tipo = tipo_filtro AND nome ILIKE '%' || nome_filtro ||
'%';
   ELSIF tipo_filtro IS NOT NULL THEN
      RAISE NOTICE 'Só tipo:';
      PERFORM * FROM Usuarios WHERE tipo = tipo_filtro;
   ELSIF nome_filtro IS NOT NULL THEN
      RAISE NOTICE 'Só nome:';
      PERFORM * FROM Usuarios WHERE nome ILIKE '%' || nome_filtro || '%';
   ELSE
      RAISE NOTICE 'Todos os usuários:';
      PERFORM * FROM Usuarios;
   END IF;
END;
$$;
```

```
CALL buscar_usuarios('aluno', 'João');  ---usado para testar---

SELECT * FROM Usuarios ---teste com exibição---
WHERE tipo = 'aluno' AND nome ILIKE '%João%';
```

***---Procedure para Registrar Reserva---***

**Obs.: Insere nova reserva ativa ativa para um usuário e livro.**

```
CREATE OR REPLACE PROCEDURE registrar_reserva(p_usuario_id INT, p_livro_id INT)
LANGUAGE plpgsql
AS $$
BEGIN
   INSERT INTO Reservas (usuario_id, livro_id, data_reserva, status)
   VALUES (p_usuario_id, p_livro_id, CURRENT_TIMESTAMP, 'ativo');
END;
$$;

CALL registrar_reserva(1, 2);
SELECT * FROM Reservas ORDER BY id DESC;
```

**---FUNCTIONS---**
**---Tipo Escalar--**

**Ex : O total de multas de um usuário**

```
CREATE OR REPLACE FUNCTION calcular_total_multas(p_usuario_id INT)
RETURNS NUMERIC AS $$
DECLARE
   total NUMERIC := 0;
BEGIN
   SELECT SUM(m.valor)
   INTO total
   FROM Multas m
   JOIN Emprestimos e ON m.emprestimo_id = e.id
   WHERE e.usuario_id = p_usuario_id;

   RETURN COALESCE(total, 0);
END;
$$ LANGUAGE plpgsql;

SELECT calcular_total_multas(3);
```

**---Retorna a Tabela**

**Ex: lista de livros emprestados por usuário**

```
CREATE OR REPLACE FUNCTION listar_livros_usuario(p_usuario_id INT)
RETURNS TABLE(titulo VARCHAR, data_emprestimo DATE) AS $$
BEGIN
   RETURN QUERY
   SELECT l.titulo, e.data_emprestimo
```

```
   FROM Emprestimos e
   JOIN Livros l ON e.livro_id = l.id
   WHERE e.usuario_id = p_usuario_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM listar_livros_usuario(1);
```

**---Tratamento de Erro---**

**Ex.: Verifica se o livro está disponível**

```
CREATE OR REPLACE FUNCTION verificar_disponibilidade(p_livro_id INT)
RETURNS BOOLEAN AS $$
DECLARE
   disponivel INT;
BEGIN
   SELECT quantidade_disponivel INTO disponivel FROM Livros WHERE id = p_livro_id;

   IF disponivel IS NULL THEN
      RAISE EXCEPTION 'Livro com ID % não encontrado.', p_livro_id;
   ELSIF disponivel < 1 THEN
      RAISE EXCEPTION 'Livro com ID % está indisponível para empréstimo.', p_livro_id;
   END IF;

   RETURN TRUE;
END;
$$ LANGUAGE plpgsql;

SELECT verificar_disponibilidade(1);
```

**---TRIGGER---**

**---Auditoria---**

**---Função---**

```
CREATE OR REPLACE FUNCTION log_auditoria_func()
RETURNS TRIGGER AS $$
BEGIN
   INSERT INTO Log_Auditoria (tabela_afetada, acao, registro_id, data_hora,
usuario_responsavel)
   VALUES (
      TG_TABLE_NAME,
      TG_OP,
      COALESCE(NEW.id, OLD.id),
      CURRENT_TIMESTAMP,
      current_user
   );
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

----Exemplo de Aplicação na Tabela Usuarios---
CREATE TRIGGER trigger_log_usuarios
AFTER INSERT OR UPDATE OR DELETE ON Usuarios
FOR EACH ROW
EXECUTE FUNCTION log_auditoria_func();


---Envolvendo Duas Tabelas---
---FUNÇÃO---
---DIMINUIR---
CREATE OR REPLACE FUNCTION diminuir_disponibilidade()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Livros
    SET quantidade_disponivel = quantidade_disponivel - 1
    WHERE id = NEW.livro_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_diminuir_disponibilidade
AFTER INSERT ON Emprestimos
FOR EACH ROW
EXECUTE FUNCTION diminuir_disponibilidade();

---AUMENTAR---
CREATE OR REPLACE FUNCTION aumentar_disponibilidade()
RETURNS TRIGGER AS $$
BEGIN

    IF NEW.data_devolucao_real IS NOT NULL AND OLD.data_devolucao_real IS NULL THEN
        UPDATE Livros
        SET quantidade_disponivel = quantidade_disponivel + 1
        WHERE id = NEW.livro_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_aumentar_disponibilidade
```

```
AFTER UPDATE ON Emprestimos
FOR EACH ROW
EXECUTE FUNCTION aumentar_disponibilidade();
```

**---Alterar Comportamento---**

**Obs.: Impede a exclusão de um usuário com um empréstimo ativo.**

**---FUNÇÃO---**

```
CREATE OR REPLACE FUNCTION bloquear_exclusao_usuario()
RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (
      SELECT 1
      FROM Emprestimos
      WHERE usuario_id = OLD.id AND status = 'ativo'
   ) THEN
      RAISE EXCEPTION 'Usuário possui empréstimos ativos e não pode ser excluído.';
   END IF;

   RETURN OLD;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_bloquear_exclusao
BEFORE DELETE ON Usuarios
FOR EACH ROW
EXECUTE FUNCTION bloquear_exclusao_usuario();
```

**---TESTES DOS TRIGGER---**

**---AUDITORIA---**

**---Inserir novo usuário---**

```
INSERT INTO Usuarios (nome, email, tipo)
VALUES ('Teste Auditoria', 'auditoria@teste.com', 'aluno');
```

**---Atualizar usuário---**

```
UPDATE Usuarios
SET nome = 'Teste Alterado'
WHERE email = 'auditoria@teste.com';
```

**---Deletar usuário---**

```
DELETE FROM Usuarios
WHERE email = 'auditoria@teste.com';
```

**---Verificação da Tabela Log---**

```
SELECT * FROM Log_Auditoria ORDER BY id DESC;
```

**---DIMINUIR---**
*SELECT id, titulo, quantidade_disponivel FROM Livros WHERE id = 1;*
**---Fazendo um empréstimo---**
*INSERT INTO Emprestimos (usuario_id, livro_id, data_emprestimo, data_devolucao_prevista, status)*
*VALUES (1, 1, CURRENT_DATE, CURRENT_DATE + INTERVAL '7 days', 'ativo');*
*---teste para ver se realmente diminuiu---*
*SELECT id, titulo, quantidade_disponivel FROM Livros WHERE id = 1;*

**---AUMENTAR---**
*SELECT quantidade_disponivel FROM Livros WHERE id = 1;*
**---atualizando---**
*UPDATE Emprestimos*
*SET data_devolucao_real = CURRENT_DATE*
*WHERE id = 1;*
**---teste para ver se realmente aumentou---**
*SELECT quantidade_disponivel FROM Livros WHERE id = 1;*

**---BLOQUEAR EXCLUSÃO---**
*SELECT * FROM Emprestimos WHERE usuario_id = 1 AND status = 'ativo';*
**---tentativa de delete(tem que dar erro)---**
*DELETE FROM Usuarios WHERE id = 1;*

**---INDEXAÇÃO E OTIMIZAÇÃO---**

**---1: Criação dos Índices---**
**---1 SIMPLES(Usuarios.email)---**
*CREATE INDEX idx_email_usuarios ON Usuarios(email);*
**---2 SIMPLES(Livros.isbn)---**
*CREATE INDEX idx_isbn_livros ON Livros(isbn);*
**---3 COMPOSTO(Emprestimos (usuario_id, livro_id))**
*CREATE INDEX idx_usuario_livro_emprestimo ON Emprestimos(usuario_id, livro_id);*

**---2: Comparar Performance com EXPLAIN ANALYZE---**
**---SEM ÍNDICE---**
**EXPLAIN ANALYZE**
*SELECT * FROM Usuarios WHERE email = 'joao@universidade.edu';*
**---COM ÍNDICE---**
**EXPLAIN ANALYZE**
*SELECT * FROM Usuarios WHERE email = 'joao@universidade.edu';*

**---MELHORIAS---**

**---Buscar empréstimos por status---**

```
CREATE INDEX idx_emprestimos_status ON Emprestimos(status);
EXPLAIN ANALYZE
SELECT * FROM Emprestimos WHERE status = 'ativo';
```

**---Buscar por data de devolução---**

```
CREATE INDEX idx_emprestimos_devolucao_prevista ON
Emprestimos(data_devolucao_prevista);
EXPLAIN ANALYZE
SELECT * FROM Emprestimos
WHERE data_devolucao_prevista < CURRENT_DATE
  AND data_devolucao_real IS NULL;
```

**---Reservas por status (usado em várias consultas)---**

```
CREATE INDEX idx_reservas_status ON Reservas(status);
EXPLAIN ANALYZE
SELECT * FROM Reservas WHERE status = 'ativo';
```