UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS — PAU DOS FERROS
DOCENTE: BRUNO SILVA BORGES

Discentes: CLAUDIO FELIPE LOPES DA SILVA,
DAVI GABRIEL DE OLIVEIRA PASCOAL
IGOR CAVALCANTE ROCHA
IARA RAQUEL DE ALMEIDA FERNANDES,
JOÃO CARLOS DE SOUSA GURGEL ROCHA,

SISTEMA DE GERENCIAMENTO DE EVENTOS

Introdução

O sistema tem como objetivo gerenciar o ciclo de vida de eventos acadêmicos e corporativos, permitindo o controle de locais, datas, vagas e a gestão das inscrições dos participantes.

Funcionalidades Principais:

- Cadastro e manutenção de eventos e categorias.
- Gerenciamento de locais e capacidade de público.
- Registro de participantes.
- Realização e cancelamento de inscrições em eventos.

Requisitos detalhados

➔ Requisitos funcionais

- O sistema deve permitir que um organizador cadastre um evento com data, hora e local.
- O sistema deve permitir o cadastro de participantes com nome, e-mail e CPF.
- O sistema deve controlar a capacidade máxima do local, impedindo inscrições quando as vagas esgotarem.
- O sistema deve permitir gerar um relatório de inscritos por evento.

➔ Caso de uso

Ator: Participante.

Pré-condição: O evento deve existir no sistema, estar com status 'Agendado' e possuir vagas disponíveis (capacidade do local não atingida).

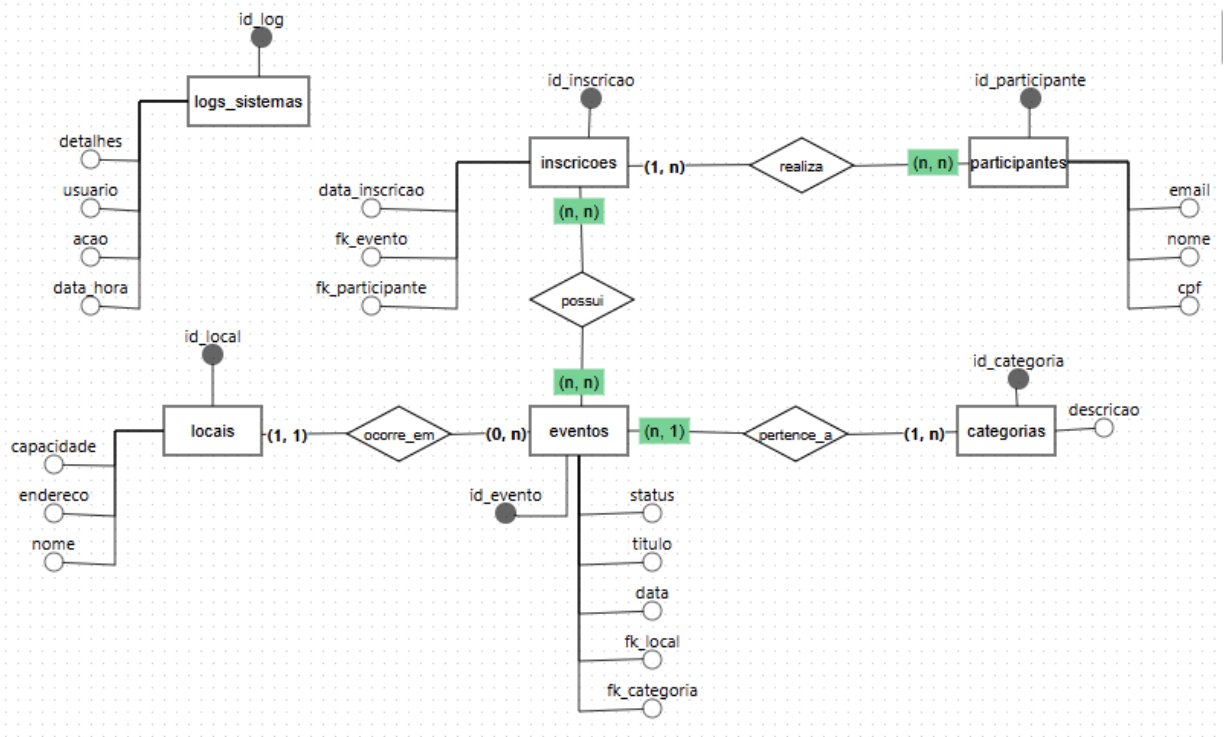
Fluxo Principal:

1. O Participante solicita a visualização dos eventos disponíveis.
2. O Sistema apresenta a lista de eventos (título, data, local e vagas).
3. O Participante seleciona o evento desejado e informa seus dados (identificação pelo CPF).

4. O Sistema verifica se o id já está inscrito naquele evento.
5. O Sistema verifica se há vagas disponíveis no local (disparando o Gatilho de verificação).
6. O Sistema confirma a inscrição.

Pós-condição: Um novo registro é criado na tabela de inscrições e a quantidade de vagas disponíveis para aquele evento é decretada virtualmente.

2.1 Modelo conceitual



2.2 Modelo Lógico

Tabela Categorias: Responsável por armazenar os tipos de eventos disponíveis. Possui o identificador único (`id_categoria`) e a descrição do tipo (ex: Workshop, Palestra).

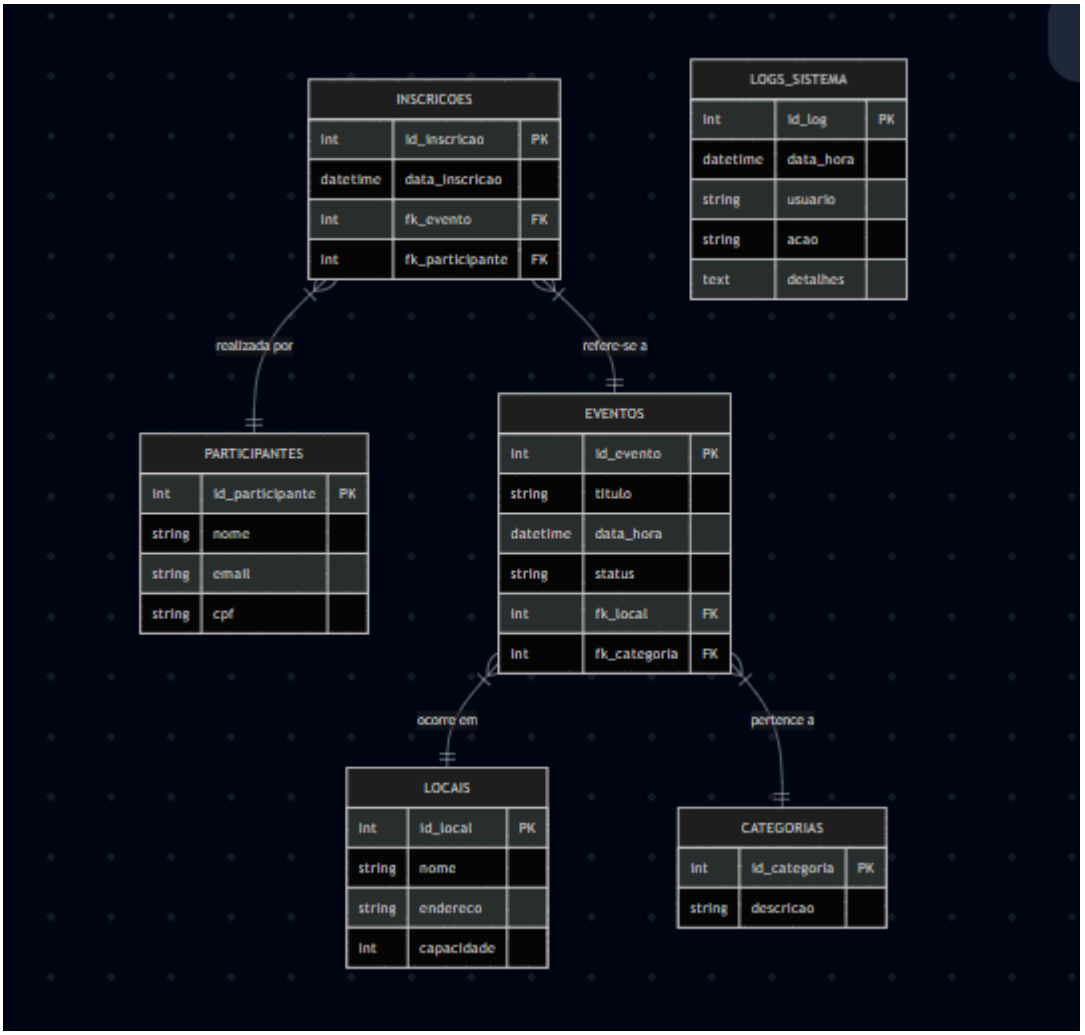
Tabela Locais: Armazena as informações sobre os espaços físicos. Define o nome do local, endereço e a capacidade máxima de lotação permitida.

Tabela Participantes: Contém os dados cadastrais das pessoas. Garante a unicidade de e-mails e CPFs para evitar duplicidades.

Tabela Eventos: É a tabela central que registra as atividades. Relaciona-se com `Locais` (onde ocorre) e `Categorias` (tipo do evento), além de controlar o status da atividade.

Tabela Inscricoes: Tabela associativa que efetiva a relação N:N entre Participantes e Eventos. Registra quando um participante se inscreve em um evento específico.

Tabela Logs_Sistema: Tabela destinada à auditoria, registrando operações críticas (como cancelamentos), juntamente com o usuário responsável e a data/hora da ação.



2.3 Modelo Físico

```
-- Criação da Tabela de Categorias
CREATE TABLE categorias (
    id_categoria SERIAL PRIMARY KEY,
    descricao VARCHAR(100) NOT NULL
);
```

```sql
-- Criação da Tabela de Locais
CREATE TABLE locais (
    id_local SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    endereco VARCHAR(200) NOT NULL,
    capacidade INT NOT NULL
);

-- Criação da Tabela de Participantes
CREATE TABLE participantes (
    id_participante SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    cpf VARCHAR(14) NOT NULL UNIQUE
);

-- Criação da Tabela de Eventos
CREATE TABLE eventos (
    id_evento SERIAL PRIMARY KEY,
    titulo VARCHAR(150) NOT NULL,
    data_hora TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'Agendado',
    fk_local INT NOT NULL,
    fk_categoria INT NOT NULL,
    FOREIGN KEY (fk_local) REFERENCES locais(id_local),
    FOREIGN KEY (fk_categoria) REFERENCES categorias(id_categoria)
);

-- Criação da Tabela de Inscrições
CREATE TABLE inscricoes (
    id_inscricao SERIAL PRIMARY KEY,
    data_inscricao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fk_evento INT NOT NULL,
    fk_participante INT NOT NULL,
    FOREIGN KEY (fk_evento) REFERENCES eventos(id_evento),
    FOREIGN KEY (fk_participante) REFERENCES participantes(id_participante),
    UNIQUE(fk_evento, fk_participante)
);

-- Criação da Tabela de Logs do Sistema
CREATE TABLE logs_sistema (
    id_log SERIAL PRIMARY KEY,
    data_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario VARCHAR(50),
    acao VARCHAR(100),
    detalhes TEXT
);
```

2.4 Normalização

CATEGORIAS (id_categoria, descricao)

LOCAIS (id_local, nome, endereco, capacidade)

PARTICIPANTES (id_participante, nome, email, cpf)

EVENTOS (id_evento, titulo, data_hora, status, fk_local, fk_categoria)

INSCRICOES (id_inscricao, data_inscricao, fk_evento, fk_participante)

LOGS_SISTEMA (id_log, data_hora, usuario, acao, detalhes).

Aplicação da Primeira Forma Normal (1FN)

- Todos os atributos devem ser **atômicos** (não divisíveis)
- Não pode haver **repetições** ou **grupos multivalorados**
- Todas as linhas devem ser **únicas**

Verificação:

Tabela	**Está em 1FN?**
categorias	Sim
locais	Sim
participantes	Sim
eventos	Sim
inscricoes	Sim
logs_sistema	Sim

Justificativa

descricao é atômica
todos atributos são indivisíveis
nome/email/cpf não têm repetição interna
data_hora é um único valor; status também
não há repetição de valores e cada inscrição é única
todos valores são atômicos

Aplicação da Terceira Forma Normal (3FN)

Requisitos da 3FN:

- Estar em 2FN
- Não pode haver **dependência transitiva** (atributo não-chave → atributo não-chave)

CATEGORIAS (id_categoria, descricao)

- descricao depende apenas da chave
- Sem atributos adicionais
  Está em 3FN

LOCAIS (id_local, nome, endereco, capacidade)

- Nenhum atributo depende de outro
  Está em 3FN

PARTICIPANTES (id_participante, nome, email, cpf)

- email e cpf são únicos, mas não causam dependência transitiva
  Está em 3FN

EVENTOS (id_evento, titulo, data_hora, status, fk_local, fk_categoria)

- titulo, data_hora e status dependem somente de id_evento
- fk_local depende da chave estrangeira, não de outro atributo
  Está em 3FN

INSCRICOES (id_inscricao, data_inscricao, fk_evento, fk_participante)

- Nenhum atributo depende de outro
  Está em 3FN

LOGS_SISTEMA (id_log, data_hora, usuario, acao, detalhes)

- detalhes não depende de usuario
- nenhum atributo depende de outro
  Está em 3FN

MODELO FINAL NORMALIZADO (3FN)

- - Entidades Principais

CATEGORIA (id_categoria, descricao)

LOCAL (id_local, nome, endereco, capacidade)

PARTICIPANTE (id_participante, nome, email, cpf)

EVENTO (id_evento, titulo, data_hora, status, fk_local, fk_categoria)

- - Relacionamento N:N (resolvido por tabela associativa)

INSCRICAO (id_inscricao, data_inscricao, fk_evento, fk_participante)

- - Entidade Auxiliar (Logs)

LOG_SISTEMA (id_log, data_hora, usuario, acao, detalhes)

3.Dicionário de Dados

Tabela Participantes

Tabela Participante	Atributos	Tipo de Dado	Tamanho	Restrições	Chave	Descrição
Participante	id_participante	INT	N/A	SERIAL PRIMARY KEY	**PK**	identificador do participante
Participante	cpf	VARCHAR	14	NOT NULL UNIQUE	-	cpf do participante
Participante	nome	VARCHAR	100	NOT NULL	-	Nome do participante
Participante	email	VARCHAR	100	NOT NULL UNIQUE	-	Endereço email do partcipante

Tabela Inscrições

Tabela de Inscrições	Atributos	Tipo de Dados	Tamanho	Restrição	Chave	Descrição
Inscrições	id_inscricao	INT	N/A	SERIAL PRIMARY KEY	**PK**	Identificador único da inscrição
Inscrições	data_inscricao	TIMESTAMP	N/A	DEFAULT	-	Data da realização da inscrição
Inscrições	fk_evento	INT	N/A	NOT NULL	**FK**	Identificador do evento
Inscrições	fk_participante	INT	N/A	NOT NULL	**FK**	Identificador do participante

Tabela Eventos

Tabela Eventos	Atributos	Tipo de Dado	Tamanho	Restrições	Chave	Descrição
Eventos	id_evento	INT	N/A	SERIAL PRIMARY KEY	**PK**	Identificador único para o evento
Eventos	titulo	VARCHAR	150	NOT NULL	-	Titulo do evento
Eventos	data_hora	TIMESTAMP	N/A	NOT NULL	-	data e hora da realização do evento
Eventos	status	VARCHAR	20	DEFAULT	-	status do evento 'Agendado'
Eventos	fk_local	INT	N/A	NOT NULL	**FK**	Identificador do local do evento
Eventos	fk_categoria	INT	N/A	NOT NULL	**FK**	Identificador da categoria do evento

Tabela Categorias

Tabela categorias	Atributos	Tipo de Dado	Tamanho	Restrições	Chave	Descrição
categorias	id_categoria	INT	N/A	SERIAL PRIMARY KEY	**PK**	identificador único da categoria
categorias	descricao	VARCHAR	100	NOT NULL	-	descrição da categoria

Tabela Locais

Tabela Locais	Atributos	Tipo de Dado	Tamanho	Restrições	Chave	Descrição
Locais	id_local	INT	N/A	SERIAL PRIMARY KEY	**PK**	identificador único para o local
Locais	nome	VARCHAR	100	NOT NULL	-	nome do local
Locais	endereco	VARCHAR	200	NOT NULL	-	endereço do local
Locais	capacidade	INT	N/A	NOT NULL	-	capacidade do local

Tabela Logs_sistema

Tabela logs_sistema	Atributos	Tipos de Dados	Tamanho	Restrição	Chave	Descrição
logs_sistema	id_log	INT	N/A	SERIAL PRIMARY KEY	**PK**	Identificador único do log
logs_sistema	data_hora	TIMESTAMP	N/A	DEFAULT	-	Data e hora do registro do log
logs_sistema	usuario	VARCHAR	50	-	-	Usuário que realizou a ação
logs_sistema	acao	VARCHAR	100	-	-	Ação executada pelo usuário
logs_sistema	detalhes	TEXT	N/A	-	-	Informações adicionais sobre o log

4. Implementação e Recursos Avançados

Esta seção apresenta a implementação prática do banco de dados, demonstrando desde as operações fundamentais de manipulação de dados até o uso de objetos avançados como Views, Triggers e Stored Functions para garantir a integridade, segurança e usabilidade do sistema.

4.1 Operações Básicas (SQL)

Abaixo estão exemplos dos comandos DML (Data Manipulation Language) utilizados para gerenciar os dados do sistema.

Inserção de Dados (INSERT)

Exemplo de inserção de novos registros nas tabelas de Categorias e Eventos.

INSERT INTO categorias (descricao) VALUES ('Workshop'), ('Palestra'), ('Minicurso');

INSERT INTO eventos (titulo, data_hora, status, fk_local, fk_categoria) VALUES ('Introdução ao SQL', '2025-11-20 14:00:00', 'Agendado', 2, 1);

Remoção de Dados (DELETE)

Exemplo de remoção de uma inscrição específica baseada no ID.

DELETE FROM inscricoes WHERE id_inscricao = 3;

Listagem de Dados (SELECT)

Exemplo de consulta para listar todos os participantes cadastrados, ordenados pelo nome.

SELECT * FROM participantes ORDER BY nome ASC;

Alteração de Dados (UPDATE)

Exemplo de atualização do status de um evento, alterando-o de 'Agendado' para 'Cancelado'.

UPDATE eventos SET status = 'Cancelado' WHERE id_evento = 3;

4.2 Views (Visualizações)

As Views foram criadas para simplificar consultas complexas e apresentar os dados de forma mais intuitiva para o usuário final.

View 1: Detalhes do Evento (v_eventos_detalhados)

Esta view tem a finalidade de exibir os dados completos dos eventos, substituindo os códigos numéricos (IDs) de local e categoria pelos seus respectivos nomes e descrições.

CREATE OR REPLACE VIEW v_eventos_detalhados AS

SELECT e.id_evento, e.titulo, e.data_hora, e.status, l.nome AS local, c.descricao AS categoria

FROM eventos e

JOIN locais l ON e.fk_local = l.id_local

JOIN categorias c ON e.fk_categoria = c.id_categoria;

View 2: Relatório de Inscritos (v_relatorio_inscritos)

Esta view gera um relatório quantitativo, mostrando o número total de participantes inscritos em cada evento, facilitando o gerenciamento de lotação.

CREATE OR REPLACE VIEW v_relatorio_inscritos AS

SELECT e.titulo AS evento, COUNT(i.id_inscricao) AS total_inscritos

FROM eventos e

LEFT JOIN inscricoes i ON e.id_evento = i.fk_evento

GROUP BY e.titulo;

View 3: Lista de Presença (v_lista_presenca)

Esta view cruza dados de participantes, inscrições e eventos para gerar uma lista nominal de presença para cada atividade.

CREATE OR REPLACE VIEW v_lista_presenca AS

SELECT i.id_inscricao, p.nome AS participante, p.email, e.titulo AS evento, e.data_hora

FROM inscricoes i

JOIN participantes p ON i.fk_participante = p.id_participante

JOIN eventos e ON i.fk_evento = e.id_evento;

4.3 Gatilhos (Triggers)

Os gatilhos foram implementados para automatizar regras de negócio e garantir a integridade e auditoria dos dados.

Gatilho 1: Auditoria de Cancelamento (trg_auditoria_cancelamento)

Este gatilho é disparado automaticamente após a exclusão de uma inscrição (AFTER DELETE). Sua função é registrar essa ação na tabela de logs_sistema, garantindo rastreabilidade sobre quem removeu a inscrição e quando.

CREATE OR REPLACE FUNCTION fn_log_cancelamento() RETURNS TRIGGER AS $$

BEGIN

INSERT INTO logs_sistema (usuario, acao, detalhes)

VALUES (current_user, 'CANCELAMENTO', 'Inscrição ID ' || OLD.id_inscricao || ' removida.');

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trg_auditoria_cancelamento

AFTER DELETE ON inscricoes

FOR EACH ROW EXECUTE FUNCTION fn_log_cancelamento();

Gatilho 2: Bloqueio de Inscrição em Eventos Fechados (trg_verifica_status_evento)

Este gatilho é disparado antes de uma nova inscrição ser inserida (BEFORE INSERT). Ele verifica o status do evento; se não for 'Agendado' (por exemplo, se for 'Cancelado' ou 'Concluído'), a operação é bloqueada e um erro é retornado ao usuário.

```
CREATE OR REPLACE FUNCTION fn_bloqueia_evento_fechado() RETURNS TRIGGER
AS $$

DECLARE

v_status VARCHAR(20);

BEGIN

SELECT status INTO v_status FROM eventos WHERE id_evento = NEW.fk_evento;

IF v_status <> 'Agendado' THEN

RAISE EXCEPTION 'Erro: Evento com status % não aceita inscrições.', v_status;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trg_verifica_status_evento

BEFORE INSERT ON inscricoes

FOR EACH ROW EXECUTE FUNCTION fn_bloqueia_evento_fechado();
```

4.4 Funções (Stored Functions)

As funções foram criadas para encapsular lógicas de cálculo e formatação de dados reutilizáveis.

Função 1: Calcular Vagas Restantes (fn_vagas_restantes)

Esta função recebe o ID de um evento e retorna o número de vagas disponíveis, calculando a diferença entre a capacidade do local e o total de inscritos atuais.

```
CREATE OR REPLACE FUNCTION fn_vagas_restantes(p_id_evento INT)

RETURNS INT AS $$

DECLARE

v_capacidade INT;
```

```
v_inscritos INT;

BEGIN

SELECT l.capacidade INTO v_capacidade

FROM eventos e JOIN locais l ON e.fk_local = l.id_local

WHERE e.id_evento = p_id_evento;

SELECT COUNT(*) INTO v_inscritos FROM inscricoes WHERE fk_evento = p_id_evento;

RETURN GREATEST(v_capacidade - v_inscritos, 0);

END;

$$ LANGUAGE plpgsql;
```

Função 2: Resumo do Participante (fn_resumo_participante)

Esta função recebe o ID de um participante e retorna um texto formatado contendo seu nome e o total de eventos nos quais ele está inscrito, útil para exibições rápidas em relatórios ou interfaces.

```
CREATE OR REPLACE FUNCTION fn_resumo_participante(p_id_participante INT)

RETURNS TEXT AS $$

DECLARE

v_nome VARCHAR(100);

v_qtd_eventos INT;

BEGIN

SELECT nome INTO v_nome FROM participantes WHERE id_participante =
p_id_participante;

SELECT COUNT(*) INTO v_qtd_eventos FROM inscricoes WHERE fk_participante =
p_id_participante;

RETURN 'Participante: ' || v_nome || ' | Total de Eventos: ' || v_qtd_eventos;

END;

$$ LANGUAGE plpgsql;
```

5.Plano de testes

5. Plano de Testes

Esta seção detalha os cenários de teste planejados para validar as funcionalidades do banco de dados, assegurando a integridade dos dados, a correção das consultas e o funcionamento das automações implementadas.

5.1 Testes de Inserção

Estes testes validam a capacidade do sistema de registrar novos dados e o funcionamento das restrições de integridade, como a unicidade de campos.

Cenário 1: Inserção de um Participante Válido

- **Objetivo:** Verificar se o sistema aceita o cadastro de um participante com todos os dados corretos.
- Comando SQL:
  INSERT INTO participantes (nome, email, cpf) VALUES ('Teste da Silva', 'teste.silva@email.com', '999.999.999-99');
- **Resultado Esperado:** O comando deve retornar sucesso ("INSERT 0 1") e o participante deve ser persistido na tabela participantes.

Cenário 2: Tentativa de Inserção com CPF Duplicado

- **Objetivo:** Validar a restrição UNIQUE na coluna CPF da tabela de participantes.
- Comando SQL:
  INSERT INTO participantes (nome, email, cpf) VALUES ('Clone da Silva', 'clone@email.com', '999.999.999-99');
- **Resultado Esperado:** O banco de dados deve rejeitar a inserção com uma mensagem de erro indicando violação da restrição de unicidade (duplicate key value violates unique constraint).

5.2 Testes de Remoção

Estes testes verificam a exclusão de dados e o comportamento das chaves estrangeiras.

Cenário 1: Remoção de Inscrição Existente

- **Objetivo:** Verificar a remoção de um registro na tabela de relacionamento.
- Comando SQL:
  DELETE FROM inscricoes WHERE fk_evento = 2 AND fk_participante = 1;
- **Resultado Esperado:** O comando deve retornar "DELETE 0 1" (caso o registro exista) e a inscrição deve desaparecer da tabela inscricoes.

Cenário 2: Tentativa de Remoção de Registro Inexistente

- **Objetivo:** Verificar a robustez do comando DELETE.

- Comando SQL:
  DELETE FROM eventos WHERE id_evento = 9999;
- **Resultado Esperado:** O comando deve retornar "DELETE 0 0", sem causar erros no sistema.

5.3 Testes de Listagem (SELECT)

Estes testes validam a recuperação de informações, uso de junções e filtros específicos.

Cenário 1: Listagem de Eventos e Locais

- **Objetivo:** Verificar a junção (JOIN) entre as tabelas eventos e locais para exibir nomes em vez de códigos.
- Comando SQL:
  SELECT e.titulo, l.nome AS local, l.capacidade FROM eventos e INNER JOIN locais l ON e.fk_local = l.id_local;
- **Resultado Esperado:** Uma lista contendo o título dos eventos e os nomes dos locais correspondentes.

Cenário 2: Filtragem por E-mail

- **Objetivo:** Testar a cláusula WHERE com o operador LIKE para filtrar participantes.
- Comando SQL:
  SELECT * FROM participantes WHERE email LIKE '%@email.com';
- **Resultado Esperado:** Retorno de todos os registros de participantes que possuem o domínio de e-mail especificado.

5.4 Testes de Alteração (UPDATE)

Estes testes validam a atualização de dados cadastrais.

Cenário 1: Alteração de Status do Evento

- **Objetivo:** Modificar o status de um evento específico.
- Comando SQL:
  UPDATE eventos SET status = 'Concluído' WHERE id_evento = 1;
- **Resultado Esperado:** O comando deve retornar "UPDATE 0 1". Uma consulta posterior ao evento 1 deve mostrar o status atualizado.

5.5 Testes de Views

Estes testes garantem que as visualizações criadas simplificam o acesso aos dados conforme planejado na Seção 4.2.

Cenário 1: Consulta à View Detalhada (v_eventos_detalhados)

- **Objetivo:** Validar a exibição amigável dos dados do evento.
- Comando SQL:
  SELECT * FROM v_eventos_detalhados;

- **Resultado Esperado:** Exibição de uma tabela virtual contendo Título, Data, Status, Nome do Local e Descrição da Categoria, sem exibir as chaves estrangeiras numéricas.

Cenário 2: Consulta à View de Relatório (v_relatorio_inscritos)

- **Objetivo:** Validar o agrupamento e contagem de inscritos.
- Comando SQL:
  SELECT * FROM v_relatorio_inscritos;
- **Resultado Esperado:** Exibição de cada evento acompanhado da contagem total de inscrições realizadas.

5.6 Testes de Gatilhos (Triggers)

Estes testes validam as regras de negócio automáticas e auditoria.

Cenário 1: Auditoria de Cancelamento (trg_auditoria_cancelamento)

- **Ação:** Executar a remoção de uma inscrição.
- Comando SQL:
  DELETE FROM inscricoes WHERE id_inscricao = 4;
  SELECT * FROM logs_sistema ORDER BY id_log DESC LIMIT 1;
- **Resultado Esperado:** Após a exclusão, a tabela logs_sistema deve conter automaticamente um novo registro com a ação 'CANCELAMENTO' e os detalhes da inscrição excluída.

Cenário 2: Bloqueio de Inscrição em Evento não Agendado (trg_verifica_status_evento)

- **Ação:** Tentar inscrever um participante em um evento com status 'Cancelado' (ID 3).
- Comando SQL:
  INSERT INTO inscricoes (fk_evento, fk_participante) VALUES (3, 2);
- **Resultado Esperado:** O sistema deve impedir a inserção e retornar uma exceção com a mensagem "Erro: Evento com status Cancelado não aceita inscrições".

5.7 Testes de Funções

Estes testes validam as Stored Functions implementadas para lógica de negócio no banco.

Cenário 1: Cálculo de Vagas Restantes (fn_vagas_restantes)

- **Objetivo:** Verificar se o cálculo dinâmico de vagas está correto.
- Comando SQL:
  SELECT fn_vagas_restantes(1) AS vagas_disponiveis;
- **Resultado Esperado:** Retorno de um valor inteiro representando a capacidade do local subtraída do número de inscritos atuais no evento 1.

Cenário 2: Resumo do Participante (fn_resumo_participante)

- **Objetivo:** Validar a formatação de texto de saída da função.

- Comando SQL:
  SELECT fn_resumo_participante(1) AS resumo;
- **Resultado Esperado:** Retorno de uma string contendo o nome do participante e a quantidade total de eventos nos quais ele está inscrito.

6. Conclusão

O presente trabalho consistiu no projeto e implementação de um banco de dados relacional para um Sistema de Gestão de Eventos, com o objetivo de controlar o ciclo de vida de eventos acadêmicos e corporativos, desde o cadastro de locais e categorias até a gestão de inscrições e participantes.

Durante o desenvolvimento, foi possível aplicar na prática os conceitos fundamentais de modelagem de dados, passando pelas etapas de modelagem conceitual e lógica, até a implementação física utilizando a linguagem SQL (Structured Query Language). Um dos pontos cruciais do aprendizado foi o processo de normalização (1FN, 2FN e 3FN), que garantiu a eliminação de redundâncias e a integridade dos dados, resultando em um esquema otimizado com tabelas bem definidas para Eventos, Participantes, Locais, Categorias e Inscrições.

A implementação de recursos avançados proporcionou um entendimento mais profundo sobre a automação e segurança no banco de dados. A criação de Views (Visualizações) permitiu simplificar consultas complexas, facilitando a geração de relatórios de presença e detalhes dos eventos. O desenvolvimento de Stored Functions agregou inteligência ao banco, permitindo cálculos dinâmicos de vagas restantes. Além disso, a configuração de Triggers (Gatilhos) foi essencial para garantir regras de negócio críticas, como o bloqueio de inscrições em eventos cancelados e a auditoria automática de exclusões, assegurando a consistência das informações.

Entre os principais desafios enfrentados, destacam-se a lógica para manipular datas e calcular capacidades em tempo real, bem como o tratamento de restrições de chaves estrangeiras durante os testes de remoção. A superação desses obstáculos resultou em um sistema robusto, funcional e capaz de atender aos requisitos propostos, consolidando o conhecimento adquirido na disciplina sobre a arquitetura e gerenciamento de bancos de dados relacionais.