Curso: Tecnologia em Sistemas para Internet (EaD)
Disciplina: Banco de Dados II
Professor: Angelo Augusto Frozza

TRABALHO FINAL BANCO DE DADOS II

Jhonn L. S. Gonçalves[1]

a) Estudo de Caso: Sistema de Gerenciamento de Academia

Objetivo:

O projeto visa desenvolver um sistema de banco de dados para gerenciar, de forma eficiente, as operações de uma academia multifuncional que oferece serviços como musculação, zumba, crossfit e artes marciais. O sistema centralizará informações de alunos, funcionários, aulas, equipamentos e planos, otimizando recursos, automatizando processos e oferecendo relatórios estratégicos para suporte à decisão.

Descrição do Sistema:

O sistema organiza os principais elementos da academia em entidades inter-relacionadas, conforme descrito abaixo:

ALUNO: Representa os membros da academia, com atributos como nome, CPF, telefone, endereço e data de inscrição. Relaciona-se com aulas (frequência) e planos contratados.
EMPREGADO: Representa os funcionários, incluindo instrutores, administradores e zeladores, com dados como nome, CPF, cargo, salário e data de contratação.
- INSTRUTOR: Ministram aulas e possuem qualificações vinculadas à entidade CAPACITAÇÃO.
- ADMINISTRADOR: Gerencia contratos, funcionários e fornecedores.
- ZELADOR: Responsável pela manutenção dos equipamentos e infraestrutura.
PLANO: Representa os contratos oferecidos (mensal, trimestral, etc.) com informações sobre preço, duração e status (ativo/inativo).
AULA: Contém dados sobre atividades como nome, capacidade, duração e horários, permitindo o controle de lotação e frequência.
EQUIPAMENTO: Representa os aparelhos, organizados por departamentos, com controle de uso e manutenção preventiva.
CAPACITAÇÃO: Registra os cursos e certificações dos instrutores.
PAGAMENTO: Controla as transações financeiras relacionadas aos planos dos alunos.

[1] Aluno da Turma 2024, nº 2024001375, jlsgo.dev@gmail.com

FORNECEDOR: Registra empresas que fornecem equipamentos e serviços.

Principais Relacionamentos:
- Frequentam (ALUNO - AULA): Monitora a participação dos alunos em aulas e a ocupação das mesmas.
- Ministram (INSTRUTOR - AULA): Relaciona as aulas aos instrutores qualificados.
- Possuem (DEPARTAMENTO - EQUIPAMENTO): Relaciona os equipamentos ao setor responsável.
- Matricula (ALUNO - PLANO): Vínculo entre alunos e seus planos contratados.

Relatórios e Consultas Frequentes:
- Gestão de Aulas: Aulas com maior/menor ocupação, frequência por aluno.
- Desempenho dos Funcionários: Carga horária e avaliações de instrutores.
- Controle Financeiro: Receita por tipo de plano, pagamentos pendentes.
- Gestão de Equipamentos: Frequência de uso e planejamento de manutenções.
- Análise de Alunos: Segmentação por faixa etária e padrões de frequência.
- Controle de Acesso: Identificação de horários de pico.

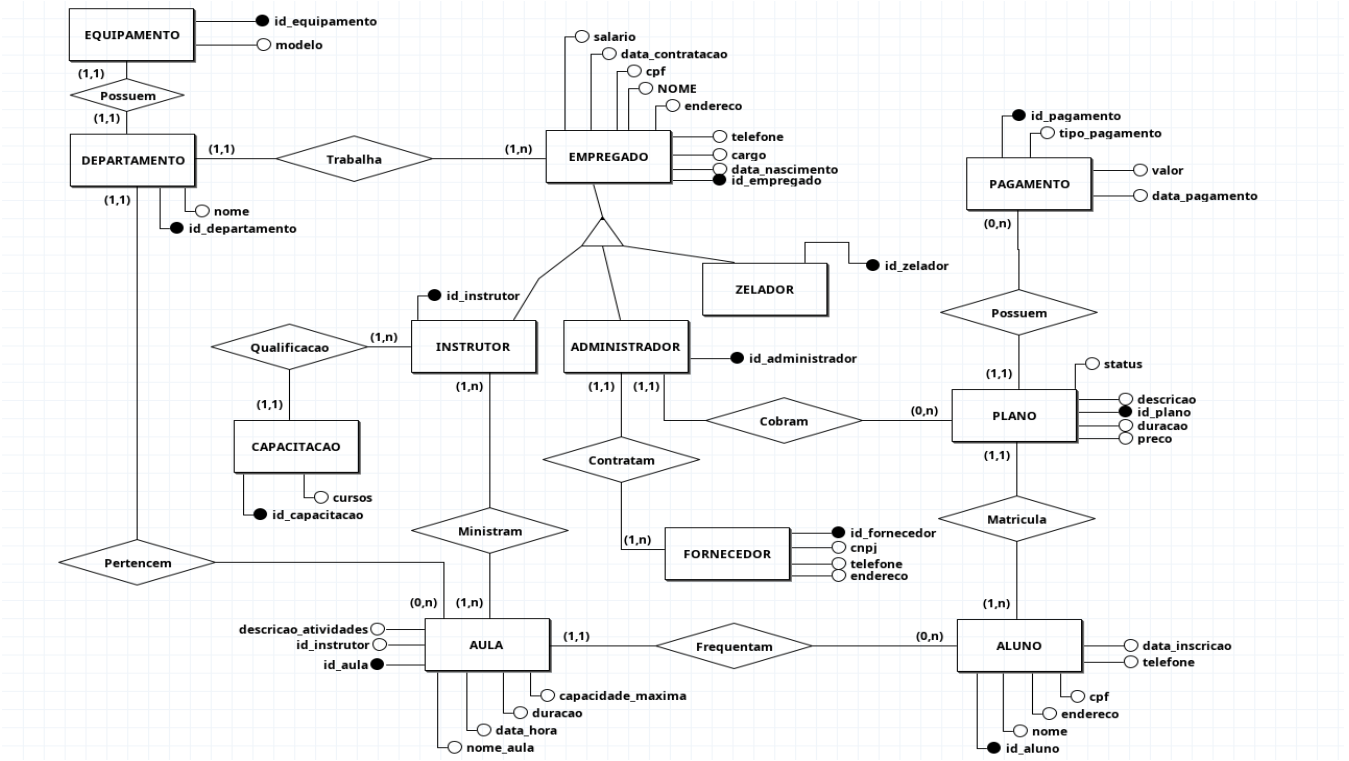
Benefícios do Sistema:
- Otimização de Recursos: Integração de dados e automação de processos.
- Gestão Estratégica: Relatórios detalhados para suporte à decisão.
- Melhoria na Experiência do Aluno: Personalização de planos e organização eficiente.
- Sustentabilidade: Monitoramento eficiente de recursos físicos e humanos.

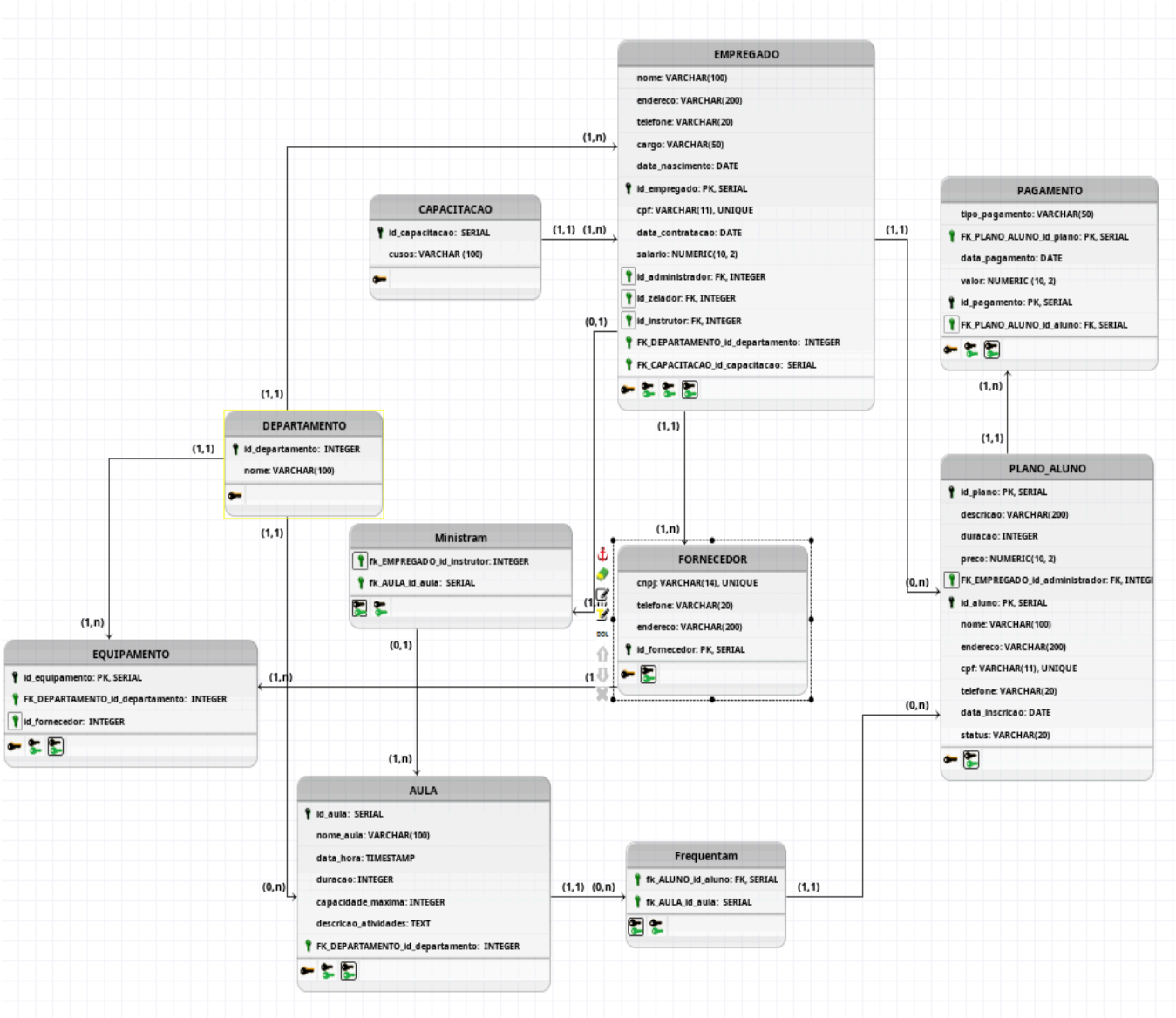
O código-fonte completo do Sistema de Gerenciamento de Academia, incluindo scripts SQL, documentação e implementação, está disponível no meu repositório GitHub:

Repositório GitHub: **https://github.com/jlsgodev/sistema-academia-bd2**

Neste repositório, você encontrará todos os arquivos necessários para reproduzir o sistema de banco de dados, incluindo o modelo de dados e os scripts para a criação de tabelas, consultas, funções, e muito mais. GitHub

b) Modelagem Conceitual e Modelagem Lógica





c) Projeto Físico DDL

```
-- Definição de Domínios
-- Usamos domínios para garantir consistência e reutilização de tipos
em várias tabelas.

CREATE DOMAIN dom_varchar_100 AS VARCHAR(100);  -- Nome de tamanho fixo
para campos textuais
CREATE DOMAIN dom_varchar_200 AS VARCHAR(200);  -- Usado para endereços
mais longos
CREATE DOMAIN dom_varchar_20 AS VARCHAR(20);    -- Para armazenar
números de telefone
CREATE DOMAIN dom_varchar_11 AS VARCHAR(11);    -- Para armazenar CPF
CREATE DOMAIN dom_varchar_14 AS VARCHAR(14);    -- Para armazenar CNPJ
CREATE DOMAIN dom_varchar_50 AS VARCHAR(50);    -- Para tipo de
pagamento e outros textos curtos
CREATE DOMAIN dom_numeric_2_2 AS NUMERIC(20, 2); -- Para valores
monetários com 2 casas decimais
CREATE DOMAIN dom_date AS DATE;                 -- Para armazenar datas
simples
CREATE DOMAIN dom_timestamp AS TIMESTAMP;       -- Para data e hora
CREATE DOMAIN dom_status AS VARCHAR(20) CHECK (VALUE IN ('ativo',
'inativo')); -- Controle de status

-- Tabela DEPARTAMENTO
-- Tabelas de referência a entidades simples e com dados não
duplicados.

CREATE TABLE DEPARTAMENTO (
   id_departamento SERIAL PRIMARY KEY,  -- SERIAL para auto incremento
   nome dom_varchar_100 NOT NULL  -- Garantimos que o nome do
departamento é obrigatório
);

-- Tabela CAPACITACAO
CREATE TABLE CAPACITACAO (
   id_capacitacao SERIAL PRIMARY KEY,  -- Auto incremento para ID
   cursos dom_varchar_100 NOT NULL  -- Campo obrigatório para o nome do
curso
);

-- Tabela FORNECEDOR
-- Armazenamento de fornecedores, onde garantimos que o CNPJ seja único
e não nulo.

CREATE TABLE FORNECEDOR (
   id_fornecedor SERIAL PRIMARY KEY,  -- SERIAL para auto incremento
```

```
    cnpj dom_varchar_14 UNIQUE NOT NULL,  -- CNPJ único, não pode ser
nulo
    endereco dom_varchar_200 NOT NULL  -- Endereço obrigatório
);

-- Tabela EMPREGADO
-- Tabela com detalhes de funcionários, incluindo referências a outros
empregados como administradores, zeladores e instrutores.

CREATE TABLE EMPREGADO (
    id_empregado SERIAL PRIMARY KEY,  -- Auto incremento
    nome dom_varchar_100 NOT NULL,  -- Nome obrigatório
    endereco dom_varchar_200,  -- Endereço opcional
    telefone dom_varchar_20,  -- Telefone opcional
    cargo dom_varchar_100,  -- Cargo do empregado
    data_nascimento dom_date,  -- Data de nascimento
    cpf dom_varchar_11 UNIQUE NOT NULL,  -- CPF único e obrigatório
    data_contratacao dom_date,  -- Data de contratação
    salario dom_numeric_2_2,  -- Salário com 2 casas decimais
    id_administrador INTEGER,  -- Referência ao administrador (auto
referência)
    id_zelador INTEGER,  -- Referência ao zelador (auto referência)
    id_instrutor INTEGER,  -- Referência ao instrutor (auto referência)
    id_departamento INTEGER REFERENCES DEPARTAMENTO(id_departamento) ON
DELETE SET NULL,  -- Relacionamento com DEPARTAMENTO
    id_capacitacao INTEGER REFERENCES CAPACITACAO(id_capacitacao) ON
DELETE SET NULL,  -- Relacionamento com CAPACITACAO
    -- Regras de integridade referencial para garantir a consistência
dos dados
    CONSTRAINT fk_administrador FOREIGN KEY (id_administrador)
REFERENCES EMPREGADO(id_empregado) ON DELETE SET NULL,
    CONSTRAINT fk_zelador FOREIGN KEY (id_zelador) REFERENCES
EMPREGADO(id_empregado) ON DELETE SET NULL,
    CONSTRAINT fk_instrutor FOREIGN KEY (id_instrutor) REFERENCES
EMPREGADO(id_empregado) ON DELETE SET NULL
);

-- Tabela EQUIPAMENTO
-- Equipamentos são associados a departamentos e fornecedores.

CREATE TABLE EQUIPAMENTO (
    id_equipamento SERIAL PRIMARY KEY,  -- Auto incremento para ID
    nome dom_varchar_100 NOT NULL,  -- Nome do equipamento
    id_departamento INTEGER REFERENCES DEPARTAMENTO(id_departamento) ON
DELETE CASCADE,  -- Relacionamento com DEPARTAMENTO
    id_fornecedor INTEGER REFERENCES FORNECEDOR(id_fornecedor) ON DELETE
SET NULL  -- Relacionamento com FORNECEDOR
```

```
);
-- Tabela AULA
-- Detalhes sobre as aulas, incluindo horário, instrutores e
capacidade.

CREATE TABLE AULA (
   id_aula SERIAL PRIMARY KEY,  -- Auto incremento para ID da aula
   nome_aula dom_varchar_100 NOT NULL,  -- Nome obrigatório para a aula
   data_hora dom_timestamp NOT NULL,  -- Data e hora obrigatória da
aula
   duracao INTEGER CHECK (duracao > 0),  -- Duracao maior que 0
   id_instrutor INTEGER REFERENCES EMPREGADO(id_empregado) ON DELETE
SET NULL,  -- Relacionamento com o instrutor
   capacidade_maxima INTEGER CHECK (capacidade_maxima > 0),  --
Capacidade maior que 0
   descricao_atividades TEXT,  -- Descrição das atividades realizadas
   id_departamento INTEGER REFERENCES DEPARTAMENTO(id_departamento) ON
DELETE CASCADE  -- Relacionamento com DEPARTAMENTO
);

-- Tabela PLANO_ALUNO
-- Relacionamento de planos com alunos, com informações detalhadas do
aluno.

CREATE TABLE PLANO_ALUNO (
   id_plano SERIAL NOT NULL,  -- Auto incremento para ID do plano
   id_aluno SERIAL NOT NULL,  -- ID do aluno
   descricao dom_varchar_200,  -- Descrição do plano
   duracao INTEGER CHECK (duracao > 0),  -- Duracao maior que 0
   preco dom_numeric_2_2 CHECK (preco >= 0),  -- Preço não pode ser
negativo
   id_administrador INTEGER REFERENCES EMPREGADO(id_empregado) ON
DELETE SET NULL,  -- Referência ao administrador responsável
   nome dom_varchar_100 NOT NULL,  -- Nome do aluno
   endereco dom_varchar_200 NOT NULL,  -- Endereço do aluno
   telefone dom_varchar_20,  -- Telefone do aluno
   cpf dom_varchar_11 UNIQUE NOT NULL,  -- CPF único do aluno
   data_insercao dom_date DEFAULT CURRENT_DATE,  -- Data de inserção,
padrão para a data atual
   status dom_status NOT NULL,  -- Status do plano
   PRIMARY KEY (id_plano, id_aluno)  -- Chave primária composta
);

-- Tabela PAGAMENTO
-- Relacionamento entre planos e pagamentos realizados pelos alunos.

CREATE TABLE PAGAMENTO (
```

```
    id_pagamento SERIAL PRIMARY KEY,  -- Auto incremento para ID do
pagamento
    tipo_pagamento dom_varchar_50 NOT NULL,  -- Tipo de pagamento
(cartão, boleto, etc)
    data_pagamento dom_date NOT NULL DEFAULT CURRENT_DATE,  -- Data do
pagamento
    valor dom_numeric_2_2 CHECK (valor >= 0),  -- Preço não pode ser
negativo
    id_plano INTEGER NOT NULL,  -- Referência ao plano do aluno
    id_aluno INTEGER NOT NULL,  -- Referência ao aluno
    FOREIGN KEY (id_plano, id_aluno) REFERENCES PLANO_ALUNO(id_plano,
id_aluno) ON DELETE CASCADE  -- Chave estrangeira composta
);

-- Tabela Ministram (Associação N:M entre EMPREGADO e AULA)
-- Registro de aulas ministradas pelos instrutores.

CREATE TABLE Ministram (
    id_instrutor INTEGER NOT NULL,  -- ID do instrutor
    id_aula INTEGER NOT NULL,  -- ID da aula
    PRIMARY KEY (id_instrutor, id_aula),  -- Chave primária composta
    FOREIGN KEY (id_instrutor) REFERENCES EMPREGADO(id_empregado) ON
DELETE CASCADE,  -- Relacionamento com EMPREGADO
    FOREIGN KEY (id_aula) REFERENCES AULA(id_aula) ON DELETE CASCADE  --
Relacionamento com AULA
);

-- Tabela Frequentam (Associação N:M entre PLANO_ALUNO e AULA)
-- Registro de quais alunos frequentam quais aulas.

CREATE TABLE Frequentam (
    id_plano INTEGER NOT NULL,  -- ID do plano do aluno
    id_aluno INTEGER NOT NULL,  -- ID do aluno
    id_aula INTEGER NOT NULL,  -- ID da aula
    PRIMARY KEY (id_plano, id_aluno, id_aula),  -- Chave primária
composta
    FOREIGN KEY (id_plano, id_aluno) REFERENCES PLANO_ALUNO(id_plano,
id_aluno) ON DELETE CASCADE,  -- Relacionamento com PLANO_ALUNO
    FOREIGN KEY (id_aula) REFERENCES AULA(id_aula) ON DELETE CASCADE  --
Relacionamento com AULA
);

-- Índices para melhorar o desempenho nas consultas frequentes

CREATE INDEX idx_empregado_cpf ON EMPREGADO(cpf);  -- Index para
melhorar buscas por CPF de empregado
CREATE INDEX idx_plano_aluno_cpf ON PLANO_ALUNO(cpf);  -- Index
```

d) Scripts SQL para consultas, algumas usando diferentes tipos de JOIN.

```sql
-- Consulta 1: Aulas e seus Instrutores por Departamento
SELECT a.nome_aula, a.data_hora, e.nome as instrutor, d.nome as
departamento
FROM AULA a
INNER JOIN EMPREGADO e ON a.id_instrutor = e.id_empregado  -- Junta a
tabela AULA com EMPREGADO para obter o nome do instrutor
INNER JOIN DEPARTAMENTO d ON a.id_departamento = d.id_departamento  --
Junta a tabela AULA com DEPARTAMENTO para obter o nome do departamento
ORDER BY a.data_hora;  -- Ordena os resultados pela data e hora da aula

-- Consulta 2: Pagamentos de Alunos Ativos
SELECT pa.nome as aluno, pa.cpf, p.tipo_pagamento, p.valor,
p.data_pagamento
FROM PLANO_ALUNO pa
LEFT JOIN PAGAMENTO p ON pa.id_plano = p.id_plano AND pa.id_aluno =
p.id_aluno  -- Junta a tabela PLANO_ALUNO com PAGAMENTO para obter os
pagamentos dos alunos
WHERE pa.status = 'ativo'  -- Filtra apenas os alunos com plano ativo
ORDER BY p.data_pagamento DESC;  -- Ordena os resultados pela data de
pagamento em ordem decrescente

-- Consulta 3: Equipamentos por Departamento e Fornecedor
SELECT d.nome as departamento,
      e.nome as equipamento,
      f.cnpj as fornecedor_cnpj
FROM EQUIPAMENTO e
JOIN DEPARTAMENTO d ON e.id_departamento = d.id_departamento  -- Junta
a tabela EQUIPAMENTO com DEPARTAMENTO para obter o nome do departamento
JOIN FORNECEDOR f ON e.id_fornecedor = f.id_fornecedor  -- Junta a
tabela EQUIPAMENTO com FORNECEDOR para obter o CNPJ do fornecedor
ORDER BY d.nome;  -- Ordena os resultados pelo nome do departamento

-- Consulta 4: Total de Aulas por Instrutor
SELECT e.nome, e.cpf,
      (SELECT COUNT(*)
       FROM MINISTRAM m
       WHERE m.id_instrutor = e.id_empregado) as total_aulas  --
Subconsulta para contar o total de aulas ministradas por cada instrutor
FROM EMPREGADO e
WHERE e.cargo = 'Instrutor'  -- Filtra apenas os empregados com cargo
de instrutor
ORDER BY total_aulas DESC;  -- Ordena os resultados pelo total de aulas
em ordem decrescente

-- Consulta 5: Estatísticas por Departamento
```

```sql
SELECT d.nome as departamento,
      COUNT(DISTINCT a.id_aula) as total_aulas,  -- Conta o total de
aulas distintas por departamento
      COUNT(DISTINCT f.id_aluno) as total_alunos,  -- Conta o total de
alunos distintos por departamento
      AVG(a.capacidade_maxima) as media_capacidade  -- Calcula a média
da capacidade máxima das aulas por departamento
FROM DEPARTAMENTO d
LEFT JOIN AULA a ON d.id_departamento = a.id_departamento  -- Junta a
tabela DEPARTAMENTO com AULA para obter as aulas por departamento
LEFT JOIN FREQUENTAM f ON a.id_aula = f.id_aula  -- Junta a tabela AULA
com FREQUENTAM para obter os alunos por aula
GROUP BY d.nome  -- Agrupa os resultados pelo nome do departamento
ORDER BY total_alunos DESC;  -- Ordena os resultados pelo total de
alunos em ordem decrescente
```

e) Scripts SQL para Stored Procedures

```sql
-- Stored Procedure 1: Cadastro de Novo Aluno
-- Esta SP realiza todo o processo de matrícula de um novo aluno na
academia
CREATE OR REPLACE PROCEDURE sp_cadastrar_aluno(
   p_nome VARCHAR(100),          -- Nome completo do aluno
   p_cpf VARCHAR(11),           -- CPF do aluno (apenas números)
   p_endereco VARCHAR(200),     -- Endereço completo
   p_telefone VARCHAR(20),      -- Telefone com DDD
   p_tipo_plano VARCHAR(200),   -- Descrição do plano escolhido
   p_duracao INTEGER,           -- Duração em dias do plano
   p_preco NUMERIC(20,2),       -- Valor do plano
   p_id_admin INTEGER           -- ID do administrador responsável
)
LANGUAGE plpgsql
AS $$
DECLARE
   v_id_plano INTEGER;          -- Variável para armazenar o ID do plano
gerado
   v_id_aluno INTEGER;          -- Variável para armazenar o ID do aluno
gerado
BEGIN
   -- Insere os dados do aluno e do plano
   INSERT INTO PLANO_ALUNO(
       nome, cpf, endereco, telefone,
       descricao, duracao, preco,
       id_administrador, status
```

```
    )
    VALUES (
        p_nome, p_cpf, p_endereco, p_telefone,
        p_tipo_plano, p_duracao, p_preco,
        p_id_admin, 'ativo'
    )
    RETURNING id_plano, id_aluno INTO v_id_plano, v_id_aluno;  --
Retorna os IDs gerados

    -- Registra o pagamento inicial da matrícula
    INSERT INTO PAGAMENTO(
        tipo_pagamento, valor, id_plano, id_aluno
    )
    VALUES (
        'Matrícula', p_preco, v_id_plano, v_id_aluno
    );

    COMMIT;                    -- Confirma a transação
END;
$$;

-- Stored Procedure 2 : Registro de Frequência em Aulas
CREATE OR REPLACE PROCEDURE sp_registrar_frequencia(
    p_cpf_aluno VARCHAR(11),   -- CPF do aluno que participará da aula
    p_id_aula INTEGER          -- ID da aula que será frequentada
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_id_plano INTEGER;        -- Armazena o ID do plano do aluno
    v_id_aluno INTEGER;        -- Armazena o ID do aluno
    v_capacidade INTEGER;      -- Armazena a capacidade máxima da aula
    v_ocupacao INTEGER;        -- Armazena o número atual de alunos na
aula
BEGIN
    -- Busca os dados do aluno
    SELECT id_plano, id_aluno
    INTO v_id_plano, v_id_aluno
    FROM PLANO_ALUNO
    WHERE cpf = p_cpf_aluno AND status = 'ativo';

    -- Verifica a capacidade da aula
    SELECT capacidade_maxima,
           (SELECT COUNT(*) FROM FREQUENTAM WHERE id_aula = p_id_aula)
    INTO v_capacidade, v_ocupacao
    FROM AULA
    WHERE id_aula = p_id_aula;
```

```
    -- Verifica se há vagas disponíveis
    IF v_ocupacao >= v_capacidade THEN
        RAISE EXCEPTION 'Aula está com capacidade máxima atingida';
    END IF;

    -- Registra a frequência do aluno
    INSERT INTO FREQUENTAM(id_plano, id_aluno, id_aula)
    VALUES (v_id_plano, v_id_aluno, p_id_aula);

EXCEPTION
    WHEN NO_DATA_FOUND THEN   -- Tratamento de erro quando aluno não é
encontrado
        RAISE EXCEPTION 'Aluno não encontrado ou plano inativo';
END;
$$;


-- Stored Procedure 3: Relatório de Frequência por Aluno
CREATE OR REPLACE PROCEDURE sp_relatorio_frequencia_aluno(
    p_cpf_aluno VARCHAR(11)    -- CPF do aluno para buscar frequência
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_nome_aluno VARCHAR(100);  -- Armazena nome do aluno
    v_total_aulas INTEGER;      -- Contador de aulas frequentadas
BEGIN
    -- Busca nome do aluno
    SELECT nome INTO v_nome_aluno
    FROM PLANO_ALUNO
    WHERE cpf = p_cpf_aluno AND status = 'ativo';

    -- Conta total de aulas frequentadas
    SELECT COUNT(*) INTO v_total_aulas
    FROM FREQUENTAM f
    JOIN PLANO_ALUNO pa ON f.id_plano = pa.id_plano
    WHERE pa.cpf = p_cpf_aluno;

    -- Exibe o relatório
    RAISE NOTICE 'Relatório de Frequência';
    RAISE NOTICE 'Aluno: %', v_nome_aluno;
    RAISE NOTICE 'Total de aulas frequentadas: %', v_total_aulas;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE EXCEPTION 'Aluno não encontrado ou inativo';
```

```
END;
$$
;

-- Teste da SP1: Cadastrar novo aluno
CALL sp_cadastrar_aluno(
   'João Silva',  -- Nome do aluno
   '12345678901',  -- CPF do aluno
   'Rua Exemplo, 123',  -- Endereço do aluno
   '48999999999',  -- Telefone do aluno
   'Plano Mensal',  -- Tipo de plano escolhido
   30,  -- Duração do plano em dias
   99.90,  -- Preço do plano
   1  -- ID do administrador responsável
);

-- Teste da SP2
CALL sp_registrar_frequencia('12345678901', 1);  -- CPF do aluno e ID
da aula

-- Teste da SP3
CALL sp_relatorio_frequencia_aluno('12345678901');  -- CPF do aluno
para gerar relatório
```

f) Scripts SQL para Functions

```
-- Function 1: Calcular total faturado por período
-- Parâmetros de entrada:
--   data_inicio: data inicial do período
--   data_fim: data final do período
-- Retorno: valor total faturado no período
CREATE OR REPLACE FUNCTION fn_calcular_faturamento(
    data_inicio DATE,  -- Data inicial do período
    data_fim DATE  -- Data final do período
)
RETURNS NUMERIC(20,2)  -- Retorna o valor total faturado no período
LANGUAGE plpgsql
AS $$
DECLARE
    total_faturado NUMERIC(20,2);  -- Variável para armazenar o total
faturado
BEGIN
    -- Calcula o total faturado no período especificado
    SELECT COALESCE(SUM(valor), 0)
    INTO total_faturado
    FROM PAGAMENTO
```

```
    WHERE data_pagamento BETWEEN data_inicio AND data_fim;

    RETURN total_faturado;  -- Retorna o total faturado
END;
$$
;

-- Function 2: Verificar disponibilidade de vaga em aula
-- Parâmetros de entrada:
--   p_id_aula: ID da aula a ser verificada
-- Retorno: boolean (true se há vagas, false se não há)
CREATE OR REPLACE FUNCTION fn_verificar_disponibilidade_aula(
    p_id_aula INTEGER  -- ID da aula a ser verificada
)
RETURNS BOOLEAN  -- Retorna true se há vagas disponíveis, false caso
contrário
LANGUAGE plpgsql
AS $$
DECLARE
    v_capacidade INTEGER;  -- Variável para armazenar a capacidade máxima
da aula
    v_ocupacao INTEGER;  -- Variável para armazenar o número atual de
alunos na aula
BEGIN
    -- Busca capacidade máxima da aula
    SELECT capacidade_maxima
    INTO v_capacidade
    FROM AULA
    WHERE id_aula = p_id_aula;

    -- Conta alunos matriculados
    SELECT COUNT(*)
    INTO v_ocupacao
    FROM FREQUENTAM
    WHERE id_aula = p_id_aula;

    -- Retorna true se há vagas disponíveis
    RETURN v_ocupacao < v_capacidade;
END;
$$
;

-- Function 3: Calcular idade do aluno
-- Parâmetros de entrada:
--   p_cpf: CPF do aluno
-- Retorno: idade em anos
CREATE OR REPLACE FUNCTION fn_calcular_idade_aluno(
    p_cpf VARCHAR(11)  -- CPF do aluno
)
RETURNS INTEGER  -- Retorna a idade em anos
LANGUAGE plpgsql
```

```
AS $$
DECLARE
    v_data_nascimento DATE;  -- Variável para armazenar a data de
nascimento do aluno
BEGIN
    -- Busca a data de nascimento do aluno na tabela EMPREGADO
    SELECT data_nascimento
    INTO v_data_nascimento
    FROM EMPREGADO
    WHERE cpf = p_cpf;

    -- Calcula e retorna a idade do aluno em anos
    RETURN EXTRACT(YEAR FROM age(current_date, v_data_nascimento));
END;
$$
;

-- Teste Function 1: Calcular faturamento
SELECT fn_calcular_faturamento('2024-01-01', '2024-12-31') as
faturamento_anual;  -- Calcula o faturamento anual para o ano de 2024

-- Teste Function 2: Verificar disponibilidade
SELECT fn_verificar_disponibilidade_aula(1) as tem_vaga_disponivel;  --
Verifica se há vagas disponíveis para a aula com ID 1

-- Teste Function 3: Calcular idade
SELECT fn_calcular_idade_aluno('12345678901') as idade_aluno;  -- Calcula
a idade do aluno com o CPF '12345678901'
```

g) Scripts SQL para Views

```
-- View 1: Visão geral das aulas com detalhes
-- Esta view fornece informações completas sobre as aulas,
-- incluindo instrutor responsável e departamento
CREATE OR REPLACE VIEW vw_detalhes_aulas AS
SELECT
   a.nome_aula,  -- Nome da aula
   a.data_hora,  -- Data e hora da aula
   a.duracao,  -- Duração da aula
   a.capacidade_maxima,  -- Capacidade máxima de alunos na aula
   e.nome as instrutor,  -- Nome do instrutor responsável
   d.nome as departamento,  -- Nome do departamento
   (SELECT COUNT(*)
    FROM FREQUENTAM f
    WHERE f.id_aula = a.id_aula) as alunos_matriculados  -- Número de
alunos matriculados na aula
FROM AULA a
```

```sql
JOIN EMPREGADO e ON a.id_instrutor = e.id_empregado  -- Junta a tabela
AULA com EMPREGADO para obter o nome do instrutor
JOIN DEPARTAMENTO d ON a.id_departamento = d.id_departamento;  -- Junta
a tabela AULA com DEPARTAMENTO para obter o nome do departamento

-- View 2: Relatório financeiro de alunos
-- Esta view apresenta um resumo financeiro dos alunos,
-- incluindo planos e pagamentos realizados
CREATE OR REPLACE VIEW vw_financeiro_alunos AS
SELECT
   pa.nome as aluno,  -- Nome do aluno
   pa.cpf,  -- CPF do aluno
   pa.descricao as plano,  -- Descrição do plano do aluno
   pa.preco as valor_plano,  -- Valor do plano do aluno
   pa.status,  -- Status do plano do aluno
   COUNT(p.id_pagamento) as total_pagamentos,  -- Total de pagamentos
realizados pelo aluno
   SUM(p.valor) as total_pago  -- Valor total pago pelo aluno
FROM PLANO_ALUNO pa
LEFT JOIN PAGAMENTO p ON pa.id_plano = p.id_plano
   AND pa.id_aluno = p.id_aluno  -- Junta a tabela PLANO_ALUNO com
PAGAMENTO para obter os pagamentos dos alunos
GROUP BY pa.nome, pa.cpf, pa.descricao, pa.preco, pa.status;  -- Agrupa
os resultados pelos dados do aluno

-- View 3: Status dos equipamentos por departamento
-- Esta view mostra a distribuição de equipamentos
-- entre os departamentos e seus fornecedores
CREATE OR REPLACE VIEW vw_equipamentos_departamento AS
SELECT
   d.nome as departamento,  -- Nome do departamento
   e.nome as equipamento,  -- Nome do equipamento
   f.cnpj as fornecedor,  -- CNPJ do fornecedor do equipamento
   COUNT(e.id_equipamento) OVER (PARTITION BY d.id_departamento) as
total_equipamentos  -- Total de equipamentos por departamento
FROM DEPARTAMENTO d
LEFT JOIN EQUIPAMENTO e ON d.id_departamento = e.id_departamento  --
Junta a tabela DEPARTAMENTO com EQUIPAMENTO para obter os equipamentos
por departamento
LEFT JOIN FORNECEDOR f ON e.id_fornecedor = f.id_fornecedor;  -- Junta
a tabela EQUIPAMENTO com FORNECEDOR para obter o CNPJ do fornecedor

-- Consultar detalhes das aulas
SELECT * FROM vw_detalhes_aulas;  -- Consulta a view vw_detalhes_aulas
para obter detalhes das aulas

-- Consultar relatório financeiro
```

```sql
SELECT * FROM vw_financeiro_alunos;  -- Consulta a view
vw_financeiro_alunos para obter o relatório financeiro dos alunos

-- Consultar equipamentos por departamento
SELECT * FROM vw_equipamentos_departamento;  -- Consulta a view
vw_equipamentos_departamento para obter o status dos equipamentos por
departamento
```

h) Scripts SQL para Triggers

```sql
-- Criar tabela de log simplificada
-- Esta tabela armazena as alterações de valores dos planos dos alunos
CREATE TABLE log_alteracao_valores (
   id_log SERIAL PRIMARY KEY,  -- Identificador único do log
   data_alteracao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  -- Data e hora
da alteração
   id_plano INTEGER,  -- ID do plano alterado
   id_aluno INTEGER,  -- ID do aluno cujo plano foi alterado
   valor_anterior dom_numeric_2_2,  -- Valor anterior do plano
   valor_novo dom_numeric_2_2,  -- Novo valor do plano
   usuario_alteracao VARCHAR(50)  -- Usuário que realizou a alteração
);

-- Nova função para a trigger
-- Esta função insere um registro na tabela de log quando o valor de um
plano é alterado
CREATE OR REPLACE FUNCTION tf_registrar_alteracao_valor()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.preco IS DISTINCT FROM NEW.preco THEN  -- Verifica se o valor
do plano foi alterado
       INSERT INTO log_alteracao_valores (
           id_plano,
           id_aluno,
           valor_anterior,
           valor_novo,
           usuario_alteracao
       )
       VALUES (
           OLD.id_plano,  -- ID do plano antes da alteração
           OLD.id_aluno,  -- ID do aluno antes da alteração
           OLD.preco,  -- Valor anterior do plano
           NEW.preco,  -- Novo valor do plano
           current_user  -- Usuário que realizou a alteração
       );
   END IF;
```

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

-- Nova trigger
-- Esta trigger chama a função tf_registrar_alteracao_valor após uma
atualização na tabela PLANO_ALUNO
CREATE TRIGGER tg_registrar_alteracao_valor
AFTER UPDATE ON PLANO_ALUNO
FOR EACH ROW
EXECUTE FUNCTION tf_registrar_alteracao_valor();

-- Verificar preço atual do plano
-- Consulta para verificar o preço atual do plano de um aluno
específico
SELECT id_plano, id_aluno, preco, status
FROM PLANO_ALUNO
WHERE id_plano = 1 AND id_aluno = 1;

-- Verificar se a tabela de log está vazia
-- Consulta para verificar se a tabela de log está vazia
SELECT * FROM log_alteracao_valores;

-- Atualizar o preço de um plano existente
-- Atualiza o preço de um plano específico para testar a trigger
UPDATE PLANO_ALUNO
SET preco = 150.00
WHERE id_plano = 1 AND id_aluno = 1;

-- Verificar se a alteração foi registrada no log
-- Consulta para verificar se a alteração foi registrada na tabela de
log
SELECT * FROM log_alteracao_valores
ORDER BY data_alteracao DESC;
```

i) Definição de Usuários e permissões

```
-- Criar usuário administrador com todas as permissões
CREATE ROLE admin_academia WITH
    LOGIN  -- Permite login
    PASSWORD 'adminIFC'  -- Define a senha do usuário
    CREATEDB  -- Permite criar bancos de dados
    CREATEROLE;  -- Permite criar novos roles

-- Conceder todas as permissões nas tabelas
```

```sql
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin_academia;
-- Concede todas as permissões em todas as tabelas
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO
admin_academia;  -- Concede todas as permissões em todas as sequências

-- Conceder permissão de execução em todas as funções
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO admin_academia;  --
Concede permissão de execução em todas as funções

-- Criar usuário instrutor com permissões limitadas
CREATE ROLE instrutor_academia WITH
   LOGIN  -- Permite login
   PASSWORD 'instrutorIFC'  -- Define a senha do usuário
   NOINHERIT;  -- Não herda permissões de outros roles

-- Permissões específicas para instrutores
GRANT SELECT ON TABLE EMPREGADO TO instrutor_academia;  -- Permite
selecionar dados da tabela EMPREGADO
GRANT SELECT, INSERT ON TABLE FREQUENTAM TO instrutor_academia;  --
Permite selecionar e inserir dados na tabela FREQUENTAM
GRANT SELECT ON TABLE PLANO_ALUNO TO instrutor_academia;  -- Permite
selecionar dados da tabela PLANO_ALUNO
GRANT SELECT, UPDATE ON TABLE AULA TO instrutor_academia;  -- Permite
selecionar e atualizar dados na tabela AULA
GRANT SELECT ON TABLE DEPARTAMENTO TO instrutor_academia;  -- Permite
selecionar dados da tabela DEPARTAMENTO
GRANT SELECT ON TABLE EQUIPAMENTO TO instrutor_academia;  -- Permite
selecionar dados da tabela EQUIPAMENTO

-- Revogar acesso a informações sensíveis do instrutor
REVOKE SELECT (cpf, salario, data_contratacao) ON TABLE EMPREGADO FROM
instrutor_academia;  -- Revoga permissão de selecionar campos sensíveis
da tabela EMPREGADO
REVOKE SELECT, INSERT, DELETE ON TABLE EMPREGADO FROM
instrutor_academia;  -- Revoga permissão de selecionar, inserir e
deletar dados na tabela EMPREGADO
REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public FROM
instrutor_academia;  -- Revoga todas as permissões em todas as tabelas
```

Conclusão

Este trabalho final consolidou os conhecimentos adquiridos na disciplina Banco de Dados II, abrangendo desde a modelagem conceitual até a implementação física em PostgreSQL. Foi desenvolvido um sistema de gerenciamento para academias, atendendo às necessidades de organização de dados, geração de relatórios e automação de processos.

A aplicação prática envolveu:

Modelagem Estruturada: A criação de modelos conceitual e lógico seguiu boas práticas de design e normalização de dados.

Implementação SQL Avançada: Scripts SQL incluíram definições completas de tabelas, domínios, consultas com diferentes tipos de JOIN, stored procedures, functions, views e triggers.

Segurança e Permissões: Definições claras de usuários e permissões reforçaram a segurança no controle de acesso.

Os desafios enfrentados, como a adaptação das stored procedures às regras de negócio e a implementação de funcionalidades específicas, foram superados por meio de pesquisa e prática. Ferramentas como o PostgreSQL 16, pgAdmin4 e o terminal Linux foram essenciais no processo.

Aprendizado e Reflexões

Este projeto destacou a importância da organização e estruturação de dados para atender às demandas reais de um sistema de informações. Foi uma oportunidade de aplicar conceitos teóricos de autores renomados como C. J. Date, Elmasri & Navathe e Silberschatz, além de explorar materiais práticos como vídeos do canal Bóson Treinamentos.

O trabalho proporcionou um aprendizado prático valioso, preparando para desafios futuros na área de banco de dados e sistemas de informação.

Agradecimentos

Gostaria de expressar minha gratidão ao professor Angelo Augusto Frozza, pela dedicação e clareza na transmissão dos conhecimentos ao longo da disciplina Banco de Dados II. As aulas foram fundamentais para o desenvolvimento deste trabalho e para o aprendizado consistente sobre o tema.