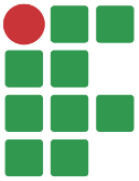


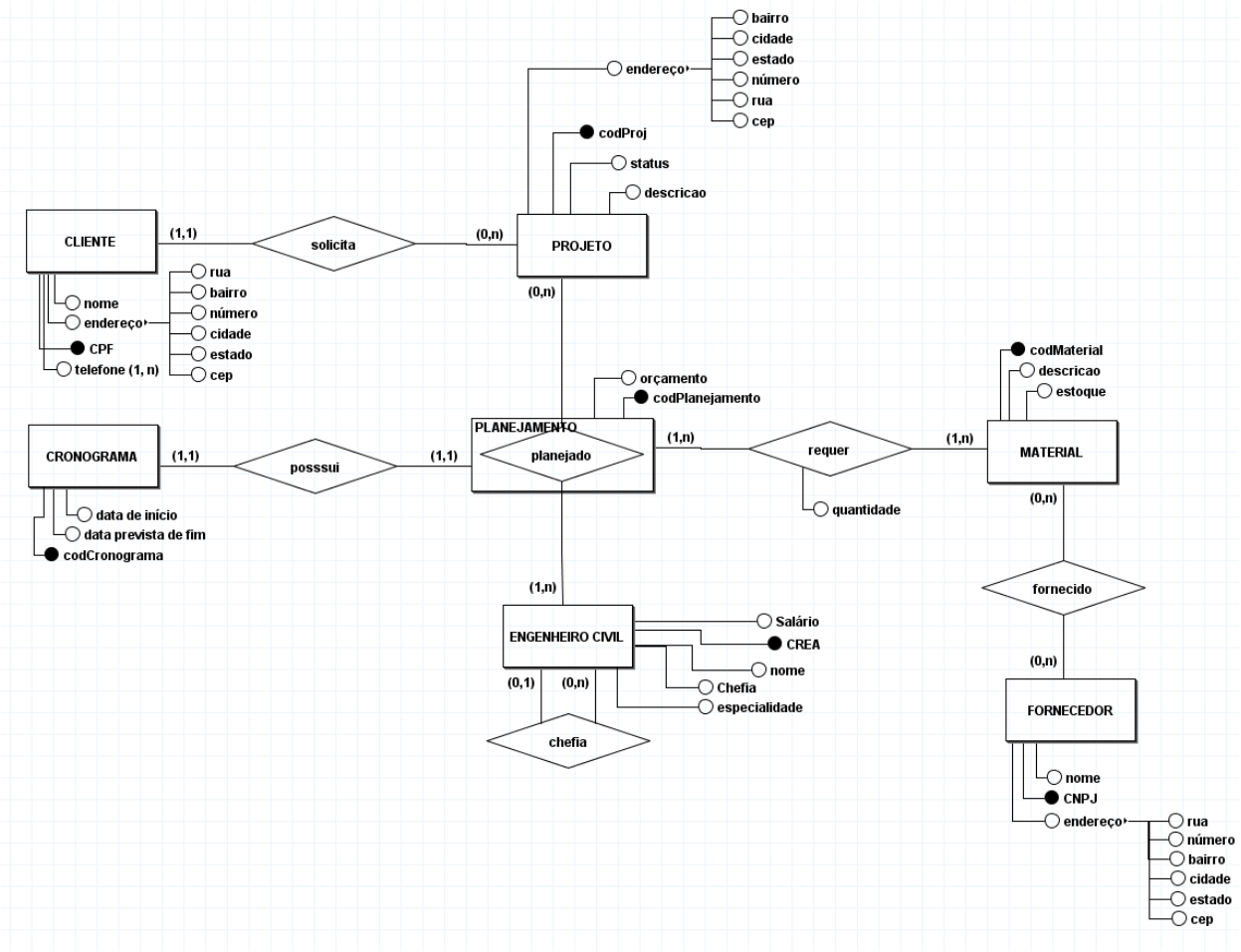
INSTITUTO FEDERAL
Paraíba | Campus João Pessoa

Projeto de Banco de Dados Relacional

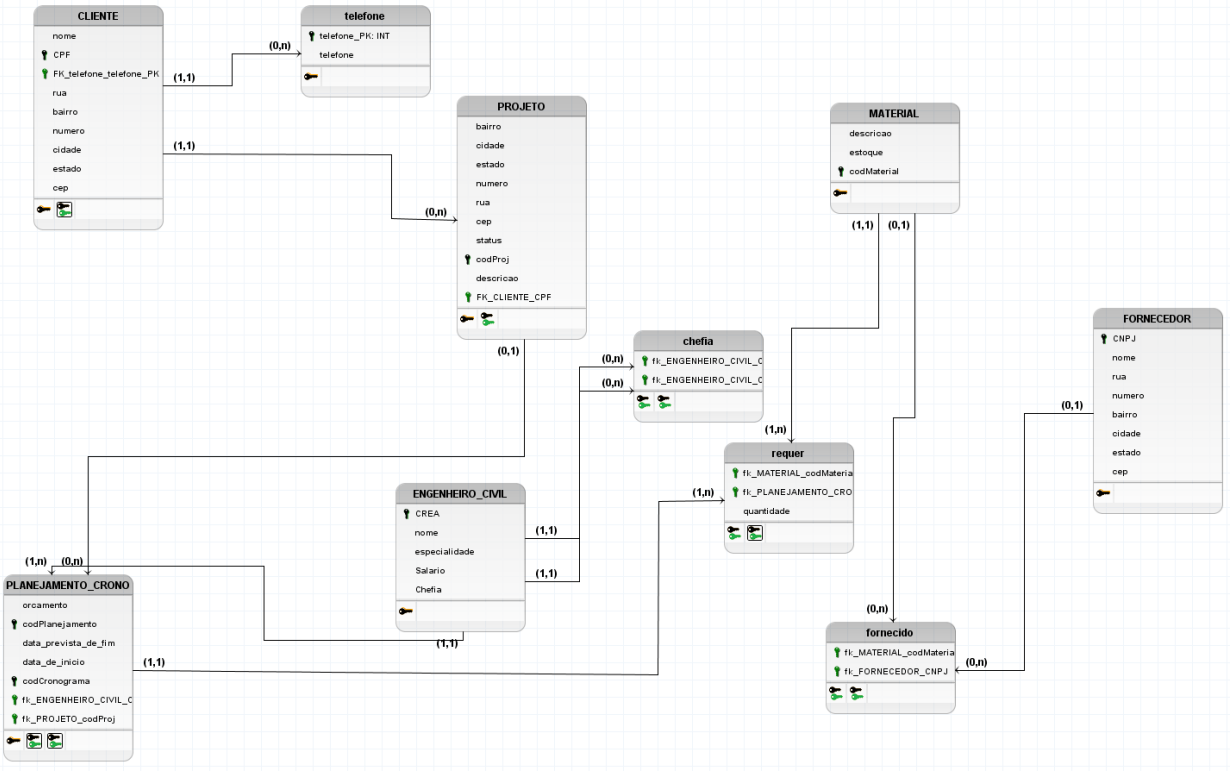
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Curso Superior de Tecnologia em Sistemas para Internet
Autores: Danillo Coelho Barbosa e Jessye Késsia de Carvalho Pereira
Disciplina: Banco de Dados 2
Professor(a): Damires Yluska Souza Fernandes

1. Diagramas Entidade-Relacionamento

1.1 Nível Conceitual



1.2 Nível Lógico



2. Implementação do projeto de BDR no SGBD PostgreSQL

2.1 Criação e uso de objetos básicos

a. Tabelas e constraints (PK, FK, UNIQUE, campos que não podem ter valores nulos,checks de validação) de acordo com as regras de negócio do projeto.

```
-- Cria a tabela telefone que pertence a cliente
CREATE TABLE TELEFONE (
   telefone CHAR(11) NOT NULL,
   CONSTRAINT PK_TELEFONE PRIMARY KEY(telefone)
);
-- Cria a tabela cliente
CREATE TABLE CLIENTE (
   nome VARCHAR(40),
   CPF CHAR(11),
   telefone CHAR(11),
   rua VARCHAR(40),
   bairro VARCHAR(40),
   numero VARCHAR(40),
   cidade VARCHAR(40),
   estado VARCHAR(40),
   cep CHAR(8),
   constraint PK_CLIENTE PRIMARY KEY(CPF),
   constraint FK_telefone_cliente foreign key(telefone) references
TELEFONE(telefone)
);
-- Cria a tabela engenheiro
CREATE TABLE ENGENHEIRO_CIVIL (
   CREA CHAR(10) NOT NULL,
   nome VARCHAR(40),
   especialidade VARCHAR(50),
   salario numeric(10,2),
   chefe CHAR(10),
   constraint PK_Eengenheiro primary key (CREA),
   constraint FK_CHEFE FOREIGN KEY (chefe) REFERENCES
ENGENHEIRO_CIVIL(CREA)
);
-- Cria a tabela projeto
CREATE TABLE PROJETO (
   codProjeto SERIAL,
```

```
   bairro VARCHAR(40),
   cidade VARCHAR(40),
   estado VARCHAR(40),
   numero INTEGER,
   rua VARCHAR(40),
   cep CHAR(8),
   status VARCHAR(40),
   descricao VARCHAR(40),
   CPF_cliente CHAR (11) NOT NULL,
   CREA_eng CHAR(10) NOT NULL,
   constraint PK_PROJETO PRIMARY KEY(codProjeto),
   constraint FK_CLIENTE FOREIGN KEY(CPF_cliente) REFERENCES
CLIENTE(CPF),
   constraint FK_ENGENHEIRO FOREIGN KEY(CREA_eng) REFERENCES
ENGENHEIRO_CIVIL(CREA)
);
-- Cria a tabela planejamento
CREATE TABLE PLANEJAMENTO (
   codPlanejamento SERIAL NOT NULL,
   autor_projeto CHAR(10),
   projeto INTEGER,
   orcamento NUMERIC(18,2),
   codProjeto SERIAL,
   data_inicio DATE,
   data_fim DATE,
   constraint PK_planejamento primary key (codPlanejamento),
   constraint FK_engenheiro_crea foreign key(autor_projeto) references
ENGENHEIRO_CIVIL(CREA),
   constraint FK_projeot_codProj foreign key(projeto) references
PROJETO(codProjeto)
);
-- Cria a tabela fornecedor
CREATE TABLE FORNECEDOR (
   CNPJ CHAR(14) NOT NULL,
   nome VARCHAR(40),
   rua VARCHAR(40),
   numero INTEGER,
   bairro VARCHAR(40),
   cidade VARCHAR(40),
   estado VARCHAR(40),
   cep CHAR(8),
```

```
    constraint PK_CNPJ PRIMARY KEY(CNPJ)
);
-- Cria a tabela material
CREATE TABLE MATERIAL (
    codMaterial SERIAL NOT NULL,
    descricao VARCHAR(40),
    estoque INTEGER DEFAULT 0,
    CNPJ_for CHAR(14) NOT NULL,
    constraint PK_material primary key (codMaterial),
    constraint FK_fornecedor FOREIGN KEY (CNPJ_for) REFERENCES
FORNECEDOR(CNPJ)
);
-- Cria a tabela requer
CREATE TABLE REQUER (
    codRequisicao SERIAL,
    planejamento INTEGER NOT NULL,
    material_solicitado INTEGER NOT NULL,
    quantidade INTEGER NOT NULL,
    constraint PK_requisicao primary key (codRequisicao),
    constraint FK_planejamento FOREIGN KEY (planejamento) REFERENCES
PLANEJAMENTO(codPlanejamento),
    constraint FK_material FOREIGN KEY (material_solicitado) REFERENCES
MATERIAL(codMaterial)
);
```

b. 10 consultas variadas de acordo com requisitos da aplicação, com justificativa semântica e conforme critérios seguintes:

- **1 consulta com uma tabela usando operadores básicos de filtro (e.g., IN,between, is null, etc).**

```
-- Selecione os projetos que estão com o status de concluído
SELECT descricao FROM projeto WHERE status = 'Concluído';
```

- **3 consultas com inner JOIN na cláusula FROM (pode ser self join, caso o domínio indique esse uso).**

```
-- Obter o nome do engenheiro através da  descrição do projeto
SELECT e.nome FROM engenheiro_civil e JOIN projeto p ON e.CREA = p.CREA_eng WHERE
p.descricao = 'Construção de parque urbano';
-- Obter a descrição do projeto através do nome do cliente
SELECT p.descricao FROM projeto p JOIN cliente c ON p.CPF_cliente = c.CPF WHERE
c.nome = 'João Pereira';
-- Obter o fornecedor que distribui o material, pela descrição do material
SELECT f.nome FROM fornecedor f JOIN material m ON m.CNPJ_for = f.CNPJ WHERE
m.descricao = 'Aço CA-50';
```

- **1 consulta com left/right/full outer join na cláusula FROM**

-- Obter todos os clientes para saber quais estão ou não com um projeto relacionado no banco de dados.

SELECT c.nome, p.descricao from cliente c LEFT JOIN projeto p ON c.CPF = p.CPF_cliente;

- **2 consultas usando Group By (e possivelmente o having)**

-- Agrupar os engenheiros que estão em algum projeto por especialidade

SELECT e.especialidade FROM engenheiro_civil e JOIN projeto p ON e.CREA = p.CREA_eng WHERE p.descricao is not null GROUP BY e.especialidade;

-- Agrupa por nome da cidade um contador para saber a quantidade de clientes por cidade.

SELECT c.cidade, COUNT(*) AS clientes_cidade FROM cliente c GROUP BY c.cidade;

- **1 consulta usando alguma operação de conjunto (union, except ou intersect)**

-- Descobrir o nome dos clientes que são do estado da paraíba e todos os projetos.

SELECT nome from cliente
where estado like 'PB'
UNION select descricao from projeto;

- **2 consultas que usem subqueries.**

-- Encontrar engenheiros que são chefes de outros engenheiros usando o exists

SELECT nome FROM engenheiro_civil e1 WHERE EXISTS (SELECT * FROM engenheiro_civil e2 WHERE e2.chefe = e1.CREA);

-- Retorna os engenheiros cujo salário é maior que a média salarial dos engenheiros.

SELECT nome from engenheiro_civil where salario > (select avg(salario) from engenheiro_civil);

2.1 Visões

- **1 visão que permita inserção**

-- Cria uma view que permite inserção de projetos

```
CREATE VIEW vw_projeto_detalhes AS
SELECT
    p.codProjeto,
    p.descricao,
    p.status,
    p.CPF_cliente,
    c.nome AS nome_cliente,
    e.nome AS nome_engenheiro,
    p.CREA_eng
FROM PROJETO p
JOIN CLIENTE c ON p.CPF_cliente = c.CPF
JOIN ENGENHEIRO_CIVIL e ON p.CREA_eng = e.CREA;
```

```
    -- Cria uma regra de inserção para a view
CREATE RULE insert_projeto_detalhes AS
ON INSERT TO vw_projeto_detalhes
DO INSTEAD
(
    -- Insere na tabela PROJETO
  INSERT INTO PROJETO(descricao, status, CPF_cliente, CREA_eng)
  VALUES (NEW.descricao, NEW.status, NEW.CPF_cliente, NEW.CREA_eng)
);
```

- **2 visões robustas (e.g., com vários joins) com justificativa semântica, de acordo com os requisitos da aplicação.**

-- Uma view que retorna a relação Engenheiro, projeto e planejamento, para obter com clareza qual o projeto, o planejamento e qual engenheiro é responsável.

```
CREATE OR REPLACE VIEW Relatorio AS
SELECT
E.nome AS nome_engenheiro,
E.CREA,
P.descricao AS nome_projeto,
Plan.data_inicio AS Inicio_projeto
FROM
ENGENHEIRO_CIVIL E
JOIN
PROJETO P ON E.CREA = P.CREA_eng
JOIN
PLANEJAMENTO Plan ON P.codprojeto = Plan.codplanejamento;
```

-- Uma view que retorna os fornecedores juntamente com os materiais que eles fornecem e a requisição de quanto material é preciso para fazer a obra.

```
CREATE OR REPLACE VIEW ANALISE_MAT AS
SELECT
F.nome AS nome_Fornecedor,
M.descricao AS nome_material,
M.estoque AS quantidade_estoque,
R.quantidade AS quantidade_necessaria
FROM
FORNECEDOR F
JOIN
MATERIAL M ON F.CNPJ = M.CNPJ_for
JOIN
REQUER R ON  M.codMaterial = R.material_solicitado;
```

2.2 Índices

- **3 índices para campos indicados com justificativa dentro do contexto das consultas formuladas na criação e uso de objetos básicos .**

-- Criar um índice para visualização da data de início dos projetos.

```
create index dataInicial ON PLANEJAMENTO(data_inicio);
```

-- Criar um índice para visualização da quantidade disponível no estoque para os materiais.

```
create index quantidadeDisponivel ON Material(estoque);
```

-- Criar um índice para visualização da especialidade do engenheiro.

```
create index especialidadeEngenheiro ON engenheiro_civil
(especialidade);
```

2.3 Reescrita de consultas

- **Identificar 2 exemplos de consultas dentro do contexto da aplicação (questão 2.a) que possam e devam ser melhoradas. Reescrevê-las. Justificar a reescrita.**

-- Usar unions no left joins Obter todos os clientes para saber quais estão ou não com um projeto relacionado no banco de dados.

```
SELECT e1.nome
FROM engenheiro_civil e1
JOIN engenheiro_civil e2 ON e2.chefe = e1.CREA;
SELECT c.nome, p.descricao
FROM cliente c
INNER JOIN projeto p ON c.CPF = p.CPF_cliente
UNION
SELECT c.nome, NULL AS descricao
FROM cliente c
LEFT JOIN projeto p ON c.CPF = p.CPF_cliente
WHERE p.CPF_cliente IS NULL;
```

Justificativa:

Para consulta mais complexas, onde pode ser preciso de registros com e sem correspondências, usar o union pode fornecer uma maneira mais facil de perceber quais partes das consultas estão sendo combinadas, sendo uma opção viável onde um left join pode dificultar pela multiplicidade de condições e junções.

-- Substituir a subquery de Encontrar engenheiros que são chefes de outros engenheiros usando o exists por JOIN

```
SELECT e1.nome
FROM engenheiro_civil e1
JOIN engenheiro_civil e2 ON e2.chefe = e1.CREA;
```

Justificativa:

O Join É Mais Interessante Que A Subquery Com Exists, Quando Você Tem Que Lidar Com Uma Quantidade Massiva De Dados Presentes Entre As Tabelas. Por processar os dados mais rápido.

2.4. Funções e procedures armazenadas

- **1 função que use SUM, MAX, MIN, AVG ou COUNT**

-- Cria a função para somar os salários dos engenheiros e exibir a soma total

```
CREATE OR REPLACE FUNCTION soma_salarios_engenheiros()
RETURNS VOID AS $$
DECLARE
   soma_salarial NUMERIC(10,2);
BEGIN
   -- Faz a soma dos salários dos engenheiros
   SELECT SUM(salario) INTO soma_salarial FROM ENGENHEIRO_CIVIL;
   -- Exibe a soma total dos salários com um RAISE NOTICE
   RAISE NOTICE 'A soma dos salários dos engenheiros é: %', soma_salarial;
END;
$$ LANGUAGE plpgsql;
```

- **2 funções e 1 procedure com justificativa semântica, conforme os requisitos da aplicação**

-- Uma função que retorna nome dos engenheiros com base nas especializações solicitadas para um determinado projeto.

```
CREATE OR REPLACE FUNCTION
mostra_engenheiros_por_especializacao(v_especializacao IN
ENGENHEIRO_CIVIL.especialidade%TYPE)
        RETURNS VOID
        AS $$
        DECLARE
                v_engenheiros_cursor CURSOR
                FOR SELECT nome
                FROM ENGENHEIRO_CIVIL
                WHERE especialidade = v_especializacao;


                v_engenheiro RECORD;
        BEGIN
                FOR v_engenheiro IN v_engenheiros_cursor LOOP
                        RAISE NOTICE 'Nome do engenheiro: %',
v_engenheiro.nome;
                END LOOP;
        END;
$$ LANGUAGE plpgsql;
SELECT mostra_engenheiros_por_especializacao('Estruturas');
```

-- Uma função para verificar se há estoque suficiente do material solicitado, caso não seja possível, retorne exceções'.

```
CREATE OR REPLACE FUNCTION solicita_estoque_material(descricao_material
VARCHAR, quantidade_solicitada INTEGER)
RETURNS VARCHAR
AS $$
DECLARE
   estoque_atual INTEGER;
BEGIN
   SELECT estoque
        INTO estoque_atual
   FROM material
   WHERE descricao = descricao_material;


        IF estoque_atual >= quantidade_solicitada THEN
                RAISE NOTICE 'Estoque suficiente para o material solicitado. Estoque
atual: %, quantidade solicitada: %.', estoque_atual, quantidade_solicitada;
        ELSIF estoque_atual < quantidade_solicitada THEN
                RAISE EXCEPTION 'Estoque insuficiente para o material solicitado.';
        END IF;
EXCEPTION
        WHEN raise_exception THEN
                return 'Estoque insuficiente para o material solicitado.';
        WHEN no_data_found THEN
                RETURN 'Material com descrição ''' || descricao_material || ''' não
encontrado.';
        WHEN OTHERS THEN
                RETURN 'Erro desconhecido';
END;
$$ LANGUAGE plpgsql;
```

-- Uma procedure que retorna o nome dos clientes e quantidade de projetos que eles possuem registrados.

```
 CREATE OR REPLACE PROCEDURE mostra_quantidade_projetos(nome_cliente
 VARCHAR)
 AS $$
 DECLARE
    contador INTEGER;
 BEGIN
    SELECT COUNT(*)
```

```
    INTO contador
    FROM CLIENTE c
    JOIN PROJETO p ON c.CPF = p.CPF_cliente
    WHERE c.nome = nome_cliente;
    RAISE NOTICE '% tem % projetos.', nome_cliente, COALESCE(contador, 0);
END;
$$ LANGUAGE plpgsql;
CALL mostra_quantidade_projetos('Carlos Silva');
```

2.5 Triggers

- **3 diferentes triggers com justificativa semântica, de acordo com os requisitos da aplicação.**

-- Acione o Trigger para mudar a data de planejamento, caso tenha atraso na entrega do projeto para uma função armazenada de planejamento atrasado, após atualizar.

```
CREATE TABLE planejamento_atrasado (
    codPlanejamento INTEGER NOT NULL,
    nova_data_fim DATE,
    motivo VARCHAR(60),
    CONSTRAINT PK_PLANEJAMENTO_ATRASADO PRIMARY
KEY(codPlanejamento)
);
CREATE OR REPLACE FUNCTION funcao_atraso_planejamento()
RETURNS TRIGGER AS $$
BEGIN
        -- Inserir na tabela de planejamento atrasado com a nova_data_fim como NULL
        INSERT INTO PLANEJAMENTO_ATRASADO (codPlanejamento,
nova_data_fim, motivo)
        VALUES (NEW.codPlanejamento, NEW.data_fim, null);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_atraso_planejamento
AFTER UPDATE ON PLANEJAMENTO
FOR EACH ROW
EXECUTE FUNCTION funcao_atraso_planejamento();
UPDATE planejamento
SET data_fim = '2024-12-31'
WHERE codPlanejamento = 1;
```

--Acionar o Trigger para salvar os projetos que foram concluídos numa tabela de projetos concluídos backup.

```
CREATE TABLE
PROJETO_CONCLUIDO_BACKUP (
        codProjeto INTEGER PRIMARY KEY,
        status varchar(40),
        data_concluido date
);
CREATE OR REPLACE FUNCTION
fn_backup_projeto_concluido()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.status = 'Concluído' THEN
      INSERT INTO
PROJETO_CONCLUIDO_BACKUP (codProjeto,
status,         data_concluido)
      VALUES (NEW.codProjeto, NEW.status,
current_date);
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_backup_projeto_concluido
AFTER UPDATE ON PROJETO
FOR EACH ROW
EXECUTE FUNCTION
fn_backup_projeto_concluido();
```

-- Adicionar o Trigger para proteger a exclusão de um engenheiro que está vinculado a um projeto em fase de execução.

```
CREATE OR REPLACE FUNCTION fn_proteger_exclusao_engenheiro()
RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (
      SELECT 1
      FROM PROJETO p
      JOIN ENGENHEIRO_CIVIL e
            ON p.CREA_eng = e.CREA
      WHERE e.CREA = OLD.CREA
      AND p.status IN ('Em Andamento', 'Em Planejamento')
   ) THEN
```

```
        RAISE EXCEPTION 'Não é possível excluir o engenheiro % pois ele está
vinculado a um projeto em fase de execução.', OLD.CREA;
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_proteger_exclusao_engenheiro
BEFORE DELETE ON ENGENHEIRO_CIVIL
FOR EACH ROW
EXECUTE FUNCTION fn_proteger_exclusao_engenheiro();
```