**INSTITUTO FEDERAL DA PARAÍBA - IFPB** 

**Unidade Acadêmica de Informação e Comunicação**

**CST em Sistemas para Internet**

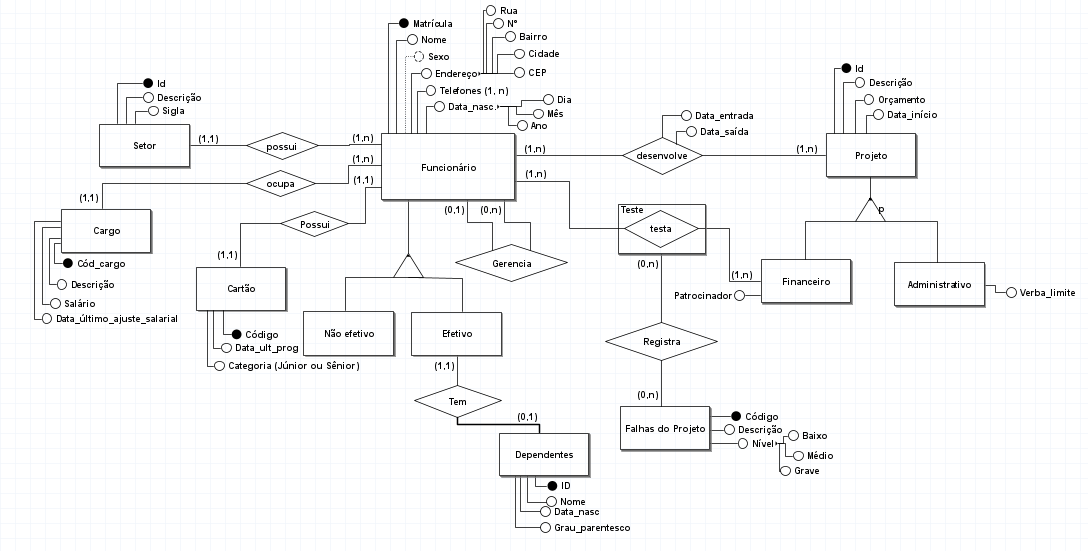
**Disciplina: Banco de Dados II – 2023.2**

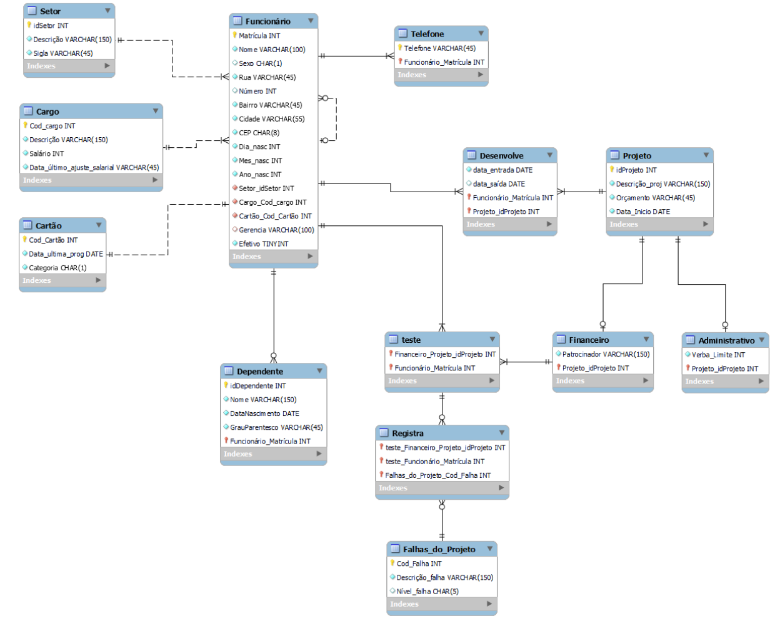
**Professores: Damires e Thiago**

**Discentes: Sammuel Luna e Maria Eduarda Vitorino**

**Roteiro para Projeto de Banco de Dados Relacional**

**1. Diagramas Entidade-Relacionamento em nível conceitual e em nível lógico atualizados (10,0)**

****

****

**2. Implementação do projeto de BDR no SGBD PostgreSQL**

--TSI - IFPB - PROJETO DE BANCO DE DADOS RELACIONAIS (POSTGRESQL)

--Discentes: Sammuel Luna e Maria Eduarda Vitorino

--Docentes: Damires Yluska de Souza Fernandes e Thiago Jose Marques Moura

--CRIAÇÃO DAS TABELAS

CREATE TABLE setor (

id\_setor INT NOT NULL PRIMARY KEY,

sigla VARCHAR(45) NOT NULL

);

CREATE TABLE cargo (

cod\_cargo INT NOT NULL PRIMARY KEY,

descricao VARCHAR(150) NOT NULL,

salario INT NOT NULL,

data\_ultimo\_ajuste\_salarial DATE NOT NULL

);

CREATE TABLE cartao (

cod\_cartao INT NOT NULL PRIMARY KEY,

data\_ult\_prog DATE NOT NULL,

categoria CHAR(1) NOT NULL CHECK (categoria IN ('J', 'S'))

);

CREATE TABLE funcionario (

matricula SERIAL NOT NULL PRIMARY KEY,

nome VARCHAR(100) NOT NULL,

sexo CHAR(1) CHECK (sexo IN ('F', 'M')),

rua VARCHAR(45) NOT NULL,

numero INT,

bairro VARCHAR(45) NOT NULL,

cidade VARCHAR(55) NOT NULL,

cep CHAR(8) NOT NULL,

data\_nasc DATE NOT NULL CHECK (EXTRACT(YEAR FROM data\_nasc) < 2004),

setor\_id INT NOT NULL REFERENCES setor(id\_setor),

cargo\_cod\_cargo INT NOT NULL REFERENCES cargo(cod\_cargo),

cartao\_cod\_cartao INT NOT NULL REFERENCES cartao(cod\_cartao),

gerencia INT REFERENCES funcionario(matricula),

efetivo BOOLEAN NOT NULL

);

CREATE TABLE dependente (

id\_dependente INT NOT NULL,

nome VARCHAR(150) NOT NULL,

data\_nascimento DATE NOT NULL CHECK (EXTRACT(YEAR FROM data\_nascimento) < 2023),

grau\_parentesco VARCHAR(45) NOT NULL,

funcionario\_matricula INT UNIQUE NOT NULL REFERENCES funcionario(matricula),

PRIMARY KEY (id\_dependente, funcionario\_matricula)

);

CREATE TABLE telefone (

telefone VARCHAR(45) NOT NULL,

funcionario\_matricula INT NOT NULL REFERENCES funcionario(matricula),

PRIMARY KEY (telefone, funcionario\_matricula)

);

CREATE TABLE projeto (

id\_projeto INT NOT NULL PRIMARY KEY,

descricao\_projeto VARCHAR(150) NOT NULL,

orcamento VARCHAR(45) NOT NULL,

data\_inicio DATE NOT NULL

);

CREATE TABLE desenvolve (

data\_entrada DATE NOT NULL,

data\_saida DATE,

funcionario\_matricula INT NOT NULL REFERENCES funcionario(matricula) ON DELETE CASCADE,

projeto\_id\_projeto INT NOT NULL REFERENCES projeto(id\_projeto),

PRIMARY KEY (funcionario\_matricula, projeto\_id\_projeto)

);

CREATE TABLE teste (

financeiro\_projeto\_id\_projeto INT NOT NULL,

funcionario\_matricula INT NOT NULL,

PRIMARY KEY (financeiro\_projeto\_id\_projeto, funcionario\_matricula),

FOREIGN KEY (financeiro\_projeto\_id\_projeto) REFERENCES projeto(id\_projeto),

FOREIGN KEY (funcionario\_matricula) REFERENCES funcionario(matricula) ON DELETE CASCADE

);

CREATE TABLE falhas\_do\_projeto (

cod\_falha INT PRIMARY KEY,

descricao\_falha VARCHAR(150),

nivel\_falha CHAR(5) CHECK (nivel\_falha IN ('baixo', 'medio', 'grave'))

);

CREATE TABLE registra (

teste\_financeiro\_projeto\_id\_projeto INT NOT NULL,

teste\_funcionario\_matricula INT NOT NULL,

falhas\_do\_projeto\_cod\_falha INT NOT NULL,

PRIMARY KEY (teste\_financeiro\_projeto\_id\_projeto, teste\_funcionario\_matricula, falhas\_do\_projeto\_cod\_falha),

FOREIGN KEY (teste\_financeiro\_projeto\_id\_projeto, teste\_funcionario\_matricula) REFERENCES teste(financeiro\_projeto\_id\_projeto, funcionario\_matricula) ON DELETE CASCADE,

FOREIGN KEY (falhas\_do\_projeto\_cod\_falha) REFERENCES falhas\_do\_projeto(cod\_falha)

);

CREATE TABLE administrativo (

verba\_limite INT NOT NULL,

projeto\_id\_projeto INT NOT NULL PRIMARY KEY,

FOREIGN KEY (projeto\_id\_projeto) REFERENCES projeto(id\_projeto)

);

CREATE TABLE financeiro (

patrocinador VARCHAR(150) NOT NULL,

projeto\_id\_projeto INT NOT NULL PRIMARY KEY,

FOREIGN KEY (projeto\_id\_projeto) REFERENCES projeto(id\_projeto)

);

---------------------QUESTÃO 2AI----------------------

INSERT INTO setor (id\_setor, sigla) VALUES

(1, 'Desenvolvimento'),

(2, 'Qualidade'),

(3, 'Gerência'),

(4, 'Infraestrutura'),

(5, 'Marketing');

INSERT INTO cargo (cod\_cargo, descricao, salario, data\_ultimo\_ajuste\_salarial) VALUES

(1, 'Desenvolvedor FullStack', 6000, '2022-01-01'),

(2, 'Analista de Qualidade', 5500, '2022-03-15'),

(3, 'Product Owner', 8000, '2022-02-20'),

(4, 'Analista de Infraestrutura', 6500, '2022-04-10'),

(5, 'Especialista de Marketing', 7000, '2022-05-05');

INSERT INTO cartao (cod\_cartao, data\_ult\_prog, categoria) VALUES

(101, '2022-01-01', 'J'),

(102, '2022-02-15', 'S'),

(103, '2022-03-20', 'J'),

(104, '2022-04-10', 'S'),

(105, '2022-05-05', 'J'),

(109, '2022-04-10', 'J'),

(110, '2022-05-05', 'J');

INSERT INTO funcionario (matricula, nome, sexo, rua, numero, bairro, cidade, cep, data\_nasc, setor\_id, cargo\_cod\_cargo, cartao\_cod\_cartao, gerencia, efetivo) VALUES

(1, 'Maria Santos', 'F', 'Rua A', 100, 'Centro', 'Cidade A', '12345678', '1990-03-12', 1, 1, 101, NULL, true),

(2, 'João Silva', 'M', 'Rua B', 200, 'Bairro B', 'Cidade B', '23456789', '1985-05-20', 1, 2, 102, NULL, true),

(3, 'Laura Oliveira', 'F', 'Rua C', 300, 'Bairro C', 'Cidade C', '34567890', '1995-07-10', 3, 3, 103, 1, true),

(4, 'Pedro Pereira', 'M', 'Rua D', 400, 'Bairro D', 'Cidade D', '45678901', '1988-09-18', 2, 4, 104, 2, true),

(5, 'Gabriel Ferreira', 'M', 'Rua E', 500, 'Bairro E', 'Cidade E', '56789012', '1993-11-25', 5, 5, 105, 3, true),

(9, 'Gustavo Ferreira', 'M', 'Rua I', 900, 'Bairro I', 'Cidade I', '90123456', '1990-04-05', 4, 4, 109, NULL, false),

(10, 'Camila Silva', 'F', 'Rua J', 1000, 'Bairro J', 'Cidade J', '01234567', '1995-12-20', 5, 5, 110, NULL, false);

--update para mudar datas de funcionarios não-efetivos (possíveis estagiários?)

UPDATE funcionario

SET data\_nasc = '2001-05-20'

WHERE matricula = 9;

UPDATE funcionario

SET data\_nasc = '2003-11-25'

WHERE matricula = 10;

INSERT INTO dependente (id\_dependente, nome, data\_nascimento, grau\_parentesco, funcionario\_matricula) VALUES

(1, 'Lucas Santos', '2010-02-28', 'Filho', 1),

(2, 'Sophia Silva', '2008-06-15', 'Filha', 2),

(3, 'Davi Oliveira', '2015-08-20', 'Filho', 3),

(4, 'Lara Pereira', '2012-10-10', 'Filha', 4),

(5, 'Giovanna Ferreira', '2017-12-05', 'Filha', 5);

INSERT INTO telefone (telefone, funcionario\_matricula) VALUES

('11111111', 1),

('22222222', 2),

('33333333', 3),

('44444444', 4),

('55555555', 5),

('99999999', 9),

('10101010', 10);

INSERT INTO projeto (id\_projeto, descricao\_projeto, orcamento, data\_inicio) VALUES

(1, 'Desenvolvimento Sistema de Gestão', '100000', '2022-01-15'),

(2, 'Implantação de Processos de Qualidade', '150000', '2022-03-20'),

(3, 'Gestão de Equipes', '80000', '2022-02-10'),

(4, 'Atualização de Infraestrutura', '120000', '2022-04-05'),

(5, 'Campanha Publicitária Online', '90000', '2022-05-01'),

(6, 'Projeto de Inovação Tecnológica', '200000', '2023-07-10'),

(7, 'Melhoria de Processos Internos', '180000', '2023-08-15'),

(8, 'Expansão de Mercado Internacional', '250000', '2023-09-20');

INSERT INTO teste (financeiro\_projeto\_id\_projeto, funcionario\_matricula) VALUES

(1, 1),

(2, 2),

(3, 3),

(4, 5),

(1, 4),

(5, 3);

INSERT INTO falhas\_do\_projeto (cod\_falha, descricao\_falha, nivel\_falha) VALUES

(1, 'Bug crítico no sistema', 'grave'),

(2, 'Problemas de documentação', 'medio'),

(3, 'Falha na comunicação', 'baixo'),

(4, 'Dificuldade de implementação', 'grave'),

(5, 'Desvio de prazos', 'medio');

INSERT INTO registra (teste\_financeiro\_projeto\_id\_projeto, teste\_funcionario\_matricula, falhas\_do\_projeto\_cod\_falha)

VALUES

(1, 1, 1),

(2, 2, 2),

(3, 3, 3),

(4, 5, 4),

(1, 4, 5),

(5, 3, 1);

INSERT INTO desenvolve (data\_entrada, data\_saida, funcionario\_matricula, projeto\_id\_projeto) VALUES

('2022-01-15', '2022-05-30', 1, 1),

('2022-03-20', '2022-08-15', 2, 2),

('2022-02-10', NULL, 3, 3),

('2022-04-05', NULL, 4, 4),

('2022-05-01', NULL, 5, 5),

('2022-01-15', '2022-05-30', 2, 1),

('2023-07-11', NULL, 9, 6),

('2023-08-23', NULL, 10, 7),

('2023-09-28', NULL, 5, 8);

INSERT INTO administrativo (verba\_limite, projeto\_id\_projeto) VALUES

(50000, 6),

(75000, 7),

(40000, 8);

INSERT INTO financeiro (patrocinador, projeto\_id\_projeto) VALUES

('ByteMaster Inc.', 1),

('TechTudo', 2),

('Codezilla', 3),

('CyberRocket', 4),

('NerdNest', 5);

---------------------QUESTÃO 2AII----------------------

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

SELECT \* FROM funcionario

WHERE data\_nasc BETWEEN '1985-01-01' AND '1990-12-31';

--seleciona os funcionários que nasceram entre 1985 e 1990

--3 consultas com inner JOIN na cláusula FROM (pode ser self join, caso o

--domínio indique esse uso).

SELECT f.nome AS nome\_funcionario, s.sigla AS sigla\_setor

FROM funcionario f INNER JOIN setor s on f.setor\_id = s.id\_setor;

-- retorna todos os funcionários e o nome dos respectivos setores.

SELECT p.descricao\_projeto, f.nome AS nome\_funcionario

FROM projeto p

INNER JOIN desenvolve d ON p.id\_projeto = d.projeto\_id\_projeto

INNER JOIN funcionario f ON d.funcionario\_matricula = f.matricula;

--retorna todos os projetos e os funcionários que estão trabalhando neles.

SELECT f.nome AS nome\_funcionario, p.descricao\_projeto, fd.descricao\_falha

FROM funcionario f

INNER JOIN registra r ON f.matricula = r.teste\_funcionario\_matricula

INNER JOIN projeto p ON r.teste\_financeiro\_projeto\_id\_projeto = p.id\_projeto

INNER JOIN falhas\_do\_projeto fd ON r.falhas\_do\_projeto\_cod\_falha = fd.cod\_falha;

--retorna o funcionario que registrou a falha, juntamente coom o projeto associado e a descrição da falha

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

SELECT f.nome AS nome\_funcionario, d.nome AS nome\_dependente

FROM funcionario f

LEFT JOIN dependente d ON f.matricula = d.funcionario\_matricula;

--retorna os funcionários que possuem dependentes, considerando todos os registros da tabela à esquerda, ou seja, mostra também os funcionários que não possuem dependentes.

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

SELECT c.descricao AS cargo, AVG(c.salario) AS media\_salario FROM cargo c

GROUP BY c.descricao

HAVING AVG(c.salario) > 6500;

--retorna a média de salário por cargo, agrupando pelo cargo e considerando apenas aqueles maiores de 6500 reais.

SELECT setor\_id, COUNT(\*) AS total\_funcionarios

FROM funcionario

GROUP BY setor\_id;

--mostra a quantidade de funcionários por setor.

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

SELECT c.cod\_cartao, c.categoria, f.nome AS nome\_funcionario

FROM cartao c

JOIN funcionario f ON c.cod\_cartao = f.cartao\_cod\_cartao

EXCEPT

SELECT c.cod\_cartao, c.categoria, f.nome AS nome\_funcionario

FROM cartao c

JOIN funcionario f ON c.cod\_cartao = f.cartao\_cod\_cartao

WHERE c.categoria = 'S';

--retorna funcionários que possuem cartões,m exceto os que sãio da categoria senior.

--2 consultas que usem subqueries.

SELECT COUNT(\*) AS num\_funcionarios\_com\_dependentes

FROM (

SELECT DISTINCT funcionario\_matricula

FROM dependente

) AS funcionarios\_com\_dependentes;

--retorna o número de funcionários com dependentes.

SELECT f.nome, f.cargo\_cod\_cargo, c.descricao

FROM funcionario f

JOIN cargo c ON f.cargo\_cod\_cargo = c.cod\_cargo

WHERE f.cargo\_cod\_cargo IN (

SELECT cod\_cargo

FROM cargo

WHERE salario > 6000

);

--retorna o nome do funcionário, o codigo do seu respectivo cargo e a descrição desses cargos cujo salário é maior que 6000

---------------------QUESTÃO 2B----------------------

--1 visão que permita inserção

CREATE OR REPLACE VIEW view\_funcionario\_efetivo AS

SELECT \* FROM funcionario

WHERE efetivo = true;

-- seleciona todas as informações de todos os funcionários nos quais os mesmos sejam efetivos.

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

--requisitos da aplicação.

CREATE OR REPLACE VIEW view\_funcionario\_detalhes AS

SELECT

f.matricula AS funcionario\_matricula,

f.nome AS funcionario\_nome,

s.sigla AS setor\_sigla,

c.descricao AS cargo,

c.salario AS salario,

car.categoria AS cartao\_categoria

FROM funcionario f

JOIN setor s ON f.setor\_id = s.id\_setor

JOIN cargo c ON f.cargo\_cod\_cargo = c.cod\_cargo

JOIN cartao car ON f.cartao\_cod\_cartao = car.cod\_cartao;

-- seleciona as colunas com detalhes relevantes dos funcionários, tais como seu setor, cargo, salario e categoria de seu respectivo cartão, por meio de joins em tabelas ligadas por fks

CREATE OR REPLACE VIEW view\_funcionario\_projeto AS

SELECT

f.matricula AS funcionario\_matricula,

f.nome AS funcionario\_nome,

p.id\_projeto, p.descricao\_projeto

FROM teste t

JOIN funcionario f ON t.funcionario\_matricula = f.matricula

JOIN projeto p ON t.financeiro\_projeto\_id\_projeto = p.id\_projeto;

-- lista os funcionários e seus respectivos projetos por meio de um join utilizando das chaves presentes na tabela teste

--------------------QUESTÃO 2C----------------------

CREATE INDEX idx\_data\_nasc ON funcionario(data\_nasc); -- para consultas que filtram por idade ou por data de nascimento. Campos consultados em forma de intervalo são um grande potencial para indice

CREATE INDEX idx\_falha ON falhas\_do\_projeto(descricao\_falha); --útil para consultas e filtragens em falhas do projeto

CREATE INDEX idx\_funcionario\_projeto ON desenvolve(funcionario\_matricula, projeto\_id\_projeto); --útil para verificar a presença de funcinários nos projetos, indicado para consultas com joins

---------------------QUESTÃO 2D----------------------

--Identificar 2 exemplos de consultas dentro do contexto da aplicação (questão 2.a) que

--possam e devam ser melhoradas. Reescrevê-las. Justificar a reescrita.

--Exemplo 1: Consulta utilizando operação de conjunto (except). Versão melhorada:

SELECT f.nome AS nome\_funcionario

FROM funcionario f

JOIN cartao c ON f.cartao\_cod\_cartao = c.cod\_cartao

WHERE c.categoria = 'J';

-- Com apenas um JOIN simples, e apenas 4 linhas de código, é possível filtrar pela categoria desejada. Uma consulta mais performática e ágil.

-- Exemplo 2 e 3: Consultas com subqueries. Versões melhoradas:

SELECT COUNT(DISTINCT funcionario\_matricula) AS num\_funcionarios\_com\_dependentes

FROM dependente;

--Menos linas de código, maior entendimento e clareza acerca do objetivo da consulta

SELECT f.nome, f.cargo\_cod\_cargo, c.descricao

FROM funcionario f

JOIN cargo c ON f.cargo\_cod\_cargo = c.cod\_cargo

WHERE c.salario > 6000;

-- Novamente, com apenas um JOIN simples é possível verificar os salarios maiores que 6000 por nome, codigo e cargo. Não é necessário o uso de JOIN + subquery.

---------------------QUESTÃO 2E----------------------

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

CREATE OR REPLACE FUNCTION QuantFuncionariosEmProjeto(project\_id INT)

RETURNS INT AS $$

DECLARE

count INT;

BEGIN

SELECT COUNT(\*)

INTO count

FROM desenvolve

WHERE projeto\_id\_projeto = project\_id

AND data\_saida IS NULL; -- Verifica se a data de saída é nula, o que indica que o funcionário ainda está no projeto

RETURN count;

END;

$$ LANGUAGE plpgsql;

-- Essa função retorna a quantidade de funcionários ativos naquele projeto, ou seja, funcionários que não possuem a data de saída.

SELECT QuantFuncionariosEmProjeto(2); -- retorna 1,apesar de haver 2 funcionários cadastrados, pois a função leva em conta apenas os funcionários que não encerraram as atividades naquele projeto.

--insert para validar a participação de mais de um funcionário em um projeto

insert into desenvolve (data\_entrada, funcionario\_matricula, projeto\_id\_projeto) VALUES

('2022-02-20', 1, 2);

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

CREATE OR REPLACE FUNCTION BuscarDependentes(func\_id INT)

RETURNS varchar

AS $$

DECLARE

dep\_name varchar(150);

BEGIN

SELECT nome INTO dep\_name

FROM dependente

WHERE funcionario\_matricula = func\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Nenhum dependente encontrado. Verifique o código do funcionário';

END IF;

RETURN dep\_name;

EXCEPTION

WHEN raise\_exception THEN

RETURN SQLERRM;

WHEN OTHERS THEN

RETURN 'Erro Desconhecido';

END;

$$ LANGUAGE plpgsql;

SELECT BuscarDependentes(3);

SELECT BuscarDependentes(9);

--Função que busca o dependente daquele funcionário, Baseando-se no id do funcionário. Essa função trata exceção para caso haja um funcionário sem dependentes, usando o RAISE EXCEPTION e o bloco EXCEPTION para capturar e tratar.

CREATE OR REPLACE FUNCTION BuscarFalhasCriticasPorProjeto(projeto\_id INT)

RETURNS TABLE (codigoFalha INT, descricaoFalha VARCHAR(150), nivelFalha CHAR(5))

AS $$

BEGIN

RETURN QUERY

SELECT f.cod\_falha, f.descricao\_falha, f.nivel\_falha

FROM falhas\_do\_projeto f

JOIN registra r ON cod\_falha = falhas\_do\_projeto\_cod\_falha

WHERE teste\_financeiro\_projeto\_id\_projeto = projeto\_id AND nivel\_falha = 'grave';

END;

$$ LANGUAGE plpgsql;

SELECT BuscarFalhasCriticasPorProjeto(1);

--Essa função retorna todas as falhas críticas do projeto desejado.

/\*

->> OPÇÃO PARA BUSCAR TODAS AS FALHAS CRÍTICAS, INDEPENDENTEMENTE DE PROJETO <<--

CREATE OR REPLACE FUNCTION BuscarFalhasCriticas()

RETURNS TABLE (projeto\_id INT, cod\_falha INT, descricao\_falha VARCHAR(150), nivel\_falha CHAR(5))

AS $$

BEGIN

RETURN QUERY

SELECT r.teste\_financeiro\_projeto\_id\_projeto, f.cod\_falha, f.descricao\_falha, f.nivel\_falha

FROM falhas\_do\_projeto f

JOIN registra r ON f.cod\_falha = r.falhas\_do\_projeto\_cod\_falha

WHERE f.nivel\_falha = 'grave';

END;

$$ LANGUAGE plpgsql;

\*/

CREATE OR REPLACE PROCEDURE AtualizarOrcamentoProjeto(projeto\_id INT, novo\_orcamento VARCHAR)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE projeto

SET orcamento = novo\_orcamento

WHERE id\_projeto = projeto\_id;

RAISE NOTICE 'O orçamento do projeto % foi atualizado para %', projeto\_id, novo\_orcamento;

EXCEPTION

WHEN others THEN

RAISE EXCEPTION 'Ocorreu um erro ao atualizar o orçamento do projeto %', projeto\_id;

END;

$$;

--A procedure acima atualiza o orçamento do projeto, uma vez que ele pode ser variável no decorrer do tempo.

SELECT \* FROM projeto;

CALL AtualizarOrcamentoProjeto(2, '50000');

SELECT \* FROM projeto;

---------------------QUESTÃO 2F----------------------

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

--aplicação.

CREATE OR REPLACE FUNCTION verifica\_criar\_cartao()

RETURNS TRIGGER AS $$

BEGIN

IF NOT EXISTS (SELECT 1 FROM cartao WHERE cod\_cartao = NEW.cartao\_cod\_cartao) THEN

INSERT INTO cartao (cod\_cartao, data\_ult\_prog, categoria)

VALUES (NEW.cartao\_cod\_cartao, CURRENT\_TIMESTAMP, 'J');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_criar\_cartao

BEFORE INSERT ON funcionario

FOR EACH ROW EXECUTE FUNCTION verifica\_criar\_cartao();

-- verifica, no momento precedente à inserção de uma linha na tabela de funcionários, a existencia do cartão passado caso não exista, um cartão de categoria Júnior será automaticamente gerado

CREATE OR REPLACE FUNCTION deletar\_dependentes()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM dependente

WHERE funcionario\_matricula = OLD.matricula;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_deletar\_dependentes

BEFORE DELETE ON funcionario

FOR EACH ROW

EXECUTE FUNCTION deletar\_dependentes();

-- após a remoção de um funcionário, seus respectivos dependentes serão buscados e propriamente removidos da tabela dependente

CREATE OR REPLACE FUNCTION deletar\_telefones()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM telefone

WHERE funcionario\_matricula = OLD.matricula;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_deletar\_telefones

BEFORE DELETE ON funcionario

FOR EACH ROW

EXECUTE FUNCTION deletar\_telefones();

-- após a remoção de um funcionário, seus respectivos números de telefone serão buscados e propriamente removidos da tabela telefone

-------------------------