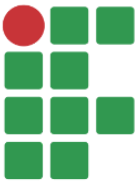


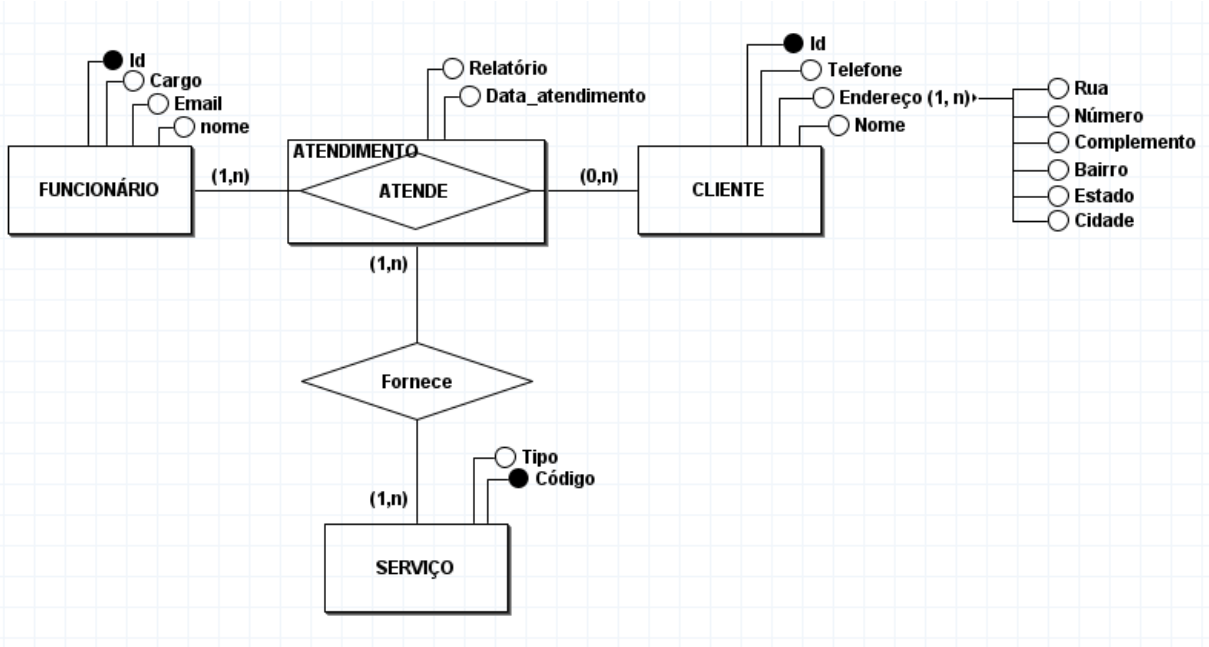
INSTITUTO FEDERAL
Paraíba | Campus João Pessoa

Projeto de Banco de Dados Relacional

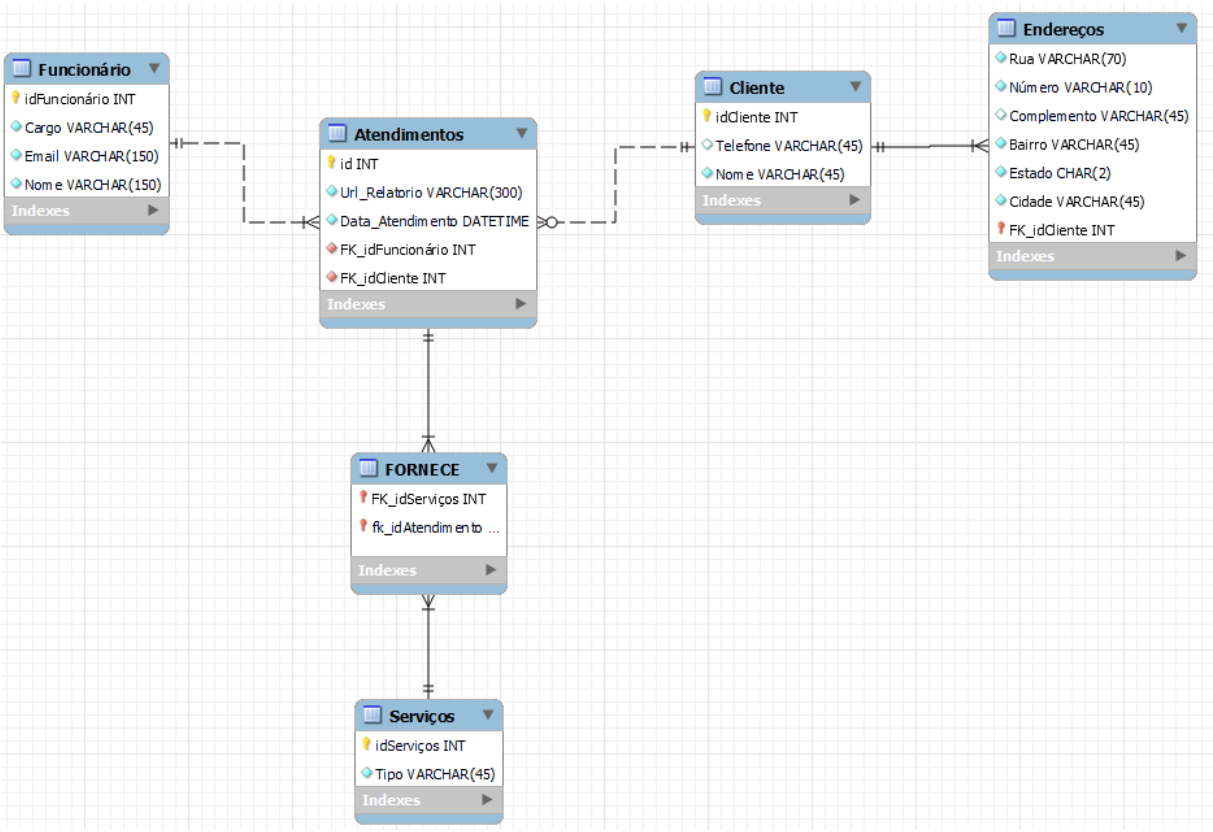
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Curso Superior de Tecnologia em Sistemas para Internet
Autores: Filipe da Silva Rodrigues e Gabryel Araujo de Sousa Dias
Disciplina: Banco de Dados 2
Professor(a): Damires Yluska Souza Fernandes

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.

```
-- CRIAÇÃO DA TABELA FUNCIONARIO
CREATE TABLE FUNCIONARIO (
	id_funcionario SERIAL NOT NULL,
	nome varchar(150) NOT NULL,
	cargo varchar(45) NOT NULL,
	email varchar(150) NOT NULL UNIQUE,
	CONSTRAINT PK_IDFUNC PRIMARY KEY (id_funcionario),
	CONSTRAINT CK_CARGO CHECK (cargo IN  ('Programador', 'Técnico', 'Gestão'))
);

-- CRIAÇÃO DA TABELA CLIENTE
CREATE TABLE CLIENTE (
	id_cliente SERIAL NOT NULL,
	telefone VARCHAR(45),
	nome VARCHAR(150) NOT NULL,
	CONSTRAINT PK_CLIENTE PRIMARY KEY(id_cliente)
);

-- CRIAÇÃO DA TABELA DE ENDEREÇOS
CREATE TABLE ENDERECO (
	id_cliente INTEGER NOT NULL,
	rua VARCHAR(70) NOT NULL,
	numero VARCHAR(10) NOT NULL DEFAULT 'S/N',
	complemento VARCHAR(45),
	bairro VARCHAR(45) NOT NULL,
	estado CHAR(2) NOT NULL DEFAULT 'PB',
	cidade VARCHAR(45) NOT NULL DEFAULT 'João Pessoa',
	CONSTRAINT PK_ENDERECO PRIMARY KEY (id_cliente),
	CONSTRAINT FK_CLIENTE FOREIGN KEY(id_cliente) REFERENCES
CLIENTE
);
```

-- CRIANDO A TABELA DE ATENDIMENTOS

```
CREATE TABLE ATENDIMENTO (
        id_atendimento SERIAL NOT NULL,
        url_relatorio VARCHAR(300) NOT NULL UNIQUE,
        data_atend DATE NOT NULL,
        id_func INTEGER NOT NULL,
        id_cli INTEGER NOT NULL,
        CONSTRAINT PK_ATEND PRIMARY KEY(id_atendimento),
        CONSTRAINT FK_FUNC FOREIGN KEY (id_func) REFERENCES
FUNCIONARIO,
        CONSTRAINT FK_CLI FOREIGN KEY (id_cli) REFERENCES CLIENTE
);
```

-- CRIAÇÃO DA TABELA DE SERVIÇOS

```
CREATE TABLE SERVICO(
        id_servico SERIAL NOT NULL,
        tipo VARCHAR(45) NOT NULL,
        CONSTRAINT PK_SERVICO PRIMARY KEY(id_servico)
);
```

-- CRIAÇÃO DA TABELA ATENDIMENTO-SERVIÇO

```
CREATE TABLE ATENDSERVICO(
        id_servico INTEGER NOT NULL,
        id_atendimento INTEGER NOT NULL,
        CONSTRAINT FK_ATEND FOREIGN KEY(id_atendimento) REFERENCES
ATENDIMENTO,
        CONSTRAINT FK_SERVICO FOREIGN KEY(id_servico) REFERENCES
SERVICO,
        CONSTRAINT PK_ATENDSERV PRIMARY KEY(id_atendimento, id_servico)
);
```

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).**

-- Consulta que retorna o nome e cargo de funcionários que possuem apenas o cargo de programador

```
SELECT NOME, CARGO FROM FUNCIONARIO
WHERE CARGO LIKE 'Programador';
```

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

-- Consulta que retorna para cada funcionário a data dos deus atendimentos e o link dos seus relatórios

```
SELECT F.NOME AS "Funcionário", A.DATA_ATEND, A.URL_RELATORIO
        FROM ATENDIMENTO A JOIN
        FUNCIONARIO F
        ON A.ID_FUNC = F.ID_FUNCIONARIO;
```

-- Consulta que retorna o nome do cliente e o nome do funcionário que o atendeu no dia 07/11/2023

```
SELECT C.NOME AS "CLIENTE", F.NOME AS "ATENDIDO POR"
        FROM ATENDIMENTO A JOIN
        CLIENTE C ON C.ID_CLIENTE = A.ID_CLI
        JOIN FUNCIONARIO F ON F.ID_FUNCIONARIO = A.ID_FUNC
        WHERE A.DATA_ATEND = '07/11/2023';
```

-- Consulta que retorna o tipo de serviço que foi realizado no atendimento do dia 07/11/2023

```
SELECT S.TIPO
        FROM SERVICO S
        JOIN ATENDSERVICO ATS ON S.ID_SERVICO = ATS.ID_SERVICO
        JOIN ATENDIMENTO A ON ATS.ID_ATENDIMENTO= A.ID_ATENDIMENTO
        WHERE DATA_ATEND = '07/11/2023';
```

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

-- Consulta que retorna todos os clientes independente de ter atendimento ou não

```
SELECT NOME, A.DATA_ATEND
        FROM CLIENTE C
        LEFT JOIN ATENDIMENTO A
        ON C.ID_CLIENTE = A.ID_CLI;
```

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

-- Consulta que retorna a quantidade de cada tipo de serviço.

```
SELECT S.TIPO, COUNT(*) AS "Quantidade"
	FROM FUNCIONARIO F
	JOIN ATENDIMENTO A
	ON F.ID_FUNCIONARIO = A.ID_FUNC
	JOIN ATENDSERVICO ATS ON ATS.ID_ATENDIMENTO =
A.ID_ATENDIMENTO
	JOIN SERVICO S ON ATS.ID_SERVICO = S.ID_SERVICO
	GROUP BY S.TIPO;
```

-- Consulta que retorna a quantidade de funcionários para cada cargo na empresa.

```
SELECT CARGO, COUNT(*) AS "Quantidade"
	FROM FUNCIONARIO
	GROUP BY CARGO;
```

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

-- Consulta que retorna o cliente que não está na tabela de atendimento.

```
SELECT C.NOME
	FROM CLIENTE C
	EXCEPT
	SELECT C.NOME
	FROM CLIENTE C
	JOIN ATENDIMENTO A
	ON C.ID_CLIENTE = A.ID_CLI;
```

- **2 consultas que usem subqueries.**

-- Consulta que retorna todos os atendimentos do funcionário cujo nome é Gabryel Araújo

```
SELECT * FROM ATENDIMENTO
WHERE ID_FUNC IN (
	SELECT ID_FUNCIONARIO
		FROM FUNCIONARIO
		WHERE NOME LIKE 'Gabryel Araújo');
```

-- Consulta que retorna os id's dos serviços realizados no mês de dezembro

```
SELECT ID_SERVICO
FROM ATENDSERVICO
WHERE ID_ATENDIMENTO IN (
```

```
SELECT ID_ATENDIMENTO
FROM ATENDIMENTO
WHERE EXTRACT(MONTH FROM DATA_ATEND) = 12);
```

2.1 Visões

- **1 visão que permita inserção**

-- View de funcionários programadores que permite inserção

```
CREATE OR REPLACE VIEW FUNC_PROG
        (ID_FUNCIONARIO, NOME, CARGO, EMAIL ) AS
        SELECT * FROM FUNCIONARIO
        WHERE CARGO LIKE 'Programador'
        WITH CHECK OPTION;
```

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

-- View que retorna as visitas de cada funcionário

```
CREATE OR REPLACE VIEW VISITAS_FUNC AS
        SELECT F.NOME, F.CARGO, A.DATA_ATEND, S.TIPO
        FROM ATENDIMENTO A
        JOIN FUNCIONARIO F
        ON A.ID_FUNC = F.ID_FUNCIONARIO
        JOIN ATENDSERVICO ATS
        ON ATS.ID_ATENDIMENTO = A.ID_ATENDIMENTO
        JOIN SERVICO S
        ON S.ID_SERVICO = ATS.ID_SERVICO
ORDER BY NOME;
```

-- View que retorna todas as visitas realizadas pelos funcionários e seus respectivos clientes

```
CREATE OR REPLACE VIEW VISITAS_CLI AS
        SELECT C.NOME AS "Cliente", F.NOME AS "Funcionário", S.TIPO as "Serviço"
        FROM ATENDIMENTO A
        JOIN CLIENTE C
        ON A.ID_CLI = C.ID_CLIENTE
        JOIN FUNCIONARIO F
        ON A.ID_FUNC = F.ID_FUNCIONARIO
```

```
        JOIN ATENDSERVICO ATS
        ON ATS.ID_ATENDIMENTO = A.ID_ATENDIMENTO
        JOIN SERVICO S
        ON ATS.ID_SERVICO = S.ID_SERVICO
ORDER BY C.NOME;
```

2.2 Índices

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

-- Índice criado para otimizar a busca do funcionário, pois o campo email é chave candidata por garantir unicidade

```
CREATE INDEX INDEX_FUNCIONARIO ON FUNCIONARIO(email);
```

-- Índice criado para otimizar a busca caso haja conflito com nomes iguais, pois cada cliente possui um telefone único

```
CREATE INDEX INDEX_CLIENTE ON CLIENTE(telefone);
```

-- Índice criado para otimizar a busca com base na chamada das chaves estrangeiras da tabela

```
CREATE INDEX INDEX_ATENDIMENTO ON ATENDIMENTO(id_func,id_cli);
```

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.**

-- Consulta que retorna todos os atendimentos do funcionário cujo nome é Gabryel Araújo

```
/*
SELECT * FROM ATENDIMENTO
WHERE ID_FUNC IN (
        SELECT ID_FUNCIONARIO
                FROM FUNCIONARIO
                WHERE NOME LIKE 'Gabryel Araújo';
```

```
);
*/
```
-- Reescrita efetuada para otimizar o tempo de execução da consulta, uma vez que não haverá a necessidade da execução de uma consulta externa para chegar ao resultado final

```
SELECT *
FROM ATENDIMENTO A
JOIN FUNCIONARIO F ON F.ID_FUNCIONARIO = A.ID_FUNC
WHERE F.NOME LIKE 'Gabryel Araújo';
```

-- Consulta que retorna os id's dos serviços realizados no mês de dezembro
```
/*
SELECT ID_SERVICO
FROM ATENDSERVICO
WHERE ID_ATENDIMENTO IN (
   SELECT ID_ATENDIMENTO
   FROM ATENDIMENTO
   WHERE EXTRACT(MONTH FROM DATA_ATEND) = 12
);
*/
```
-- Reescrita efetuada para otimizar o tempo de execução da consulta, uma vez que não haverá a necessidade da execução de uma consulta externa para chegar ao resultado final

```
SELECT AD.ID_SERVICO
FROM ATENDSERVICO AD
JOIN ATENDIMENTO A ON A.ID_ATENDIMENTO = AD.ID_ATENDIMENTO
WHERE EXTRACT(MONTH FROM A.DATA_ATEND) = 12;
```

2.4. Funções e procedures armazenadas

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

-- Função que retorna o número total de visitas registradas na tabela ATENDIMENTO para um determinado mês de referência.
```
CREATE OR REPLACE FUNCTION visitasMes(mesReferencia integer)
RETURNS INTEGER
AS $$
```

```
DECLARE TOTAL INTEGER;
BEGIN
  SELECT COUNT(*) INTO TOTAL FROM ATENDIMENTO
  WHERE EXTRACT(MONTH FROM data_atend) = mesReferencia;
RETURN TOTAL;
END
$$ LANGUAGE plpgsql;
```

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

**/*
Função que tem o objetivo de contar o número de funcionários que possuem o cargo de 'Técnico' na tabela FUNCIONARIO.
Ela também verifica se o número total de técnicos é maior que 6 e, se for, levanta uma exceção indicando que o limite de 6 técnicos na equipe foi excedido. A função retorna o número total de técnicos
*/**

```
CREATE OR REPLACE FUNCTION contar_tecnicos()
RETURNS INTEGER
AS $$
DECLARE
   total_funcionarios INTEGER;
BEGIN
   SELECT COUNT(*) INTO total_funcionarios
         FROM FUNCIONARIO
         WHERE CARGO LIKE 'Técnico';
   IF total_funcionarios > 6 THEN
      RAISE EXCEPTION 'Limite de 6 técnicos na equipe foi excedido';
   END IF;
   RETURN total_funcionarios;
END;
$$ LANGUAGE plpgsql;
```

-- Função que retorna uma tabela com duas colunas: mes e quantidade. Ela conta o número de visitas na tabela ATENDIMENTO para três meses específicos fornecidos como parâmetros (mes1, mes2, mes3)

```
CREATE OR REPLACE FUNCTION getVisitasTri(mes1 integer, mes2 integer, mes3 integer)
RETURNS TABLE (mes integer, quantidade integer)
AS $$
```

```
BEGIN
  RETURN QUERY
  SELECT EXTRACT(MONTH FROM data_atend)::integer AS mes, COUNT(*)::integer AS
quantidade
  FROM atendimento
  WHERE EXTRACT(MONTH FROM data_atend) IN (mes1, mes2, mes3)
  GROUP BY mes
  ORDER BY mes;
END;
$$
LANGUAGE plpgsql;
```

-- Procedure que insere um novo tipo de serviço na tabela SERVICO.

```
CREATE OR REPLACE PROCEDURE INSERIR_SERVICO(
   NOVOTIPO VARCHAR(45)
)
AS $$
BEGIN
   INSERT INTO SERVICO(tipo) VALUES (NOVOTIPO);
END;
$$ LANGUAGE plpgsql;
```

2.5 Triggers

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

-- Conjunto de triggers (1.1, 1.2, 1.3) responsável por gerar logs das operações feitas nas principais tabelas do banco de dados (FUNCIONARIO, CLIENTE ,ATENDIMENTO)

-- Criação da tabela de log

```
CREATE TABLE log_operacoes (
   id_log SERIAL NOT NULL,
   tabela VARCHAR(255) NOT NULL,
   operacao VARCHAR(10) NOT NULL,
   id_registro INT,
   data_operacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (id_log));
```

-- Trigger 1.1

-- Criação da função de log para o funcionário

```
CREATE OR REPLACE FUNCTION log_operacoes_func()
RETURNS TRIGGER AS $$
BEGIN
   IF (TG_OP = 'DELETE') THEN
      INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
      VALUES (TG_TABLE_NAME, 'Delete', OLD.id_funcionario,
CURRENT_TIMESTAMP);
      RETURN OLD;
   ELSIF (TG_OP = 'UPDATE') THEN
      INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
      VALUES (TG_TABLE_NAME, 'Update', NEW.id_funcionario,
CURRENT_TIMESTAMP);
      RETURN NEW;
   ELSIF (TG_OP = 'INSERT') THEN
      INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
      VALUES (TG_TABLE_NAME, 'Insert', NEW.id_funcionario,
CURRENT_TIMESTAMP);
      RETURN NEW;
   END IF;
   RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

-- Criação do trigger para funcionário

```
CREATE TRIGGER operacoes_funcionario
AFTER INSERT OR UPDATE OR DELETE
ON funcionario
FOR EACH ROW
EXECUTE FUNCTION log_operacoes_func();
```

-- Trigger 1.2

-- Criação da função de log para o cliente

```
CREATE OR REPLACE FUNCTION log_operacoes_cli()
RETURNS TRIGGER AS $$
BEGIN
   IF (TG_OP = 'DELETE') THEN
```

```
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Delete', OLD.id_cliente, CURRENT_TIMESTAMP);
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Update', NEW.id_cliente, CURRENT_TIMESTAMP);
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Insert', NEW.id_cliente, CURRENT_TIMESTAMP);
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

-- Criação do trigger para Cliente

```
CREATE TRIGGER operacoes_cliente
AFTER INSERT OR UPDATE OR DELETE
ON cliente
FOR EACH ROW
EXECUTE FUNCTION log_operacoes_cli();
```

-- Trigger 1.3

-- Criação da função de log para o Atendimento

```
CREATE OR REPLACE FUNCTION log_operacoes_atend()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Delete', OLD.id_atendimento,
CURRENT_TIMESTAMP);
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Update', NEW.id_atendimento,
CURRENT_TIMESTAMP);
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
```

```
        INSERT INTO log_operacoes (tabela, operacao, id_registro, data_operacao)
        VALUES (TG_TABLE_NAME, 'Insert', NEW.id_atendimento,
CURRENT_TIMESTAMP);
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

-- Criação do trigger para Atendimento

```
CREATE TRIGGER operacoes_atendimento
AFTER INSERT OR UPDATE OR DELETE
ON atendimento
FOR EACH ROW
EXECUTE FUNCTION log_operacoes_atend();
```

-- Trigger 2

-- Trigger associado à tabela ATENDIMENTO para validar a data antes de realizar a operação de inserção, se a data estiver no passado, será acionada uma exceção e o registro não será concretizado.

-- Função para validar a data de atendimento

```
CREATE OR REPLACE FUNCTION valida_data_atendimento()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.data_atend < CURRENT_DATE) THEN
        RAISE EXCEPTION 'A data do atendimento não pode ser no passado.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

-- Trigger para validar a data de atendimento

```
CREATE TRIGGER trigger_valida_data_atendimento
BEFORE INSERT
ON atendimento
FOR EACH ROW
EXECUTE FUNCTION valida_data_atendimento();
```

-- Trigger 3

-- Trigger associado à tabela FUNCIONARIO para validar o formato do e-mail antes de realizar operações de inserção ou atualização, se o formato não estiver de acordo, uma exceção será levantada e a operação será interrompida.

-- Função para validar o formato do e-mail

```
CREATE OR REPLACE FUNCTION valida_formato_email()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.email IS NOT NULL AND NOT NEW.email ~
'^[a-zA-Z0-9.%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$' THEN
      RAISE EXCEPTION 'Formato de e-mail inválido: %', NEW.email;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

-- Trigger para validar o formato do e-mail

```
CREATE TRIGGER trigger_valida_formato_email
BEFORE INSERT OR UPDATE
ON funcionario
FOR EACH ROW
EXECUTE FUNCTION valida_formato_email();
```

-- A expressão regular '^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$' valida se o formato do e-mail é válido. Ela permite letras maiúsculas e minúsculas, números, pontos, hífens e porcentagens no nome do usuário, seguidos por um símbolo "@" e um domínio contendo letras, números e pontos, e uma extensão de domínio de 2 a 4 caracteres.