Atividade a Distância – Estudos de caso para o projeto conceitual
Disciplina: Banco de dados II
Professor: Ângelo Augusto Frozza
Aluno: Luis Fernando Pizzirani

TRABALHO FINAL DE BANCO DE DADOS II

a) Estudo de caso: Ecommerce

Este trabalho descreve o desenvolvimento de um banco de dados destinado ao gerenciamento de um e-commerce especializado na venda de produtos. O sistema foi projetado para proporcionar aos usuários uma experiência de compra completa, com funcionalidades que permitem a visualização de produtos, a realização de pedidos e o acompanhamento do status das compras. Para os administradores, o banco de dados oferece ferramentas eficientes para o controle de estoque, a gestão de vendas e a análise de dados por meio de consultas e relatórios detalhados.

O banco de dados suporta funcionalidades essenciais para o funcionamento da plataforma, incluindo o cadastro de clientes, o cadastro de produtos e o gerenciamento de pedidos. Cada cliente é identificado de forma única, o que permite o rastreamento de seus pedidos e interações. Os produtos são registrados com informações detalhadas, como nome, descrição, preço e categoria, e são associados ao estoque, permitindo uma gestão precisa. O sistema também possibilita a criação de pedidos, onde cada compra é vinculada a um cliente específico e detalhada com informações sobre os produtos adquiridos.

Além dessas funcionalidades, o banco de dados oferece suporte a consultas dinâmicas e relatórios, como a análise de vendas por região e categoria de produto, e o monitoramento do estoque. Esses relatórios são fundamentais para a avaliação do desempenho de vendas e para ações de reposição de produtos. O sistema foi modelado seguindo boas práticas de normalização e de relacionamentos entre as entidades, com o uso de chaves estrangeiras e domínios para garantir a integridade e consistência dos dados, como validação de e-mails, números de telefone e valores monetários.

A aplicação deverá emitir consultas e relatórios como:

-- Consulta de Produtos por Categoria: Busca simples de produtos dentro de uma específica.

-- Consulta de Pedidos por Clientes: Exibe os pedidos realizados por um cliente, com detalhes sobre os produtos e valores.

-- Relatório de Estoque Baixo: Identificação de produtos com estoque abaixo de um valor de referência.

-- Relatório de Vendas Mensais: Análise do desempenho de vendas ao longo do tempo, considerando variáveis como quantidade e valor.

-- Relatório de Vendas por Região: Análise mais complexa que envolve a distribuição geográfica das vendas e seu desempenho por região.

b) Diagrama conceitual

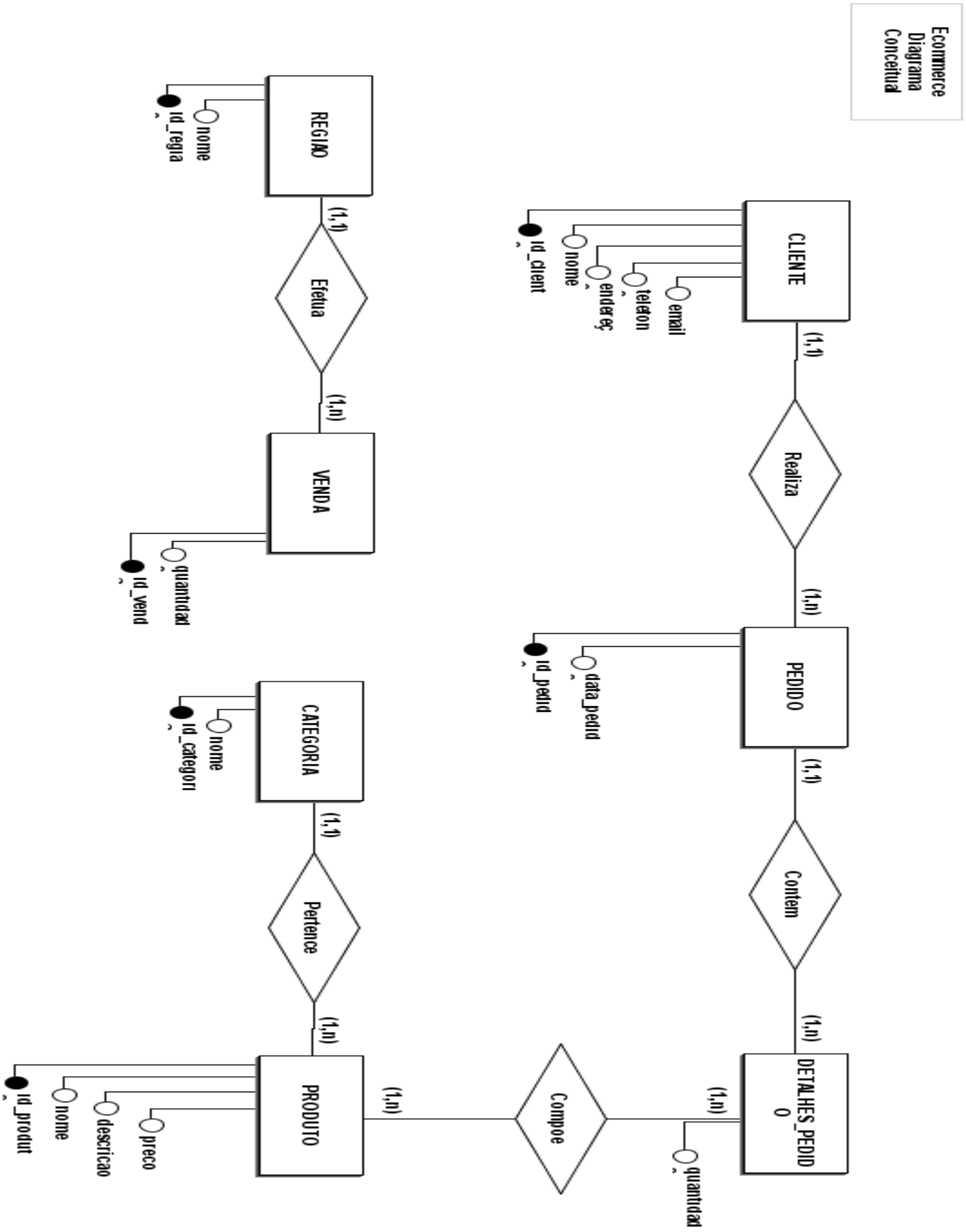
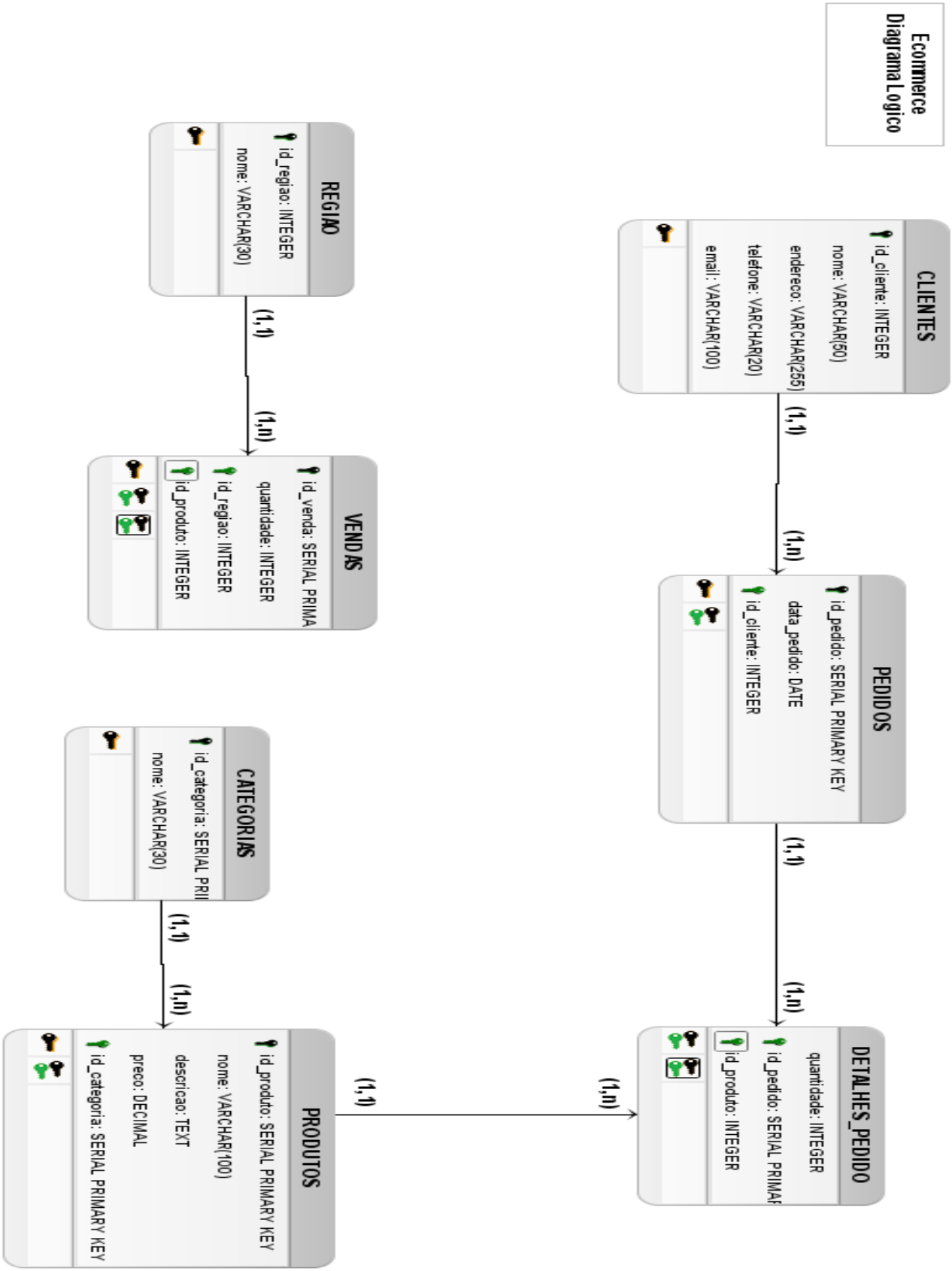


Diagrama logico



c) Projeto Físico, scripts DDL e DML para o PostgreSQL

```
CREATE DATABASE Ecommerce;
```

-- Criação de Dominios para dados comuns

```
CREATE DOMAIN telDominio AS VARCHAR(20)
    CHECK (VALUE ~ '^\(?\d{2}\)?\s?\d{4,5}-?\d{4}$');

CREATE DOMAIN emailDominio AS VARCHAR(100)
    CHECK (VALUE ~ '^[\w\.-]+@[\w\.-]+\.\w{2,4}$');

CREATE DOMAIN precoDominio AS DECIMAL(10, 2)
    CHECK (VALUE >= 0);

CREATE DOMAIN quantidadeDominio AS INTEGER
    CHECK (VALUE >= 0);
```

-- Criação de tabelas

```
CREATE TABLE categorias (
    id_categoria SERIAL PRIMARY KEY,
    nome VARCHAR(30) NOT NULL
);

CREATE TABLE produtos (
    id_produto SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
```

```
    descricao TEXT,
    preco precoDominio NOT NULL,
    id_categoria INTEGER,
    CONSTRAINT produto_categoria FOREIGN KEY (id_categoria)
    REFERENCES categorias(id_categoria) ON DELETE RESTRICT
);

CREATE TABLE clientes (
    id_cliente SERIAL PRIMARY KEY,
    nome VARCHAR(30) NOT NULL,
    endereco VARCHAR(255),
    telefone telDominio,
    email emailDominio
);

CREATE TABLE pedidos (
    id_pedido SERIAL PRIMARY KEY,
    data_pedido DATE NOT NULL,
    id_cliente INTEGER,
    CONSTRAINT pedido_cliente FOREIGN KEY (id_cliente)
    REFERENCES clientes(id_cliente) ON DELETE RESTRICT
);

CREATE TABLE detalhes_pedido (
    quantidade quantidadeDominio NOT NULL,
    id_pedido INTEGER,
    id_produto INTEGER,
    CONSTRAINT detalhes_pedido FOREIGN KEY (id_pedido)
    REFERENCES pedidos(id_pedido) ON DELETE RESTRICT,
    CONSTRAINT detalhes_produto FOREIGN KEY (id_produto)
    REFERENCES produtos(id_produto) ON DELETE RESTRICT
```

```
 );

CREATE TABLE regiao (
    id_regiao SERIAL PRIMARY KEY,
    nome VARCHAR(30) NOT NULL
);

CREATE TABLE vendas (
    id_venda SERIAL PRIMARY KEY,
    quantidade quantidadeDominio NOT NULL,
    id_regiao INTEGER,
    id_produto INTEGER,
    CONSTRAINT venda_regiao FOREIGN KEY (id_regiao)
    REFERENCES regiao(id_regiao) ON DELETE RESTRICT,
    CONSTRAINT venda_produto FOREIGN KEY (id_produto)
    REFERENCES produtos(id_produto) ON DELETE RESTRICT
);
```

-- DML do banco de dados Ecommerce

-- Inclusão de 5 registros para cada tabela;

-- Inserindo categorias

```
INSERT INTO categorias (nome)
VALUES
('Livros'),
('Eletrodomésticos'),
('Vestuário'),
('Móveis'),
```

```
 ('Brinquedos');


-- Inserindo produtos
INSERT INTO produtos (nome, descricao, preco, id_categoria)
VALUES
('Cadeira de Escritório', 'Cadeira ergonômica para
escritório', 199.99, 4),
('Livro: O Senhor dos Anéis', 'Livro de fantasia épica',
49.90, 1),
('Televisão LED 32"', 'Televisão de 32 polegadas, LED Full
HD', 999.90, 2),
('Camisa Polo', 'Camisa polo masculina', 79.90, 3),
('Bicicleta Infantil', 'Bicicleta infantil para crianças',
299.90, 5);


-- Inserindo clientes
INSERT INTO clientes (nome, endereco, telefone, email)
VALUES
('João Silva', 'Rua A, 123', '11987654321',
'joao.silva@example.com'),
('Maria Oliveira', 'Rua B, 456', '21987654321',
'maria.oliveira@example.com'),
('Carlos Souza', 'Rua C, 789', '31987654321',
'carlos.souza@example.com'),
('Ana Santos', 'Rua D, 101', '41987654321',
'ana.santos@example.com'),
('Roberta Lima', 'Rua E, 202', '61987654321',
'roberta.lima@example.com');


-- Inserindo regiões
INSERT INTO regiao (nome)
VALUES
('Sudeste'),
('Nordeste'),
```

```
 ('Sul'),
('Centro-Oeste'),
('Norte');

-- Inserindo pedidos
INSERT INTO pedidos (data_pedido, id_cliente)
VALUES
('2023-09-01', 1),
('2020-07-12', 2),
('2024-05-30', 3),
('2024-12-20', 4),
('2024-10-01', 5);

-- Inserindo vendas
INSERT INTO vendas (quantidade, id_regiao, id_produto)
VALUES
(10, 1, 2),
(5, 2, 3),
(8, 3, 4),
(7, 4, 5),
(12, 5, 1);

-- Inserindo detalhes de pedidos
INSERT INTO detalhes_pedido (quantidade, id_pedido,
id_produto)
VALUES
(2, 1, 2),
(3, 2, 3),
(1, 3, 4),
(5, 4, 5),
(4, 5, 1);
```

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

-- Consulta de Produtos por Categoria

Aqui '' podemos escolher a categoria desejada, podendo ser filtrada pelo nome ou id_Categoria

```
SELECT
    pr.id_produto,
    pr.nome AS produto_nome,
    pr.descricao,
    pr.preco,
    c.nome AS categoria_nome
FROM
    produtos pr
INNER JOIN
    categorias c ON pr.id_categoria = c.id_categoria
WHERE
    c.nome = 'Eletrodomésticos';
```

Consulta de Pedidos por Clientes

Esta consulta exibe os pedidos realizados por um cliente, incluindo os detalhes dos produtos, quantidades e valores totais.

```
SELECT
    p.id_pedido,
    p.data_pedido,
    c.nome AS cliente_nome,
    pr.nome AS produto_nome,
    dp.quantidade,
```

```
    pr.preco,
    dp.quantidade * pr.preco AS total_item
FROM
    pedidos p
INNER JOIN
    clientes c ON p.id_cliente = c.id_cliente
INNER JOIN
    detalhes_pedido dp ON p.id_pedido = dp.id_pedido
INNER JOIN
    produtos pr ON dp.id_produto = pr.id_produto
WHERE
    c.id_cliente = 1;
```

Relatório de Estoque Baixo

Identifica produtos com estoque abaixo de um valor de referência, no quaso escolhi '<10' como referencia

```
SELECT
    pr.id_produto,
    pr.nome AS produto_nome,
    SUM(v.quantidade) AS total_vendido,
    pr.preco,
    (SUM(v.quantidade) < 10) AS estoque_baixo
FROM
    produtos pr
INNER JOIN
    vendas v ON pr.id_produto = v.id_produto
GROUP BY
    pr.id_produto, pr.nome, pr.preco
HAVING
```

```
    SUM(v.quantidade) < 10;
```

-- Relatorio das Vendas mensais

Análise de vendas realizadas por mês, considerando a quantidade vendida e o valor total, a consulta considera que a data das vendas está na tabela vendas e a data do pedido está na tabela pedidos.

```
SELECT
    EXTRACT(YEAR FROM p.data_pedido) AS ano,
    EXTRACT(MONTH FROM p.data_pedido) AS mes,
    pr.nome AS produto_nome,
    SUM(dp.quantidade) AS total_vendido,
    SUM(dp.quantidade * pr.preco) AS valor_total
FROM
    pedidos p
INNER JOIN
    detalhes_pedido dp ON p.id_pedido = dp.id_pedido
INNER JOIN
    produtos pr ON dp.id_produto = pr.id_produto
GROUP BY
    ano, mes, pr.id_produto
ORDER BY
    ano, mes, total_vendido DESC;
```

-- Relatório de Vendas por Região

Análise de vendas por região, considerando a quantidade de vendas e o valor total. Esse relatório envolve a tabela vendas e a tabela regiao.

```
SELECT
    r.nome AS regiao_nome,
    pr.nome AS produto_nome,
    SUM(v.quantidade) AS total_vendido,
    SUM(v.quantidade * pr.preco) AS valor_total
FROM
    vendas v
INNER JOIN
    regiao r ON v.id_regiao = r.id_regiao
INNER JOIN
    produtos pr ON v.id_produto = pr.id_produto
GROUP BY
    r.id_regiao, pr.id_produto
ORDER BY
    valor_total DESC;
```

---

e) Scripts SQL para Stored Procedures (incluir comentário explicando para que serve a

SP e os parâmetros caso utilizar).

Stored Procedure para cadastrar um novo cliente

Parâmetros:

```
-- p_nome (nome do cliente)
-- p_endereco (endereço do cliente)
-- p_telefone (telefone do cliente)
-- p_email (email do cliente)
```

```
 CREATE OR REPLACE PROCEDURE sp_Cadastrar_Cliente(
    p_nome VARCHAR(30),
    p_endereco VARCHAR(255),
    p_telefone VARCHAR(20),
    p_email VARCHAR(100)
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Inserir um novo cliente na tabela clientes
    INSERT INTO clientes (nome, endereco, telefone, email)
    VALUES (p_nome, p_endereco, p_telefone, p_email);
END;
$$;
```

--Chamando a Procedure

```
CALL sp_Cadastrar_Cliente('João Silva', 'Rua A, 123',
'11987654321', 'joao.silva@example.com');
```

- Stored Procedure para atualizar o preço de um produto

Parâmetros:

```
-- p_id_produto (ID do produto a ser atualizado)
-- p_novo_preco (novo preço do produto)

CREATE OR REPLACE PROCEDURE sp_Atualizar_Preco_Produto(
    p_id_produto INTEGER,
    p_novo_preco DECIMAL(10, 2)
)
```

```
 LANGUAGE plpgsql
AS $$
BEGIN
    -- Atualiza o preço do produto com base no ID do
produto
    UPDATE produtos
    SET preco = p_novo_preco
    WHERE id_produto = p_id_produto;

    -- Caso deseje verificar se a atualização foi realizada
    IF NOT FOUND THEN
        RAISE NOTICE 'Produto com ID % não encontrado.',
p_id_produto;
    END IF;
END;
$$;
```

-- Chamando a procedure

```
CALL sp_Atualizar_Preco_Produto(2, 150.99);
```

---

f) Scripts SQL para Functions (incluir comentário explicando para que serve a Function e os parâmetros de entrada e valor de saída).

Função para aplicar desconto a todos os produtos de uma categoria

```
-- Parâmetros:
-- p_id_categoria: ID da categoria dos produtos
-- p_desconto: porcentagem de desconto a ser aplicada
-- Retorna o número de produtos que tiveram o desconto
aplicado.

CREATE OR REPLACE FUNCTION fn_Aplicar_Desconto_Categoria(
```

```
    p_id_categoria INTEGER,
    p_desconto DECIMAL(5, 2)
)
RETURNS INTEGER AS $$
DECLARE
    v_produtos_afetados INTEGER;
BEGIN
    -- Aplica o desconto a todos os produtos da categoria
    UPDATE produtos
    SET preco = preco - (preco * p_desconto / 100)
    WHERE id_categoria = p_id_categoria;

    -- Conta quantos produtos foram afetados
    GET DIAGNOSTICS v_produtos_afetados = ROW_COUNT;

    -- Retorna o número de produtos que tiveram o desconto
aplicado
    RETURN v_produtos_afetados;
END;
$$ LANGUAGE plpgsql;
```

Chamando a Function que ira aplicar 10% de desconto aos produtos da categoria ID 2, ou seja os Eletrodomésticos

```
SELECT fn_Aplicar_Desconto_Categoria(2, 10);
```

Função para calcular o total de vendas de um produto em uma região específica

```
-- Parâmetros:
-- p_id_produto: ID do produto
-- p_id_regiao: ID da região
-- Retorna o valor total de vendas do produto na região especificada.

CREATE OR REPLACE FUNCTION fn_Calcular_Vendas_Por_Regiao(
    p_id_produto INTEGER,
    p_id_regiao INTEGER
)
RETURNS DECIMAL(10, 2) AS $$
DECLARE
    v_total_vendas DECIMAL(10, 2);
BEGIN
    -- Calcula o total de vendas do produto na região especificada
    SELECT SUM(v.quantidade * pr.preco)
    INTO v_total_vendas
    FROM vendas v
    JOIN produtos pr ON v.id_produto = pr.id_produto
    WHERE v.id_produto = p_id_produto
    AND v.id_regiao = p_id_regiao;

    -- Se não houver vendas, retorna 0
    IF v_total_vendas IS NULL THEN
        RETURN 0;
    END IF;

    -- Retorna o total de vendas
```

```
    RETURN v_total_vendas;
END;
$$ LANGUAGE plpgsql;

-- Chamando a function que retorna o total de vendas da regiao '1' ou seja Sudeste

SELECT fn_Calcular_Vendas_Por_Regiao(2, 1);
```

g) Scripts SQL para Views (incluir comentário explicando para que serve a View).

View para exibir as vendas por produto e região

Objetivo: Esta view mostra o total de vendas de cada produto por região, incluindo a quantidade e o valor total das vendas. Ela une as tabelas de vendas, produtos e regiões para gerar o relatório.

```
CREATE OR REPLACE VIEW vw_vendas_por_produto_regiao AS
SELECT
    pr.nome AS produto_nome,
    r.nome AS regiao_nome,
    SUM(v.quantidade) AS total_vendido,
    SUM(v.quantidade * pr.preco) AS valor_total_vendido
FROM
    vendas v
JOIN
    produtos pr ON v.id_produto = pr.id_produto
JOIN
    regiao r ON v.id_regiao = r.id_regiao
GROUP BY
    pr.id_produto, r.id_regiao
```

```
ORDER BY
    valor_total_vendido DESC;
```

-- Chamando a View

```
SELECT * FROM vw_vendas_por_produto_regiao;
```

-- View para exibir os clientes e seus pedidos mais recentes

Objetivo: Esta view retorna os clientes com seu respectivo pedido mais recente, incluindo a data e o ID do pedido. Ela utiliza as tabelas clientes e pedidos para obter as informações.

```
CREATE OR REPLACE VIEW vw_clientes_ultimos_pedidos AS
SELECT
    c.nome AS cliente_nome,
    p.id_pedido,
    p.data_pedido
FROM
    clientes c
JOIN
    pedidos p ON c.id_cliente = p.id_cliente
WHERE
    p.data_pedido = (
        SELECT MAX(data_pedido)
        FROM pedidos
        WHERE id_cliente = c.id_cliente
    );
```

-- chamando view

```
SELECT * FROM vw_clientes_ultimos_pedidos;
```

h) Scripts SQL para Triggers (incluir comentário explicando para que serve a Trigger).

-- Essa trigger ajuda a manter o estoque atualizado automaticamente, e evita que o sistema permita pedidos quando não houver estoque suficiente.

Objetivo: Atualizar automaticamente o estoque de um produto quando um pedido for realizado.

-- A trigger será acionada após a inserção de um novo item na tabela 'detalhes_pedido'

-- (ou seja, após a inserção de um pedido).

-- A trigger vai atualizar o estoque do produto subtraindo a quantidade do pedido.

```
CREATE OR REPLACE FUNCTION
fn_atualizar_estoque_apos_pedido()

RETURNS TRIGGER AS $$

BEGIN

    -- Atualiza o estoque do produto, subtraindo a
quantidade pedida

    UPDATE produtos

    SET quantidade_estoque = quantidade_estoque -
NEW.quantidade

    WHERE id_produto = NEW.id_produto;



    -- Retorna o novo registro inserido na tabela
'detalhes_pedido'

    RETURN NEW;

END;
```

```
$$ LANGUAGE plpgsql;
```

-- Criando a trigger 'trg_atualizar_estoque, a trigger será acionada 'AFTER INSERT' na tabela 'detalhes_pedido'

-- Para cada nova linha inserida, ela chama a função 'fn_atualizar_estoque_apos_pedido'que atualiza o estoque do produto correspondente.

```
CREATE TRIGGER trg_atualizar_estoque

AFTER INSERT ON detalhes_pedido

FOR EACH ROW

EXECUTE FUNCTION fn_atualizar_estoque_apos_pedido();
```

-- Teste: Ao inserir um novo pedido, por exemplo:

```
INSERT INTO detalhes_pedido (quantidade, id_pedido,
id_produto)

VALUES (3, 1, 2);
```

-- A trigger irá automaticamente reduzir a quantidade disponível do produto com id_produto = 2

-- no estoque. Se a quantidade em estoque não for suficiente para atender à quantidade do pedido (3),

-- a trigger gerará um erro.

---

i) Definição de Usuários e permissões (incluir os comandos para implementação dessas regras).

-Criação do usuário administrador do sistema de e-commerce

CREATE USER admin_ecommerce PASSWORD 'admin123456';

-- Criação do usuário gerente de estoque

CREATE USER gerente_estoque PASSWORD 'estoque123456';

-- Criação do usuário vendedor

CREATE USER vendedor PASSWORD 'venda123456';

-- Permissões para o administrador

GRANT SELECT, INSERT, UPDATE, DELETE ON produtos, pedidos, clientes, vendas, regiao, categorias, detalhes_pedido TO admin_ecommerce;

-- Permissões para o gerente de estoque

GRANT SELECT, INSERT, UPDATE ON produtos, categorias TO gerente_estoque;

GRANT SELECT ON vendas TO gerente_estoque;

-- Permissões para o vendedor

GRANT SELECT, INSERT ON pedidos, clientes, detalhes_pedido TO vendedor;

GRANT SELECT ON produtos, categorias TO vendedor;

Link para os arquivos:
https://drive.google.com/drive/u/0/folders/1iZZGFVr3hcBfA516mwzGJKA4VJLsj55G