UNIVERSIDADE POSITIVO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VITTOR MARQUES DALLACQUA LONGATI

TRABALHO FINAL – BANCO DE DADOS

SISTEMA DE GERENCIAMENTO DE ESTOQUE E VENDAS

CURITIBA
2024

SUMÁRIO

1. **MODELAGEM DO BANCO DE DADOS**

Código referente à criação da tabela ‘Produtos’:

```sql
CREATE TABLE produtos(
    produto_id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco DECIMAL (10, 2) NOT NULL,
    estoque INT NOT NULL,
    categoria_id INT,
    fornecedor_id INT,
    FOREIGN KEY(categoria_id) REFERENCES categorias(categoria_id),
    FOREIGN KEY(fornecedor_id) REFERENCES fornecedor(fornecedor_id)
);
```

Registra informações sobre cada produto, incluindo ‘produto_id’ (chave primária), ‘nome’, ‘descricao’, ‘preco’, estoque e duas chaves estrangeiras (‘categoria_id’ e ‘fornecedor_id’) que se relacionam com as tabelas categorias e fornecedores, indicando a categoria e o fornecedor de cada produto.

Código referente à criação da tabela ‘Categorias’:

```sql
CREATE TABLE categorias(
    categoria_id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    descricao TEXT
);
```

Armazena informações sobre as categorias dos produtos. Inclui um ‘categoria_id’ (chave primária única com incremento automático), ‘nome’ obrigatório para a categoria e uma ‘descricao’ opcional, permitindo organizar produtos em diferentes categorias.

Código referente à criação da tabela ‘Fornecedores’:

```
CREATE TABLE fornecedores(
    fornecedor_id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    contato VARCHAR(50),
    endereco VARCHAR(150),
    cidade VARCHAR(50)
);
```

Contém dados sobre os fornecedores dos produtos, com ‘fornecedor_id’ como chave primária, além de ‘nome’ obrigatório, ‘contato’ opcional, 'endereco’ e ‘cidade’. Esses dados são úteis para identificar e contatar os fornecedores.

Código referente à criação da tabela ‘Clientes’:

```
CREATE TABLE clientes(
    cliente_id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefone VARCHAR(20),
    cpf VARCHAR(14) NOT NULL,
    endereco VARCHAR(150),
    cidade VARCHAR(50)
);
```

Armazena os dados dos clientes, incluindo ‘cliente_id’ (chave primária), ‘nome’, ‘email’, ‘telefone’, ‘cpf’, ‘endereco’ e ‘cidade’, que são informações necessárias para o contato e identificação dos clientes.

Código referente à criação da tabela ‘Funcionários’:

```
CREATE TABLE funcionarios(
    funcionarios_id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    cargo VARCHAR(50) NOT NULL,
    salario DECIMAL(10, 2) NOT NULL,
    data_contratacao DATE,
    email VARCHAR(100),
    telefone VARCHAR(20)
);
```

Guarda informações sobre os funcionários, com ‘funcionarios_id’ como chave primária, além de ‘nome’, ‘cargo’, ‘salario’, ‘data_contratacao’, ‘email’, e ‘telefone’. Esses dados são importantes para gerenciar quem participa dos processos de venda e atendimento ao cliente.

Código referente à criação da tabela ‘Pedidos’:

```
CREATE TABLE pedidos(
        pedido_id INT AUTO_INCREMENT PRIMARY KEY,
        data_pedido DATE NOT NULL,
        cliente_id INT,
        funcionario_id INT,
        status_pedido VARCHAR(20),
        FOREIGN KEY(cliente_id) REFERENCES clientes(cliente_id),
        FOREIGN KEY(funcionario_id) REFERENCES funcionarios(funcionario_id)
    );
```

Armazena os pedidos realizados, com ‘pedido_id’ como chave primária, ‘data_pedido’, e ‘status_pedido’, além de ‘cliente_id’ e ‘funcionario_id’ como chaves estrangeiras, relacionando cada pedido a um cliente e a um funcionário que o processou.

Código referente à criação da tabela ‘Itens-Pedido’:

```
CREATE TABLE itens_pedido(
    item_pedido_id INT AUTO_INCREMENT PRIMARY KEY,
    pedido_id INT,
    produto_id INT,
    quantidade INT NOT NULL,
    preco_unitario DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY(pedido_id) REFERENCES pedidos(pedido_id),
    FOREIGN KEY(produto_id) REFERENCES produtos(produto_id)
);
```

Contém os detalhes dos itens de cada pedido, incluindo ‘item_pedido_id’ (chave primária), ‘pedido_id’ e ‘produto_id’ como chaves estrangeiras, ‘quantidade’ de cada item no pedido e ‘preco_unitario’. Essa tabela permite detalhar cada produto incluído em um pedido específico.

Código referente à criação da tabela ‘Pagamentos’:

```
CREATE TABLE pagamentos(
    pagamento_id INT AUTO_INCREMENT PRIMARY KEY,
    pedido_id INT,
    valor DECIMAL(10, 2) NOT NULL,
    metodo_pagamento VARCHAR(50),
    data_pagamento DATE NOT NULL,
    FOREIGN KEY(pedido_id) REFERENCES pedidos(pedido_id)
);
```

Registra os pagamentos dos pedidos, com ‘pagamento_id’ como chave primária, ‘pedido_id’ como chave estrangeira para associar ao pedido, ‘valor’ do pagamento, ‘metodo_pagamento’, e ‘data_pagamento’.

2. OPERAÇÕES DE MANIPULAÇÃO DE DADOS

Código referente às inserções na tabela ‘Categorias’:

```
INSERT INTO categorias (nome)
VALUES
    ('Tecnologia e Eletrônicos'),
    ('Eletrodomésticos'),
    ('Acessórios de Tecnologia'),
    ('Utensílios para Cozinha'),
    ('Decoração e Móveis'),
    ('Moda e Vestuário');
```

Este bloco insere seis categorias de produtos, como "Tecnologia e Eletrônicos", "Eletrodomésticos" e "Decoração e Móveis". Cada categoria representa um tipo distinto de produto para organizar o estoque de forma segmentada e fácil de navegar.

Código referente às inserções na tabela ‘Fornecedores’:

```
INSERT INTO fornecedores (nome, contato, endereco, cidade)
VALUES
    ('Tech World', 'techworld@exemplo.com', 'Av. das Tecnologias, 1234', 'São Paulo'),
    ('Digital Store', 'contato@digitalstore.com.br', 'Rua dos Eletrônicos, 567', 'Rio de Janeiro'),
    ('FrioTech', 'atendimento@friotech.com', 'Av. dos Eletros, 890', 'Curitiba'),
    ('KitchenPlus', 'suporte@kitchenplus.com', 'Rua da Gastronomia, 321', 'Belo Horizonte'),
    ('HomeDeco', 'contato@homedeco.com', 'Av. do Design, 654', 'Porto Alegre');
```

Neste bloco, são adicionados cinco fornecedores com informações como nome, contato e endereço. Esses dados permitem que o sistema rastreie a origem dos produtos e facilite a comunicação e logística com cada fornecedor.

Código referente às inserções na tabela ‘Produtos’:

```
INSERT INTO produtos (nome, preco, estoque, categoria_id, fornecedor_id)
VALUES
    ('Smartphone X10', 1500.00, 25, 1, 1),
    ('Notebook Ultrafino', 3200.00, 3, 1, 2),
    ('Fone de Ouvido Bass+', 150.00, 18, 3, 1),
    ('Geladeira FrostMax', 2800.00, 5, 2, 3),
    ('Cafeteira Express', 400.00, 30, 4, 4),
    ('Air Fryer HealthPlus', 300.00, 4, 2, 4),
    ('Tênis Confort 2024', 250.00, 2, 6, 5),
    ('Jaqueta Impermeável', 350.00, 50, 6, 5),
    ('Camiseta Eco', 70.00, 15, 6, 5),
    ('Mesa de Escritório Compact', 600.00, 4, 5, 5),
    ('Cadeira Gamer', 1200.00, 10, 5, 2),
    ('Mouse sem fio', 80.00, 45, 3, 1),
    ('Teclado Mecânico Pro', 200.00, 8, 3, 2),
    ('Caixa de Som Bluetooth', 300.00, 27, 3, 1),
    ('Micro-ondas Compact', 500.00, 12, 2, 4),
    ('Liquidificador Turbo', 150.00, 6, 2, 3),
    ('Relógio Digital', 120.00, 40, 3, 1),
    ('Cadeira Dobrável', 150.00, 3, 5, 5),
    ('Aspirador de Pó PowerClean', 750.00, 21, 2, 3),
    ('Smart TV 50\" 4K', 2800.00, 7, 1, 2);
```

Este bloco insere vinte produtos, detalhando nome, preço, quantidade em estoque, categoria e fornecedor. Essa tabela associa cada produto a uma categoria específica e a um fornecedor, organizando o inventário e ajudando no controle de estoque e reabastecimento.

Código referente às inserções na tabela ‘Clientes’:

```
INSERT INTO clientes (nome, email, telefone, cpf, endereco, cidade)
VALUES
    ('Ana Souza', 'ana.souza@email.com', '(11) 91234-5678', '123.456.789-01', 'Rua das Flores, 100', 'São Paulo'),
    ('Bruno Oliveira', 'bruno.oliveira@email.com', '(21) 98765-4321', '234.567.890-12', 'Av. Atlântica, 555', 'Rio de Janeiro'),
    ('Carla Mendes', 'carla.mendes@email.com', '(31) 99876-5432', '345.678.901-23', 'Rua das Acácias, 234', 'Belo Horizonte'),
    ('Daniela Lima', 'daniela.lima@email.com', '(41) 91234-8765', '456.789.012-34', 'Av. Paraná, 321', 'Curitiba'),
    ('Eduardo Silva', 'eduardo.silva@email.com', '(61) 92345-6789', '567.890.123-45', 'Rua Brasília, 120', 'Brasília'),
    ('Fernanda Costa', 'fernanda.costa@email.com', '(71) 98712-3456', '678.901.234-56', 'Rua do Campo, 456', 'Salvador'),
    ('Gabriel Santos', 'gabriel.santos@email.com', '(81) 92134-5678', '789.012.345-67', 'Av. Recife, 987', 'Recife'),
    ('Helena Martins', 'helena.martins@email.com', '(85) 91122-3344', '890.123.456-78', 'Rua das Palmeiras, 200', 'Fortaleza'),
    ('Isabel Ribeiro', 'isabel.ribeiro@email.com', '(51) 97865-4321', '901.234.567-89', 'Av. Ipiranga, 654', 'Porto Alegre'),
    ('João Carvalho', 'joao.carvalho@email.com', '(19) 93456-7890', '012.345.678-90', 'Rua Campinas, 300', 'Campinas');
```

Com este bloco, são cadastrados dez clientes, incluindo informações como nome, e-mail, telefone, CPF e endereço. Esses dados são fundamentais para identificar e contatar os clientes durante o processo de venda e entrega de pedidos.

Código referente às inserções na tabela ‘Funcionarios’:

```
INSERT INTO funcionarios (nome, cargo, salario, data_contratacao, email, telefone)
VALUES
    ('Carlos Almeida', 'Vendedor', 2500.00, '2022-04-10', 'carlos.almeida@empresa.com', '(11) 91234-5678'),
    ('Mariana Oliveira', 'Gerente', 4500.00, '2021-02-20', 'mariana.oliveira@empresa.com', '(21) 98765-4321'),
    ('Lucas Pereira', 'Caixa', 1800.00, '2023-07-15', 'lucas.pereira@empresa.com', '(31) 99876-5432'),
    ('Fernanda Costa', 'Consultora de Vendas', 2700.00, '2020-11-12', 'fernanda.costa@empresa.com', '(41) 91234-8765'),
    ('Rafael Silva', 'Assistente de Estoque', 2000.00, '2023-03-05', 'rafael.silva@empresa.com', '(61) 92345-6789');
```

O bloco insere informações de cinco funcionários, como nome, cargo, salário, data de contratação, e dados de contato. Esse cadastro auxilia na organização da equipe e permite o controle de funções, salários e histórico de contratação.

Código referente às inserções na tabela ‘Pedidos’:

```
INSERT INTO pedidos (data_pedido, cliente_id, funcionario_id, status_pedido)
VALUES
    ('2024-10-01', 1, 1, 'Concluído'),
    ('2024-10-02', 2, 1, 'Em andamento'),
    ('2024-10-03', 3, 2, 'Concluído'),
    ('2024-10-04', 4, 2, 'Concluído'),
    ('2024-10-05', 5, 3, 'Em andamento'),
    ('2024-10-06', 6, 1, 'Concluído'),
    ('2024-10-07', 7, 4, 'Em andamento'),
    ('2024-10-08', 8, 4, 'Concluído'),
    ('2024-10-09', 9, 5, 'Em andamento'),
    ('2024-10-10', 10, 3, 'Concluído');
```

Este bloco insere pedidos, especificando a data, cliente, funcionário responsável e o status do pedido. Essas informações são cruciais para o gerenciamento de vendas, ajudando a acompanhar o progresso e status de cada transação.

Código referente às inserções na tabela ‘Itens-Pedido’:

```
INSERT INTO itens_pedido (pedido_id, produto_id, quantidade, preco_unitario)
VALUES
    (1, 1, 2, 1500.00), # Pedido 1
    (1, 4, 1, 2800.00),
    (2, 6, 1, 300.00), # Pedido 2
    (2, 7, 1, 250.00),
    (3, 3, 3, 150.00), # Pedido 3
    (3, 10, 1, 600.00),
    (4, 5, 1, 400.00), # Pedido 4
    (4, 9, 2, 70.00),
    (5, 11, 1, 1200.00), # Pedido 5
    (5, 15, 1, 500.00),
    (6, 13, 2, 200.00), # Pedido 6
    (6, 19, 1, 750.00),
    (7, 17, 1, 120.00), # Pedido 7
    (7, 8, 1, 350.00),
    (8, 14, 1, 300.00), # Pedido 8
    (8, 2, 1, 3200.00),
    (9, 16, 1, 150.00), # Pedido 9
    (9, 20, 1, 2800.00),
    (10, 18, 1, 150.00), # Pedido 10
    (10, 12, 1, 80.00);
```

Aqui são inseridos os itens de cada pedido, incluindo o produto, quantidade e preço unitário. Esse detalhamento permite que o sistema registre os produtos específicos em cada venda, calculando o valor total dos pedidos e controlando o inventário.

Código referente às inserções na tabela ‘Pagamentos’:

```
INSERT INTO pagamentos (pedido_id, valor, metodo_pagamento, data_pagamento)
VALUES
    (1, 5800.00, 'Cartão de Crédito', '2024-10-02'), # Pedido concluído
    (2, 550.00, 'Boleto', '2024-10-03'), # Pedido em andamento, pagamento pendente
    (3, 1050.00, 'Transferência Bancária', '2024-10-04'), # Pedido concluído
    (4, 540.00, 'Cartão de Crédito', '2024-10-05'), # Pedido concluído
    (5, 1700.00, 'Cartão de Crédito', '2024-10-06'), # Pedido em andamento, pagamento pendente
    (6, 1150.00, 'Boleto', '2024-10-07'), # Pedido concluído
    (7, 470.00, 'Transferência Bancária', '2024-10-08'), # Pedido em andamento, pagamento pendente
    (8, 3500.00, 'Cartão de Crédito', '2024-10-09'), # Pedido concluído
    (9, 2950.00, 'Boleto', '2024-10-10'), # Pedido em andamento, pagamento pendente
    (10, 230.00, 'Transferência Bancária', '2024-10-11'); # Pedido concluído
```

Neste bloco, são inseridos os detalhes dos pagamentos, incluindo o valor, método de pagamento e data. Essas informações completam o ciclo de vendas, associando cada pedido a um pagamento, o que permite acompanhar o fluxo financeiro e o status dos pagamentos de clientes.

1. ATUALIZAÇÃO DE PREÇOS DE PRODUTOS

```
UPDATE produtos SET preco = preco * 1.1
WHERE categoria_id = (
    SELECT categoria_id FROM categorias
    WHERE nome = 'Tecnologia e Eletrônicos'
);
```

Aqui, o novo preço é calculado multiplicando o valor atual por 1.1, o que representa um aumento de 10%. A cláusula WHERE especifica uma condição que limita a atualização apenas aos produtos que pertencem a uma determinada categoria. A subconsulta busca o 'categoria_id' correspondente à categoria "Tecnologia e Eletrônicos". Esse valor é então usado na condição WHERE, aplicando a atualização somente aos produtos que pertencem a essa categoria.

produto_id	nome	descricao	preco
1	Smartphone X10	NULL	1500.00
2	Notebook Ultrafino	NULL	3200.00
20	Smart TV 50" 4K	NULL	2800.00

Figura 1 – Preço dos produtos da categoria 'Tecnologia e Eletrônicos' antes da atualização

produto_id	nome	descricao	preco
1	Smartphone X10	NULL	1650.00
2	Notebook Ultrafino	NULL	3520.00
20	Smart TV 50" 4K	NULL	3080.00

Figura 2 – Preço dos produtos da categoria 'Tecnologia e Eletrônicos' depois da atualização

2. DESCONTO EM PRODUTOS COM ESTOQUE ALTO

```
UPDATE produtos SET preco = preco * 0.95
WHERE estoque > 20;
```

Aqui utilizei a instrução UPDATE, reduzindo o preço ao multiplicá-lo por 0.95, representando um desconto de 5%. A cláusula WHERE aplica essa atualização apenas aos produtos com um estoque superior a 20 unidades.

produto_id	nome	preco	estoque
1	Smartphone X10	1650.00	25
5	Cafeteira Express	400.00	30
8	Jaqueta Impermeável	350.00	50
12	Mouse sem fio	80.00	45
14	Caixa de Som Bluetooth	300.00	27
17	Relógio Digital	120.00	40
19	Aspirador de Pó PowerClean	750.00	21

Figura 3 – Preço dos produtos com mais de 20 itens em estoque antes da atualização

produto_id	nome	preco	estoque
1	Smartphone X10	1567.50	25
5	Cafeteira Express	380.00	30
8	Jaqueta Impermeável	332.50	50
12	Mouse sem fio	76.00	45
14	Caixa de Som Bluetooth	285.00	27
17	Relógio Digital	114.00	40
19	Aspirador de Pó PowerClean	712.50	21

Figura 4 – Preço dos produtos com mais de 20 itens em estoque depois da atualização

3. PROMOÇÃO EM FORNECEDORES SELECIONADOS

```
UPDATE produtos SET preco = preco * 0.85
WHERE fornecedor_id = (
    SELECT fornecedor_id FROM fornecedores
    WHERE nome = 'Tech World'
);
```

Nessa etapa, utilizando UPDATE, atualizei a coluna ‘preco’ na tabela ‘produtos’, multiplicando o preço atual por 0,85. Isso equivale a um desconto de 15%, reduzindo o valor de cada produto aplicável. A cláusula WHERE limita a atualização aos produtos com um ‘fornecedor_id’ específico. Esta subconsulta obtém o ‘fornecedor_id’ da "Tech World" a partir da tabela ‘fornecedores’. Esse ID é usado como critério para filtrar os produtos fornecidos por essa empresa.

produto_id	nome	preco
1	Smartphone X10	1567.50
3	Fone de Ouvido Bass+	150.00
12	Mouse sem fio	76.00
14	Caixa de Som Bluetooth	285.00
17	Relógio Digital	114.00

Figura 5 – Preço dos produtos do fornecedor Tech World antes da atualização

produto_id	nome	preco
1	Smartphone X10	1332.38
3	Fone de Ouvido Bass+	127.50
12	Mouse sem fio	64.60
14	Caixa de Som Bluetooth	242.25
17	Relógio Digital	96.90

Figura 6 – Preço dos produtos do fornecedor Tech World depois da atualização

4. ATUALIZAÇÃO DE DADOS DE CLIENTES

```
UPDATE clientes SET
    telefone = '(61) 1234-5678',
    endereco = 'Rua Paraná, 51'
WHERE cliente_id = 5;
```

A instrução começa especificando a tabela ‘clientes’, onde as informações serão atualizadas, então os novos valores para o ‘endereco’ e o ‘telefone’ do cliente são definidos. A cláusula WHERE garante que a atualização se aplique apenas ao cliente específico com o cliente_id igual a 5.

cliente_id	nome	telefone	endereco
5	Eduardo Silva	(61) 92345-6789	Rua Brasília, 120

Figura 7 – Telefone e endereço do cliente de ID igual a 5 (Eduardo Silva) antes da atualização

cliente_id	nome	telefone	endereco
5	Eduardo Silva	(61) 1234-5678	Rua Paraná, 51

Figura 8 – Telefone e endereço do cliente de ID igual a 5 (Eduardo Silva) após a atualização

5. PROMOÇÃO DE FUNCIONÁRIOS

```
UPDATE funcionarios SET
    cargo = 'Gerente',
    salario = salario * 1.20
WHERE funcionario_id = 3;
```

A instrução começa especificando a tabela ‘funcionarios’ onde as informações serão atualizadas, então os novos valores para o ‘cargo’ e o ‘salario’ do funcionário serão inseridos. O ‘salario’ atual será multiplicado por 1,20, o que resulta em um aumento de 20%. A cláusula WHERE filtra os registros que serão atualizados, garantindo que as mudanças afetem apenas o funcionário com ‘funcionario_id’ = 3.

funcionario_id	nome	cargo	salario
3	Lucas Pereira	Caixa	1800.00

Figura 9 – Cargo e salário do cliente de ID igual a 3 (Lucas Pereira) antes da atualização

funcionario_id	nome	cargo	salario
3	Lucas Pereira	Gerente	2160.00

Figura 10 – Cargo e salário do cliente de ID igual a 3 (Lucas Pereira) depois da atualização

6. ATUALIZAÇÃO DO STATUS DE PEDIDOS

```
UPDATE pedidos
SET status_pedido = 'Concluído'
WHERE EXISTS (
    SELECT 1
    FROM pagamentos
    WHERE pagamentos.pedido_id = pedidos.pedido_id
);
```

A instrução inicia atualizando a tabela ‘pedidos’. Depois é definido o novo valor do campo ‘status_pedido’ como 'Concluído' para os pedidos que atendem à condição. Para consultar se um pedido atende a condição, usamos uma subconsulta para verificar se existe um pagamento associado ao pedido. Se a condição for verdadeira para um pedido (ou seja, se o pedido tiver ao menos um pagamento), o status desse pedido será atualizado para "Concluído". A subconsulta busca na tabela pagamentos os registros onde ‘pedido_id’ coincide com ‘pedido_id’ na tabela pedidos. A subconsulta retorna um valor se existir um pagamento para o pedido, fazendo com que a condição EXISTS seja verdadeira.

pedido_id	status_pedido	valor	metodo_pagamento	data_pagamento
1	Concluído	5800.00	Cartão de Crédito	2024-10-02
2	Em andamento	550.00	Boleto	2024-10-03
3	Concluído	1050.00	Transferência Bancária	2024-10-04
4	Concluído	540.00	Cartão de Crédito	2024-10-05
5	Em andamento	1700.00	Cartão de Crédito	2024-10-06
6	Concluído	1150.00	Boleto	2024-10-07
7	Em andamento	470.00	Transferência Bancária	2024-10-08
8	Concluído	3500.00	Cartão de Crédito	2024-10-09
9	Em andamento	2950.00	Boleto	2024-10-10
10	Concluído	230.00	Transferência Bancária	2024-10-11

Figura 11 - Status dos pedidos que possuem algum pagamento antes da atualização

pedido_id	status_pedido	valor	metodo_pagamento	data_pagamento
1	Concluído	5800.00	Cartão de Crédito	2024-10-02
2	Concluído	550.00	Boleto	2024-10-03
3	Concluído	1050.00	Transferência Bancária	2024-10-04
4	Concluído	540.00	Cartão de Crédito	2024-10-05
5	Concluído	1700.00	Cartão de Crédito	2024-10-06
6	Concluído	1150.00	Boleto	2024-10-07
7	Concluído	470.00	Transferência Bancária	2024-10-08
8	Concluído	3500.00	Cartão de Crédito	2024-10-09
9	Concluído	2950.00	Boleto	2024-10-10
10	Concluído	230.00	Transferência Bancária	2024-10-11

Figura 12 - Status dos pedidos que possuem algum pagamento após a atualização

7. REAJUSTE DE PREÇOS PÓS-PROMOÇÃO

```sql
ALTER TABLE produtos ADD COLUMN preco_original DECIMAL(10, 2);

UPDATE produtos
SET preco_original = CASE produto_id
    WHEN 1 THEN 1500.00
    WHEN 2 THEN 3200.00
    WHEN 3 THEN 150.00
    WHEN 4 THEN 2800.00
    WHEN 5 THEN 400.00
    WHEN 6 THEN 300.00
    WHEN 7 THEN 250.00
    WHEN 8 THEN 350.00
    WHEN 9 THEN 70.00
    WHEN 10 THEN 600.00
    WHEN 11 THEN 1200.00
    WHEN 12 THEN 80.00
    WHEN 13 THEN 200.00
    WHEN 14 THEN 300.00
    WHEN 15 THEN 500.00
    WHEN 16 THEN 150.00
    WHEN 17 THEN 120.00
    WHEN 18 THEN 150.00
    WHEN 19 THEN 750.00
    WHEN 20 THEN 2800.00
END
WHERE produto_id IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20);

UPDATE produtos SET preco_original = preco;
```

Na primeira sessão de código adicionei uma coluna nova na tabela ‘produtos’ chamada ‘preco_original’, nela armazenei o valor inicial dos produtos, ou seja, antes das promoções serem aplicadas. Depois utilizei o comando UPDATE com a expressão CASE para restaurar manualmente o preço original de produtos na tabela produtos. Ele verifica o ‘produto_id’ de cada item e atribui o preço correspondente ao valor inicial. A cláusula WHERE ‘produto_id’ IN (...) garante que apenas os produtos especificados tenham seu preço atualizado, mantendo outros registros da tabela inalterados. Isso é útil para restaurar preços originais após ajustes temporários de promoção ou desconto.

produto_id	nome	preco	produto_id	nome	preco
1	Smartphone X10	1332.38	11	Cadeira Gamer	1200.00
2	Notebook Ultrafino	3520.00	12	Mouse sem fio	64.60
3	Fone de Ouvido Bass+	127.50	13	Teclado Mecânico Pro	200.00
4	Geladeira FrostMax	2800.00	14	Caixa de Som Bluetooth	242.25
5	Cafeteira Express	380.00	15	Micro-ondas Compact	500.00
6	Air Fryer HealthPlus	300.00	16	Liquidificador Turbo	150.00
7	Tênis Confort 2024	250.00	17	Relógio Digital	96.90
8	Jaqueta Impermeável	332.50	18	Cadeira Dobrável	150.00
9	Camiseta Eco	70.00	19	Aspirador de Pó PowerClean	712.50
10	Mesa de Escritório Compact	600.00	20	Smart TV 50" 4K	3080.00

Figura 13 – Preço dos produtos antes de restaurar ao valor original

produto_id	nome	preco
1	Smartphone X10	1500.00
2	Notebook Ultrafino	3200.00
3	Fone de Ouvido Bass+	150.00
4	Geladeira FrostMax	2800.00
5	Cafeteira Express	400.00
6	Air Fryer HealthPlus	300.00
7	Tênis Confort 2024	250.00
8	Jaqueta Impermeável	350.00
9	Camiseta Eco	70.00
10	Mesa de Escritório Compact	600.00
11	Cadeira Gamer	1200.00
12	Mouse sem fio	80.00
13	Teclado Mecânico Pro	200.00
14	Caixa de Som Bluetooth	300.00
15	Micro-ondas Compact	500.00
16	Liquidificador Turbo	150.00
17	Relógio Digital	120.00
18	Cadeira Dobrável	150.00
19	Aspirador de Pó PowerClean	750.00
20	Smart TV 50" 4K	2800.00

Figura 14 – Preço dos produtos após restaurar ao valor original

8. ATUALIZAÇÃO DE ESTOQUE APÓS VENDAS

```
UPDATE produtos p
JOIN itens_pedido ip ON p.produto_id = ip.produto_id
JOIN pedidos ped ON ip.pedido_id = ped.pedido_id
SET p.estoque = p.estoque - ip.quantidade
WHERE ped.status_pedido = 'Concluído';
```

Nessa etapa, utilizei o comando JOIN entre ‘produtos’, ‘itens_pedido’ e ‘pedidos’ para vincular cada produto ao pedido correspondente. A linha SET ‘p.estoque’ = ‘p.estoque’ – ‘ip.quantidade’ diminui o estoque do produto pela quantidade comprada. A cláusula WHERE ‘ped.status_pedido’ = 'Concluído' garante que o estoque seja reduzido apenas para pedidos com o status "Concluído".

pedido_id	status_pedido	produto_id	quantidade	produto_nome	estoque
1	Concluído	1	2	Smartphone X10	25
1	Concluído	4	1	Geladeira FrostMax	5
2	Concluído	6	1	Air Fryer HealthPlus	4
2	Concluído	7	1	Tênis Confort 2024	2
3	Concluído	3	3	Fone de Ouvido Bass+	18
3	Concluído	10	1	Mesa de Escritório Compact	4
4	Concluído	5	1	Cafeteira Express	30
4	Concluído	9	2	Camiseta Eco	15
5	Concluído	11	1	Cadeira Gamer	10
5	Concluído	15	1	Micro-ondas Compact	12
6	Concluído	13	2	Teclado Mecânico Pro	8
6	Concluído	19	1	Aspirador de Pó PowerClean	21
7	Concluído	17	1	Relógio Digital	40
7	Concluído	8	1	Jaqueta Impermeável	50
8	Concluído	14	1	Caixa de Som Bluetooth	27
8	Concluído	2	1	Notebook Ultrafino	3
9	Concluído	16	1	Liquidificador Turbo	6
9	Concluído	20	1	Smart TV 50" 4K	7
10	Concluído	18	1	Cadeira Dobrável	3
10	Concluído	12	1	Mouse sem fio	45

Figura 15 – Estoque dos produtos antes das vendas

pedido_id	status_pedido	produto_id	quantidade	produto_nome	estoque
1	Concluído	1	2	Smartphone X10	23
1	Concluído	4	1	Geladeira FrostMax	4
2	Concluído	6	1	Air Fryer HealthPlus	3
2	Concluído	7	1	Tênis Confort 2024	1
3	Concluído	3	3	Fone de Ouvido Bass+	15
3	Concluído	10	1	Mesa de Escritório Compact	3
4	Concluído	5	1	Cafeteira Express	29
4	Concluído	9	2	Camiseta Eco	13
5	Concluído	11	1	Cadeira Gamer	9
5	Concluído	15	1	Micro-ondas Compact	11
6	Concluído	13	2	Teclado Mecânico Pro	6
6	Concluído	19	1	Aspirador de Pó PowerClean	20
7	Concluído	17	1	Relógio Digital	39
7	Concluído	8	1	Jaqueta Impermeável	49
8	Concluído	14	1	Caixa de Som Bluetooth	26
8	Concluído	2	1	Notebook Ultrafino	2
9	Concluído	16	1	Liquidificador Turbo	5
9	Concluído	20	1	Smart TV 50" 4K	6
10	Concluído	18	1	Cadeira Dobrável	2
10	Concluído	12	1	Mouse sem fio	44

Figura 16 – Estoque dos produtos após as vendas

9. CORREÇÃO DE INFORMAÇÕES DE FORNECEDORES

```sql
UPDATE fornecedores SET
    endereco = 'Rua dos Digitais, 123',
    contato = 'sac@digitalstore.com.br'
WHERE fornecedor_id = 2;
```

O código atualiza as informações de um fornecedor na tabela fornecedores. Ele altera o endereço para 'Rua dos Digitais, 123' e o contato para 'sac@digitalstore.com.br' apenas para o fornecedor com ‘fornecedor_id’ = 2. A cláusula WHERE garante que somente esse registro específico seja modificado. Esse comando é usado para manter os dados do banco atualizados e consistentes com mudanças reais.

fornecedor_id	nome	contato	endereco	cidade
2	Digital Store	contato@digitalstore.com.br	Rua dos Eletrônicos, 567	Rio de Janeiro

Figura 17 – Contato e endereço do fornecedor “Digital Store” antes da atualização

fornecedor_id	nome	contato	endereco	cidade
2	Digital Store	sac@digitalstore.com.br	Rua dos Digitais, 123	Rio de Janeiro

Figura 18 – Contato e endereço do fornecedor “Digital Store” após a atualização

10. ATUALIZAÇÃO DE QUANTIDADES EM ITENS DE PEDIDOS

```
UPDATE itens_pedido SET
    quantidade = 3
WHERE item_pedido_id = 1
```

Este comando altera a quantidade de produtos comprados (associados a um pedido) no item identificado pelo 'item_pedido_id' = 1, atualizando a quantidade para 3. O uso da cláusula WHERE é essencial para garantir que apenas o registro correto seja modificado.

item_pedido_id	pedido_id	produto_nome	quantidade	preco_unitario
1	1	Smartphone X10	2	1500.00

Figura 19 – Quantidade do produto de ID = 1 ("Smartphone X10") antes da atualização

item_pedido_id	pedido_id	produto_nome	quantidade	preco_unitario
1	1	Smartphone X10	3	1500.00

Figura 20 – Quantidade do produto de ID = 1 ("Smartphone X10") após atualização

3. CONSULTAS SQL AVANÇADAS

1. PRODUTOS POR CATEGORIA E FORNECEDOR

```sql
SELECT
    p.nome AS produto,
    c.nome AS categoria,
    f.nome AS fornecedor,
    p.estoque
FROM
    produtos p
JOIN
    categorias c ON p.categoria_id = c.categoria_id
JOIN
    fornecedores f ON p.fornecedor_id = f.fornecedor_id
WHERE
    p.estoque > 10;
```

Para realizar a consulta iniciei o bloco com o comando SELECT para definir as colunas que serão exibidas. Os apelidos (AS) tornam o nome mais legível. A parte 'FROM produtos p' define a tabela principal (produtos) com um apelido p. Após definir a tabela principal, utilizei o comando 'JOIN categorias c', que relaciona a tabela produtos com categorias usando a chave estrangeira 'categoria_id'. Em seguida, o código 'JOIN fornecedores f' relaciona a tabela produtos com fornecedores usando a chave estrangeira 'fornecedor_id'. Por fim, o código 'WHERE p.estoque > 10' filtra apenas os produtos com estoque superior a 10 unidades.

produto	categoria	fornecedor	estoque
Smartphone X10	Tecnologia e Eletrônicos	Tech World	23
Fone de Ouvido Bass+	Acessórios de Tecnologia	Tech World	15
Cafeteira Express	Utensílios para Cozinha	KitchenPlus	29
Jaqueta Impermeável	Moda e Vestuário	HomeDeco	49
Camiseta Eco	Moda e Vestuário	HomeDeco	13
Mouse sem fio	Acessórios de Tecnologia	Tech World	44
Caixa de Som Bluetooth	Acessórios de Tecnologia	Tech World	26
Micro-ondas Compact	Eletrodomésticos	KitchenPlus	11
Relógio Digital	Acessórios de Tecnologia	Tech World	39
Aspirador de Pó PowerClean	Eletrodomésticos	FrioTech	20

Figura 21 – Resultado da consulta

2. PEDIDOS DE CLIENTES

```sql
SELECT
    cl.nome AS cliente,
    p.data_pedido,
    f.nome AS funcionario_responsavel,
    p.status_pedido
FROM
    pedidos p
JOIN
    clientes cl ON p.cliente_id = cl.cliente_id
JOIN
    funcionarios f ON p.funcionario_id = f.funcionario_id
WHERE
    cl.cliente_id = 1 AND
    p.data_pedido >= CURDATE() - INTERVAL 60 DAY;
```

O código busca listar os pedidos realizados por um cliente específico nos últimos 60 dias, mostrando o nome do cliente, a data do pedido, o funcionário responsável e o status do pedido. Ele utiliza as tabelas ‘pedidos’, ‘clientes’, e ‘funcionarios’, conectando-as por meio de chaves estrangeiras. O filtro WHERE garante que apenas os pedidos do cliente indicado e realizados no intervalo de 60 dias sejam exibidos. A consulta utiliza JOIN para relacionar os dados entre as tabelas e retorna informações organizadas em colunas específicas para facilitar a análise. **(Obs.: Para fins de consulta, Utilizei um intervalo de 60 dias ao invés de 30 para retornar algum valor)**

cliente	data_pedido	funcionario_responsavel	status_pedido
Ana Souza	2024-10-01	Carlos Almeida	Concluído

Figura 22 – Resultado da consulta

3. TOTAL DE VENDAS POR PRODUTO

```sql
SELECT
    p.nome AS produto_nome,
    SUM(ip.quantidade) AS total_vendido
FROM
    itens_pedido ip
JOIN
    produtos p USING (produto_id)
GROUP BY
    p.nome
HAVING
    total_vendido > 1;
```

A consulta calcula o total vendido de cada produto com vendas superiores a 1 unidades. Ela usa ‘SUM(ip.quantidade)’ para somar as quantidades vendidas por produto e ‘p.nome’ para exibir o nome do produto. As tabelas ‘itens_pedido’ (com alias ip) e ‘produtos’ (com alias p) são unidas pela coluna ‘produto_id’. A condição ‘HAVING SUM(ip.quantidade) > 1’ filtra os produtos com vendas acima de 1 unidades, e a consulta agrupa os resultados por ‘p.nome’ para calcular o total de cada produto. . **(Obs.: Utilizei um o valor 1 ao invés de 5 para poder exibir resultados mais fiéis)**

produto_nome	total_vendido
Smartphone X10	3
Fone de Ouvido Bass+	3
Camiseta Eco	2
Teclado Mecânico Pro	2

Figura 23 – Resultado da consulta

4. PAGAMENTOS POR PEDIDO

```sql
SELECT
    p.pedido_id,
    p.valor AS valor_pagamento,
    p.metodo_pagamento,
    p.data_pagamento
FROM
    pagamentos p
WHERE
    p.data_pagamento >= CURDATE() - INTERVAL 60 DAY;
```

O código consulta a tabela pagamentos para exibir os pagamentos realizados nos últimos 60 dias, retornando o 'pedido_id', o valor pago (renomeado como 'valor_pagamento'), o método de pagamento e a data de pagamento. A condição 'p.data_pagamento >= CURDATE() - INTERVAL 60 DAY' filtra os registros com base no intervalo de tempo desejado.

pedido_id	valor_pagamento	metodo_pagamento	data_pagamento
1	5800.00	Cartão de Crédito	2024-10-02
2	550.00	Boleto	2024-10-03
3	1050.00	Transferência Bancária	2024-10-04
4	540.00	Cartão de Crédito	2024-10-05
5	1700.00	Cartão de Crédito	2024-10-06
6	1150.00	Boleto	2024-10-07
7	470.00	Transferência Bancária	2024-10-08
8	3500.00	Cartão de Crédito	2024-10-09
9	2950.00	Boleto	2024-10-10
10	230.00	Transferência Bancária	2024-10-11

Figura 24 – Resultado da consulta

5. PEDIDOS PENDENTES

```
UPDATE pedidos SET status_pedido = 'Em andamento'
WHERE
    pedido_id % 2 = 0;
```

Para fins de consulta, por meio da cláusula UPDATE alterei o ‘status_pedido’ para “Em andamento” para os ‘pedido_id’ de número par.

```
SELECT
    p.pedido_id,
    p.data_pedido,
    p.status_pedido,
    SUM(ip.quantidade * ip.preco_unitario) AS valor_total
FROM pedidos p
LEFT JOIN itens_pedido ip ON p.pedido_id = ip.pedido_id
LEFT JOIN pagamentos pg ON p.pedido_id = pg.pedido_id
WHERE (pg.pedido_id IS NULL OR p.status_pedido = 'Em andamento')
GROUP BY p.pedido_id, p.data_pedido, p.status_pedido
HAVING SUM(ip.quantidade * ip.preco_unitario) > 100
ORDER BY p.data_pedido;
```

A consulta retorna pedidos cujo pagamento não foi registrado (‘pg.pedido_id IS NULL’) ou cujo status é "Em andamento". Utilizei o comando LEFT JOIN para combinar as tabelas de pedidos, itens do pedido e pagamentos, calculando o valor total do pedido com ‘SUM(ip.quantidade * ip.preco_unitario)’. Apenas pedidos com valor total acima de R$100 são exibidos devido à cláusula HAVING. Os resultados são agrupados por ID, data e status do pedido, e ordenados pela data do pedido.

pedido_id	data_pedido	status_pedido	valor_total
2	2024-10-02	Em andamento	550.00
4	2024-10-04	Em andamento	540.00
6	2024-10-06	Em andamento	1150.00
8	2024-10-08	Em andamento	3500.00
10	2024-10-10	Em andamento	230.00

Figura 25 – Resultado da consulta

6. RELATÓRIO DE ESTOQUE

```sql
SELECT
    p.nome AS nome_produto,
    p.estoque AS quantidade_estoque,
    p.preco AS preco,
    f.nome AS nome_fornecedor
FROM produtos p
INNER JOIN fornecedores f ON p.fornecedor_id = f.fornecedor_id
WHERE p.estoque < 5
ORDER BY p.nome;
```

A consulta seleciona o nome dos produtos, a quantidade em estoque, o preço e o nome do fornecedor, unindo as tabelas produtos e fornecedores através do 'fornecedor_id'. Depois apliquei um filtro para exibir apenas produtos com menos de 5 unidades em estoque (WHERE estoque < 5). A ordenação é feita pelo nome do produto em ordem alfabética (ORDER BY p.nome). A INNER JOIN garante que apenas produtos com fornecedores registrados sejam exibidos.

nome_produto	quantidade_estoque	preco	nome_fornecedor
Air Fryer HealthPlus	3	300.00	KitchenPlus
Cadeira Dobrável	2	150.00	HomeDeco
Geladeira FrostMax	4	2800.00	FrioTech
Mesa de Escritório Compact	3	600.00	HomeDeco
Notebook Ultrafino	2	3200.00	Digital Store
Tênis Confort 2024	1	250.00	HomeDeco

Figura 26 – Resultado da consulta

7. TOTAL DE VENDAS POR CLIENTE

```sql
SELECT
    c.nome AS cliente_nome,
    SUM(ip.quantidade * ip.preco_unitario) AS total_gasto
FROM clientes c
INNER JOIN pedidos p ON c.cliente_id = p.cliente_id
INNER JOIN itens_pedido ip ON p.pedido_id = ip.pedido_id
GROUP BY c.cliente_id, c.nome
HAVING SUM(ip.quantidade * ip.preco_unitario) > 500;
```

A consulta une as tabelas ‘clientes’, ‘pedidos’ e ‘itens_pedido’ para calcular o valor total gasto por cliente com base na soma de ‘quantidade * preço_unitário’. A cláusula GROUP BY agrupa os dados por cliente, e a cláusula HAVING filtra clientes cujo total gasto excede R$ 500. O resultado inclui o nome do cliente e o total gasto, ordenado em ordem decrescente de valor. A INNER JOIN garante que apenas clientes com pedidos associados sejam exibidos.

cliente_nome	total_gasto
Ana Souza	7300.00
Bruno Oliveira	550.00
Carla Mendes	1050.00
Daniela Lima	540.00
Eduardo Silva	1700.00
Fernanda Costa	1150.00
Helena Martins	3500.00
Isabel Ribeiro	2950.00

Figura 27 – Resultado da consulta

8. FUNCIONÁRIOS COM VENDAS ACIMA DA MÉDIA

```sql
SELECT
    f.nome AS nome_funcionario,
    SUM(ip.quantidade * ip.preco_unitario) AS total_vendas
FROM funcionarios f
INNER JOIN pedidos p ON f.funcionario_id = p.funcionario_id
INNER JOIN itens_pedido ip ON p.pedido_id = ip.pedido_id
GROUP BY f.nome
HAVING SUM(ip.quantidade * ip.preco_unitario) > (
    SELECT AVG(total_vendas) FROM (
        SELECT SUM(ip.quantidade * ip.preco_unitario) AS total_vendas
        FROM pedidos p
        INNER JOIN itens_pedido ip ON p.pedido_id = ip.pedido_id
        GROUP BY p.funcionario_id
    ) AS subquery
);
```

Essa consulta exibe os funcionários que realizaram vendas acima da média total de vendas. Primeiro, ela calcula o total de vendas por funcionário (usando SUM e agrupando por 'f.nome'). Em seguida, uma subconsulta determina a média dos totais de vendas por funcionário. A cláusula HAVING filtra apenas os funcionários cujo total de vendas excede essa média. Os resultados mostram o nome do funcionário e o total de vendas realizadas. A subconsulta calcula os totais de vendas por funcionário agrupados com base nos pedidos.

nome_funcionario	total_vendas
Carlos Almeida	9000.00
Mariana Oliveira	1590.00
Lucas Pereira	1930.00
Fernanda Costa	3970.00
Rafael Silva	2950.00

Figura 28 – Resultado da consulta de todos os funcionários e suas vendas

nome_funcionario	total_vendas
Carlos Almeida	9000.00
Fernanda Costa	3970.00

Figura 29 – Resultado da consulta dos funcionários acima da média

9. ANÁLISE DE VENDAS MENSAIS

```sql
SELECT
    YEAR(p.data_pedido) AS ano,
    MONTH(p.data_pedido) AS mes,
    SUM(ip.quantidade * ip.preco_unitario) AS total_vendas
FROM
    pedidos p
INNER JOIN itens_pedido ip ON p.pedido_id = ip.pedido_id
GROUP BY
    YEAR(p.data_pedido), MONTH(p.data_pedido)
HAVING
    SUM(ip.quantidade * ip.preco_unitario) > 1000
ORDER BY
    ano, mes;
```

Essa consulta exibe as vendas totais agrupadas por ano e mês para pedidos cujo total de vendas excede R$ 1000. A função YEAR extrai o ano e MONTH extrai o mês da data do pedido. O total de vendas é calculado como o somatório de ‘quantidade * preco_unitario’. A cláusula GROUP BY organiza os resultados por ano e mês, e a cláusula HAVING filtra os meses com vendas acima de R$1000. Por fim, a consulta ordena os resultados por ano e mês. Como todos os pedidos foram realizados em outubro, foi o único mês que apareceu na consulta.

ano	mes	total_vendas
2024	10	19440.00

Figura 30 – Resultado da consulta

10. PRODUTOS COM BAIXA ROTATIVIDADE

```sql
SELECT
    p.nome AS nome_produto,
    c.nome AS categoria
FROM
    produtos p
INNER JOIN categorias c ON p.categoria_id = c.categoria_id
WHERE
    p.produto_id NOT IN (
        SELECT DISTINCT ip.produto_id
        FROM itens_pedido ip
        INNER JOIN pedidos pe ON ip.pedido_id = pe.pedido_id
        WHERE pe.data_pedido >= CURDATE() - INTERVAL 45 DAY
    )
ORDER BY
    p.nome;
```

A consulta identifica produtos que não foram vendidos nos últimos 45 dias, exibindo o nome do produto e sua categoria. A subconsulta encontra os produtos vendidos nesse período verificando ‘data_pedido’ na tabela pedidos. Em seguida, a cláusula NOT IN filtra os produtos cujo ‘produto_id’ não aparece na lista de produtos vendidos. A consulta principal associa cada produto à sua categoria usando a tabela categorias e organiza o resultado pelo nome do produto. Isso facilita identificar produtos encalhados no estoque. **(Obs.: Para fins de consulta, utilizei um intervalo de 45 dias para retornar algum valor)**

nome_produto	categoria
Air Fryer HealthPlus	Eletrodomésticos
Geladeira FrostMax	Eletrodomésticos
Smartphone X10	Tecnologia e Eletrônicos
Tênis Confort 2024	Moda e Vestuário

Figura 31 – Resultado da consulta com o intervalo de 45 dias

nome_produto	categoria

Figura 32 – Resultado da consulta com o intervalo de 90 dias

11. RELATÓRIO DE FUNCIONÁRIOS POR CARGO

```sql
SELECT
    f.cargo,
    COUNT(f.funcionario_id) AS quantidade_funcionarios
FROM
    funcionarios f
GROUP BY
    f.cargo
HAVING
    COUNT(f.funcionario_id) > 0;
```

O código agrupa os funcionários por cargo e conta quantos existem em cada um, exibindo apenas os cargos com mais de dois funcionários. Primeiro, seleciona os cargos e usa a função `COUNT` para contar os IDs de funcionários. Em seguida, o agrupamento com `GROUP BY` organiza os dados por cargo. Por fim, o filtro `HAVING` é aplicado para mostrar apenas os cargos onde a contagem é maior que dois. Assim, a consulta retorna uma lista resumida de cargos com sua respectiva quantidade de funcionários, obedecendo ao critério definido. **(Obs.: Para fins de consulta, utilizei um COUNT > 0 para retornar algum valor)**

cargo	quantidade_funcionarios
Vendedor	1
Gerente	2
Consultora de Vendas	1
Assistente de Estoque	1

Figura 33 – Resultado da consulta com o COUNT > 0

cargo	quantidade_funcionarios

Figura 34 – Resultado da consulta com o COUNT > 2

12. RESUMO DE PEDIDOS POR CLIENTE E FUNCIONÁRIO

```sql
SELECT
    c.nome AS cliente,
    COUNT(p.pedido_id) AS total_pedidos,
    f.nome AS funcionario_responsavel,
    COUNT(CASE WHEN p.status_pedido = 'Concluído' THEN 1 END) AS pedidos_atendidos_por_funcionario
FROM
    clientes c
JOIN
    pedidos p ON c.cliente_id = p.cliente_id
JOIN
    funcionarios f ON p.funcionario_id = f.funcionario_id
GROUP BY
    c.cliente_id, f.nome
HAVING
    COUNT(p.pedido_id) >= 1;
```

A consulta exibe clientes que realizaram mais de 3 pedidos atendidos, mostrando o total de pedidos feitos por cada cliente e os atendidos por cada funcionário responsável. Usa o JOIN para combinar as tabelas ‘pedidos’, ‘clientes’ e ‘funcionarios’ e filtra apenas pedidos com status "Atendido". O agrupamento por cliente e funcionário permite calcular o total de pedidos de cada cliente e os atendidos por funcionário. A cláusula HAVING garante que somente clientes com mais de 3 pedidos sejam exibidos. O resultado mostra os clientes, a quantidade de pedidos e o funcionário responsável. **(Obs.: Para fins de consulta, utilizei um COUNT >= 1 para retornar algum valor)**

cliente	total_pedidos	funcionario_responsavel	pedidos_atendidos_por_funcionario
Ana Souza	1	Carlos Almeida	1
Bruno Oliveira	1	Carlos Almeida	1
Fernanda Costa	1	Carlos Almeida	1
Carla Mendes	1	Mariana Oliveira	1
Daniela Lima	1	Mariana Oliveira	1
Eduardo Silva	1	Lucas Pereira	1
João Carvalho	1	Lucas Pereira	1
Gabriel Santos	1	Fernanda Costa	1
Helena Martins	1	Fernanda Costa	1
Isabel Ribeiro	1	Rafael Silva	1

Figura 35 – Resultado da consulta com o COUNT >= 1

cliente	total_pedidos	funcionario_responsavel	pedidos_atendidos_por_funcionario

Figura 36 – Resultado da consulta com o COUNT > 3

13. HISTÓRICO DE COMPRAS DE UM CLIENTE

```sql
SELECT
    c.nome AS cliente,
    p.nome AS produto,
    ip.quantidade,
    ip.preco_unitario,
    pd.data_pedido
FROM
    itens_pedido ip
JOIN
    pedidos pd ON ip.pedido_id = pd.pedido_id
JOIN
    produtos p ON ip.produto_id = p.produto_id
JOIN
    clientes c ON pd.cliente_id = c.cliente_id
WHERE
    c.cliente_id = 4
    AND pd.data_pedido >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
ORDER BY
    pd.data_pedido DESC;
```

A consulta mostra o histórico de compras de um cliente específico no último ano, exibindo o nome do produto, quantidade comprada, preço unitário e data do pedido. Para isso, conecta as tabelas ‘itens_pedido’, ‘pedidos’, ‘produtos’ e ‘clientes’ usando JOINs, relacionando os pedidos ao cliente e produtos correspondentes. Filtra as compras do cliente pelo nome especificado e limita os resultados a pedidos feitos no último ano com a condição ‘data_pedido >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)’. Os resultados são organizados em ordem decrescente pela data do pedido, mostrando as compras mais recentes primeiro.

cliente	produto	quantidade	preco_unitario	data_pedido
Daniela Lima	Cafeteira Express	1	400.00	2024-10-04
Daniela Lima	Camiseta Eco	2	70.00	2024-10-04

Figura 37 – Resultado da consulta com o ID = 4, para fins de exemplo.

14. HISTÓRICO DE COMPRAS DE UM CLIENTE

```sql
SELECT
    p.nome AS produto,
    SUM(ip.preco_unitario * ip.quantidade) AS receita_total
FROM
    produtos p
INNER JOIN itens_pedido ip ON p.produto_id = ip.produto_id
GROUP BY
    p.produto_id, p.nome
ORDER BY
    receita_total DESC
LIMIT 1;
```

A consulta busca em todas as vendas (itens de pedido) e, para cada produto, calcula o total de vendas multiplicando a quantidade vendida pelo preço unitário. Em seguida, ordena os produtos pela receita total, do maior para o menor. Ao limitar o resultado a apenas uma linha, a consulta retorna o produto que gerou a maior receita de todas.

produto	receita_total
Smartphone X10	4500.00

Figura 38 – Resultado da consulta

4. ALTERAÇÕES E MANUTENÇÕES NO BANCO DE DADOS

```
ALTER TABLE fornecedores
ADD COLUMN telefone VARCHAR(20);

ALTER TABLE funcionarios
ADD COLUMN cpf VARCHAR(14);

ALTER TABLE clientes
ADD COLUMN cep VARCHAR(8);
```

ALTER TABLE fornecedores ADD COLUMN telefone VARCHAR(20);
Esse comando modifica a estrutura da tabela "fornecedores", adicionando uma nova coluna chamada "telefone". Essa coluna será do tipo texto (VARCHAR) com um tamanho máximo de 20 caracteres, o que é adequado para armazenar números de telefone. Em resumo, estamos incluindo um campo para registrar o número de telefone de cada fornecedor na base de dados.

ALTER TABLE funcionarios ADD COLUMN cpf VARCHAR(14);
De forma semelhante, essa instrução altera a tabela "funcionarios", adicionando uma nova coluna chamada "cpf" (Cadastro de Pessoa Física). A coluna "cpf" será do tipo texto (VARCHAR) com um tamanho fixo de 14 caracteres, que é o padrão para o número de CPF no Brasil. Com essa alteração, passaremos a armazenar o CPF de cada funcionário na tabela.

ALTER TABLE clientes ADD COLUMN cep VARCHAR(8);
Por fim, esse comando modifica a tabela "clientes", adicionando uma nova coluna chamada "cep" (Código de Endereçamento Postal). A coluna "cep" será do tipo texto (VARCHAR) com um tamanho fixo de 8 caracteres, que é o padrão para o CEP no Brasil. Essa nova coluna permitirá armazenar o CEP de cada cliente, facilitando a localização e organização dos dados.

5. DIAGRAMA DE RELACIONAMENTO DE ENTIDADES (ERD)