Projeto de Banco de Dados: Sistema de Gerenciamento de Pedidos, Estoque, Fornecedores e Clientes

Sumário

1. Introdução

Este projeto modela um banco de dados para uma fábrica de componentes eletrônicos com foco no gerenciamento integrado de pedidos, estoque, fornecedores e clientes. O sistema permite garantir a disponibilidade de estoque antes de confirmar pedidos, rastrear fornecedores, armazenar informações de clientes e manter um histórico completo de todas as transações realizadas.

O sistema utiliza views, functions, procedures e triggers para automatizar processos e manter a integridade dos dados, proporcionando um controle eficiente e robusto das operações comerciais.

2. Estrutura do Banco

Banco: `sistema_gerenciamento`

Tabelas:

- **Fornecedores** - Cadastro dos fornecedores de componentes
- **Clientes** - Informações dos compradores
- **Estoque** - Controle de disponibilidade de produtos
- **Pedidos** - Solicitações dos clientes
- **Historico_Pedidos** - Registro completo de todas as transações

Relacionamentos:

- `Estoque.ID_Fornecedor` → `Fornecedores.ID`
- `Pedidos.ID_Cliente` → `Clientes.ID`
- `Historico_Pedidos.ID_Cliente` → `Clientes.ID`

3. Script SQL Completo (Comentado)

```
-- ============================================
-- SISTEMA DE GERENCIAMENTO DE PEDIDOS, ESTOQUE, FORNECEDORES E
CLIENTES
-- SENAI - Exercício Projeto
-- ============================================

-- Criação do banco de dados
CREATE DATABASE IF NOT EXISTS sistema_gerenciamento;
USE sistema_gerenciamento;

-- ============================================
-- CRIAÇÃO DAS TABELAS (ORDEM CORRETA)
-- ============================================

-- Primeiro: Tabelas sem dependências (tabelas pais)
-- Tabela Fornecedores
CREATE TABLE Fornecedores (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Nome VARCHAR(100),
    Contato VARCHAR(100)
);

-- Tabela Clientes
CREATE TABLE Clientes (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Nome VARCHAR(100),
    Email VARCHAR(100)
);

-- Segundo: Tabelas que dependem das anteriores (tabelas filhas)
-- Tabela Estoque
CREATE TABLE Estoque (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Produto VARCHAR(100),
    Quantidade INT,
    ID_Fornecedor INT,
    FOREIGN KEY (ID_Fornecedor) REFERENCES Fornecedores(ID)
);

-- Tabela Pedidos
CREATE TABLE Pedidos (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Produto VARCHAR(100),
    Quantidade INT,
    ID_Cliente INT,
    Status VARCHAR(50) DEFAULT 'Pendente',
    DataPedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (ID_Cliente) REFERENCES Clientes(ID)
);

-- Tabela Historico_Pedidos
CREATE TABLE Historico_Pedidos (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Produto VARCHAR(100),
    Quantidade INT,
    ID_Cliente INT,
    Status VARCHAR(50),
    DataPedido TIMESTAMP,
```

```
    FOREIGN KEY (ID_Cliente) REFERENCES Clientes(ID)
);
```

4. Objetos do Banco

A. VIEWS – Consultas Facilitadas

View 1: Pedidos com Informações dos Clientes

```
CREATE VIEW VIEW_Pedidos_Clientes AS
SELECT
    p.ID as ID_Pedido,
    p.Produto,
    p.Quantidade,
    p.Status,
    p.DataPedido,
    c.Nome as Nome_Cliente,
    c.Email as Email_Cliente
FROM Pedidos p
INNER JOIN Clientes c ON p.ID_Cliente = c.ID;
```

View 2: Estoque com Informações dos Fornecedores

```
CREATE VIEW VIEW_Estoque_Fornecedores AS
SELECT
    e.ID as ID_Estoque,
    e.Produto,
    e.Quantidade as Estoque_Disponivel,
    f.Nome as Nome_Fornecedor,
    f.Contato as Contato_Fornecedor
FROM Estoque e
INNER JOIN Fornecedores f ON e.ID_Fornecedor = f.ID;
```

B. FUNCTION – Verificação de Estoque

```
DELIMITER //
CREATE FUNCTION VerificarEstoque(produto_nome VARCHAR(100))
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE estoque_atual INT DEFAULT 0;

    SELECT Quantidade INTO estoque_atual
    FROM Estoque
    WHERE Produto = produto_nome;

    RETURN COALESCE(estoque_atual, 0);
END //
DELIMITER ;
```

C. PROCEDURE – Processar Pedidos

```
DELIMITER //
CREATE PROCEDURE ProcessarPedido(IN pedido_id INT)
BEGIN
    DECLARE produto_pedido VARCHAR(100);
```

```
    DECLARE quantidade_pedida INT;
    DECLARE estoque_disponivel INT;
    DECLARE cliente_id INT;
    DECLARE data_pedido TIMESTAMP;
    DECLARE estoque_id INT;

    -- Buscar informações do pedido
    SELECT Produto, Quantidade, ID_Cliente, DataPedido
    INTO produto_pedido, quantidade_pedida, cliente_id, data_pedido
    FROM Pedidos
    WHERE ID = pedido_id AND Status = 'Pendente';

    -- Verificar estoque disponível e obter ID do estoque
    SELECT ID, Quantidade INTO estoque_id, estoque_disponivel
    FROM Estoque
    WHERE Produto = produto_pedido;

    -- Se há estoque suficiente
    IF estoque_disponivel >= quantidade_pedida THEN
        -- Atualizar estoque usando a chave primária
        UPDATE Estoque
        SET Quantidade = Quantidade - quantidade_pedida
        WHERE ID = estoque_id;

        -- Confirmar pedido usando a chave primária
        UPDATE Pedidos
        SET Status = 'Confirmado'
        WHERE ID = pedido_id;

        -- Registrar no histórico
        INSERT INTO Historico_Pedidos (Produto, Quantidade,
ID_Cliente, Status, DataPedido)
        VALUES (produto_pedido, quantidade_pedida, cliente_id,
'Confirmado', data_pedido);

        SELECT 'Pedido confirmado com sucesso!' as Resultado;
    ELSE
        -- Pedido negado por falta de estoque
        UPDATE Pedidos
        SET Status = 'Negado - Estoque Insuficiente'
        WHERE ID = pedido_id;

        -- Registrar no histórico
        INSERT INTO Historico_Pedidos (Produto, Quantidade,
ID_Cliente, Status, DataPedido)
        VALUES (produto_pedido, quantidade_pedida, cliente_id, 'Negado
- Estoque Insuficiente', data_pedido);

        SELECT CONCAT('Pedido negado! Estoque disponível: ',
estoque_disponivel, ', Solicitado: ', quantidade_pedida) as Resultado;
    END IF;
END //
DELIMITER ;
```

D. TRIGGER – Histórico Automático

```
DELIMITER //
CREATE TRIGGER TriggerHistoricoPedidos
AFTER UPDATE ON Pedidos
FOR EACH ROW
```

```
BEGIN
    IF NEW.Status != OLD.Status THEN
        INSERT INTO Historico_Pedidos (Produto, Quantidade,
ID_Cliente, Status, DataPedido)
        VALUES (NEW.Produto, NEW.Quantidade, NEW.ID_Cliente,
NEW.Status, NEW.DataPedido);
    END IF;
END //
DELIMITER ;
```

5. Dados de Teste (INSERTs)

Fornecedores

```
INSERT INTO Fornecedores (Nome, Contato) VALUES
('TechComponents Ltda', 'contato@techcomponents.com'),
('EletroSupplies S.A.', 'vendas@eletrosupplies.com'),
('ComponentesPro', 'comercial@componentespro.com'),
('MicroParts Brasil', 'pedidos@microparts.com.br');
```

Clientes

```
INSERT INTO Clientes (Nome, Email) VALUES
('João Silva', 'joao.silva@email.com'),
('Maria Santos', 'maria.santos@email.com'),
('Pedro Oliveira', 'pedro.oliveira@email.com'),
('Ana Costa', 'ana.costa@email.com'),
('Carlos Ferreira', 'carlos.ferreira@email.com');
```

Estoque Inicial

```
INSERT INTO Estoque (Produto, Quantidade, ID_Fornecedor) VALUES
('Resistor 1kΩ', 1000, 1),
('Capacitor 100µF', 500, 1),
('LED Verde 5mm', 800, 2),
('Transistor BC547', 300, 2),
('Arduino Uno R3', 50, 3),
('Protoboard 830 pontos', 75, 3),
('Jumpers Macho-Fêmea', 200, 4),
('Display LCD 16x2', 25, 4),
('Sensor Ultrassônico HC-SR04', 40, 1),
('Servo Motor SG90', 60, 2);
```

Pedidos de Exemplo

```
INSERT INTO Pedidos (Produto, Quantidade, ID_Cliente, Status) VALUES
('Arduino Uno R3', 2, 1, 'Pendente'),
('LED Verde 5mm', 50, 2, 'Pendente'),
('Resistor 1kΩ', 100, 3, 'Pendente'),
('Display LCD 16x2', 3, 4, 'Pendente'),
('Servo Motor SG90', 5, 5, 'Pendente'),
('Capacitor 100µF', 20, 1, 'Pendente'),
('Protoboard 830 pontos', 2, 2, 'Pendente');
```

Exemplos de Uso

```
-- Processar pedidos
CALL ProcessarPedido(1); -- Arduino Uno R3
CALL ProcessarPedido(2); -- LED Verde 5mm
CALL ProcessarPedido(3); -- Resistor 1kΩ

-- Verificar estoque de um produto
SELECT VerificarEstoque('Arduino Uno R3') as Estoque_Atual;

-- Consultar pedidos com informações dos clientes
SELECT * FROM VIEW_Pedidos_Clientes;

-- Consultar estoque com informações dos fornecedores
SELECT * FROM VIEW_Estoque_Fornecedores;
```

6. Explicações Técnicas

Views

- **VIEW_Pedidos_Clientes**: Facilita a visualização de pedidos com informações completas dos clientes, eliminando a necessidade de escrever JOINs repetitivos
- **VIEW_Estoque_Fornecedores**: Permite consultar o estoque junto com dados dos fornecedores de forma simplificada

Function

- **VerificarEstoque()**: Função reutilizável que retorna a quantidade disponível de qualquer produto no estoque, facilitando consultas rápidas

Procedure

- **ProcessarPedido()**: Centraliza toda a lógica de processamento de pedidos, incluindo:
    - Verificação de estoque disponível
    - Atualização automática do estoque
    - Mudança de status do pedido
    - Registro no histórico
    - Tratamento de casos de estoque insuficiente

Trigger

- **TriggerHistoricoPedidos**: Automatiza o registro no histórico sempre que o status de um pedido é alterado, garantindo rastreabilidade completa sem intervenção manual

Recursos Especiais

- **Timestamps automáticos**: Controle temporal preciso das operações
- **Chaves estrangeiras**: Garantem integridade referencial
- **Tratamento de erros**: Sistema robusto com validações
- **Safe update mode**: Compatível com configurações de segurança do MySQL

7. Conclusão

Este projeto implementa um sistema completo e robusto de gerenciamento comercial para fábricas de componentes eletrônicos, seguindo as melhores práticas de banco de dados.

Principais Benefícios:

- **Automação completa**: Processamento de pedidos sem intervenção manual
- **Controle de estoque**: Verificação antes da confirmação evita overselling
- **Rastreabilidade total**: Histórico completo de todas as transações
- **Integridade de dados**: Relacionamentos e validações garantem consistência
- **Facilidade de consulta**: Views simplificam relatórios e análises
- **Escalabilidade**: Estrutura preparada para crescimento do negócio

O sistema é ideal para pequenas e médias empresas que precisam de controle eficiente de pedidos, estoque e relacionamento com fornecedores e clientes, proporcionando uma base sólida para operações comerciais automatizadas e confiáveis.