Projeto de Banco de Dados: E-LogiWare

Sumário

1. Introdução

Este projeto modela um banco de dados para uma indústria com foco em produção e controle de estoque. O sistema permite cadastrar funcionários, produtos, registrar produção, atualizar estoques e consultar dados relevantes usando *views*, *functions*, *procedures* e *triggers*.

2. Estrutura do Banco

Banco: logiware

Tabelas:

- Fornecedor
- Armazenamento
- Produto
- Pedido
- Transportadora
- log_estoque
- Fornecedor_Produto

3. Script SQL (Comentado)

```
/* Criação do banco de dados */
DROP DATABASE IF EXISTS logiware;

CREATE DATABASE logiware;

USE logiware;

/* Criação das tabelas */
CREATE TABLE Fornecedor (
      id INT,
    nome VARCHAR(45),
    contato VARCHAR(20),
    PRIMARY KEY (id)
);

CREATE TABLE Armazenamento (
      id INT,
    capTotal INT,
    capUsada INT,
    PRIMARY KEY (id)
);
```

```
CREATE TABLE Produto (
      id INT AUTO_INCREMENT,
    id_armazenamento INT,
    descricao VARCHAR(50),
    qntdEstoque INT,
    PRIMARY KEY (id),
    FOREIGN KEY (id_armazenamento)
            REFERENCES Armazenamento (id)
);

CREATE TABLE Pedido (
      id INT,
    data DATE,
    status VARCHAR(45),
    id_fornecedor INT,
    id_produto INT,
    PRIMARY KEY (id),
    FOREIGN KEY (id_fornecedor)
            REFERENCES Fornecedor (id),
      FOREIGN KEY (id_produto)
            REFERENCES Produto (id)
);

CREATE TABLE Transportadora (
      id INT,
    nome VARCHAR(45),
    contato VARCHAR(20),
    id_armazenamento INT,
    PRIMARY KEY (id),
    FOREIGN KEY (id_armazenamento)
            REFERENCES Armazenamento (id)
);

/* Tabela de controle de estoque */
CREATE TABLE log_estoque (
      id_log INT AUTO_INCREMENT PRIMARY KEY,
    id_produto INT,
    quantidade_antiga INT,
    quantidade_nova INT,
    data_alteracao TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

/* Tabela de controle de reposição */
CREATE TABLE Fornecedor_Produto (
    id_fornecedor INT,
    id_produto INT,
    data_ultima_reposicao DATE,
    quantidade_reposicao INT,
    PRIMARY KEY (id_fornecedor, id_produto),
    FOREIGN KEY (id_fornecedor)
            REFERENCES Fornecedor(id),
    FOREIGN KEY (id_produto)
            REFERENCES Produto(id)
);
```

4. Objetos de Banco

A. PROCEDURE – Atualiza o estoque

```
/* Criação da procedure de atualização de estoque */
DELIMITER $$
```

```
CREATE PROCEDURE atualiza_estoque (
      IN p_id INT,
    IN p_retirada INT
)
BEGIN
      UPDATE Produto SET qntdEstoque = (qntdEstoque - p_retirada)
WHERE id = p_id;
END $$
DELIMITER ;
```

B. TRIGGER – Monitora o log de estoque

```
/* Criação do TRIGGER de monitoração do log de estoque */
DELIMITER $$
CREATE TRIGGER after_log_estoque
AFTER UPDATE ON Produto
FOR EACH ROW
BEGIN
      IF OLD.qntdEstoque != NEW.qntdEstoque THEN
            INSERT INTO log_estoque (id_produto, quantidade_antiga,
quantidade_nova, data_alteracao) VALUES (OLD.id, OLD.qntdEstoque,
NEW.qntdEstoque, NOW());
      END IF;
END $$
DELIMITER ;
```

C. PROCEDURE – Atribuição dinâmica de locais de armazenamento

```
/* Criação de PROCEDURE que atribui dinamicamente os locais de
armazenamento */
DELIMITER $$
CREATE PROCEDURE atribuir_armazenamento (
    IN p_descricao VARCHAR(50),
    IN p_qntdEstoque INT
)
BEGIN
    DECLARE v_id_armazenamento INT;
    DECLARE v_espaco_disponivel INT;

    /* Encontrar um armazenamento com espaço suficiente */
    SELECT id INTO v_id_armazenamento
    FROM Armazenamento
    WHERE (capTotal - capUsada) >= p_qntdEstoque
    ORDER BY (capTotal - capUsada)
    LIMIT 1;

    /* Se encontrou um local, inserir o produto e atualizar o espaço
usado */
    IF v_id_armazenamento IS NOT NULL THEN
        INSERT INTO Produto (id_armazenamento, descricao, qntdEstoque)
        VALUES (v_id_armazenamento, p_descricao, p_qntdEstoque);

        UPDATE Armazenamento
        SET capUsada = capUsada + p_qntdEstoque
        WHERE id = v_id_armazenamento;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Sem espaço disponível em nenhum
armazenamento.';
    END IF;
END $$
```

```
DELIMITER ;
```

D. VIEW – Monitora o estoque dos produtos no armazenamento

```
/* Criação de VIEW que monitora o estoque dos produtos no
armazenamento */
CREATE VIEW vw_monitorar_armazenamento AS
SELECT id, capTotal, capUsada,
      CASE
            WHEN capTotal >= capUsada THEN 'Estoque suficiente'
        ELSE 'Estoque insuficiente'
      END AS status_estoque
FROM Armazenamento;
```

E. PROCEDURE – Registro detalhado dos pedidos

```
/* Criação de PROCEDURE que mostra o registro detalhado de pedidos */
DELIMITER $$
CREATE PROCEDURE pedido_detalhado(
      IN pedido_id INT
)
BEGIN
      SELECT Pedido.id AS numero_pedido, Produto.descricao AS
nome_produto, Pedido.status, Fornecedor.nome as nome_fornecedor,
Transportadora.nome AS nome_transportadora
    FROM Pedido
    JOIN Produto ON Pedido.id_produto = Produto.id
    JOIN Fornecedor ON Pedido.id_fornecedor = Fornecedor.id
    JOIN Armazenamento ON Produto.id_armazenamento = Armazenamento.id
    JOIN Transportadora ON Armazenamento.id =
Transportadora.id_armazenamento
    WHERE Pedido.id = pedido_id;
END $$
DELIMITER ;
```

F. PROCEDURE – Reposição dos fornecedores com produtos

```
/* Controle de reposição dos fornecedores com seus respectivos
produtos */
DELIMITER $$
CREATE PROCEDURE associar_fornecedor_produto(
    IN p_id_fornecedor INT,
    IN p_id_produto INT,
    IN p_data_ultima_reposicao DATE,
    IN p_quantidade_reposicao INT
)
BEGIN
    INSERT INTO Fornecedor_Produto (id_fornecedor, id_produto,
data_ultima_reposicao, quantidade_reposicao)
    VALUES (p_id_fornecedor, p_id_produto, p_data_ultima_reposicao,
p_quantidade_reposicao);
END $$
DELIMITER ;
```

G. FUNCTION – Verifica o estoque total de produtos

```
/* Função para verificar o estoque total de produtos */
DELIMITER $$
CREATE FUNCTION fn_verificar_estoque()
RETURNS INT
READS SQL DATA
BEGIN
      DECLARE amt INT;
    SELECT SUM(qntdEstoque) INTO amt FROM Produto;
```

```
    RETURN amt;
END $$
DELIMITER ;
```

5. Dados de Teste (INSERTs)

```
/* Inserindo informações */
/* Fornecedor */
INSERT INTO Fornecedor (id, nome, contato) VALUES
(1, 'Fornecedor A', '11987654321'),
(2, 'Fornecedor B', '21912345678'),
(3, 'Fornecedor C', '31998765432');

/* Armazenamento */
INSERT INTO Armazenamento (id, capTotal, capUsada) VALUES
(1, 10000, 2500),
(2, 8000, 3000),
(3, 12000, 12001);

/* Produto */
INSERT INTO Produto (id, id_armazenamento, descricao, qntdEstoque)
VALUES
(1, 1, 'Notebook Dell Inspiron', 150),
(2, 2, 'Mouse Logitech Wireless', 500),
(3, 3, 'Monitor Samsung 24"', 200);

/* Pedido */
INSERT INTO Pedido (id, data, status, id_fornecedor, id_produto)
VALUES
(1, '2025-05-01', 'Entregue', 1, 1),
(2, '2025-05-03', 'Pendente', 2, 2),
(3, '2025-05-05', 'Cancelado', 3, 3);

/* Transportadora */
INSERT INTO Transportadora (id, nome, contato, id_armazenamento)
VALUES
(1, 'Transporte Rápido', '11911112222', 1),
(2, 'Carga Segura', '21933334444', 2),
(3, 'Logística Sul', '31955556666', 3);

/* Exemplo da funcionalidade da procedure de atualização de estoque */
CALL atualiza_estoque (1, 3);
SELECT * FROM Produto;

/* Exemplo de funcionamento do TRIGGER de monitoração de estoque */
UPDATE Produto SET qntdEstoque = 130 WHERE id = 1;
SELECT * FROM log_estoque;

/* Exemplo de funcionamento da PROCEDURE de atribuição dinâmica de
locais de armazenamento */
CALL atribuir_armazenamento('Monitor Samsung 24', 200);

/* VIEW de monitoramento de armazenamento */
SELECT * FROM vw_monitorar_armazenamento;

/* Exemplo do funcionamento da PROCEDURE que mostra o registro
detalhado de pedidos */
CALL pedido_detalhado(2);

/* Exemplo de funcionamento da PROCEDURE de controle de reposição */
CALL associar_fornecedor_produto(1, 2, '2025-05-01', 500);
```

```
SELECT * FROM Fornecedor_Produto;

/* Exemplo da função de verificar estoque de determinado produto */
SELECT fn_verificar_estoque() FROM Produto;
```

6. Explicações Técnicas

- **View** facilita visualização de dados complexos sem reescrever queries.
- **Function** permite cálculos reutilizáveis como valor total do estoque.
- **Procedure** centraliza lógica de produção e atualização de estoque.
- **Trigger** automatiza atualização de estoque sem intervenção manual.

7. Conclusão

Este projeto implementa um sistema robusto de controle industrial com boas práticas de banco de dados, usando objetos SQL para manter a lógica e integridade dos dados.
Ele é ideal para o controle de estoques, monitoramento e detalhamento de pedidos.