Documentação Técnica do Projeto de Banco de Dados

Introdução ao Projeto

Descrição do Projeto

Este projeto foi desenvolvido para gerenciar um site de sabonetes artesanais, permitindo que clientes escolham entre diferentes tipos de sabonetes, preencham seus dados e realizem pedidos. Após a finalização, o sistema armazena todas as informações no banco de dados.

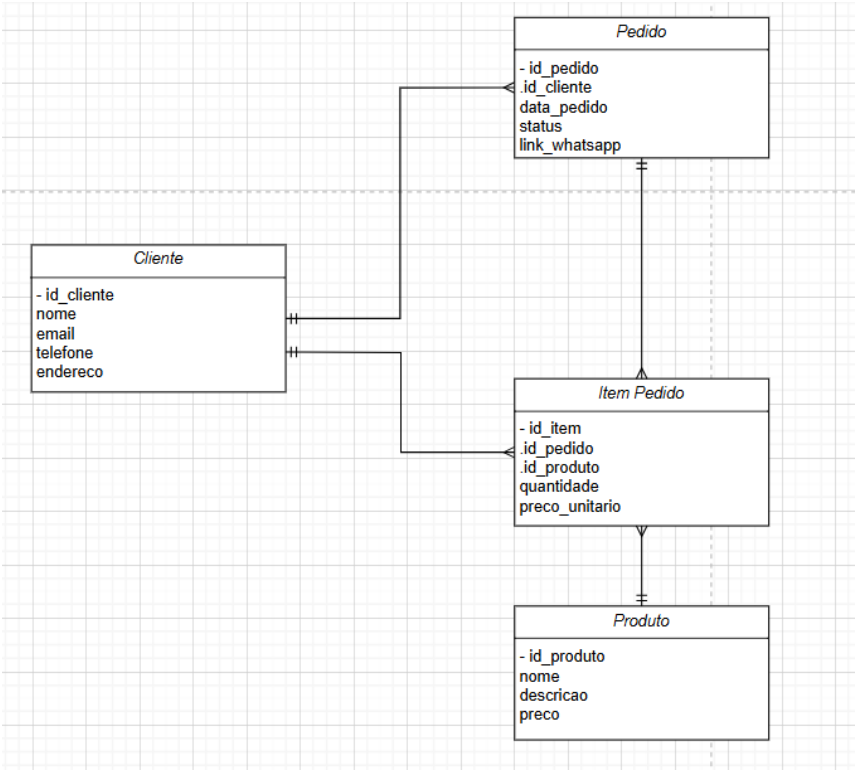
Objetivo do Banco de Dados

O objetivo do banco de dados é organizar e gerenciar informações dos pedidos, clientes, pedidos e pagamentos, garantindo eficiência no processamento de transações e relatórios.

Modelagem de Dados

Diagrama Entidade-Relacionamento (DER)

O Diagrama Entidade-Relacionamento (DER) ilustra as entidades e relacionamentos do sistema, como mostrado abaixo:



Estrutura do Banco de Dados

Scripts de Criação do Banco de Dados

Os scripts SQL para criação das tabelas e seus relacionamentos são apresentados abaixo:

```sql
CREATE TABLE produtos (
    id_produto INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco DECIMAL(10,2) NOT NULL
);

CREATE TABLE clientes (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefone VARCHAR(20),
    endereco TEXT
);

CREATE TABLE pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    data_pedido DATETIME DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(50) DEFAULT 'Pendente',
    link_whatsapp VARCHAR(255),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);

CREATE TABLE itens_pedido (
    id_item INT AUTO_INCREMENT PRIMARY KEY,
    id_pedido INT,
    id_produto INT,
    quantidade INT DEFAULT 1,
    preco_unitario DECIMAL(10,2),
    FOREIGN KEY (id_pedido) REFERENCES pedidos(id_pedido),
    FOREIGN KEY (id_produto) REFERENCES produtos(id_produto)
);
```

Descrição das Tabelas

Tabela clientes: Armazena informações dos clientes que realizam pedidos no site.

- **id_cliente: Identificador único do cliente.**
- **nome: Nome completo do cliente.**

- **email: Endereço de e-mail do cliente.**
- **telefone: Contato telefônico.**
- **endereco: Endereço de entrega informado pelo cliente.**

Tabela produtos: Armazena os sabonetes artesanais disponíveis para compra.

- **id_produto: Identificador único do produto.**
- **nome: Nome do sabonete.**
- **descricao: Descrição do sabonete, com suas propriedades.**
- **preco: Preço unitário do produto.**

Tabela pedidos: Registra os pedidos realizados pelos clientes.

- **id_pedido: Identificador único do pedido.**
- **id_cliente: Cliente que realizou o pedido.**
- **data_pedido: Data e hora em que o pedido foi feito.**
- **status: Status atual do pedido (ex: "Pendente", "Enviado").**
- **link_whatsapp: Link para o cliente acompanhar o pedido via WhatsApp.**
-

Tabela itens_pedido: Detalha os sabonetes incluídos em cada pedido.

- **id_item: Identificador único do item do pedido.**
- **id_pedido: Pedido ao qual o item pertence.**
- **id_produto: Produto relacionado ao item.**
- **quantidade: Quantidade comprada do sabonete.**
- **preco_unitario: Preço unitário no momento do pedido.**

Consultas e Funcionalidades

Descrição das Consultas

1. **Consulta de Pedidos por Cliente**: Listar todos os pedidos com nome do cliente e status

```
SELECT p.id_pedido, c.nome, p.data_pedido, p.status
FROM pedidos p
JOIN clientes c ON p.id_cliente = c.id_cliente;
```

2. **Consulta Produtos vendidos**: Produtos mais vendidos

```
-- Produtos mais vendidos
SELECT pr.nome, SUM(ip.quantidade) AS total_vendido
FROM itens_pedido ip
JOIN produtos pr ON ip.id_produto = pr.id_produto
GROUP BY pr.nome
ORDER BY total_vendido DESC;
```

3. **Consulta Total gasto**: Total gasto por cliente

```
SELECT c.nome, SUM(ip.quantidade * ip.preco_unitario) AS total_gasto
FROM clientes c
JOIN pedidos p ON c.id_cliente = p.id_cliente
JOIN itens_pedido ip ON p.id_pedido = ip.id_pedido
GROUP BY c.nome;
```

4. Detalhes de um pedido : Detalhes de um pedido específico (exemplo: pedido 1)

```
-- Detalhes de um pedido específico (exemplo: pedido 1)
SELECT p.id_pedido, pr.nome AS produto, ip.quantidade, ip.preco_unitario
FROM itens_pedido ip
JOIN produtos pr ON ip.id_produto = pr.id_produto
JOIN pedidos p ON ip.id_pedido = p.id_pedido
WHERE p.id_pedido = 1;
```

View:

1. **Detalhes do pedido**: Detalha os pedidos

```
-- View 1: Detalhes dos pedidos

DROP VIEW IF EXISTS vw_detalhes_pedidos;

CREATE VIEW vw_detalhes_pedidos AS
SELECT p.id_pedido, c.nome AS cliente, p.data_pedido, p.status, pr.nome AS produto, ip.quantidade, ip.preco_unitario
FROM pedidos p
JOIN clientes c ON p.id_cliente = c.id_cliente
JOIN itens_pedido ip ON p.id_pedido = ip.id_pedido
JOIN produtos pr ON ip.id_produto = pr.id_produto;

SELECT * FROM vw_detalhes_pedidos;
```

2. **Registro de Pagamentos**: Produtos mais vendidos

```sql
CREATE VIEW vw_produtos_mais_vendidos AS
SELECT pr.nome, SUM(ip.quantidade) AS total_vendido
FROM itens_pedido ip
JOIN produtos pr ON ip.id_produto = pr.id_produto
GROUP BY pr.nome;
```

3. **Total gasto por cliente:** Mostra quanto o cliente já gastou

```sql
CREATE VIEW vw_total_gasto_cliente AS
SELECT c.nome, SUM(ip.quantidade * ip.preco_unitario) AS total_gasto
FROM clientes c
JOIN pedidos p ON c.id_cliente = p.id_cliente
JOIN itens_pedido ip ON p.id_pedido = ip.id_pedido
GROUP BY c.nome;
```

4. **Pedidos pendentes:** Mostra os pedidos pendentes

```sql
DROP VIEW IF EXISTS vw_pedidos_pendentes;

CREATE VIEW vw_pedidos_pendentes AS
SELECT p.id_pedido, c.nome, p.data_pedido
FROM pedidos p
JOIN clientes c ON p.id_cliente = c.id_cliente
WHERE p.status = 'Pendente';
```

Procedures e Funções:

Procedure 1: Inserir novo cliente

```
DELIMITER //
CREATE PROCEDURE sp_inserir_cliente(
    IN p_nome VARCHAR(100),
    IN p_email VARCHAR(100),
    IN p_telefone VARCHAR(20),
    IN p_endereco TEXT
)
BEGIN
    INSERT INTO clientes (nome, email, telefone, endereco)
    VALUES (p_nome, p_email, p_telefone, p_endereco);
END;
//
DELIMITER ;
```

Procedure 2: Atualizar status do pedido

```
DELIMITER //
CREATE PROCEDURE sp_atualizar_status_pedido(
    IN p_id_pedido INT,
    IN p_novo_status VARCHAR(50)
)
BEGIN
    UPDATE pedidos
    SET status = p_novo_status
    WHERE id_pedido = p_id_pedido;
END;
//
DELIMITER ;
```

Função 1: Calcular total de um pedido

```
DELIMITER //
CREATE FUNCTION fn_total_pedido(p_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(quantidade * preco_unitario) INTO total
    FROM itens_pedido
    WHERE id_pedido = p_id;
    RETURN IFNULL(total, 0);
END;
//
DELIMITER ;
```

Função 2: Contar quantidade de pedidos de um cliente

```
DELIMITER //
CREATE FUNCTION fn_total_pedidos_cliente(p_id_cliente INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total INT;
    SELECT COUNT(*) INTO total
    FROM pedidos
    WHERE id_cliente = p_id_cliente;
    RETURN total;
END;
//
DELIMITER ;
```

Triggers

Trigger 1: Impede que alguém insira ou atualize um item de pedido com quantidade menor que 1

```
DELIMITER //

CREATE TRIGGER tg_verifica_quantidade_itens_update
BEFORE UPDATE ON itens_pedido
FOR EACH ROW
BEGIN
    IF NEW.quantidade < 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A quantidade deve ser maior ou igual a 1.';
    END IF;
END;
//

DELIMITER ;
```

Trigger 2: Impedir que pedido seja deletado se tiver itens

```
DELIMITER //
CREATE TRIGGER tg_bloquear_delete_pedido
BEFORE DELETE ON pedidos
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM itens_pedido WHERE id_pedido = OLD.id_pedido) > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Não é possível deletar um pedido que possui itens.';
    END IF;
END;
//
DELIMITER ;
```

Transação

Exemplo de transação: Criar um pedido com dois itens

```sql
START TRANSACTION;

INSERT INTO pedidos (id_cliente, status) VALUES (2, 'Pendente');
SET @id_pedido = LAST_INSERT_ID();

INSERT INTO itens_pedido (id_pedido, id_produto, quantidade, preco_unitario)
VALUES (@id_pedido, 1, 2, 29.90);

INSERT INTO itens_pedido (id_pedido, id_produto, quantidade, preco_unitario)
VALUES (@id_pedido, 3, 1, 8.00);

COMMIT;
```
