Esta função é o construtor da classe principal da sua aplicação. Ela é executada automaticamente quando um novo objeto da janela principal é criado

```
def __init__(self):
        super().__init__()
        self.setWindowTitle("Sistema de Vendas ReVeste")
        self.setGeometry(100, 100, 800, 600)

        self.init_db()
        self.init_ui()
        self.carregar_dados_iniciais()
```

Esta função é responsável por configurar a estrutura principal da interface do usuário.

```
    def init_ui(self):
        """Inicializa os componentes da interface gráfica."""
        self.tabs = QTabWidget()
        self.setCentralWidget(self.tabs)

        # Criação das abas
        self.init_cadastro_tab()
        self.init_vendas_tab()
        self.init_historico_tab()
        self.init_clientes_tab()

        self.aplicar_estilos()
```

O objetivo desta função é garantir que a estrutura do banco de dados SQLite exista.

```
    def init_db(self):
        """Garante que as tabelas da base de dados existem."""
        schema_queries = [
            """
            CREATE TABLE IF NOT EXISTS Usuario (
                id_usuario INTEGER PRIMARY KEY AUTOINCREMENT,
                nome TEXT NOT NULL,
                email TEXT NOT NULL UNIQUE,
                tipo_perfil TEXT NOT NULL DEFAULT 'Consumidor',
```

```
            senha TEXT NOT NULL DEFAULT '123'
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS Material (
            id_material INTEGER PRIMARY KEY AUTOINCREMENT,
            nome_material TEXT NOT NULL UNIQUE,
            tipo_material TEXT NOT NULL DEFAULT 'roupa',
            condicao TEXT NOT NULL DEFAULT 'novo'
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS Transacao (
            id_transacao INTEGER PRIMARY KEY AUTOINCREMENT,
            id_vendedor INTEGER NOT NULL DEFAULT 1,
            id_comprador INTEGER NOT NULL,
            id_material INTEGER NOT NULL,
            detalhes TEXT,
            data_transacao TEXT NOT NULL,
            status TEXT NOT NULL DEFAULT 'pendente',
            FOREIGN KEY (id_comprador) REFERENCES Usuario(id_usuario),
            FOREIGN KEY (id_material) REFERENCES Material(id_material)
        )
        """
    ]
    for query in schema_queries:
        self._executar_query(query)
```

Esta função serve para carregar os dados já existentes do banco de dados e exibi-los na interface gráfica assim que a aplicação é iniciada.

```
def carregar_dados_iniciais(self):
    """Carrega todos os dados da DB para a UI quando a aplicação inicia."""
    self.atualizar_clientes_ui()
    self.atualizar_produtos_ui()
    self.atualizar_historico_ui()
```

Esta função constrói a interface gráfica da aba "Cadastro".

```
    def init_cadastro_tab(self):
        """Cria a aba de cadastro de clientes e produtos."""
        self.cadastro_tab = QWidget()
        self.tabs.addTab(self.cadastro_tab, "Cadastro")
        layout = QFormLayout(self.cadastro_tab)

        self.cliente_input = QLineEdit()
        self.produto_input = QLineEdit()
        self.salvar_btn = QPushButton("Salvar")
        self.salvar_btn.setObjectName("salvarBtn")

        layout.addRow("Nome do Cliente:", self.cliente_input)
        layout.addRow("Nome do Produto:", self.produto_input)
        layout.addRow(self.salvar_btn)

        self.salvar_btn.clicked.connect(self.handle_salvar_cadastro)
```

Cria a aba "Registro de Vendas", onde o usuário informa os dados de uma venda e registra no banco de dados.

```
def init_vendas_tab(self):
    """Cria a aba para registro de vendas."""
    self.vendas_tab = QWidget()
    self.tabs.addTab(self.vendas_tab, "Registro de Vendas")
    layout = QFormLayout(self.vendas_tab)

    # MELHORIA: Data atual como padrão
    self.data_input = QLineEdit(datetime.now().strftime('%d/%m/%Y'))
    self.cliente_combo = QComboBox()
    self.produto_combo = QComboBox()
    self.itens_input = QTextEdit(placeholderText="Detalhes sobre a venda, condição da peça,
etc.")
    self.registrar_btn = QPushButton("Registrar Venda")
    self.registrar_btn.setObjectName("registrarBtn")

    layout.addRow("Data (dd/mm/aaaa):", self.data_input)
    layout.addRow("Cliente:", self.cliente_combo)
    layout.addRow("Produto:", self.produto_combo)
    layout.addRow("Detalhes Adicionais:", self.itens_input)
    layout.addRow(self.registrar_btn)
```

```
    self.registrar_btn.clicked.connect(self.handle_registrar_venda)
```

Essa função cria a aba "Histórico de Vendas", exibindo uma tabela com vendas realizadas e um botão para calcular totais por período.

```
def init_historico_tab(self):
    """Cria a aba de histórico de vendas."""
    self.historico_tab = QWidget()
    self.tabs.addTab(self.historico_tab, "Histórico de Vendas")
    layout = QVBoxLayout(self.historico_tab)

    self.tabela_vendas = QTableWidget()
    self.tabela_vendas.setColumnCount(4)
    self.tabela_vendas.setHorizontalHeaderLabels(["Data", "Cliente", "Produto", "Detalhes"])
    self.tabela_vendas.setEditTriggers(QAbstractItemView.NoEditTriggers)
    self.tabela_vendas.setSelectionBehavior(QAbstractItemView.SelectRows)

    header = self.tabela_vendas.horizontalHeader()
    header.setSectionResizeMode(0, QHeaderView.ResizeToContents)
    header.setSectionResizeMode(1, QHeaderView.ResizeToContents)
    header.setSectionResizeMode(2, QHeaderView.ResizeToContents)
    header.setSectionResizeMode(3, QHeaderView.Stretch)

    self.btn_totais = QPushButton("Calcular Totais de Vendas por Período")
    self.btn_totais.setObjectName("totaisBtn")

    layout.addWidget(self.tabela_vendas)
    layout.addWidget(self.btn_totais)

    self.btn_totais.clicked.connect(self.handle_calcular_totais)
```

Essa função cria a aba "Clientes Cadastrados", que mostra a lista de clientes existentes e permite remover um cliente selecionado.

```
def init_clientes_tab(self):
    """Cria a aba de gestão de clientes cadastrados."""
    self.clientes_tab = QWidget()
    self.tabs.addTab(self.clientes_tab, "Clientes Cadastrados")
    layout = QVBoxLayout(self.clientes_tab)

    self.tabela_clientes = QTableWidget()
    self.tabela_clientes.setColumnCount(1)
    self.tabela_clientes.setHorizontalHeaderLabels(["Clientes"])
```

```
    self.tabela_clientes.setEditTriggers(QAbstractItemView.NoEditTriggers)
    self.tabela_clientes.setSelectionBehavior(QAbstractItemView.SelectRows)
    self.tabela_clientes.horizontalHeader().setSectionResizeMode(0, QHeaderView.Stretch)

    self.btn_remover_cliente = QPushButton("Remover Cliente Selecionado")
    self.btn_remover_cliente.setObjectName("removerClienteBtn")

    layout.addWidget(self.tabela_clientes)
    layout.addWidget(self.btn_remover_cliente)

    self.btn_remover_cliente.clicked.connect(self.handle_remover_cliente)
```

Salva o nome do cliente e/ou produto no banco de dados. Também limpa os campos e atualiza a interface após o cadastro

```
def handle_salvar_cadastro(self):
    """Lida com o clique do botão 'Salvar' na aba de cadastro."""
    cliente_nome = self.cliente_input.text().strip()
    produto_nome = self.produto_input.text().strip()

    if not cliente_nome and not produto_nome:
        self.show_message("Preencha o nome do cliente ou do produto.", error=True)
        return

    try:
        if cliente_nome:
            email_cliente = f'{cliente_nome.lower().replace(" ", ".")}@email.com'
            query = "INSERT INTO Usuario (nome, email) VALUES (?, ?)"
            self._executar_query(query, (cliente_nome, email_cliente))
            self.cliente_input.clear()
            self.show_message(f"Cliente '{cliente_nome}' cadastrado com sucesso!")
            self.atualizar_clientes_ui()

        if produto_nome:
            query = "INSERT INTO Material (nome_material) VALUES (?)"
            self._executar_query(query, (produto_nome,))
            self.produto_input.clear()
            self.show_message(f"Produto '{produto_nome}' cadastrado com sucesso!")
            self.atualizar_produtos_ui()

    except sqlite3.IntegrityError:
        self.show_message("Erro: Cliente ou Produto já existe.", error=True)
    except Exception as e:
        self.show_message(f"Ocorreu um erro inesperado: {e}", error=True)
```

Registra uma nova venda no banco de dados com os dados informados. Valida os campos e atualiza a aba de histórico.

```
def handle_registrar_venda(self):
    """Lida com o clique do botão 'Registrar Venda'."""
    data_str = self.data_input.text().strip()
    cliente_nome = self.cliente_combo.currentText()
    produto_nome = self.produto_combo.currentText()
    detalhes = self.itens_input.toPlainText().strip()

    data_db = self._formatar_data_para_db(data_str)
    if not data_db:
        self.show_message(f"Formato de data inválido: '{data_str}'.\nUse o formato dd/mm/aaaa.",
error=True)
        return
    if not (cliente_nome and produto_nome):
        self.show_message("Por favor, selecione um Cliente e um Produto.", error=True)
        return

    try:
        comprador_id = self._get_id_por_nome('Usuario', 'id_usuario', 'nome', cliente_nome)
        material_id = self._get_id_por_nome('Material', 'id_material', 'nome_material',
produto_nome)

        if comprador_id is None or material_id is None:
            self.show_message("Erro: Cliente ou Produto não encontrado na base de dados.",
error=True)
            return

        query = "INSERT INTO Transacao (id_comprador, id_material, detalhes, data_transacao)
VALUES (?, ?, ?, ?)"
        self._executar_query(query, (comprador_id, material_id, detalhes, data_db))

        self.itens_input.clear()
        self.show_message("Venda registrada com sucesso!")
        self.atualizar_historico_ui()

    except Exception as e:
        self.show_message(f"Ocorreu um erro ao registrar a venda: {e}", error=True)
```

Consulta o banco para contar quantas vendas foram feitas por dia, semana e mês. Exibe os resultados em uma janela de mensagem.

```
def handle_calcular_totais(self):
    """Busca na DB e exibe os totais de vendas por períodos."""
    try:
        queries = {
            "DIA": "SELECT strftime('%d/%m/%Y', data_transacao), COUNT(*) FROM Transacao
GROUP BY data_transacao ORDER BY data_transacao DESC",
            "SEMANA": "SELECT strftime('%Y-W%W', data_transacao), COUNT(*) FROM
Transacao GROUP BY strftime('%Y-W%W', data_transacao) ORDER BY strftime('%Y-W%W',
data_transacao) DESC",
            "MÊS": "SELECT strftime('%Y-%m', data_transacao), COUNT(*) FROM Transacao
GROUP BY strftime('%Y-%m', data_transacao) ORDER BY strftime('%Y-%m', data_transacao)
DESC"
        }

        relatorio = []
        for periodo, query in queries.items():
            resultados = self._executar_query(query, fetch_all=True)
            if resultados:
                relatorio.append(f"📅 Totais por {periodo}:\n" + "\n".join([f"{data}: {total} venda(s)" for
data, total in resultados]))

        if not relatorio:
            self.show_message("Nenhuma venda encontrada para calcular os totais.")
            return

        QMessageBox.information(self, "Totais de Vendas", "\n\n".join(relatorio))

    except Exception as e:
        self.show_message(f"Erro ao calcular totais: {e}", error=True)
```

Permite excluir um cliente da base de dados, desde que ele não tenha vendas registradas. Confirma antes de remover.

```
def handle_remover_cliente(self):
    """Lida com a remoção de um cliente selecionado."""
    selected_items = self.tabela_clientes.selectedItems()
    if not selected_items:
        self.show_message("Por favor, selecione um cliente na tabela para remover.", error=True)
```

```
        return

    nome_cliente = selected_items[0].text()
    reply = QMessageBox.question(self, 'Confirmar Remoção',
                                 f"Tem a certeza que deseja remover o cliente '{nome_cliente}'?",
                                 QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

    if reply == QMessageBox.No:
        return

    try:
        id_cliente = self._get_id_por_nome('Usuario', 'id_usuario', 'nome', nome_cliente)
        if id_cliente is None:
            self.show_message("Cliente não encontrado.", error=True)
            return

        query_check = "SELECT COUNT(*) FROM Transacao WHERE id_comprador = ?"
        transacao_count = self._executar_query(query_check, (id_cliente,), fetch_one=True)[0]

        if transacao_count > 0:
            self.show_message(f"Não é possível remover '{nome_cliente}', pois possui
{transacao_count} venda(s) no histórico.", error=True)
            return

        query_delete = "DELETE FROM Usuario WHERE id_usuario = ?"
        self._executar_query(query_delete, (id_cliente,))
        self.show_message(f"Cliente '{nome_cliente}' removido com sucesso!")
        self.atualizar_clientes_ui()

    except Exception as e:
        self.show_message(f"Ocorreu um erro ao remover o cliente: {e}", error=True)
```