TEGRSI12
Guilherme Paulo
Henrique Saraiva
5097

PROJECTO FINAL – Loja Online

Tabela de Conteúdos

Introdução e Objetivos

O objetivo do projecto é desenvolver uma base de dados relacional em MySQL para a BuyPy, uma loja online de livros e eletrónica. Juntamente com a base de dados, será desenvolvida uma aplicação de back-office para gerir as operações da loja. As tarefas opcionais incluem a criação de um frontend web ou uma aplicação gráfica de back-office usando Qt.

Desta forma, podemos sintetizar os nossos conhecimentos de Python e MySQL (entre outros) num unico produto funcional.

A implementação do programa requere vários scripts, um de Python para servir de front-end do back-office para gerir o funcionamento da loja, e vários scripts de MySQL para gerir a base de dados.

Análise

Este projeto envolve vários componentes: scripts SQL para o esquema da base de dados, para preencher essa base com os dados iniciais e para definir os papéis e privilégios dos utilizadores, bem como scripts Python para a aplicação de back-office, incluindo funcionalidades de linha de comando para iniciar sessão, gerir produtos, utilizadores e respetivas encomendas.

Bases de dados:

Produtos: Cada produto (livro ou eletrónico) terá um código único, quantidade, preço, taxa de imposto, pontuação de popularidade, caminho da imagem e estado (ativo/inativo).

Livros: Os campos adicionais incluem ISBN, título, género, editora, autor e data de publicação.

Eletrónica: Os campos adicionais incluem número de série, marca, modelo, especificações e tipo.

Clientes: Utilizadores registados com detalhes como nome próprio, apelido, e-mail (nome de utilizador), palavra-passe, morada, código postal, cidade, país e número de telefone.
As palavras-passe devem ter pelo menos 6 caracteres com uma mistura de letras, dígitos e símbolos especiais.

Encomendas: Cada encomenda terá um número único, data e hora, método de envio, estado, detalhes de pagamento (número do cartão, nome do titular do cartão e data de validade).

Recomendações: Cada cliente pode ter produtos recomendados com uma data de recomendação.

Aplicação BackOffice:

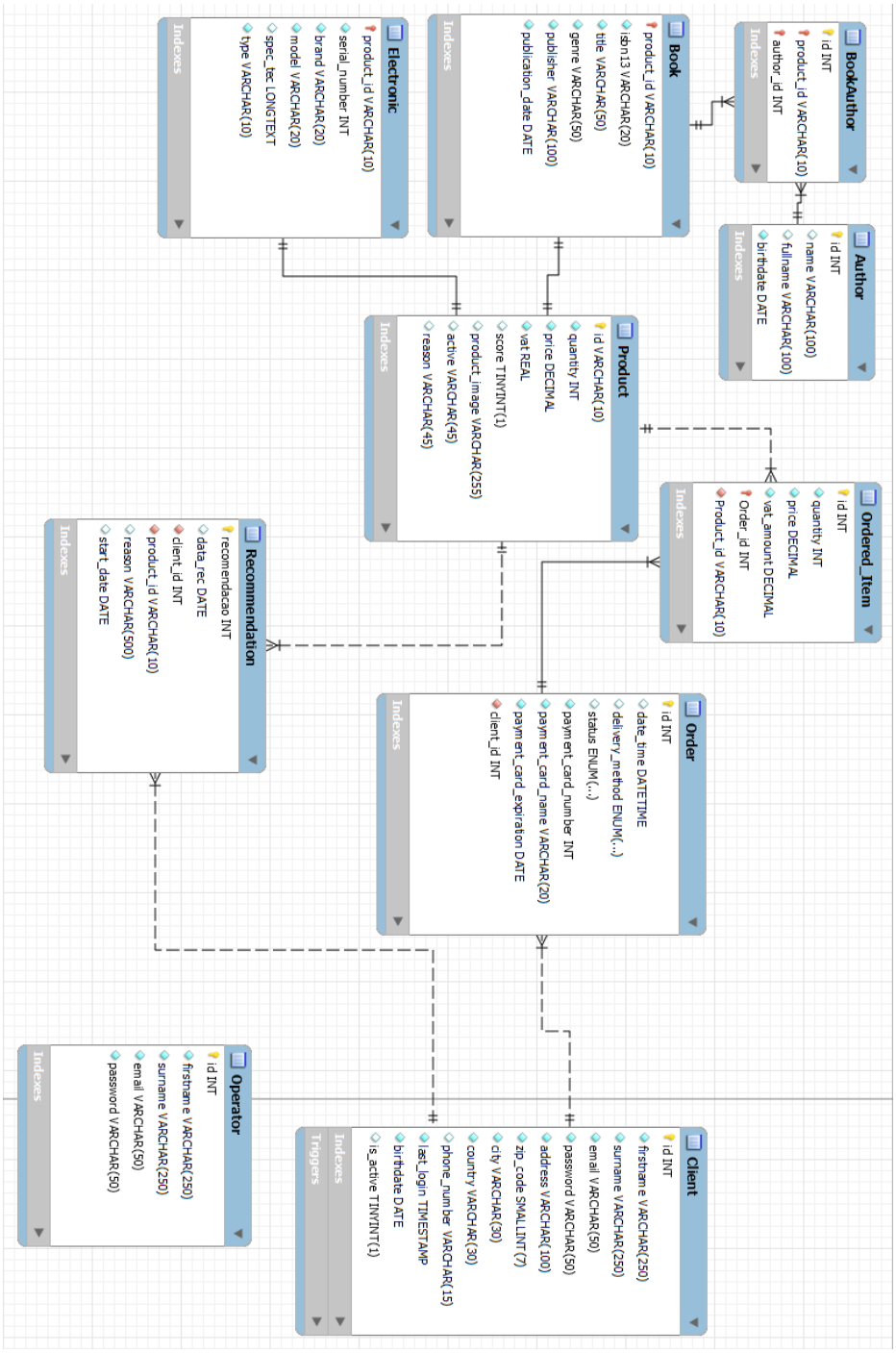
Gestão de utilizadores:
Funcionalidade de início de sessão que armazena as credenciais num ficheiro config.ini encriptado.
Pesquisa utilizadores por ID ou nome de utilizador, apresenta os detalhes do utilizador e altera o estado da conta (ativa/inativa/bloqueada).

Gestão de produtos:
Lista de produtos com filtros para tipo, quantidade e faixa de preço.
Opcional: Adicionar novos produtos e importar produtos de um ficheiro XLSX.

Gestão de encomendas:
Relatórios diários e anuais de encomendas.
Criar encomendas e adicionar produtos às encomendas.
Cálculo do montante total da encomenda.

Cópia de segurança da base de dados: Executar backups usando *mysqldump*.

Desenho e Estrutura



Utilizando o modelo de base de dados que nos foi fornecido (ver “Projeto 1 - Loja Online - Modelo Dados” na pasta docs) foi criado o seguinte esquema MySQL:

Implementação

MySQL Workbench Forward Engineering:
Utilizando o modelo da secção anterior, utilizámos o MySQL Workbench's Forward Engineering para gerar um script para criar a base de dados (que não será publicado aqui por uma questão de brevidade, consulte o ficheiro /src/buypy_tables_firstorder.sql).
Em seguida, adicionámos alguns selects/calls para verificar os processos de loja:

```
-- Testar os store procedures criados

#CALL ProductByType('Book');

#CALL DailyOrders('2024-04-26');

#CALL AnnualOrders(11, 2024);

#Select * from BuyPy.Order;
#CALL CreateOrder(12, 'urgent', 1234567890, 'Guilherme Novo', '2026-12-01');
#Select * from BuyPy.Order;
#CALL GetOrderTotal(28);
#CALL AddProductToOrder(28, 'P011', 5);
#CALL GetOrderTotal(28);
#Select * from BuyPy.Product;
#CALL AddBook('ID_DO_LIVRO', 'ISBN_DO_LIVRO', 'TÍTULO_DO_LIVRO', 'GÊNERO_DO_LIVRO', 'EDITORA_DO_LIVRO', 'DATA_DE_PUBLICAÇÃO_DO_LIVRO');
#CALL AddElec('ID_DO_PRODUTO', 'NUMERO_DE_SERIE', 'MARCA', 'MODELO', 'ESPECIFICACOES_TECNICAS', 'TIPO');
```

Store_procedures.sql (procedimentos armazenados):
Este script cria vários procedimentos armazenados para a base de dados. Estes procedimentos tratam de várias operações relacionadas com produtos, encomendas e artigos.

```
DELIMITER //

CREATE PROCEDURE ProductByType (
    IN product_type VARCHAR(10)
)
BEGIN
    IF product_type IS NULL THEN
        -- Return all products
        SELECT p.id, p.price, p.score, p.active, p.product_image,
            CASE
                WHEN e.product_id IS NOT NULL THEN 'Electronic'
                WHEN b.product_id IS NOT NULL THEN 'Book'
                ELSE NULL
            END AS type
        FROM Product p
        LEFT JOIN Electronic e ON p.id = e.product_id
        LEFT JOIN Book b ON p.id = b.product_id;
    ELSE
        -- Return products by type
        CASE product_type
            WHEN 'Electronic' THEN
                SELECT p.id, p.price, p.score, p.active, p.product_image, 'Electronic' AS type
                FROM Product p
                INNER JOIN Electronic e ON p.id = e.product_id;
            WHEN 'Book' THEN
                SELECT p.id, p.price, p.score, p.active, p.product_image, 'Book' AS type
                FROM Product p
                INNER JOIN Book b ON p.id = b.product_id;
            ELSE
                -- Invalid product type
                SELECT 'Invalid product type' AS Error;
        END CASE;
    END IF;
END//

DELIMITER ;
```

Este procedimento obtém todos os produtos com base no seu tipo (’Eletrónico’ ou ’Livro’). Se o tipo for NULL ele retorna todos os produtos, caso contrário ele retorna os produtos filtrados pelo tipo especificado.

```
DELIMITER //

CREATE PROCEDURE DailyOrders (
    IN order_date DATE
)
BEGIN
    SELECT *
    FROM `Order`
    WHERE DATE(date_time) = order_date;
END//

DELIMITER ;
```

Este procedimento obtém as encomendas efetuadas numa data especifica (order_date).

```
60  DELIMITER //
61
62  CREATE PROCEDURE AnnualOrders(IN cliente_id INT, IN ano_encomenda INT)
63  BEGIN
64      -- Seleciona todas as encomendas feitas por um cliente específico durante um ano específico
65      SELECT *
66      FROM `BuyPy`.`Order`
67      WHERE `client_id` = cliente_id AND YEAR(`date_time`) = ano_encomenda;
68  END //
69
70  DELIMITER ;
```

Este procedimento obtém todas as encomendas feitas por um determinado cliente (client_id) num determinado ano (ano_encomenda).

```
72  DELIMITER //
73
74  CREATE PROCEDURE CreateOrder(
75      IN cliente_id INT,
76      IN metodo_expedicao ENUM('regular', 'urgent'),
77      IN numero_cartao INT,
78      IN nome_cartao VARCHAR(100),
79      IN data_validade_cartao DATE
80  )
81  BEGIN
82      -- Insere uma nova encomenda com os dados fornecidos
83      INSERT INTO `BuyPy`.`Order` (
84          `date_time`,
85          `delivery_method`,
86          `status`,
87          `payment_card_number`,
88          `payment_card_name`,
89          `payment_card_expiration`,
90          `client_id`
91      )
92      VALUES (
93          NOW(),
94          metodo_expedicao,
95          'open', -- Definindo o status da encomenda como "open" por padrão
96          numero_cartao,
97          nome_cartao,
98          data_validade_cartao,
99          cliente_id
100     );
101 END //
102
103 DELIMITER ;
```

Este procedimento cria uma encomenda com os dados fornecidos: metodo_expedicao, numero_cartao, nome_cartao, data_validade_cartao e cliente_id.

```
105  DELIMITER //
106
107  CREATE PROCEDURE `GetOrderTotal` (IN order_id INT)
108  BEGIN
109      DECLARE total DECIMAL(10, 2);
110
111      -- Calcula o montante total da encomenda
112      SELECT SUM(price * quantity) INTO total
113      FROM Ordered_Item
114      WHERE Order_id = order_id;
115
116      -- Devolve o montante total
117      SELECT total;
118  END//
119
120  DELIMITER ;
```

Este procedimento devolve o montante total de uma encomenda (order_id).

```
122  DELIMITER //
123
124  CREATE PROCEDURE AddProductToOrder(
125      IN order_id INT,
126      IN product_id VARCHAR(10),
127      IN quantity INT
128  )
129  BEGIN
130      DECLARE total_price DECIMAL(10, 2);
131
132      -- Calcular o preço total do produto
133      SELECT price * quantity INTO total_price
134      FROM Product
135      WHERE id = product_id;
136
137      -- Inserir o item da encomenda na tabela Ordered_Item
138      INSERT INTO Ordered_Item (quantity, price, Order_id, Product_id)
139      VALUES (quantity, total_price, order_id, product_id);
140
141      -- Atualizar a quantidade de produtos disponíveis na tabela Product
142      UPDATE Product
143      SET quantity = quantity - quantity
144      WHERE id = product_id;
145  END //
146
147  DELIMITER ;
```

Este procedimento adiciona um produto (product_id) a uma encomenda (order_id) e atualiza a quantidade do produto (quantity).

```
150  DELIMITER //
151
152  CREATE PROCEDURE AddBook(
153      IN book_id VARCHAR(10),
154      IN isbn13 VARCHAR(20),
155      IN title VARCHAR(50),
156      IN genre VARCHAR(50),
157      IN publisher VARCHAR(100),
158      IN publication_date DATE
159  )
160  BEGIN
161      INSERT INTO Book (product_id, isbn13, title, genre, publisher, publication_date)
162      VALUES (book_id, isbn13, title, genre, publisher, publication_date);
163  END //
164
165  DELIMITER ;
166  DELIMITER //
167
168  CREATE PROCEDURE AddElec(
169      IN product_id VARCHAR(10),
170      IN serial_number INT,
171      IN brand VARCHAR(20),
172      IN model VARCHAR(20),
173      IN spec_tec LONGTEXT,
174      IN type VARCHAR(10)
175  )
176  BEGIN
177      INSERT INTO Electronic (product_id, serial_number, brand, model, spec_tec, type)
178      VALUES (product_id, serial_number, brand, model, spec_tec, type);
179  END //
180
181  DELIMITER ;
```

Estes dois procedimentos adicionam um novo livro ou um produto eletrónico às suas respetivas tabelas (’Book’ ou ’Electronic’).

Users_script.sql (script de utilizadores):

```sql
-- Crie o usuário WEB_CLIENT
CREATE USER 'WEB_CLIENT'@'%' IDENTIFIED BY 'Lmxy20#a';

-- Conceda permissões de leitura para todas as tabelas do sistema
GRANT SELECT ON *.* TO 'WEB_CLIENT'@'%';

-- Conceda permissões de escrita e atualização para todas as tabelas relacionadas com dados dos clientes e encomendas
GRANT INSERT, UPDATE ON BuyPy.Client TO 'WEB_CLIENT'@'%';
GRANT INSERT, UPDATE ON BuyPy.Order TO 'WEB_CLIENT'@'%';

-- Conceda permissões para apagar dados nas tabelas Order e Ordered_Item
GRANT DELETE ON BuyPy.Order TO 'WEB_CLIENT'@'%';
GRANT DELETE ON BuyPy.Ordered_Item TO 'WEB_CLIENT'@'%';


-- Conceda permissões para atualizar o campo quantidade na tabela Product
GRANT UPDATE (quantity) ON BuyPy.Product TO 'WEB_CLIENT'@'%';

-- Conceda permissões para executar os procedimentos CreateOrder, GetOrderTotal e AddProductToOrder
GRANT EXECUTE ON PROCEDURE CreateOrder TO 'WEB_CLIENT'@'%';
GRANT EXECUTE ON PROCEDURE GetOrderTotal TO 'WEB_CLIENT'@'%';
GRANT EXECUTE ON PROCEDURE AddProductToOrder TO 'WEB_CLIENT'@'%';

-- ----------------------------------------------------------------------------------------------------------------------------

-- Criar o usuário BUYDB_OPERATOR
CREATE USER 'BUYDB_OPERATOR'@'%' IDENTIFIED BY 'Lmxy20#a';

-- Conceder permissões para leitura, escrita, atualização e remoção de dados de todas as tabelas
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'BUYDB_OPERATOR'@'%';

-- Conceder permissões para executar todos os procedimentos armazenados
GRANT EXECUTE ON PROCEDURE *.* TO 'BUYDB_OPERATOR'@'%';

-- Criar o usuário BUYDB_ADMIN
CREATE USER 'BUYDB_ADMIN'@'%' IDENTIFIED BY 'Lmxy20#a';

-- Conceder permissões para leitura, escrita, atualização e remoção de dados de todas as tabelas
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'BUYDB_ADMIN'@'%';

-- Conceder permissões para executar todos os procedimentos armazenados
GRANT EXECUTE ON PROCEDURE *.* TO 'BUYDB_ADMIN'@'%';


GRANT ALL PRIVILEGES ON BUYPY.* TO 'BUYDB_ADMIN'@'%';
```

A função deste script é criar três utilizadores (WEB_CLIENT, BUYDB_OPERATOR e BUYDB_ADMIN) com diferentes níveis de direitos de acesso. Todos utilizando a palavra passe especificada no enunciado (Lmxy20#a).

O user WEB_CLIENT tem permissão para ler todas as tabelas na base de dados e fazer INSERT e UPDATE nas tabelas ‘Client’ (cliente) e ‘Order’ (encomenda), permissão para DELETE na tabela de encomendas e de itens encomendados, e de UPDATE na tabela ‘Product’ (de produtos) mas apenas a quantidade. Também tem permissão para executar certos procedures (CreateOrder, GetOrderTotal e AddProductToOrder).

O user BUYDB_OPERATOR tem permissão de CREATE, READ, UPDATE e DELETE para todas as tabelas em todas as bases de dados. Também tem permissão para executar todos os stored procedures.

O user BUYDB_ADMIN tem as mesmas permissões que o BUYDB_OPERATOR, mas também tem privilégios administrativos na base de dados BuyPy.

Importação dos módulos (import):

```
1   import pymysql.cursors
2   import configparser
3   import re
4   import subprocess
5   import os
6   import sys
```

pymysql.cursors: Permite a interação com bases de dados MySQL.

configparser: Analisa arquivos de configuração.
Para o config.ini.

re: Fornece operações de correspondência de expressões regulares (regex).
Para os requisitos da palavra-passe.

subprocess: Executa novos applicativos ou programas através de novos processos.
Para a função de backup.

os: Fornece funções para interagir com o sistema operacional.

sys: Fornece acesso a algumas variáveis usadas ou mantidas pelo interpretador Python.

conectar_bd (Ligação á base de dados):

```
 8   def conectar_bd(config_path):
 9       try:
10           config = configparser.ConfigParser()
11           config.read(config_path)
12           db_config = config['DATABASE']
13
14           conexao = pymysql.connect(
15               host=db_config['host'],
16               user=db_config['user'],
17               password=db_config['password'],
18               database=db_config['database']
19           )
20           return conexao
21       except Exception as e:
22           print(f"Erro ao conectar ao banco de dados: {e}")
23           sys.exit(1)
```

Esta função lê as credenciais da base de dados a partir de um ficheiro e estabelece uma ligação á base de dados MySQL. Se falhar, imprime uma mensagem de erro e sai do programa.

Login (Iniciar sessão):

```python
def login(config_path):
    email = input("Insira o email: ")
    password = input("Insira a password: ")

    conexao = conectar_bd(config_path)
    try:
        with conexao.cursor() as cursor:
            sql = "SELECT * FROM Operator WHERE email=%s AND password=%s"
            cursor.execute(sql, (email, password))
            resultado = cursor.fetchone()
            if resultado:
                print("Login bem-sucedido!")
                return resultado
            else:
                print("Nome de usuário ou senha inválidos.")
                return None
    except Exception as e:
        print(f"Erro ao executar a consulta SQL: {e}")
    finally:
        conexao.close()
```

Esta função trata do início de sessão do utilizador. Pede ao utilizador o e-mail e a palavra-passe, liga-se à base de dados e verifica se as credenciais são válidas consultando a tabela Operator. Se forem válidas, devolve as informações do utilizador; caso contrário, imprime uma mensagem de erro.

Pesquisar usuário:

```python
def pesquisar_usuario(config_path):
    idoremail = input("Digite o ID do usuário ou email do utilizador: ")
    conexao = conectar_bd(config_path)

    try:
        with conexao.cursor() as cursor:
            sql = "SELECT * FROM Client WHERE id=%s OR email=%s"
            cursor.execute(sql, (idoremail, idoremail))
            resultado = cursor.fetchall()

            if resultado:
                print("Usuários encontrados:")
                larguras = [max(len(str(field)) for field in col) for col in zip(*resultado)]
                nomes_colunas = [column[0] for column in cursor.description]

                print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")
                print("|" + "|".join(f" {campo:{largura}} " for campo, largura in zip(nomes_colunas, larguras)) + "|")
                print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")

                for row in resultado:
                    print("|" + "|".join(f" {str(valor):{largura}} " for valor, largura in zip(row, larguras)) + "|")
                    print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")

                acao = input("Você deseja (b)loquear ou (d)esbloquear estes usuários? ")
                if acao.lower() == 'b':
                    # Implementar lógica de bloqueio aqui
                    print("Contas dos usuários bloqueadas com sucesso.")
                elif acao.lower() == 'd':
                    # Implementar lógica de desbloqueio aqui
                    print("Contas dos usuários desbloqueadas com sucesso.")
                else:
                    print("Opção inválida.")
            else:
                print("Nenhum usuário encontrado.")
    finally:
        conexao.close()
```

Esta função procura um utilizador por ID ou e-mail. Liga-se à base de dados, executa uma consulta e apresenta os resultados. Também permite ao utilizador bloquear ou desbloquear os utilizadores encontrados.

Listar produtos:

```python
def listar_produtos(config_path):
    conexao = conectar_bd(config_path)
    try:
        with conexao.cursor() as cursor:
            tipo_produto = input("Digite o tipo de produto (Book ou Electronic) ou deixe em branco para todos: ")
            quantidade_min = input("Digite a quantidade mínima ou deixe em branco para todos: ")
            quantidade_max = input("Digite a quantidade máxima ou deixe em branco para todos: ")
            preco_min = input("Digite o preço mínimo ou deixe em branco para todos: ")
            preco_max = input("Digite o preço máximo ou deixe em branco para todos: ")

            sql = "SELECT * FROM Product WHERE 1=1"
            params = []

            if tipo_produto:
                sql += " AND product_type = %s"
                params.append(tipo_produto)
            if quantidade_min:
                sql += " AND quantity >= %s"
                params.append(quantidade_min)
            if quantidade_max:
                sql += " AND quantity <= %s"
                params.append(quantidade_max)
            if preco_min:
                sql += " AND price >= %s"
                params.append(preco_min)
            if preco_max:
                sql += " AND price <= %s"
                params.append(preco_max)

            cursor.execute(sql, tuple(params))
            resultado = cursor.fetchall()
            if resultado:
                print("Produtos encontrados:")
                larguras = [max(len(str(field)) for field in col) for col in zip(*resultado)]
                nomes_colunas = [column[0] for column in cursor.description]

                print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")
                print("|" + "|".join(f" {campo:{largura}} " for campo, largura in zip(nomes_colunas, larguras)) + "|")
                print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")

                for row in resultado:
                    print("|" + "|".join(f" {str(valor):{largura}} " for valor, largura in zip(row, larguras)) + "|")
                    print("+" + "+".join("-" * (largura + 2) for largura in larguras) + "+")
            else:
                print("Nenhum produto encontrado com os critérios fornecidos.")
    finally:
        conexao.close()
```

Esta função lista produtos com base em filtros fornecidos pelo utilizador (tipo de produto, intervalo de quantidades e intervalo de preços). Constrói dinamicamente uma consulta SQL com base nos dados fornecidos e apresenta os resultados em formato de tabela.

Backup (Executar cópia de segurança):

```python
def executar_backup(config_path):
    config = configparser.ConfigParser()
    config.read(config_path)
    db_config = config['DATABASE']

    nome_arquivo = input("Digite o nome do arquivo de backup: ")

    comando = f"mysqldump -u {db_config['user']} -p{db_config['password']} {db_config['database']} > {nome_arquivo}.sql"

    try:
        subprocess.run(comando, shell=True, check=True)
        print("Backup concluído com sucesso!")
    except subprocess.CalledProcessError as e:
        print(f"Erro ao executar o backup: {e}")
```

Esta função executa um backup do banco de dados usando *mysqldump*. Ela constrói o comando usando as credenciais do banco de dados do arquivo de configuração e executa-o usando *subprocess.run*. Se o comando for bem-sucedido, imprime uma mensagem de sucesso; caso contrário, imprime um erro.

Registar produto:

```python
def registar_produto(config_path):
    conexao = conectar_bd(config_path)
    try:
        with conexao.cursor() as cursor:
            nome = input("Digite o nome do produto: ")
            descricao = input("Digite a descrição do produto: ")
            preco = input("Digite o preço do produto: ")
            quantidade = input("Digite a quantidade do produto: ")
            tipo_produto = input("Digite o tipo do produto (Book ou Electronic): ")

            sql = "INSERT INTO Product (name, description, price, quantity, product_type) VALUES (%s, %s, %s, %s, %s)"
            cursor.execute(sql, (nome, descricao, preco, quantidade, tipo_produto))
            conexao.commit()
            print("Produto registrado com sucesso!")
    except Exception as e:
        print(f"Erro ao registrar o produto: {e}")
    finally:
        conexao.close()
```

Esta função regista um novo produto na base de dados. Pede ao utilizador os detalhes do produto, constrói uma consulta SQL INSERT e executa-a. Se for bem-sucedida, confirma a transação e imprime uma mensagem de sucesso.

Menu principal:

```
165  def menu_principal(utilizador, config_path):
166      while True:
167          print("\nMenu Principal:")
168          print("1. Pesquisar Utilizador")
169          print("2. Listar Produtos")
170          print("3. Registar Produtos")
171          print("4. Backup")
172          print("5. Logout")
173
174          escolha = input("Escolha uma opção (1, 2, 3, 4 ou 5): ")
175
176          if re.match(r'1', escolha):
177              pesquisar_usuario(config_path)
178          elif re.match(r'2', escolha):
179              listar_produtos(config_path)
180          elif re.match(r'3', escolha):
181              registar_produto(config_path)
182          elif re.match(r'4', escolha):
183              executar_backup(config_path)
184          elif re.match(r'5', escolha):
185              print("ogout...")
186              return
187          else:
188              print("Escolha inválida. Por favor, tente novamente.")
```

Esta função apresenta o menu principal e trata a entrada do utilizador para diferentes opções: procurar utilizadores, listar produtos, registar um produto, executar uma cópia de segurança ou terminar a sessão.

Execução principal:

```
190  if __name__ == "__main__":
191      config_path = input("Digite o caminho para o arquivo config.ini (pressione Enter para usar o diretório atual): ").strip()
192
193      # If the input is empty, set the config file path to the current directory
194      if not config_path:
195          config_path = "config.ini"
196
197      abs_config_path = os.path.abspath(config_path)
198
199      if not os.path.isfile(abs_config_path):
200          print(f"O arquivo de configuração {abs_config_path} não foi encontrado.")
201          sys.exit(1)
202
203      while True:
204          utilizador = login(abs_config_path)
205          if utilizador:
206              print(f"Bem-vindo, {utilizador[1]}")  # Adjusted to print the name or appropriate field from the user record
207              menu_principal(utilizador, abs_config_path)
208          else:
209              print("Falha no login. Tente novamente.")
210
```

Esta é a execução do script principal. Pede ao utilizador o caminho para o ficheiro de configuração, verifica se existe e entra num ciclo que trata do início de sessão do utilizador e das operações do menu principal.

Conclusão

O código Python fornecido em conjunto com os scripts de SQL forma a infraestrutura de backend da loja online BuyPy. É uma integração eficaz de Python com uma base de dados MySQL, permitindo uma maneira eficiente e dinâmica para os utilizadores fazerem as suas tarefas de gestão de dados.

O script serve para facilitar operações críticas como a autenticação de utilizadores, processar encomendas, e gestão de produtos. Graças ao PyMySQL é simples conectar á base de dados e executar os procedimentos armazenados.

A integração entre o Python e o MySQL através deste script estabelece um sistema robusto e eficiente, fornecendo uma base para crescimento e melhorias futuras (ex: uma interface gráfica ou web frontend).