



## **RELATÓRIO**

Projeto BuyPy



UFCD 5091

Diogo Pereira e June Pinto

<https://github.com/DiogoPereira43/BuyPy.git>



## Conteúdo

Introdução e Objetivos.....	3
Desenho da Base de Dados .....	4
Criação da Base de Dados.....	5
Stored Procedures e Utilizadores .....	7
BackOffice .....	11
Conclusão.....	17
Referências .....	18



## **Introdução e Objetivos**

O presente documento constitui o relatório do projeto BuyPy – BackOffice para Loja Online, o trabalho final da unidade curricular 5107. Este projeto alia os conhecimentos de Python objeto de estudo nas unidades anterior com a organização e sistematização de bases de dados através da linguagem SQL, concretizando-se na implementação de uma base de dados relacional (SQL) com uma aplicação BackOffice (Python) que gerem uma loja online ficcional que comercializa quer livros que produtos eletrónicos.

O projeto subdivide-se em duas componentes que correspondem a etapas da sua implementação: a base de dados em si, para a qual criámos um desenho, corrigimos de acordo com indicações do professor e depois implementámos, procurando garantir a conformidade dos tipos de dados pedidos e a integridade da relação entre tabelas. Estando esta consolidada, podemos passar à construção das stored procedures e aplicação de BackOffice em Python que permitirá operações pedidas como a gestão de utilizadores, listagem de produtos e atualização de dados.

Apresentamos de seguida o projeto passo a passo, começando pelo desenho do modelo dos dados, a sua correção e implementação das bases de dados, a gestão de utilizadores e permissões e por fim o desenvolvimento da aplicação BackOffice.



## Desenho da Base de Dados

O primeiro desafio colocado foi a conceção de uma proposta do desenho da base de dados de acordo com uma descrição escrita da mesma. Esta foi a nossa sugestão:

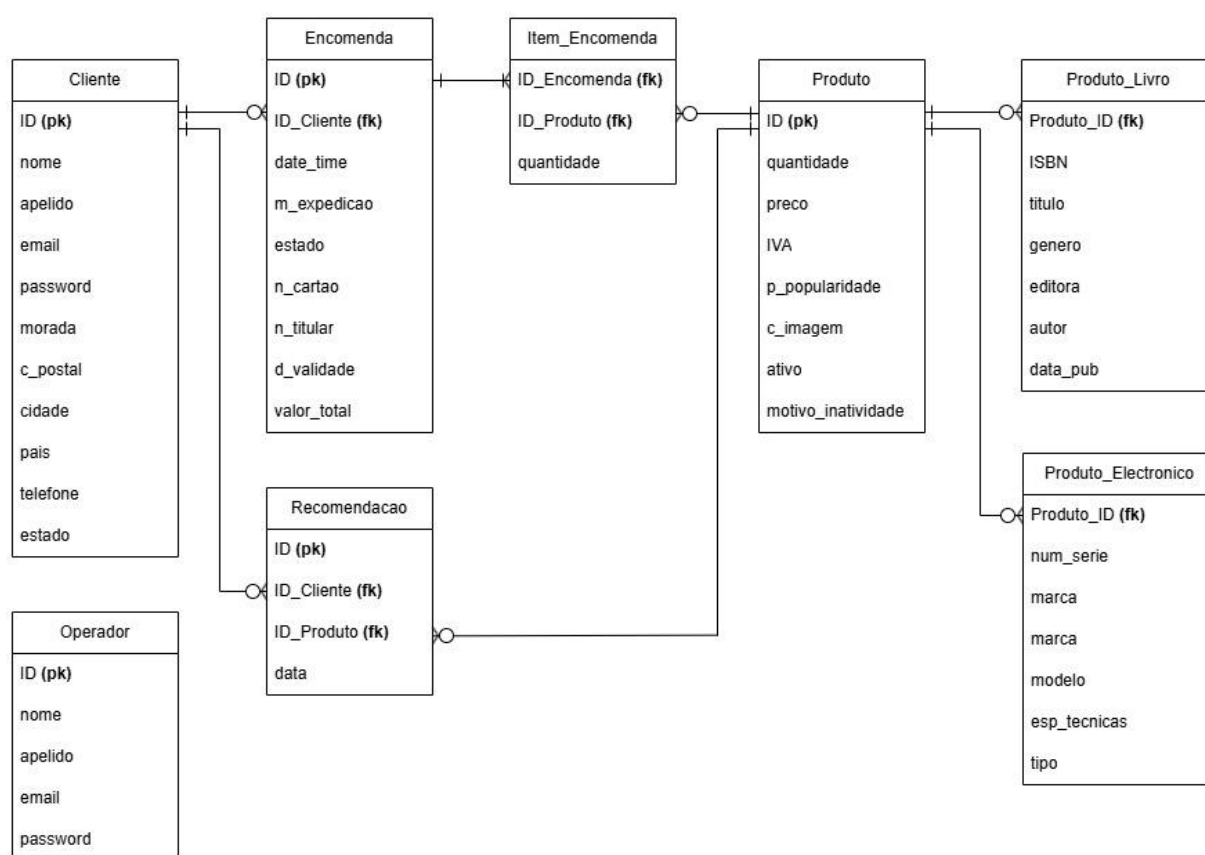


Figura 1 – Proposta do Desenho da Base de Dados

A proposta lançada depois pelo professor era muito semelhante à nossa, melhorando, contudo, o registo de livros e autores ao separar os mesmos em duas tabelas com mais uma auxiliar. Faz sentido, uma vez que um mesmo autor pode ter vários livros, facilitando esse registo e ainda as pesquisas por autor.

Tendo esta base consolidada, passámos à implementação em SQL do esquema corrigido.



## Criação da Base de Dados

Foi fornecido também uma sugestão da tipologia dos dados de cada campo. Tendo em conta esta, o primeiro passo foi criar em SQL as tabelas e a sua inter-relação. Usando o comando create table, avançámos cuidadosamente, coluna por coluna inserindo os dados de acordo com a formatação sugerida. Incluímos ainda checks de forma a validar dados sem necessitar de stored procedures, conseguindo assim incluir de forma mais simples as validações necessárias pedidas:

```
CREATE TABLE Ordered_Item(  
    ordered_item_id int primary key auto_increment,  
    order_id int not null,  
    product_id varchar(10) not null,  
    quantity int not null check (quantity >= 0), #validacao do positivo tem de ser em stored procedure  
    price decimal(19, 4) not null check (price >= 0), #validacao do positivo tem de ser em stored procedure  
    vat_amount decimal(19, 4) not null check (vat_amount >= 0 AND vat_amount <= 100), # tem de ser numero positivo, usamos o check entre numeros  
    foreign key (order_id) references Orders(order_id),  
    foreign key (product_id) references Product(product_id)  
);
```

Figura 2 – Implementação da Tabela Ordered\_Item

Este mesmo processo foi seguido coluna por coluna, tabela por tabela, tendo o cuidado ainda de criar as que não têm dependências (foreign keys) de outras e só criando essas segundas quando já existem os dados que vão receber.

```
#Inserir valores  
INSERT INTO Clients (client_id, fullname, email, password_client, address, zip_code, city, country, phone_number, last_login, birthdate, is_active)  
VALUES  
(1, 'Diogo Pedro Silva', 'diogo.silva@email.com', 'Pass#123', 'Rua das Porcas, nº10', 1234, 'Lisboa', 'Portugal', '912345678', '2025-07-26 14:00:00', '1990-01-15', TRUE),  
(2, 'Ana Sofia Costa', 'ana.costa@email.com', 'Test$456', 'Avenida Central, nº25', 5678, 'Porto', 'Portugal', '923456789', '2025-07-26 14:05:00', '1985-03-22', TRUE),  
(3, 'Pedro Miguel Santos', 'pedro.santos@email.com', 'Abc?789', 'Rua do Sol, nº50', 9012, 'Portugal', '934567890', '2025-07-26 14:10:00', '1995-07-10', FALSE),  
(4, 'Maria Clara Oliveira', 'maria.oliveira@email.com', 'XyzX101', 'Praça da Liberdade, nº15', 3456, 'Coimbra', 'Portugal', '945678901', '2025-07-26 14:15:00', '1988-11-30', TRUE),  
(5, 'Lucas André Almeida', 'lucas.almeida@email.com', 'Qwe!202', 'Rua da Carlota, nº30', 7890, 'Braga', 'Portugal', '956789012', '2025-07-26 14:20:00', '1992-05-05', FALSE);  
SELECT * FROM Clients;  
  
INSERT INTO Orders (order_id, date_time, delivery_method, order_status, payment_card_number, payment_card_name, payment_card_expiration, client_id)  
VALUES  
(1, '2025-07-25 10:00:00', 'regular', 'open', 1234567890123456, 'João Silva', '2027-12-31', 1),  
(2, '2025-07-25 15:30:00', 'express', 'open', 9876543210987654, 'Ana Costa', '2028-01-31', 2),  
(3, '2025-07-26 09:00:00', 'regular', 'open', 4567891234567890, 'Pedro Santos', '2027-11-30', 3),  
(4, '2025-07-26 12:45:00', 'express', 'open', 3210987654321098, 'Maria Oliveira', '2028-06-30', 4),  
(5, '2025-07-26 16:20:00', 'regular', 'open', 6543219876543210, 'Lucas Almeida', '2027-09-30', 5);  
SELECT * FROM Orders;  
  
INSERT INTO Product (product_id, quantity, price, vat, score, product_image, is_active, reason)  
VALUES #? que o Product_ID é VARCHAR, podemos fazer algo alfanumerico, em empresas acho que é normal usar P001 - P002 por aí fora. Assim tb fica unico  
( 'P001', 10, 999.99, 23.00, 85, 'smartphone.jpg', TRUE, NULL),  
( 'P002', 5, 29.99, 6.00, 90, 'novel.jpg', TRUE, NULL),  
( 'P003', 15, 4999.99, 23.00, 95, 'laptop.jpg', FALSE, 'Discontinued model'),  
( 'P004', 8, 19.99, 6.00, 88, 'book.jpg', TRUE, NULL),  
( 'P005', 20, 149.99, 23.00, 82, 'tablet.jpg', FALSE, 'Out of stock'),  
( 'P006', 12, 199.99, 23.00, 87, 'headphones.jpg', TRUE, NULL),  
( 'P007', 7, 1299.99, 23.00, 92, 'tv.jpg', TRUE, NULL),  
( 'P008', 10, 24.99, 6.00, 85, 'scifi.jpg', TRUE, NULL),  
( 'P009', 6, 22.99, 6.00, 89, 'mystery.jpg', TRUE, NULL),  
( 'P010', 8, 27.99, 6.00, 91, 'fantasy.jpg', TRUE, NULL);  
SELECT * FROM Product;
```

Figura 3 - Inserção de Dados



## Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI08

A criação dos dados não levantou particular desafio, sendo feito com insert de values cuidadosamente a tipologia de dados que tínhamos anteriormente definidos.

Foram ainda incluídos triggers para validar informações, como o email, este feito através de uma expressão regular que analisa o texto recebido.

```
# Trigger para verificar cliente email antes de um insert e update
DROP TRIGGER IF EXISTS validar_cliente_email;
DROP TRIGGER IF EXISTS validar_client_email_update;
DELIMITER //
CREATE TRIGGER validar_client_email
BEFORE INSERT ON Clients
FOR EACH ROW
BEGIN
    IF NEW.email NOT REGEXP '^([a-z0-9!#$%&'"+/=?^_`{|}~]+(\.[a-z0-9!#$%&'"+/=?^_`{|}~]+)*)@([a-z0-9]([a-z0-9-]*[a-z0-9]))?(\.[a-z0-9]([a-z0-9-]*[a-z0-9]))?$'
    THEN SIGNAL SQLSTATE '45000' # 45 uma classe reservado para erros definidos pelo usuário, li que é uma boa prática
    SET MESSAGE_TEXT = 'Erro: Email do cliente inválido. Não seja tóto.';
    END IF;
END //
DELIMITER ;
```

Figura 4 - Validação do Email

Esta validação é repetida para as operações que envolvem o email, como as suas inserções e updates, e é reutilizada conforme necessário. Após esta, criámos um sistema de validação das passwords, garantindo que a mesma não é inferior a 6 caracteres nem contém caracteres que não seriam válidos.

```
# Validar password de Cliente e Operator

DELIMITER //
CREATE TRIGGER validar_client_password
BEFORE INSERT ON Clients
FOR EACH ROW
BEGIN
    IF LENGTH(NEW.password_client) < 6
    OR NEW.password_client NOT REGEXP '[0-9]'
    OR NEW.password_client NOT REGEXP '[a-z]'
    OR NEW.password_client NOT REGEXP '[A-Z]'
    OR NEW.password_client NOT REGEXP '[$#?%]'
    THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: Senha do cliente inválida. És tóto.';
    END IF;
END //
DELIMITER ;
```

Figura 5 - Validação da Password



## Stored Procedures e Utilizadores

De seguida pudemos avançar para a criação das Stored Procedures. Começámos por criar a que pesquisa o produto por tipo:

```
# 3 Storage Procedures

# EXTRA ProductbyType
DROP PROCEDURE IF EXISTS GetProductByType;
DELIMITER //
CREATE PROCEDURE GetProductByType (IN input_type VARCHAR(50))
BEGIN
    SELECT
        Product.product_id AS 'ID do Produto',
        Product.price AS 'Preço do Produto',
        Product.score AS 'Score do Produto',
        Recommendation.reason AS 'Motivo da Recomendação',
        Product.is_active AS 'Está ativo?',
        Electronic.electronic_type AS 'Tipo de Eletrónico',
        Book.genre AS 'Genero de Livro'
    FROM Product
    LEFT JOIN Recommendation ON Product.product_id = Recommendation.product_id #Usamos LEFT JOIN em vez de INNER se não o SELECT mostra nada, porque nao existe um produto que tenha registo em ambos electronic_type nem
    LEFT JOIN Electronic ON Product.product_id = Electronic.product_id
    LEFT JOIN Book ON Product.product_id = Book.product_id
    WHERE input_type IS NULL
    OR Electronic.electronic_type = input_type
    OR Book.genre = input_type; # Retorna todos os produtos se input_type for NULL
                                # ou caso o input for correspondente a electronic_type
                                # ou caso o input for correspondente a Book.Genero
END //
DELIMITER ;

# Testagem
CALL GetProductByType ('smartphone');
```

Figura 6 - Stored Procedure para Pesquisa por Tipo

A Stored Procedure recebe em Varchar o nome da categoria em questão e procura se a identifica quer nos electronic.types quer nos book.genre e devolve o ID, preço, pontuação, motivo da recomendação e disponibilidade da loja dos produtos em questão. Acrescentámos ainda a hipótese de poder não recebendo nenhum input devolver todos os produtos.

Depois avançámos para a criação da pesquisa de encomendas num dado dia (Daily Orders) e de um determinado cliente num ano dado por nós também (Annual Orders). O procedimento para ambas é muito semelhante, realizando uma pesquisa em que associa os dados da encomenda (Order) à identificação do cliente (Clients.fullname), filtrando no primeiro caso por uma data específica, e no segundo por Id de cliente e apenas o ano no campo da data.



```
# EXTRA DailyOrders
DROP PROCEDURE IF EXISTS DailyOrders;
DELIMITER //
CREATE PROCEDURE DailyOrders (IN input_type date)
BEGIN
SELECT
Orders.order_id AS 'ID da Encomenda',
Orders.date_time AS 'Data Encomenda',
Orders.delivery_method AS 'Metodo de envio',
Orders.order_status AS 'Status da encomenda',
Orders.payment_card_number AS 'Numero de cartão',
Orders.payment_card_name AS 'Nome do Cartão',
Orders.payment_card_expiration AS 'Expiração do cartão',
Clients.fullname AS 'Nome do cliente'
FROM Orders
INNER JOIN Clients ON Orders.client_id = Clients.client_id
WHERE DATE (Orders.date_time) = input_type;
END //
DELIMITER ;

# Testagem
CALL DailyOrders ('2025-07-25');

# EXTRA AnnualOrders
DROP PROCEDURE IF EXISTS AnnualOrders;
DELIMITER //
CREATE PROCEDURE AnnualOrders (IN input_client_id INT, IN input_year INT)
BEGIN SELECT
Orders.order_id AS 'ID da Encomenda',
Orders.date_time AS 'Data Encomenda',
Orders.delivery_method AS 'Metodo de envio',
Orders.order_status AS 'Status da encomenda',
Orders.payment_card_number AS 'Numero de cartão',
Orders.payment_card_name AS 'Nome do Cartão',
Orders.payment_card_expiration AS 'Expiração do cartão',
Clients.fullname AS 'Nome do cliente'
FROM Orders
INNER JOIN Clients ON Orders.client_id = Clients.client_id
WHERE Orders.client_id = input_client_id
AND YEAR (Orders.date_time) = input_year;
END //
DELIMITER ;

# Testagem
CALL AnnualOrders ('1' , '2025');
```

Figura 7 - Stored Procedures de Pesquisa de Encomenda por Data ou por Ano e Cliente

O enunciado pedia ainda de seguida a criação de um a stored procedure que permitisse inserir os dados de uma nova encomenda, tarefa que considerámos mais simples de realizar:





---

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI08

---

```
# CreateOrder
DROP PROCEDURE IF EXISTS CreateOrder;
DELIMITER //
CREATE PROCEDURE CreateOrder (
  IN input_client_id INT,
  IN input_delivery_method ENUM('regular', 'urgent'),
  IN input_payment_card_number BIGINT,
  IN input_payment_card_name VARCHAR(20),
  IN input_payment_card_expiration DATE)
BEGIN INSERT INTO Orders (client_id,
  delivery_method,
  payment_card_number,
  payment_card_name,
  payment_card_expiration)
VALUES
  (input_client_id, input_delivery_method, input_payment_card_number, input_payment_card_name, input_payment_card_expiration);
END //
DELIMITER ;

# Testagem
CALL CreateOrder(1, 'urgent', 1234567890123456, 'João Silva', '2027-12-31');
SELECT * FROM Orders;
```

*Figura 8 - Stored Procedure para Criar Registo de Order*

A Stored Procedure recebe todos os dados necessários para um novo registo e guarda-os na tabela através de um Insert, como fizemos inicialmente com os nossos dados.

Por fim, para terminar as Stored Procedures, criámos uma última que calcula o montante final da encomenda tendo em conta o preço de cada produto, a quantidade pedida e ainda calcula o valor do IVA, devolvendo a soma destes valores.

```
# GetOrderTotal
DROP PROCEDURE IF EXISTS GetOrderTotal;
DELIMITER //
CREATE PROCEDURE GetOrderTotal (IN input_order_id INT)
BEGIN
  SELECT
    SUM(price * quantity * (1 + vat_amount / 100)) AS 'Montante Total da encomenda' # Temos de usar 1 + para incluir o preço base, senão isto só conta o valor do IVA
  FROM Ordered_Item
  WHERE order_id = input_order_id;
END //
DELIMITER ;

# Testagem
CALL GetOrderTotal(1);
```

*Figura 9 - Stored Procedure para Calcular Valor Total da Encomenda*

Demos assim por concluídas as Stored Procedures, passando à criação dos utilizadores. Para este passo, definimos em primeiro lugar:



```
DROP USER IF EXISTS 'WEB_CLIENT'@'localhost';
DROP USER IF EXISTS 'BUYDB_OPERATOR'@'localhost';
DROP USER IF EXISTS 'BUYDB_ADMIN'@'localhost';
# Criar utilizador WEB_CLIENT
CREATE USER 'WEB_CLIENT'@'localhost' IDENTIFIED BY 'Limaxy20#a';

GRANT SELECT ON BuyPy.Clients TO 'WEB_CLIENT'@'localhost';
GRANT SELECT ON BuyPy.Operator TO 'WEB_CLIENT'@'localhost';
GRANT SELECT ON BuyPy.Orders TO 'WEB_CLIENT'@'localhost';
GRANT SELECT ON BuyPy.Ordered_Item TO 'WEB_CLIENT'@'localhost';
GRANT SELECT ON BuyPy.Product TO 'WEB_CLIENT'@'localhost';

GRANT INSERT, UPDATE ON BuyPy.Clients TO 'WEB_CLIENT'@'localhost';
GRANT INSERT, UPDATE ON BuyPy.Orders TO 'WEB_CLIENT'@'localhost';
GRANT INSERT, UPDATE ON BuyPy.Ordered_Item TO 'WEB_CLIENT'@'localhost';

GRANT DELETE ON BuyPy.Orders TO 'WEB_CLIENT'@'localhost';
GRANT DELETE ON BuyPy.Ordered_Item TO 'WEB_CLIENT'@'localhost';

GRANT UPDATE (quantity) ON BuyPy.Product TO 'WEB_CLIENT'@'localhost';

GRANT EXECUTE ON PROCEDURE BuyPy.CreateOrder TO 'WEB_CLIENT'@'localhost';
GRANT EXECUTE ON PROCEDURE BuyPy.GetOrderTotal TO 'WEB_CLIENT'@'localhost';

# Criar utilizador BUYDB_OPERATOR
CREATE USER 'BUYDB_OPERATOR'@'localhost' IDENTIFIED BY 'Limaxy20#a';

GRANT SELECT, INSERT, UPDATE, DELETE ON BuyPy.* TO 'BUYDB_OPERATOR'@'localhost';

GRANT EXECUTE ON PROCEDURE BuyPy.CreateOrder TO 'BUYDB_OPERATOR'@'localhost';
GRANT EXECUTE ON PROCEDURE BuyPy.GetOrderTotal TO 'BUYDB_OPERATOR'@'localhost';

# Criar utilizador BUYDB_ADMIN
CREATE USER 'BUYDB_ADMIN'@'localhost' IDENTIFIED BY 'Limaxy20#a';

GRANT ALL PRIVILEGES ON BuyPy.* TO 'BUYDB_ADMIN'@'localhost' WITH GRANT OPTION;

# Aplicar todas as alterações
FLUSH PRIVILEGES;
```

Figura 10 - Criação dos Users e suas Permissoes

Para o utilizador WEB\_CLIENTE, é permitido consultar por SELECT todas as tabelas do sistema e ainda inserir (INSERT) e também atualizar (UPDATE) registos nas tabelas Clients, Orders e Ordered\_Item, bem como apenas o campo “quantity” em Product. Tem autorização para apagar com DELETE registos em Orders e Ordered Items. Por fim, é permitido o execute de duas stored procedures: CreateOrder e GetOrderTotal.

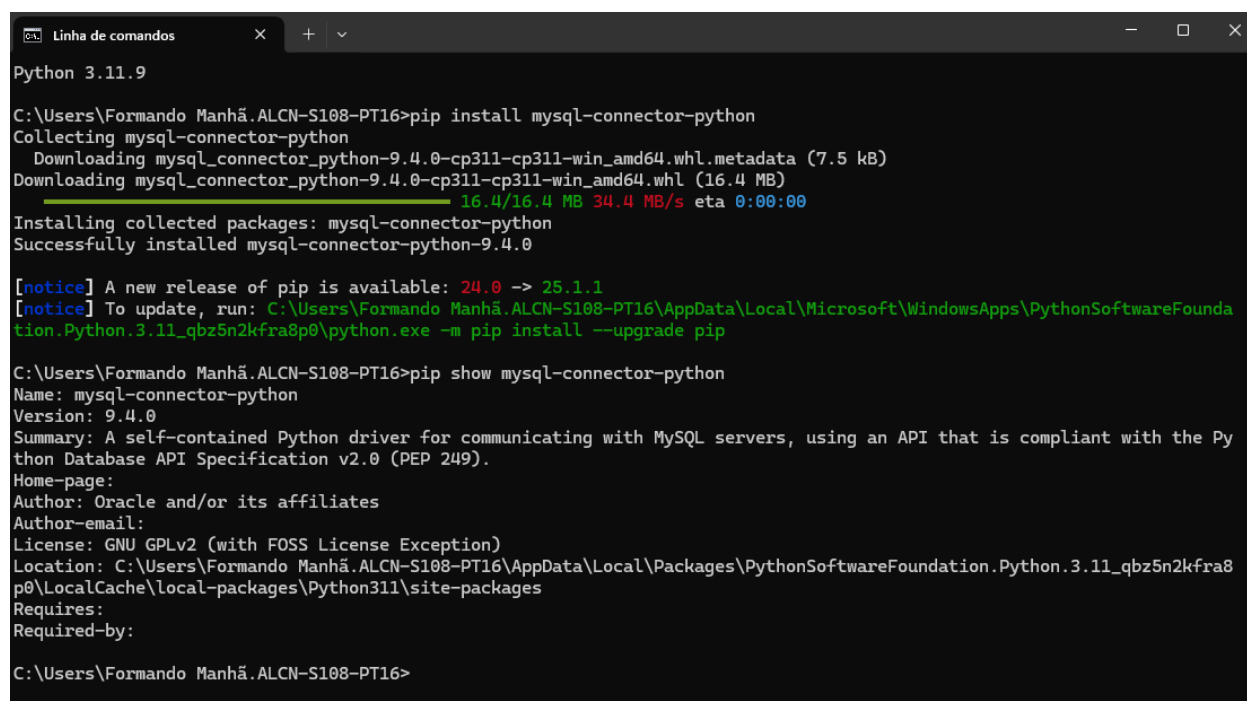
Já o user OPERATOR tem permissões para consultar, inserir atualizar e apagar em todas as tabelas da base de dados BuyPy e pode ainda executar todos os procedures.

Por fim, existe o ADMIN, que tem todas as possibilidades do OPERATOR mas pode ainda dar permissões a outros users com "WITH GRANT OPTION".



## BackOffice

Antes de começarmos a desenvolver o código identificamos que seria necessário de acordo com exemplos verificados no StackOverflow descarregar e instalar o mysql-connector-python. Esta funcionalidade permitiria a comunicação entre o programa de backoffice desenvolvido em python, e o que criámos até agora em mysql.



```
Python 3.11.9

C:\Users\Formando Manhã.ALCN-S108-PT16>pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.4.0-cp311-cp311-win_amd64.whl.metadata (7.5 kB)
  Downloading mysql_connector_python-9.4.0-cp311-cp311-win_amd64.whl (16.4 MB)
    16.4/16.4 MB 34.4 MB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.4.0

[notice] A new release of pip is available: 24.0 -> 25.1.1
[notice] To update, run: C:\Users\Formando Manhã.ALCN-S108-PT16\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFounda
tion.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip

C:\Users\Formando Manhã.ALCN-S108-PT16>pip show mysql-connector-python
Name: mysql-connector-python
Version: 9.4.0
Summary: A self-contained Python driver for communicating with MySQL servers, using an API that is compliant with the Py
thon Database API Specification v2.0 (PEP 249).
Home-page:
Author: Oracle and/or its affiliates
Author-email:
License: GNU GPLv2 (with FOSS License Exception)
Location: C:\Users\Formando Manhã.ALCN-S108-PT16\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8
p0\LocalCache\local-packages\Python311\site-packages
Requires:
Required-by:

C:\Users\Formando Manhã.ALCN-S108-PT16>
```

Figura 11- Instalação do mysql-connector-python

Tinhamos ainda um recurso importante fornecido pelo professor João Galamba, o ficheiro console\_utils.py que inclui já desenvolvidas funções úteis para a interação com o utilizador.

```
from console_utils import ask, show_msg, confirm, pause, cls
import mysql.connector
from getpass import getpass
import os
```

Figura 12 - Imports

Começamos por importar estas funções e ainda o getpass, que lemos ser mais seguro para passwords do que usar um input().



```
# LIGAR À BASE DE DADOS
# Pede ao utilizador nome e password para ligar a uma base de dados chamada
# BuyPy, a nossa. Se a ligação for bem sucedida, retorna o conn (connection, funcionalidade
# do mysql.connector), caso contrário, retorna o erro

def ligar_bd():
    cls()
    show_msg("LOGIN OPERADOR")
    user = ask("Utilizador: ")
    password = getpass("Palavra-passe: ")

    try:
        conn = mysql.connector.connect(
            host="localhost",
            user=user,
            password=password,
            database="BuyPy"
        )
        show_msg(["Ligação com sucesso!"])
        pause()
        return conn
    except mysql.connector.Error as err:
        show_msg(f"Erro na ligação: {err}")
        pause()
        return None
```

Figura 13 - Ligação à Base de Dados

Começámos por realizar a função que liga à base de dados, lendo sobre as funcionalidades do mysql.connector. Uma delas, a connect, permite criar a ligação e retorná-la como um objecto, conn que pode mais a frente ser usado para interagir com os conteúdos da mesma. A função procura ligar à base de dados com a informação de login que pede e caso consiga, confirma com o utilizador esse sucesso e retorna o conn. Caso contrário, acusa o erro (de login ou outro). Especificámos à partida já a base de dados ser a BuyPy pois neste caso é a com que estamos a trabalhar, mas seria possível também essa informação ser dada pelo utilizador.



Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI08

```
35 # MENU UTILIZADOR
36 #Pesquisa cliente pelo ID ou Nome, apresentando os dados pedidos numa tabela
37 #em conformidade com a demonstrada no enunciado:
38 #cursor é um objeto que permite executar comandos SQL e ler os seus resultados
39 #dentro do programa em python
40
41 def menu_utilizador(conn):
42     cls()
43     show_msg("MENU UTILIZADOR")
44     cursor = conn.cursor() #cursor vai interagir com a base de dados
45
46     termo = ask("Pesquisar por ID ou nome: ")
47
48     if termo.isdigit():
49         #se for numero pesquisa por ID
50         cursor.execute("SELECT client_id, fullname, email, zip_code, city, birthdate FROM Clients WHERE client_id = %s", (termo,))
51     else:
52         #caso contrario, assume o nome
53         cursor.execute("SELECT client_id, fullname, email, zip_code, city, birthdate FROM Clients WHERE fullname LIKE %s", ("%"+termo+"%"))
54
55     resultados = cursor.fetchall() #lê os resultados, cada um deles fica gravado como um tuplo
56
57     if not resultados:
58         show_msg("Nenhum utilizador encontrado.")
59         pause()
60     return
```

Figura 14 - Menu Utilizador - Pesquisa dos Dados

De seguida, avançámos para o menu utilizador, este permite procurar um cliente usando ou o nome ou ID e apresenta os dados de acordo com o pedido no enunciado. Para isso, usamos a função cursor, que existindo um connect permite ao Python executar comandos SQL, como queries (que iremos fazer).

No inicio, pedimos ao utilizador o que quer pesquisar. Se o programa receber um número, vai assumir que é ID, caso contrário, assumirá ser o nome. Seleccionada a opção correta, é executado um select das informações de acordo com esse critério, e o mesmo é adicionado como tuplo a uma lista, resultados.

Pode ainda haver o caso de não encontrar nada. Salvaguardamos este com uma mensagem nesse sentido.

```
print("+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+")
print("| ID | Nome      | Apelido  | Email      | Cod. Post | Cidade   | Nascimento |")
print("+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+")
for cliente in resultados: #percorre o conteudo de resultados, uma lista de tuplos
    id_, fullname, email, zip_code, city, birthdate = cliente
    partes_nome = fullname.split(" ", 1) # divide só no primeiro espaço
    nome = partes_nome[0]
    apelido = partes_nome[1] if len(partes_nome) > 1 else "" #se só houver um nome, fica em branco

    print(f"| {id_} | {nome} | {apelido} | {email} | {zip_code} | {city} | {birthdate} |")
print("+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+")

pause()
#caso tenhamos pesquisado por nome, pode haver mais do que um com o mesmo nome
escolha = ask("ID do cliente a alterar (Enter para sair): ")
if not escolha:
    return
```

Figura 15 - Menu Utilizador - Apresentação de Resultados





---

**Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI08**

Em seguida, criamos o cabeçalho da tabela e percorremos o conteúdo do tuplo. A impressão do mesmo é relativamente simples, tendo so em conta antes dividir o nome completo (fullname) em nome e apelido, dividindo-o no primeiro espaço.

O menu dos utilizadores compreende ainda a possibilidade de seleccionar e bloquear ou desbloquear um dos resultados da pesquisa. Para isto pedimos que confirme o ID do cliente a seleccionar, e de seguida lemos o booleano `is_active`. Se for 0, que corresponde a true, perguntamos se quer bloquear, e mudamos para 1 (false). Caso contrário, segue-se o processo inverso.

```
pause()
#caso tenhamos pesquisado por nome, pode haver mais do que um com o mesmo nome
escolha = ask("ID do cliente a alterar (Enter para sair): ")
if not escolha:
    return

cursor.execute("SELECT is_active FROM Clients WHERE client_id = %s", (escolha,))
resultado = cursor.fetchone()

if resultado is None:
    show_msg("Utilizador não encontrado.")
    pause()
    return

is_active = resultado[0] #vamos buscar o tuplo desse utilizador e iremos consultar se está activo ou não

if is_active:
    #se estiver activo - is_active 0, num bool 0 corresponde a true, pergunta se desejamos bloquear
    if confirm("Deseja bloquear este utilizador?"):
        cursor.execute("UPDATE Clients SET is_active = 0 WHERE client_id = %s", (escolha,))
        conn.commit()
        show_msg("Utilizador bloqueado.")
else:
    #caso identifique um 1 - false no bool - pergunta se pretendemos desbloquear
    if confirm("Deseja desbloquear este utilizador?"):
        cursor.execute("UPDATE Clients SET is_active = 1 WHERE client_id = %s", (escolha,))
        conn.commit()
        show_msg("Utilizador desbloqueado.")

pause()
```

*Figura 16 - Menu Utilizador - Opções de Bloqueio / Desbloqueio*

Terminada esta etapa, avançamos depois para a o menu do produto. Neste recebemos cinco elementos, o tipo de produto que queremos procurar, a quantidade mínima e máxima e o preço mínimo e máximo.

Esses elementos são guardados e é definida uma estrutura base para a query.



```
# MENU PRODUTO
# Pesquisa produtos com o tipo livro ou eletrónico e com os seguintes critérios definidos no enunciado:
# intervalo de quantidade (definimos um mínimo e máximo)
# intervalo de preço (idem)
# Mostra tipo, quantidade, preço e descrição do produto

def menu_produto(conn):
    cls()
    show_msg("MENU PRODUTO")
    cursor = conn.cursor()

    #recolhemos os dados para pesquisa
    tipo = ask("Tipo (livro/eletronico ou Enter para pesquisar em ambos): ").lower()
    q_min = ask("Quantidade mínima (Enter para ignorar): ")
    q_max = ask("Quantidade máxima (Enter para ignorar): ")
    p_min = ask("Preço mínimo (Enter para ignorar): ")
    p_max = ask("Preço máximo (Enter para ignorar): ")

    # Começa a query base
    query = """
        SELECT Product.product_id, quantity, price,
        COALESCE(Book.title, CONCAT(Electronic.brand, ' ', Electronic.model)) AS descricao
        FROM Product
        LEFT JOIN Book ON Product.product_id = Book.product_id
        LEFT JOIN Electronic ON Product.product_id = Electronic.product_id
    """
```

Figura 17 - Menu Produto - Recolha dos Dados

Em seguida, estes resultados são mostrados de forma semelhante à função anterior.

```
# Executa a query
cursor.execute(query)
produtos = cursor.fetchall()

if not produtos:
    show_msg("Nenhum produto encontrado.")
    pause()
    return

# Mostrar resultados
print("+-----+-----+-----+-----+")
print("| Código | Quantidade| Preço    | Descrição |")
print("+-----+-----+-----+-----+")
for p in produtos:
    print(f"| {p[0]:<8} | {p[1]:<9} | {p[2]:<8.2f} | {p[3]:<27} |")
print("+-----+-----+-----+-----+")
pause()
```

Figura 18 - Menu Produto - Apresentação dos Resultados

Terminamos assim o menu dos produtos, e avançamos a para função de backup. Desenvolvemos em Windows que sabendo não ser talvez a preferência era o que tínhamos em casa para facilitar a realização do trabalho fora do tempo do curso. Identificamos o comando no StackOverflow e aplicámos.



```
# FAZER BACKUP

def fazer_backup():
    cls()
    show_msg("A fazer backup da base de dados...")
    #comando para o windows, em casa desenvolvemos nesse sistema operativo
    os.system(r"C:\wamp64\bin\mysql\mysql9.1.0\bin\mysqldump.exe" -u root BuyPy > backup_buypy.sql')
    show_msg("Backup feito com sucesso!")
    pause()
```

Figura 19 - Backup

Para realizar o menu principal, aproveitámos e adaptámos o que tínhamos realizado para o projeto efeitos, sendo a finalidade e funcionalidade idêntica, recebendo no entanto a conn (connection) de modo a poder transmiti-la na primeira e segunda opções onde esta função é necessária:

```
# MENU PRINCIPAL
#mostra o menu e chama a função correspondente à escolha do utilizador
def menu_principal(conn):
    while True:
        cls() #clearscreen
        #menu recriado à imagem do do projeto efeitos, reaproveitamos esse
        print('*'*52)
        print('*' + ' '*50 + '*')
        print('*' + ' '*2 + 'BACKOFFICE - MENU PRINCIPAL' + ' '*21+'*')
        print('*' + ' '*50 + '*')
        print('*' + ' '*4 + '1 - Menu Utilizador' + ' '*27+'*')
        print('*' + ' '*4 + '2 - Menu Produto' + ' '*30+'*')
        print('*' + ' '*4 + '3 - Fazer Backup' + ' '*30+'*')
        print('*' + ' '*4 + '4 ☐ Sair' + ' '*38+'*')
        print('*' + ' '*50 + '*')
        print('*'*52)
        print('')

        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            menu_utilizador(conn)
        elif opcao == '2':
            menu_produto(conn)
        elif opcao == '3':
            fazer_backup()
        elif opcao == '4':
            show_msg("A sair...")
            break
        else:
            show_msg("Opção inválida.")
```

Figura 10 - Menu Inicial





Por fim, faltava apenas a função que inicializa o programa:

```
# PROGRAMA PRINCIPAL
# Tenta ligar a base de dados e caso bem sucedido, corre o programa
def main():
    conn = ligar_bd()
    if conn:
        menu_principal(conn)
        conn.close()
    else:
        show_msg("Não foi possível ligar à base de dados.")

if __name__ == "__main__":
    main()
```

*Figura 11 - Inicialização do Programa*

Ao iniciar faz a ligação e caso consiga envia a conn ao menu principal, fechando-a após o utilizador optar por sair. Caso contrário, alerta que não conseguiu a ligação.

## Conclusão

O projeto BuyPy alia conhecimentos de Python a SQL, combinando-os e integrando-os de forma a criar uma aplicação de gestão de base de dados e os seus conteúdos. Concebemos e criámos tabelas, implementámos triggers de modo a validar os dados fornecidos pelo utilizador, desenvolvemos stored procedures e uma aplicação de backoffice em Python que ligámos a essa base de dados com o mysql-connector-python.

Consideramos que de modo geral o projeto foi bem sucedido, tendo apenas não conseguido por uma questão de tempo e coordenação dos vários trabalhos finais realizar os extras, e tendo encontrado algumas dificuldades que podem ser naturais para quem está a aprender, como a ordem da criação das tabelas tendo em conta os foreign keys, a criação das permissões para os utilizadores que requereu algum estudo, assim como a interacção entre o Python e SQL utilizando as funções que a permitem e implementando no BackOffice – a montagem de queries dinamicas e formatação da tabela dos resultados exigiram alguma pesquisa, testagem e correções até serem bem sucedidas.

No seu total pensamos que conseguimos um resultado satisfatório, bem estruturado e funcional que valida o projeto e a UFCD.



## Referências

**GeeksForGeeks.** (2021). Enumerator (ENUM) in MySQL. Tópico disponível em:

<https://www.geeksforgeeks.org/sql/enumerator-enum-in-mysql/>

(Consultado em julho de 2025)

**W3Schools.** (n.d.). MySQL CHECK Constraint. Tópico disponível em:

[https://www.w3schools.com/mysql/mysql\\_check.asp](https://www.w3schools.com/mysql/mysql_check.asp)

(Consultado em julho de 2025)

**Stack Overflow.** (2012). MySQL validate email records. Tópico disponível em:

<https://stackoverflow.com/questions/8338535/mysql-validate-email-records>

(Consultado em julho de 2025)

**Stack Overflow.** (2009). How to raise an error within a MySQL function. Tópico disponível em:

<https://stackoverflow.com/questions/465727/how-to-raise-an-error-within-a-mysql-function>

(Consultado em julho de 2025)

**MySQL Documentation.** (2024). CREATE TRIGGER Statement. Tópico disponível em:

<https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html>

(Consultado em julho de 2025)

**MySQL Connector/Python Developer Guide.** (2024). Chapter 5: Connector/Python Connection Arguments. Disponível em:

<https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html>

(Consultado em julho de 2025)

**MySQL Connector/Python Developer Guide.** (2024). Chapter 7: Using Cursor Classes.

Disponível em: <https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqldcursor.html>

(Consultado em julho de 2025)

**Devart.** (n.d.). Triggers in MySQL. Tópico disponível em:

<https://www.devart.com/dbforge/mysql/studio/triggers-in-mysql.html>

(Consultado em julho de 2025)

**Python Docs.** (2024). getpass — Prompting for passwords safely. Disponível em:

<https://docs.python.org/3/library/getpass.html>

(Consultado em julho de 2025)