Projeto Programação de Base de Dados

BuyPy

Índice

INTRODUÇÃO

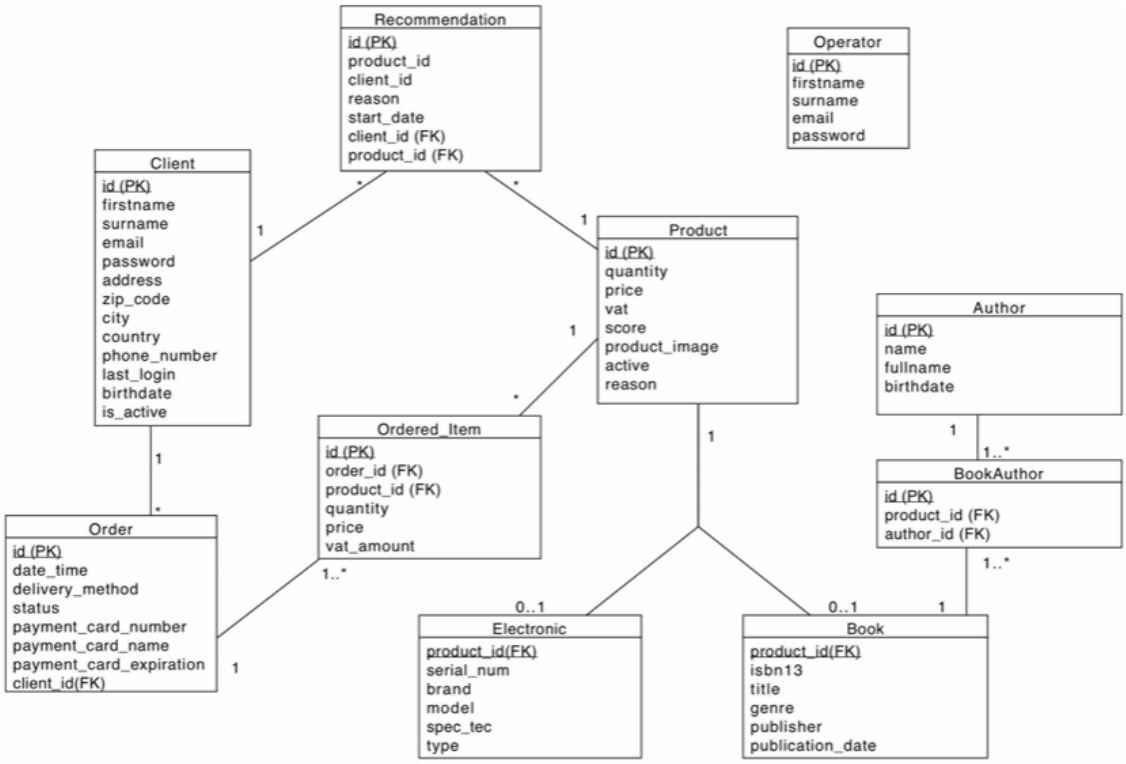
Este projeto de programação de base de dados visa a desenvolver uma base de dados relacional em MySQL para a **BuyPy**, que será uma empresa dedicada ao comércio online. Uma loja que pretende oferecer uma plataforma de compras online eficiente e de fácil utilização para os clientes, especializando-se na venda de livros e produtos eletrónicos, como leitores de MP3, televisões, computadores, entre outros.

Além da criação da base de dados, o projeto pretende incluir também o desenvolvimento de uma aplicação de BackOffice que vai permitir aos funcionários, incluindo operadores e administradores de sistemas de informação, gerir o estado do sistema, adicionar novos produtos, validar acessos e realizar outras funcionalidades essenciais para o funcionamento eficaz da empresa. Para realização extra do projeto, é considerado o desenvolvimento de um esboço das páginas do Web Frontend da loja, juntamente com a implementação da lógica server-side necessária para seu funcionamento.

Este projeto permite criar uma infraestrutura que possibilita o sucesso da loja no mercado online, atendendo às necessidades dos clientes e garantindo uma gestão eficaz dos recursos da empresa.

BackOffice

Diagrama Modelo de Dados



No modelo de dados já normalizado temos de ter as seguintes tabelas como é demonstrado no diagrama em cima:

→ Tabela de 'Product';
→ Tabela de 'Ordered Item';
→ Tabela de 'Order';
→ Tabela de 'Client';
→ Tabela de 'Recommendation';
→ Tabela de 'Eletronic';
→ Tabela de 'Book';
→ Tabela de 'BookAuthor';
→ Tabela de 'Author';
→ Tabela de 'Operator';

Relacionamentos e Identidades:

Ligação **Product** para **Eletronic** → De um para nenhum ou um (ou seja, um produto pode estar associado a zero ou um produto eletrónico. Isso quer dizer que um produto pode ou não ser eletrónico, se for estará associado apenas a um produto eletrónico específico);

Ligação **Product** para **Book** → De um para nenhum ou um (ou seja, um produto pode estar associado a zero ou um livro. Isso quer dizer que um produto pode ou não ser um livro, se for estará associado apenas a esse livro específico);

Ligação **Book** para **BookAuthor** → De um para um ou muitos (Isso significa que um livro pode ter um ou muitos autores, mas um autor está associado a um único livro);

Ligação **BookAuthor** para **Author** → De um ou muitos para um (Isto significa que vários autores podem estar associados a um único registo de autor. Por exemplo, se um livro é escrito por mais de um autor, todos esses autores estarão ligados ao mesmo registo de autor. Essa abordagem simplifica a estrutura do banco de dados, evitando a duplicação de informações sobre os autores);

Ligação **Product** para **Ordered_Item** → De um para muitos (ou seja, um produto pode estar associado a vários produtos pedidos, mas cada produto pedido está relacionado apenas um produto);

Ligação de **Ordered_Item** para **Order** → De um ou muitos para um (significa que vários produtos pedidos podem estar associados a um único pedido);

Ligação de **Order** para **Client** → De muitos para um (vários pedidos podem estar associados a um único cliente, ele pode fazer vários pedidos ao longo do tempo, e todos esses pedidos estarão relacionados a esse mesmo cliente);

Ligação de **Client** para **Recommendations** → De um para muitos (um cliente pode ter várias recomendações associadas a ele. Um cliente pode receber várias recomendações de produtos com base no histórico de compras, preferências ou outros dados relevantes);

Ligação de **Recommendations** para **Product** → De muitos para um

Base de dados – MySQL

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema BuyPy
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema BuyPy
-- -----------------------------------------------------
```

Criação de um banco de dados chamado BuyPy, usando o recurso de Engenharia Reversa:

Criação do Esquema BuyPy → Esta parte cria o esquema BuyPy, se ele ainda não existir, e define-o como esquema ativo.

```
CREATE SCHEMA IF NOT EXISTS `BuyPy` DEFAULT CHARACTER SET utf8 ;
USE `BuyPy` ;

-- -----------------------------------------------------
-- Table `BuyPy`.`Product`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Product` (
  `id` VARCHAR(10) NOT NULL,
  `quantity` INT UNSIGNED NOT NULL,
  `price` DECIMAL UNSIGNED NOT NULL,
  `vat` REAL UNSIGNED NOT NULL,
  `score` TINYINT(1) NULL,
  `product_image` VARCHAR(45) NULL,
  `active` VARCHAR(45) NULL,
  `reason` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

Tabela Product → Esta parte define a tabela Product, que armazena informações sobre os produtos disponíveis para compra.

```
-- Table `BuyPy`.`Electronic`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Electronic` (
  `product_id` VARCHAR(10) NOT NULL,
  `serial_number` INT NOT NULL,
  `brand` VARCHAR(20) NOT NULL,
  `model` VARCHAR(20) NOT NULL,
  `spec_tec` LONGTEXT NULL,
  `type` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`product_id`),
  INDEX `fk_electronica_produtos1_idx` (`product_id` ASC) VISIBLE,
  UNIQUE INDEX `serial_number_UNIQUE` (`serial_number` ASC) VISIBLE,
  CONSTRAINT `fk_electronica_produtos1`
    FOREIGN KEY (`product_id`)
    REFERENCES `BuyPy`.`Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela Electronic → armazena informações específicas para produtos eletrónicos. Vai possuir também uma chave estrangeira da tabela Product.

```
-- Table `BuyPy`.`Book`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Book` (
  `product_id` VARCHAR(10) NOT NULL,
  `isbn13` VARCHAR(20) NOT NULL,
  `title` VARCHAR(50) NOT NULL,
  `genre` VARCHAR(50) NOT NULL,
  `publisher` VARCHAR(100) NOT NULL,
  `publication_date` DATE NOT NULL,
  INDEX `fk_livros_produtos1_idx` (`product_id` ASC) VISIBLE,
  UNIQUE INDEX `isbn13_UNIQUE` (`isbn13` ASC) VISIBLE,
  PRIMARY KEY (`product_id`),
  CONSTRAINT `fk_livros_produtos1`
    FOREIGN KEY (`product_id`)
    REFERENCES `BuyPy`.`Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela Book → armazena informações específicas para produtos livro. Também possui uma chave estrangeira da tabela Product.

```
-- Table `BuyPy`.`Client`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Client` (
  `id` INT NOT NULL,
  `firstname` VARCHAR(250) NOT NULL,
  `surname` VARCHAR(250) NOT NULL,
  `email` VARCHAR(50) NOT NULL,
  `password` VARCHAR(50) NOT NULL,
  `address` VARCHAR(100) NOT NULL,
  `zip_code` VARCHAR(20) NOT NULL,
  `city` VARCHAR(30) NOT NULL,
  `country` VARCHAR(30) NOT NULL DEFAULT 'Portugal',
  `phone_number` VARCHAR(15) NULL,
  `last_login` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `birthdate` DATE NOT NULL,
  `is_active` TINYINT(1) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB;
```

Tabela Client → armazena informações dos clientes que utilizam a plataforma. Usa a chave primária id e um índice único em email para garantir a unicidade dos emails dos clientes.

```
-- Table `BuyPy`.`Order`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Order` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `date_time` DATETIME NULL DEFAULT NOW(),
  `delivery_method` ENUM('regular', 'urgent') NULL DEFAULT 'regular',
  `status` ENUM('open', 'processing', 'pending', 'closed', 'cancelled') NULL DEFAULT 'open',
  `payment_card_number` INT NOT NULL,
  `payment_card_name` VARCHAR(20) NOT NULL,
  `payment_card_expiration` DATE NOT NULL,
  `client_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Order_Client1_idx` (`client_id` ASC) VISIBLE,
  CONSTRAINT `fk_Order_Client1`
    FOREIGN KEY (`client_id`)
    REFERENCES `BuyPy`.`Client` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela Order → Regista informações sobre os pedidos feitos pelos clientes. Usa a chave primária 'id e um índice em 'client_id' para facilitar a consulta de pedidos por cliente.

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Recommendation` (
  `recomendacao` INT NOT NULL AUTO_INCREMENT,
  `data_rec` DATE NULL,
  `client_id` INT NOT NULL,
  `product_id` VARCHAR(10) NOT NULL,
  `reason` VARCHAR(500) NULL,
  `start_date` DATE NULL,
  PRIMARY KEY (`recomendacao`),
  INDEX `fk_Recommendation_Product1_idx` (`product_id` ASC) VISIBLE,
  INDEX `fk_Recommendation_Client1_idx` (`client_id` ASC) VISIBLE,
  CONSTRAINT `fk_Recommendation_Product1`
    FOREIGN KEY (`product_id`)
    REFERENCES `BuyPy`.`Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Recommendation_Client1`
    FOREIGN KEY (`client_id`)
    REFERENCES `BuyPy`.`Client` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela Recommendation → armazena recomendações feitas aos clientes sobre produtos. Usa a chave primária 'recomendacao' e índices 'client_id' e 'product_id' para facilitar consultas.

```
-- -----------------------------------------------------
-- Table `BuyPy`.`Operator`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Operator` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `firstname` VARCHAR(250) NOT NULL,
  `surname` VARCHAR(250) NOT NULL,
  `email` VARCHAR(50) NOT NULL,
  `password` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB;
```

Tabela Operator → Armazena informações dos operadores (funcionários) da plataforma. Usa a chave primária 'id' e um indice único em 'email' para garantir emails únicos.

```
-- -----------------------------------------------------
-- Table `BuyPy`.`Author`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`Author` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NULL,
  `fullname` VARCHAR(100) NULL,
  `birthdate` DATE NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

Tabela Author → armazena informações dos autores dos livros. Usa a chave primária 'id' para identificar os autores.

```
CREATE TABLE IF NOT EXISTS `BuyPy`.`Ordered_Item` (
  `id` INT NOT NULL,
  `quantity` INT UNSIGNED NOT NULL,
  `price` DECIMAL UNSIGNED NOT NULL,
  `vat_amount` DECIMAL UNSIGNED NOT NULL,
  `Order_id` INT NOT NULL,
  `Product_id` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`id`, `Order_id`),
  INDEX `fk_Ordered_Item_Order1_idx` (`Order_id` ASC) VISIBLE,
  INDEX `fk_Ordered_Item_Product1_idx` (`Product_id` ASC) VISIBLE,
  CONSTRAINT `fk_Ordered_Item_Order1`
    FOREIGN KEY (`Order_id`)
    REFERENCES `BuyPy`.`Order` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Ordered_Item_Product1`
    FOREIGN KEY (`Product_id`)
    REFERENCES `BuyPy`.`Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela Ordered_Item → regista os artigos especifícos de cada pedido. Usa a chave primária composta por ‘id’ e ‘order_id’ e indices em ‘order_id’ e ‘product_id’.

```
-- Table `BuyPy`.`BookAuthor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `BuyPy`.`BookAuthor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `product_id` VARCHAR(10) NOT NULL,
  `author_id` INT NOT NULL,
  PRIMARY KEY (`id`, `product_id`, `author_id`),
  INDEX `fk_BookAuthor_Book1_idx` (`product_id` ASC) VISIBLE,
  INDEX `fk_BookAuthor_Author1_idx` (`author_id` ASC) VISIBLE,
  CONSTRAINT `fk_BookAuthor_Book1`
    FOREIGN KEY (`product_id`)
    REFERENCES `BuyPy`.`Book` (`product_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_BookAuthor_Author1`
    FOREIGN KEY (`author_id`)
    REFERENCES `BuyPy`.`Author` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Tabela BookAuthor → faz a associação entre os livros e os seus autores. Usa a chave primária composta por ‘id’, ‘product_id’ e ‘author_id’.

Criadas as tabelas vamos começar por inserir os nossos registos nas tabelas apresentadas anteriormente num novo script.

Tabelas de inserção de dados:

```
USE `BuyPy` ;

DELETE FROM BuyPy.Electronic;
DELETE FROM BuyPy.Book;
DELETE FROM BuyPy.Product;
DELETE FROM BuyPy.Client;
DELETE FROM BuyPy.Order;
DELETE FROM BuyPy.Recommendation;
DELETE FROM BuyPy.Operator;
DELETE FROM BuyPy.Author;
DELETE FROM BuyPy.Ordered_Item;
DELETE FROM BuyPy.BookAuthor;
```

```
INSERT INTO BuyPy.Product (id, quantity, price, vat, score, product_image, active, reason)
VALUES
    ('P001', 100, 29.33, 0.23, 5, 'product1.jpg', 'active', 'Reason 1'),
    ('P002', 80, 39.44, 0.23, 4, 'product2.jpg', 'inactive', 'Reason 2'),
    ('P003', 120, 19.99, 0.23, 3, 'product3.jpg', 'active', 'Reason 3'),
    ('P004', 150, 49.99, 0.23, 4, 'product4.jpg', 'active', 'Reason 4'),
    ('P005', 90, 59.78, 0.23, 5, 'product5.jpg', 'inactive', 'Reason 5'),
    ('P006', 110, 69.89, 0.23, 3, 'product6.jpg', 'active', 'Reason 6'),
    ('P007', 130, 79.99, 0.23, 4, 'product7.jpg', 'active', 'Reason 7'),
    ('P008', 70, 89.98, 0.23, 5, 'product8.jpg', 'inactive', 'Reason 8'),
    ('P009', 100, 99.99, 0.23, 4, 'product9.jpg', 'active', 'Reason 9'),
    ('P010', 200, 109.00, 0.23, 5, 'product10.jpg', 'active', 'Reason 10'),
    ('P011', 180, 79.99, 0.23, 4, 'product11.jpg', 'active', 'Reason 11'),
    ('P012', 220, 89.99, 0.23, 5, 'product12.jpg', 'inactive', 'Reason 12'),
    ('P013', 130, 99.99, 0.23, 3, 'product13.jpg', 'active', 'Reason 13'),
    ('P014', 170, 109.99, 0.23, 4, 'product14.jpg', 'active', 'Reason 14'),
    ('P015', 150, 119.99, 0.23, 5, 'product15.jpg', 'inactive', 'Reason 15'),
    ('P016', 200, 129.99, 0.23, 4, 'product16.jpg', 'active', 'Reason 16'),
    ('P017', 250, 139.99, 0.23, 3, 'product17.jpg', 'active', 'Reason 17'),
    ('P018', 160, 149.99, 0.23, 5, 'product18.jpg', 'inactive', 'Reason 18'),
    ('P019', 190, 159.99, 0.23, 4, 'product19.jpg', 'active', 'Reason 19'),
    ('P020', 210, 169.99, 0.23, 3, 'product20.jpg', 'active', 'Reason 20');
```

Esta tabela vai ter representado a totalidade dos produtos disponíveis, livros e produtos de eletrónica. Cada alínea representa um dos produtos com as suas informações, desde o ID, quantidade, uma imagem ilustrativa do produto, o seu preço, etc.

```
INSERT INTO BuyPy.Electronic (product_id, serial_number, brand, model, spec_tec, type)
VALUES
    ('P001', 123456789, 'Garmin', 'AA', 'Spec 1', 'Electronic'),
    ('P002', 987654321, 'Suunto', 'ABA', 'Spec 2', 'Electronic'),
    ('P003', 456789123, 'Polar', 'ABB', 'Spec 3', 'Electronic'),
    ('P004', 789123456, 'Bryton', 'ACC', 'Spec 4', 'Electronic'),
    ('P005', 321654987, 'Sigma', 'CBA', 'Spec 5', 'Electronic'),
    ('P006', 654987321, 'TwoNav', 'BAA', 'Spec 6', 'Electronic'),
    ('P007', 987321654, 'Tanita', 'BBC', 'Spec 7', 'Electronic'),
    ('P008', 321987654, 'Beurer', 'BC', 'Spec 8', 'Electronic'),
    ('P009', 654321987, 'Polar', 'CA', 'Spec 9', 'Electronic'),
    ('P010', 123789456, 'Sigma', 'CCA', 'Spec 10', 'Electronic');
```

De seguida construímos a tabela dos produtos de eletrónica e dos livros, duas tabelas distintas com as devidas características.

```
-- Inserting into Book table
INSERT INTO BuyPy.Book (product_id, isbn13, title, genre, publisher, publication_date)
VALUES
    ('P011', '1234567890123', 'Cem anos de solidão', 'Fiction', 'Companhia das Letras', '2023-01-15'),
    ('P012', '9876543210123', 'A arte da guerra', 'Non-fiction', 'Editora Record', '2022-05-20'),
    ('P013', '4567891230123', 'A sombra do vento', 'Mystery', 'Editora Intrínseca', '2021-12-10'),
    ('P014', '7891234560123', 'O silêncio dos Inocentes', 'Thriller', 'Editora Rocco', '2020-08-05'),
    ('P015', '3216549870123', 'Um dia', 'Romance', 'Editora Zahar', '2019-03-25'),
    ('P016', '6549873210123', 'O senhor dos anéis', 'Fantasy', 'Editora Arqueiro', '2018-11-30'),
    ('P017', '9873216540123', 'O homem do castelo alto', 'Sci-fi', 'Editora Novo Conceito', '2017-06-15'),
    ('P018', '3219876540123', 'O iluminado', 'Horror', 'Editora Objetiva', '2016-02-20'),
    ('P019', '6543219870123', 'Uma Vida Interrompida: Memórias de um Anjo Assassinado', 'Biography', 'Editora Globo', '2015-09-10'),
    ('P020', '1237894560123', 'Guns, Germs, and Steel: The Fates of Human Societies', 'History', 'Editora Sextante', '2014-04-05');


-- Inserting into Client table
INSERT INTO BuyPy.Client (id, firstname, surname, email, password, address, zip_code, city, country, phone_number, birthdate, is_active)
VALUES
    (11, 'Eva', 'Garcia', 'eva@example.com', 'password11', '123 Boulevard', 12345, 'City 11', 'Country 11', '1234567890', '1990-01-11', 1),
    (12, 'Daniel', 'Martinez', 'daniel@example.com', 'password12', '456 Street', 54321, 'City 12', 'Country 12', '9876543210', '1995-05-12', 1),
    (13, 'Sophia', 'Rodriguez', 'sophia@example.com', 'password13', '789 Avenue', 67890, 'City 13', 'Country 13', '2345678901', '1985-10-13', 0),
    (14, 'Alexander', 'Lopez', 'alexander@example.com', 'password14', '101 Road', 98765, 'City 14', 'Country 14', '8765432109', '1980-03-14', 1),
    (15, 'Isabella', 'Hernandez', 'isabella@example.com', 'password15', '202 Lane', 34567, 'City 15', 'Country 15', '7654321098', '1975-07-15', 1),
    (16, 'Mia', 'Gonzalez', 'mia@example.com', 'password16', '303 Court', 98765, 'City 16', 'Country 16', '6543210987', '1970-12-16', 0),
    (17, 'Charlotte', 'Perez', 'charlotte@example.com', 'password17', '404 Street', 34567, 'City 17', 'Country 17', '5432109876', '1965-08-17', 1),
    (18, 'Lucas', 'Torres', 'lucas@example.com', 'password18', '505 Avenue', 87654, 'City 18', 'Country 18', '4321098765', '1960-04-18', 1),
    (19, 'Liam', 'Rivera', 'liam@example.com', 'password19', '606 Road', 56789, 'City 19', 'Country 19', '3210987654', '1955-09-19', 0),
    (20, 'James', 'Gomez', 'james@example.com', 'password20', '707 Boulevard', 23456, 'City 20', 'Country 20', '2109876543', '1950-02-20', 1);
```

Precisamos também de uma tabela de Clientes da loja, vai ter as seguintes informações do utilizador: o ID, o seu nome e apelido, o seu email, a sua password de acesso à plataforma, a sua morada, código postal, cidade e país para o serviço de entrega das encomendas. Vai ter também o seu contacto telefónico, o seu ano de aniversário e o seu estado na plataforma.

```
-- Inserting into Order table
INSERT INTO BuyPy.Order (date_time, delivery_method, status, payment_card_number, payment_card_name, payment_card_expiration, client_id)
VALUES
    ('2024-04-26 10:00:00', 'regular', 'open', 3423, 'John Doe', '2026-01-01', 11),
    ('2024-04-26 11:00:00', 'urgent', 'processing', 3242, 'Jane Doe', '2025-01-01', 12),
    ('2024-04-26 12:00:00', 'regular', 'pending', 1241, 'Alice Smith', '2024-01-01', 13),
    ('2024-04-26 13:00:00', 'urgent', 'closed', 7634, 'Bob Johnson', '2023-01-01', 14),
    ('2024-04-26 14:00:00', 'regular', 'cancelled', 43578, 'Emily Brown', '2022-01-01', 15),
    ('2024-04-26 15:00:00', 'urgent', 'open', 3543, 'David Wilson', '2021-01-01', 16),
    ('2024-04-26 16:00:00', 'regular', 'processing', 3246, 'Sarah Taylor', '2020-01-01', 17),
    ('2024-04-26 17:00:00', 'urgent', 'pending', 8945, 'Michael Miller', '2019-01-01', 18),
    ('2024-04-26 18:00:00', 'regular', 'closed', 9541, 'Olivia Davis', '2018-01-01', 19),
    ('2024-04-26 19:00:00', 'urgent', 'cancelled', 1245, 'William Martinez', '2017-01-01', 20);
```

A seguinte tabela, é uma tabela de pedidos (encomenda) vai possuir a data em que foi feita, o seu método de envio, o estado da encomenda, o número do cartão bancário do utilizador, o nome do utilizador do cartão bancário e a sua data de expiração, fora isso vai ainda possuir o id de cliente.

```
-- Inserting into Recommendation table
INSERT INTO BuyPy.Recommendation (data_rec, client_id, product_id, reason, start_date)
VALUES
    ('2024-04-26', 1, 'P001', 'Reason 1', '2024-04-26'),
    ('2024-04-26', 2, 'P002', 'Reason 2', '2024-04-26'),
    ('2024-04-26', 3, 'P003', 'Reason 3', '2024-04-26'),
    ('2024-04-26', 4, 'P004', 'Reason 4', '2024-04-26'),
    ('2024-04-26', 5, 'P005', 'Reason 5', '2024-04-26'),
    ('2024-04-26', 6, 'P006', 'Reason 6', '2024-04-26'),
    ('2024-04-26', 7, 'P007', 'Reason 7', '2024-04-26'),
    ('2024-04-26', 8, 'P008', 'Reason 8', '2024-04-26'),
    ('2024-04-26', 9, 'P009', 'Reason 9', '2024-04-26'),
    ('2024-04-26', 10, 'P010', 'Reason 10', '2024-04-26');
```

Aqui temos uma tabela das recomendações feitas pelos utilizadores.

```sql
-- Inserting into Operator table
INSERT INTO BuyPy.Operator (firstname, surname, email, password)
VALUES
    ('Ana Sousa', '1', 'anasousa@example.com', 'Passw0rd'),
    ('Daniel Palma', '2', 'daniel@example.com', 'Passw0rd');

-- Inserting into Author table
INSERT INTO BuyPy.Author (name, fullname, birthdate)
VALUES
    ('Afonso', 'Afonso Alberto', '1980-01-01'),
    ('Maria', 'Maria Cavaco', '1975-02-02'),
    ('Maria', 'Maria Teresa', '1970-03-03'),
    ('Joana', 'Joana Afonso', '1965-04-04'),
    ('Lucas', 'Lucas Afonso', '1960-05-05'),
    ('Mariana', 'Mariana Rebelo', '1955-06-06'),
    ('Rosa', 'Rosa Conceição', '1950-07-07'),
    ('Joana', 'Joana Azevedo', '1945-08-08'),
    ('Carla', 'Carla Pereira', '1940-09-09'),
    ('João', 'João Salgado', '1935-10-10');
```

Vamos criar uma tabela para os operadores da loja.
De seguida precisamos ainda de uma tabela de autor que vai possuir os registos dos autores dos livros disponíveis para venda.

```sql
-- Inserting into Ordered_Item table
INSERT INTO BuyPy.Ordered_Item (id, quantity, price, vat_amount, Order_id, Product_id)
VALUES
    (1, 2, 29.33, 3.66, 1, 'P001'),
    (2, 1, 39.44, 4.88, 1, 'P002'),
    (3, 3, 19.99, 2.44, 2, 'P003'),
    (4, 2, 49.99, 6.11, 2, 'P004'),
    (5, 1, 59.78, 7.33, 3, 'P005'),
    (6, 4, 69.89, 8.55, 3, 'P006'),
    (7, 2, 79.99, 9.80, 4, 'P007'),
    (8, 3, 89.98, 11.03, 4, 'P008'),
    (9, 1, 99.99, 12.24, 5, 'P009'),
    (10, 5, 109.00, 13.47, 5, 'P010');
```

Outra tabela de registos de encomendas, com o id, a quantidade pedida, o valor, a taxa de iva , o id da encomenda e o respetivo id do produto.

```sql
-- Inserting into BookAuthor table
INSERT INTO BuyPy.BookAuthor (product_id, author_id)
VALUES
    ('P001', 1),
    ('P002', 2),
    ('P003', 3),
    ('P004', 4),
    ('P005', 5),
    ('P006', 6),
    ('P007', 7),
    ('P008', 8),
    ('P009', 9),
    ('P010', 10);
```

Por fim uma tabela de ligação entre o livro e o autor, com o id do livro e o id do autor.

Feito o script dos registos vamos fazer os procedures antes de fazermos os testes, criamos um novo script denominado store_procedures.

STORE PROCEDURES:

```
USE `BuyPy` ;

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 ;
```

ProductByType é projetado para recuperar produtos com base no tipo especificado.
O procedimento aceita um parâmetro de entrada chamado **product_type**, que é usado para filtrar os resultados com base no tipo de produto. Se **product_type** for nulo, todos os produtos serão retornados.
Dentro do procedimento, há uma verificação condicional para determinar se **product_type** é nulo. Se for nulo, todos os produtos serão selecionados. Caso contrário, os produtos serão selecionados com base no tipo especificado.
Se **product_type** for nulo, uma consulta é realizada para selecionar todos os produtos da tabela **Product**. Os produtos são recuperados juntamente com o ID, preço, pontuação, estado (ativo ou inativo), imagem do produto e tipo de produto. O tipo de produto é determinado usando as tabelas **Electronic** e **Book** através de left joins.

Se product_type não for nulo, uma verificação é realizada para determinar se o tipo especificado é '**Electronic**', **'Book**' ou outro valor. Dependendo do tipo especificado, uma consulta é executada para selecionar os produtos da tabela **Product** que correspondem ao tipo especificado. O tipo de produto é explicitamente definido como 'Electronic' ou 'Book' na consulta.

Se **product_type** não for nulo e não corresponder a nenhum dos tipos válidos ('Electronic' ou 'Book'), uma mensagem de erro é devolvida a indicar que o tipo de produto é inválido.

```
DELIMITER //

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

DELIMITER ;
```

DailyOrders é projetado para recuperar pedidos com base na data especificada.
O procedimento aceita um parâmetro de entrada chamado **order_date**, que é do tipo DATE e representa a data para a qual os pedidos devem ser recuperados.
Dentro do procedimento, uma consulta é realizada na tabela **Order** para selecionar todos os campos (*). A cláusula **WHERE** é usada para filtrar os resultados com base na data especificada. A função **DATE**(date_time) extrai apenas a parte da data do campo **date_time**, e compara com a data fornecida como entrada.
A consulta retorna todos os pedidos que têm a mesma data de date_time que a data fornecida como entrada. O resultado final é a lista de pedidos que ocorreram na data especificada.

```
DELIMITER //

CREATE PROCEDURE AnnualOrders(IN cliente_id INT, IN ano_encomenda INT)
BEGIN
    -- Seleciona todas as encomendas feitas por um cliente especifico durante um ano especifico
    SELECT *
    FROM `BuyPy`.`Order`
    WHERE `client_id` = cliente_id AND YEAR(`date_time`) = ano_encomenda;
END //

DELIMITER ;
```

AnnualOrders é projetado para recuperar todos os pedidos feitos por um cliente específico durante um ano específico. O procedimento aceita dois parâmetros de entrada:

cliente_id: Representa o ID do cliente para o qual desejamos recuperar os pedidos.
ano_encomenda: Representa o ano para o qual desejamos recuperar os pedidos.

Dentro do procedimento, uma consulta é realizada na tabela **Order** para selecionar todos os campos (*). A cláusula **WHERE** é usada para filtrar os resultados com base no **client_id** fornecido e no ano da data de encomenda (date_time). Isso é feito utilizando a função **YEAR**(date_time) para extrair o ano da data de cada pedido e comparando-o com o ano fornecido como entrada. A consulta retorna todos os pedidos que foram feitos pelo cliente especificado durante o ano especificado.
O resultado final é a lista de pedidos feitos pelo cliente durante o ano fornecido.

```
DELIMITER //

CREATE PROCEDURE CreateOrder(
    IN cliente_id INT,
    IN metodo_expedicao ENUM('regular', 'urgent'),
    IN numero_cartao INT,
    IN nome_cartao VARCHAR(100),
    IN data_validade_cartao DATE
)
BEGIN
    -- Insere uma nova encomenda com os dados fornecidos
    INSERT INTO `BuyPy`.`Order` (
        `date_time`,
        `delivery_method`,
        `status`,
        `payment_card_number`,
        `payment_card_name`,
        `payment_card_expiration`,
        `client_id`
    )
    VALUES (
        NOW(),
        metodo_expedicao,
        'open', -- Definindo o status da encomenda como "open" por padrão
        numero_cartao,
        nome_cartao,
        data_validade_cartao,
        cliente_id
    );
END //

DELIMITER ;
```

CreateOrder cria uma nova encomenda com os dados fornecidos. O procedimento aceita cinco parâmetros de entrada:

cliente_id: Representa o ID do cliente que está fazendo a encomenda.
metodo_expedicao: Representa o método de envio da encomenda, que deve ser 'regular' ou 'urgent'.
numero_cartao: Representa o número do cartão de pagamento usado para a encomenda.
nome_cartao: Representa o nome impresso no cartão de pagamento.
data_validade_cartao: Representa a data de validade do cartão de pagamento.
Dentro do procedimento, uma instrução **INSERT INTO** é usada para adicionar uma nova entrada na tabela **Order**. Os valores para as colunas são definidos com base nos parâmetros de entrada fornecidos, juntamente com a data e hora atual (NOW()) para a coluna **date_time**.

O estado da encomenda é definido como "open" por padrão. Isso pode ser alterado posteriormente à medida que o estado da encomenda é atualizado durante o processamento. Após a execução bem-sucedida do procedimento, uma nova encomenda será criada na tabela **Order** com os dados fornecidos.

```
DELIMITER //

CREATE PROCEDURE `GetOrderTotal` (IN order_id INT)
BEGIN
    DECLARE total DECIMAL(10, 2);

    -- Calcula o montante total da encomenda
    SELECT SUM(price * quantity) INTO total
    FROM Ordered_Item
    WHERE Order_id = order_id;

    -- Devolve o montante total
    SELECT total;
END//

DELIMITER ;
```

GetOrderTotal calcula o montante total de uma encomenda com base no ID da encomenda fornecido. O procedimento aceita um parâmetro de entrada chamado **order_id**, que representa o ID da encomenda para a qual o montante total deve ser calculado.

Dentro do procedimento, uma variável local chamada total é declarada como um decimal com precisão de até 10 dígitos, dos quais 2 são decimais. Uma instrução **SELECT** é usada para calcular o montante total da encomenda. Isto é feito somando o preço multiplicado pela quantidade de cada artigo na tabela **Ordered_Item** que corresponde ao ID da encomenda fornecido. O resultado do cálculo é armazenado na variável total.

Finalmente, uma segunda instrução **SELECT** é usada para devolver o montante total calculado como resultado do procedimento.

```
DELIMITER //

CREATE PROCEDURE AddProductToOrder(
    IN order_id INT,
    IN product_id VARCHAR(10),
    IN quantity INT
)
BEGIN
    DECLARE total_price DECIMAL(10, 2);

    -- Calcular o preço total do produto
    SELECT price * quantity INTO total_price
    FROM Product
    WHERE id = product_id;

    -- Inserir o item da encomenda na tabela Ordered_Item
    INSERT INTO Ordered_Item (quantity, price, Order_id, Product_id)
    VALUES (quantity, total_price, order_id, product_id);

    -- Atualizar a quantidade de produtos disponíveis na tabela Product
    UPDATE Product
    SET quantity = quantity - quantity
    WHERE id = product_id;
END //

DELIMITER ;
```

AddProductToOrder é projetado para adicionar um produto a uma encomenda existente. O procedimento aceita três parâmetros de entrada: **order_id, product_id e quantity.**

order_id: O ID da encomenda à qual o produto será adicionado.

product_id: O ID do produto que será adicionado à encomenda.
quantity: A quantidade do produto a ser adicionada à encomenda.
Dentro do procedimento, uma variável local chamada **total_price** é declarada como um decimal com precisão de até 10 dígitos, dos quais 2 são decimais.

Uma instrução **SELECT** é usada para calcular o preço total do produto a ser adicionado, multiplicando o preço do produto pela quantidade especificada. O resultado do cálculo é armazenado na variável **total_price.**
Uma instrução **INSERT** é usada para inserir o artigo da encomenda na tabela **Ordered_Item,** incluindo a quantidade, o preço total calculado, o ID da encomenda e o ID do produto.
Uma instrução **UPDATE** é usada para atualizar a quantidade de produtos disponíveis na tabela **Product**. A quantidade do produto especificado é subtraída da quantidade disponível, pois está a ser adicionada à encomenda.

```
DELIMITER //

CREATE PROCEDURE AddBook(
    IN book_id VARCHAR(10),
    IN isbn13 VARCHAR(20),
    IN title VARCHAR(50),
    IN genre VARCHAR(50),
    IN publisher VARCHAR(100),
    IN publication_date DATE
)
BEGIN
    INSERT INTO Book (product_id, isbn13, title, genre, publisher, publication_date)
    VALUES (book_id, isbn13, title, genre, publisher, publication_date);
END //

DELIMITER ;
```

AddBook adiciona um novo livro à base de dados BuyPy.

O procedimento aceita seis parâmetros de entrada:
book_id: O ID do livro, que também é o ID do produto correspondente.
isbn13: O número ISBN-13 do livro.
title: O título do livro.
genre: O gênero do livro.
publisher: O editora do livro.
publication_date: A data de publicação do livro.
Dentro do procedimento, uma instrução **INSERT** é usada para adicionar os detalhes do livro à tabela **Book**. Os valores dos parâmetros de entrada são inseridos nas colunas correspondentes da tabela.

```
DELIMITER //

CREATE PROCEDURE AddElec(
    IN product_id VARCHAR(10),
    IN serial_number INT,
    IN brand VARCHAR(20),
    IN model VARCHAR(20),
    IN spec_tec LONGTEXT,
    IN type VARCHAR(10)
)
BEGIN
    INSERT INTO Electronic (product_id, serial_number, brand, model, spec_tec, type)
    VALUES (product_id, serial_number, brand, model, spec_tec, type);
END //

DELIMITER ;
```

AddElec é projetado para adicionar um novo produto eletrónico à base de dados BuyPy.
O procedimento aceita seis parâmetros de entrada:

product_id: O ID do produto eletrónico, que também é o ID do produto correspondente.
serial_number: O número de série do produto eletrónico.
brand: A marca do produto.
model: O modelo do produto.
spec_tec: As especificações técnicas do produto eletrónico, armazenadas como um campo LONGTEXT.
type: O tipo do produto eletrónico.
Dentro do procedimento, uma instrução **INSERT** é usada para adicionar os detalhes do produto eletrónico à tabela **Electronic**. Os valores dos parâmetros de entrada são inseridos nas colunas correspondentes da tabela.

Testar Store Procedures - Calls

```
220
221     -- Testar os store procedures criados
222
223
224 •   CALL ProductByType('Book');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	price	score	active	product_image	type
P011	80	4	active	product11.jpg	Book
P012	90	5	inactive	product12.jpg	Book
P013	100	3	active	product13.jpg	Book
P014	110	4	active	product14.jpg	Book
P015	120	5	inactive	product15.jpg	Book
P016	130	4	active	product16.jpg	Book
P017	140	3	active	product17.jpg	Book
P018	150	5	inactive	product18.jpg	Book
P019	160	4	active	product19.jpg	Book
P020	170	3	active	product20.jpg	Book

ProductByType é uma função que adquire todos os produtos de um tipo específico da tabela de produtos, neste caso a call vai buscar todos os produtos da tabela dos livros.

```
226 •  CALL DailyOrders('2024-04-26');
227
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	date_time	delivery_method	status	payment_card_number	payment_card_name	payment_card_expiration	client_id
34	2024-04-26 10:00:00	regular	open	3423	John Doe	2026-01-01	11
35	2024-04-26 11:00:00	urgent	processing	3242	Jane Doe	2025-01-01	12
36	2024-04-26 12:00:00	regular	pending	1241	Alice Smith	2024-01-01	13
37	2024-04-26 13:00:00	urgent	closed	7634	Bob Johnson	2023-01-01	14
38	2024-04-26 14:00:00	regular	cancelled	43578	Emily Brown	2022-01-01	15
39	2024-04-26 15:00:00	urgent	open	3543	David Wilson	2021-01-01	16
40	2024-04-26 16:00:00	regular	processing	3246	Sarah Taylor	2020-01-01	17
41	2024-04-26 17:00:00	urgent	pending	8945	Michael Miller	2019-01-01	18
42	2024-04-26 18:00:00	regular	closed	9541	Olivia Davis	2018-01-01	19
43	2024-04-26 19:00:00	urgent	cancelled	1245	William Martinez	2017-01-01	20

DailyOrders devolve todos os pedidos de um determinado dia, usamos uma função que aceita uma data como parâmetro e vai devolver uma tabela com os pedidos desse dia, aqui testamos com o dia 26 de Abril de 2024.

```
227
228 •  CALL AnnualOrders(11, 2024);
229
230    #Select * from BuyPy.Order;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	date_time	delivery_method	status	payment_card_number	payment_card_name	payment_card_expiration	client_id
34	2024-04-26 10:00:00	regular	open	3423	John Doe	2026-01-01	11

AnnualOrders dá-nos todos os pedidos de um determinado cliente num ano específico, usamos uma função que aceita o ID do cliente e o ano como parâmetros e devolve uma tabela com os pedidos desse cliente no ano especificado.

```
230 •  Select * from BuyPy.Order;
231 •  CALL CreateOrder(12, 'urgent', 1234567890, 'Guilherme Novo', '2026-12-01');
232    #Select * from BuyPy.Order;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	date_time	delivery_method	status	payment_card_number	payment_card_name	payment_card_expiration	client_id
34	2024-04-26 10:00:00	regular	open	3423	John Doe	2026-01-01	11
35	2024-04-26 11:00:00	urgent	processing	3242	Jane Doe	2025-01-01	12
36	2024-04-26 12:00:00	regular	pending	1241	Alice Smith	2024-01-01	13
37	2024-04-26 13:00:00	urgent	closed	7634	Bob Johnson	2023-01-01	14
38	2024-04-26 14:00:00	regular	cancelled	43578	Emily Brown	2022-01-01	15
39	2024-04-26 15:00:00	urgent	open	3543	David Wilson	2021-01-01	16
40	2024-04-26 16:00:00	regular	processing	3246	Sarah Taylor	2020-01-01	17
41	2024-04-26 17:00:00	urgent	pending	8945	Michael Miller	2019-01-01	18
42	2024-04-26 18:00:00	regular	closed	9541	Olivia Davis	2018-01-01	19
43	2024-04-26 19:00:00	urgent	cancelled	1245	William Martinez	2017-01-01	20
44	2024-05-16 12:18:29	urgent	open	1234567890	Guilherme Novo	2026-12-01	12

CreateOrder insere um novo pedido na tabela Orders, como podemos ver no final da tabela, ao criarmos o store procedure de CreateOrder, conseguimos adicionar novas informações à nossa tabela neste caso, um “novo pedido”.

```
235 •  CALL GetOrderTotal(28);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total
1672.00

GetOrderTotal devolve o valor total de um pedido específico, usamos uma função que aceita o ID do pedido como parâmetro e devolve o valor total do pedido.

```
-- 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'@'%';
```

PYTHON – CONEXÃO COM A BASE DE DADOS

sqlrequests.py 1 × config.ini

home > formando > Desktop > sqlrequests.py > conectar_bd

```python
import pymysql.cursors
import configparser
from cryptography.fernet import Fernet
import re
import subprocess


# Função para estabelecer a conexão com o banco de dados
# Função para estabelecer a conexão com o banco de dados
def conectar_bd():
    # Ler as configurações do arquivo config.ini
    config = configparser.ConfigParser()
    config.read('/home/formando/Desktop/src/Projeto_Final/config.ini')

    # Extrair as configurações do banco de dados do arquivo config.ini
    db_config = config['database']

    # Conectar ao banco de dados
    conexao = pymysql.connect(
        host=db_config['host'],
        user=db_config['user'],
        password=db_config['password'],
        database=db_config['database']
    )
    return conexao
```

```python
def login():
    # Obter entrada do usuário para nome de usuário e senha
    email = input("Digite o email de Operador: ")
    password = input("Digite a password: ")

    # Validar credenciais do usuário
    conexao = conectar_bd()
    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
    finally:
        conexao.close()
```

```python
def pesquisar_usuario():
    # Obter entrada do usuário para ID do usuário ou email do utilizador
    idoremail = input("Digite o ID do usuário ou email do utilizador: ")

    # Conectar ao banco de dados
    conexao = conectar_bd()

    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:")
                # Lista para armazenar larguras das colunas
                larguras = [max(len(str(field)) for field in row) for row in zip(*resultado)]
```

```python
                # Obter nomes das colunas
                nomes_colunas = [column[0] for column in cursor.description]

                # Imprimir cabeçalho da tabela
                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) + "+")

                # Imprimir linhas de dados
                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':
                    # Bloquear conta do usuário
                    # Implementar lógica de bloqueio aqui
                    print("Contas dos usuários bloqueadas com sucesso.")
                elif acao.lower() == 'd':
                    # Desbloquear conta do usuário
                    # Implementar lógica de desbloqueio aqui
                    print("Contas dos usuários desbloqueadas com sucesso.")
                else:
                    print("Opção inválida.")
            else:
                print("Nenhum usuário encontrado.")
    finally:
        conexao.close()
```

```python
def listar_produtos():
    # Conectar ao banco de dados
    conexao = conectar_bd()
    try:
        with conexao.cursor() as cursor:
            # Listar produtos com filtros opcionais
            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 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:")
                for linha in resultado:
                    print(linha)
            else:
                print("Nenhum produto encontrado com os critérios fornecidos.")
    finally:
        conexao.close()
```

```python
def executar_backup():
    # Nome do arquivo de backup
    config = configparser.ConfigParser()
    config.read('/home/formando/Desktop/src/Projeto_Final/config.ini')

    # Extrair as configurações do banco de dados do arquivo config.ini
    db_config = config['database']

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

    # Comando mysqldump para fazer o backup
    comando = f"mysqldump -u {db_config['user']} -p{db_config['password']} {db_config['database']} > {nome_arquivo}.sql"

    try:
        # Executar o comando de backup
        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}")
```

```python
def menu_principal(utilizador):
    # Loop até que o usuário escolha sair
    while True:
        print("1. Pesquisar Utilizador")
        print("2. Listar Produtos")
        print("3. Registar Produtos")
        print("4. Backup")
        print("5. Sair")

        escolha = input("Escolha uma opção (1, 2,3,4 ou 5): ")

        if re.match(r'1', escolha):
            pesquisar_usuario()
        elif re.match(r'2', escolha):
                listar_produtos()
        elif re.match(r'3', escolha):
                print("em obras")
        elif re.match(r'4', escolha):
            executar_backup()
        elif re.match(r'5', escolha):
            print("Saindo...")
            return
        else:
            print("Escolha inválida. Por favor, tente novamente.")
```

```python

# Função principal
if __name__ == "__main__":
    # Login
    utilizador = login()
    if utilizador:
        menu_principal(utilizador)

```

Importações de Módulos:

Pymysql
Configparser
re (expressões regulares)
subprocess (para executar comandos do sistema)
bcrypt (para hash de senha)

Conexão com a base de dados (conectar_bd()):

Função para estabelecer conexão com a base de dados MySQL, lê as informações de configuração do ficheiro config.ini e depois usar essas mesmas informações para criar a conexão.

Login (login()):

Solicita ao utilizador o email e password.
Verifica se as credenciais estão corretas consultando a base de dados.
Devolve os detalhes do utilizador se o login for bem-sucedido, caso contrário, devolve None.

Pesquisa de utilizador (pesquisar_usuario()):

Solicita ao utilizador um ID ou email.
Executa uma consulta SQL para encontrar utilizadores com base no ID ou email fornecido.
Formata e exibe os resultados da consulta.
Oferece opções para bloquear ou desbloquear utilizadores encontrados (não está finalizado).

Listagem de Produtos (listar_produtos()):

Solicita ao user filtros opcionais para tipo de produto, quantidade mínima/máxima e preço mínimo/máximo.
Monta uma consulta SQL dinâmica com base nos filtros fornecidos.
Executa a consulta na base de dados e exibe os resultados.

Backup da base de dados (executar_backup()):

Solicita ao utilizador um nome para o ficheiro de backup.
Usa o comando mysqldump para fazer um backup da base de dados MySQL e guarda o ficheiro com o nome fornecido pelo utilizador.

Menu Principal (menu_principal()):

Exibe um menu de opções para o utilizador, incluindo pesquisa pelo user, listagem de produtos, registo de produtos (em desenvolvimento), backup da base de dados e sair.
Solicita a escolha do utilizador e executa a função correspondente com base na escolha.

Função Principal (__main__):

Inicia o programa.
Chama a função de login e, se o login for bem-sucedido, apresenta o menu principal para o utilizador.

Testar o programa:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

    from cryptography.hazmat.bindings._padding import lib
ModuleNotFoundError: No module named '_cffi_backend'
19:19:30 gestor $ /bin/python3.10 /home/formando/Desktop/gestor/sqlrequests.py
Digite o email de Operador: anasousa@example.com
Digite a password: Passw0rd
Login bem-sucedido!
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5):
```

```
Digite o email de Operador: anasousa@example.com
Digite a password: Passw0rd
Login bem-sucedido!
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5): 1
Digite o ID do usuário ou email do utilizador: 2
Nenhum usuário encontrado.
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5): 1
Digite o ID do usuário ou email do utilizador: 15
Usuários encontrados:
+----+-----------+-----------+---------------------+----------+----------+------+----------+---------+--------------+---------------------+-----------+-----------+
| id | firstname | surname   | email               | password | address  | zip_code | city     | country | phone_number | last_login          | birthdate | is_active |
+----+-----------+-----------+---------------------+----------+----------+------+----------+---------+--------------+---------------------+-----------+-----------+
| 15 | Isabella  | Hernandez | isabella@example.com | password15 | 202 Lane | 34567 | Bucareste | Roménia | 7654321098 | 2024-05-24 15:51:00 | 1975-07-15 | 1 |
+----+-----------+-----------+---------------------+----------+----------+------+----------+---------+--------------+---------------------+-----------+-----------+
Você deseja (b)loquear ou (d)esbloquear estes usuários?
```

```
('P019', 190, Decimal('180'), 0.23, 4, 'product19.jpg', 'active', 'Reason 19')
('P020', 210, Decimal('170'), 0.23, 3, 'product20.jpg', 'active', 'Reason 20')
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5): 2
Digite o tipo de produto (Book ou Electronic) ou deixe em branco para todos:
Digite a quantidade mínima ou deixe em branco para todos:
Digite a quantidade máxima ou deixe em branco para todos:
Digite o preço mínimo ou deixe em branco para todos: 10
Digite o preço máximo ou deixe em branco para todos: 50
Produtos encontrados:
('P001', 100, Decimal('29'), 0.23, 5, 'product1.jpg', 'active', 'Reason 1')
('P002', 80, Decimal('39'), 0.23, 4, 'product2.jpg', 'inactive', 'Reason 2')
('P003', 120, Decimal('20'), 0.23, 3, 'product3.jpg', 'active', 'Reason 3')
('P004', 150, Decimal('50'), 0.23, 4, 'product4.jpg', 'active', 'Reason 4')
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5):
```

```
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5): 3
em obras
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5): 4
Digite o nome do arquivo de backup: lojaonline
mysqldump: [Warning] Using a password on the command line interface can be insecure.
Backup concluído com sucesso!
1. Pesquisar Utilizador
2. Listar Produtos
3. Registar Produtos
4. Backup
5. Sair
Escolha uma opção (1, 2,3,4 ou 5):
```

Conclusão

Ao longo deste trabalho, foram abordados os processos de criação e configuração de uma base de dados implementada em mysql, começamos por modelar a base de dados e de seguida criamos as tabelas e store procedures no mysql. As etapas seguintes incluíram as calls para testar os store procedures, desde criar um pedido (order) até dar-nos os pedidos de um determinado cliente num ano específico. Por fim desenvolveu-se uma miniaplicação da loja, começando por pedir o login de acesso do utilizador neste caso de um dos operadores da base de dados, podemos também listar utilizador a utilizador da nossa base de dados através de id ou username. No menu temos também a parte dos produtos em que podemos listar os mesmos, por categoria, por valores ou quantidades.

No entanto, algumas especificidades do projeto não foram concluídas, em particular o registo 3 (Registar Produtos). Os extras de QT Backoffice e Web FrontEnd também não ficaram finalizados.