Name: - Dheeraj Vemula

Case study: - Ecommerce application

Schema Design:

Create following tables in SQL Schema with appropriate class and write the unit test case for the Ecommerce application.

1. customers table:

• customer_id (Primary Key)

• name

• email

• password

```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| customer_id | int          | NO   | PRI | NULL    | auto_increment |
| name        | varchar(255) | YES  |     | NULL    |                |
| email       | varchar(255) | YES  |     | NULL    |                |
| password    | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
4 rows in set (0.12 sec)
```

2. products table:

• product_id (Primary Key)

• name

• price

• description

• stockQuantity

```
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| product_id    | int           | NO   | PRI | NULL    | auto_increment |
| name          | varchar(255)  | NO   |     | NULL    |                |
| price         | decimal(10,2) | NO   |     | NULL    |                |
| description   | text          | YES  |     | NULL    |                |
| stockQuantity | int           | NO   |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

3. cart table:

• cart_id (Primary Key)

• customer_id (Foreign Key)

• product_id (Foreign Key)

• quantity

```
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| cart_id     | int  | NO   | PRI | NULL    | auto_increment |
| customer_id | int  | YES  | MUL | NULL    |                |
| product_id  | int  | YES  | MUL | NULL    |                |
| quantity    | int  | YES  |     | NULL    |                |
+-------------+------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

4. orders table:

• order_id (Primary Key)

• customer_id (Foreign Key)

• order_date

• total_price

• shipping_address

```
+------------------+---------------+------+-----+---------+----------------+
| Field            | Type          | Null | Key | Default | Extra          |
+------------------+---------------+------+-----+---------+----------------+
| order_id         | int           | NO   | PRI | NULL    | auto_increment |
| customer_id      | int           | YES  | MUL | NULL    |                |
| order_date       | timestamp     | YES  |     | NULL    |                |
| total_price      | decimal(10,2) | YES  |     | NULL    |                |
| shipping_address | varchar(255)  | YES  |     | NULL    |                |
+------------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

5. order_items table (to store order details):

• order_item_id (Primary Key)

• order_id (Foreign Key)

• product_id (Foreign Key)

• quantity

```
+---------------+------+------+-----+---------+----------------+
| Field         | Type | Null | Key | Default | Extra          |
+---------------+------+------+-----+---------+----------------+
| order_item_id | int  | NO   | PRI | NULL    | auto_increment |
| order_id      | int  | YES  | MUL | NULL    |                |
| product_id    | int  | YES  | MUL | NULL    |                |
| quantity      | int  | YES  |     | NULL    |                |
+---------------+------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Cutomers class:

```python
class Customer:
    def __init__(self, customer_id, name, email, password):
        self.__customer_id = customer_id
        self.__name = name
        self.__email = email
        self.__password = password

    # Getters and Setters
    3 usages
    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name
```

```python
def get_email(self):
    return self.__email

def set_email(self, email):
    self.__email = email

def get_password(self):
    return self.__password

def set_password(self, password):
    self.__password = password
```

Products class:

```python
class Product:
    def __init__(self, product_id, name, price, description, stock_quantity):
        self.__product_id = product_id
        self.__name = name
        self.__price = price
        self.__description = description
        self.__stock_quantity = stock_quantity

    # Getters and Setters
    1 usage
    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_price(self):
        return self.__price

    def set_price(self, price):
        self.__price = price
```

```python
def get_description(self):
    return self.__description

def set_description(self, description):
    self.__description = description

def get_stock_quantity(self):
    return self.__stock_quantity

def set_stock_quantity(self, stock_quantity):
    self.__stock_quantity = stock_quantity
```

Cart class:

```python
class Cart:
    def __init__(self, cart_id, customer_id, product_id, quantity):
        self.__cart_id = cart_id
        self.__customer_id = customer_id
        self.__product_id = product_id
        self.__quantity = quantity

    # Getters and Setters
    def get_cart_id(self):
        return self.__cart_id

    def set_cart_id(self, cart_id):
        self.__cart_id = cart_id

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_quantity(self):
        return self.__quantity
```

Orders class:

```python
class Order:
    def __init__(self, order_id, customer_id, order_date, total_price, shipping_address):
        self.__order_id = order_id
        self.__customer_id = customer_id
        self.__order_date = order_date
        self.__total_price = total_price
        self.__shipping_address = shipping_address

    # Getters and Setters
    def get_order_id(self):
        return self.__order_id

    def set_order_id(self, order_id):
        self.__order_id = order_id

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_order_date(self):
        return self.__order_date

    def set_order_date(self, order_date):
        self.__order_date = order_date

    def get_total_price(self):
        return self.__total_price
```

```python
    def set_total_price(self, total_price):
        self.__total_price = total_price

    def get_shipping_address(self):
        return self.__shipping_address

    def set_shipping_address(self, shipping_address):
        self.__shipping_address = shipping_address
```

Order Items class:

```python
class OrderItem:
    def __init__(self, order_item_id, order_id, product_id, quantity):
        self.__order_item_id = order_item_id
        self.__order_id = order_id
        self.__product_id = product_id
        self.__quantity = quantity

    # Getters and Setters
    def get_order_item_id(self):
        return self.__order_item_id

    def set_order_item_id(self, order_item_id):
        self.__order_item_id = order_item_id

    def get_order_id(self):
        return self.__order_id

    def set_order_id(self, order_id):
        self.__order_id = order_id

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def get_quantity(self):
        return self.__quantity
```

Service Provider Interface/Abstract class:

Keep the interfaces and implementation classes in package dao

• Define an OrderProcessorRepository interface/abstract class with methods for adding/removing products to/from the cart and placing orders. The following methods will interact with database.

1.createProduct()

2. createCustomer()

3. deleteProduct()

4. deleteCustomer(customerId)

5. addToCart()

6. removeFromCart()

7. getAllFromCart()

8. placeOrder()

9. getOrdersByCustomer()

```python
class OrderProcessorRepository(ABC):
    @abstractmethod
    def create_product(self) -> bool:
        pass

    @abstractmethod
    def create_customer(self) -> bool:
        pass

    @abstractmethod
    def delete_product(self, product_id: int) -> bool:
        pass

    @abstractmethod
    def delete_customer(self, customer_id: int) -> bool:
        pass

    @abstractmethod
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
        pass

    @abstractmethod
    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        pass
```

```python
    @abstractmethod
    def get_all_from_cart(self, customer: Customer) -> list:
        pass

    @abstractmethod
    def place_order(self, customer: Customer, order_details: list, shipping_address: str) -> bool:
        pass

    @abstractmethod
    def get_orders_by_customer(self, customer_id: int) -> list:
        pass
```

Implement the above interface in a class called OrderProcessorRepositoryImpl in package dao.

1. createProduct()

```
class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    2 usages
    def create_product(self) -> bool:
        cursor = None
        product_name = input("Enter product name: ")
        product_price = float(input("Enter product price: "))
        product_description = input("Enter product description: ")
        stock_quantity = int(input("Enter product stock quantity: "))
        values = (product_name, product_price, product_description, stock_quantity)
        try:
            connection = get_connection()
            cursor = connection.cursor()
            query = "INSERT INTO products (name, price, description, stockQuantity) VALUES (%s, %s, %s, %s)"
            cursor.execute(query, values)
            print("product created successfully")
            connection.commit()
            return True
        except Exception as e:
            print(f"Error creating product: {e}")
            return False
        finally:
            if cursor:
                connection.close()
```

2. createCustomer()

```
def create_customer(self) -> bool:
    cursor = None
    customer_name = input("Enter customer name: ")
    customer_email = input("Enter customer email: ")
    customer_password = input("Enter customer password: ")
    values = (customer_name, customer_email, customer_password)

    try:
        connection = get_connection()
        cursor = connection.cursor()
        query = "INSERT INTO customers (name, email, password) VALUES (%s, %s, %s)"
        cursor.execute(query, values)
        print("customer registered successfully")
        connection.commit()
        return True
    except Exception as e:
        print(f"Error creating customer: {e}")
        return False
    finally:
        if cursor:
            connection.close()
```

3. deleteProduct()

```python
def delete_product(self, product_id: int) -> bool:
    cursor = None
    try:
        connection = get_connection()
        cursor = connection.cursor()
        product_query = "SELECT * FROM products WHERE product_id = %s"
        cursor.execute(product_query, (product_id,))
        existing_product = cursor.fetchone()

        if not existing_product:
            raise Exception(f"Product with ID {product_id} not found")

        query = "DELETE FROM products WHERE product_id = %s"
        cursor.execute(query, (product_id,))
        connection.commit()
        print("Product deleted successfully")
        return True
    except Exception as e:
        print(f"Error deleting product: {e}")
        return False
    finally:
        cursor.close()
        connection.close()
```

4. deleteCustomer(customerId)

```python
def delete_customer(self, customer_id: int) -> bool:
    try:
        connection = get_connection()
        cursor = connection.cursor()



        query = "DELETE FROM customers WHERE customer_id = %s"
        cursor.execute(query, (customer_id,))

        connection.commit()
        return True
    except Exception as e:
        print(f"Error deleting customer: {e}")
        return False
    finally:
        cursor.close()
        connection.close()
```

5. addToCart()

```python
def add_to_cart(self) -> bool:
    connection = get_connection()
    cursor = connection.cursor()
    query = "SELECT * FROM products"
    cursor.execute(query)
    result = cursor.fetchall()
    for row in result:
        print(row)

    cursor = None
    customer_id = input("Enter customer id: ")
    product_id = int(input("Enter the product ID: "))
    quantity = input("Enter the quantity: ")
    cursor = connection.cursor()
    query = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
    values = (customer_id, product_id, quantity)
    try:
        cursor.execute(query, values)
        connection.commit()
        print("product added to cart successfully")
        return True
    except Exception as e:
        print(f"Error adding to cart")
        return False
    finally:
        cursor.close()
        connection.close()
```

6. removeFromCart()

```python
def remove_from_cart(self, customer: Customer, product: Product) -> bool:
    try:
        connection = get_connection()
        cursor = connection.cursor()

        query = "DELETE FROM cart WHERE customer_id = %s AND product_id = %s"
        cursor.execute(query, (customer.customer_id, product.product_id))

        connection.commit()
        return True
    except Exception as e:
        print(f"Error removing from cart: {e}")
        return False
    finally:
        cursor.close()
        connection.close()
```

7. getAllFromCart()

```python
def get_all_from_cart(self) -> list:
    customer_id = int(input("enter the customer ID : "))
    cursor = None
    connection = None
    try:
        connection = get_connection()
        cursor = connection.cursor()

        query = "SELECT p.product_id, p.name, p.price, c.quantity FROM cart c JOIN products p ON c.product_id
        values = (customer_id,)
        cursor.execute(query, values)
        result = cursor.fetchall()
        cart_items = []
        for row in result:
            print(row)
            product = Cart(row[0], row[1], row[2], row[3])
            cart_items.append(product)
        return cart_items
    except Exception as e:
        print(f"Error getting items from cart: {e}")
        return []
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

8. placeOrder()

```python
def place_order(self) -> bool:                                                                    16
    customer_id = int(input("enter customer ID :"))
    shippingAddress = input("enter address: ")
    email = 1
    name = 1
    password = 1
    customer = Customer(customer_id, email, name, password)
    try:
        connection = get_connection()
        cursor = connection.cursor()
        cart_query = "SELECT product_id, quantity FROM cart WHERE customer_id = %s"
        cursor.execute(cart_query, (customer.get_customer_id(),))
        cart_items = cursor.fetchall()
        total_price = 0
        order_details = []
        for item in cart_items:
            product_id, quantity = item
            product_query = "SELECT name, price,description,stockQuantity FROM products WHERE product_id = %s"
            cursor.execute(product_query, (product_id,))
            product_info = cursor.fetchone()
            if product_info:
                product_name, product_price, product_description, product_stock_quantity = product_info
                total_price = total_price + product_price * quantity
                product = Product(product_id, product_name, product_price, product_description,  stock_quantity:
                order_details.append({'product':product, 'quantity': quantity})
        # Insert into orders table
        order_query = "INSERT INTO orders (customer_id, order_date, total_price, shipping_address) VALUES (%s,
        values_order = (customer.get_customer_id(), datetime.now(), total_price, shippingAddress)
        cursor.execute(order_query, values_order)
```

9. getOrdersByCustomer()

```python
def get_orders_by_customer(self) -> list:
    cursor = None
    connection = None
    customer_id = input("enter customer ID : ")
    try:
        connection = get_connection()
        cursor = connection.cursor()

        query = "SELECT o.order_id, o.order_date, o.total_price, oi.product_id, p.name, oi.quantity " \
                "FROM orders o " \
                "JOIN order_items oi ON o.order_id = oi.order_id " \
                "JOIN products p ON oi.product_id = p.product_id " \
                "WHERE o.customer_id = %s"
        cursor.execute(query, (customer_id,))
        result = cursor.fetchall()

        orders = []
        for row in result:
            print(row)
            order_id, order_date, total_price, product_id, product_name, quantity = row
            product = Product(product_id, product_name, price: 0.0, description: 0, stock_quantity: 0)
            order_item = OrderItem(product_name, product_id, quantity, quantity)
            orders.append((order_id, order_date, total_price, order_item))

        return orders
    except Exception as e:
        print(f"Error getting orders by customer: {e}")
        return []
```

Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.

```python
from mysql.connector import connect
from assignments.ecommer.util.PropertyUtil import PropertyUtil

12 usages
def get_connection():
    connection = connect(host='localhost', database='ecom', user='root', password='Kevink25*',
                         port='3306')
    return connection
```

Create the exceptions in package myexceptions and create the following custom exceptions andthrow them in methods whenever needed. Handle all the exceptions in main method,

CustomerNotFoundException

ProductNotFoundException

OrderNotFoundException

```
class CustomerNotFoundException(Exception):
    pass

class ProductNotFoundException(Exception):
    pass

class OrderNotFoundException(Exception):
    pass
```

Create class named EcomApp with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.

1. Register Customer.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer Order

```
class EcomApp:
    1 usage
    @staticmethod
    def main():
        order_processor_repository = OrderProcessorRepositoryImpl()

        while True:
            print("\nE-commerce Application Menu:")
            print("1. Register Customer.")
            print("2. Create Product.")
            print("3. Delete Product.")
            print("4. Add to cart.")
            print("5. View cart.")
            print("6. Place order.")
            print("7. View Customer Order.")
            print("8. Exit.")
```

```
    choice = input("Enter your choice: ")

    if choice == '1':
        # Register Customer
        order_processor_repository.create_customer()

    elif choice == '2':
        # Create Product
        order_processor_repository.create_product()

    elif choice == '3':
        # Delete Product
        product_id = int(input("Enter product ID to delete: "))
        order_processor_repository.delete_product(product_id)

    elif choice == '4':
        order_processor_repository.add_to_cart()

    elif choice == '5':
        order_processor_repository.get_all_from_cart()

    elif choice == '6':

        order_processor_repository.place_order()
```

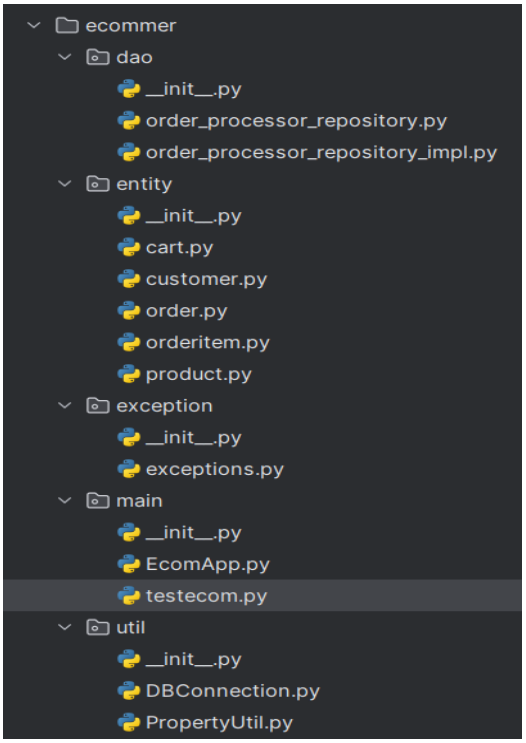
```
            elif choice == '7':
                order_processor_repository.get_orders_by_customer()

            elif choice == '8':
                print("Exiting E-commerce Application.")
                break

            else:
                print("Invalid choice. Please enter a valid option.")

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

Create Unit test cases for Ecommerce System are essential to ensure the correctness andreliability of your system. Following questions to guide the creation of Unit test cases:

- Write test case to test Product created successfully or not.
- Write test case to test product is added to cart successfully or not.
- Write test case to test product is ordered successfully or not.

```
import unittest
from unittest.mock import patch
from assignments.ecommer.dao.order_processor_repository_impl import OrderProcessorRepositoryImpl
from assignments.ecommer.entity.customer import Customer
from assignments.ecommer.entity.product import Product

class TestOrderProcessorRepositoryImpl(unittest.TestCase):

    @patch( target: 'builtins.input', side_effect=['19', '33', '2'])
    def test_add_to_cart_success(self, mock_input):
        order_processor_repository = OrderProcessorRepositoryImpl()
        result = order_processor_repository.add_to_cart()
        self.assertTrue(result)

    @patch( target: 'builtins.input', side_effect=['19', '123 Main St, City'])
    def test_place_order_success(self, mock_input):
        order_processor_repository = OrderProcessorRepositoryImpl()
        result = order_processor_repository.place_order()
        self.assertTrue(result)

    @patch( target: 'builtins.input', side_effect=['bars', '555.50', 'Heavy steel', '10'])
    def test_create_product_success(self, mock_input):
        order_processor_repository = OrderProcessorRepositoryImpl()
        result = order_processor_repository.create_product()
        self.assertTrue(result)


if __name__ == '__main__':
    unittest.main()
```

Package Management:


ecommer
dao
__init__.py
order_processor_repository.py
order_processor_repository_impl.py
entity
__init__.py
cart.py
customer.py
order.py
orderitem.py
product.py
exception
__init__.py
exceptions.py
main
__init__.py
EcomApp.py
testecom.py
util
__init__.py
DBConnection.py
PropertyUtil.py