DB Lab Project Phase 2

By:
Hashir Hassan F2023332040
Muhammad Ahab Raza F2023332016

UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
UMT
Estd. 1990

Contents

1. Introduction:

Cash & Carry Management System is a database oriented project to efficiently manage the primary functions of a retail or wholesale store. This system is made in such a manner so as to help create smooth day-to-day operations concerning human resources, inventories, customers, suppliers, sales and purchase orders.

In this project, PostgreSQL is used to maintain the backend database, while Python is employed for the CLI based frontend. This arrangement provides the users with a simple but usable way of interacting with the system. With the help of Python integrated with PostgreSQL, the system thus supports inserting, retrieving, and updating real-time data while retaining data consistency and integrity.

2. Recap of Phase 1:

The first phase of the project implemented the basic design of the database system. This encompassed the following:

2.1. **Problem Identification:** The real-world situation was identified wherein manual handling of store operations could be made inefficient, erroneous, and untraceable.

2.2. **Entity Details:** These are entities with their attributes, such as name, phone_no, salary, unit_price, etc.

2.3. **Entity-Relationship Diagram (ERD):** An ERD must have been created by using either Chen or Crow's Foot notation to visually present entities such as Employee, Customer, Product, Sale, Supplier, etc. and their relationships.

2.4. **Relational Model:** From the entity, relational tables were derived along with suitable data types and constraints like primary key, foreign key, unique constraints to maintain data integrity.

2.5. **Design Validation:** The schema was validated to ensure that all design specifications were fulfilled such as attaching a sale to a customer, purchase to a supplier, and managing inventory from products via line items.

This phase created the foundation for the actual development and implementation of the next phase using SQL and Python.

3. Database Schema Implementation

The database schema for the Cash & Carry Management System was implemented in **PostgreSQL**. It consists of multiple interrelated tables that collectively support the core functionalities of a retail system such as managing employees, tracking inventory, handling customer and supplier information, recording sales, and processing purchase orders.

3.1. Tables Created

Below is a list of the main tables and their purposes:

- **employee** – Stores information about store employees (name, role, salary, etc.)

  Code:

```sql
-- Employee table
CREATE TABLE employee(
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    role VARCHAR(100),
    dob DATE,
    salary INT
);
```

- **address** – Stores address details linked to each employee

  Code:

```sql
-- Address table
CREATE TABLE address(
    id SERIAL PRIMARY KEY,
    employee_id INT REFERENCES employee(id),
    house_no VARCHAR(10) NOT NULL,
    street_no VARCHAR(10) NOT NULL,
    city VARCHAR(30) NOT NULL,
    province VARCHAR(30) NOT NULL,
    zipcode VARCHAR(30)
);
```

- **customer** – Holds customer information including contact details

  Code:

```sql
-- Customer table
CREATE TABLE customer(
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone_no VARCHAR(20) NOT NULL UNIQUE
);
```

- **supplier** – Maintains supplier data who provide inventory

  Code:

```sql
CREATE TABLE supplier(
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone_no VARCHAR(20) NOT NULL UNIQUE
);
```

- **category** – Organizes products into defined categories

  Code:

```sql
CREATE TABLE category(
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description VARCHAR(255)
);
```

- **product** – Stores product details like name, pricing, and stock quantity

  Code:

```sql
-- Product table
CREATE TABLE product(
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    unit_price NUMERIC(10,2),
    cost_price NUMERIC(10,2),
    quantity INT,
    category_id INT REFERENCES category(id)
);
```

- **sale** – Records each sale with reference to the employee and customer

  Code:

```sql
-- Sale table
CREATE TABLE sale(
    id SERIAL PRIMARY KEY,
    date DATE DEFAULT CURRENT_DATE,
    total_amount NUMERIC(10,2),
    employee_id INT REFERENCES employee(id),
    customer_id INT REFERENCES customer(id)
);
```

- **sale_line_item** – Stores item-level details of each sale (product, quantity, price)

  Code:

```sql
-- Sale Line Item table
CREATE TABLE sale_line_item(
    sale_id INT REFERENCES sale(id),
    product_id INT REFERENCES product(id),
    quantity INT NOT NULL,
    price NUMERIC(10,2) NOT NULL,
    PRIMARY KEY (sale_id, product_id)
);
```

- **purchase_order** – Records purchase orders placed to suppliers

  Code:

```sql
-- Purchase Order Table
CREATE TABLE purchase_order(
    id SERIAL PRIMARY KEY,
    date DATE DEFAULT CURRENT_DATE,
    total_cost NUMERIC(10,2),
    employee_id     INT REFERENCES employee(id),
    supplier_id     INT REFERENCES supplier(id)
);
```

- **purchase_item** – Item-level details of purchase orders

  Code:

```sql
-- Purchase Item Table
CREATE TABLE purchase_item(
    purchase_order_id INT REFERENCES purchase_order(id),
    product_id INT REFERENCES product(id),
    quantity INT NOT NULL,
    cost_price NUMERIC(10,2) NOT NULL,
    PRIMARY KEY (purchase_order_id, product_id)
);
```

4. Python CLI Overview

To interact with the PostgreSQL database, a **Command Line Interface (CLI)** was developed in **Python**. This interface allows users to perform various operations related to store management directly from the terminal, simulating real-time business activities without the need for a graphical UI.

4.1. Purpose

The CLI provides a simple and efficient way to:

- Add, update, delete, and view data
- Record sales and purchases
- Manage inventory levels
- Connect and work seamlessly with the database backend

4.2. Technologies Used

- **Python** for logic and user interaction
- **psycopg2** library for database connectivity with PostgreSQL

4.3. Structure

The system is modular, with each core feature separated into different functions and files for clarity and reusability. The main menu routes to submenus like:

4.3.1. **Manage Employees**: Add/view/update/delete employees and manage their addresses

Code:

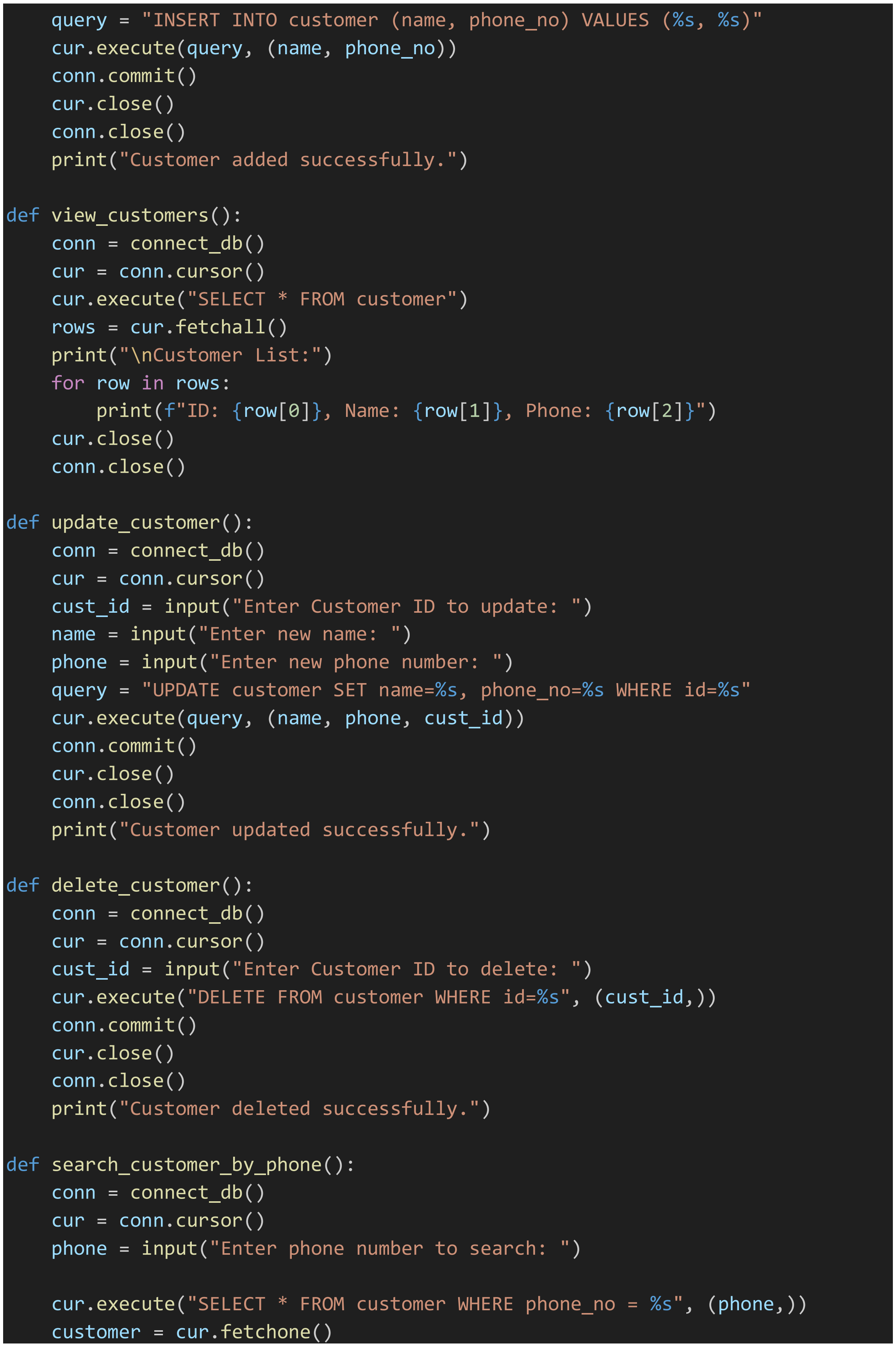
```python
def manage_employees():
    while True:
        print("\n--- Manage Employees ---")
        print("1. Add Employee")
        print("2. View All Employees")
        print("3. Update Employee")
        print("4. Delete Employee")
        print("5. Manage Addresses")
        print("6. Back to Main Menu")

        choice = input("Choose an option: ")

        if choice == '1':
            add_employee()
        elif choice == '2':
            view_employees()
        elif choice == '3':
            update_employee()
        elif choice == '4':
            delete_employee()
        elif choice == '5':
            manage_employee_addresses()
        elif choice == '6':
            break
        else:
            print("Invalid input. Try again.")

def add_employee():
    name = input("Enter name: ")
    role = input("Enter role: ")
    dob = input("Enter date of birth (YYYY-MM-DD): ")
    salary = input("Enter salary: ")

    try:
        conn = connect_db()
        cur = conn.cursor()
        cur.execute(
            "INSERT INTO employee (name, role, dob, salary) VALUES (%s, %s,
%s, %s)",
            (name, role, dob, salary)
        )
        conn.commit()
        print("Employee added successfully.")
    except Exception as e:
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def view_employees():
    try:
        conn = connect_db()
        cur = conn.cursor()
        cur.execute("SELECT * FROM employee")
        rows = cur.fetchall()
        print("\n--- Employee List ---")
        for row in rows:
            print(f"ID: {row[0]}, Name: {row[1]}, Role: {row[2]}, DOB:
{row[3]}, Salary: {row[4]}")
    except Exception as e:
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def update_employee():
    emp_id = input("Enter Employee ID to update: ")
    new_salary = input("Enter new salary: ")

    try:
        conn = connect_db()
        cur = conn.cursor()
        cur.execute("UPDATE employee SET salary = %s WHERE id = %s",
(new_salary, emp_id))
        if cur.rowcount == 0:
            print("Employee not found.")
        else:
            conn.commit()
            print("Employee updated successfully.")
    except Exception as e:
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def delete_employee():
    emp_id = input("Enter Employee ID to delete: ")

    try:
        conn = connect_db()
        cur = conn.cursor()
        cur.execute("DELETE FROM employee WHERE id = %s", (emp_id,))
        if cur.rowcount == 0:
            print("Employee not found.")
        else:
            conn.commit()
            print("Employee deleted successfully.")
    except Exception as e:
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def manage_employee_addresses():
    while True:
        print("\n--- Manage Employee Addresses ---")
        print("1. Add Address")
        print("2. View Addresses")
        print("3. Back")

        choice = input("Select an option: ")

        if choice == '1':
            add_address()
        elif choice == '2':
            view_addresses()
        elif choice == '3':
            break
        else:
            print("Invalid option. Try again.")
def add_address():
    conn = connect_db()
    cur = conn.cursor()
    emp_id = input("Enter Employee ID: ")
    house_no = input("Enter House No: ")
    street_no = input("Enter Street No: ")
    city = input("Enter City: ")
    province = input("Enter Province: ")
    zipcode = input("Enter Zipcode: ")

    query = """
        INSERT INTO address (employee_id, house_no, street_no, city, province,
zipcode)
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    cur.execute(query, (emp_id, house_no, street_no, city, province, zipcode))
    conn.commit()
    cur.close()
    conn.close()
    print("Address added successfully.")
```

```python
def view_addresses():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM address")
    rows = cur.fetchall()
    for row in rows:
        print(row)
    cur.close()
    conn.close()
```

4.3.2. **Manage Customers**: Perform customer CRUD operations and search by phone number

Code:

```python
def manage_customers():
    while True:
        print("\n--- Manage Customers ---")
        print("1. Add Customer")
        print("2. View Customers")
        print("3. Update Customer")
        print("4. Delete Customer")
        print("5. Search Customer by Phone Number")
        print("6. Back to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            add_customer()
        elif choice == '2':
            view_customers()
        elif choice == '3':
            update_customer()
        elif choice == '4':
            delete_customer()
        elif choice == '5':
            search_customer_by_phone()
        elif choice == '6':
            break
        else:
            print("Invalid option. Try again.")

def add_customer():
    conn = connect_db()
    cur = conn.cursor()
    name = input("Enter Customer Name: ")
    phone_no = input("Enter Phone Number: ")
```



```python
    query = "INSERT INTO customer (name, phone_no) VALUES (%s, %s)"
    cur.execute(query, (name, phone_no))
    conn.commit()
    cur.close()
    conn.close()
    print("Customer added successfully.")

def view_customers():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM customer")
    rows = cur.fetchall()
    print("\nCustomer List:")
    for row in rows:
        print(f"ID: {row[0]}, Name: {row[1]}, Phone: {row[2]}")
    cur.close()
    conn.close()

def update_customer():
    conn = connect_db()
    cur = conn.cursor()
    cust_id = input("Enter Customer ID to update: ")
    name = input("Enter new name: ")
    phone = input("Enter new phone number: ")
    query = "UPDATE customer SET name=%s, phone_no=%s WHERE id=%s"
    cur.execute(query, (name, phone, cust_id))
    conn.commit()
    cur.close()
    conn.close()
    print("Customer updated successfully.")

def delete_customer():
    conn = connect_db()
    cur = conn.cursor()
    cust_id = input("Enter Customer ID to delete: ")
    cur.execute("DELETE FROM customer WHERE id=%s", (cust_id,))
    conn.commit()
    cur.close()
    conn.close()
    print("Customer deleted successfully.")

def search_customer_by_phone():
    conn = connect_db()
    cur = conn.cursor()
    phone = input("Enter phone number to search: ")

    cur.execute("SELECT * FROM customer WHERE phone_no = %s", (phone,))
    customer = cur.fetchone()
    if customer:
        print(f"\nCustomer Found - ID: {customer[0]}, Name: {customer[1]}, 
Phone: {customer[2]}")
    else:
        print("No customer found with that phone number.")

    cur.close()
    conn.close()
```

4.3.3. **Manage Categories**: Create, view, update and delete product categories

Code:

```
def manage_categories():
    while True:
        print("\n--- Manage Categories ---")
        print("1. Add Category")
        print("2. View Categories")
        print("3. Update Category")
        print("4. Delete Category")
        print("5. Back to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            add_category()
        elif choice == '2':
            view_categories()
        elif choice == '3':
            update_category()
        elif choice == '4':
            delete_category()
        elif choice == '5':
            break
        else:
            print("Invalid option. Try again.")
def add_category():
    conn = connect_db()
    cur = conn.cursor()

    name = input("Enter category name: ")
    desc = input("Enter category description (optional): ")

    cur.execute("INSERT INTO category (name, description) VALUES (%s, %s)", 
(name, desc))
    conn.commit()

    print("Category added successfully.")
```

```python
    cur.close()
    conn.close()


def view_categories():
    conn = connect_db()
    cur = conn.cursor()

    cur.execute("SELECT * FROM category")
    rows = cur.fetchall()

    print("\n--- Categories ---")
    for row in rows:
        print(f"ID: {row[0]}, Name: {row[1]}, Description: {row[2]}")

    cur.close()
    conn.close()


def update_category():
    conn = connect_db()
    cur = conn.cursor()

    cat_id = input("Enter category ID to update: ")
    name = input("Enter new name: ")
    desc = input("Enter new description: ")

    cur.execute("UPDATE category SET name = %s, description = %s WHERE id =
%s", (name, desc, cat_id))
    conn.commit()

    print("Category updated successfully.")

    cur.close()
    conn.close()


def delete_category():
    conn = connect_db()
    cur = conn.cursor()

    cat_id = input("Enter category ID to delete: ")

    cur.execute("DELETE FROM category WHERE id = %s", (cat_id,))
    conn.commit()

    print("Category deleted successfully.")
    cur.close()
    conn.close()
```

4.3.4. **Manage Products**: Handle inventory, including stock, pricing, and category linkage

Code:

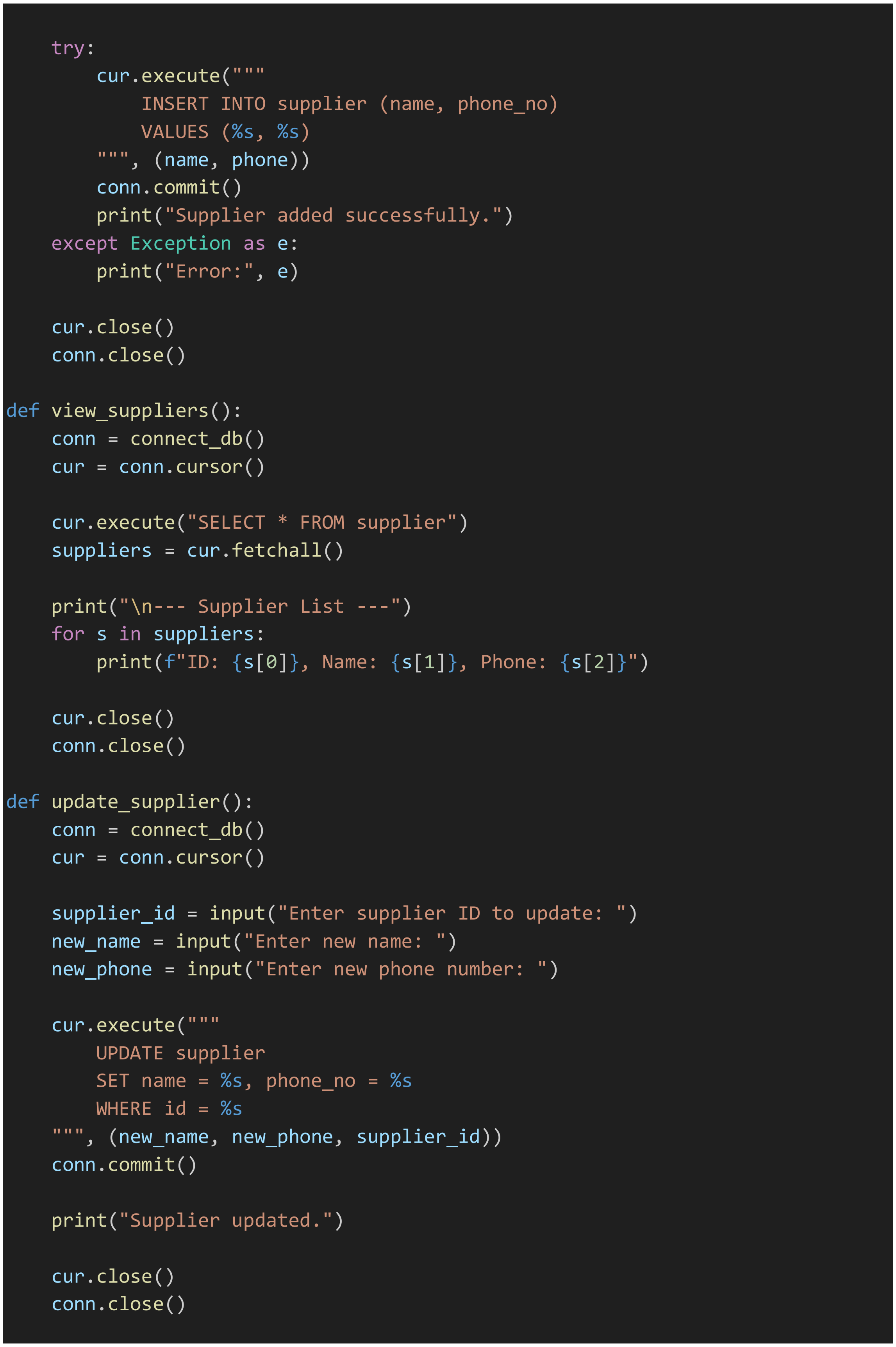
```
def manage_products():
    while True:
        print("\n--- Manage Products ---")
        print("1. Add Product")
        print("2. View Products")
        print("3. Update Product")
        print("4. Delete Product")
        print("5. Search by Name")
        print("6. Back to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            add_product()
        elif choice == '2':
            view_products()
        elif choice == '3':
            update_product()
        elif choice == '4':
            delete_product()
        elif choice == '5':
            search_product_by_name()
        elif choice == '6':
            break
        else:
            print("Invalid option. Try again.")


def add_product():
    conn = connect_db()
    cur = conn.cursor()

    # Show available categories
    cur.execute("SELECT id, name FROM category")
    categories = cur.fetchall()

    print("\nAvailable Categories:")
    for cat in categories:
        print(f"  ID: {cat[0]}, Name: {cat[1]}")
```

```python
    name = input("\nEnter product name: ")
    unit_price = float(input("Enter unit price: "))
    cost_price = float(input("Enter cost price: "))
    quantity = int(input("Enter quantity: "))
    category_id = input("Enter category ID from above: ")

    # Validate category ID
    cur.execute("SELECT id FROM category WHERE id = %s", (category_id,))
    if cur.fetchone() is None:
        print("Error: Category ID does not exist. Product not added.")
    else:
        cur.execute("""
            INSERT INTO product (name, unit_price, cost_price, quantity, 
category_id)
            VALUES (%s, %s, %s, %s, %s)
        """, (name, unit_price, cost_price, quantity, category_id))
        conn.commit()
        print("Product added successfully.")

    cur.close()
    conn.close()




def view_products():
    conn = connect_db()
    cur = conn.cursor()

    cur.execute("""
        SELECT p.id, p.name, p.unit_price, p.cost_price, p.quantity, c.name AS 
category
        FROM product p
        LEFT JOIN category c ON p.category_id = c.id
    """)
    rows = cur.fetchall()

    print("\n--- Product List ---")
    for row in rows:
        print(f"ID: {row[0]}, Name: {row[1]}, Unit Price: {row[2]}, Cost 
Price: {row[3]}, Quantity: {row[4]}, Category: {row[5]}")

    cur.close()
    conn.close()


def update_product():
    conn = connect_db()
    cur = conn.cursor()

    prod_id = input("Enter product ID to update: ")
    name = input("New name: ")
    unit_price = float(input("New unit price: "))
    cost_price = float(input("New cost price: "))
    quantity = int(input("New quantity: "))
    category_id = input("New category ID: ")

    cur.execute("""
        UPDATE product
        SET name = %s, unit_price = %s, cost_price = %s, quantity = %s,
category_id = %s
        WHERE id = %s
    """, (name, unit_price, cost_price, quantity, category_id, prod_id))

    conn.commit()
    print("Product updated successfully.")

    cur.close()
    conn.close()
def delete_product():
    conn = connect_db()
    cur = conn.cursor()

    prod_id = input("Enter product ID to delete: ")

    cur.execute("DELETE FROM product WHERE id = %s", (prod_id,))
    conn.commit()

    print("Product deleted successfully.")

    cur.close()
    conn.close()


def search_product_by_name():
    conn = connect_db()
    cur = conn.cursor()

    search = input("Enter product name to search: ")
    cur.execute("""
        SELECT id, name, unit_price, quantity
        FROM product
        WHERE name ILIKE %s
    """, ('%' + search + '%',))

    rows = cur.fetchall()
    if rows:
        print("\n--- Search Results ---")
        for row in rows:
            print(f"ID: {row[0]}, Name: {row[1]}, Price: {row[2]}, Quantity: 
{row[3]}")
    else:
        print("No products found.")

    cur.close()
    conn.close()
```

4.3.5. **Manage Suppliers:** Add or manage supplier records

Code:

```python
def manage_suppliers():
    while True:
        print("\n--- Manage Suppliers ---")
        print("1. Add Supplier")
        print("2. View All Suppliers")
        print("3. Update Supplier")
        print("4. Delete Supplier")
        print("5. Search Supplier by Phone Number")
        print("6. Back to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            add_supplier()
        elif choice == '2':
            view_suppliers()
        elif choice == '3':
            update_supplier()
        elif choice == '4':
            delete_supplier()
        elif choice == '5':
            search_supplier_by_phone()
        elif choice == '6':
            break
        else:
            print("Invalid choice, try again.")

def add_supplier():
    conn = connect_db()
    cur = conn.cursor()

    name = input("Enter supplier name: ")
    phone = input("Enter supplier phone number: ")
```



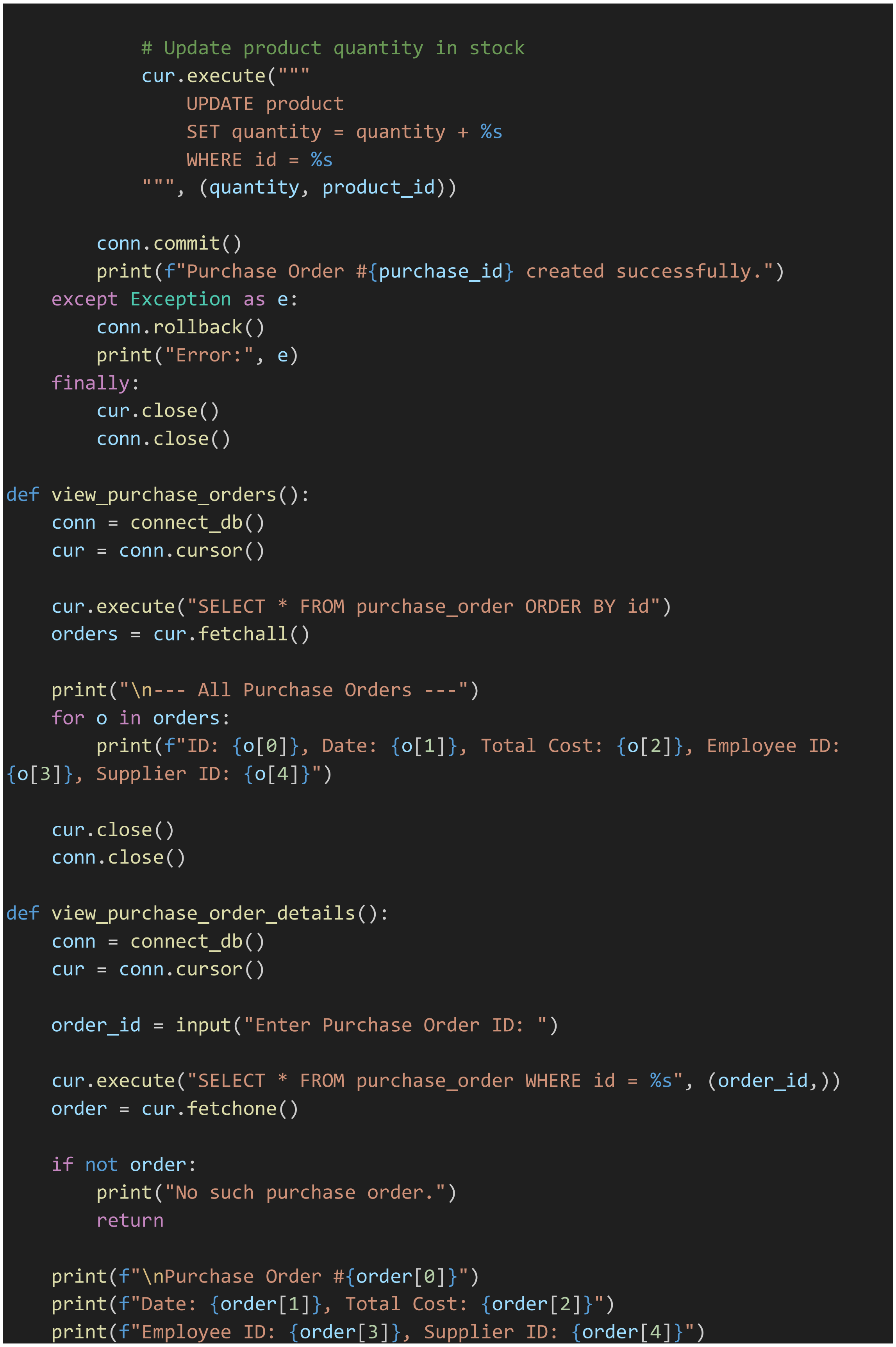
```python
    try:
        cur.execute("""
            INSERT INTO supplier (name, phone_no)
            VALUES (%s, %s)
        """, (name, phone))
        conn.commit()
        print("Supplier added successfully.")
    except Exception as e:
        print("Error:", e)

    cur.close()
    conn.close()

def view_suppliers():
    conn = connect_db()
    cur = conn.cursor()

    cur.execute("SELECT * FROM supplier")
    suppliers = cur.fetchall()

    print("\n--- Supplier List ---")
    for s in suppliers:
        print(f"ID: {s[0]}, Name: {s[1]}, Phone: {s[2]}")

    cur.close()
    conn.close()

def update_supplier():
    conn = connect_db()
    cur = conn.cursor()

    supplier_id = input("Enter supplier ID to update: ")
    new_name = input("Enter new name: ")
    new_phone = input("Enter new phone number: ")

    cur.execute("""
        UPDATE supplier
        SET name = %s, phone_no = %s
        WHERE id = %s
    """, (new_name, new_phone, supplier_id))
    conn.commit()

    print("Supplier updated.")

    cur.close()
    conn.close()
```

```python
def delete_supplier():
    conn = connect_db()
    cur = conn.cursor()

    supplier_id = input("Enter supplier ID to delete: ")
    cur.execute("DELETE FROM supplier WHERE id = %s", (supplier_id,))
    conn.commit()

    print("Supplier deleted.")

    cur.close()
    conn.close()

def search_supplier_by_phone():
    conn = connect_db()
    cur = conn.cursor()

    phone = input("Enter phone number to search: ")
    cur.execute("SELECT * FROM supplier WHERE phone_no = %s", (phone,))
    supplier = cur.fetchone()

    if supplier:
        print(f"Found: ID: {supplier[0]}, Name: {supplier[1]}, Phone: 
{supplier[2]}")
    else:
        print("No supplier found with that phone number.")

    cur.close()
    conn.close()
```

4.3.6. **Manage Purchase Orders**: Record supplier orders along with line items

Code:

```python
def manage_purchases():
    while True:
        print("\n--- Manage Purchase Orders ---")
        print("1. Create Purchase Order")
        print("2. View All Purchase Orders")
        print("3. View Purchase Order Details")
        print("4. Delete Purchase Order")
        print("5. Back to Main Menu")

        choice = input("Select an option: ")

        if choice == '1':
            create_purchase_order()
        elif choice == '2':
            view_purchase_orders()
        elif choice == '3':
            view_purchase_order_details()
        elif choice == '4':
            delete_purchase_order()
        elif choice == '5':
            break
        else:
            print("Invalid option. Please try again.")


def create_purchase_order():
    conn = connect_db()
    cur = conn.cursor()

    employee_id = input("Enter employee ID: ")
    supplier_id = input("Enter supplier ID: ")

    purchase_items = []
    total_cost = 0

    while True:
        product_id = input("Enter product ID: ")
        quantity = int(input("Enter quantity: "))
        cost_price = float(input("Enter cost price per unit: "))

        item_total = quantity * cost_price
        total_cost += item_total

        purchase_items.append((product_id, quantity, cost_price))

        more = input("Add another item? (y/n): ")
        if more.lower() != 'y':
            break

    try:
        # Insert into purchase_order table
        cur.execute("""
            INSERT INTO purchase_order (employee_id, supplier_id, total_cost)
            VALUES (%s, %s, %s) RETURNING id
        """, (employee_id, supplier_id, total_cost))
        purchase_id = cur.fetchone()[0]

        # Insert into purchase_item table
        for product_id, quantity, cost_price in purchase_items:
            cur.execute("""
                INSERT INTO purchase_item (purchase_order_id, product_id,
quantity, cost_price)
                VALUES (%s, %s, %s, %s)
            """, (purchase_id, product_id, quantity, cost_price))
```



```
            # Update product quantity in stock
            cur.execute("""
                UPDATE product
                SET quantity = quantity + %s
                WHERE id = %s
            """, (quantity, product_id))

        conn.commit()
        print(f"Purchase Order #{purchase_id} created successfully.")
    except Exception as e:
        conn.rollback()
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def view_purchase_orders():
    conn = connect_db()
    cur = conn.cursor()

    cur.execute("SELECT * FROM purchase_order ORDER BY id")
    orders = cur.fetchall()

    print("\n--- All Purchase Orders ---")
    for o in orders:
        print(f"ID: {o[0]}, Date: {o[1]}, Total Cost: {o[2]}, Employee ID:
{o[3]}, Supplier ID: {o[4]}")

    cur.close()
    conn.close()

def view_purchase_order_details():
    conn = connect_db()
    cur = conn.cursor()

    order_id = input("Enter Purchase Order ID: ")

    cur.execute("SELECT * FROM purchase_order WHERE id = %s", (order_id,))
    order = cur.fetchone()

    if not order:
        print("No such purchase order.")
        return

    print(f"\nPurchase Order #{order[0]}")
    print(f"Date: {order[1]}, Total Cost: {order[2]}")
    print(f"Employee ID: {order[3]}, Supplier ID: {order[4]}")
    cur.execute("""
        SELECT product_id, quantity, cost_price FROM purchase_item
        WHERE purchase_order_id = %s
    """, (order_id,))
    items = cur.fetchall()

    print("--- Items ---")
    for item in items:
        print(f"Product ID: {item[0]}, Qty: {item[1]}, Cost/unit: {item[2]}")

    cur.close()
    conn.close()


def delete_purchase_order():
    conn = connect_db()
    cur = conn.cursor()

    order_id = input("Enter Purchase Order ID to delete: ")

    try:
        cur.execute("DELETE FROM purchase_item WHERE purchase_order_id = %s",
(order_id,))
        cur.execute("DELETE FROM purchase_order WHERE id = %s", (order_id,))
        conn.commit()
        print("Purchase order deleted.")
    except Exception as e:
        conn.rollback()
        print("Error:", e)
    finally:
        cur.close()
        conn.close()
```

4.3.7. **Manage Sales**: Process sales, update product stock, and calculate totals

Code:

```
def manage_sales():
    while True:
        print("\n--- Manage Sales ---")
        print("1. Create Sale")
        print("2. View All Sales")
        print("3. View Sale Details")
        print("4. Delete Sale")
        print("5. Back to Main Menu")

        choice = input("Select an option: ")
```

```python
        if choice == '1':
            create_sale()
        elif choice == '2':
            view_sales()
        elif choice == '3':
            view_sale_details()
        elif choice == '4':
            delete_sale()
        elif choice == '5':
            break
        else:
            print("Invalid option. Please try again.")

def create_sale():
    conn = connect_db()
    cur = conn.cursor()

    employee_id = input("Enter employee ID: ")
    customer_id = input("Enter customer ID: ")

    sale_items = []
    total_amount = 0

    while True:
        product_id = input("Enter product ID: ")
        quantity = int(input("Enter quantity: "))

        cur.execute("SELECT unit_price, quantity FROM product WHERE id = %s",
(product_id,))
        result = cur.fetchone()
        if not result:
            print("Product not found.")
            continue
        unit_price, available_qty = result

        if quantity > available_qty:
            print("Not enough stock.")
            continue

        item_total = quantity * unit_price
        total_amount += item_total

        sale_items.append((product_id, quantity, unit_price))

        more = input("Add another item? (y/n): ")
        if more.lower() != 'y':
            break
    try:

        cur.execute("""
            INSERT INTO sale (employee_id, customer_id, total_amount)
            VALUES (%s, %s, %s) RETURNING id
        """, (employee_id, customer_id, total_amount))
        sale_id = cur.fetchone()[0]

        for product_id, quantity, price in sale_items:
            cur.execute("""
                INSERT INTO sale_line_item (sale_id, product_id, quantity, 
price)
                VALUES (%s, %s, %s, %s)
            """, (sale_id, product_id, quantity, price))

            cur.execute("""
                UPDATE product
                SET quantity = quantity - %s
                WHERE id = %s
            """, (quantity, product_id))

        conn.commit()
        print(f"Sale #{sale_id} created successfully.")
    except Exception as e:
        conn.rollback()
        print("Error:", e)
    finally:
        cur.close()
        conn.close()

def view_sales():
    conn = connect_db()
    cur = conn.cursor()

    cur.execute("SELECT * FROM sale ORDER BY id")
    sales = cur.fetchall()

    print("\n--- All Sales ---")
    for s in sales:
        print(f"ID: {s[0]}, Date: {s[1]}, Total: {s[2]}, Employee ID: {s[3]}, 
Customer ID: {s[4]}")

    cur.close()
    conn.close()

def view_sale_details():
    conn = connect_db()
    cur = conn.cursor()
    sale_id = input("Enter Sale ID: ")

    cur.execute("SELECT * FROM sale WHERE id = %s", (sale_id,))
    sale = cur.fetchone()

    if not sale:
        print("Sale not found.")
        return

    print(f"\nSale #{sale[0]} | Date: {sale[1]}, Total: {sale[2]}")
    print(f"Employee ID: {sale[3]}, Customer ID: {sale[4]}")

    cur.execute("""
        SELECT product_id, quantity, price FROM sale_line_item
        WHERE sale_id = %s
    """, (sale_id,))
    items = cur.fetchall()

    print("--- Items ---")
    for item in items:
        print(f"Product ID: {item[0]}, Qty: {item[1]}, Price/unit: {item[2]}")

    cur.close()
    conn.close()

def delete_sale():
    conn = connect_db()
    cur = conn.cursor()

    sale_id = input("Enter Sale ID to delete: ")

    try:
        cur.execute("DELETE FROM sale_line_item WHERE sale_id = %s",
(sale_id,))
        cur.execute("DELETE FROM sale WHERE id = %s", (sale_id,))
        conn.commit()
        print("Sale deleted.")
    except Exception as e:
        conn.rollback()
        print("Error:", e)
    finally:
        cur.close()
        conn.close()
```

Each of these modules includes error handling, user prompts, and database validations, ensuring that no invalid data is added (e.g., preventing product insertion if the category doesn't exist).

4.4. Benefits

- Keeps the system interactive and easy to test
- Can be later extended into a full GUI or web application if needed

5. Integration with PostgreSQL and Main Menu

The Python CLI is connected to the PostgreSQL database using the psycopg2 library. A dedicated function was created to establish the database connection and handle queries securely.

5.1. Database Connection

A reusable connection function is implemented at the start of the program to simplify interaction with the database from different modules.

Code:

```
import psycopg2
def connect_db():
    return psycopg2.connect(
        dbname="CashnCarry",
        user="postgres",
        password="1234",
        host="localhost",
        port="5433"
    )
```

5.2. Main Menu Navigation

The CLI begins with a main menu that allows the user to choose between various system modules. Based on the user's choice, control is routed to the relevant submenu for employee management, customer handling, product operations, etc.

Code:

```
def main_menu():
    while True:
        print("\n=== Cash & Carry Management System ===")
        print("1. Manage Employees")
        print("2. Manage Customers")
        print("3. Manage Category")
        print("4. Manage Products")
        print("5. Manage Suppliers")
        print("6. Manage Purchase Orders")
        print("7. Manage Sales")

        print("8. Exit")

        choice = input("Select an option: ")
```

```python
        if choice == '1':
            manage_employees()
        elif choice == '2':
            manage_customers()
        elif choice == '3':
            manage_categories()
        elif choice == '4':
            manage_products()
        elif choice == '5':
            manage_suppliers()
        elif choice == '6':
            manage_purchases()
        elif choice == '7':
            manage_sales()
        elif choice == '8':
            print("Exiting... Goodbye!")
            break
        else:
            print("Invalid option. Please try again.")
```

Each submenu supports full CRUD operations and is stored in separate files or organized modularly inside the same script for better readability and scalability.

6. Conclusion

The Cash & Carry Management System project successfully demonstrates the core functionalities of a small retail business. Through the use of PostgreSQL for a normalized and well-structured database and a Python CLI for user interaction, the system provides:

- Efficient data handling and integrity via proper relational constraints
- Interactive control over operations such as product management, sales, and purchases
- Scalability for future enhancement like adding a GUI or real-time analytics

This phase not only helped reinforce the concepts of database design, ER modeling, and SQL implementation but also provided practical experience in integrating Python with a real-world database. The completed project can serve as a foundation for further academic or personal development in the field of database applications.

-- The End --