Ex. No.: 9
Date: / / 2024

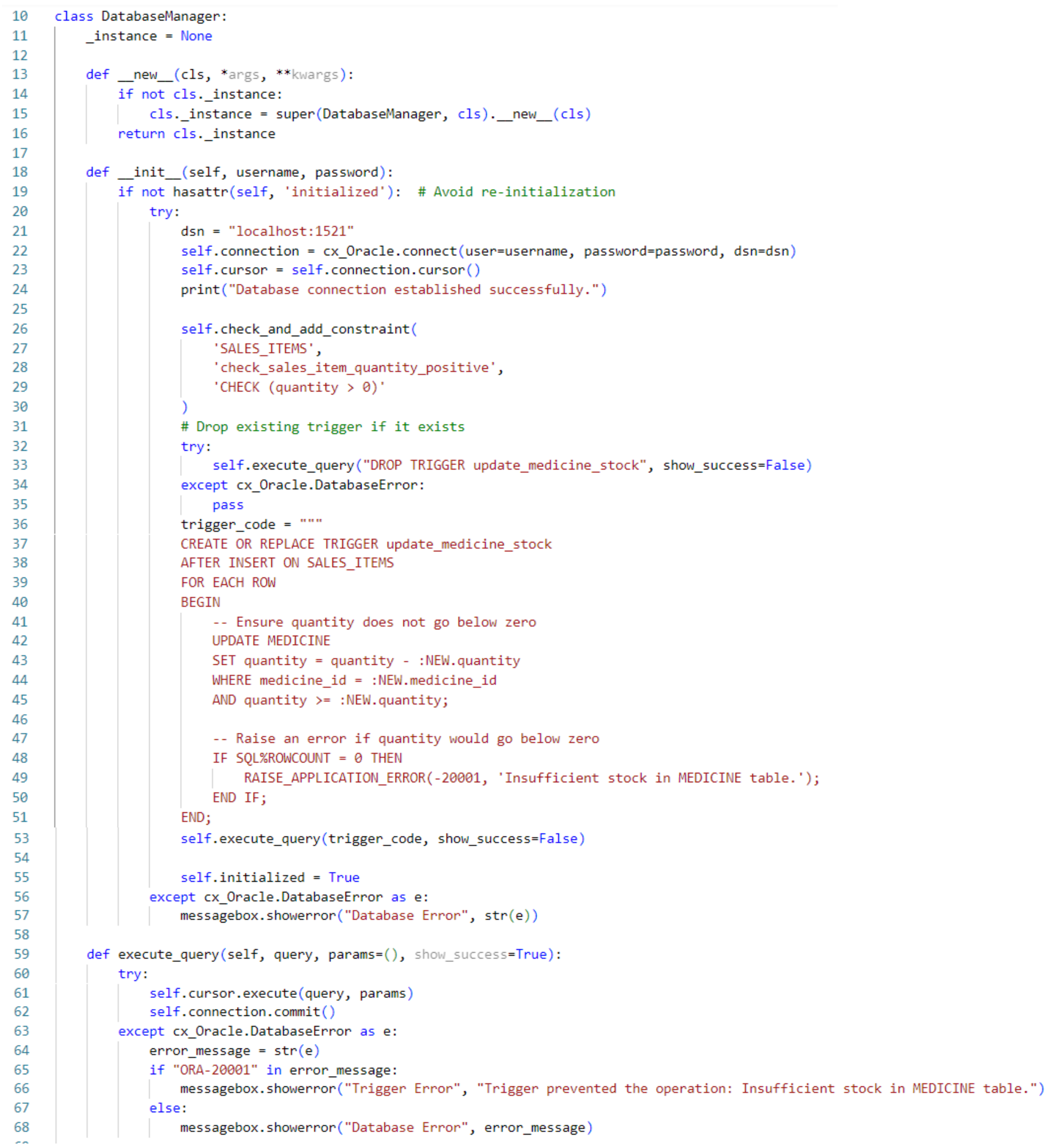
Meghana Kumar
3122235002070
Nikhil S
3122235002080

PROJECT – MEDICAL SALES DATABASE MANAGEMENT

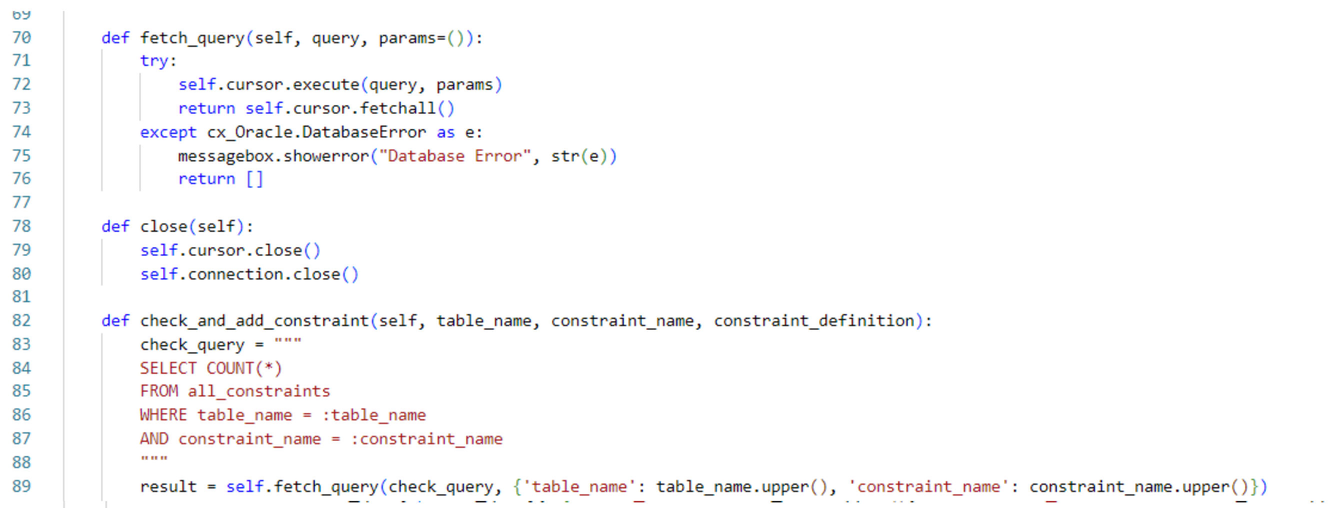
AIM:

SCHEMA DIAGRAM:

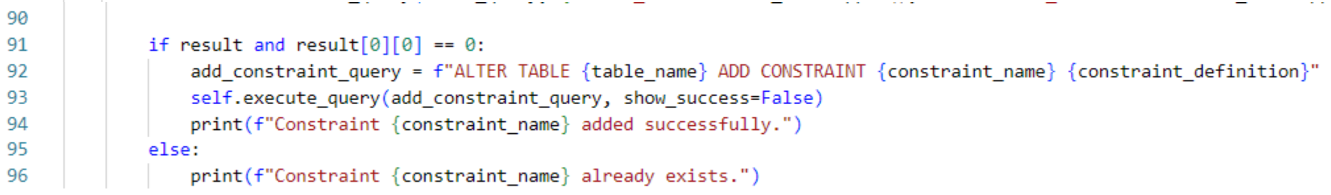
CODE SNIPPET:



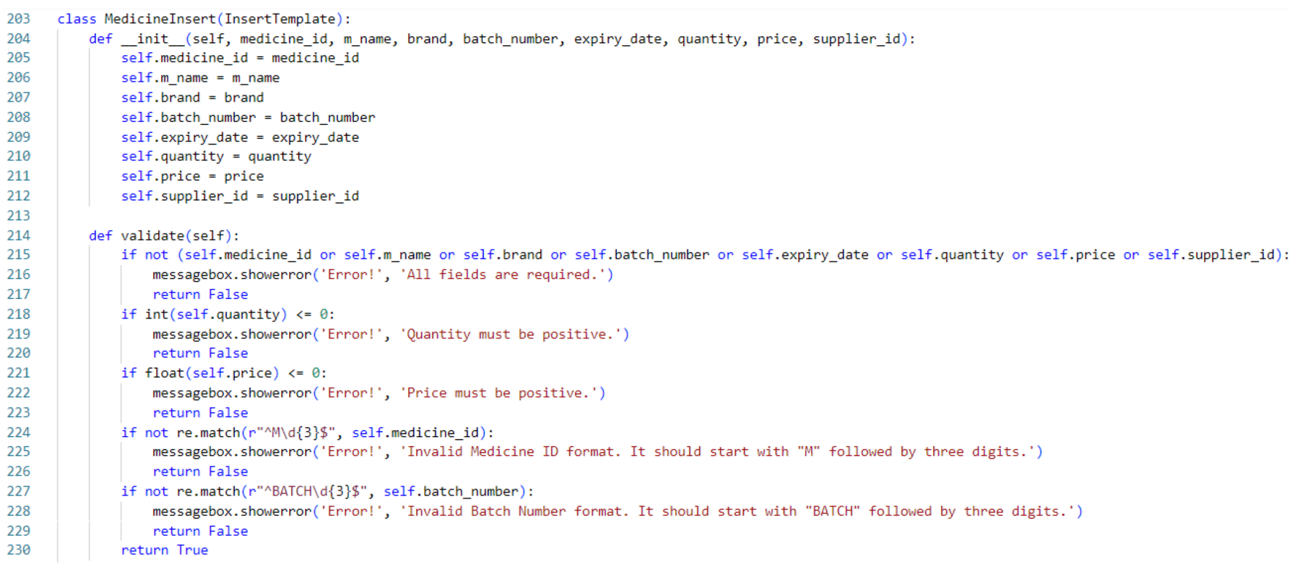
```
class DatabaseManager:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(DatabaseManager, cls).__new__(cls)
        return cls._instance

    def __init__(self, username, password):
        if not hasattr(self, 'initialized'):  # Avoid re-initialization
            try:
                dsn = "localhost:1521"
                self.connection = cx_Oracle.connect(user=username, password=password, dsn=dsn)
                self.cursor = self.connection.cursor()
                print("Database connection established successfully.")

                self.check_and_add_constraint(
                    'SALES_ITEMS',
                    'check_sales_item_quantity_positive',
                    'CHECK (quantity > 0)'
                )
                # Drop existing trigger if it exists
                try:
                    self.execute_query("DROP TRIGGER update_medicine_stock", show_success=False)
                except cx_Oracle.DatabaseError:
                    pass
                trigger_code = """
                CREATE OR REPLACE TRIGGER update_medicine_stock
                AFTER INSERT ON SALES_ITEMS
                FOR EACH ROW
                BEGIN
                    -- Ensure quantity does not go below zero
                    UPDATE MEDICINE
                    SET quantity = quantity - :NEW.quantity
                    WHERE medicine_id = :NEW.medicine_id
                    AND quantity >= :NEW.quantity;

                    -- Raise an error if quantity would go below zero
                    IF SQL%ROWCOUNT = 0 THEN
                        RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock in MEDICINE table.');
                    END IF;
                END;
                self.execute_query(trigger_code, show_success=False)

                self.initialized = True
            except cx_Oracle.DatabaseError as e:
                messagebox.showerror("Database Error", str(e))

    def execute_query(self, query, params=(), show_success=True):
        try:
            self.cursor.execute(query, params)
            self.connection.commit()
        except cx_Oracle.DatabaseError as e:
            error_message = str(e)
            if "ORA-20001" in error_message:
                messagebox.showerror("Trigger Error", "Trigger prevented the operation: Insufficient stock in MEDICINE table.")
            else:
                messagebox.showerror("Database Error", error_message)
```



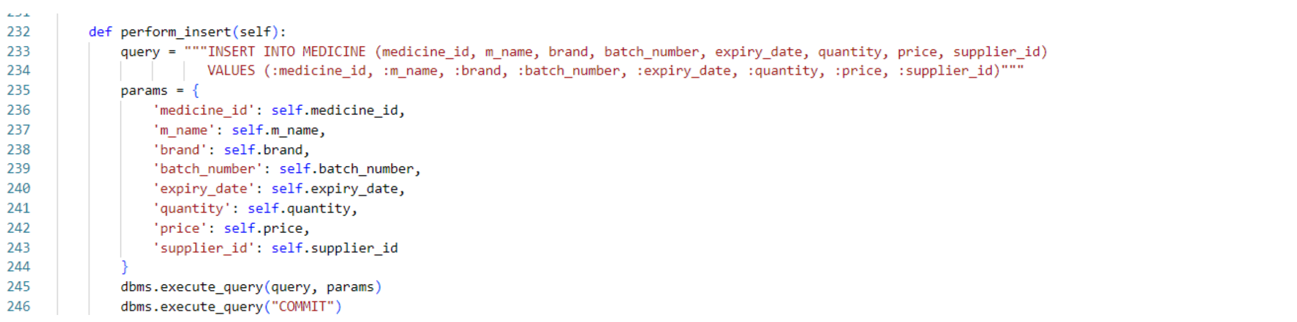
```python
    def fetch_query(self, query, params=()):
        try:
            self.cursor.execute(query, params)
            return self.cursor.fetchall()
        except cx_Oracle.DatabaseError as e:
            messagebox.showerror("Database Error", str(e))
            return []

    def close(self):
        self.cursor.close()
        self.connection.close()

    def check_and_add_constraint(self, table_name, constraint_name, constraint_definition):
        check_query = """
        SELECT COUNT(*)
        FROM all_constraints
        WHERE table_name = :table_name
        AND constraint_name = :constraint_name
        """
        result = self.fetch_query(check_query, {'table_name': table_name.upper(), 'constraint_name': constraint_name.upper()})
```



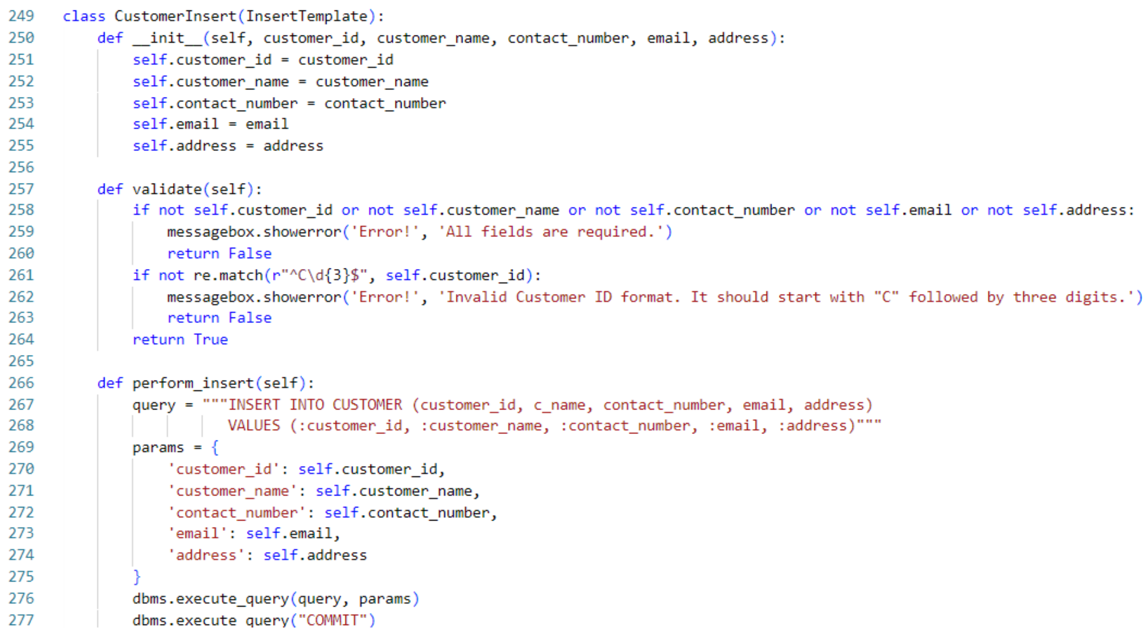
```python

        if result and result[0][0] == 0:
            add_constraint_query = f"ALTER TABLE {table_name} ADD CONSTRAINT {constraint_name} {constraint_definition}"
            self.execute_query(add_constraint_query, show_success=False)
            print(f"Constraint {constraint_name} added successfully.")
        else:
            print(f"Constraint {constraint_name} already exists.")
```



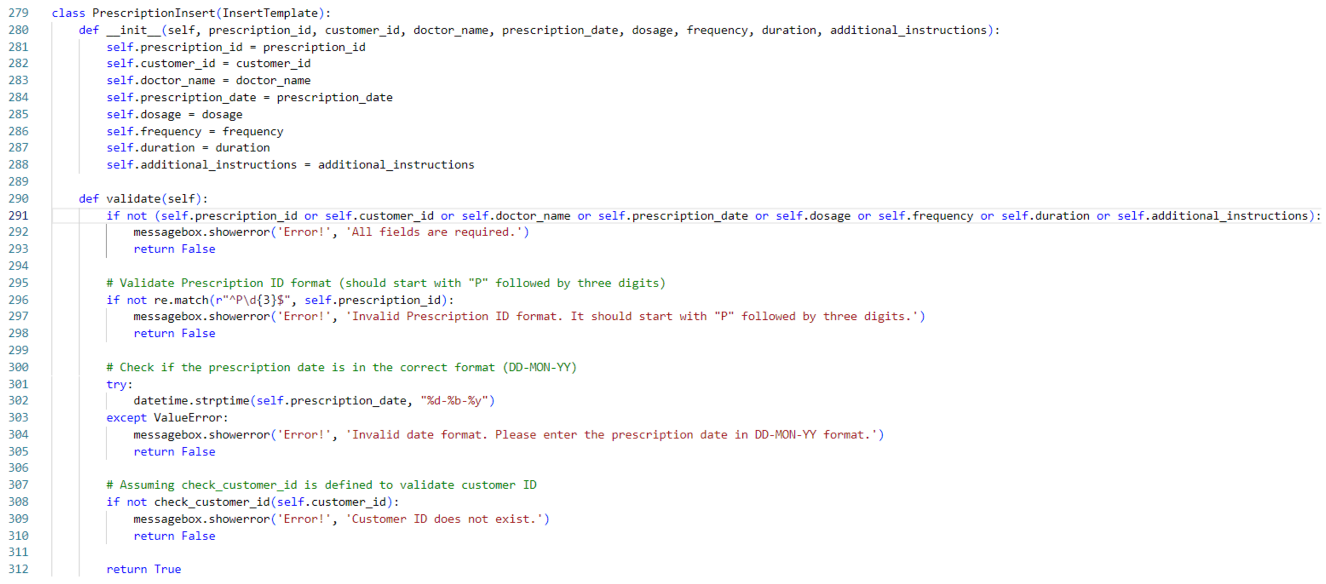
```python
class MedicineInsert(InsertTemplate):
    def __init__(self, medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id):
        self.medicine_id = medicine_id
        self.m_name = m_name
        self.brand = brand
        self.batch_number = batch_number
        self.expiry_date = expiry_date
        self.quantity = quantity
        self.price = price
        self.supplier_id = supplier_id

    def validate(self):
        if not (self.medicine_id or self.m_name or self.brand or self.batch_number or self.expiry_date or self.quantity or self.price or self.supplier_id):
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if int(self.quantity) <= 0:
            messagebox.showerror('Error!', 'Quantity must be positive.')
            return False
        if float(self.price) <= 0:
            messagebox.showerror('Error!', 'Price must be positive.')
            return False
        if not re.match(r"^M\d{3}$", self.medicine_id):
            messagebox.showerror('Error!', 'Invalid Medicine ID format. It should start with "M" followed by three digits.')
            return False
        if not re.match(r"^BATCH\d{3}$", self.batch_number):
            messagebox.showerror('Error!', 'Invalid Batch Number format. It should start with "BATCH" followed by three digits.')
            return False
        return True
```



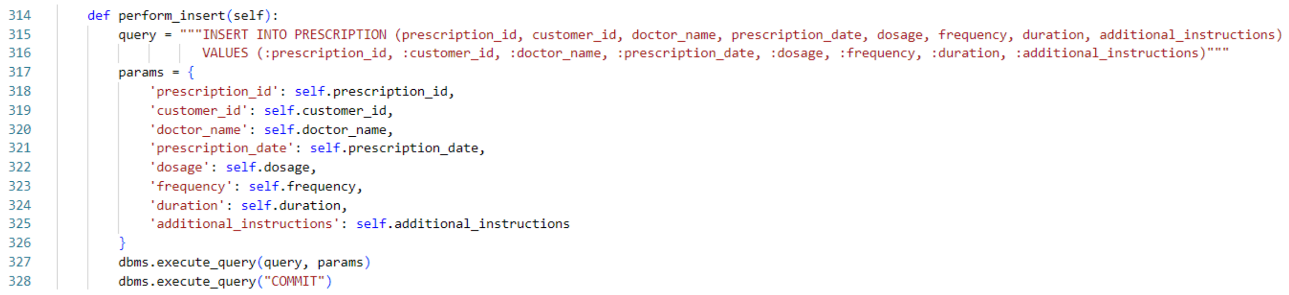
```python

    def perform_insert(self):
        query = """INSERT INTO MEDICINE (medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id)
                   VALUES (:medicine_id, :m_name, :brand, :batch_number, :expiry_date, :quantity, :price, :supplier_id)"""
        params = {
            'medicine_id': self.medicine_id,
            'm_name': self.m_name,
            'brand': self.brand,
            'batch_number': self.batch_number,
            'expiry_date': self.expiry_date,
            'quantity': self.quantity,
            'price': self.price,
            'supplier_id': self.supplier_id
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")
```



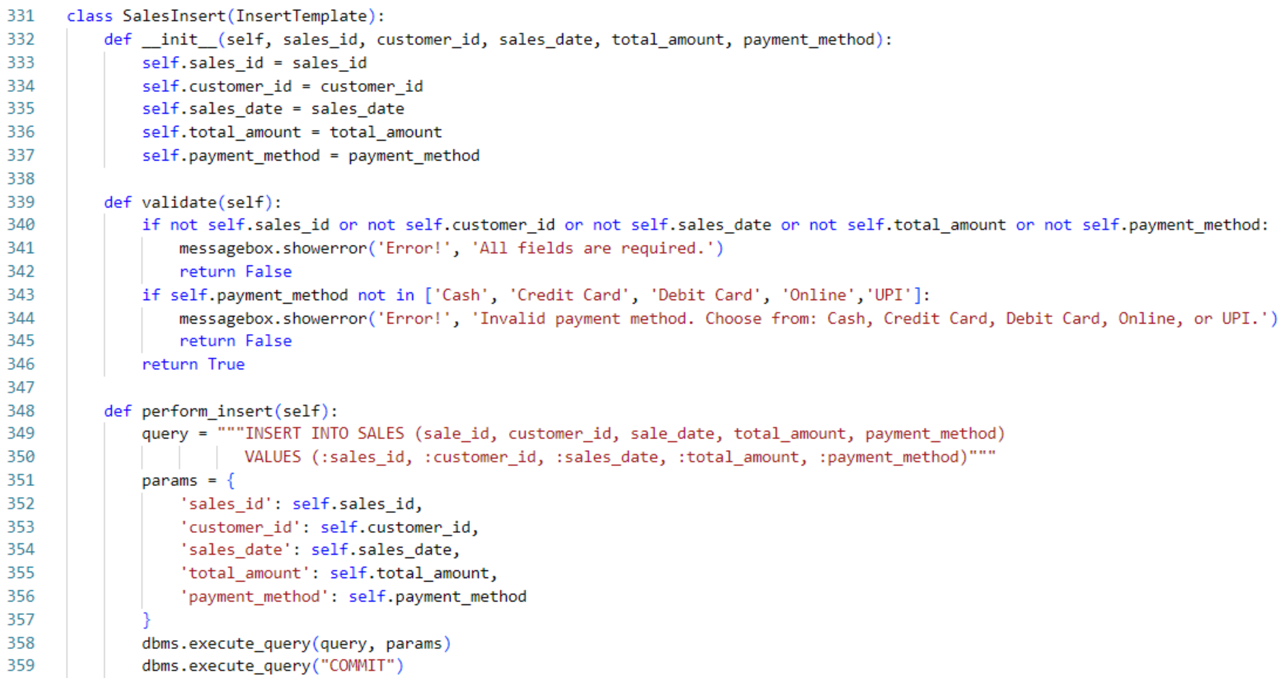
```python
class CustomerInsert(InsertTemplate):
    def __init__(self, customer_id, customer_name, contact_number, email, address):
        self.customer_id = customer_id
        self.customer_name = customer_name
        self.contact_number = contact_number
        self.email = email
        self.address = address

    def validate(self):
        if not self.customer_id or not self.customer_name or not self.contact_number or not self.email or not self.address:
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if not re.match(r"^C\d{3}$", self.customer_id):
            messagebox.showerror('Error!', 'Invalid Customer ID format. It should start with "C" followed by three digits.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO CUSTOMER (customer_id, c_name, contact_number, email, address)
                VALUES (:customer_id, :customer_name, :contact_number, :email, :address)"""
        params = {
            'customer_id': self.customer_id,
            'customer_name': self.customer_name,
            'contact_number': self.contact_number,
            'email': self.email,
            'address': self.address
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")
```



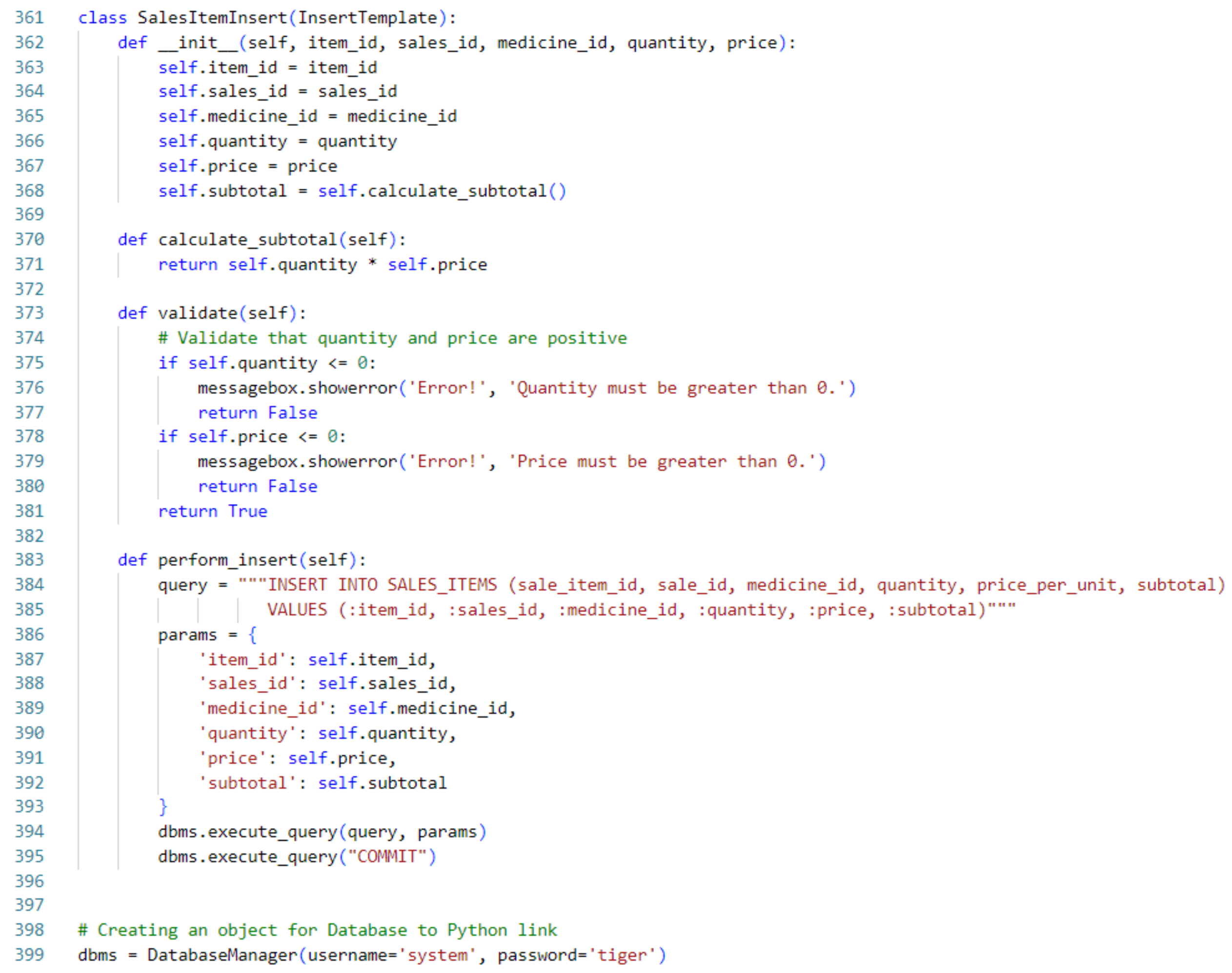
```python
class PrescriptionInsert(InsertTemplate):
    def __init__(self, prescription_id, customer_id, doctor_name, prescription_date, dosage, frequency, duration, additional_instructions):
        self.prescription_id = prescription_id
        self.customer_id = customer_id
        self.doctor_name = doctor_name
        self.prescription_date = prescription_date
        self.dosage = dosage
        self.frequency = frequency
        self.duration = duration
        self.additional_instructions = additional_instructions

    def validate(self):
        if not (self.prescription_id or self.customer_id or self.doctor_name or self.prescription_date or self.dosage or self.frequency or self.duration or self.additional_instructions):
            messagebox.showerror('Error!', 'All fields are required.')
            return False

        # Validate Prescription ID format (should start with "P" followed by three digits)
        if not re.match(r"^P\d{3}$", self.prescription_id):
            messagebox.showerror('Error!', 'Invalid Prescription ID format. It should start with "P" followed by three digits.')
            return False

        # Check if the prescription date is in the correct format (DD-MON-YY)
        try:
            datetime.strptime(self.prescription_date, "%d-%b-%y")
        except ValueError:
            messagebox.showerror('Error!', 'Invalid date format. Please enter the prescription date in DD-MON-YY format.')
            return False

        # Assuming check_customer_id is defined to validate customer ID
        if not check_customer_id(self.customer_id):
            messagebox.showerror('Error!', 'Customer ID does not exist.')
            return False

        return True
```



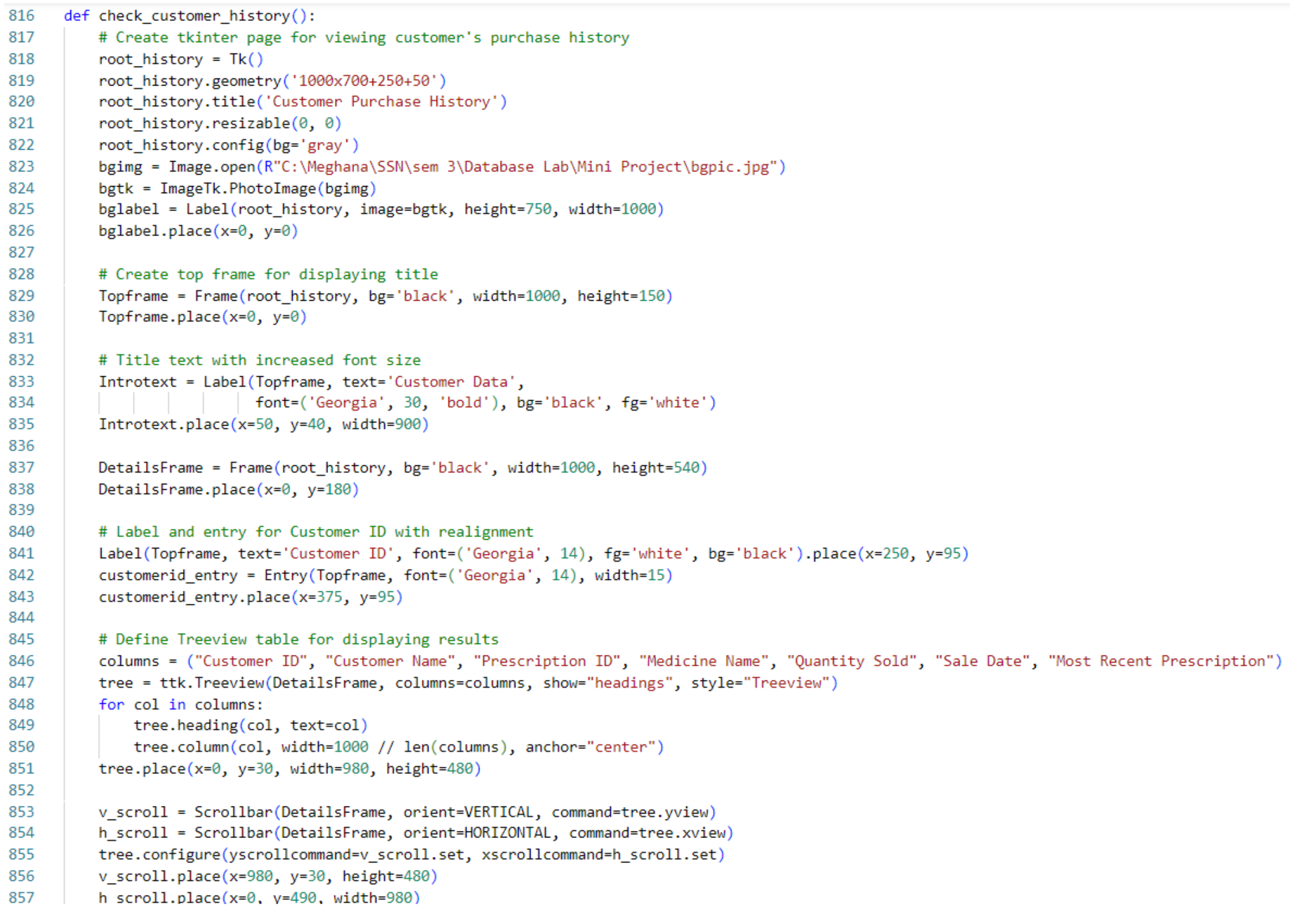
```python
    def perform_insert(self):
        query = """INSERT INTO PRESCRIPTION (prescription_id, customer_id, doctor_name, prescription_date, dosage, frequency, duration, additional_instructions)
                VALUES (:prescription_id, :customer_id, :doctor_name, :prescription_date, :dosage, :frequency, :duration, :additional_instructions)"""
        params = {
            'prescription_id': self.prescription_id,
            'customer_id': self.customer_id,
            'doctor_name': self.doctor_name,
            'prescription_date': self.prescription_date,
            'dosage': self.dosage,
            'frequency': self.frequency,
            'duration': self.duration,
            'additional_instructions': self.additional_instructions
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")
```



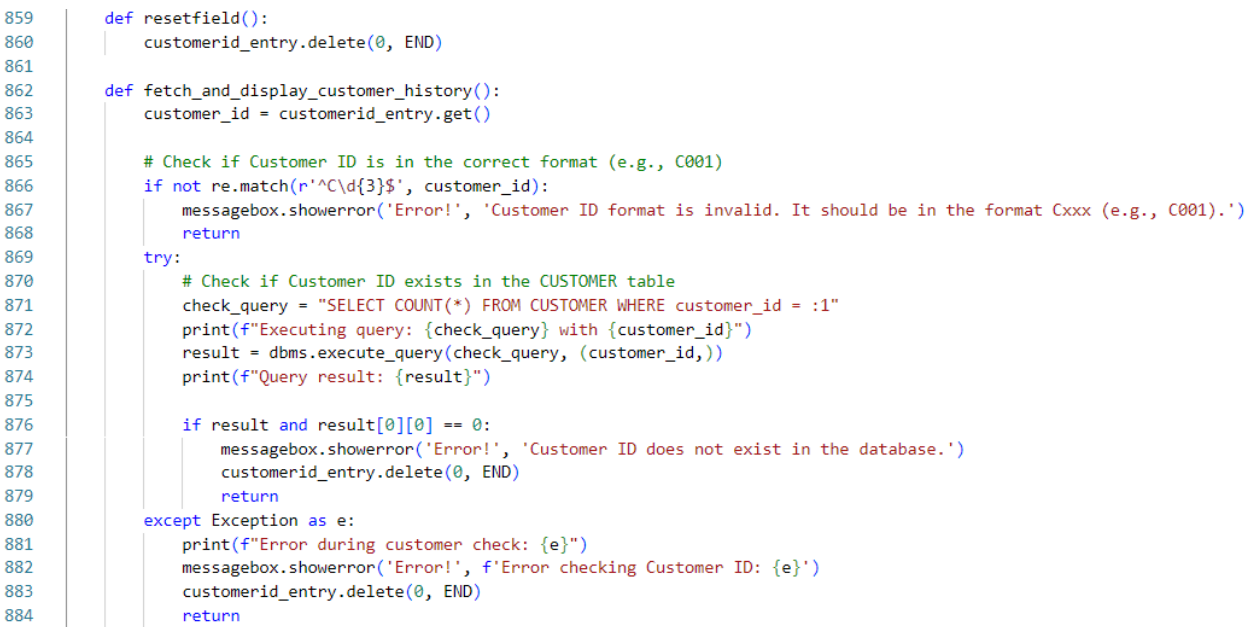
```python
class SalesInsert(InsertTemplate):
    def __init__(self, sales_id, customer_id, sales_date, total_amount, payment_method):
        self.sales_id = sales_id
        self.customer_id = customer_id
        self.sales_date = sales_date
        self.total_amount = total_amount
        self.payment_method = payment_method

    def validate(self):
        if not self.sales_id or not self.customer_id or not self.sales_date or not self.total_amount or not self.payment_method:
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if self.payment_method not in ['Cash', 'Credit Card', 'Debit Card', 'Online','UPI']:
            messagebox.showerror('Error!', 'Invalid payment method. Choose from: Cash, Credit Card, Debit Card, Online, or UPI.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO SALES (sale_id, customer_id, sale_date, total_amount, payment_method)
                   VALUES (:sales_id, :customer_id, :sales_date, :total_amount, :payment_method)"""
        params = {
            'sales_id': self.sales_id,
            'customer_id': self.customer_id,
            'sales_date': self.sales_date,
            'total_amount': self.total_amount,
            'payment_method': self.payment_method
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")
```



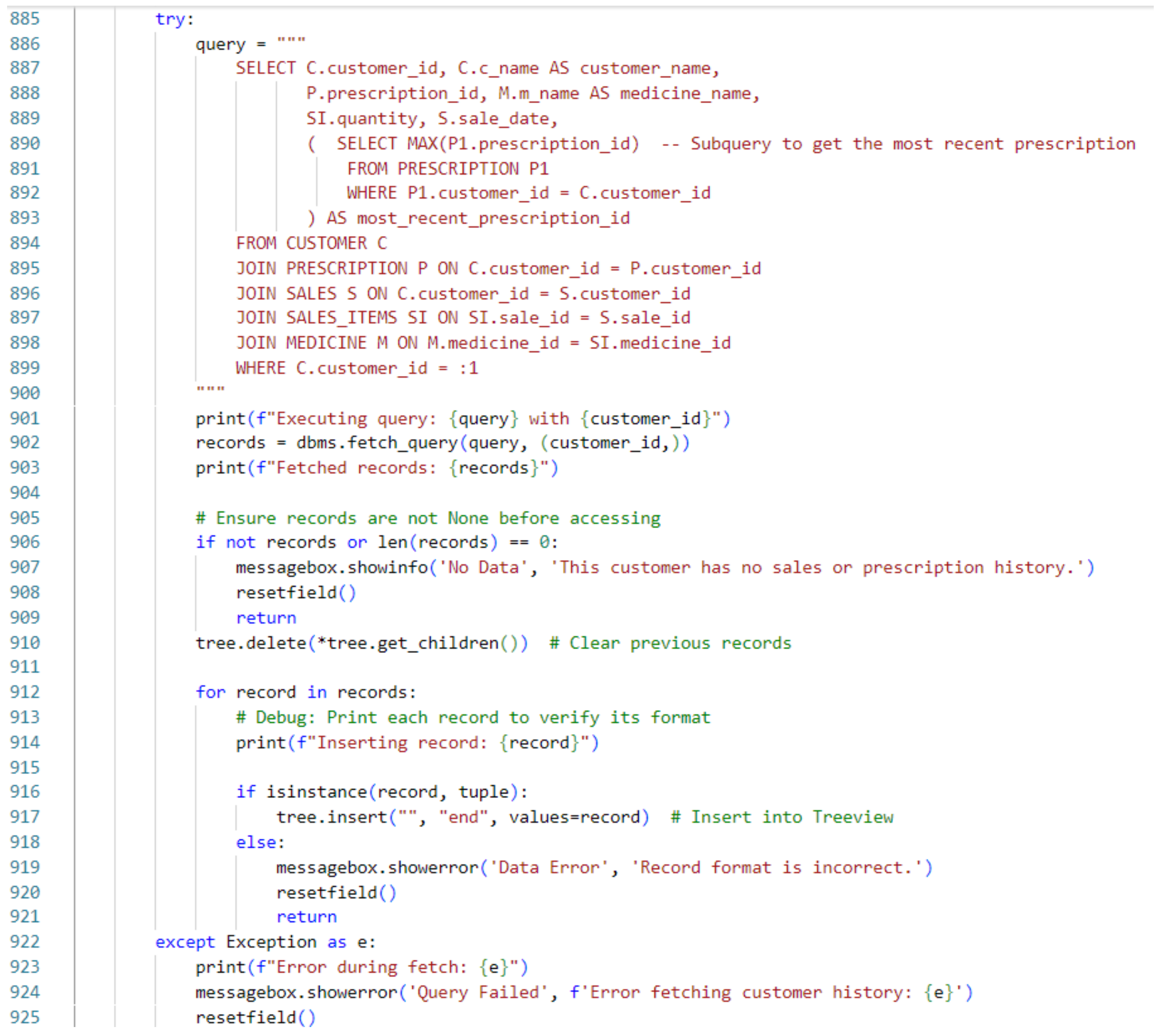
```python
class SalesItemInsert(InsertTemplate):
    def __init__(self, item_id, sales_id, medicine_id, quantity, price):
        self.item_id = item_id
        self.sales_id = sales_id
        self.medicine_id = medicine_id
        self.quantity = quantity
        self.price = price
        self.subtotal = self.calculate_subtotal()

    def calculate_subtotal(self):
        return self.quantity * self.price

    def validate(self):
        # Validate that quantity and price are positive
        if self.quantity <= 0:
            messagebox.showerror('Error!', 'Quantity must be greater than 0.')
            return False
        if self.price <= 0:
            messagebox.showerror('Error!', 'Price must be greater than 0.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO SALES_ITEMS (sale_item_id, sale_id, medicine_id, quantity, price_per_unit, subtotal)
                   VALUES (:item_id, :sales_id, :medicine_id, :quantity, :price, :subtotal)"""
        params = {
            'item_id': self.item_id,
            'sales_id': self.sales_id,
            'medicine_id': self.medicine_id,
            'quantity': self.quantity,
            'price': self.price,
            'subtotal': self.subtotal
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")


# Creating an object for Database to Python link
dbms = DatabaseManager(username='system', password='tiger')
```



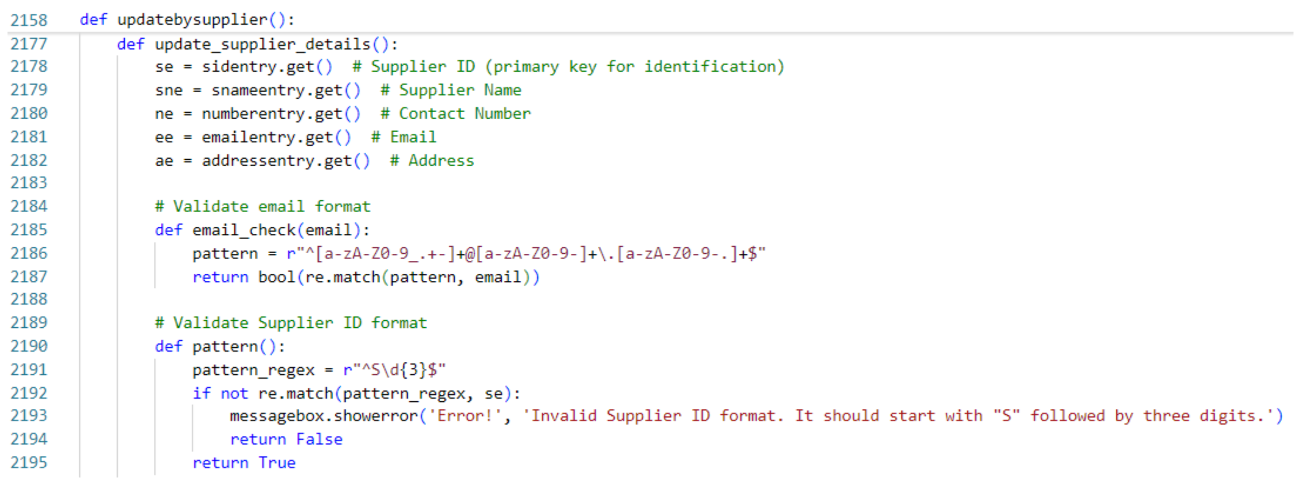
```python
def check_customer_history():
    # Create tkinter page for viewing customer's purchase history
    root_history = Tk()
    root_history.geometry('1000x700+250+50')
    root_history.title('Customer Purchase History')
    root_history.resizable(0, 0)
    root_history.config(bg='gray')
    bgimg = Image.open(R"C:\Meghana\SSN\sem 3\Database Lab\Mini Project\bgpic.jpg")
    bgtk = ImageTk.PhotoImage(bgimg)
    bglabel = Label(root_history, image=bgtk, height=750, width=1000)
    bglabel.place(x=0, y=0)

    # Create top frame for displaying title
    Topframe = Frame(root_history, bg='black', width=1000, height=150)
    Topframe.place(x=0, y=0)

    # Title text with increased font size
    Introtext = Label(Topframe, text='Customer Data',
                      font=('Georgia', 30, 'bold'), bg='black', fg='white')
    Introtext.place(x=50, y=40, width=900)

    DetailsFrame = Frame(root_history, bg='black', width=1000, height=540)
    DetailsFrame.place(x=0, y=180)

    # Label and entry for Customer ID with realignment
    Label(Topframe, text='Customer ID', font=('Georgia', 14), fg='white', bg='black').place(x=250, y=95)
    customerid_entry = Entry(Topframe, font=('Georgia', 14), width=15)
    customerid_entry.place(x=375, y=95)

    # Define Treeview table for displaying results
    columns = ("Customer ID", "Customer Name", "Prescription ID", "Medicine Name", "Quantity Sold", "Sale Date", "Most Recent Prescription")
    tree = ttk.Treeview(DetailsFrame, columns=columns, show="headings", style="Treeview")
    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, width=1000 // len(columns), anchor="center")
    tree.place(x=0, y=30, width=980, height=480)

    v_scroll = Scrollbar(DetailsFrame, orient=VERTICAL, command=tree.yview)
    h_scroll = Scrollbar(DetailsFrame, orient=HORIZONTAL, command=tree.xview)
    tree.configure(yscrollcommand=v_scroll.set, xscrollcommand=h_scroll.set)
    v_scroll.place(x=980, y=30, height=480)
    h_scroll.place(x=0, y=490, width=980)
```



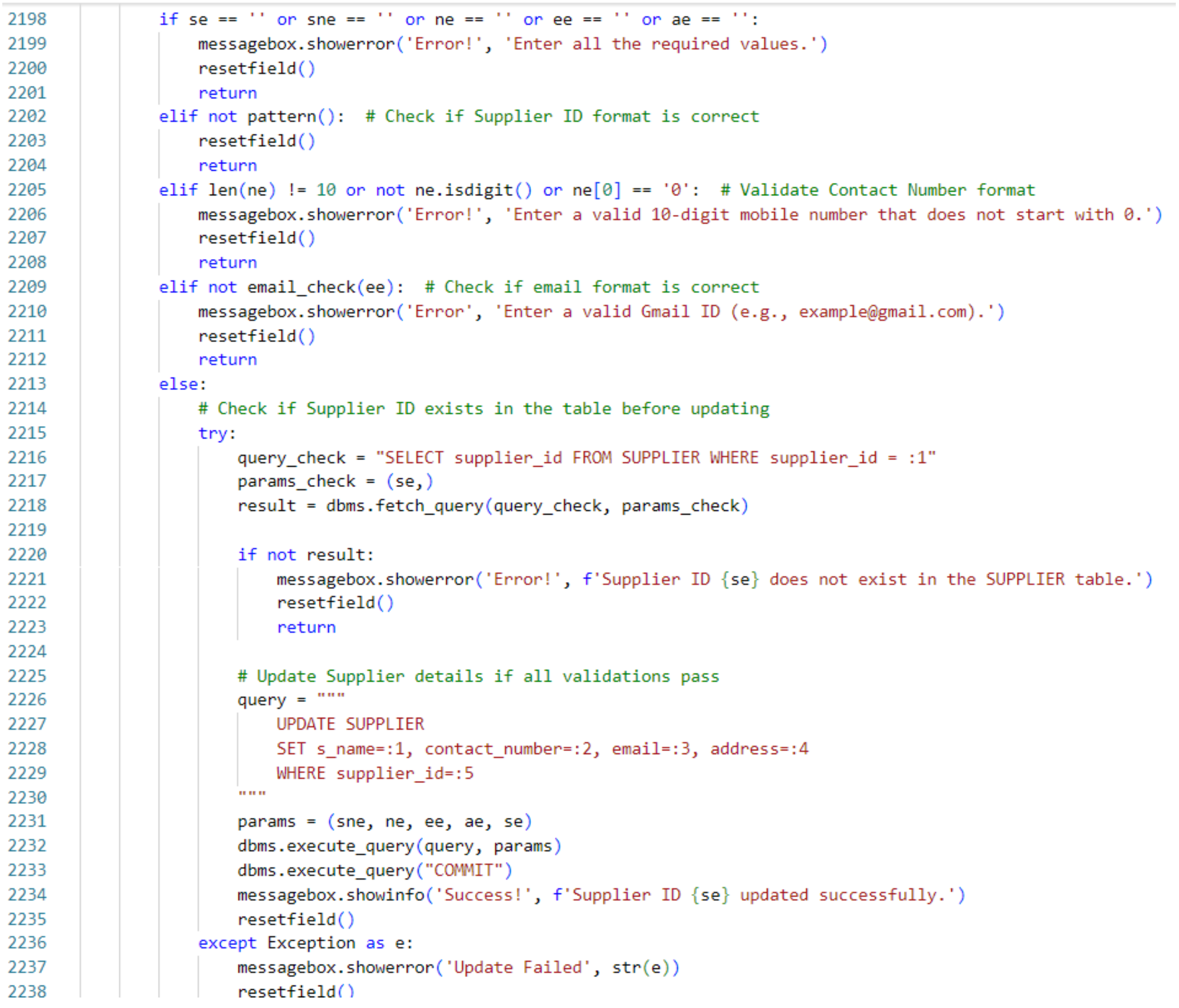
```python
    def resetfield():
        customerid_entry.delete(0, END)

    def fetch_and_display_customer_history():
        customer_id = customerid_entry.get()

        # Check if Customer ID is in the correct format (e.g., C001)
        if not re.match(r'^C\d{3}$', customer_id):
            messagebox.showerror('Error!', 'Customer ID format is invalid. It should be in the format Cxxx (e.g., C001).')
            return
        try:
            # Check if Customer ID exists in the CUSTOMER table
            check_query = "SELECT COUNT(*) FROM CUSTOMER WHERE customer_id = :1"
            print(f"Executing query: {check_query} with {customer_id}")
            result = dbms.execute_query(check_query, (customer_id,))
            print(f"Query result: {result}")

            if result and result[0][0] == 0:
                messagebox.showerror('Error!', 'Customer ID does not exist in the database.')
                customerid_entry.delete(0, END)
                return
        except Exception as e:
            print(f"Error during customer check: {e}")
            messagebox.showerror('Error!', f'Error checking Customer ID: {e}')
            customerid_entry.delete(0, END)
            return
```



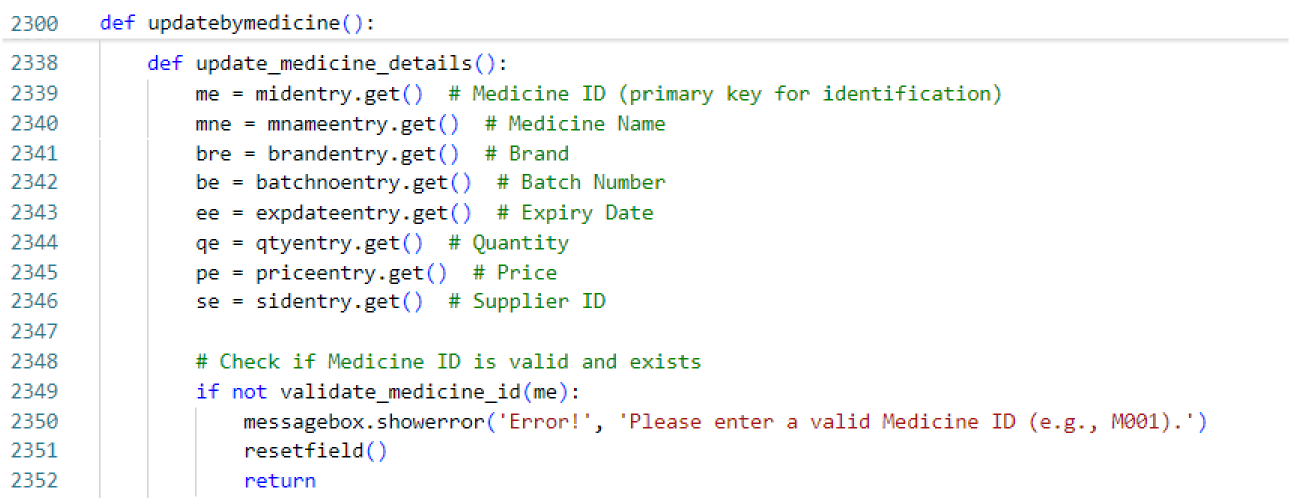
```
        try:
            query = """
                SELECT C.customer_id, C.c_name AS customer_name,
                       P.prescription_id, M.m_name AS medicine_name,
                       SI.quantity, S.sale_date,
                       (  SELECT MAX(P1.prescription_id)  -- Subquery to get the most recent prescription
                           FROM PRESCRIPTION P1
                           WHERE P1.customer_id = C.customer_id
                       ) AS most_recent_prescription_id
                FROM CUSTOMER C
                JOIN PRESCRIPTION P ON C.customer_id = P.customer_id
                JOIN SALES S ON C.customer_id = S.customer_id
                JOIN SALES_ITEMS SI ON SI.sale_id = S.sale_id
                JOIN MEDICINE M ON M.medicine_id = SI.medicine_id
                WHERE C.customer_id = :1
            """
            print(f"Executing query: {query} with {customer_id}")
            records = dbms.fetch_query(query, (customer_id,))
            print(f"Fetched records: {records}")

            # Ensure records are not None before accessing
            if not records or len(records) == 0:
                messagebox.showinfo('No Data', 'This customer has no sales or prescription history.')
                resetfield()
                return
            tree.delete(*tree.get_children())  # Clear previous records

            for record in records:
                # Debug: Print each record to verify its format
                print(f"Inserting record: {record}")

                if isinstance(record, tuple):
                    tree.insert("", "end", values=record)  # Insert into Treeview
                else:
                    messagebox.showerror('Data Error', 'Record format is incorrect.')
                    resetfield()
                    return
        except Exception as e:
            print(f"Error during fetch: {e}")
            messagebox.showerror('Query Failed', f'Error fetching customer history: {e}')
            resetfield()
```



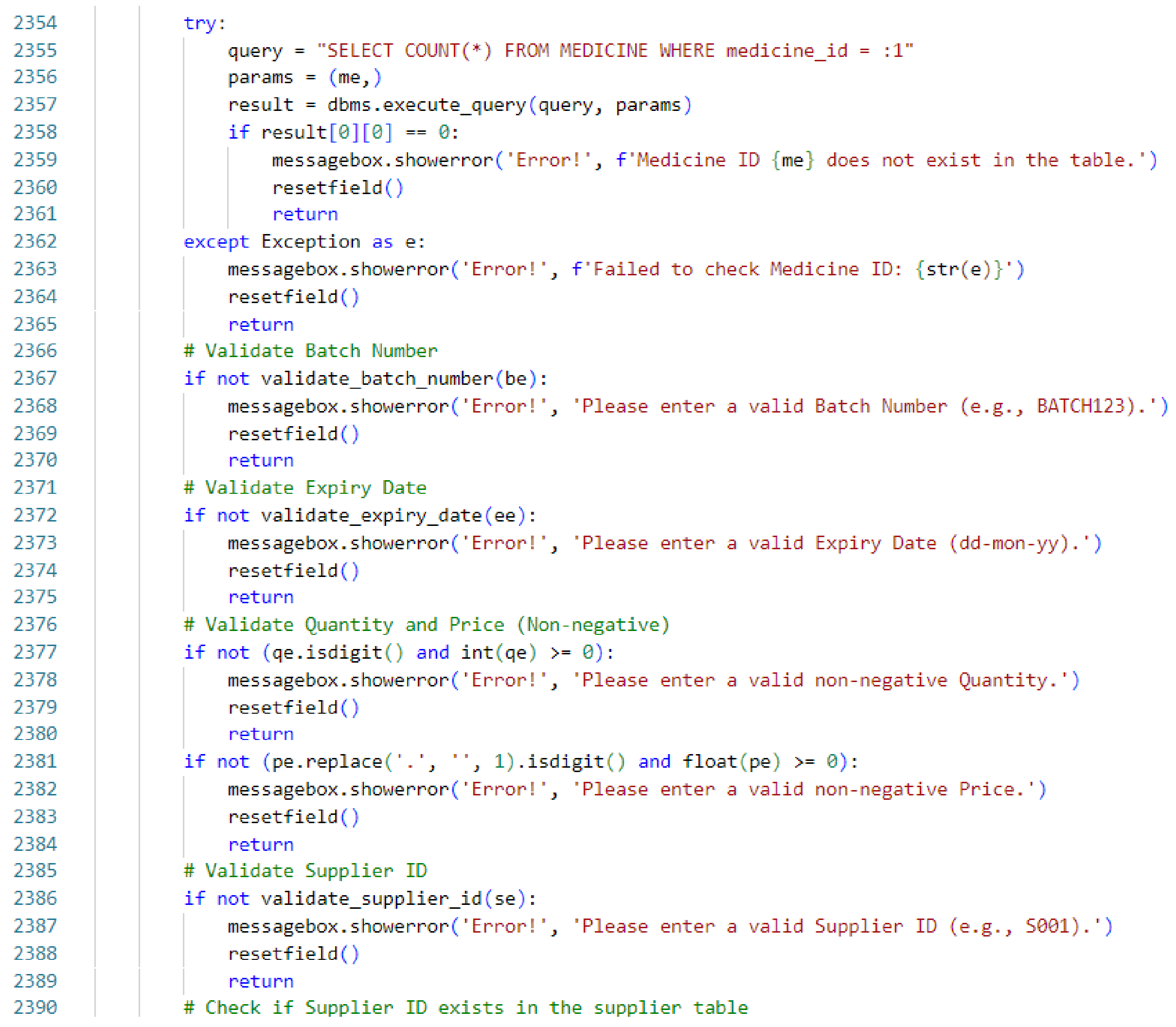
```
def updatebysupplier():
    def update_supplier_details():
        se = sidentry.get()  # Supplier ID (primary key for identification)
        sne = snameentry.get()  # Supplier Name
        ne = numberentry.get()  # Contact Number
        ee = emailentry.get()  # Email
        ae = addressentry.get()  # Address

        # Validate email format
        def email_check(email):
            pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
            return bool(re.match(pattern, email))

        # Validate Supplier ID format
        def pattern():
            pattern_regex = r"^S\d{3}$"
            if not re.match(pattern_regex, se):
                messagebox.showerror('Error!', 'Invalid Supplier ID format. It should start with "S" followed by three digits.')
                return False
            return True
```



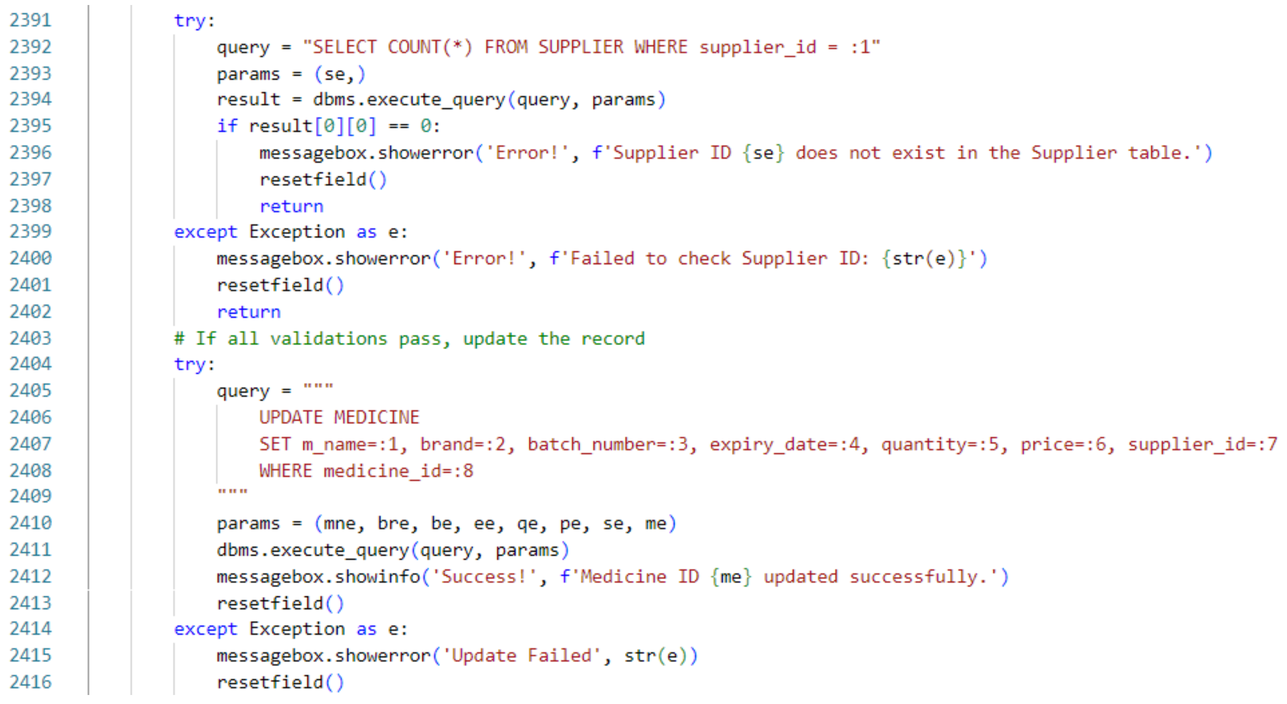
```python
        if se == '' or sne == '' or ne == '' or ee == '' or ae == '':
            messagebox.showerror('Error!', 'Enter all the required values.')
            resetfield()
            return
        elif not pattern():  # Check if Supplier ID format is correct
            resetfield()
            return
        elif len(ne) != 10 or not ne.isdigit() or ne[0] == '0':  # Validate Contact Number format
            messagebox.showerror('Error!', 'Enter a valid 10-digit mobile number that does not start with 0.')
            resetfield()
            return
        elif not email_check(ee):  # Check if email format is correct
            messagebox.showerror('Error', 'Enter a valid Gmail ID (e.g., example@gmail.com).')
            resetfield()
            return
        else:
            # Check if Supplier ID exists in the table before updating
            try:
                query_check = "SELECT supplier_id FROM SUPPLIER WHERE supplier_id = :1"
                params_check = (se,)
                result = dbms.fetch_query(query_check, params_check)

                if not result:
                    messagebox.showerror('Error!', f'Supplier ID {se} does not exist in the SUPPLIER table.')
                    resetfield()
                    return

                # Update Supplier details if all validations pass
                query = """
                    UPDATE SUPPLIER
                    SET s_name=:1, contact_number=:2, email=:3, address=:4
                    WHERE supplier_id=:5
                """
                params = (sne, ne, ee, ae, se)
                dbms.execute_query(query, params)
                dbms.execute_query("COMMIT")
                messagebox.showinfo('Success!', f'Supplier ID {se} updated successfully.')
                resetfield()
            except Exception as e:
                messagebox.showerror('Update Failed', str(e))
                resetfield()
```



```python
def updatebymedicine():
    def update_medicine_details():
        me = midentry.get()  # Medicine ID (primary key for identification)
        mne = mnameentry.get()  # Medicine Name
        bre = brandentry.get()  # Brand
        be = batchnoentry.get()  # Batch Number
        ee = expdateentry.get()  # Expiry Date
        qe = qtyentry.get()  # Quantity
        pe = priceentry.get()  # Price
        se = sidentry.get()  # Supplier ID

        # Check if Medicine ID is valid and exists
        if not validate_medicine_id(me):
            messagebox.showerror('Error!', 'Please enter a valid Medicine ID (e.g., M001).')
            resetfield()
            return
```



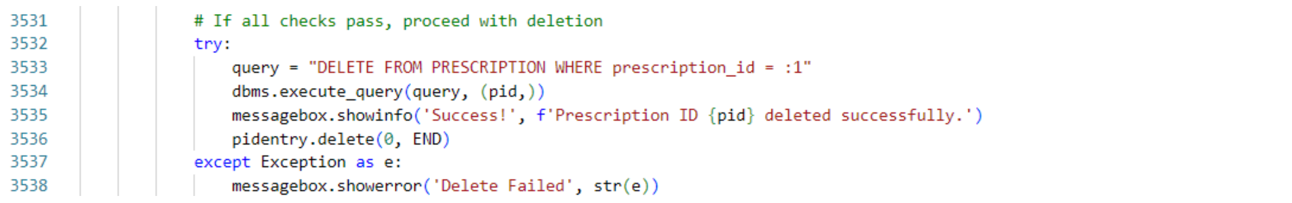
```python
        try:
            query = "SELECT COUNT(*) FROM MEDICINE WHERE medicine_id = :1"
            params = (me,)
            result = dbms.execute_query(query, params)
            if result[0][0] == 0:
                messagebox.showerror('Error!', f'Medicine ID {me} does not exist in the table.')
                resetfield()
                return
        except Exception as e:
            messagebox.showerror('Error!', f'Failed to check Medicine ID: {str(e)}')
            resetfield()
            return
        # Validate Batch Number
        if not validate_batch_number(be):
            messagebox.showerror('Error!', 'Please enter a valid Batch Number (e.g., BATCH123).')
            resetfield()
            return
        # Validate Expiry Date
        if not validate_expiry_date(ee):
            messagebox.showerror('Error!', 'Please enter a valid Expiry Date (dd-mon-yy).')
            resetfield()
            return
        # Validate Quantity and Price (Non-negative)
        if not (qe.isdigit() and int(qe) >= 0):
            messagebox.showerror('Error!', 'Please enter a valid non-negative Quantity.')
            resetfield()
            return
        if not (pe.replace('.', '', 1).isdigit() and float(pe) >= 0):
            messagebox.showerror('Error!', 'Please enter a valid non-negative Price.')
            resetfield()
            return
        # Validate Supplier ID
        if not validate_supplier_id(se):
            messagebox.showerror('Error!', 'Please enter a valid Supplier ID (e.g., S001).')
            resetfield()
            return
        # Check if Supplier ID exists in the supplier table
```



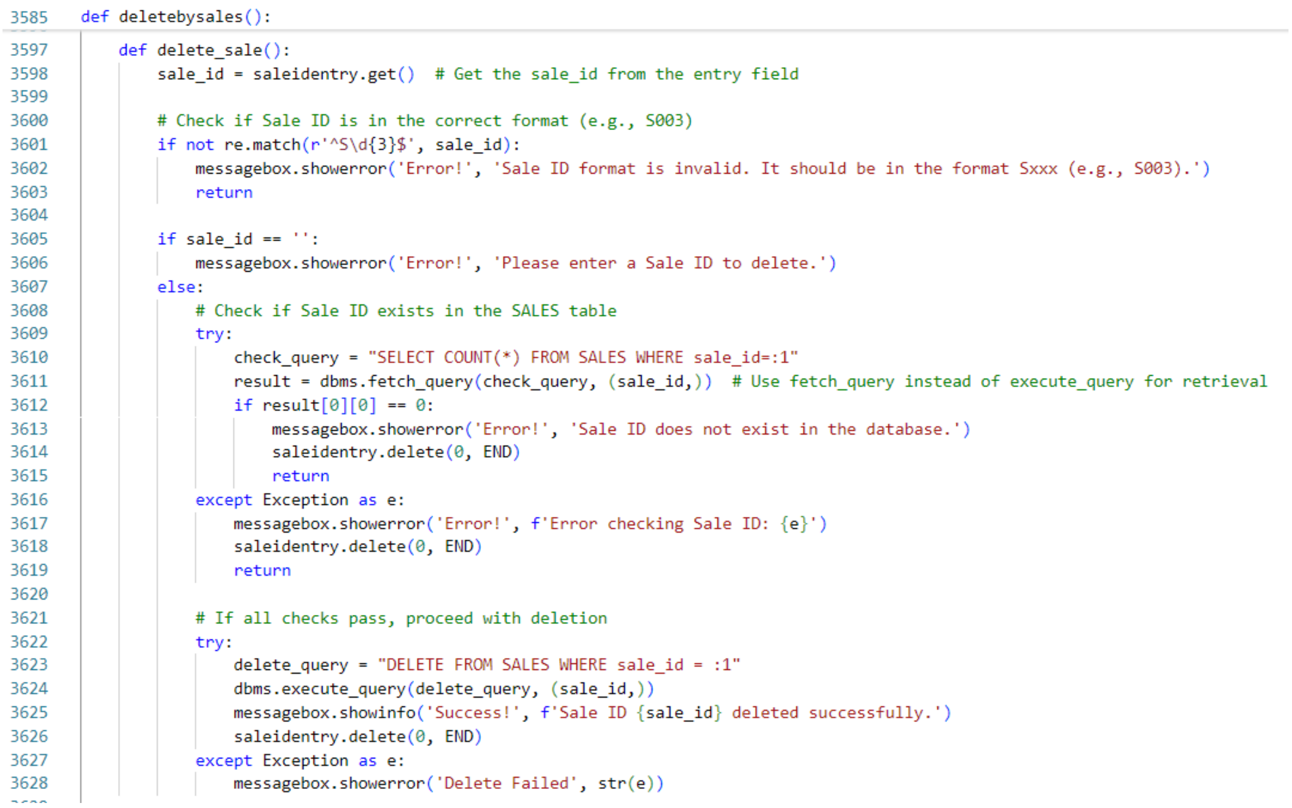
```python
        try:
            query = "SELECT COUNT(*) FROM SUPPLIER WHERE supplier_id = :1"
            params = (se,)
            result = dbms.execute_query(query, params)
            if result[0][0] == 0:
                messagebox.showerror('Error!', f'Supplier ID {se} does not exist in the Supplier table.')
                resetfield()
                return
        except Exception as e:
            messagebox.showerror('Error!', f'Failed to check Supplier ID: {str(e)}')
            resetfield()
            return
        # If all validations pass, update the record
        try:
            query = """
                UPDATE MEDICINE
                SET m_name=:1, brand=:2, batch_number=:3, expiry_date=:4, quantity=:5, price=:6, supplier_id=:7
                WHERE medicine_id=:8
            """
            params = (mne, bre, be, ee, qe, pe, se, me)
            dbms.execute_query(query, params)
            messagebox.showinfo('Success!', f'Medicine ID {me} updated successfully.')
            resetfield()
        except Exception as e:
            messagebox.showerror('Update Failed', str(e))
            resetfield()
```

```python
def deletebycustomer():
    def delete_customer():
        cid = cidentry.get()  # Get the customer_id from the entry field

        # Check if Customer ID is in the correct format (e.g., C002)
        if not re.match(r'^C\d{3}$', cid):
            messagebox.showerror('Error!', 'Customer ID format is invalid. It should be in the format Cxxx (e.g., C002).')
            return

        if cid == '':
            messagebox.showerror('Error!', 'Please enter a Customer ID to delete.')
            return
        else:
            # Check if Customer ID exists in the CUSTOMER table
            try:
                check_query = "SELECT COUNT(*) FROM CUSTOMER WHERE customer_id=:1"
                result = dbms.fetch_query(check_query, (cid,))

                # Check if result is valid and contains data
                if result and result[0][0] > 0:
                    customer_count = result[0][0]
                else:
                    messagebox.showerror('Error!', 'Customer ID does not exist in the database.')
                    cidentry.delete(0, END)
                    return
            except Exception as e:
                messagebox.showerror('Error!', f'Error checking Customer ID: {e}')
                cidentry.delete(0, END)
                return

            # If all checks pass, proceed with deletion
            try:
                query = "DELETE FROM CUSTOMER WHERE customer_id = :1"
                dbms.execute_query(query, (cid,))
                messagebox.showinfo('Success!', f'Customer ID {cid} deleted successfully.')
                cidentry.delete(0, END)
            except Exception as e:
                messagebox.showerror('Delete Failed', str(e))
```

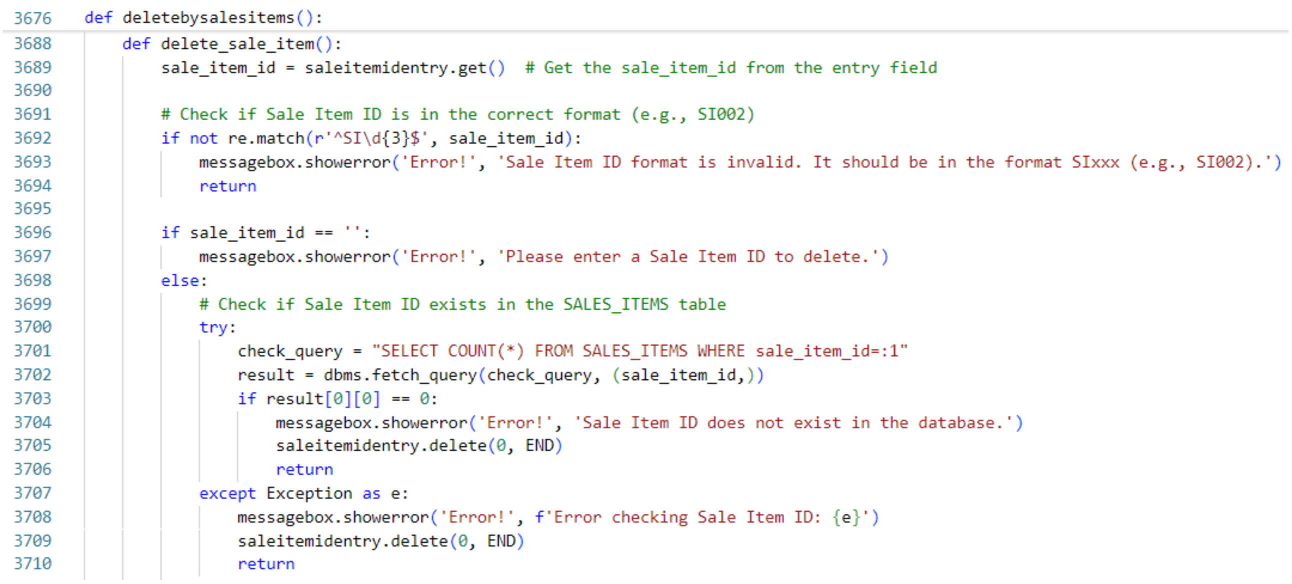
```python
def deletebyprescription():
    def delete_prescription():
        pid = pidentry.get()  # Get the prescription_id from the entry field

        # Check if Prescription ID is in the correct format (e.g., P001)
        if not re.match(r'^P\d{3}$', pid):
            messagebox.showerror('Error!', 'Prescription ID format is invalid. It should be in the format Pxxx (e.g., P001).')
            return

        if pid == '':
            messagebox.showerror('Error!', 'Please enter a Prescription ID to delete.')
            return
        else:
            # Check if Prescription ID exists in the PRESCRIPTION table
            try:
                check_query = "SELECT COUNT(*) FROM PRESCRIPTION WHERE prescription_id=:1"
                result = dbms.fetch_query(check_query, (pid,))

                # Check if result is valid and contains data
                if result and result[0][0] > 0:
                    prescription_count = result[0][0]
                else:
                    messagebox.showerror('Error!', 'Prescription ID does not exist in the database.')
                    pidentry.delete(0, END)
                    return
            except Exception as e:
                messagebox.showerror('Error!', f'Error checking Prescription ID: {e}')
                pidentry.delete(0, END)
                return
```



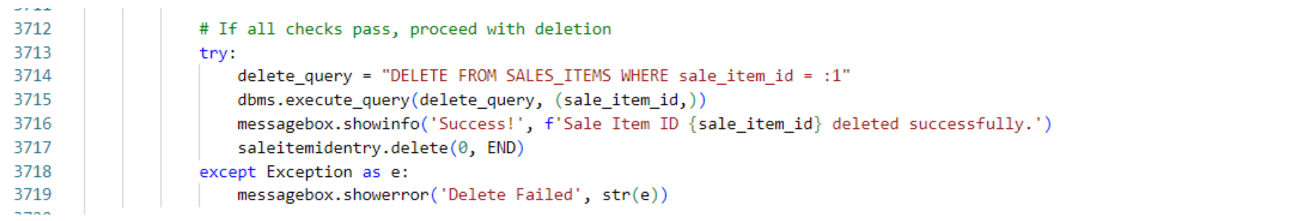
```python
            # If all checks pass, proceed with deletion
            try:
                query = "DELETE FROM PRESCRIPTION WHERE prescription_id = :1"
                dbms.execute_query(query, (pid,))
                messagebox.showinfo('Success!', f'Prescription ID {pid} deleted successfully.')
                pidentry.delete(0, END)
            except Exception as e:
                messagebox.showerror('Delete Failed', str(e))
```



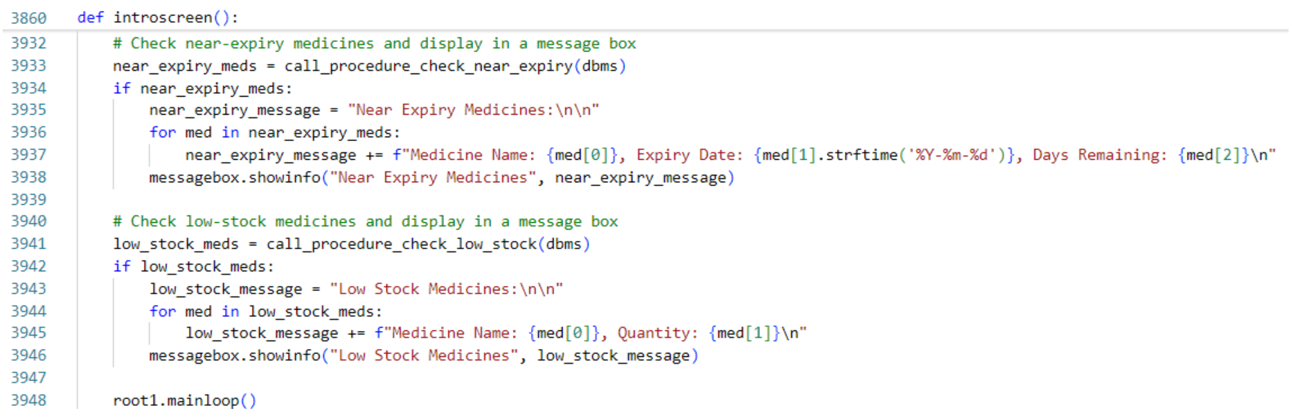
```python
def deletebysales():
    def delete_sale():
        sale_id = saleidentry.get()  # Get the sale_id from the entry field

        # Check if Sale ID is in the correct format (e.g., S003)
        if not re.match(r'^S\d{3}$', sale_id):
            messagebox.showerror('Error!', 'Sale ID format is invalid. It should be in the format Sxxx (e.g., S003).')
            return

        if sale_id == '':
            messagebox.showerror('Error!', 'Please enter a Sale ID to delete.')
        else:
            # Check if Sale ID exists in the SALES table
            try:
                check_query = "SELECT COUNT(*) FROM SALES WHERE sale_id=:1"
                result = dbms.fetch_query(check_query, (sale_id,))  # Use fetch_query instead of execute_query for retrieval
                if result[0][0] == 0:
                    messagebox.showerror('Error!', 'Sale ID does not exist in the database.')
                    saleidentry.delete(0, END)
                    return
            except Exception as e:
                messagebox.showerror('Error!', f'Error checking Sale ID: {e}')
                saleidentry.delete(0, END)
                return

            # If all checks pass, proceed with deletion
            try:
                delete_query = "DELETE FROM SALES WHERE sale_id = :1"
                dbms.execute_query(delete_query, (sale_id,))
                messagebox.showinfo('Success!', f'Sale ID {sale_id} deleted successfully.')
                saleidentry.delete(0, END)
            except Exception as e:
                messagebox.showerror('Delete Failed', str(e))
```



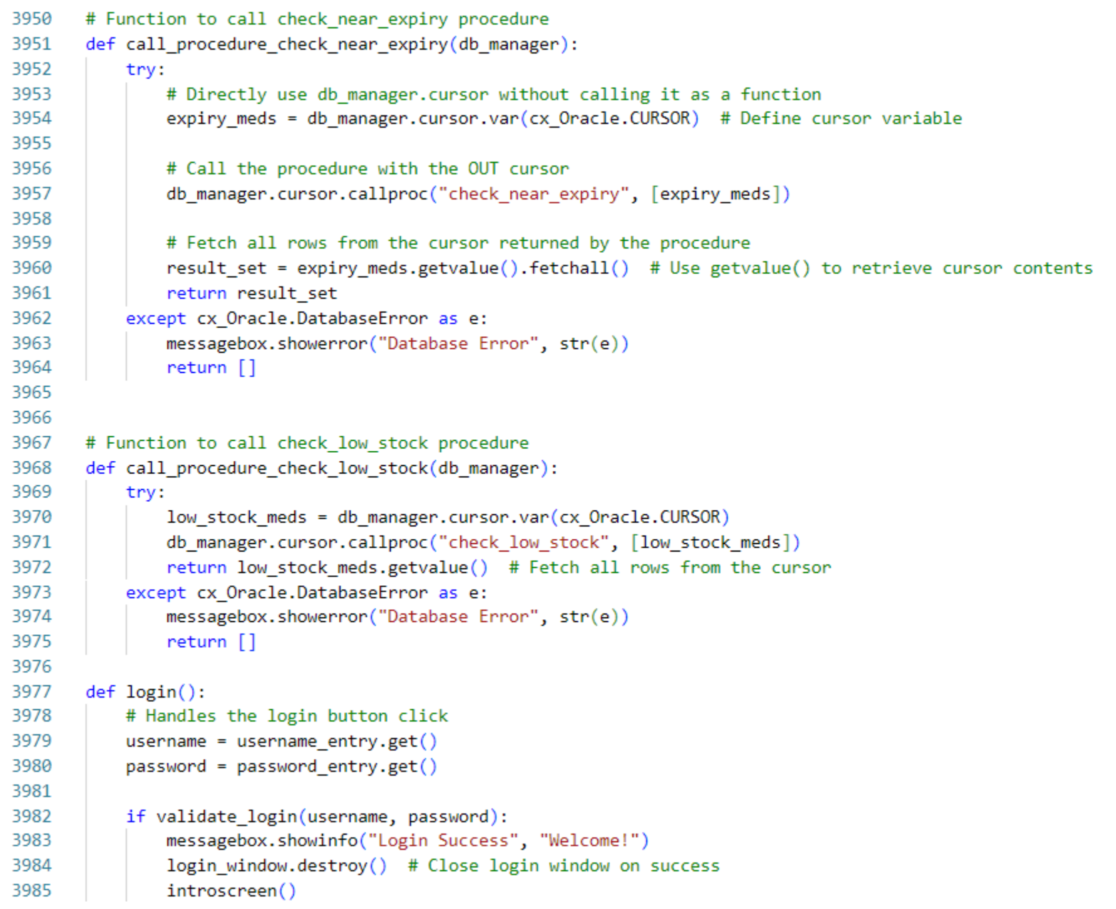
```python
def deletebysalesitems():
    def delete_sale_item():
        sale_item_id = saleitemidentry.get()  # Get the sale_item_id from the entry field

        # Check if Sale Item ID is in the correct format (e.g., SI002)
        if not re.match(r'^SI\d{3}$', sale_item_id):
            messagebox.showerror('Error!', 'Sale Item ID format is invalid. It should be in the format SIxxx (e.g., SI002).')
            return

        if sale_item_id == '':
            messagebox.showerror('Error!', 'Please enter a Sale Item ID to delete.')
        else:
            # Check if Sale Item ID exists in the SALES_ITEMS table
            try:
                check_query = "SELECT COUNT(*) FROM SALES_ITEMS WHERE sale_item_id=:1"
                result = dbms.fetch_query(check_query, (sale_item_id,))
                if result[0][0] == 0:
                    messagebox.showerror('Error!', 'Sale Item ID does not exist in the database.')
                    saleitemidentry.delete(0, END)
                    return
            except Exception as e:
                messagebox.showerror('Error!', f'Error checking Sale Item ID: {e}')
                saleitemidentry.delete(0, END)
                return
```



```python
            # If all checks pass, proceed with deletion
            try:
                delete_query = "DELETE FROM SALES_ITEMS WHERE sale_item_id = :1"
                dbms.execute_query(delete_query, (sale_item_id,))
                messagebox.showinfo('Success!', f'Sale Item ID {sale_item_id} deleted successfully.')
                saleitemidentry.delete(0, END)
            except Exception as e:
                messagebox.showerror('Delete Failed', str(e))
```



```python
def introscreen():
    # Check near-expiry medicines and display in a message box
    near_expiry_meds = call_procedure_check_near_expiry(dbms)
    if near_expiry_meds:
        near_expiry_message = "Near Expiry Medicines:\n\n"
        for med in near_expiry_meds:
            near_expiry_message += f"Medicine Name: {med[0]}, Expiry Date: {med[1].strftime('%Y-%m-%d')}, Days Remaining: {med[2]}\n"
        messagebox.showinfo("Near Expiry Medicines", near_expiry_message)

    # Check low-stock medicines and display in a message box
    low_stock_meds = call_procedure_check_low_stock(dbms)
    if low_stock_meds:
        low_stock_message = "Low Stock Medicines:\n\n"
        for med in low_stock_meds:
            low_stock_message += f"Medicine Name: {med[0]}, Quantity: {med[1]}\n"
        messagebox.showinfo("Low Stock Medicines", low_stock_message)

    root1.mainloop()
```



```python
# Function to call check_near_expiry procedure
def call_procedure_check_near_expiry(db_manager):
    try:
        # Directly use db_manager.cursor without calling it as a function
        expiry_meds = db_manager.cursor.var(cx_Oracle.CURSOR)  # Define cursor variable

        # Call the procedure with the OUT cursor
        db_manager.cursor.callproc("check_near_expiry", [expiry_meds])

        # Fetch all rows from the cursor returned by the procedure
        result_set = expiry_meds.getvalue().fetchall()  # Use getvalue() to retrieve cursor contents
        return result_set
    except cx_Oracle.DatabaseError as e:
        messagebox.showerror("Database Error", str(e))
        return []


# Function to call check_low_stock procedure
def call_procedure_check_low_stock(db_manager):
    try:
        low_stock_meds = db_manager.cursor.var(cx_Oracle.CURSOR)
        db_manager.cursor.callproc("check_low_stock", [low_stock_meds])
        return low_stock_meds.getvalue()  # Fetch all rows from the cursor
    except cx_Oracle.DatabaseError as e:
        messagebox.showerror("Database Error", str(e))
        return []

def login():
    # Handles the login button click
    username = username_entry.get()
    password = password_entry.get()

    if validate_login(username, password):
        messagebox.showinfo("Login Success", "Welcome!")
        login_window.destroy()  # Close login window on success
        introscreen()
```

```python
def check_password_strength(password):
    # Regular expression to check password strength
    if (len(password) >= 8 and
        re.search(r'[A-Z]', password) and  # At least one uppercase letter
        re.search(r'[a-z]', password) and  # At least one lowercase letter
        re.search(r'[0-9]', password) and  # At least one digit
        re.search(r'[!@#$%^&*()_+]', password)):  # At least one special character
        return True
    return False

def create_account():
    # Handles the creation of a new account
    c_username = c_username_entry.get()
    c_password = c_password_entry.get()

    # Check if username already exists
    if dbms.fetch_query("SELECT * FROM LOGIN WHERE username = :1", (c_username,)):
        messagebox.showerror("Account Creation Failed", "Username already exists.")
        return

    # Check password strength
    if not check_password_strength(c_password):
        messagebox.showerror("Weak Password", "Password must be at least 8 characters long,\
             contain uppercase and lowercase letters, a digit, and a special character.")
        return

    # Insert new credentials into the LOGIN table
    dbms.execute_query("INSERT INTO LOGIN (username, password) VALUES (:1, :2)", (c_username, c_password))

    messagebox.showinfo("Account Created", "Your account has been successfully created. Please log in.")
    create_login_window.destroy()
    show_login_window()

def validate_login(username, password):
    # Validates the username and password with the LOGIN table
    result = dbms.fetch_query("SELECT * FROM LOGIN WHERE username = :1 AND password = :2", (username, password))
    return bool(result)
```

DATABASE – BACKEND :

```
REM Medical Management System Database

SET ECHO ON;

DROP TABLE IF EXISTS SALES_ITEMS;
DROP TABLE IF EXISTS SALES;
DROP TABLE IF EXISTS PRESCRIPTION;
DROP TABLE IF EXISTS MEDICINE;
DROP TABLE IF EXISTS SUPPLIER;
DROP TABLE IF EXISTS CUSTOMER cascade constraints;

REM Creating table SUPPLIER

CREATE TABLE SUPPLIER(
supplier_id VARCHAR2(10) PRIMARY KEY,
s_name VARCHAR2(50) NOT NULL,
contact_number INT,
email VARCHAR2(100),
address VARCHAR2(150)
);

REM Creating table MEDICINE

CREATE TABLE MEDICINE(
medicine_id VARCHAR2(10) PRIMARY KEY,
m_name VARCHAR2(20) NOT NULL,
brand VARCHAR2(20),
batch_number VARCHAR2(10),
expiry_date DATE,
quantity INT DEFAULT 0,
price DECIMAL(10,2) NOT NULL,
supplier_id VARCHAR2(10),
FOREIGN KEY (supplier_id) REFERENCES SUPPLIER(supplier_id) ON DELETE CASCADE
);

REM Creating table CUSTOMER

CREATE TABLE CUSTOMER(
customer_id VARCHAR2(10) PRIMARY KEY,
c_name VARCHAR2(30) NOT NULL,
contact_number INT,
email VARCHAR2(40),
address VARCHAR2(50)
);
REM Creating table PRESCRIPTION

CREATE TABLE PRESCRIPTION(
prescription_id VARCHAR2(15) PRIMARY KEY,
customer_id VARCHAR2(10),
doctor_name VARCHAR2(25),
prescription_date DATE,
dosage VARCHAR2(30),
frequency VARCHAR2(40),
duration VARCHAR2(10),
additional_instructions VARCHAR2(75),
FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id) ON DELETE CASCADE
);

REM Creating table SALES

CREATE TABLE SALES(
sale_id VARCHAR2(10) PRIMARY KEY,
customer_id VARCHAR2(10),
sale_date DATE NOT NULL,
total_amount DECIMAL(10,2) NOT NULL,
payment_method VARCHAR2(20),
FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id) ON DELETE CASCADE
);
```

```sql
REM Creating table SALES_ITEMS

CREATE TABLE SALES_ITEMS (
sale_item_id VARCHAR2(10) PRIMARY KEY,
sale_id VARCHAR2(10),
medicine_id VARCHAR2(10),
quantity INT DEFAULT 1,
price_per_unit DECIMAL(10, 2),
subtotal DECIMAL(10,2),
FOREIGN KEY (sale_id) REFERENCES SALES(sale_id) ON DELETE CASCADE,
FOREIGN KEY (medicine_id) REFERENCES MEDICINE(medicine_id) ON DELETE CASCADE
);

REM Creating login table

CREATE TABLE IF NOT EXISTS LOGIN (
username VARCHAR2(50) PRIMARY KEY,
password VARCHAR2(50) NOT NULL);
```

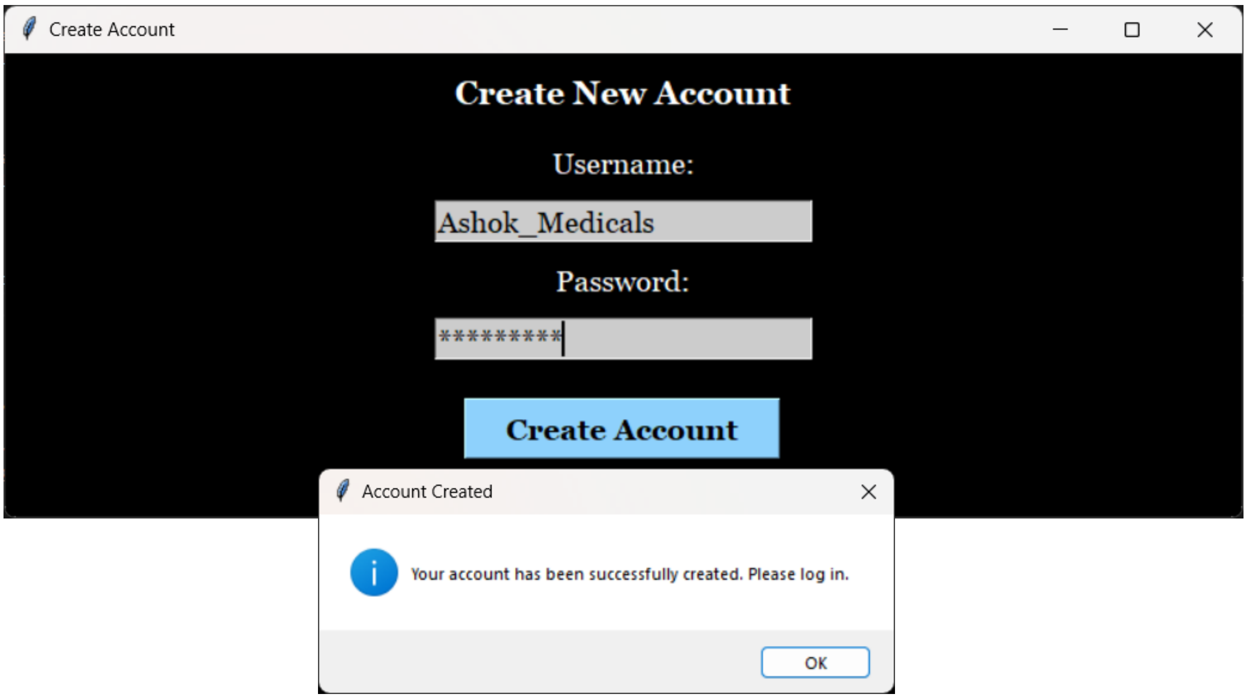
```sql
REM Inserting into SUPPLIER Table

INSERT INTO SUPPLIER (supplier_id, s_name, contact_number, email, address)
VALUES
    ('S005', 'Ramesh Kumar', 9876543210, 'ramesh.kumar@example.com', '123, Kottivakkam, Chennai, Tamil Nadu'),
    ('S012', 'Kavitha Nair', 9654321098, 'kavitha.nair@example.com', '456, R.S. Puram, Coimbatore, Tamil Nadu'),
    ('S021', 'Arunachalam V', 9843214567, 'arunachalam.v@example.com', '789, Simmakkal, Madurai, Tamil Nadu'),
    ('S009', 'Lakshmi S', 9887654321, 'lakshmi.s@example.com', '321, Kilpauk, Chennai, Tamil Nadu'),
    ('S014', 'Mohan Raj', 9776543210, 'mohan.raj@example.com', '654, Tirunelveli Road, Tirunelveli, Tamil Nadu'),
    ('S001', 'Anjali Devi', 9567890123, 'anjali.devi@example.com', '111, Perundurai, Erode, Tamil Nadu'),
    ('S017', 'Suresh Babu', 9443216789, 'suresh.babu@example.com', '222, Thiru Vi Ka Nagar, Tiruchirappalli, Tamil Nadu'),
    ('S018', 'Vani Duraisamy', 9334567890, 'vani.duraisamy@example.com', '333, Marappalam, Salem, Tamil Nadu'),
    ('S004', 'Ganesh Prasad', 9198765432, 'ganesh.prasad@example.com', '444, Alangulam, Dindigul, Tamil Nadu'),
    ('S011', 'Aarti Sundar', 9654321987, 'aarti.sundar@example.com', '555, Ekkatuthangal, Vellore, Tamil Nadu'),
    ('S022', 'Karthik Subramanian', 9478561230, 'karthik.subramanian@example.com', '666, Thanjavur Road, Thanjavur, Tamil Nadu'),
    ('S023', 'Vidhya Ramesh', 9345678901, 'vidhya.ramesh@example.com', '777, Karur Bypass Road, Karur, Tamil Nadu'),
    ('S003', 'Rajesh Kannan', 9845621789, 'rajesh.kannan@example.com', '888, Kanyakumari Road, Kanyakumari, Tamil Nadu'),
    ('S015', 'Sankar Raghavan', 9654321098, 'sankar.raghavan@example.com', '999, Palayamkottai, Thoothukudi, Tamil Nadu'),
    ('S019', 'Naveen Kumar', 9567890234, 'naveen.kumar@example.com', '101, Dindigul Main Road, Namakkal, Tamil Nadu'),
    ('S006', 'Srinivasan Mani', 9445698743, 'srinivasan.mani@example.com', '202, Thiruvalluvar Street, Pudukkottai, Tamil Nadu'),
    ('S020', 'Priya Natarajan', 9334567890, 'priya.natarajan@example.com', '303, Veeravanallur, Tiruppur, Tamil Nadu'),
    ('S008', 'Balaji Reddy', 9198765432, 'balaji.reddy@example.com', '404, Pollachi Road, Pollachi, Tamil Nadu'),
    ('S002', 'Deepika Saravanan', 9887654321, 'deepika.saravanan@example.com', '505, Aruppukottai, Tamil Nadu'),
    ('S016', 'Vikram Selvam', 9478561234, 'vikram.selvam@example.com', '606, Kallakurichi, Virudhunagar, Tamil Nadu'),
    ('S010', 'Aditi Raghavan', 9345678901, 'aditi.raghavan@example.com', '707, Chengalpattu Road, Chengalpattu, Tamil Nadu'),
    ('S013', 'Gayathri Menon', 9567890123, 'gayathri.menon@example.com', '808, Thiruvallur, Perambalur, Tamil Nadu'),
    ('S024', 'Harish Karthik', 9443216789, 'harish.karthik@example.com', '909, Sathy Road, Karamadai, Tamil Nadu'),
    ('S007', 'Manoj Krishnan', 9345126780, 'manoj.krishnan@example.com', '888, Gandhi Road, Thanjavur, Tamil Nadu'),
    ('S025', 'Vasanth Kumar', 9743214567, 'vasanth.kumar@example.com', '010, Thiruvalankadu, Cuddalore, Tamil Nadu');


REM Inserting into MEDICINE Table

INSERT INTO MEDICINE (medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id)
VALUES
    ('M005', 'Paracetamol', 'Acetaminophen', 'BATCH123', TO_DATE('2026-05-01', 'YYYY-MM-DD'), 10, 50.00, 'S023'),
    ('M018', 'Amoxicillin', 'Amoxil', 'BATCH543', TO_DATE('2025-09-15', 'YYYY-MM-DD'), 15, 120.00, 'S010'),
    ('M011', 'Ibuprofen', 'Advil', 'BATCH213', TO_DATE('2026-02-20', 'YYYY-MM-DD'), 200, 75.50, 'S018'),
    ('M002', 'Lisinopril', 'Prinivil', 'BATCH321', TO_DATE('2025-12-10', 'YYYY-MM-DD'), 90, 80.00, 'S002'),
    ('M024', 'Cetirizine', 'Zyrtec', 'BATCH999', TO_DATE('2026-04-05', 'YYYY-MM-DD'), 120, 45.00, 'S017'),
    ('M014', 'Metformin', 'Glucophage', 'BATCH888', TO_DATE('2025-08-30', 'YYYY-MM-DD'), 75, 60.00, 'S022'),
    ('M007', 'Amlodipine', 'Norvasc', 'BATCH456', TO_DATE('2026-01-25', 'YYYY-MM-DD'), 60, 100.00, 'S005'),
    ('M022', 'Omeprazole', 'Prilosec', 'BATCH111', TO_DATE('2025-07-18', 'YYYY-MM-DD'), 80, 55.00, 'S004'),
    ('M016', 'Simvastatin', 'Zocor', 'BATCH222', TO_DATE('2024-11-30', 'YYYY-MM-DD'), 50, 90.00, 'S015'),
    ('M001', 'Clopidogrel', 'Plavix', 'BATCH333', TO_DATE('2025-10-20', 'YYYY-MM-DD'), 40, 85.00, 'S019'),
    ('M025', 'Levothyroxine', 'Synthroid', 'BATCH777', TO_DATE('2026-08-14', 'YYYY-MM-DD'), 110, 70.00, 'S021'),
    ('M013', 'Doxycycline', 'Vibramycin', 'BATCH444', TO_DATE('2025-03-11', 'YYYY-MM-DD'), 30, 40.00, 'S006'),
    ('M003', 'Hydrochlorothiazide', 'Hydrodiuril', 'BATCH555', TO_DATE('2026-03-12', 'YYYY-MM-DD'), 95, 65.00, 'S014'),
    ('M019', 'Furosemide', 'Lasix', 'BATCH888', TO_DATE('2026-12-25', 'YYYY-MM-DD'), 70, 50.00, 'S012'),
    ('M008', 'Montelukast', 'Singulair', 'BATCH666', TO_DATE('2025-05-09', 'YYYY-MM-DD'), 85, 60.00, 'S009'),
    ('M012', 'Pantoprazole', 'Protonix', 'BATCH999', TO_DATE('2025-11-01', 'YYYY-MM-DD'), 150, 95.00, 'S008'),
    ('M004', 'Cetirizine', 'Zyrtec', 'BATCH444', TO_DATE('2026-10-30', 'YYYY-MM-DD'), 65, 45.00, 'S020'),
    ('M015', 'Venlafaxine', 'Effexor', 'BATCH777', TO_DATE('2025-06-17', 'YYYY-MM-DD'), 120, 150.00, 'S001'),
    ('M006', 'Escitalopram', 'Lexapro', 'BATCH555', TO_DATE('2026-09-09', 'YYYY-MM-DD'), 75, 80.00, 'S003'),
    ('M020', 'Sertraline', 'Zoloft', 'BATCH222', TO_DATE('2025-02-02', 'YYYY-MM-DD'), 50, 70.00, 'S011'),
    ('M009', 'Bupropion', 'Wellbutrin', 'BATCH888', TO_DATE('2024-12-05', 'YYYY-MM-DD'), 80, 65.00, 'S016'),
    ('M010', 'Azithromycin', 'Zithromax', 'BATCH112', TO_DATE('2025-08-15', 'YYYY-MM-DD'), 100, 90.00, 'S005'),
    ('M017', 'Atorvastatin', 'Lipitor', 'BATCH334', TO_DATE('2026-04-22', 'YYYY-MM-DD'), 85, 75.00, 'S010'),
    ('M021', 'Gabapentin', 'Neurontin', 'BATCH556', TO_DATE('2025-10-05', 'YYYY-MM-DD'), 70, 120.00, 'S012'),
    ('M023', 'Losartan', 'Cozaar', 'BATCH778', TO_DATE('2026-02-18', 'YYYY-MM-DD'), 60, 65.00, 'S018');
```

```
REM Inserting into CUSTOMER TABLE

INSERT INTO CUSTOMER (customer_id, c_name, contact_number, email, address)
VALUES
    ('C001', 'Rajesh Sharma', 9876543210, 'rajesh.sharma@example.com', '123, Anna Nagar, Chennai, Tamil Nadu'),
    ('C002', 'Sita Rao', 9988776655, 'sita.rao@example.com', '456, Kotturpuram, Chennai, Tamil Nadu'),
    ('C003', 'Vikram Singh', 9701234567, 'vikram.singh@example.com', '789, T. Nagar, Chennai, Tamil Nadu'),
    ('C004', 'Anjali Gupta', 9598765432, 'anjali.gupta@example.com', '321, Mylapore, Chennai, Tamil Nadu'),
    ('C005', 'Karan Mehta', 9843216789, 'karan.mehta@example.com', '654, Adyar, Chennai, Tamil Nadu'),
    ('C006', 'Pooja Verma', 9871234560, 'pooja.verma@example.com', '987, Saidapet, Chennai, Tamil Nadu'),
    ('C007', 'Rahul Jain', 9912345678, 'rahul.jain@example.com', '654, Kodambakkam, Chennai, Tamil Nadu'),
    ('C008', 'Suresh Babu', 9865432109, 'suresh.babu@example.com', '321, T. Nagar, Chennai, Tamil Nadu'),
    ('C009', 'Meena Reddy', 9809876543, 'meena.reddy@example.com', '432, Alwarpet, Chennai, Tamil Nadu'),
    ('C010', 'Nitin Sharma', 9798765432, 'nitin.sharma@example.com', '543, Tambaram, Chennai, Tamil Nadu'),
    ('C011', 'Aditi Raghavan', 9638527410, 'aditi.raghavan@example.com', '876, Mambalam, Chennai, Tamil Nadu'),
    ('C012', 'Sunita Menon', 9743214567, 'sunita.menon@example.com', '234, Velachery, Chennai, Tamil Nadu'),
    ('C013', 'Rajkumar Iyer', 9865321470, 'rajkumar.iyer@example.com', '345, Kottivakkam, Chennai, Tamil Nadu'),
    ('C014', 'Lakshmi Nair', 9765432109, 'lakshmi.nair@example.com', '456, Nungambakkam, Chennai, Tamil Nadu'),
    ('C015', 'Vivek Krishnan', 9812345670, 'vivek.krishnan@example.com', '567, Ashok Nagar, Chennai, Tamil Nadu'),
    ('C016', 'Priya Das', 9709876543, 'priya.das@example.com', '678, Sholinganallur, Chennai, Tamil Nadu'),
    ('C017', 'Niharika Ramesh', 9898765432, 'niharika.ramesh@example.com', '789, Ekkatuthangal, Chennai, Tamil Nadu'),
    ('C018', 'Ganesh Kumar', 9945678901, 'ganesh.kumar@example.com', '890, Kotturpuram, Chennai, Tamil Nadu'),
    ('C019', 'Srinivas Balakrishnan', 9901234567, 'srinivas.balakrishnan@example.com', '321, Thiruvanmiyur, Chennai, Tamil Nadu'),
    ('C020', 'Rita Choudhury', 9856789012, 'rita.choudhury@example.com', '432, Kottivakkam, Chennai, Tamil Nadu'),
    ('C021', 'Arvind Kumar', 9812347650, 'arvind.kumar@example.com', '543, Anna Nagar, Chennai, Tamil Nadu'),
    ('C022', 'Deepak Reddy', 9701253486, 'deepak.reddy@example.com', '654, T. Nagar, Chennai, Tamil Nadu'),
    ('C023', 'Samantha Joshi', 9897654321, 'samantha.joshi@example.com', '765, Besant Nagar, Chennai, Tamil Nadu'),
    ('C024', 'Akash Bhatia', 9912345678, 'akash.bhatia@example.com', '876, Kottivakkam, Chennai, Tamil Nadu'),
    ('C025', 'Gita Pillai', 9909876543, 'gita.pillai@example.com', '987, Mylapore, Chennai, Tamil Nadu');
```

```
REM Inserting into SALES Table

INSERT INTO SALES (sale_id, customer_id, sale_date, total_amount, payment_method)
VALUES
    ('S001', 'C007', TO_DATE('2024-10-10', 'YYYY-MM-DD'), 201.00, 'Credit Card'),
    ('S002', 'C004', TO_DATE('2024-10-11', 'YYYY-MM-DD'), 255.00, 'UPI'),
    ('S003', 'C010', TO_DATE('2024-10-12', 'YYYY-MM-DD'), 220.00, 'Cash'),
    ('S004', 'C001', TO_DATE('2024-10-13', 'YYYY-MM-DD'), 165.50, 'UPI'),
    ('S005', 'C003', TO_DATE('2024-10-14', 'YYYY-MM-DD'), 175.00, 'Debit Card'),
    ('S006', 'C012', TO_DATE('2024-10-15', 'YYYY-MM-DD'), 140.00, 'Cash'),
    ('S007', 'C015', TO_DATE('2024-10-16', 'YYYY-MM-DD'), 50.00, 'Cash'),
    ('S008', 'C022', TO_DATE('2024-10-17', 'YYYY-MM-DD'), 100.00, 'Cash'),
    ('S009', 'C005', TO_DATE('2024-10-18', 'YYYY-MM-DD'), 170.00, 'UPI'),
    ('S010', 'C020', TO_DATE('2024-10-19', 'YYYY-MM-DD'), 95.00, 'Debit Card');



REM Inserting into SALES_ITEMS Table

INSERT INTO SALES_ITEMS (sale_item_id, sale_id, medicine_id, quantity, price_per_unit, subtotal)
VALUES
    ('SI001', 'S001', 'M005', 1, 50.00, 50.00),
    ('SI002', 'S001', 'M011', 2, 75.50, 151.00),
    ('SI003', 'S002', 'M018', 1, 120.00, 120.00),
    ('SI004', 'S002', 'M024', 3, 45.00, 135.00),
    ('SI005', 'S003', 'M005', 2, 50.00, 100.00),
    ('SI006', 'S003', 'M018', 1, 120.00, 120.00),
    ('SI007', 'S004', 'M024', 2, 45.00, 90.00),
    ('SI008', 'S004', 'M011', 1, 75.50, 75.50),
    ('SI009', 'S005', 'M013', 3, 40.00, 120.00),
    ('SI010', 'S005', 'M022', 1, 55.00, 55.00),
    ('SI011', 'S006', 'M020', 2, 70.00, 140.00),
    ('SI012', 'S007', 'M019', 1, 50.00, 50.00),
    ('SI013', 'S008', 'M007', 1, 100.00, 100.00),
    ('SI014', 'S009', 'M001', 2, 85.00, 170.00),
    ('SI015', 'S010', 'M012', 1, 95.00, 95.00);
```

```
COMMIT;

SELECT * FROM SUPPLIER;
SELECT * FROM CUSTOMER;
SELECT * FROM MEDICINE;
SELECT * FROM PRESCRIPTION;
SELECT * FROM SALES;
SELECT * FROM SALES_ITEMS;


CREATE OR REPLACE PROCEDURE check_near_expiry(expiry_meds OUT SYS_REFCURSOR)
AS
BEGIN
    OPEN expiry_meds FOR
    SELECT M_NAME, EXPIRY_DATE, ROUND(EXPIRY_DATE - SYSDATE) AS days_remaining
    FROM MEDICINE
    WHERE EXPIRY_DATE - SYSDATE <= 30;
END;
/


CREATE OR REPLACE PROCEDURE check_low_stock(low_stock_meds OUT SYS_REFCURSOR)
AS
BEGIN
    OPEN low_stock_meds FOR
    SELECT M_NAME, QUANTITY
    FROM MEDICINE
    WHERE QUANTITY < 25;
END;
/

DROP TRIGGER update_medicine_stock;

CREATE OR REPLACE TRIGGER update_medicine_stock
AFTER INSERT ON SALES_ITEMS
FOR EACH ROW
BEGIN
    -- Check if the medicine exists before updating
    UPDATE MEDICINE
    SET quantity = quantity - :NEW.quantity
    WHERE medicine_id = :NEW.medicine_id
    AND EXISTS (SELECT 1 FROM MEDICINE WHERE medicine_id = :NEW.medicine_id);
END;
/
```

OUTPUT – FRONT END:


Create Account

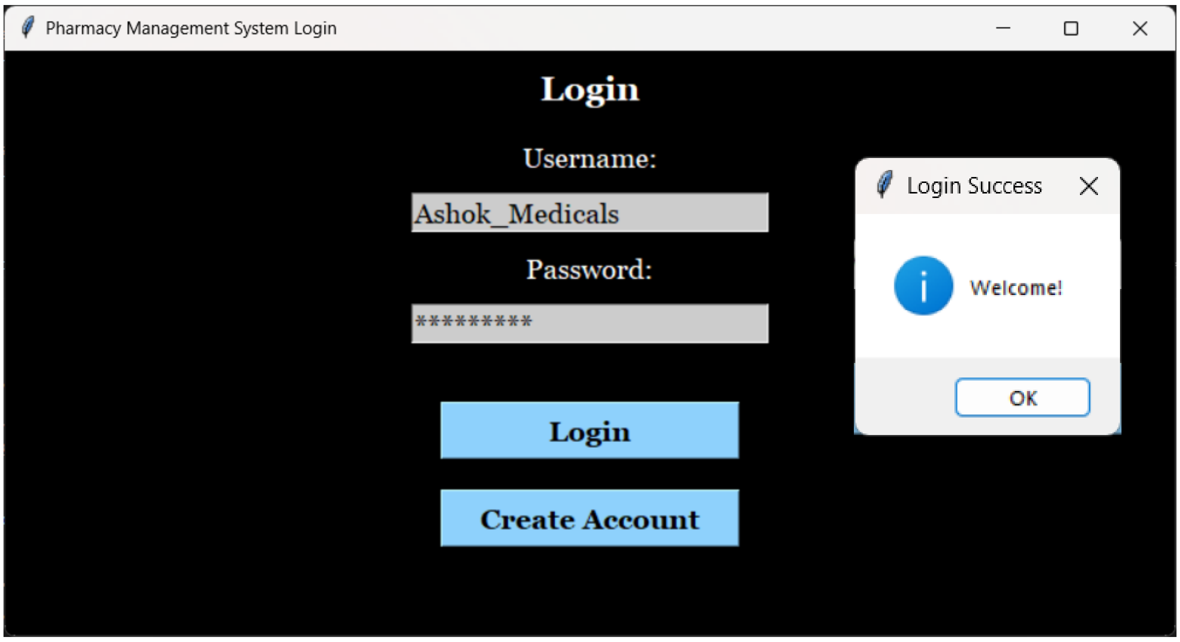
Create New Account

Username: Ashok_Medicals

Password: *********

Account Created

Your account has been successfully created. Please log in.


Pharmacy Management System Login

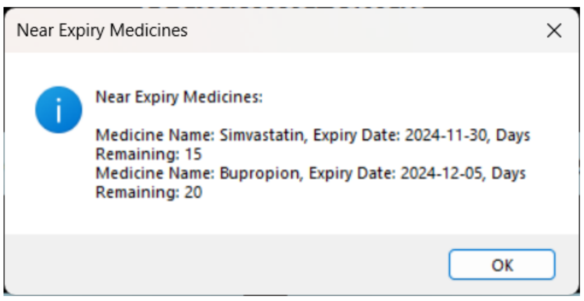
Login

Username: Ashok_Medicals

Password: *********

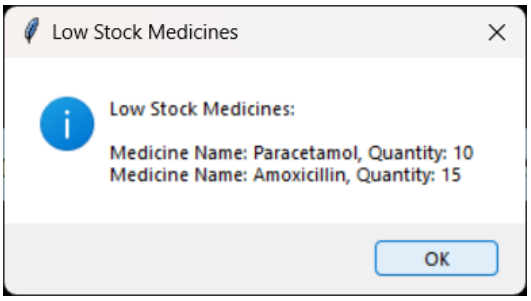
Login Success

Welcome!


Near Expiry Medicines

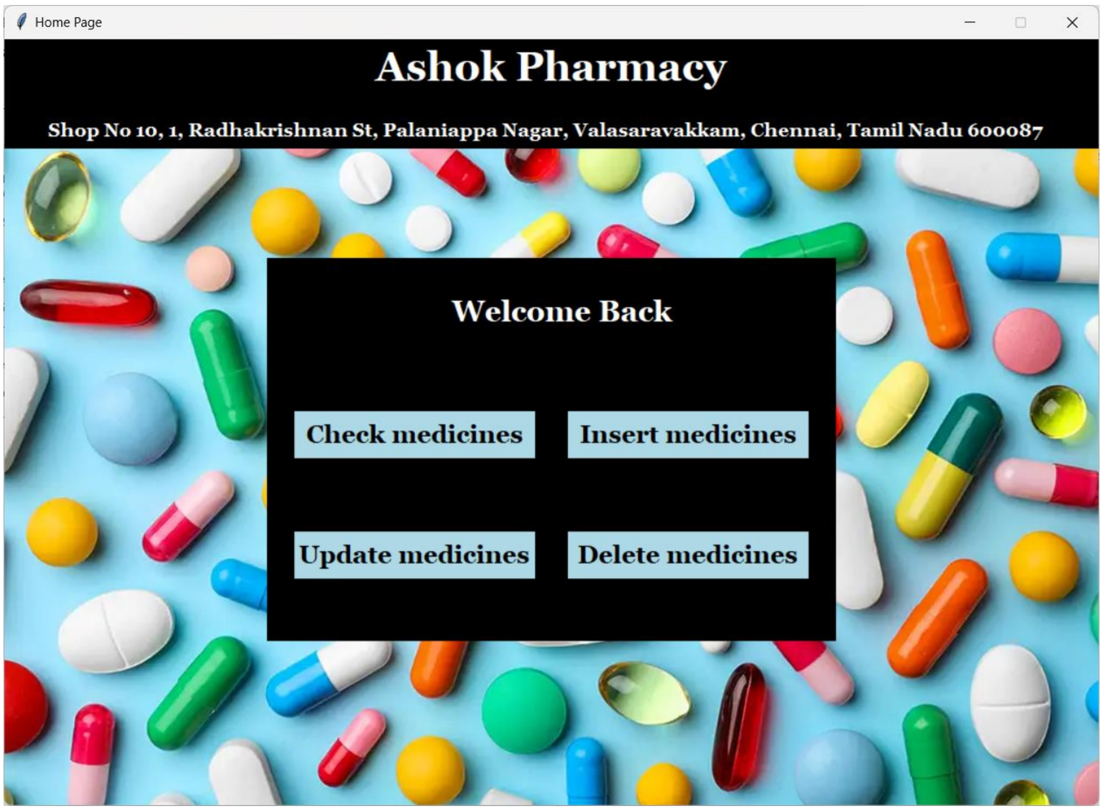
Near Expiry Medicines:

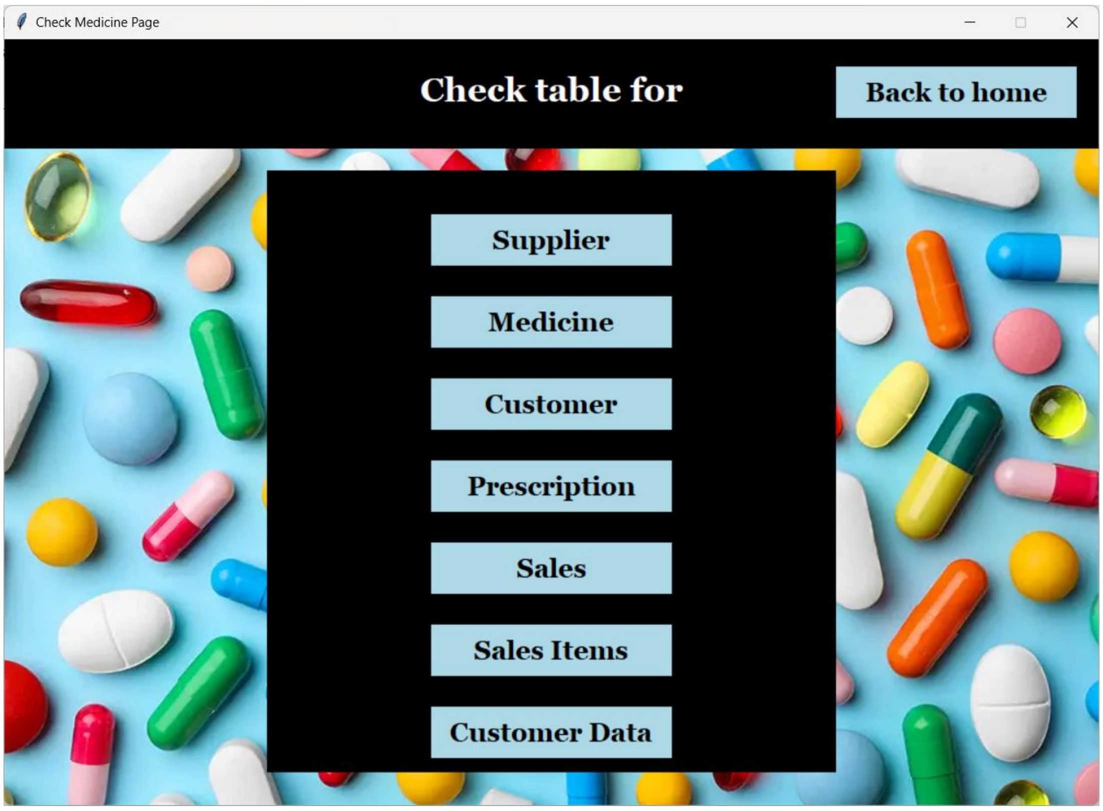
Medicine Name: Simvastatin, Expiry Date: 2024-11-30, Days Remaining: 15
Medicine Name: Bupropion, Expiry Date: 2024-12-05, Days Remaining: 20

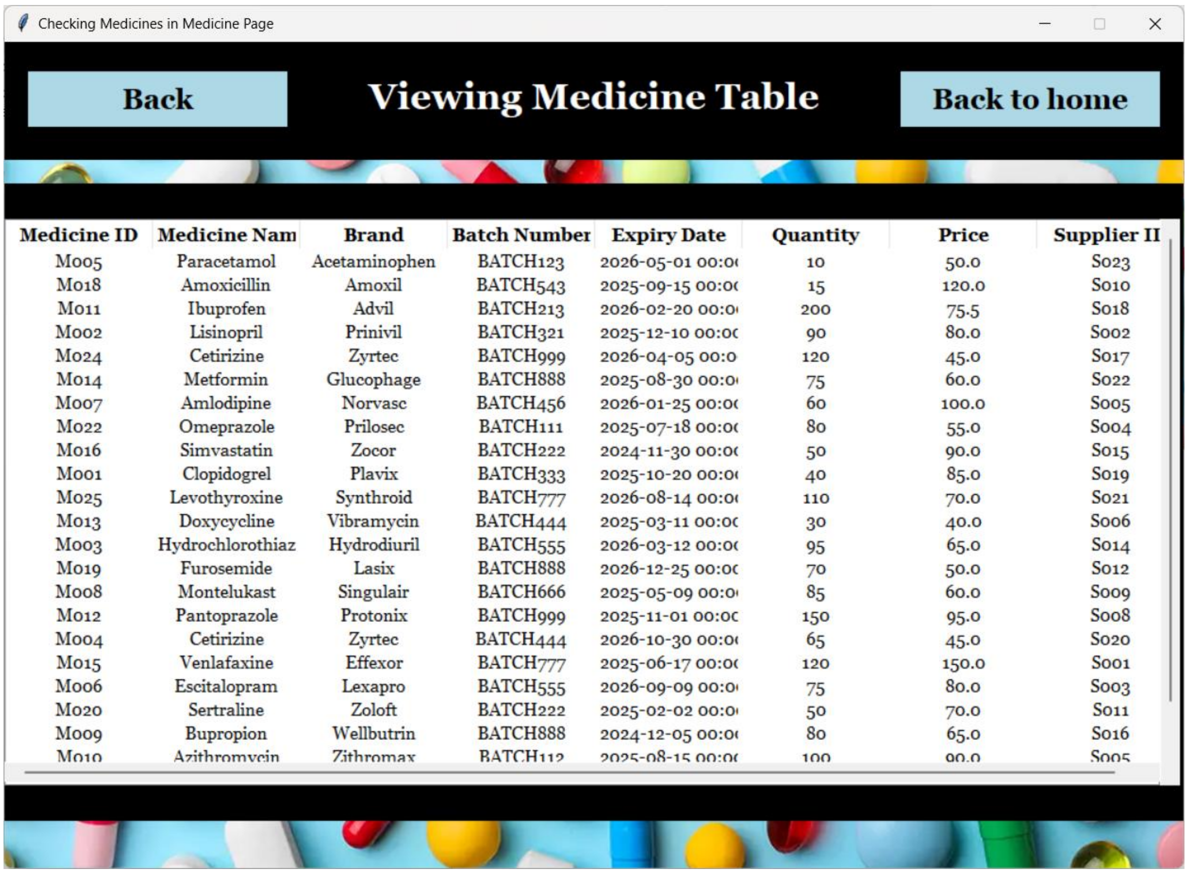

Low Stock Medicines

Low Stock Medicines:

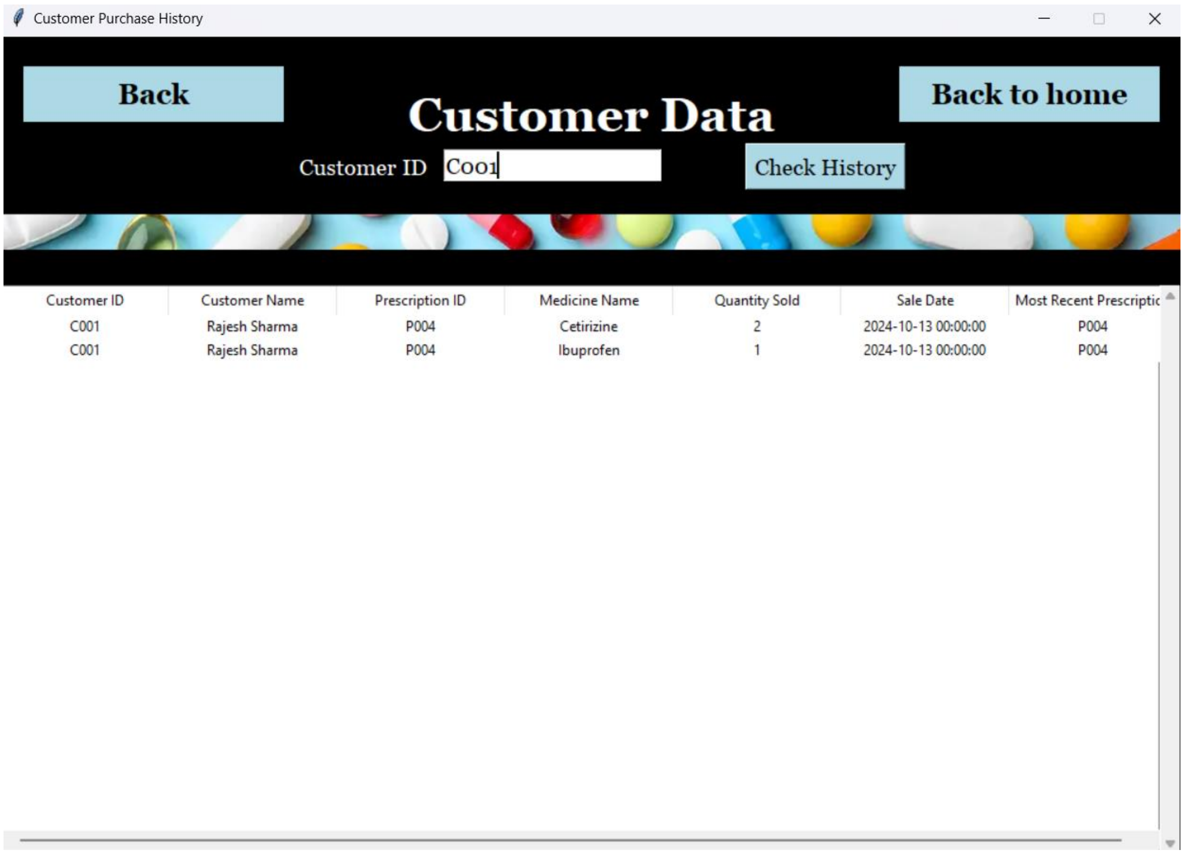
Medicine Name: Paracetamol, Quantity: 10
Medicine Name: Amoxicillin, Quantity: 15


Home Page
Ashok Pharmacy
Shop No 10, 1, Radhakrishnan St, Palaniappa Nagar, Valasaravakkam, Chennai, Tamil Nadu 600087
Welcome Back
Check medicines
Insert medicines
Update medicines
Delete medicines

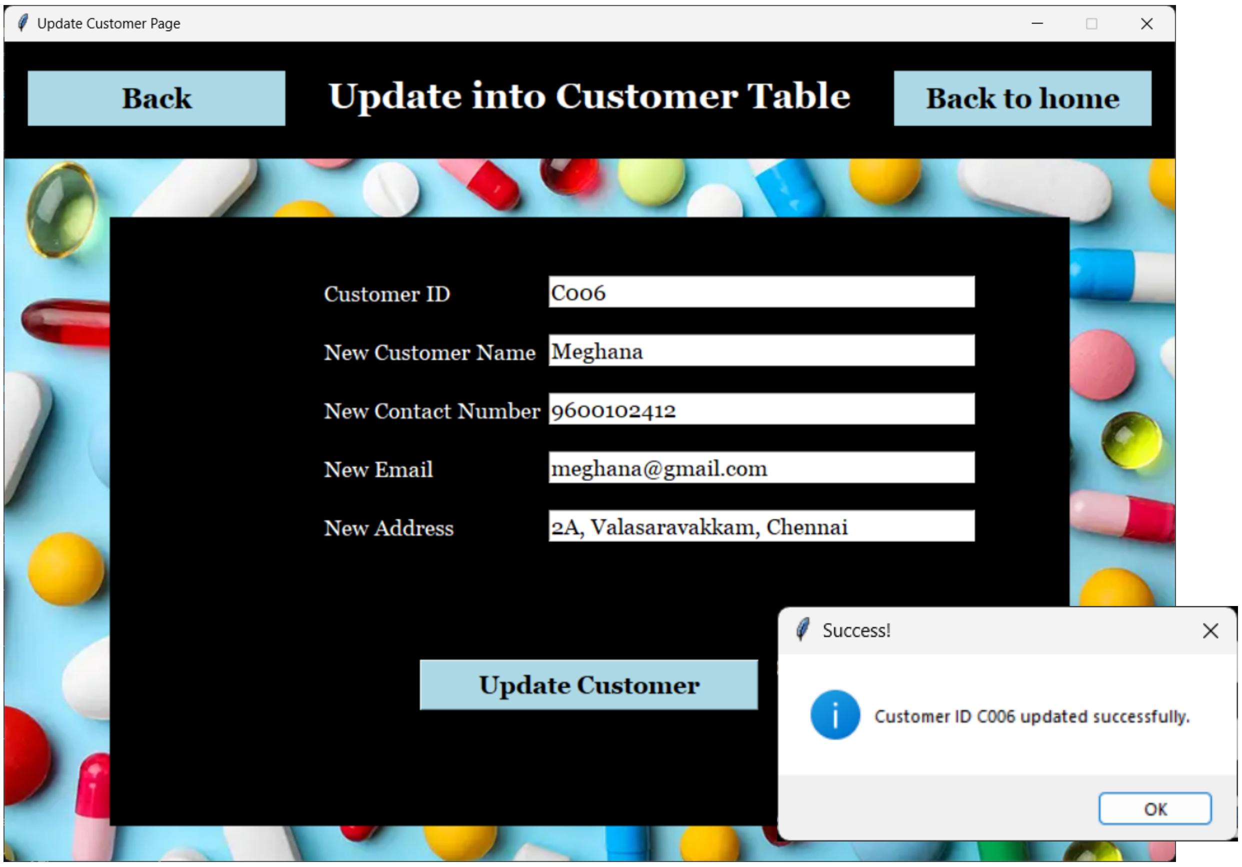

Check Medicine Page
Check table for
Back to home
Supplier
Medicine
Customer
Prescription
Sales
Sales Items
Customer Data



Medicine ID	Medicine Nam	Brand	Batch Number	Expiry Date	Quantity	Price	Supplier II
M005	Paracetamol	Acetaminophen	BATCH123	2026-05-01 00:0	10	50.0	S023
M018	Amoxicillin	Amoxil	BATCH543	2025-09-15 00:0	15	120.0	S010
M011	Ibuprofen	Advil	BATCH213	2026-02-20 00:0	200	75.5	S018
M002	Lisinopril	Prinivil	BATCH321	2025-12-10 00:0	90	80.0	S002
M024	Cetirizine	Zyrtec	BATCH999	2026-04-05 00:0	120	45.0	S017
M014	Metformin	Glucophage	BATCH888	2025-08-30 00:0	75	60.0	S022
M007	Amlodipine	Norvasc	BATCH456	2026-01-25 00:0	60	100.0	S005
M022	Omeprazole	Prilosec	BATCH111	2025-07-18 00:0	80	55.0	S004
M016	Simvastatin	Zocor	BATCH222	2024-11-30 00:0	50	90.0	S015
M001	Clopidogrel	Plavix	BATCH333	2025-10-20 00:0	40	85.0	S019
M025	Levothyroxine	Synthroid	BATCH777	2026-08-14 00:0	110	70.0	S021
M013	Doxycycline	Vibramycin	BATCH444	2025-03-11 00:0	30	40.0	S006
M003	Hydrochlorothiaz	Hydrodiuril	BATCH555	2026-03-12 00:0	95	65.0	S014
M019	Furosemide	Lasix	BATCH888	2026-12-25 00:0	70	50.0	S012
M008	Montelukast	Singulair	BATCH666	2025-05-09 00:0	85	60.0	S009
M012	Pantoprazole	Protonix	BATCH999	2025-11-01 00:0	150	95.0	S008
M004	Cetirizine	Zyrtec	BATCH444	2026-10-30 00:0	65	45.0	S020
M015	Venlafaxine	Effexor	BATCH777	2025-06-17 00:0	120	150.0	S001
M006	Escitalopram	Lexapro	BATCH555	2026-09-09 00:0	75	80.0	S003
M020	Sertraline	Zoloft	BATCH222	2025-02-02 00:0	50	70.0	S011
M009	Bupropion	Wellbutrin	BATCH888	2024-12-05 00:0	80	65.0	S016
M010	Azithromycin	Zithromax	BATCH112	2025-08-15 00:0	100	90.0	S005



Customer ID	Customer Name	Prescription ID	Medicine Name	Quantity Sold	Sale Date	Most Recent Prescriptic
C001	Rajesh Sharma	P004	Cetirizine	2	2024-10-13 00:00:00	P004
C001	Rajesh Sharma	P004	Ibuprofen	1	2024-10-13 00:00:00	P004


Update Customer Page

Update into Customer Table

Customer ID C006

New Customer Name Meghana

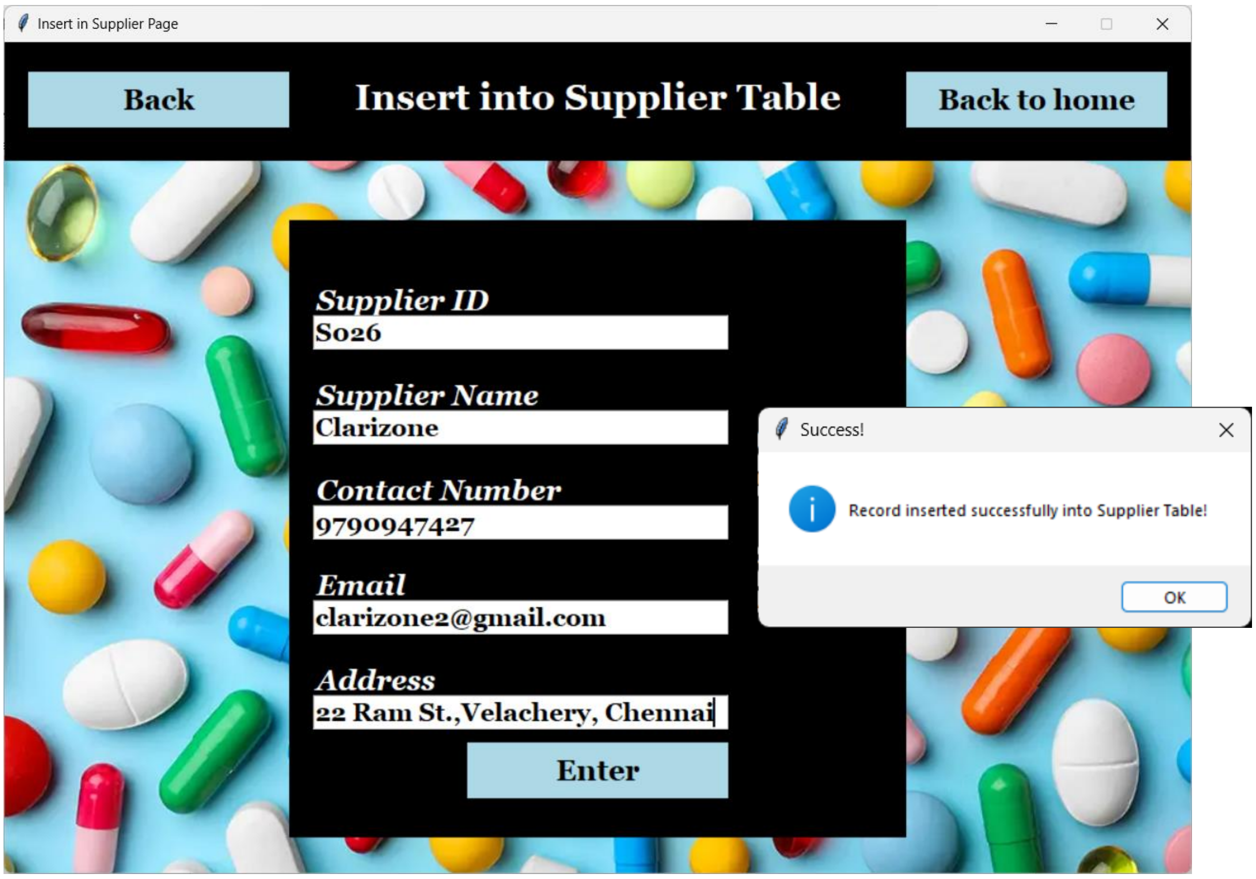
New Contact Number 9600102412

New Email meghana@gmail.com

New Address 2A, Valasaravakkam, Chennai

Success!

Customer ID C006 updated successfully.


Insert in Supplier Page

Insert into Supplier Table

Supplier ID S026

Supplier Name Clarizone

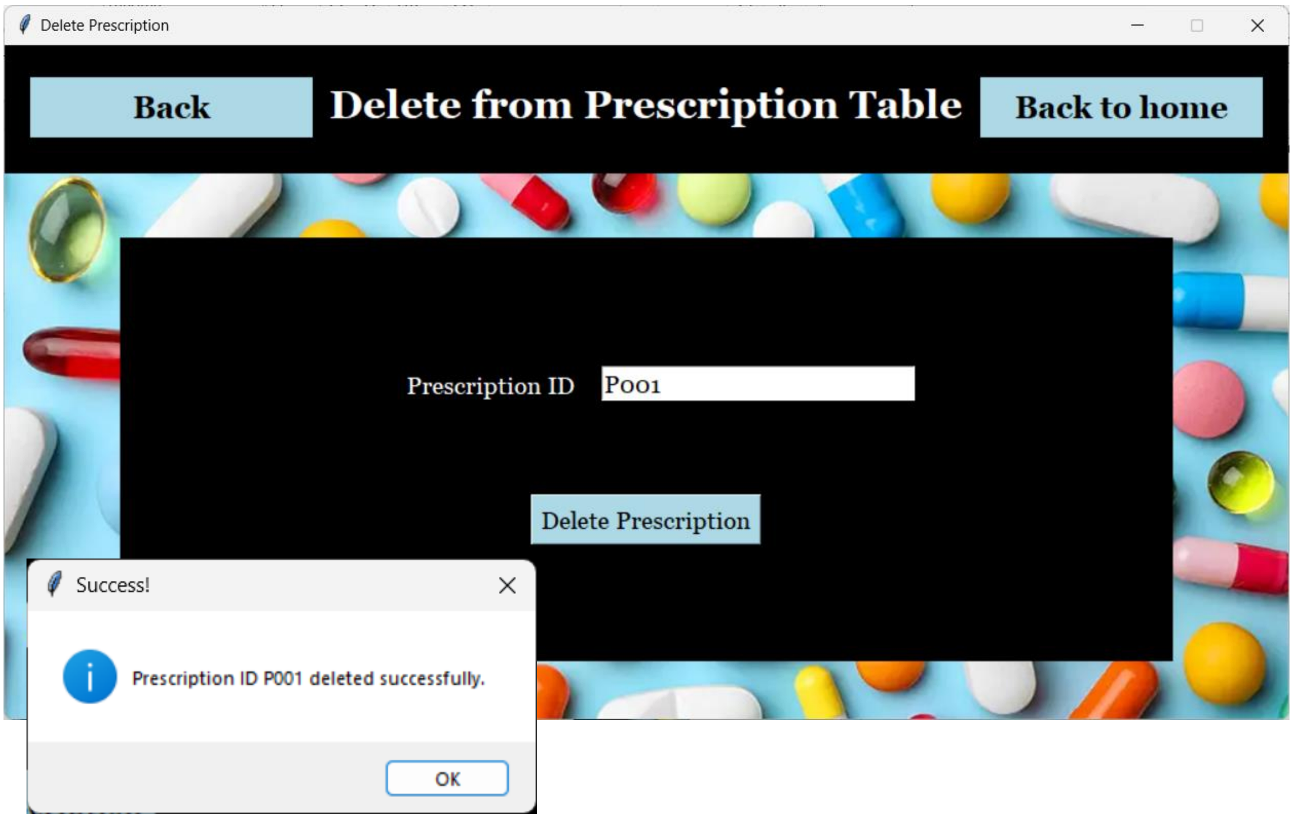
Contact Number 9790947427

Email clarizone2@gmail.com

Address 22 Ram St.,Velachery, Chennai

Success!

Record inserted successfully into Supplier Table!



Prescription ID P001

Prescription ID P001 deleted successfully.

RESULT: