Inventory Management System

I'll create a comprehensive Inventory Management System using Python with SQLite and Tkinter as requested. This solution will include product management, sales tracking, and stock control functionality.

```python
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
from datetime import datetime

class InventoryManagementSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Inventory Management System")
        self.root.geometry("1000x700")
        self.root.configure(bg='#f0f0f0')

        # Create database and tables
        self.create_database()

        # Style configuration
        self.style = ttk.Style()
        self.style.theme_use('clam')

        # Configure colors
```

```
        self.style.configure('TFrame', background='#f0f0f0')
        self.style.configure('Header.TLabel', background='#333', foreground='white', font=('Arial', 12, 'bold'))
        self.style.configure('TButton', font=('Arial', 10), background='#4CAF50', foreground='black')
        self.style.map('TButton', background=[('active', '#45a049')])

        # Create main frames
        self.header_frame = ttk.Frame(root, style='Header.TFrame', height=50)
        self.header_frame.pack(fill=tk.X, pady=(0, 10))

        self.content_frame = ttk.Frame(root)
        self.content_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Header
        ttk.Label(self.header_frame, text="Inventory Management System", style='Header.TLabel').pack(side=tk.LEFT, padx=10)

        # Create tabs
        self.tab_control = ttk.Notebook(self.content_frame)

        self.products_tab = ttk.Frame(self.tab_control)
        self.sales_tab = ttk.Frame(self.tab_control)
        self.stock_tab = ttk.Frame(self.tab_control)
        self.reports_tab = ttk.Frame(self.tab_control)

        self.tab_control.add(self.products_tab, text='Products')
```

```
        self.tab_control.add(self.sales_tab, text='Sales')
        self.tab_control.add(self.stock_tab, text='Stock Management')
        self.tab_control.add(self.reports_tab, text='Reports')

        self.tab_control.pack(expand=1, fill='both')

        # Initialize tabs
        self.setup_products_tab()
        self.setup_sales_tab()
        self.setup_stock_tab()
        self.setup_reports_tab()

        # Load initial data
        self.load_products()
        self.load_sales()
        self.load_stock_alerts()

    def create_database(self):
        """Create database and tables if they don't exist"""
        self.conn = sqlite3.connect('inventory.db')
        self.cursor = self.conn.cursor()

        # Products table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS products (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        name TEXT NOT NULL,
        description TEXT,
        price REAL NOT NULL,
        category TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
''')

# Inventory table
self.cursor.execute('''
    CREATE TABLE IF NOT EXISTS inventory (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        product_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL DEFAULT 0,
        min_stock_level INTEGER DEFAULT 5,
        last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (product_id) REFERENCES products (id)
    )
''')

# Sales table
self.cursor.execute('''
    CREATE TABLE IF NOT EXISTS sales (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        product_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
```

```
            sale_price REAL NOT NULL,
            total_amount REAL NOT NULL,
            sale_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (product_id) REFERENCES products (id)
        )
    ''')

    self.conn.commit()

def setup_products_tab(self):
    """Setup the products management tab"""
    # Left frame for form
    form_frame = ttk.LabelFrame(self.products_tab, text="Product Details", padding=10)
    form_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=5)

    # Right frame for product list
    list_frame = ttk.LabelFrame(self.products_tab, text="Product List", padding=10)
    list_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

    # Form fields
    ttk.Label(form_frame, text="Product Name:").grid(row=0, column=0, sticky=tk.W,
pady=5)
    self.name_entry = ttk.Entry(form_frame, width=25)
    self.name_entry.grid(row=0, column=1, pady=5, padx=5)

    ttk.Label(form_frame, text="Description:").grid(row=1, column=0, sticky=tk.W, pady=5)
```

```
        self.desc_entry = ttk.Entry(form_frame, width=25)
        self.desc_entry.grid(row=1, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Price:").grid(row=2, column=0, sticky=tk.W, pady=5)
        self.price_entry = ttk.Entry(form_frame, width=25)
        self.price_entry.grid(row=2, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Category:").grid(row=3, column=0, sticky=tk.W, pady=5)
        self.category_entry = ttk.Entry(form_frame, width=25)
        self.category_entry.grid(row=3, column=1, pady=5, padx=5)

        # Buttons
        btn_frame = ttk.Frame(form_frame)
        btn_frame.grid(row=4, column=0, columnspan=2, pady=10)

        ttk.Button(btn_frame, text="Add Product",
command=self.add_product).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Update Product",
command=self.update_product).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Delete Product",
command=self.delete_product).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Clear Form",
command=self.clear_product_form).pack(side=tk.LEFT, padx=5)

        # Product list
        columns = ('id', 'name', 'description', 'price', 'category')
        self.products_tree = ttk.Treeview(list_frame, columns=columns, show='headings')
```

```
        self.products_tree.heading('id', text='ID')
        self.products_tree.heading('name', text='Name')
        self.products_tree.heading('description', text='Description')
        self.products_tree.heading('price', text='Price')
        self.products_tree.heading('category', text='Category')

        self.products_tree.column('id', width=50)
        self.products_tree.column('name', width=150)
        self.products_tree.column('description', width=200)
        self.products_tree.column('price', width=100)
        self.products_tree.column('category', width=100)

        scrollbar = ttk.Scrollbar(list_frame, orient=tk.VERTICAL,
command=self.products_tree.yview)
        self.products_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        self.products_tree.pack(fill=tk.BOTH, expand=True)
        self.products_tree.bind('<<TreeviewSelect>>', self.on_product_select)

    def setup_sales_tab(self):
        """Setup the sales management tab"""
        # Left frame for form
        form_frame = ttk.LabelFrame(self.sales_tab, text="Record Sale", padding=10)
        form_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=5)
```

```
        # Right frame for sales list
        list_frame = ttk.LabelFrame(self.sales_tab, text="Sales History", padding=10)
        list_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Form fields
        ttk.Label(form_frame, text="Product:").grid(row=0, column=0, sticky=tk.W, pady=5)
        self.sale_product_var = tk.StringVar()
        self.sale_product_combo = ttk.Combobox(form_frame,
textvariable=self.sale_product_var, state='readonly')
        self.sale_product_combo.grid(row=0, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Quantity:").grid(row=1, column=0, sticky=tk.W, pady=5)
        self.sale_qty_entry = ttk.Entry(form_frame, width=25)
        self.sale_qty_entry.grid(row=1, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Sale Price:").grid(row=2, column=0, sticky=tk.W, pady=5)
        self.sale_price_entry = ttk.Entry(form_frame, width=25)
        self.sale_price_entry.grid(row=2, column=1, pady=5, padx=5)

        # Buttons
        btn_frame = ttk.Frame(form_frame)
        btn_frame.grid(row=3, column=0, columnspan=2, pady=10)

        ttk.Button(btn_frame, text="Record Sale",
command=self.record_sale).pack(side=tk.LEFT, padx=5)
```

```
        ttk.Button(btn_frame, text="Clear Form",
command=self.clear_sale_form).pack(side=tk.LEFT, padx=5)

        # Sales list
        columns = ('id', 'product', 'quantity', 'sale_price', 'total', 'date')
        self.sales_tree = ttk.Treeview(list_frame, columns=columns, show='headings')

        self.sales_tree.heading('id', text='ID')
        self.sales_tree.heading('product', text='Product')
        self.sales_tree.heading('quantity', text='Quantity')
        self.sales_tree.heading('sale_price', text='Sale Price')
        self.sales_tree.heading('total', text='Total Amount')
        self.sales_tree.heading('date', text='Sale Date')

        self.sales_tree.column('id', width=50)
        self.sales_tree.column('product', width=150)
        self.sales_tree.column('quantity', width=80)
        self.sales_tree.column('sale_price', width=100)
        self.sales_tree.column('total', width=100)
        self.sales_tree.column('date', width=150)

        scrollbar = ttk.Scrollbar(list_frame, orient=tk.VERTICAL,
command=self.sales_tree.yview)
        self.sales_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        self.sales_tree.pack(fill=tk.BOTH, expand=True)
```

```
    def setup_stock_tab(self):
        """Setup the stock management tab"""
        # Left frame for form
        form_frame = ttk.LabelFrame(self.stock_tab, text="Stock Management", padding=10)
        form_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=5)

        # Right frame for stock list
        list_frame = ttk.LabelFrame(self.stock_tab, text="Current Stock Levels", padding=10)
        list_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Form fields
        ttk.Label(form_frame, text="Product:").grid(row=0, column=0, sticky=tk.W, pady=5)
        self.stock_product_var = tk.StringVar()
        self.stock_product_combo = ttk.Combobox(form_frame,
textvariable=self.stock_product_var, state='readonly')
        self.stock_product_combo.grid(row=0, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Quantity:").grid(row=1, column=0, sticky=tk.W, pady=5)
        self.stock_qty_entry = ttk.Entry(form_frame, width=25)
        self.stock_qty_entry.grid(row=1, column=1, pady=5, padx=5)

        ttk.Label(form_frame, text="Min Stock Level:").grid(row=2, column=0, sticky=tk.W,
pady=5)
        self.min_stock_entry = ttk.Entry(form_frame, width=25)
        self.min_stock_entry.grid(row=2, column=1, pady=5, padx=5)
```

```
        # Buttons
        btn_frame = ttk.Frame(form_frame)
        btn_frame.grid(row=3, column=0, columnspan=2, pady=10)

        ttk.Button(btn_frame, text="Add Stock", command=self.add_stock).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Update Stock", command=self.update_stock).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Clear Form", command=self.clear_stock_form).pack(side=tk.LEFT, padx=5)

        # Stock list with alerts
        columns = ('product', 'quantity', 'min_level', 'status')
        self.stock_tree = ttk.Treeview(list_frame, columns=columns, show='headings')

        self.stock_tree.heading('product', text='Product')
        self.stock_tree.heading('quantity', text='Current Stock')
        self.stock_tree.heading('min_level', text='Min Stock Level')
        self.stock_tree.heading('status', text='Status')

        self.stock_tree.column('product', width=200)
        self.stock_tree.column('quantity', width=100)
        self.stock_tree.column('min_level', width=120)
        self.stock_tree.column('status', width=150)

        scrollbar = ttk.Scrollbar(list_frame, orient=tk.VERTICAL, command=self.stock_tree.yview)
```

```
        self.stock_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        self.stock_tree.pack(fill=tk.BOTH, expand=True)
        self.stock_tree.bind('<<TreeviewSelect>>', self.on_stock_select)

    def setup_reports_tab(self):
        """Setup the reports tab"""
        # Frame for report controls
        control_frame = ttk.Frame(self.reports_tab)
        control_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(control_frame, text="Sales Report",
command=self.generate_sales_report).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Stock Report",
command=self.generate_stock_report).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Low Stock Alert",
command=self.generate_low_stock_report).pack(side=tk.LEFT, padx=5)

        # Frame for report display
        report_frame = ttk.Frame(self.reports_tab)
        report_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.report_text = tk.Text(report_frame, wrap=tk.WORD, state=tk.DISABLED)
        scrollbar = ttk.Scrollbar(report_frame, orient=tk.VERTICAL,
command=self.report_text.yview)
        self.report_text.configure(yscrollcommand=scrollbar.set)
```

```
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.report_text.pack(fill=tk.BOTH, expand=True)

    def load_products(self):
        """Load products into the products tree and combo boxes"""
        # Clear existing data
        for item in self.products_tree.get_children():
            self.products_tree.delete(item)

        # Fetch products from database
        self.cursor.execute("SELECT * FROM products")
        products = self.cursor.fetchall()

        # Add products to treeview
        for product in products:
            self.products_tree.insert('', tk.END, values=product)

        # Update combo boxes
        product_names = [f"{p[0]} - {p[1]}" for p in products]
        self.sale_product_combo['values'] = product_names
        self.stock_product_combo['values'] = product_names

    def load_sales(self):
        """Load sales into the sales tree"""
        # Clear existing data
```

```
    for item in self.sales_tree.get_children():
        self.sales_tree.delete(item)

    # Fetch sales from database with product names
    self.cursor.execute('''
        SELECT s.id, p.name, s.quantity, s.sale_price, s.total_amount, s.sale_date
        FROM sales s
        JOIN products p ON s.product_id = p.id
        ORDER BY s.sale_date DESC
    ''')
    sales = self.cursor.fetchall()

    # Add sales to treeview
    for sale in sales:
        self.sales_tree.insert('', tk.END, values=sale)

def load_stock_alerts(self):
    """Load stock levels with alerts"""
    # Clear existing data
    for item in self.stock_tree.get_children():
        self.stock_tree.delete(item)

    # Fetch stock data with product names
    self.cursor.execute('''
        SELECT p.name, i.quantity, i.min_stock_level,
            CASE
```

```
            WHEN i.quantity <= 0 THEN 'Out of Stock'
            WHEN i.quantity <= i.min_stock_level THEN 'Low Stock'
            ELSE 'In Stock'
        END as status
    FROM inventory i
    JOIN products p ON i.product_id = p.id
    ''')
    stock_data = self.cursor.fetchall()

    # Add stock data to treeview with color coding
    for stock in stock_data:
        item_id = self.stock_tree.insert('', tk.END, values=stock)

        # Color code based on status
        if stock[2] is not None and stock[1] <= stock[2]:
            self.stock_tree.item(item_id, tags=('low',))
        elif stock[1] <= 0:
            self.stock_tree.item(item_id, tags=('out',))

    # Configure tags for coloring
    self.stock_tree.tag_configure('low', background='#fff9c4')  # Light yellow for low stock
    self.stock_tree.tag_configure('out', background='#ffcdd2')  # Light red for out of stock

def add_product(self):
    """Add a new product to the database"""
    name = self.name_entry.get().strip()
```

```
description = self.desc_entry.get().strip()
price = self.price_entry.get().strip()
category = self.category_entry.get().strip()

if not name or not price:
    messagebox.showerror("Error", "Product name and price are required!")
    return

try:
    price = float(price)
    if price <= 0:
        raise ValueError("Price must be positive")
except ValueError:
    messagebox.showerror("Error", "Please enter a valid price!")
    return

# Insert product into database
self.cursor.execute(
    "INSERT INTO products (name, description, price, category) VALUES (?, ?, ?, ?)",
    (name, description, price, category)
)
self.conn.commit()

# Create inventory entry for the new product
product_id = self.cursor.lastrowid
self.cursor.execute(
```

```
            "INSERT INTO inventory (product_id, quantity, min_stock_level) VALUES (?, 0, 5)",
            (product_id,)
        )
        self.conn.commit()

        messagebox.showinfo("Success", "Product added successfully!")
        self.clear_product_form()
        self.load_products()
        self.load_stock_alerts()

    def update_product(self):
        """Update selected product"""
        selected = self.products_tree.selection()
        if not selected:
            messagebox.showerror("Error", "Please select a product to update!")
            return

        item = self.products_tree.item(selected[0])
        product_id = item['values'][0]

        name = self.name_entry.get().strip()
        description = self.desc_entry.get().strip()
        price = self.price_entry.get().strip()
        category = self.category_entry.get().strip()

        if not name or not price:
```

```
            messagebox.showerror("Error", "Product name and price are required!")
            return

        try:
            price = float(price)
            if price <= 0:
                raise ValueError("Price must be positive")
        except ValueError:
            messagebox.showerror("Error", "Please enter a valid price!")
            return

        # Update product in database
        self.cursor.execute(
            "UPDATE products SET name=?, description=?, price=?, category=? WHERE id=?",
            (name, description, price, category, product_id)
        )
        self.conn.commit()

        messagebox.showinfo("Success", "Product updated successfully!")
        self.load_products()

    def delete_product(self):
        """Delete selected product"""
        selected = self.products_tree.selection()
        if not selected:
            messagebox.showerror("Error", "Please select a product to delete!")
```

```
            return

        if not messagebox.askyesno("Confirm", "Are you sure you want to delete this
product?"):
            return

        item = self.products_tree.item(selected[0])
        product_id = item['values'][0]

        # Delete product from database
        self.cursor.execute("DELETE FROM products WHERE id=?", (product_id,))
        self.cursor.execute("DELETE FROM inventory WHERE product_id=?", (product_id,))
        self.cursor.execute("DELETE FROM sales WHERE product_id=?", (product_id,))
        self.conn.commit()

        messagebox.showinfo("Success", "Product deleted successfully!")
        self.clear_product_form()
        self.load_products()
        self.load_sales()
        self.load_stock_alerts()

    def record_sale(self):
        """Record a new sale"""
        product_selection = self.sale_product_var.get()
        if not product_selection:
            messagebox.showerror("Error", "Please select a product!")
```

```
            return

        quantity = self.sale_qty_entry.get().strip()
        sale_price = self.sale_price_entry.get().strip()

        if not quantity or not sale_price:
            messagebox.showerror("Error", "Quantity and sale price are required!")
            return

        try:
            quantity = int(quantity)
            sale_price = float(sale_price)
            if quantity <= 0 or sale_price <= 0:
                raise ValueError("Values must be positive")
        except ValueError:
            messagebox.showerror("Error", "Please enter valid quantity and price!")
            return

        # Extract product ID from selection
        product_id = int(product_selection.split(' - ')[0])

        # Check if enough stock is available
        self.cursor.execute("SELECT quantity FROM inventory WHERE product_id=?",
(product_id,))
        stock = self.cursor.fetchone()
```

```
        if not stock or stock[0] < quantity:
            messagebox.showerror("Error", "Not enough stock available!")
            return

        # Calculate total amount
        total_amount = quantity * sale_price

        # Record sale
        self.cursor.execute(
            "INSERT INTO sales (product_id, quantity, sale_price, total_amount) VALUES (?, ?, ?,
?)",
            (product_id, quantity, sale_price, total_amount)
        )

        # Update inventory
        self.cursor.execute(
            "UPDATE inventory SET quantity = quantity - ? WHERE product_id=?",
            (quantity, product_id)
        )

        self.conn.commit()

        messagebox.showinfo("Success", "Sale recorded successfully!")
        self.clear_sale_form()
        self.load_sales()
        self.load_stock_alerts()
```

```
def add_stock(self):
    """Add stock for a product"""
    product_selection = self.stock_product_var.get()
    if not product_selection:
        messagebox.showerror("Error", "Please select a product!")
        return

    quantity = self.stock_qty_entry.get().strip()
    min_stock = self.min_stock_entry.get().strip()

    if not quantity:
        messagebox.showerror("Error", "Quantity is required!")
        return

    try:
        quantity = int(quantity)
        if min_stock:
            min_stock = int(min_stock)
        else:
            min_stock = 5  # Default value

        if quantity < 0 or min_stock < 0:
            raise ValueError("Values must not be negative")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers!")
```

```
            return

        # Extract product ID from selection
        product_id = int(product_selection.split(' - ')[0])

        # Check if inventory entry exists
        self.cursor.execute("SELECT id FROM inventory WHERE product_id=?", (product_id,))
        inventory = self.cursor.fetchone()

        if inventory:
            # Update existing inventory
            self.cursor.execute(
                "UPDATE inventory SET quantity = quantity + ?, min_stock_level = ? WHERE
product_id=?",
                (quantity, min_stock, product_id)
            )
        else:
            # Create new inventory entry
            self.cursor.execute(
                "INSERT INTO inventory (product_id, quantity, min_stock_level) VALUES (?, ?, ?)",
                (product_id, quantity, min_stock)
            )

        self.conn.commit()

        messagebox.showinfo("Success", "Stock updated successfully!")
```

```
    self.clear_stock_form()
    self.load_stock_alerts()

def update_stock(self):
    """Update stock level for a product"""
    product_selection = self.stock_product_var.get()
    if not product_selection:
        messagebox.showerror("Error", "Please select a product!")
        return

    quantity = self.stock_qty_entry.get().strip()
    min_stock = self.min_stock_entry.get().strip()

    if not quantity:
        messagebox.showerror("Error", "Quantity is required!")
        return

    try:
        quantity = int(quantity)
        if min_stock:
            min_stock = int(min_stock)
        else:
            min_stock = 5  # Default value

        if quantity < 0 or min_stock < 0:
            raise ValueError("Values must not be negative")
```

```
        except ValueError:
            messagebox.showerror("Error", "Please enter valid numbers!")
            return

        # Extract product ID from selection
        product_id = int(product_selection.split(' - ')[0])

        # Update inventory
        self.cursor.execute(
            "UPDATE inventory SET quantity = ?, min_stock_level = ? WHERE product_id=?",
            (quantity, min_stock, product_id)
        )

        self.conn.commit()

        messagebox.showinfo("Success", "Stock updated successfully!")
        self.clear_stock_form()
        self.load_stock_alerts()

    def generate_sales_report(self):
        """Generate sales report"""
        self.cursor.execute('''
            SELECT
                p.name as product,
                SUM(s.quantity) as total_quantity,
                SUM(s.total_amount) as total_revenue,
```

```
            AVG(s.sale_price) as avg_price
        FROM sales s
        JOIN products p ON s.product_id = p.id
        GROUP BY p.name
        ORDER BY total_revenue DESC
    ''')
    sales_data = self.cursor.fetchall()

    self.display_report("Sales Report", ["Product", "Total Quantity", "Total Revenue", "Avg Price"], sales_data)

  def generate_stock_report(self):
    """Generate stock report"""
    self.cursor.execute('''
        SELECT
            p.name as product,
            i.quantity as stock,
            i.min_stock_level as min_level,
            CASE
                WHEN i.quantity <= 0 THEN 'Out of Stock'
                WHEN i.quantity <= i.min_stock_level THEN 'Low Stock'
                ELSE 'In Stock'
            END as status
        FROM inventory i
        JOIN products p ON i.product_id = p.id
        ORDER BY p.name
```

```
        ''')
        stock_data = self.cursor.fetchall()

        self.display_report("Stock Report", ["Product", "Current Stock", "Min Level", "Status"],
stock_data)

    def generate_low_stock_report(self):
        """Generate low stock alert report"""
        self.cursor.execute('''
            SELECT
                p.name as product,
                i.quantity as stock,
                i.min_stock_level as min_level,
                (i.min_stock_level - i.quantity) as needed
            FROM inventory i
            JOIN products p ON i.product_id = p.id
            WHERE i.quantity <= i.min_stock_level
            ORDER BY needed DESC
        ''')
        low_stock_data = self.cursor.fetchall()

        self.display_report("Low Stock Alert", ["Product", "Current Stock", "Min Level",
"Needed"], low_stock_data)

    def display_report(self, title, headers, data):
        """Display report in the report text area"""
        self.report_text.config(state=tk.NORMAL)
```

```
        self.report_text.delete(1.0, tk.END)

        # Add title
        self.report_text.insert(tk.END, f"{title}\n", 'title')
        self.report_text.insert(tk.END, "=" * 50 + "\n\n")

        # Add headers
        header_line = " | ".join(headers) + "\n"
        self.report_text.insert(tk.END, header_line, 'header')
        self.report_text.insert(tk.END, "-" * len(header_line) + "\n")

        # Add data
        for row in data:
            line = " | ".join(str(item) for item in row) + "\n"
            self.report_text.insert(tk.END, line)

        # Add summary
        self.report_text.insert(tk.END, f"\nTotal Records: {len(data)}\n")
        self.report_text.insert(tk.END, f"Generated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

        # Configure tags for formatting
        self.report_text.tag_configure('title', font=('Arial', 14, 'bold'))
        self.report_text.tag_configure('header', font=('Arial', 10, 'bold'))

        self.report_text.config(state=tk.DISABLED)
```

```
def on_product_select(self, event):
    """Handle product selection"""
    selected = self.products_tree.selection()
    if not selected:
        return

    item = self.products_tree.item(selected[0])
    values = item['values']

    self.name_entry.delete(0, tk.END)
    self.name_entry.insert(0, values[1])

    self.desc_entry.delete(0, tk.END)
    self.desc_entry.insert(0, values[2] if values[2] else "")

    self.price_entry.delete(0, tk.END)
    self.price_entry.insert(0, values[3])

    self.category_entry.delete(0, tk.END)
    self.category_entry.insert(0, values[4] if values[4] else "")

def on_stock_select(self, event):
    """Handle stock selection"""
    selected = self.stock_tree.selection()
    if not selected:
```

```
        return

    item = self.stock_tree.item(selected[0])
    product_name = item['values'][0]

    # Find the product ID from the name
    self.cursor.execute("SELECT id FROM products WHERE name=?", (product_name,))
    product = self.cursor.fetchone()

    if product:
        product_id = product[0]
        # Set the combo box to this product
        self.stock_product_var.set(f"{product_id} - {product_name}")

        # Get current stock values
        self.cursor.execute("SELECT quantity, min_stock_level FROM inventory WHERE
product_id=?", (product_id,))
        stock = self.cursor.fetchone()

        if stock:
            self.stock_qty_entry.delete(0, tk.END)
            self.stock_qty_entry.insert(0, stock[0])

            self.min_stock_entry.delete(0, tk.END)
            self.min_stock_entry.insert(0, stock[1])
```

```
    def clear_product_form(self):
        """Clear the product form"""
        self.name_entry.delete(0, tk.END)
        self.desc_entry.delete(0, tk.END)
        self.price_entry.delete(0, tk.END)
        self.category_entry.delete(0, tk.END)
        self.products_tree.selection_remove(self.products_tree.selection())

    def clear_sale_form(self):
        """Clear the sale form"""
        self.sale_product_var.set('')
        self.sale_qty_entry.delete(0, tk.END)
        self.sale_price_entry.delete(0, tk.END)

    def clear_stock_form(self):
        """Clear the stock form"""
        self.stock_product_var.set('')
        self.stock_qty_entry.delete(0, tk.END)
        self.min_stock_entry.delete(0, tk.END)
        self.stock_tree.selection_remove(self.stock_tree.selection())

if __name__ == "__main__":
    root = tk.Tk()
    app = InventoryManagementSystem(root)
    root.mainloop()
```

Features Implemented

This Inventory Management System includes:

1. Product Management
   · Add, update, and delete products
   · Track product details (name, description, price, category)
2. Sales Management
   · Record sales transactions
   · Track sale date, quantity, and price
   · View sales history
3. Stock Management
   · Add and update stock levels
   · Set minimum stock levels
   · Visual alerts for low stock items
4. Reporting
   · Sales reports with revenue analysis
   · Stock level reports
   · Low stock alerts
5. Database Integration
   · SQLite database for data persistence
   · Proper relationships between products, inventory, and sales