```
import tkinter
from tkinter import *
from tkinter import messagebox, ttk
import random
from PIL import Image, ImageTk
from datetime import datetime
import tkinter as tk
import mysql.connector

window = tkinter.Tk()
window.configure(background="#ffe4c4")
window.title("Stock Management System")
window.geometry("1200x600")

screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

window.attributes('-fullscreen', True)
my_tree = ttk.Treeview(window,show='headings',height=20)
style= ttk.Style()

style.configure("my_tree.heading", bg="gray", foreground="black",
font=("Helvetica", 10, "bold"))
style.configure("heading", background="gray", foreground="black")
style.map('Treeview', background=[('selected', 'blue'), ('!selected',
'khaki')])

placeholderArray = ['','','','','']
numeric= '1234567890'
alpha='ABCDEFGHIJKLMNOPQRSTUVWXYZ'


conn = mysql.connector.connect(
       host="localhost",
       user="root",         password="",
       database="delispread_sms"
   )
cursor = conn.cursor()


for i in range(0,5):
   placeholderArray[i] = tkinter.StringVar()

def read():
   try:
       conn.ping(reconnect=True, attempts=3, delay=0)
       sql = "SELECT item_id, name, price, quantity, category, date FROM stocks
ORDER BY id DESC"
       cursor.execute(sql)
```

```
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        messagebox.showerror("Error", f"Error reading data from database: {e}")
        return []


def refreshTable():
    try:
        for item in my_tree.get_children():
            my_tree.delete(item)
        rows = read()
        for i, row in enumerate(rows):
            if i % 2 == 0:
                my_tree.insert('', 'end', values=row, tags=('evenrow',))
            else:
                my_tree.insert('', 'end', values=row, tags=('oddrow',))
    except mysql.connector.Error as e:
        messagebox.showerror("Error", f"Error fetching data from database: {e}")


def setph(word,num):
    for ph in range(0,5):
        if ph == num:
            placeholderArray[ph].set(word)

def generateRand():
    itemId=''
    for i in range(0,3):
        randno=random.randrange(0,(len(numeric)-1))
        itemId = itemId+str(numeric[randno])
    randno=random.randrange(0,(len(alpha)-1))
    itemId= itemId+'-' +str(alpha[randno])
    print("generated: "+itemId)
    setph(itemId,0)


def save():
    item_id = itemIdEntry.get()
    name = nameEntry.get()
    price = float(priceEntry.get())
    quantity = int(qntEntry.get())
    category = categoryCombo.get()

    sql = "INSERT INTO stocks (item_id, name, price, quantity, category, date)
VALUES (%s, %s, %s, %s, %s, NOW())"
    values = (item_id, name, price, quantity, category)

    try:
```

```
        cursor.execute(sql, values)
        conn.commit()
        refreshTable()
        clear()
        messagebox.showinfo("Success", "Record inserted successfully!")
    except Exception as e:
        conn.rollback()
        messagebox.showerror("Error", f"Error inserting record: {e}")

def update():
    try:
        selectedItem = my_tree.selection()[0]
        selectedItemId = str(my_tree.item(selectedItem)['values'][0])
    except IndexError:
        messagebox.showwarning("", "Please select a data row")
        return

    item_id = itemIdEntry.get()
    if selectedItemId != item_id:
        messagebox.showwarning("", "You can't change Item ID")
        return

    name = nameEntry.get()
    price = priceEntry.get()
    quantity = qntEntry.get()
    category = categoryCombo.get()

    if not all([item_id, name, price, quantity, category]):
        messagebox.showwarning("", "Please fill up all entries")
        return

    sql = "UPDATE stocks SET name=%s, price=%s, quantity=%s, category=%s,
date=%s WHERE item_id=%s"
    values = (name, price, quantity, category, datetime.now().strftime('%Y-%m-%d
%H:%M:%S'), item_id)

    try:
        cursor.execute(sql, values)
        conn.commit()
        refreshTable()
        clear()
        messagebox.showinfo("Success", "Record updated successfully!")
    except mysql.connector.Error as err:
        conn.rollback()
        messagebox.showwarning("", f"Error occurred: {str(err)}")



def delete():
```

```
    try:
        selectedItem = my_tree.selection()[0]
        itemId = str(my_tree.item(selectedItem)['values'][0])
    except IndexError:
        messagebox.showwarning("", "Please select a data row")
        return

    decision = messagebox.askquestion("", "Delete the selected data?")
    if decision != 'yes':
        return

    sql = "DELETE FROM stocks WHERE item_id=%s"
    try:
        cursor.execute(sql, (itemId,))
        conn.commit()
        messagebox.showinfo("", "Data has been successfully deleted")
        refreshTable()
    except mysql.connector.Error as e:
        conn.rollback()
        messagebox.showinfo("", f"Sorry, an error occurred: {e}")


def select():
    try:
        selected_item = my_tree.focus()
        values = my_tree.item(selected_item, 'values')

        if values:
            itemIdEntry.delete(0, tk.END)
            itemIdEntry.insert(0, values[0])

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

            priceEntry.delete(0, tk.END)
            priceEntry.insert(0, values[2])

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

            categoryCombo.set(values[4])
        else:
            messagebox.showinfo("Select", "Please select an item from the list")
    except Exception as e:
        messagebox.showerror("Error", f"Error selecting item: {e}")


def find():
    itemId = str(itemIdEntry.get()).strip()
```

```
    name = str(nameEntry.get()).strip()
    price = str(priceEntry.get()).strip()
    qnt = str(qntEntry.get()).strip()
    cat = str(categoryCombo.get()).strip()

    conn.ping(reconnect=True, attempts=3, delay=0)

    sql = "SELECT `item_id`, `name`, `price`, `quantity`, `category`, `date`
FROM stocks WHERE"
    params = []
    conditions = []


    if itemId:
        conditions.append("`item_id` LIKE %s")
        params.append(f"%{itemId}%")
    if name:
        conditions.append("`name` LIKE %s")
        params.append(f"%{name}%")
    if price:
        conditions.append("`price` LIKE %s")
        params.append(f"%{price}%")
    if qnt:
        conditions.append("`quantity` LIKE %s")
        params.append(f"%{qnt}%")
    if cat:
        conditions.append("`category` LIKE %s")
        params.append(f"%{cat}%")

    if not conditions:
        messagebox.showwarning("", "Please fill up one of the entries")
        return
    sql += " OR ".join(conditions)
    try:
        cursor.execute(sql, params)
        result = cursor.fetchall()
        clear()
        if result:
            for num in range(min(5, len(result[0]))):
                setph(result[0][num], num)
        else:
            messagebox.showwarning("", "No data found")
    except Exception as e:
        messagebox.showwarning("", f"Error occurred: {str(e)}")
    finally:
        conn.commit()
        conn.close()
```

```
def clear():
   itemIdEntry.delete(0, tk.END)
   nameEntry.delete(0, tk.END)
   priceEntry.delete(0, tk.END)
   qntEntry.delete(0, tk.END)
   categoryCombo.set("")

def refreshComboBoxes():
   try:
       cursor.execute("SELECT DISTINCT name FROM stocks ORDER BY name")
       names = [name[0] for name in cursor.fetchall()]
       nameCombo['values'] = names
       print(f"Fetched names: {names}")  # Debug statement
   except mysql.connector.Error as e:
       messagebox.showerror("Error", f"Error fetching names from database:
{e}")


def updateCategoryCombo(event):
   selected_name = nameCombo.get()
   try:
       cursor.execute("SELECT DISTINCT category FROM stocks WHERE name = %s
ORDER BY category", (selected_name,))
       categories = [category[0] for category in cursor.fetchall()]
       catCombo['values'] = categories
       if categories:
           catCombo.current(0)
       print(f"Fetched categories for {selected_name}: {categories}")
   except mysql.connector.Error as e:
       messagebox.showerror("Error", f"Error fetching categories: {e}")

def clearAddForm():
   nameCombo.set('')
   catCombo.set('')
   quantityEntry.delete(0, 'end')
def add_quantity():
   selected_name = nameCombo.get()
   selected_category = catCombo.get()
   quantity_str = quantityEntry.get().strip()

   if quantity_str == "":
       messagebox.showerror("Error", "Please enter a quantity.")
       return

   try:
       quantity = int(quantity_str)
   except ValueError:
       messagebox.showerror("Error", "Quantity must be a valid integer.")
       return
```

```
    try:
        cursor.execute("SELECT quantity FROM stocks WHERE name = %s AND category
= %s",
                       (selected_name, selected_category))
        current_quantity = cursor.fetchone()[0]
        current_quantity = int(current_quantity)
        new_quantity = current_quantity + quantity
        cursor.execute("UPDATE stocks SET quantity = %s WHERE name = %s AND
category = %s",
                       (new_quantity, selected_name, selected_category))
        conn.commit()

        messagebox.showinfo("Add Quantity",
                            f"Added {quantity} units for {selected_name}
({selected_category}). New quantity: {new_quantity}")
    except mysql.connector.Error as e:
        messagebox.showerror("Error", f"Error updating quantity: {e}")

    refreshComboBoxes()
    refreshTable()
    clearAddForm()

def clearDeductForm():
    nameCombo.set('')
    catCombo.set('')
    quantityEntry.delete(0, 'end')


def deduct_quantity():
    selected_name = nameCombo.get()
    selected_category = catCombo.get()
    quantity_str = quantityEntry.get().strip()

    if quantity_str == "":
        messagebox.showerror("Error", "Please enter a quantity.")
        return

    try:
        quantity = int(quantity_str)
    except ValueError:
        messagebox.showerror("Error", "Quantity must be a valid integer.")
        return




    try:
```

```
        cursor.execute("SELECT quantity FROM stocks WHERE name = %s AND category
= %s",
                       (selected_name, selected_category))
        current_quantity = cursor.fetchone()[0]
        current_quantity = int(current_quantity)

        if current_quantity >= quantity:
            new_quantity = current_quantity - quantity
            cursor.execute("UPDATE stocks SET quantity = %s WHERE name = %s AND
category = %s",
                           (new_quantity, selected_name, selected_category))
            conn.commit()

            messagebox.showinfo("Deduct Quantity",
                                f"Deducted {quantity} units for {selected_name}
({selected_category}). New quantity: {new_quantity}")
        else:
            messagebox.showwarning("Error", f"Insufficient quantity available
for deduction.")
    except mysql.connector.Error as e:
        messagebox.showerror("Error", f"Error updating quantity: {e}")

    refreshComboBoxes()
    refreshTable()
    clearDeductForm()
def search():
    try:
        for item in my_tree.get_children():
            my_tree.delete(item)
        search_term = searchEntry.get()

        sql = "SELECT item_id, name, price, quantity, category, date FROM stocks
WHERE name LIKE %s "

        cursor.execute(sql, (f'%{search_term}%',))
        rows = cursor.fetchall()

        for index, row in enumerate(rows):
            tag = 'evenrow' if index % 2 == 0 else 'oddrow'
            my_tree.insert('', 'end', values=row, tags=(tag,))

        conn.commit()
    except mysql.connector.Error as e:
        messagebox.showerror("Error", f"Error searching data: {e}")
```

```
def clear_treeview():
   for item in my_tree.get_children():
       my_tree.delete(item)


def clear_search_fields():
   itemIdEntry.delete(0, tk.END)
   nameEntry.delete(0, tk.END)
   priceEntry.delete(0, tk.END)
   qntEntry.delete(0, tk.END)
   categoryCombo.set("")


frame = tkinter.Frame(window, bg="#856d4d")
frame.grid(row=0, column=0, columnspan=2, pady=0, sticky='ew')

image_path = "240113965_108035594941525_1820501856483688060_n.jpg"
img = Image.open(image_path)
img = img.resize((150, 100), Image.LANCZOS)
photo = ImageTk.PhotoImage(img)
img_label = tk.Label(frame, image=photo, bg="#856d4d")
img_label.grid(row=0, column=0, padx=10, pady=10)

title_label = tkinter.Label(frame, text="           Deli Spread Stock
Management System", font=("Bold Serif", 30, "bold"), fg="white", bg="#856d4d")
title_label.grid(row=0, column=1, padx=5, pady=(5, 10))

btnColor = "#196E78"
def exit_application():
   if messagebox.askokcancel("Exit", "Do you want to exit?"):
       window.destroy()
exit_icon = Image.open("exit-removebg-preview.png")
exit_icon = exit_icon.resize((62, 52), Image.LANCZOS)
exit_icon = ImageTk.PhotoImage(exit_icon)

exit_label = tk.Label(window, image=exit_icon, cursor="hand2",bg="#856d4d")
exit_label.place(x=1200,y=10)
exit_label.bind("<Button-1>", lambda e: exit_application())

#-------------------MANAGE-------------------------------------------------------
manageFrameBtns = tkinter.LabelFrame(window,
text="Manage",font=('Roboto',12,'bold'),
padx=10,pady=10,borderwidth=5,bg="#ffe4c4")
manageFrameBtns.place( x=10,y=160,width=575,height=110)
saveBtn = Button(manageFrameBtns, text="SAVE", width=11,
borderwidth=3,font=('Roboto',10,'bold'), bg="#008000", fg='white',
command=save)
```

```
updateBtn = Button(manageFrameBtns, text="UPDATE", width=11,
borderwidth=3,font=('Roboto',10,'bold'), bg="#0000ff", fg='white',
command=update)
deleteBtn = Button(manageFrameBtns, text="DELETE", width=11,
borderwidth=3,font=('Roboto',10,'bold'), bg="#ff0000", fg='white',
command=delete)
selectBtn = Button(manageFrameBtns, text="SELECT", width=11,
borderwidth=3,font=('Roboto',10,'bold'), bg="#008080", fg='white',
command=select)
clearBtn = Button(manageFrameBtns, text="CLEAR", width=11,
borderwidth=3,font=('Roboto',10,'bold'), bg="#d3d3d3", fg='white',
command=clear)

saveBtn.grid(row=1,column=0,padx=5,pady=5)
updateBtn.grid(row=1,column=1,padx=5,pady=5)
deleteBtn.grid(row=1,column=2,padx=5,pady=5)
selectBtn.grid(row=1,column=3,padx=5,pady=5)
clearBtn.grid(row=1,column=4,padx=5,pady=5)

#---------------STOCK CONTROL FRAME--------------------------
stock_controlFrame = tkinter.LabelFrame(window, text="Stock
Control",font=('Roboto',12,'bold'), padx=10,pady=10,borderwidth=5,bg="#ffe4c4")
stock_controlFrame.place( x=10,y=520)
inBtn = Button(stock_controlFrame, text="IN PRODUCT",width=10,
borderwidth=3,bg="#000080", fg='white' , command=add_quantity)
outBtn = Button(stock_controlFrame, text=" OUT PRODUCT ",width=11,
borderwidth=3, bg="#ff7518",fg='white' ,command= deduct_quantity)
inBtn.grid(row=1, column=3, padx=5)
outBtn.grid(row=3, column=3, padx=5)
nameLabel=
Label(stock_controlFrame,text="NAME",anchor="e",width=10,bg="#ffe4c4")
categoryLabel=
Label(stock_controlFrame,text="CATEGORY",anchor="e",width=10,bg="#ffe4c4")
quantityLabel=
Label(stock_controlFrame,text="QUANTITY",anchor="e",width=10,bg="#ffe4c4")
nameLabel.grid(row=1,column=0,padx=10)
quantityLabel.grid(row=3,column=0,padx=10)
categoryLabel.grid(row=2,column=0,padx=10)
nameCombo= ttk.Combobox(stock_controlFrame,width=50)
nameCombo.grid(row=1,column=2,padx=5,pady=5)
nameCombo.bind("<<ComboboxSelected>>", updateCategoryCombo)
catCombo= ttk.Combobox(stock_controlFrame,width=50)
catCombo.grid(row=2,column=2,padx=5,pady=5)
quantityEntry= Entry(stock_controlFrame,width=50)
quantityEntry.grid(row=3,column=2,padx=5,pady=5)
refreshComboBoxes()
```

```
#-----------------ENTRIES FRAME-----------------------------------------
entriesFrame = tk.LabelFrame(window, text="Form",font=('Roboto',12,'bold'),
padx=10, pady=10, bg="#ffe4c4", borderwidth=5)
entriesFrame.place( x=10,y=300,width=550,height=200)

itemIdLabel= Label(entriesFrame,text="ITEM
ID",anchor="e",width=10,bg="#ffe4c4")
nameLabel= Label(entriesFrame,text="NAME",anchor="e",width=10,bg="#ffe4c4")
priceLabel= Label(entriesFrame,text="PRICE",anchor="e",width=10,bg="#ffe4c4")
qntLabel= Label(entriesFrame,text="QUANTITY",anchor="e",width=10,bg="#ffe4c4")
categoryLabel=
Label(entriesFrame,text="CATEGORY",anchor="e",width=10,bg="#ffe4c4")

itemIdLabel.grid(row=0,column=0,padx=10)
nameLabel.grid(row=1,column=0,padx=10)
priceLabel.grid(row=2,column=0,padx=10)
qntLabel.grid(row=3,column=0,padx=10)
categoryLabel.grid(row=4,column=0,padx=10)

categoryArray=['260 grams','380 grams','560 grams']

itemIdEntry= Entry(entriesFrame,width=50,textvariable=placeholderArray[0])
nameEntry= Entry(entriesFrame,width=50,textvariable=placeholderArray[1])
priceEntry= Entry(entriesFrame,width=50,textvariable=placeholderArray[2])
qntEntry= Entry(entriesFrame,width=50,textvariable=placeholderArray[3])
categoryCombo=
ttk.Combobox(entriesFrame,width=50,textvariable=placeholderArray[4],values=cate
goryArray)

itemIdEntry.grid(row=0,column=2,padx=5,pady=5)
nameEntry.grid(row=1,column=2,padx=5,pady=5)
priceEntry.grid(row=2,column=2,padx=5,pady=5)
qntEntry.grid(row=3,column=2,padx=5,pady=5)
categoryCombo.grid(row=4,column=2,padx=5,pady=5)

generateIdBtn= Button(entriesFrame,text="GENERATE
ID",borderwidth=3,bg="#800080",fg='white',command=generateRand)
generateIdBtn.grid(row=0,column=3,padx=5,pady=5)

#---------------------TREE FRAME----------------------------------
treeFrame = tk.LabelFrame(window,text="Stock
Records",font=('Roboto',12,'bold'),padx=10, pady=10,bg="#ffe4c4",borderwidth=5)
treeFrame.place(x=600,y=160,width=650,height=520)

searchFrame = tk.Frame(treeFrame,bg="#ffe4c4")
searchFrame.grid(row=0, column=0, padx=5, pady=6, sticky='ne')
searchEntry = tk.Entry(searchFrame)
searchEntry.grid(row=0, column=0, padx=5, pady=5)
```

```
searchBtn = Button(searchFrame, text="Search",width=10, borderwidth=3,
bg="#996515", fg='white', command=search)
searchBtn.grid(row=0, column=1, padx=5, pady=5)

show_all_button = tk.Button(treeFrame, text="Show All", width=10,
borderwidth=3, bg="#ffa500", fg='white', command=refreshTable)
show_all_button.grid(row=0, column=0, padx=5, pady=5, sticky='w')


my_tree = ttk.Treeview(treeFrame)
my_tree.grid(row=1, column=0, columnspan=2,sticky='nsew')
my_tree['columns'] = ("Item Id", "Name", "Price", "Quantity", "Category",
"Date")
my_tree.column("#0", width=0, stretch=NO)
my_tree.column("Item Id", anchor=W, width=60)
my_tree.column("Name", anchor=W, width=100)
my_tree.column("Price", anchor=W, width=80)
my_tree.column("Quantity", anchor=W, width=80)
my_tree.column("Category", anchor=W, width=80)
my_tree.column("Date", anchor=W, width=120)


my_tree.heading("Item Id", text=" Item Id", anchor=W)
my_tree.heading("Name", text="Name", anchor=W)
my_tree.heading("Price", text="Price", anchor=W)
my_tree.heading("Quantity", text="Quantity", anchor=W)
my_tree.heading("Category", text="Category", anchor=W)
my_tree.heading("Date", text="Date", anchor=W)

refreshTable()

window.grid_columnconfigure(1, weight=1)
window.grid_rowconfigure(2, weight=1)
treeFrame.grid_columnconfigure(0, weight=1)
treeFrame.grid_rowconfigure(1, weight=1)


window.mainloop()
cursor.close()
conn.close()
```