TRƯỜNG ĐẠI HỌC VĂN LANG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP 2

MÔN HỌC LẬP TRÌNH PYTHON NÂNG CAO

Đề bài:

LẬP TRÌNH GIAO DIỆN GUI CÓ KẾT NỐI DATABASE

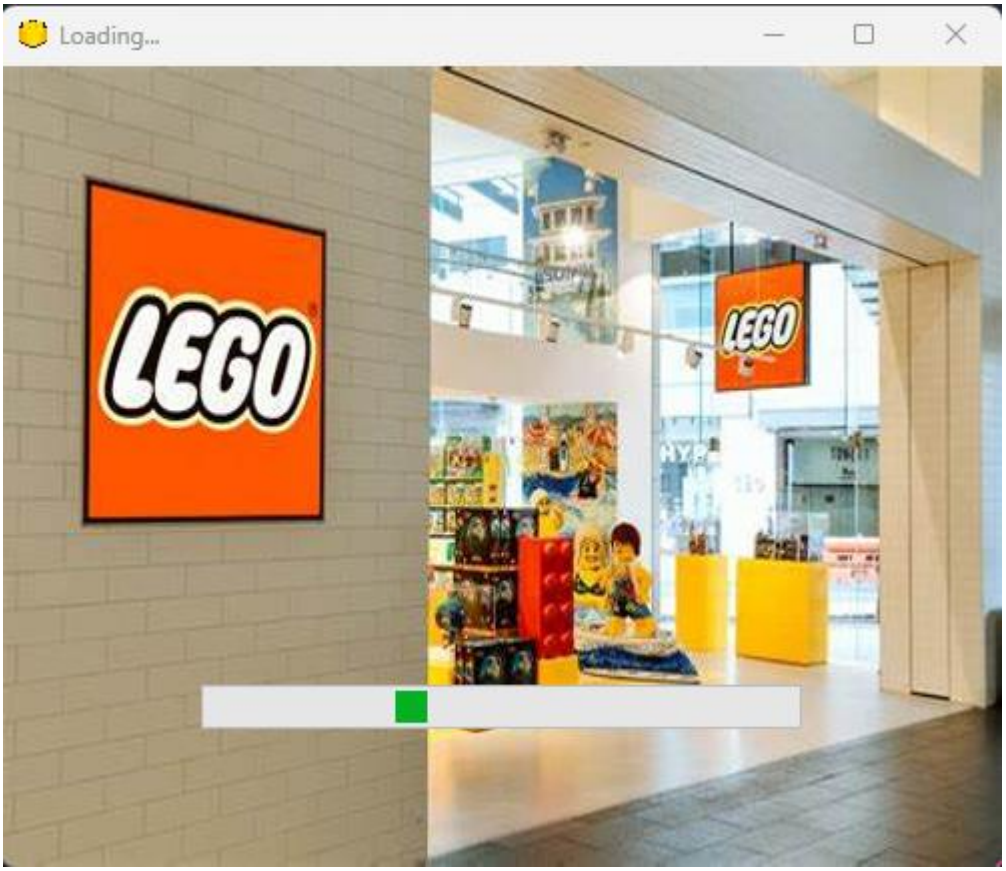
SINH VIÊN THỰC HIỆN:
NGUYỄN TIẾN PHÚC 2274802010685
LỚP: 241_71ITSE31003_02

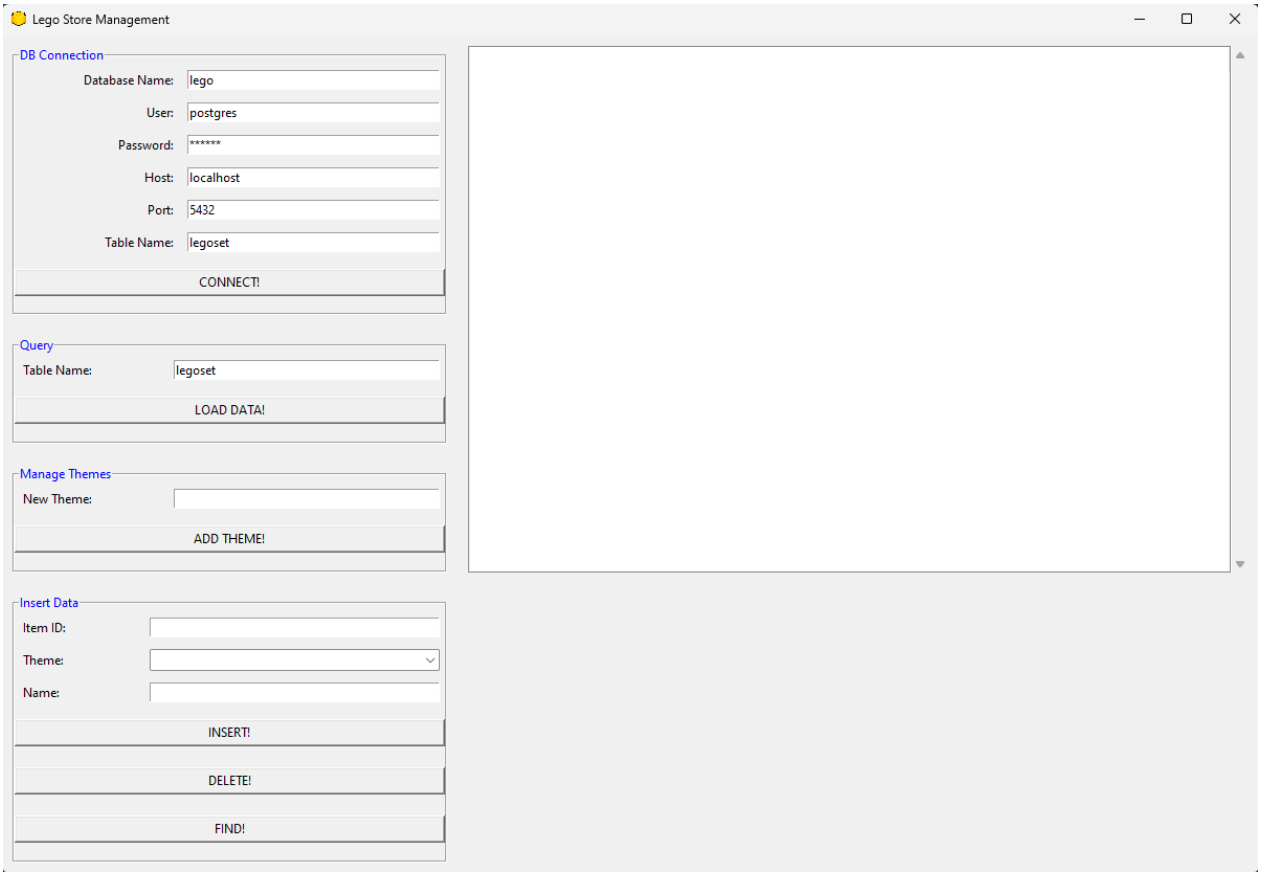
GVHD: HUỲNH THÁI HỌC

TP. Hồ Chí Minh – 18/10/2024

MỤC LỤC

I. GIAO DIỆN

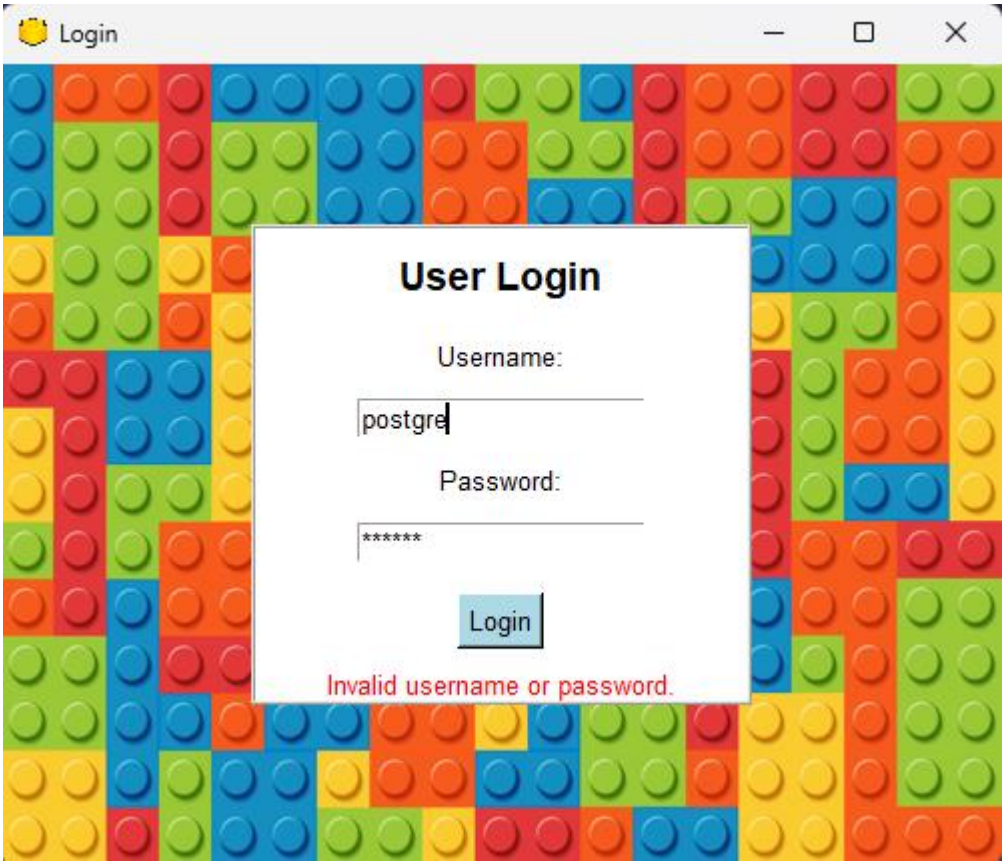

Loading...
LEGO
LEGO


Lego Store Management
DB Connection
Database Name: lego
User: postgres
Password: ******
Host: localhost
Port: 5432
Table Name: legoset
CONNECT!
Query
Table Name: legoset
LOAD DATA!
Manage Themes
New Theme:
ADD THEME!
Insert Data
Item ID:
Theme:
Name:
INSERT!
DELETE!
FIND!

II. CHỨC NĂNG

1. Login

Chức năng Login yêu cầu người dùng đăng nhập trước khi vào được trang ứng dụng chính. Nếu người dùng nhập username hoặc password không hợp lệ thì sẽ hiện dòng chữ màu đỏ nhắc nhở.



Sau khi người dùng nhập đúng username và password thì trang đăng nhập sẽ đóng và sẽ dẫn đến trang tiếp theo là LoadingScreen.

Toàn bộ code để làm giao diện Login nằm trong class LoginApp:

```
class LoginApp:
    def __init__(self, win):
        self.win = win
        self.win.title("Login")


        IMG_PATH = "background.jpg"
        self.WIDTH, self.HEIGHT = 500, 400


        #Cài đặt kích thước tối thiểu cho cửa sổ
         self.win.geometry(f"{self.WIDTH}x{self.HEIGHT}")  #Cài đặt chiều dài và chiều cao của cửa sổ
```

```
        self.win.minsize(400, 300)  #Kích thước cửa sổ tối thiểu


        #Tạo widget canvas không viền và có thể thay đổi kích thước để chứa background image
        self.canvas = tk.Canvas(self.win, highlightthickness=0)
        self.canvas.pack(fill="both", expand=True)

        self.bg_image_original = Image.open(IMG_PATH)
        self.update_background_image()

        self.user = tk.StringVar(value="postgres")
        self.password = tk.StringVar(value="123456")

        self.show_login_app()

        self.win.bind("<Configure>", self.on_resize)

    def update_background_image(self):
        #Thay đổi kích thước background image
            resized_bg = self.bg_image_original.resize((self.win.winfo_width(), self.win.winfo_height()))
        self.bg_photo = ImageTk.PhotoImage(resized_bg)
        self.canvas.create_image(0, 0, anchor=tk.NW, image=self.bg_photo)

    def show_login_app(self):
        self.form_frame = tk.Frame(self.win, bg="white", bd=2, relief=tk.RIDGE)
```

self.form_frame.place(*relx*=0.5, *rely*=0.5, *anchor*=tk.CENTER, *relwidth*=0.5, *relheight*=0.6)

title_label = tk.Label(*self*.form_frame, *text*="User Login", *font*=("Arial", 14, "bold"), *bg*="white")

title_label.pack(*pady*=10)

user_label = tk.Label(*self*.form_frame, *text*="Username:", *bg*="white", *font*=("Arial", 10))

user_label.pack(*pady*=5)

user_entry = tk.Entry(*self*.form_frame, *textvariable*=*self*.user, *font*=("Arial", 10))

user_entry.pack(*pady*=5)

password_label = tk.Label(*self*.form_frame, *text*="Password:", *bg*="white", *font*=("Arial", 10))

password_label.pack(*pady*=5)

password_entry = tk.Entry(*self*.form_frame, *textvariable*=*self*.password, *show*="*", *font*=("Arial", 10))

password_entry.pack(*pady*=5)

login_button = tk.Button(*self*.form_frame, *text*="Login", *command*=*self*.validate_login, *bg*="lightblue", *font*=("Arial", 10))

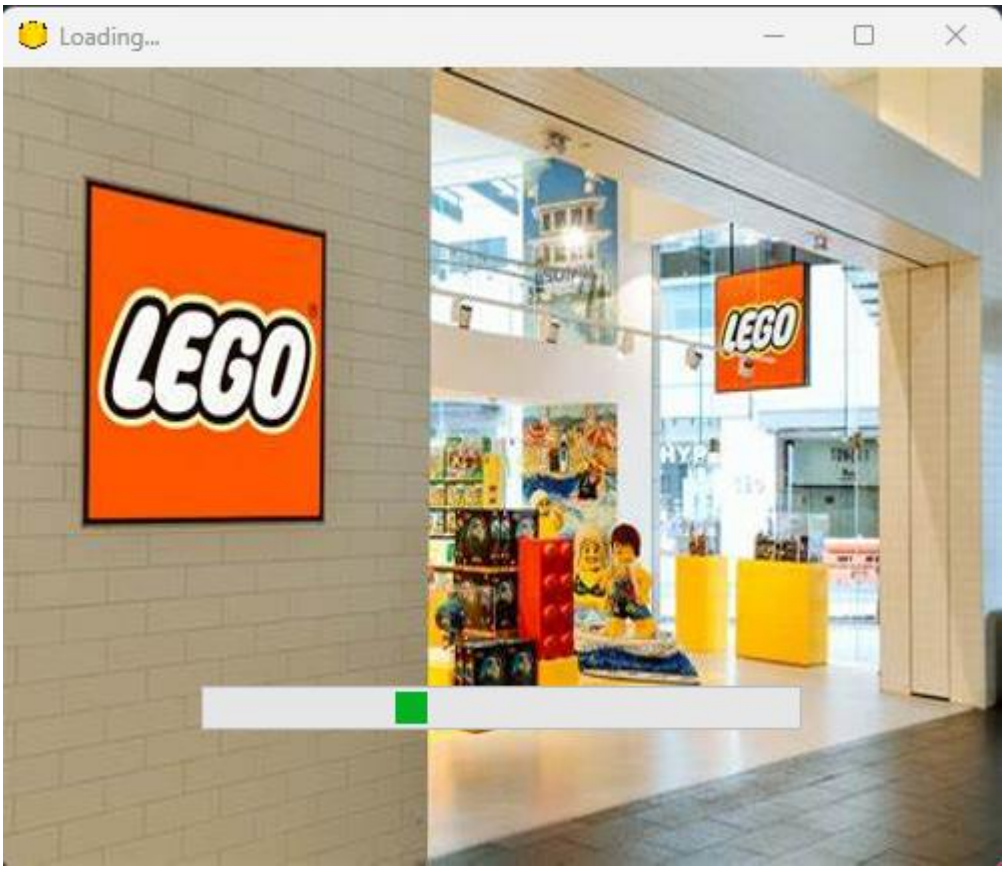
login_button.pack(*pady*=10)

self.error_message = tk.Label(*self*.form_frame, *text*="", *fg*="red", *bg*="white", *font*=("Arial", 10))

self.error_message.pack()

```
def on_resize(self, event):
    self.update_background_image()

def validate_login(self):
    username = self.user.get()
    password = self.password.get()

    if username == "postgres" and password == "123456":
        self.error_message.config(text="Login successful!", fg="green")
        self.show_loading_screen()
    else:
        self.error_message.config(text="Invalid username or password.", fg="red")

def show_loading_screen(self):
    self.win.destroy()
    main_window = tk.Tk()
    main_window.iconbitmap("lego.ico")
    app = LoadingScreen(main_window)
    main_window.mainloop()
```

2. Loading



Mục đích của trang này là làm hiệu ứng giống như chờ để đăng nhập vào trang ứng dụng chính. Trang này sử dụng hình ảnh background được cài đặt phóng to thu nhỏ theo kích thước của cửa sổ và tính năng ProgressBar tạo hiệu ứng loading trong vòng khoảng 3 giây sẽ chuyển hướng đến trang ứng dụng chính.

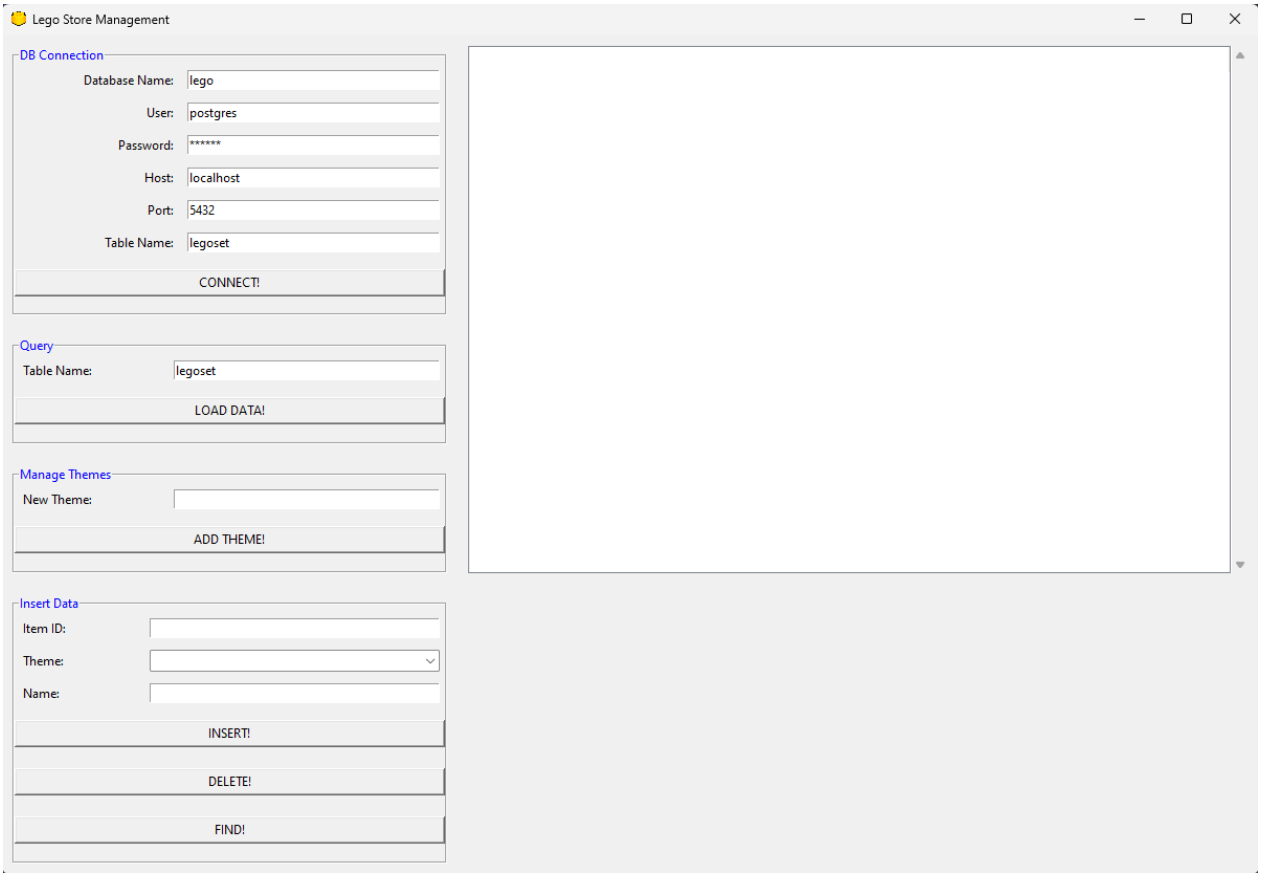
Toàn bộ code để làm LoadingScreen nằm trong class LoadingScreen:

```
class LoadingScreen:
    def __init__(self, win):
        self.win = win
        self.win.title("Loading...")
        self.win.geometry("500x400")
        self.win.configure(bg="#333")
```

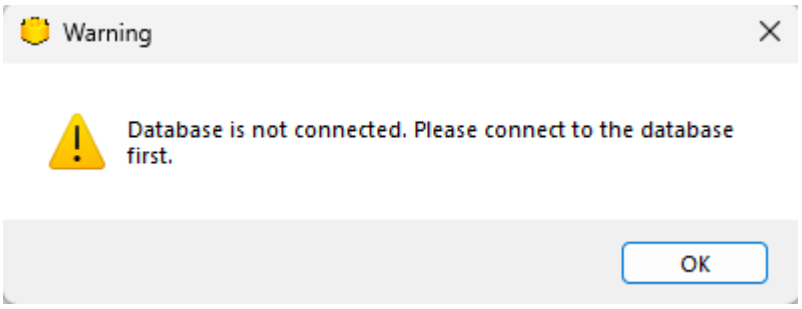
```
        self.bg_image = Image.open("OIP.jpg")
        self.bg_photo = ImageTk.PhotoImage(self.bg_image.resize((500, 400)))

        self.bg_label = tk.Label(self.win, image=self.bg_photo)
        self.bg_label.pack(fill=tk.BOTH, expand=True)

        self.win.bind("<Configure>", self.resize_background)

        self.progress = ttk.Progressbar(self.win, mode="indeterminate")
        self.progress.place(relx=0.5, rely=0.8, anchor=tk.CENTER, width=300)
        self.progress.start(10)

        self.win.after(3000, self.load_main_app)

    def resize_background(self, event):
        new_width = event.width
        new_height = event.height
        resized_image = self.bg_image.resize((new_width, new_height))
        self.bg_photo = ImageTk.PhotoImage(resized_image)

        self.bg_label.configure(image=self.bg_photo)
        self.bg_label.image = self.bg_photo

    def load_main_app(self):
        self.win.withdraw()
        self.show_main_app()
```

```
def show_main_app(self):
    self.win.destroy()
    main_window = tk.Tk()
    main_window.title("Main Application")
    main_window.iconbitmap("lego.ico")
    app = DatabaseApp(main_window)
    main_window.mainloop()
```

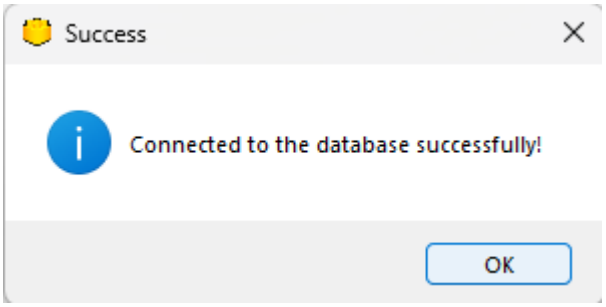
3. Kết nối database và thực thi lệnh



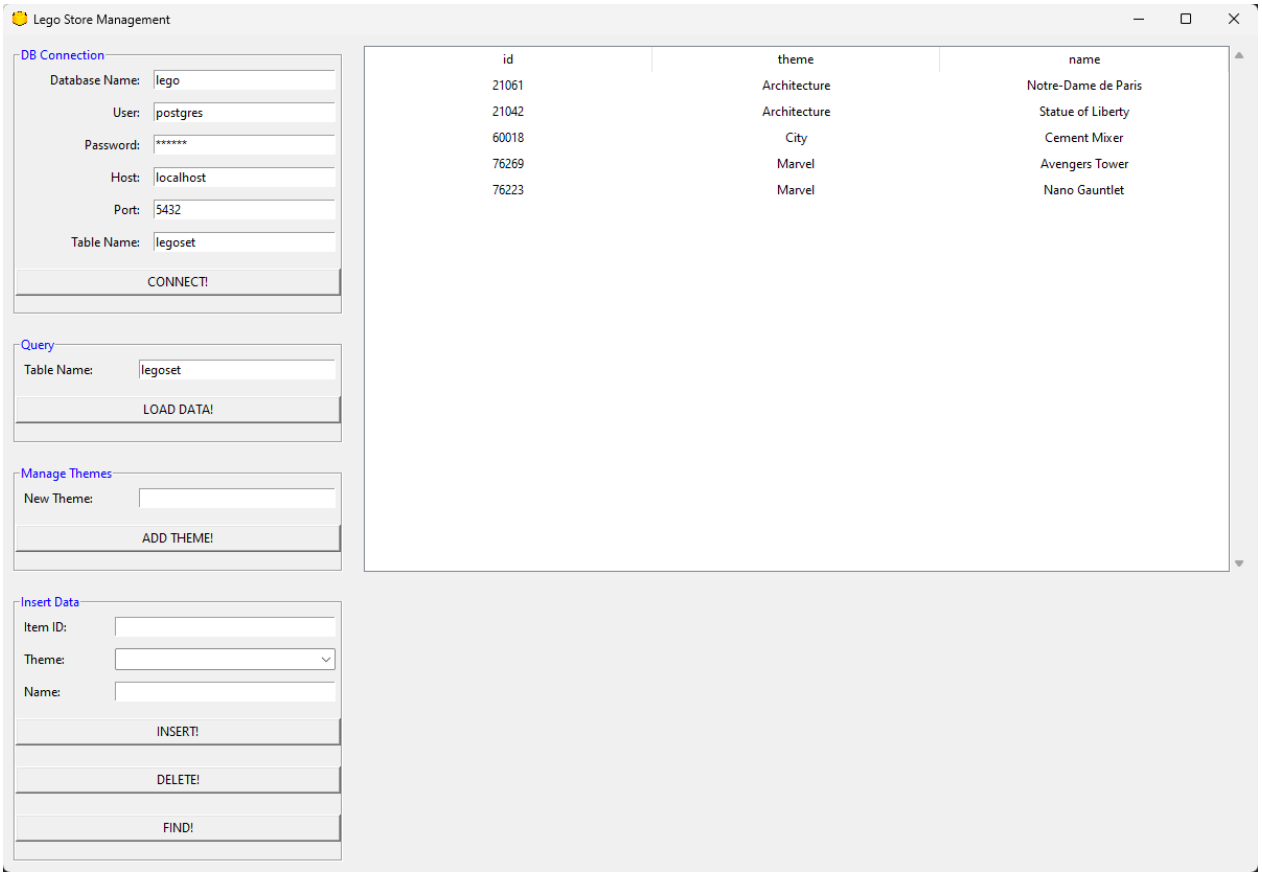
Sau khi LoadingScreen hết 3 giây thì sẽ chuyển hướng đến trang ứng dụng chính. Trang này gồm có các tính kết nối database và thực thi lệnh trong database. Khi vào đến trang này, đầu tiên ứng dụng sẽ hiện thông báo nhắc nhở người dùng rằng chưa kết nối database như sau:



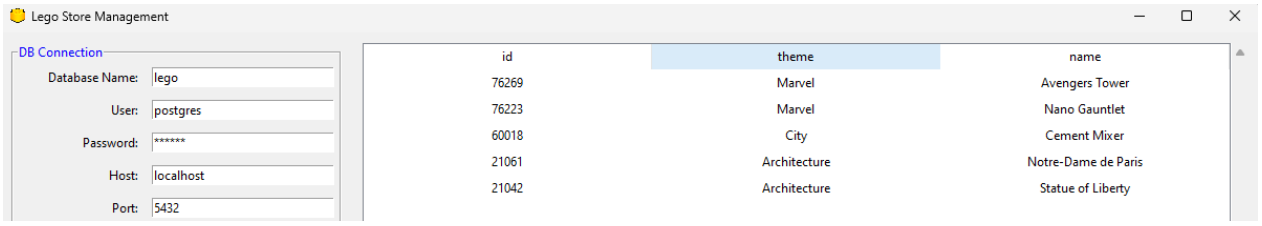
Khi người dùng kết nối thành công với database thì sẽ hiện thông báo sau:

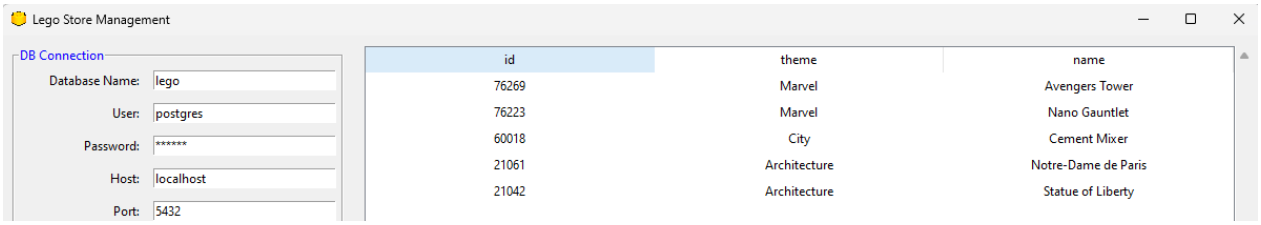


Khi người dùng nhập tên bảng và nhấn “LOAD DATA!” thì sẽ hiện toàn bộ nội dung bảng ở kế bên dưới dạng tương tự như sheet trong Excel.

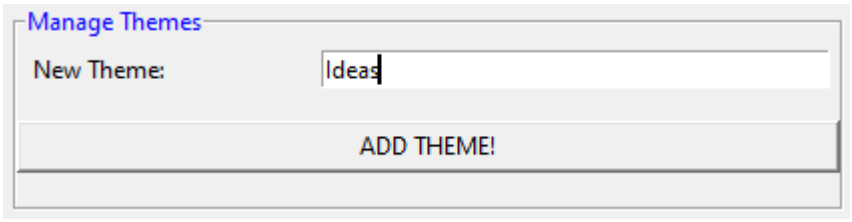


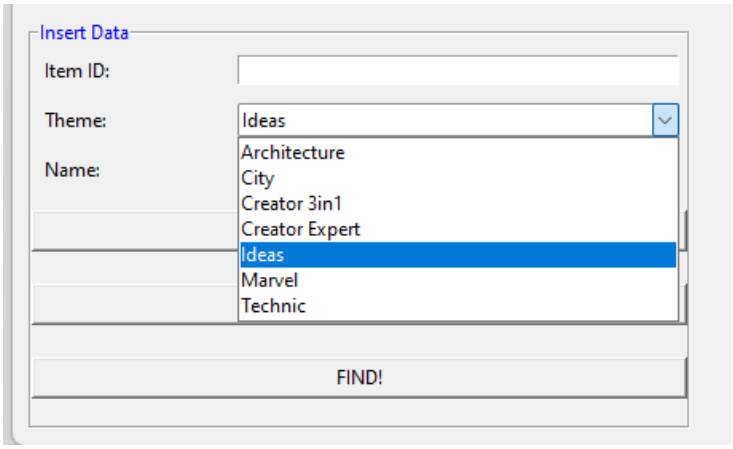
Bảng này có tính năng năng sắp xếp theo theo bảng chữ cái hoặc tăng dần, giảm dần khi người dùng nhấn vào tên cột.



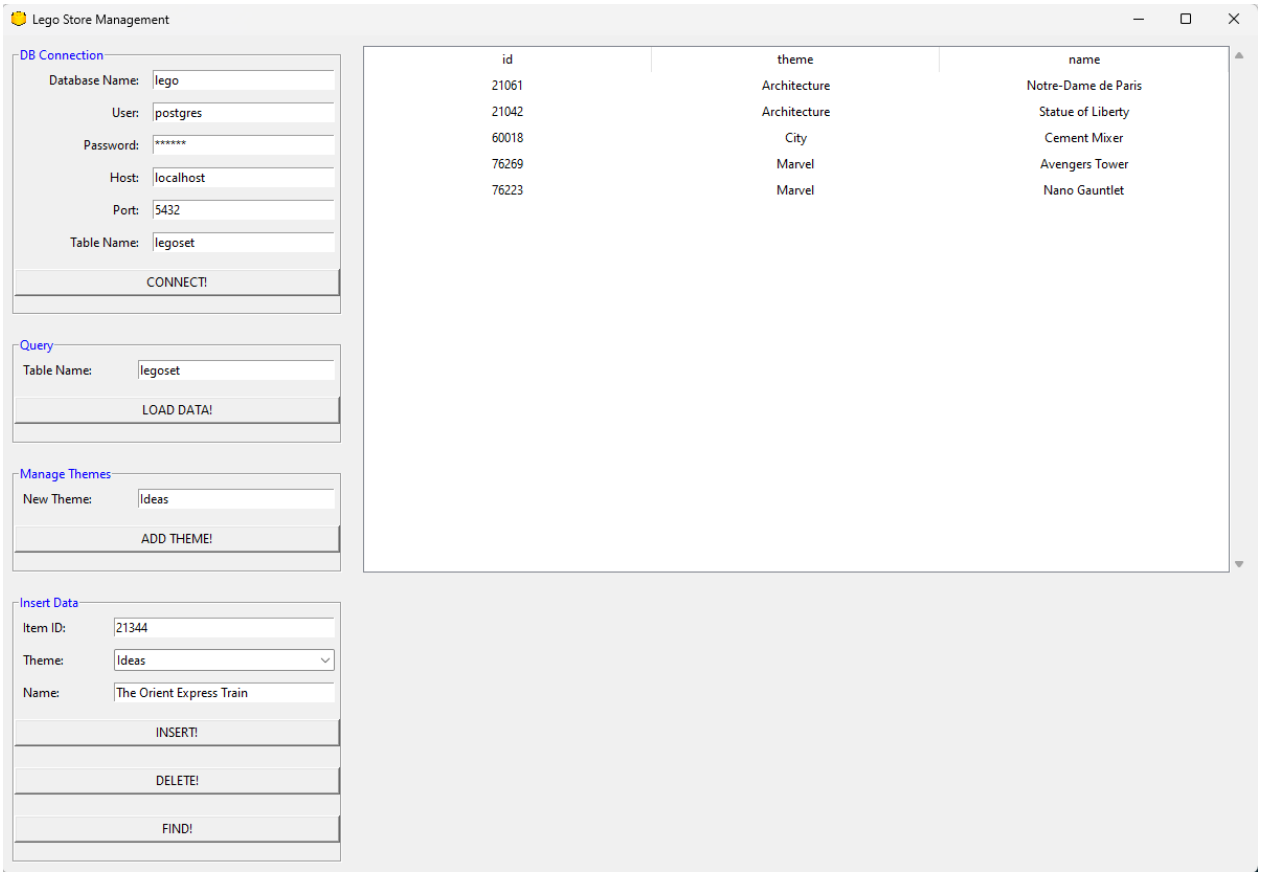


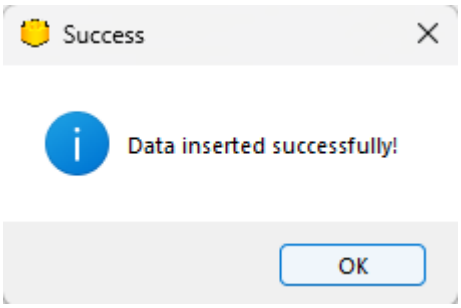
Trong database "lego" có 2 bảng: 1 là "legoset" để hiển thị toàn bộ sản phẩm được thêm vào, 2 là "themes" để hiển thị những chủ đề được thêm vào. Những chủ đề này được thêm bằng tính năng Manage Themes, sau đó dữ liệu được lưu vào ComboBox của tính năng Insert Data.

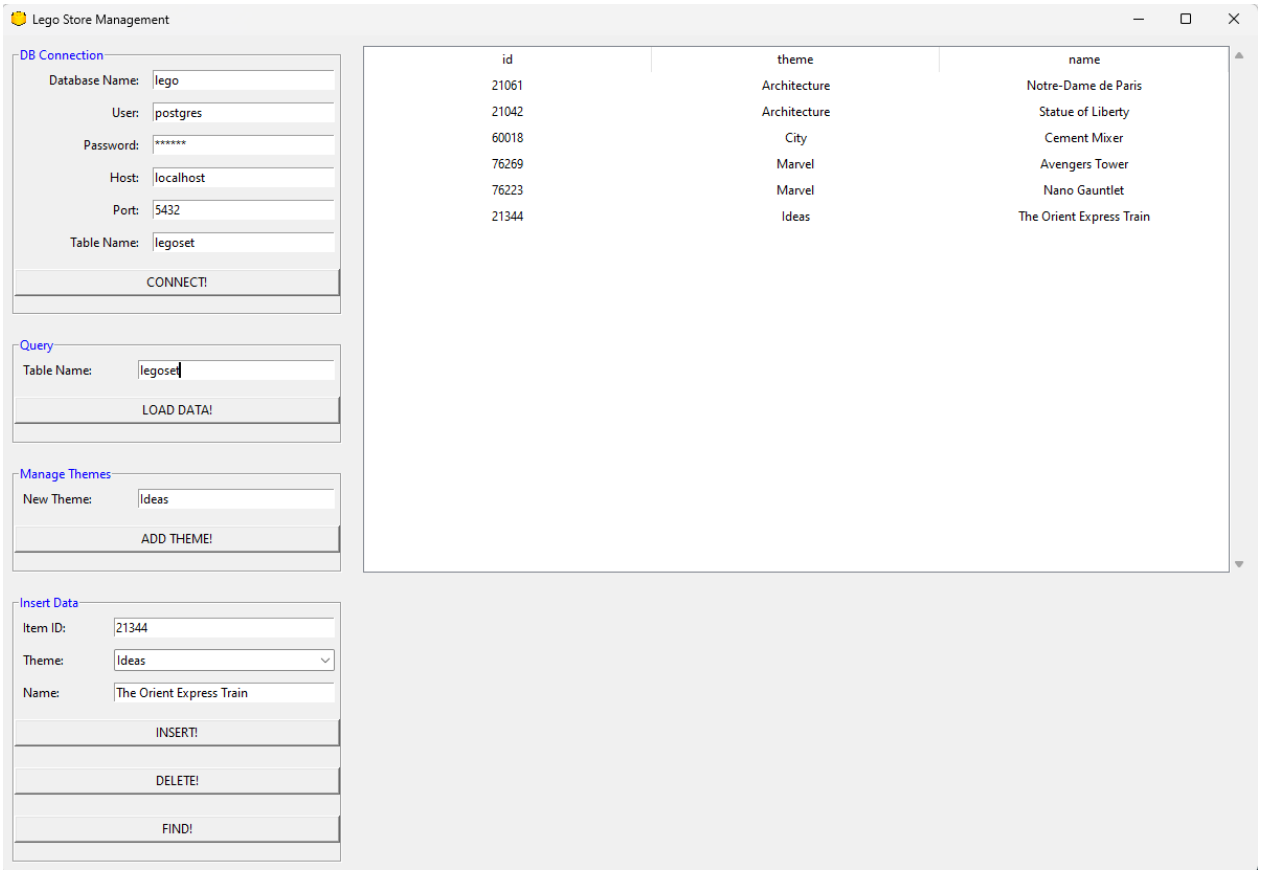




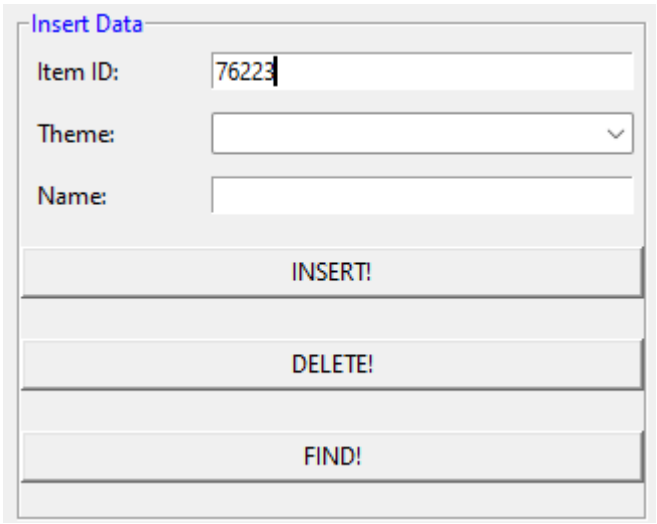
Tiếp theo là Thêm sản phẩm mới vào bảng "legoset".

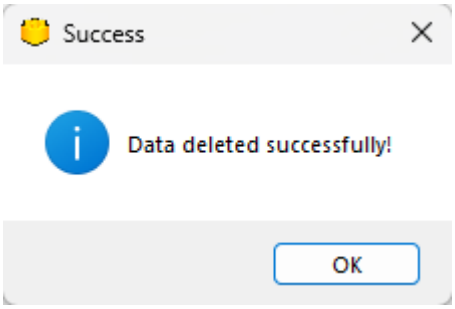

Lego Store Management
DB Connection
Database Name: lego
User: postgres
Password: ******
Host: localhost
Port: 5432
Table Name: legoset
CONNECT!
Query
Table Name: legoset
LOAD DATA!
Manage Themes
New Theme: Ideas
ADD THEME!
Insert Data
Item ID: 21344
Theme: Ideas
Name: The Orient Express Train
INSERT!
DELETE!
FIND!
id theme name
21061 Architecture Notre-Dame de Paris
21042 Architecture Statue of Liberty
60018 City Cement Mixer
76269 Marvel Avengers Tower
76223 Marvel Nano Gauntlet

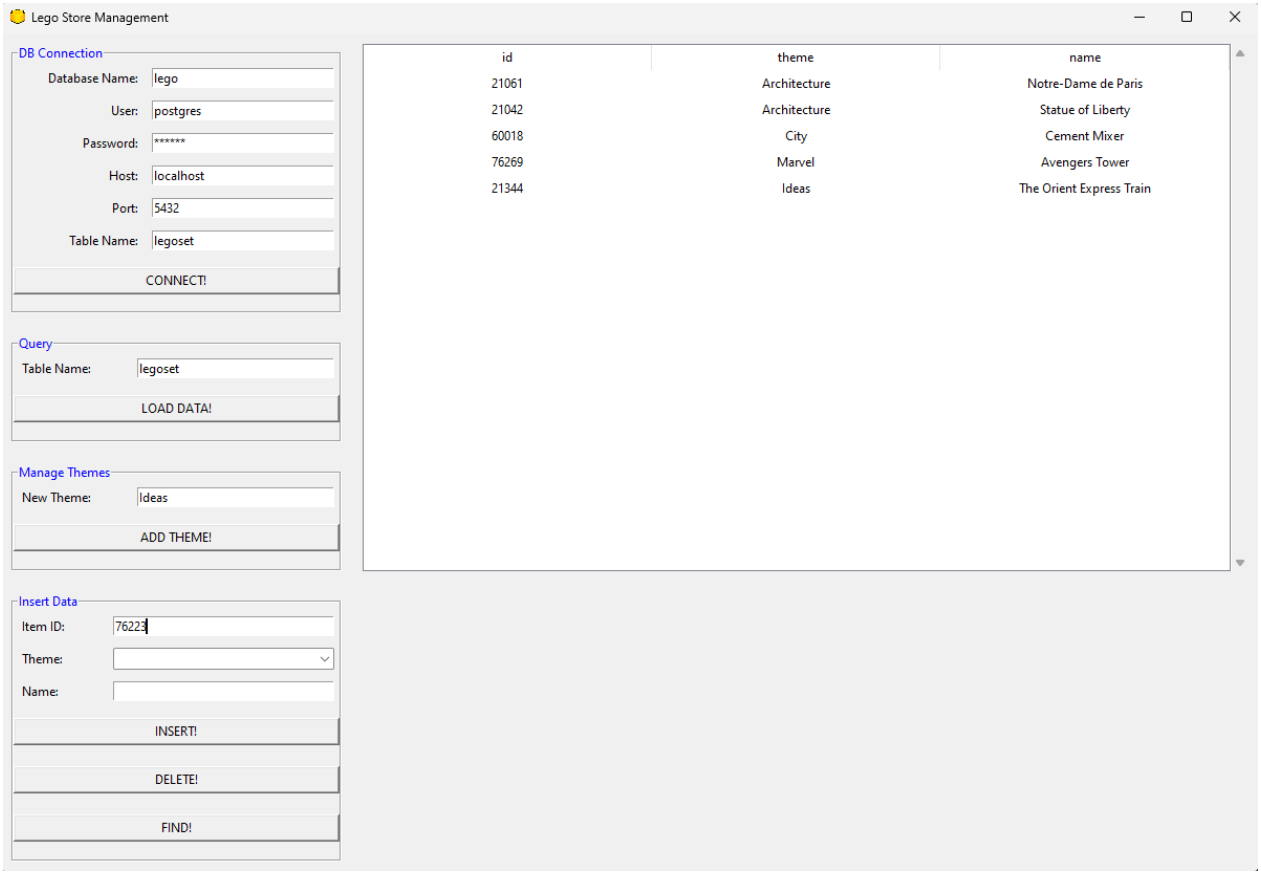

Success
Data inserted successfully!
OK



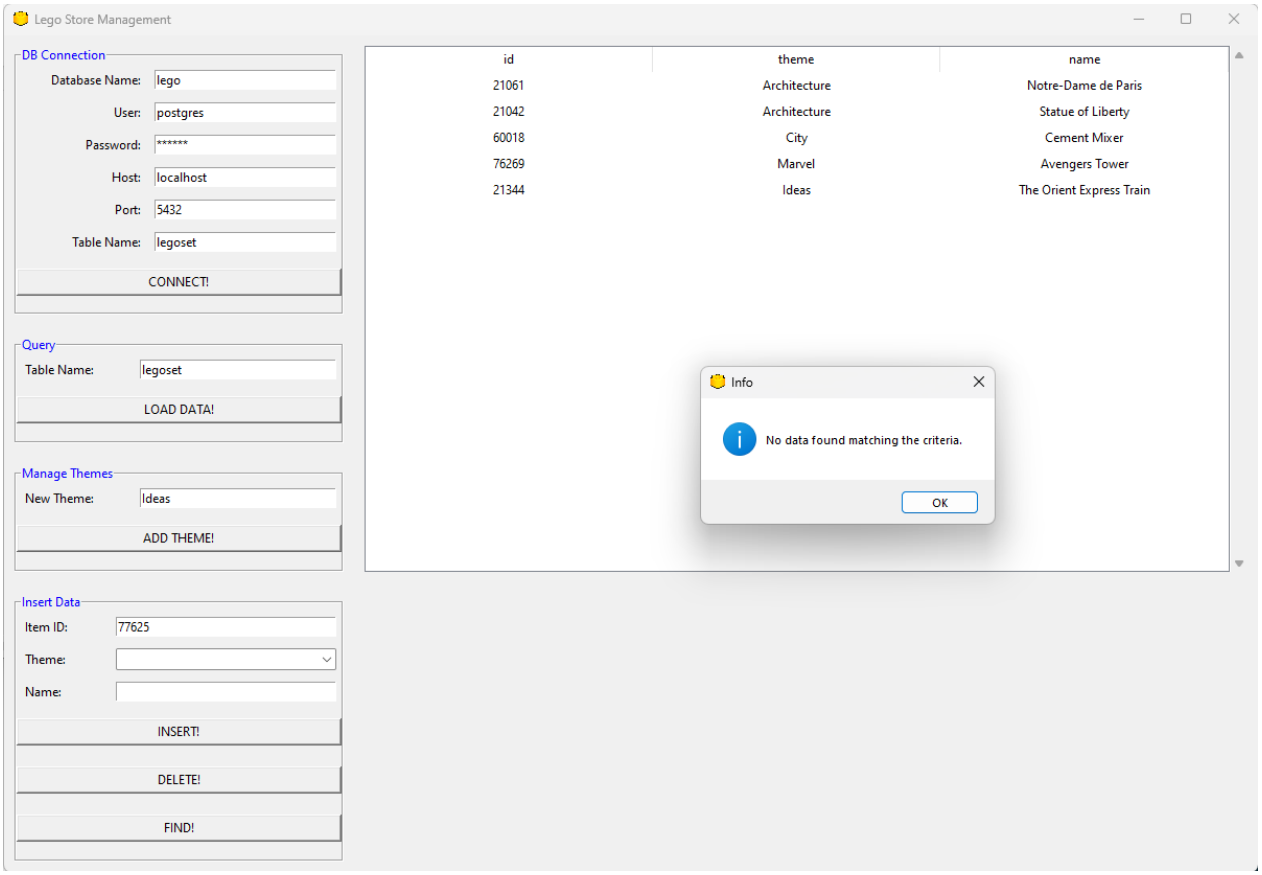
Tiếp theo là Xóa sản phẩm theo Id.



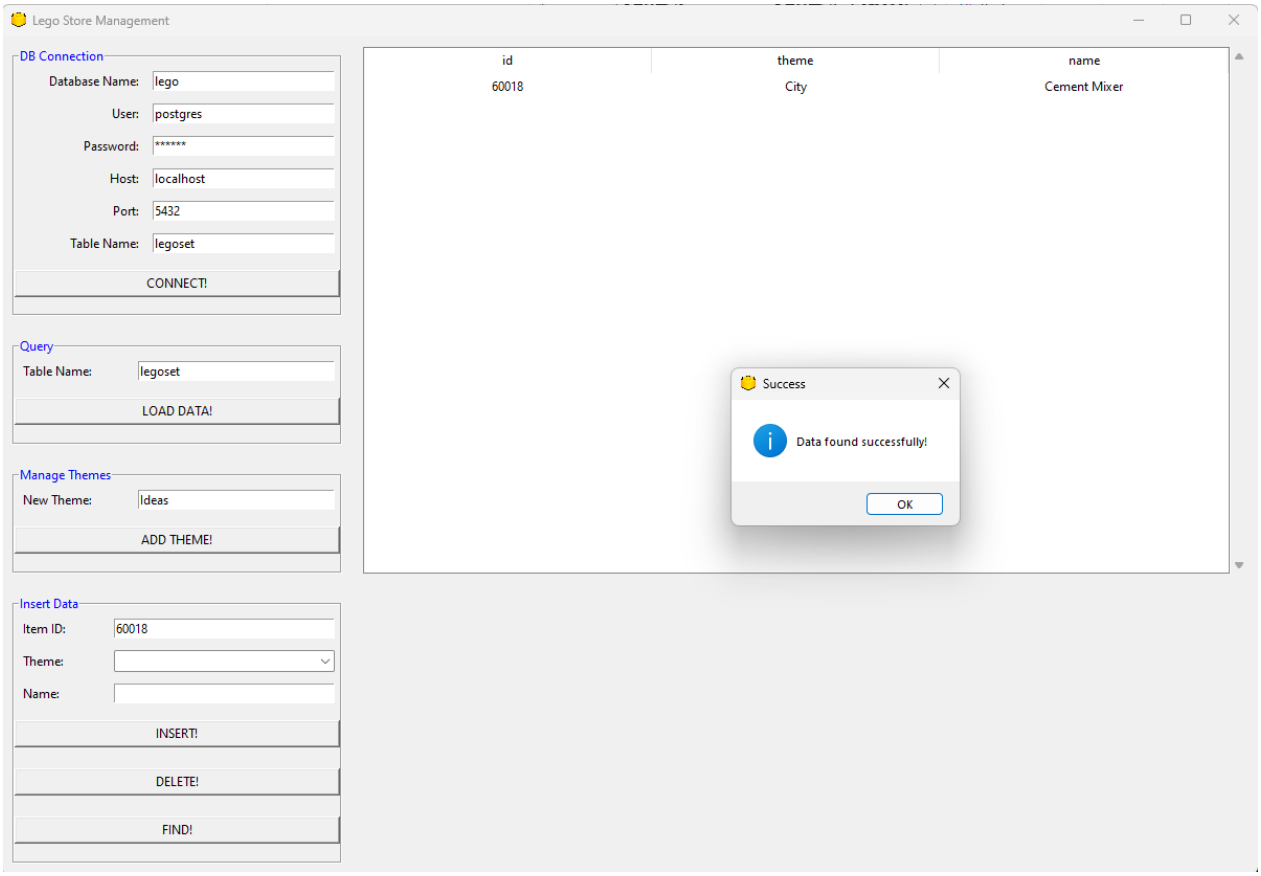




Tiếp theo là Tìm kiếm sản phẩm theo Id. Nếu Id sản phẩm không tồn tại trong bảng thì ứng dụng sẽ hiển thị thông báo sau:



Còn nếu tồn tại thì ứng dụng sẽ hiển thị thông báo sau:


Lego Store Management
DB Connection
Database Name: lego
User: postgres
Password: ******
Host: localhost
Port: 5432
Table Name: legoset
CONNECT!
Query
Table Name: legoset
LOAD DATA!
Manage Themes
New Theme: Ideas
ADD THEME!
Insert Data
Item ID: 60018
Theme:
Name:
INSERT!
DELETE!
FIND!
id
theme
name
60018
City
Cement Mixer
Success
Data found successfully!
OK

III. MÃ CHƯƠNG TRÌNH

```
import tkinter as tk
from tkinter import messagebox as msg
from tkinter import ttk
import psycopg2
from psycopg2 import sql
from PIL import Image, ImageTk

class LoginApp:
    def __init__(self, win):
        self.win = win
        self.win.title("Login")

        IMG_PATH = "Baitap2/background.jpg"
        self.WIDTH, self.HEIGHT = 500, 400

        #Cài đặt kích thước tối thiểu cho cửa sổ
        self.win.geometry(f"{self.WIDTH}x{self.HEIGHT}")  #Cài đặt
chiều dài và chiều cao của cửa sổ
        self.win.minsize(400, 300)  #Kích thước cửa sổ tối thiểu

        #Tạo widget canvas không viền và có thể thay đổi kích thước
để chứa background image
        self.canvas = tk.Canvas(self.win, highlightthickness=0)
        self.canvas.pack(fill="both", expand=True)

        self.bg_image_original = Image.open(IMG_PATH)
        self.update_background_image()

        self.user = tk.StringVar(value="postgres")
        self.password = tk.StringVar(value="123456")

        self.show_login_app()

        self.win.bind("<Configure>", self.on_resize)

    def update_background_image(self):
        #Thay đổi kích thước background image
```

```
        resized_bg =
self.bg_image_original.resize((self.win.winfo_width(),
self.win.winfo_height()))
        self.bg_photo = ImageTk.PhotoImage(resized_bg)
        self.canvas.create_image(0, 0, anchor=tk.NW,
image=self.bg_photo)

    def show_login_app(self):
        self.form_frame = tk.Frame(self.win, bg="white", bd=2,
relief=tk.RIDGE)
        self.form_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER,
relwidth=0.5, relheight=0.6)

        title_label = tk.Label(self.form_frame, text="User Login",
font=("Arial", 14, "bold"), bg="white")
        title_label.pack(pady=10)

        user_label = tk.Label(self.form_frame, text="Username:",
bg="white", font=("Arial", 10))
        user_label.pack(pady=5)
        user_entry = tk.Entry(self.form_frame,
textvariable=self.user, font=("Arial", 10))
        user_entry.pack(pady=5)

        password_label = tk.Label(self.form_frame, text="Password:",
bg="white", font=("Arial", 10))
        password_label.pack(pady=5)
        password_entry = tk.Entry(self.form_frame,
textvariable=self.password, show="*", font=("Arial", 10))
        password_entry.pack(pady=5)

        login_button = tk.Button(self.form_frame, text="Login",
command=self.validate_login, bg="lightblue", font=("Arial", 10))
        login_button.pack(pady=10)

        self.error_message = tk.Label(self.form_frame, text="",
fg="red", bg="white", font=("Arial", 10))
        self.error_message.pack()

    def on_resize(self, event):
```

```
        self.update_background_image()

    def validate_login(self):
        username = self.user.get()
        password = self.password.get()

        if username == "postgres" and password == "123456":
            self.error_message.config(text="Login successful!",
fg="green")
            self.show_loading_screen()
        else:
            self.error_message.config(text="Invalid username or
password.", fg="red")

    def show_loading_screen(self):
        self.win.destroy()
        main_window = tk.Tk()
        main_window.iconbitmap("Baitap2/lego.ico")
        app = LoadingScreen(main_window)
        main_window.mainloop()

class LoadingScreen:
    def __init__(self, win):
        self.win = win
        self.win.title("Loading...")
        self.win.geometry("500x400")
        self.win.configure(bg="#333")

        self.bg_image = Image.open("Baitap2/OIP.jpg")
        self.bg_photo =
ImageTk.PhotoImage(self.bg_image.resize((500, 400)))

        self.bg_label = tk.Label(self.win, image=self.bg_photo)
        self.bg_label.pack(fill=tk.BOTH, expand=True)

        self.win.bind("<Configure>", self.resize_background)

        self.progress = ttk.Progressbar(self.win,
mode="indeterminate")
```

```
        self.progress.place(relx=0.5, rely=0.8, anchor=tk.CENTER,
width=300)
        self.progress.start(10)

        self.win.after(3000, self.load_main_app)

    def resize_background(self, event):
        new_width = event.width
        new_height = event.height
        resized_image = self.bg_image.resize((new_width,
new_height))
        self.bg_photo = ImageTk.PhotoImage(resized_image)

        self.bg_label.configure(image=self.bg_photo)
        self.bg_label.image = self.bg_photo

    def load_main_app(self):
        self.win.withdraw()
        self.show_main_app()

    def show_main_app(self):
        self.win.destroy()
        main_window = tk.Tk()
        main_window.title("Main Application")
        main_window.iconbitmap("Baitap2/lego.ico")
        app = DatabaseApp(main_window)
        main_window.mainloop()

class DatabaseApp:
    def __init__(self, win):
        self.win = win
        self.win.title("Lego Store Management")
        self.win.geometry("1200x800")
        self.win.minsize(1200, 800)

        #Khởi tạo trạng thái sắp xếp
        self.sort_order = {}

        self.db_name = tk.StringVar(value="lego")
        self.user = tk.StringVar(value="postgres")
```

```
        self.password = tk.StringVar(value="123456")
        self.host = tk.StringVar(value="localhost")
        self.port = tk.StringVar(value="5432")
        self.table_name = tk.StringVar(value="legoset")

        self.create_widgets()
        self.configure_grid()

    def create_widgets(self):
        frame1 = tk.LabelFrame(self.win, text="DB Connection",
fg="blue")
        frame1.grid(row=0, column=0, padx=10, pady=10,
sticky="nsew")

        self.create_db_connection_widgets(frame1)

        frame2 = tk.LabelFrame(self.win, text="Query", fg="blue")
        frame2.grid(row=1, column=0, padx=10, pady=10,
sticky="nsew")

        self.create_query_widgets(frame2)

        frame_data = tk.Frame(self.win)
        frame_data.grid(row=0, column=1, rowspan=3, padx=10,
pady=10, sticky="nsew")

        self.create_data_treeview(frame_data)

        frame3 = tk.LabelFrame(self.win, text="Insert Data",
fg="blue")
        frame3.grid(row=3, column=0, padx=10, pady=10,
sticky="nsew")

        self.create_insert_data_widgets(frame3)

        frame4 = tk.LabelFrame(self.win, text="Manage Themes",
fg="blue")
        frame4.grid(row=2, column=0, padx=10, pady=10,
sticky="nsew")
```

```
        self.create_theme_management_widgets(frame4)

    def configure_grid(self):
        self.win.grid_columnconfigure(0, weight=1)
        self.win.grid_columnconfigure(1, weight=3)
        self.win.grid_rowconfigure(0, weight=1)
        self.win.grid_rowconfigure(1, weight=1)
        self.win.grid_rowconfigure(2, weight=1)
        self.win.grid_rowconfigure(3, weight=1)

    def create_db_connection_widgets(self, frame):
        labels = ["Database Name:", "User:", "Password:", "Host:",
"Port:", "Table Name:"]
        vars_ = [self.db_name, self.user, self.password, self.host,
self.port, self.table_name]

        for i, (label_text, var) in enumerate(zip(labels, vars_)):
            tk.Label(frame, text=label_text).grid(column=0, row=i,
padx=5, pady=5, sticky=tk.E)
            show = "*" if label_text == "Password:" else None
            tk.Entry(frame, textvariable=var,
show=show).grid(column=1, row=i, padx=5, pady=5, sticky="ew")

        tk.Button(frame, text="CONNECT!",
command=self.connect_db).grid(row=len(labels), columnspan=2,
pady=10, sticky="ew")

        frame.grid_columnconfigure(0, weight=1)
        frame.grid_columnconfigure(1, weight=2)

    def create_query_widgets(self, frame):
        tk.Label(frame, text="Table Name:").grid(column=0, row=0,
padx=5, pady=5, sticky=tk.W)
        tk.Entry(frame, textvariable=self.table_name).grid(column=1,
row=0, padx=5, pady=5, sticky="ew")

        # Load Data Button
        tk.Button(frame, text="LOAD DATA!",
command=self.load_data).grid(row=4, columnspan=2, pady=10,
sticky="ew")
```

```
        frame.grid_columnconfigure(0, weight=1)
        frame.grid_columnconfigure(1, weight=2)

    def create_data_treeview(self, frame):
        self.data_tree = ttk.Treeview(frame,
style="Custom.Treeview")
        self.data_tree.pack(side="left", fill="both", expand=True)

        scrollbar = ttk.Scrollbar(frame, orient="vertical",
command=self.data_tree.yview)
        self.data_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side="right", fill="y")

        style = ttk.Style()
        style.configure("Custom.Treeview", rowheight=25)
        style.map("Treeview", background=[("selected",
"lightblue")], foreground=[("selected", "black")])

    def create_insert_data_widgets(self, frame):
        self.column1 = tk.StringVar()
        self.column2 = tk.StringVar()
        self.column3 = tk.StringVar()

        labels = ["Item ID:", "Theme:", "Name:"]
        vars_ = [self.column1, self.column2, self.column3]

        for i, (label_text, var) in enumerate(zip(labels, vars_)):
            tk.Label(frame, text=label_text).grid(column=0, row=i,
padx=5, pady=5, sticky=tk.W)
            if label_text == "Theme:":
                self.combobox_column2 = ttk.Combobox(frame,
textvariable=self.column2)
                self.combobox_column2.grid(column=1, row=i, padx=5,
pady=5, sticky="ew")
                self.load_combobox_data()
            else:
                tk.Entry(frame, textvariable=var,
width=23).grid(column=1, row=i, padx=5, pady=5, sticky="ew")
```

```
        tk.Button(frame, text="INSERT!",
command=self.insert_data).grid(row=3, columnspan=2, pady=10,
sticky="ew")
        tk.Button(frame, text="DELETE!",
command=self.delete_data).grid(row=4, columnspan=2, pady=10,
sticky="ew")
        tk.Button(frame, text="FIND!",
command=self.find_data).grid(row=5, columnspan=2, pady=10,
sticky="ew")

        frame.grid_columnconfigure(0, weight=1)
        frame.grid_columnconfigure(1, weight=2)

    def create_theme_management_widgets(self, frame):
        self.new_theme = tk.StringVar()
        tk.Label(frame, text="New Theme:").grid(column=0, row=0,
padx=5, pady=5, sticky=tk.W)
        tk.Entry(frame, textvariable=self.new_theme).grid(column=1,
row=0, padx=5, pady=5, sticky="ew")
        tk.Button(frame, text="ADD THEME!",
command=self.add_theme).grid(row=1, columnspan=2, pady=10,
sticky="ew")

        frame.grid_columnconfigure(0, weight=1)
        frame.grid_columnconfigure(1, weight=2)

    def connect_db(self):
        try:
            self.conn = psycopg2.connect(
                dbname=self.db_name.get(),
                user=self.user.get(),
                password=self.password.get(),
                host=self.host.get(),
                port=self.port.get()
            )
            self.cur = self.conn.cursor()
            msg.showinfo("Success", "Connected to the database
successfully!")
            self.load_combobox_data()
        except Exception as e:
```

```
            msg.showerror("Error", f"Error connecting to the
database: {e}")

    def load_data(self, sort_column=None):
        try:
            base_query = sql.SQL("SELECT * FROM
{}").format(sql.Identifier(self.table_name.get()))
            if sort_column:
                current_order = self.sort_order.get(sort_column,
"ASC")
                new_order = "DESC" if current_order == "ASC" else
"ASC"
                self.sort_order[sort_column] = new_order
                base_query += sql.SQL(" ORDER BY {} {}").format(
                    sql.Identifier(sort_column),
                    sql.SQL(new_order)
                )
            else:
                self.sort_order = {}

            self.cur.execute(base_query)
            rows = self.cur.fetchall()
            column_names = [desc[0] for desc in
self.cur.description]

            self.data_tree.delete(*self.data_tree.get_children())
            self.data_tree["columns"] = column_names
            self.data_tree["show"] = "headings"

            for col in column_names:
                self.data_tree.heading(col, text=col, command=lambda
_col=col: self.load_data(_col))
                self.data_tree.column(col, width=100,
anchor="center")

            for row in rows:
                self.data_tree.insert("", "end", values=row)

        except Exception as e:
            msg.showerror("Error", f"Error loading data: {e}")
```

```
    def insert_data(self):
        try:
            if not self.column1.get() or not self.column2.get() or
not self.column3.get():
                msg.showerror("Error", "All fields (ID, theme, and
name) must be filled before inserting data.")
                return
            insert_query = sql.SQL("INSERT INTO {} (id, theme, name)
VALUES (%s, %s, %s)").format(sql.Identifier(self.table_name.get()))
            data_to_insert = (self.column1.get(),
self.column2.get(), self.column3.get())
            self.cur.execute(insert_query, data_to_insert)
            self.conn.commit()
            msg.showinfo("Success", "Data inserted successfully!")
            self.load_data()
        except Exception as e:
            msg.showerror("Error", f"Error inserting data: {e}")

    def delete_data(self):
        try:
            data_to_delete = (self.column1.get(),)
            delete_query = sql.SQL("DELETE FROM {} where id =
%s").format(sql.Identifier(self.table_name.get()))
            self.cur.execute(delete_query, data_to_delete)
            self.conn.commit()
            msg.showinfo("Success", "Data deleted successfully!")
            self.load_data()
        except Exception as e:
            msg.showerror("Error", f"Error deleting data: {e}")

    def find_data(self):
        try:
            search_value = self.column1.get()
            find_query = sql.SQL("SELECT * FROM {} WHERE id =
%s").format(sql.Identifier(self.table_name.get()))
            self.cur.execute(find_query, (search_value,))
            found_data = self.cur.fetchall()

            if found_data:
```

```
                self.data_tree.delete(*self.data_tree.get_children()
)

                column_names = [desc[0] for desc in
self.cur.description]
                self.data_tree["columns"] = column_names
                self.data_tree["show"] = "headings"

                for col in column_names:
                    self.data_tree.heading(col, text=col,
command=lambda _col=col: self.load_data(_col))
                    self.data_tree.column(col, width=100,
anchor="center")

                for row in found_data:
                    self.data_tree.insert("", "end", values=row)

                msg.showinfo("Success", "Data found successfully!")
            else:
                msg.showinfo("Info", "No data found matching the
criteria.")
        except Exception as e:
            msg.showerror("Error", f"Error finding data: {e}")

    def add_theme(self):
        try:
            theme_name = self.new_theme.get()
            insert_query = sql.SQL("INSERT INTO themes (theme)
VALUES (%s)")
            self.cur.execute(insert_query, (theme_name,))
            self.conn.commit()
            msg.showinfo("Success", "Theme added successfully!")
            self.load_combobox_data()
        except Exception as e:
            self.conn.rollback()
            msg.showerror("Error", f"Error adding theme: {e}")

    def load_combobox_data(self):
        try:
            if not hasattr(self, 'cur'):
```

```
                msg.showwarning("Warning", "Database is not
connected. Please connect to the database first.")
                return
            query = sql.SQL("SELECT DISTINCT theme FROM themes ORDER
BY theme ASC")
            self.cur.execute(query)
            themes = [row[0] for row in self.cur.fetchall()]
            self.combobox_column2['values'] = themes
        except Exception as e:
            msg.showerror("Error", f"Error loading Combobox data:
{e}")

if __name__ == "__main__":
    win = tk.Tk()
    win.iconbitmap("Baitap2/lego.ico")
    app = LoginApp(win)
    win.mainloop()
```

IV. GITHUB

Nguồn: https://github.com/YueTruong/pythonnc

HẾT