**STORE MANAGEMENT SYSTEM**

**ABSTRACT:**

This project involves the development of a comprehensive Shop Management System utilizing MySQL as the backend database and Python for the frontend interface. The system aims to streamline various aspects of shop management including inventory tracking, sales management, customer relations, and reporting.

The backend of the system will leverage MySQL to store and manage data related to products, customers, sales transactions, and employee records. Structured tables will be designed to efficiently store and retrieve information necessary for day-to-day operations of the shop.

The frontend of the system will be developed using Python, integrating graphical user interfaces (GUI) to provide an intuitive and user-friendly experience. Libraries such as Tkinter or PyQt will be utilized to design interactive screens for different functionalities, enabling users to perform tasks such as adding new products, updating inventory, processing sales, generating reports, and managing customer information.

Key features of the Shop Management System will include:

* **Inventory Management:** Tracking product details, stock levels, and supplier information.
* **Sales Management:** Recording and processing customer orders, generating invoices, and managing payments.
* **Customer Relationship Management (CRM):** Storing customer profiles, purchase histories, and preferences for personalized service.
* **Reporting and Analytics:** Generating reports on sales trends, inventory status, and financial performance.

The use of MySQL ensures data integrity, scalability, and reliability, supporting concurrent access and efficient data retrieval. Python's flexibility and extensive libraries facilitate the development of a dynamic and responsive frontend, enhancing usability and overall user experience.

In summary, this project will deliver a robust Shop Management System that integrates MySQL as the backend database and Python as the frontend interface, providing shop owners and staff with powerful tools to streamline operations, improve efficiency, and enhance customer service.

INTRODUCTION:

In today's dynamic retail landscape, efficient management of shop operations is crucial for ensuring smooth transactions, optimal inventory control, and enhanced customer satisfaction. The Shop Management System presented in this project aims to address these needs by leveraging the combined power of MySQL as the backend database and Python as the frontend interface.

This system offers a comprehensive solution for shop owners and managers to streamline essential tasks such as inventory management, sales processing, customer relationship management (CRM), and reporting. By integrating a robust backend database with a user-friendly frontend interface, the Shop Management System enables users to efficiently handle day-to-day operations and make informed business decisions.

The utilization of MySQL as the backend database ensures data integrity and scalability, providing a secure repository for storing and managing critical information including product details, inventory levels, customer records, and transaction histories. MySQL's relational database management system (RDBMS) capabilities offer efficient data storage and retrieval, supporting concurrent access from multiple users within the shop environment.

On the frontend, Python is employed to develop a responsive graphical user interface (GUI) using libraries such as Tkinter or PyQt. This frontend interface empowers users with intuitive screens and interactive functionalities, enabling them to perform tasks ranging from adding new products and updating inventory to processing sales transactions and generating analytical reports.

Through this project, shop owners and staff will benefit from a unified and efficient platform that enhances operational efficiency, optimizes resource utilization, and strengthens customer relationships. The Shop Management System's integration of MySQL and Python underscores a commitment to delivering a robust and scalable solution tailored to the needs of modern retail environments.

In the subsequent sections, we will delve deeper into the design, development, and implementation of this Shop Management System, highlighting key features, technical aspects, and the overall impact on improving shop operations and customer service.

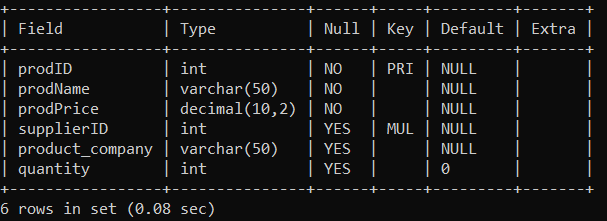
SQL TABLES:

**1.PRODUCT**

**SQL QUERY:**

CREATE TABLE products (  
 prodID INT PRIMARY KEY,  
 prodName VARCHAR(50) NOT NULL,  
 prodPrice DECIMAL(10, 2) NOT NULL,  
 supplierID INT,  
 product\_company VARCHAR(50),  
 quantity INT DEFAULT 0,  
 FOREIGN KEY (supplierID) REFERENCES supplier(supplierID))

**Structure of the table:**

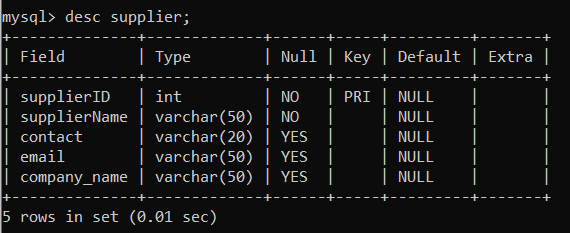


**2.Supplier**

**SQL QUERY:**

CREATE TABLE supplier (  
 supplierID INT PRIMARY KEY,  
 supplierName VARCHAR(50) NOT NULL,  
 contact VARCHAR(20),  
 email VARCHAR(50),  
 company\_name VARCHAR(50)  
)

**Structure of the table:**



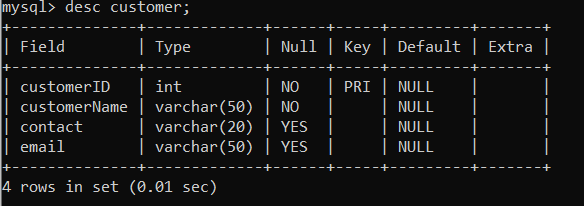
Top of Form

**3.Customer**

**SQL QUERY:**

CREATE TABLE customer (  
 customerID INT PRIMARY KEY,  
 customerName VARCHAR(50) NOT NULL,  
 contact VARCHAR(20),  
 email VARCHAR(50)  
)

**Structure of the table:**

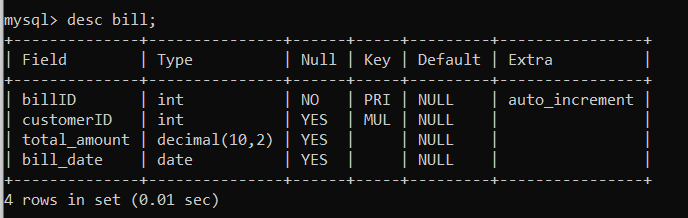


**4.Bill**

**SQL QUERY:**

CREATE TABLE bill (  
 billID INT AUTO\_INCREMENT PRIMARY KEY,  
 customerID INT,  
 total\_amount DECIMAL(10, 2),  
 bill\_date DATE,  
 FOREIGN KEY (customerID) REFERENCES customer(customerID)  
)

**Structure of the table:**



**JOINS:**

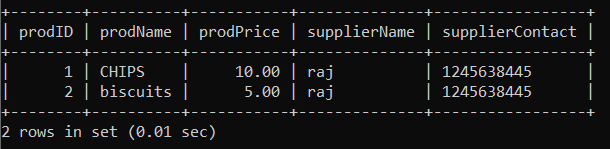
**1.INNER JOIN**

SELECT p.prodID, p.prodName, p.prodPrice, s.supplierName, s.contact AS supplierContact

FROM products p

INNER JOIN supplier s ON p.supplierID = s.supplierID;

**OUTPUT:**



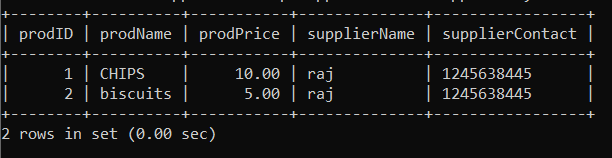
**2.LEFT JOIN**

SELECT p.prodID, p.prodName, p.prodPrice, s.supplierName, s.contact AS supplierContact

FROM products p

LEFT JOIN supplier s ON p.supplierID = s.supplierID;

**OUTPUT:**



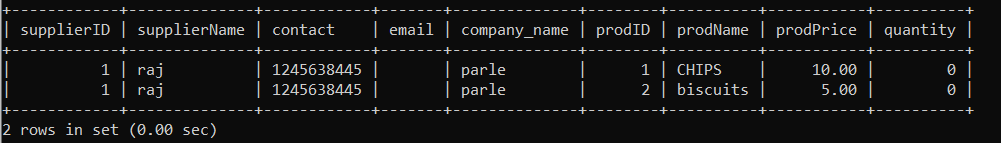
**3.RIGHT JOIN**

SELECT s.supplierID, s.supplierName, s.contact, s.email, s.company\_name, p.prodID, p.prodName, p.prodPrice, p.quantity

FROM supplier s

RIGHT JOIN products p ON s.supplierID = p.supplierID;

**OUTPUT:**



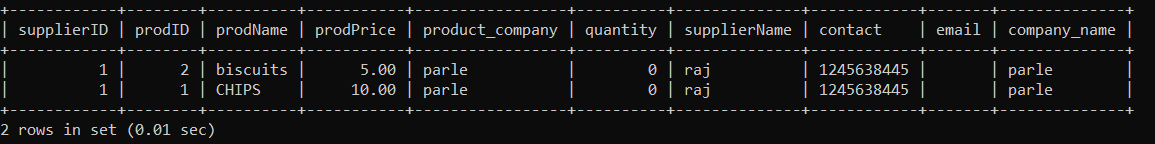
**4.NATURAL JOIN**

SELECT \*

FROM products

NATURAL JOIN supplier;

**OUTPUT:**

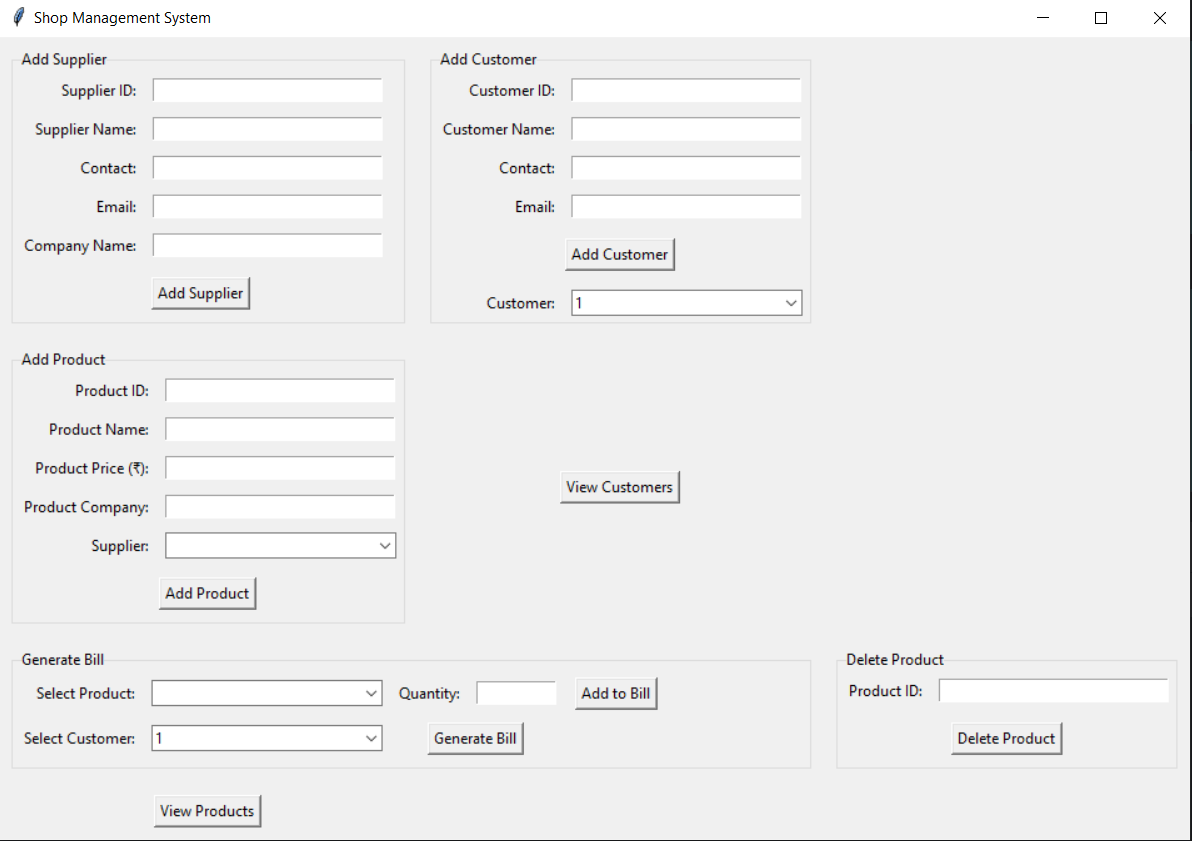


**CODE:**

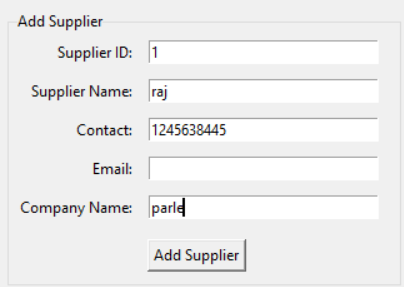
import tkinter as tk  
from tkinter import messagebox, ttk  
import mysql.connector  
import datetime # Import datetime module for date-related operations  
  
# Connect to MySQL database  
db = mysql.connector.connect(user="root", passwd="chetan17012005", host="localhost", database='sem4')  
cursor = db.cursor()  
  
# Create necessary tables if they don't exist  
cursor.execute("""  
 CREATE TABLE IF NOT EXISTS supplier (  
 supplierID INT PRIMARY KEY,  
 supplierName VARCHAR(50) NOT NULL,  
 contact VARCHAR(20),  
 email VARCHAR(50),  
 company\_name VARCHAR(50)  
 )  
""")  
  
cursor.execute("""  
 CREATE TABLE IF NOT EXISTS products (  
 prodID INT PRIMARY KEY,  
 prodName VARCHAR(50) NOT NULL,  
 prodPrice DECIMAL(10, 2) NOT NULL,  
 supplierID INT,  
 product\_company VARCHAR(50),  
 quantity INT DEFAULT 0,  
 FOREIGN KEY (supplierID) REFERENCES supplier(supplierID)  
 )  
""")  
  
cursor.execute("""  
 CREATE TABLE IF NOT EXISTS customer (  
 customerID INT PRIMARY KEY,  
 customerName VARCHAR(50) NOT NULL,  
 contact VARCHAR(20),  
 email VARCHAR(50)  
 )  
""")  
  
cursor.execute("""  
 CREATE TABLE IF NOT EXISTS bill (  
 billID INT AUTO\_INCREMENT PRIMARY KEY,  
 customerID INT,  
 total\_amount DECIMAL(10, 2),  
 bill\_date DATE,  
 FOREIGN KEY (customerID) REFERENCES customer(customerID)  
 )  
""")  
  
# Function to update supplier options in the dropdown  
def update\_supplier\_options():  
 cursor.execute("SELECT supplierID, supplierName FROM supplier")  
 suppliers = cursor.fetchall()  
 supplier\_options = {supplier[0]: supplier[1] for supplier in suppliers}  
  
 if supplier\_options:  
 supplier\_var.set(list(supplier\_options.keys())[0])  
 supplier\_dropdown['values'] = list(supplier\_options.values())  
 else:  
 supplier\_var.set("")  
 supplier\_dropdown['values'] = []  
  
  
# Function to update product options in the dropdown  
def update\_product\_options():  
 cursor.execute("SELECT prodID, prodName FROM products")  
 products = cursor.fetchall()  
 product\_options = {product[1]: product[0] for product in products}  
  
 if product\_options:  
 product\_var.set(list(product\_options.keys())[0])  
 product\_dropdown['values'] = list(product\_options.keys())  
 else:  
 product\_var.set("")  
 product\_dropdown['values'] = []  
  
  
# Function to add a new supplier to the database  
def add\_supplier():  
 supplier\_id = int(supplier\_id\_entry.get())  
 name = supplier\_name\_entry.get()  
 contact = contact\_entry.get()  
 email = email\_entry.get()  
 company = company\_name\_entry.get()  
  
 if name and company:  
 try:  
 # Insert Supplier into Database  
 query = "INSERT INTO supplier (supplierID, supplierName, contact, email, company\_name) VALUES (%s, %s, %s, %s, %s)"  
 cursor.execute(query, (supplier\_id, name, contact, email, company))  
 db.commit()  
 messagebox.showinfo("Success", "Supplier added successfully")  
 update\_supplier\_options() # Refresh supplier options after adding new supplier  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to add supplier: {e}")  
 else:  
 messagebox.showerror("Error", "Please provide supplier ID, name, and company name")  
  
  
# Function to add a new product to the database  
def add\_product():  
 name = prod\_name\_entry.get()  
 price = float(prod\_price\_entry.get())  
 company = prod\_company\_entry.get()  
 supplier\_id = supplier\_var.get()  
 product\_id = int(prod\_id\_entry.get()) if prod\_id\_entry.get() else None # Get the product ID if provided  
  
 if name and price > 0 and supplier\_id:  
 try:  
 # Insert Product into Database  
 if product\_id:  
 query = "INSERT INTO products (prodID, prodName, prodPrice, supplierID, product\_company) VALUES (%s, %s, %s, %s, %s)"  
 cursor.execute(query, (product\_id, name, price, supplier\_id, company))  
 else:  
 query = "INSERT INTO products (prodName, prodPrice, supplierID, product\_company) VALUES (%s, %s, %s, %s)"  
 cursor.execute(query, (name, price, supplier\_id, company))  
  
 db.commit()  
 messagebox.showinfo("Success", "Product added successfully")  
 update\_product\_options() # Refresh product options after adding new product  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to add product: {e}")  
 else:  
 messagebox.showerror("Error", "Please provide product name, price, company, and select a supplier")  
  
  
# Function to delete a product from the database by ID  
def delete\_product\_by\_id():  
 product\_id = int(delete\_id\_entry.get())  
  
 if product\_id:  
 try:  
 # Delete Product from Database  
 query = "DELETE FROM products WHERE prodID = %s"  
 cursor.execute(query, (product\_id,))  
 db.commit()  
 messagebox.showinfo("Success", f"Product with ID '{product\_id}' deleted successfully")  
 update\_product\_options() # Refresh product options after deletion  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to delete product: {e}")  
 else:  
 messagebox.showerror("Error", "Please enter a valid product ID")  
  
  
# Function to view all products  
def view\_products():  
 # Fetch products from the database  
 cursor.execute("SELECT prodID, prodName, prodPrice, product\_company FROM products")  
 products = cursor.fetchall()  
  
 if products:  
 # Create a new window to display products  
 products\_window = tk.Toplevel(root)  
 products\_window.title("Products")  
  
 # Display products in a treeview  
 tree = ttk.Treeview(products\_window, columns=("Name", "Price (₹)", "Company"))  
 tree.heading("#0", text="ID")  
 tree.heading("Name", text="Name")  
 tree.heading("Price (₹)", text="Price (₹)")  
 tree.heading("Company", text="Company")  
  
 for product in products:  
 tree.insert("", "end", text=product[0], values=(product[1], f"₹{product[2]:.2f}", product[3]))  
  
 tree.pack(expand=True, fill="both")  
 else:  
 messagebox.showinfo("No Products", "No products found in the database.")  
  
def add\_customer():  
 customer\_id = int(customer\_id\_entry.get())  
 name = customer\_name\_entry.get()  
 contact = customer\_contact\_entry.get()  
 email = customer\_email\_entry.get()  
  
 if name:  
 try:  
 # Insert Customer into Database  
 query = "INSERT INTO customer (customerID, customerName, contact, email) VALUES (%s, %s, %s, %s)"  
 cursor.execute(query, (customer\_id, name, contact, email))  
 db.commit()  
 messagebox.showinfo("Success", "Customer added successfully")  
 update\_customer\_options() # Refresh customer options after adding new customer  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to add customer: {e}")  
 else:  
 messagebox.showerror("Error", "Please provide customer ID and name")  
  
  
def update\_customer\_options():  
 cursor.execute("SELECT customerID, customerName FROM customer")  
 customers = cursor.fetchall()  
 customer\_options = {customer[0]: customer[1] for customer in customers}  
  
 if customer\_options:  
 customer\_var.set(list(customer\_options.keys())[0])  
 customer\_dropdown['values'] = list(customer\_options.values())  
 else:  
 customer\_var.set("")  
 customer\_dropdown['values'] = []  
  
  
def view\_customers():  
 # Fetch customers from the database  
 cursor.execute("SELECT customerID, customerName, contact, email FROM customer")  
 customers = cursor.fetchall()  
  
 if customers:  
 # Create a new window to display customers  
 customers\_window = tk.Toplevel(root)  
 customers\_window.title("Customers")  
  
 # Display customers in a treeview  
 tree = ttk.Treeview(customers\_window, columns=("Name", "Contact", "Email"))  
 tree.heading("#0", text="ID")  
 tree.heading("Name", text="Name")  
 tree.heading("Contact", text="Contact")  
 tree.heading("Email", text="Email")  
  
 for customer in customers:  
 tree.insert("", "end", text=customer[0], values=(customer[1], customer[2], customer[3]))  
  
 tree.pack(expand=True, fill="both")  
 else:  
 messagebox.showinfo("No Customers", "No customers found in the database.")  
  
# Function to generate a receipt  
def generate\_receipt(customer\_name, bill\_date, products, total\_amount):  
 receipt\_window = tk.Toplevel(root)  
 receipt\_window.title("Receipt")  
  
 receipt\_frame = ttk.Frame(receipt\_window, padding="20")  
 receipt\_frame.grid(row=0, column=0, sticky="nsew")  
  
 tk.Label(receipt\_frame, text=f"Customer Name: {customer\_name}").grid(row=0, column=0, sticky="w")  
 tk.Label(receipt\_frame, text=f"Bill Date: {bill\_date}").grid(row=1, column=0, sticky="w")  
  
 tk.Label(receipt\_frame, text="Products:").grid(row=2, column=0, sticky="w")  
 for idx, product in enumerate(products, start=3):  
 tk.Label(receipt\_frame, text=f"{product[0]} (Qty: {product[1]}) - ₹{product[2]:.2f}").grid(row=idx, column=0, sticky="w")  
  
 tk.Label(receipt\_frame, text="-------------------------------------------").grid(row=idx+1, column=0, sticky="w")  
 tk.Label(receipt\_frame, text=f"Total Amount: ₹{total\_amount:.2f}").grid(row=idx+2, column=0, sticky="w")  
  
# Function to generate a bill  
def generate\_bill():  
 customer\_id = customer\_var.get()  
 total\_amount = calculate\_total\_amount()  
 bill\_date = datetime.date.today()  
  
 if customer\_id and total\_amount > 0:  
 try:  
 # Insert Bill into Database  
 query = "INSERT INTO bill (customerID, total\_amount, bill\_date) VALUES (%s, %s, %s)"  
 cursor.execute(query, (customer\_id, total\_amount, bill\_date))  
 db.commit()  
 messagebox.showinfo("Success", "Bill generated and stored successfully")  
  
 # Fetch customer details  
 cursor.execute("SELECT customerName FROM customer WHERE customerID = %s", (customer\_id,))  
 customer\_name = cursor.fetchone()[0]  
  
 # Fetch products selected for the bill  
 products = []  
 for product\_id, quantity in selected\_products.items():  
 cursor.execute("SELECT prodName, prodPrice FROM products WHERE prodID = %s", (product\_id,))  
 product\_details = cursor.fetchone()  
 product\_name = product\_details[0]  
 product\_price = product\_details[1]  
 products.append((product\_name, quantity, product\_price))  
  
 # Generate and display the receipt  
 generate\_receipt(customer\_name, bill\_date, products, total\_amount)  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to generate bill: {e}")  
 else:  
 messagebox.showerror("Error", "Please select a customer and add products to generate a bill")  
  
def calculate\_total\_amount():  
 # Fetch products selected for the bill and calculate total amount  
 total\_amount = 0  
  
 for product\_id, quantity in selected\_products.items():  
 cursor.execute("SELECT prodPrice FROM products WHERE prodID = %s", (product\_id,))  
 product\_price = cursor.fetchone()[0]  
 total\_amount += product\_price \* quantity  
  
 return total\_amount  
  
# Function to handle adding selected products to the bill  
selected\_products = {}  
  
def add\_product\_to\_bill():  
 product\_id = product\_var.get()  
 quantity = int(quantity\_entry.get())  
  
 if product\_id and quantity > 0:  
 selected\_products[product\_id] = quantity  
 messagebox.showinfo("Success", "Product added to bill")  
 else:  
 messagebox.showerror("Error", "Please select a product and provide quantity")  
  
# GUI Setup  
root = tk.Tk()  
root.title("Shop Management System")  
  
# Supplier Section  
supplier\_frame = ttk.LabelFrame(root, text="Add Supplier")  
supplier\_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")  
  
tk.Label(supplier\_frame, text="Supplier ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")  
supplier\_id\_entry = tk.Entry(supplier\_frame, width=30)  
supplier\_id\_entry.grid(row=0, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(supplier\_frame, text="Supplier Name:").grid(row=1, column=0, padx=5, pady=5, sticky="e")  
supplier\_name\_entry = tk.Entry(supplier\_frame, width=30)  
supplier\_name\_entry.grid(row=1, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(supplier\_frame, text="Contact:").grid(row=2, column=0, padx=5, pady=5, sticky="e")  
contact\_entry = tk.Entry(supplier\_frame, width=30)  
contact\_entry.grid(row=2, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(supplier\_frame, text="Email:").grid(row=3, column=0, padx=5, pady=5, sticky="e")  
email\_entry = tk.Entry(supplier\_frame, width=30)  
email\_entry.grid(row=3, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(supplier\_frame, text="Company Name:").grid(row=4, column=0, padx=5, pady=5, sticky="e")  
company\_name\_entry = tk.Entry(supplier\_frame, width=30)  
company\_name\_entry.grid(row=4, column=1, padx=5, pady=5, sticky="w")  
  
add\_supplier\_button = tk.Button(supplier\_frame, text="Add Supplier", command=add\_supplier)  
add\_supplier\_button.grid(row=5, column=0, columnspan=2, pady=10)  
  
# Product Section  
product\_frame = ttk.LabelFrame(root, text="Add Product")  
product\_frame.grid(row=1, column=0, padx=10, pady=10, sticky="nsew")  
  
tk.Label(product\_frame, text="Product ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")  
prod\_id\_entry = tk.Entry(product\_frame, width=30)  
prod\_id\_entry.grid(row=0, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(product\_frame, text="Product Name:").grid(row=1, column=0, padx=5, pady=5, sticky="e")  
prod\_name\_entry = tk.Entry(product\_frame, width=30)  
prod\_name\_entry.grid(row=1, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(product\_frame, text="Product Price (₹):").grid(row=2, column=0, padx=5, pady=5, sticky="e")  
prod\_price\_entry = tk.Entry(product\_frame, width=30)  
prod\_price\_entry.grid(row=2, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(product\_frame, text="Product Company:").grid(row=3, column=0, padx=5, pady=5, sticky="e")  
prod\_company\_entry = tk.Entry(product\_frame, width=30)  
prod\_company\_entry.grid(row=3, column=1, padx=5, pady=5, sticky="w")  
  
# Fetch initial supplier options and populate the dropdown  
supplier\_var = tk.IntVar()  
supplier\_dropdown = ttk.Combobox(product\_frame, textvariable=supplier\_var, width=27)  
supplier\_dropdown.grid(row=4, column=1, padx=5, pady=5, sticky="w")  
  
update\_supplier\_options() # Populate supplier options initially  
  
tk.Label(product\_frame, text="Supplier:").grid(row=4, column=0, padx=5, pady=5, sticky="e")  
  
add\_product\_button = tk.Button(product\_frame, text="Add Product", command=add\_product)  
add\_product\_button.grid(row=5, column=0, columnspan=2, pady=10)  
  
# Delete Product Section  
delete\_frame = ttk.LabelFrame(root, text="Delete Product")  
delete\_frame.grid(row=2, column=2, padx=10, pady=10, sticky="nsew")  
  
tk.Label(delete\_frame, text="Product ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")  
delete\_id\_entry = tk.Entry(delete\_frame, width=30)  
delete\_id\_entry.grid(row=0, column=1, padx=5, pady=5, sticky="w")  
  
delete\_product\_button = tk.Button(delete\_frame, text="Delete Product", command=delete\_product\_by\_id)  
delete\_product\_button.grid(row=1, column=0, columnspan=2, pady=10)  
  
  
# View Products Section  
view\_products\_button = tk.Button(root, text="View Products", command=view\_products)  
view\_products\_button.grid(row=3, column=0, padx=10, pady=10)  
  
# Customer Section  
customer\_frame = ttk.LabelFrame(root, text="Add Customer")  
customer\_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")  
  
tk.Label(customer\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="e")  
customer\_id\_entry = tk.Entry(customer\_frame, width=30)  
customer\_id\_entry.grid(row=0, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(customer\_frame, text="Customer Name:").grid(row=1, column=0, padx=5, pady=5, sticky="e")  
customer\_name\_entry = tk.Entry(customer\_frame, width=30)  
customer\_name\_entry.grid(row=1, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(customer\_frame, text="Contact:").grid(row=2, column=0, padx=5, pady=5, sticky="e")  
customer\_contact\_entry = tk.Entry(customer\_frame, width=30)  
customer\_contact\_entry.grid(row=2, column=1, padx=5, pady=5, sticky="w")  
  
tk.Label(customer\_frame, text="Email:").grid(row=3, column=0, padx=5, pady=5, sticky="e")  
customer\_email\_entry = tk.Entry(customer\_frame, width=30)  
customer\_email\_entry.grid(row=3, column=1, padx=5, pady=5, sticky="w")  
  
add\_customer\_button = tk.Button(customer\_frame, text="Add Customer", command=add\_customer)  
add\_customer\_button.grid(row=4, column=0, columnspan=2, pady=10)  
  
# Fetch initial customer options and populate the dropdown  
customer\_var = tk.IntVar()  
customer\_dropdown = ttk.Combobox(customer\_frame, textvariable=customer\_var, width=27)  
customer\_dropdown.grid(row=5, column=1, padx=5, pady=5, sticky="w")  
  
update\_customer\_options() # Populate customer options initially  
  
tk.Label(customer\_frame, text="Customer:").grid(row=5, column=0, padx=5, pady=5, sticky="e")  
  
# View Customers Section  
view\_customers\_button = tk.Button(root, text="View Customers", command=view\_customers)  
view\_customers\_button.grid(row=1, column=1, padx=10, pady=10)  
  
# GUI Updates  
bill\_frame = ttk.LabelFrame(root, text="Generate Bill")  
bill\_frame.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")  
  
tk.Label(bill\_frame, text="Select Product:").grid(row=0, column=0, padx=5, pady=5, sticky="e")  
product\_var = tk.IntVar()  
product\_dropdown = ttk.Combobox(bill\_frame, textvariable=product\_var, width=27)  
product\_dropdown.grid(row=0, column=1, padx=5, pady=5, sticky="w")  
  
update\_product\_options() # Populate product options initially  
  
tk.Label(bill\_frame, text="Quantity:").grid(row=0, column=2, padx=5, pady=5, sticky="e")  
quantity\_entry = tk.Entry(bill\_frame, width=10)  
quantity\_entry.grid(row=0, column=3, padx=5, pady=5, sticky="w")  
  
add\_to\_bill\_button = tk.Button(bill\_frame, text="Add to Bill", command=add\_product\_to\_bill)  
add\_to\_bill\_button.grid(row=0, column=4, padx=10, pady=5)  
  
tk.Label(bill\_frame, text="Select Customer:").grid(row=1, column=0, padx=5, pady=5, sticky="e")  
customer\_dropdown\_bill = ttk.Combobox(bill\_frame, textvariable=customer\_var, width=27)  
customer\_dropdown\_bill.grid(row=1, column=1, padx=5, pady=5, sticky="w")  
  
update\_customer\_options() # Populate customer options initially  
  
generate\_bill\_button = tk.Button(bill\_frame, text="Generate Bill", command=generate\_bill)  
generate\_bill\_button.grid(row=1, column=2, columnspan=2, padx=10, pady=5)  
  
root.mainloop()

**OUTPUT:**

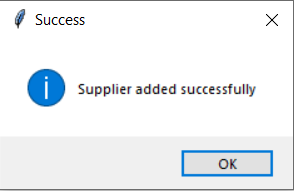
**1.MAIN WINDOW**



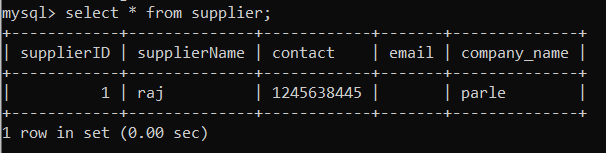
**2. ADDING SUPPLIER**



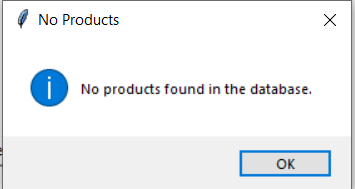
**AFTER ADDING POP UP MESSAGE:**



**CHANGES IN DBMS**



**2.VIEW PRODUCT WHEN NO ITEMS IS PRESENT**



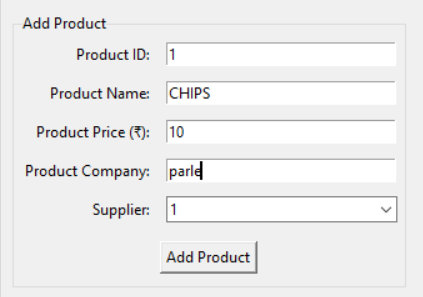
**AFTER ADDING A PRODUCT:**

**3.ADDING A PRODUCT**

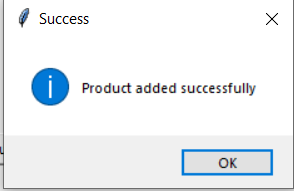
**BEFORE ADDING A PRODUCT**



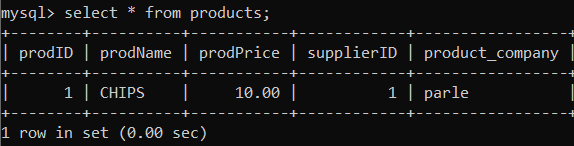
**AFTER ADDING A PRODUCT**



**SUCCESS MESSAGE POPUP:**

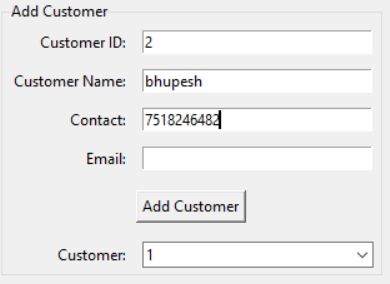


**CHANGES IN DBMS:**

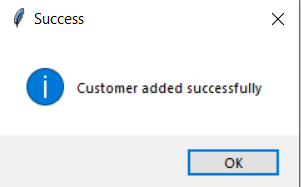


**3.ADDING A CUSTOMER**

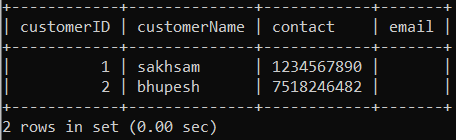
**GUI PANNEL:**



**SUCCESS POP-UP**

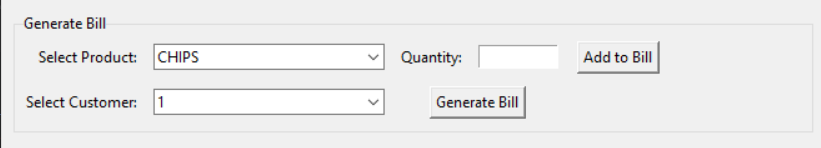


**AFTER ADDING A CUSTOMER**

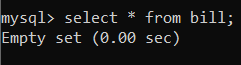


**4.GENERATING A BILL**

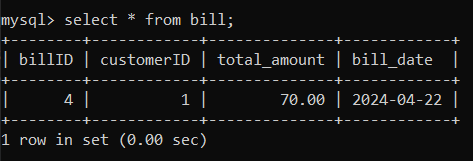
**GUI WINDOW**



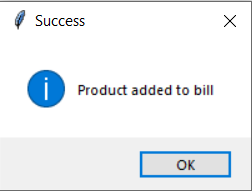
**BEFORE BILL GENERATION DBMS TABLE:**



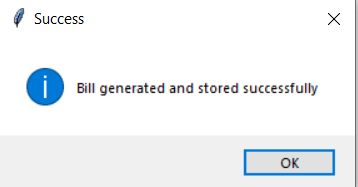
**AFTER BILL GENERATION:**



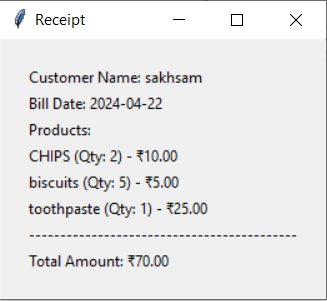
**PRODUCT ADDED TO BILL POP UP MESSAGE:**



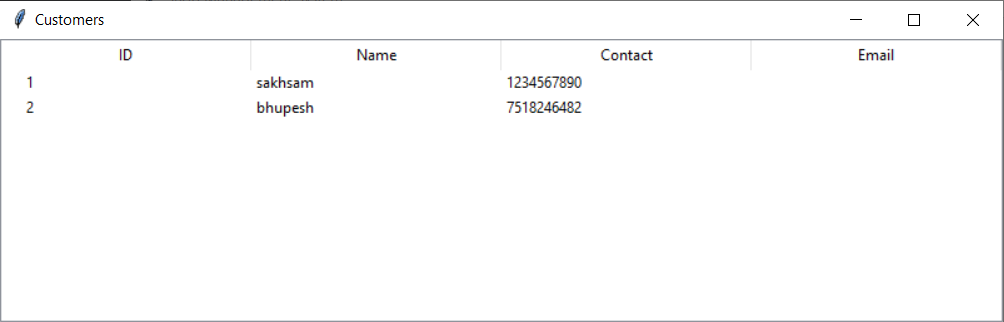
**BILL GENERATION SUCCESS POP UP:**

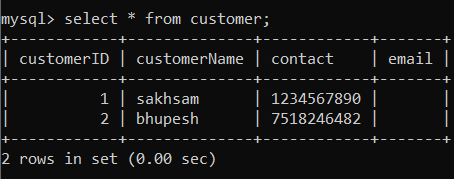


**RECIEPT**



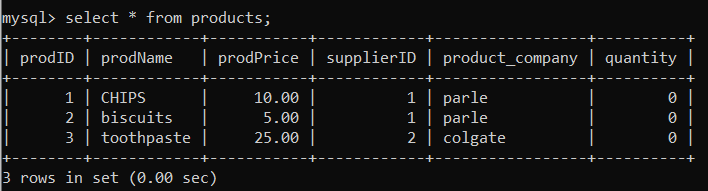
**5.VIEW CUSTOMERS:**



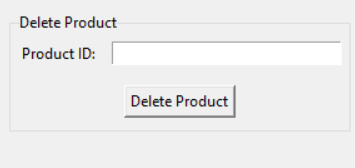


**6. DELETING A PRODUCT**

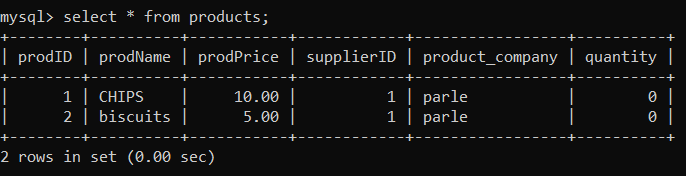
**BEFORE DELETING A PRODUCT:**



**6.DELETING GUI:**



**AFTER DELETING DBMS TABLE:**



**SUCCESS POP UP MESSAGE:**

