Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Mini-Project Report

Academic Year 2024-2025

Course: Database Technologies Laboratory | Course Code: DJS23SCMD301

Class: S.Y.B.Tech. | Semester: III | Division: S

Department: Artificial Intelligence (AI) and Data Science | Batch: A1

Inventory Management System

Prepared by

Sr. No.	Roll No.	Name	SAP ID
1	S009	Chaitanya Shah	60018230034
2	S007	Bansari Naik	60018230089
3	S027	Kavish Shah	60018230084
4	S016	Harsh Chandramania	60018230010

Table of Contents

1 Problem Statement

In a business environment, managing inventory efficiently is crucial for ensuring smooth operations. Companies struggle to track employees, suppliers, products, customers, and payments within a single interface. The lack of a unified system results in fragmented data management, slow response times, and errors in handling customer care, warehouse capacity, product details, and payments. This Inventory Management System project aims to solve these issues by offering a comprehensive solution to manage and track inventory data efficiently.

2 Project Idea

The Inventory Management System project is designed to provide an integrated solution for businesses to manage their inventory effectively. This system brings together key functionalities to streamline the handling of various aspects of inventory data, such as employee details, customer care support, product information, suppliers, warehouse management, and customer transactions. By consolidating these modules into a single, user-friendly platform, the system helps companies maintain organized and up-to-date records, improving overall operational efficiency.

This project includes a graphical user interface (GUI) that allows users to perform essential tasks, like adding, viewing, updating, or deleting records across different modules. Each table, such as “Employee,” “Customer Care,” “Warehouse,” “Products,” and “Customer,” represents a specific aspect of inventory management, and the system facilitates smooth data operations while preserving data integrity and enforcing relationships between tables. For instance, customer care records are linked to employee records, and each product has a price, expiry date, and associated warehouse, creating a comprehensive, interconnected dataset.

By implementing this system, businesses can reduce errors, improve data accessibility, and make informed decisions based on real-time information. It provides a flexible foundation that could be expanded with additional features, such as automated inventory tracking or reporting capabilities, as business needs evolve.

3 Tech Stack

- **Python:** Programming language to connect with MySQL database and create GUI Application.
- **Tkinter:** Python library for creating the graphic user interface (GUI), providing an interactive front-end to users.
- **MySQL:** SQL Database to store, access, update and manage Inventory data.
- **mysql-connector-python:** Python library used to connect MySQL Database with Python for database communication.

4 Code

Python Code:

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector

# Database connection function
def create_connection():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="pass@123",
            database="InventoryManagement"
        )
        return conn
    except mysql.connector.Error as e:
        messagebox.showerror("Connection Error", f"Error
connecting to database: {e}")
        return None


# Main application class
class InventoryManagementApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Inventory Management System")
        self.root.geometry("800x600")
```

```
        # Label and buttons for each table
        tk.Label(self.root, text="Select a Table to Manage",
font=("Arial", 18)).pack(pady=10)

        # Buttons to manage tables
        self.manage_buttons = {
            "Employees": self.manage_employees,
            "Customer Care": self.manage_customer_care,
            "Warehouse": self.manage_warehouse,
            "Providers": self.manage_providers,
            "Products": self.manage_products,
            "Offers": self.manage_offers,
            "Customers": self.manage_customers,
            "Payments": self.manage_payments,
            "Online Payments": self.manage_online,
            "Offline Payments": self.manage_offline
        }

        for table_name, action in self.manage_buttons.items():
            tk.Button(self.root, text=f"Manage {table_name}",
command=action, width=20).pack(pady=5)

    def manage_employees(self):
        self.manage_table("employee", ["e_id", "e_name",
"e_age", "e_experience"])

    def manage_customer_care(self):
        self.manage_table("customer_care", ["cc_id",
"cc_contact", "cc_location", "e_id"])

    def manage_warehouse(self):
        self.manage_table("warehouse", ["w_no", "w_capacity",
"w_location"])

    def manage_providers(self):
        self.manage_table("provider", ["pr_id", "pr_type",
"pr_address"])

    def manage_products(self):
        self.manage_table("products", ["p_id", "p_price",
"p_expiry"])

    def manage_offers(self):
        self.manage_table("offers", ["o_no", "o_name",
"o_type"])
```

```
    def manage_customers(self):
        self.manage_table("customer", ["c_id", "c_name",
"c_contact", "c_age"])

    def manage_payments(self):
        self.manage_table("payment", ["py_id", "py_time",
"py_date", "py_mode"])

    def manage_online(self):
        self.manage_table("online", ["on_upi", "on_credit",
"on_debit"])

    def manage_offline(self):
        self.manage_table("offline", ["off_cod"])

    def manage_table(self, table_name, columns):
        # Create a new window for managing the table
        table_window = tk.Toplevel(self.root)
        table_window.title(f"Manage {table_name}")

        # Display all data in a Treeview above the action
buttons
        tree_frame = tk.Frame(table_window)
        tree_frame.pack(pady=10)

        tree = ttk.Treeview(tree_frame, columns=columns,
show="headings")
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=120)
        tree.pack(fill=tk.BOTH, expand=True)

        # Action buttons (Insert, Update, Delete, Retrieve)
        action_frame = tk.Frame(table_window)
        action_frame.pack(pady=10)

        tk.Button(action_frame, text="Insert", command=lambda:
self.show_insert_fields(table_window, table_name, columns,
tree)).pack(side=tk.LEFT, padx=10)
        tk.Button(action_frame, text="Update", command=lambda:
self.show_update_fields(table_window, table_name, columns,
tree)).pack(side=tk.LEFT, padx=10)
```

```
        tk.Button(action_frame, text="Delete", command=lambda:
self.show_delete_fields(table_window, table_name, columns,
tree)).pack(side=tk.LEFT, padx=10)
        tk.Button(action_frame, text="Retrieve", command=lambda:
self.show_retrieve_fields(table_window, table_name, columns,
tree)).pack(side=tk.LEFT, padx=10)

        # Load data when window is opened
        self.view_all_data(tree, table_name)

    def show_insert_fields(self, table_window, table_name,
columns, tree):
        self.clear_fields(table_window)  # Clear previous fields
if any

        insert_frame = tk.Frame(table_window)
        insert_frame.pack(pady=10)

        entries = []
        for col in columns:
            tk.Label(insert_frame, text=col).pack(side=tk.LEFT,
padx=5)
            entry = tk.Entry(insert_frame)
            entry.pack(side=tk.LEFT, padx=5)
            entries.append(entry)

        def add_record():
            values = [entry.get() for entry in entries]
            conn = create_connection()
            if conn:
                try:
                    cursor = conn.cursor()
                    cursor.execute(f"INSERT INTO {table_name}
({', '.join(columns)}) VALUES ({', '.join(['%s'] *
len(columns))})", values)
                    conn.commit()
                    conn.close()
                    messagebox.showinfo("Success", "Record added
successfully.")
                    self.view_all_data(tree, table_name)  #
Refresh data
                    self.clear_fields(table_window)  # Clear
fields after insertion
                    insert_frame.destroy()  # Remove insert
fields after operation
```

```
                except mysql.connector.IntegrityError as err:
                    messagebox.showerror("Error", f"Duplicate
entry or integrity error: {err}")
                except Exception as e:
                    messagebox.showerror("Error", f"An error
occurred: {e}")

        tk.Button(insert_frame, text="Add",
command=add_record).pack(side=tk.LEFT, padx=5)

    def show_update_fields(self, table_window, table_name,
columns, tree):
        self.clear_fields(table_window)  # Clear previous fields
if any

        update_frame = tk.Frame(table_window)
        update_frame.pack(pady=10)

        entries = []
        for col in columns:
            tk.Label(update_frame, text=col).pack(side=tk.LEFT,
padx=5)
            entry = tk.Entry(update_frame)
            entry.pack(side=tk.LEFT, padx=5)
            entries.append(entry)

        def update_record():
            values = [entry.get() for entry in entries]
            conn = create_connection()
            if conn:
                try:
                    update_stmt = f"UPDATE {table_name} SET " +
", ".join([f"{col} = %s" for col in columns[1:]]) + f" WHERE
{columns[0]} = %s"
                    cursor = conn.cursor()
                    cursor.execute(update_stmt, values[1:] +
[values[0]])
                    conn.commit()
                    conn.close()
                    messagebox.showinfo("Success", "Record
updated successfully.")
                    self.view_all_data(tree, table_name)  #
Refresh data
                    self.clear_fields(table_window)  # Clear
fields after update
```

```
                    update_frame.destroy()  # Remove update
fields after operation
                except Exception as e:
                    messagebox.showerror("Error", f"An error
occurred: {e}")

        tk.Button(update_frame, text="Update",
command=update_record).pack(side=tk.LEFT, padx=5)

    def show_delete_fields(self, table_window, table_name,
columns, tree):
        self.clear_fields(table_window)  # Clear previous fields
if any

        delete_frame = tk.Frame(table_window)
        delete_frame.pack(pady=10)

        tk.Label(delete_frame, text=f"Enter {columns[0]} to
Delete").pack(pady=5)
        delete_entry = tk.Entry(delete_frame)
        delete_entry.pack(pady=5)

        def delete_record():
            record_id = delete_entry.get()
            if not record_id:
                messagebox.showerror("Error", "Please enter an
ID to delete.")
                return

            conn = create_connection()
            if conn:
                try:
                    cursor = conn.cursor()
                    cursor.execute(f"DELETE FROM {table_name}
WHERE {columns[0]} = %s", (record_id,))
                    conn.commit()

                    if cursor.rowcount == 0:
                        messagebox.showinfo("Not Found", "No
record found with the provided ID.")
                    else:
                        messagebox.showinfo("Success", f"Record
with {columns[0]} = {record_id} deleted successfully.")
                    conn.close()
```

```
                    self.view_all_data(tree, table_name)  #
Refresh data
                    self.clear_fields(table_window)  # Clear
fields after deletion
                    delete_frame.destroy()  # Remove delete
fields after operation
                except Exception as e:
                    messagebox.showerror("Error", f"An error
occurred: {e}")

        tk.Button(delete_frame, text="Delete",
command=delete_record).pack(pady=5)

    def show_retrieve_fields(self, table_window, table_name,
columns, tree):
        self.clear_fields(table_window)  # Clear previous fields
if any

        retrieve_frame = tk.Frame(table_window)
        retrieve_frame.pack(pady=10)

        tk.Label(retrieve_frame, text=f"Enter {columns[0]} to
Retrieve").pack(pady=5)
        retrieve_entry = tk.Entry(retrieve_frame)
        retrieve_entry.pack(pady=5)

        def retrieve_record():
            record_id = retrieve_entry.get()
            if not record_id:
                messagebox.showerror("Error", "Please enter an
ID to retrieve.")
                return

            conn = create_connection()
            if conn:
                try:
                    cursor = conn.cursor()
                    cursor.execute(f"SELECT * FROM {table_name}
WHERE {columns[0]} = %s", (record_id,))
                    record = cursor.fetchone()

                    if record:
                        record_str = "\n".join([f"{col}: {val}"
for col, val in zip(columns, record)])
```

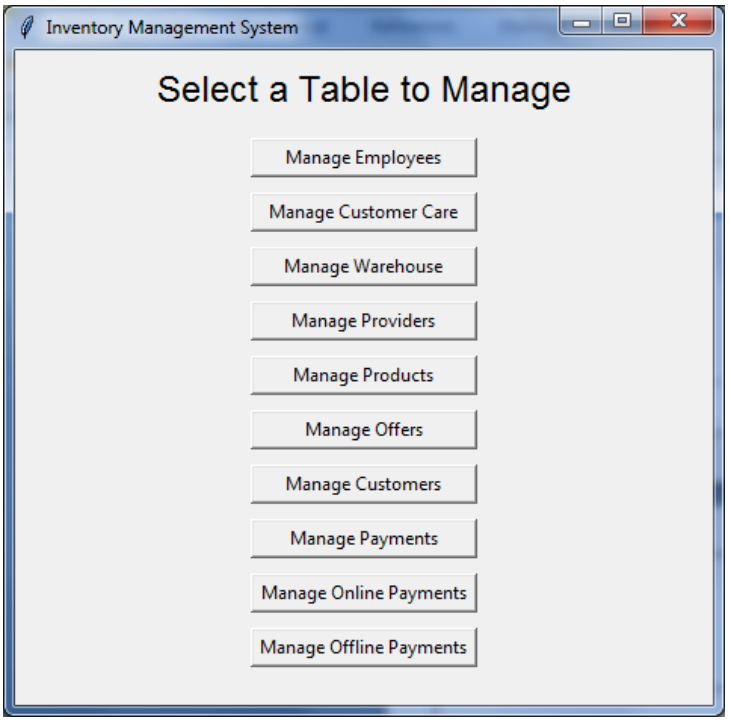
```
                        messagebox.showinfo("Record Found",
f"Record:\n{record_str}")
                    else:
                        messagebox.showinfo("Not Found", "No
record found with the provided ID.")
                    conn.close()
                    self.clear_fields(table_window)  # Clear
fields after retrieve
                    retrieve_frame.destroy()  # Remove retrieve
fields after operation
                except Exception as e:
                    messagebox.showerror("Error", f"An error
occurred: {e}")

        tk.Button(retrieve_frame, text="Retrieve",
command=retrieve_record).pack(pady=5)

    def view_all_data(self, tree, table_name):
        conn = create_connection()
        if conn:
            try:
                cursor = conn.cursor()
                cursor.execute(f"SELECT * FROM {table_name}")
                records = cursor.fetchall()
                for row in tree.get_children():
                    tree.delete(row)
                for record in records:
                    tree.insert("", "end", values=record)
                conn.close()
            except Exception as e:
                messagebox.showerror("Error", f"An error
occurred: {e}")

    def clear_fields(self, table_window):
        for widget in table_window.winfo_children():
            if isinstance(widget, tk.Entry):
                widget.delete(0, tk.END)


# Run the application
root = tk.Tk()
app = InventoryManagementApp(root)
root.mainloop()
```

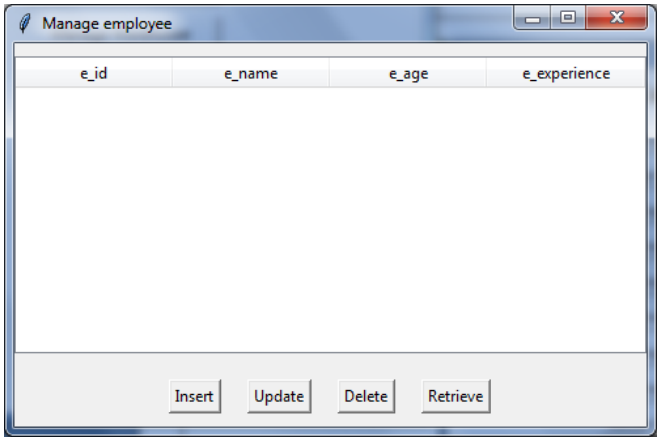
MySQL Code:

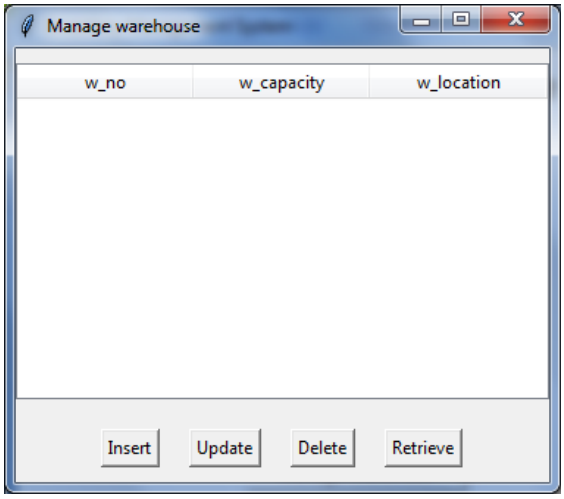
```
create database InventoryManagement;
use InventoryManagement;

create table employee
(
    e_id int primary key,
    e_name varchar(50) not null,
    e_age int,
    e_experience int
);

create table customer_care
(
    cc_id int primary key,
    cc_contact int,
    cc_location varchar(100),
    e_id int,
    foreign key (e_id) references employee(e_id)
);

create table warehouse
(
    w_no int primary key,
    w_capacity int,
    w_location varchar(100)
);

create table provider
(
    pr_id int primary key,
    pr_type varchar(40),
    pr_address varchar(100)
);

create table products
(
    p_id int primary key,
    p_price int,
    p_expiry varchar(20)
);

create table offers
(
```

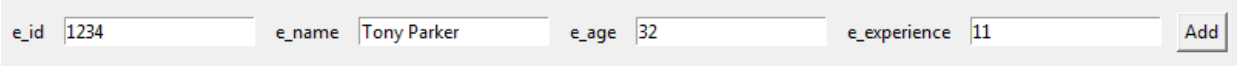
```
    o_no int primary key,
    o_name varchar(50),
    o_type varchar(20)
);

create table customer
(
    c_id int primary key,
    c_name varchar(50) not null,
    c_contact int,
    c_age int
);

create table payment
(
    py_id int primary key,
    py_time varchar(10),
    py_date varchar(16),
    py_mode varchar(50) not null
);

create table online
(
    on_upi varchar(80),
    on_credit varchar(80),
    on_debit varchar(80)
);

create table offline
(
    off_cod varchar(80)
);
```

5 Output









e_id	e_name	e_age	e_experience
1234	Tony Parker	32	11

e_id 1234 e_name Tony Parker e_age 33 e_experience 12 Update

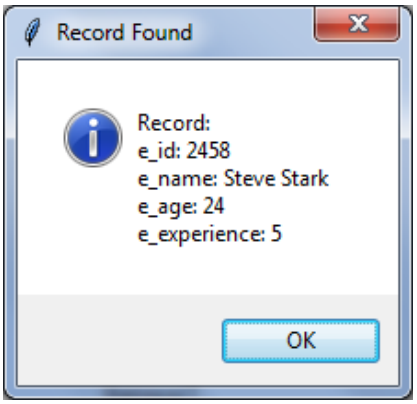
e_id	e_name	e_age	e_experience
1234	Tony Parker	33	12

e_id	e_name	e_age	e_experience
1234	Tony Parker	33	12
2458	Steve Stark	24	5

Enter e_id to Retrieve

2458

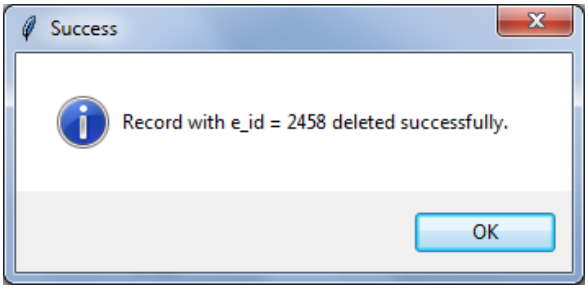
Retrieve



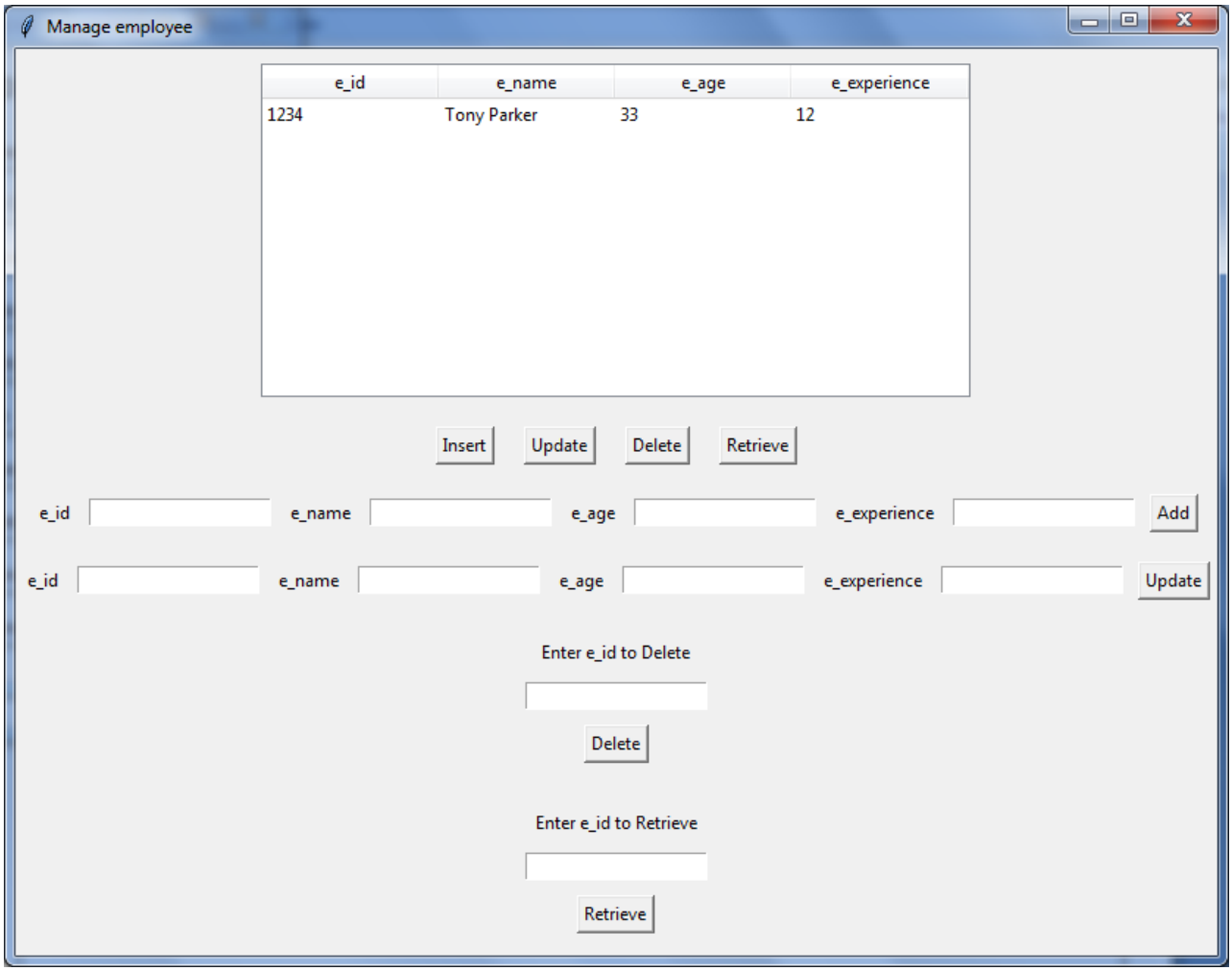
Enter e_id to Delete

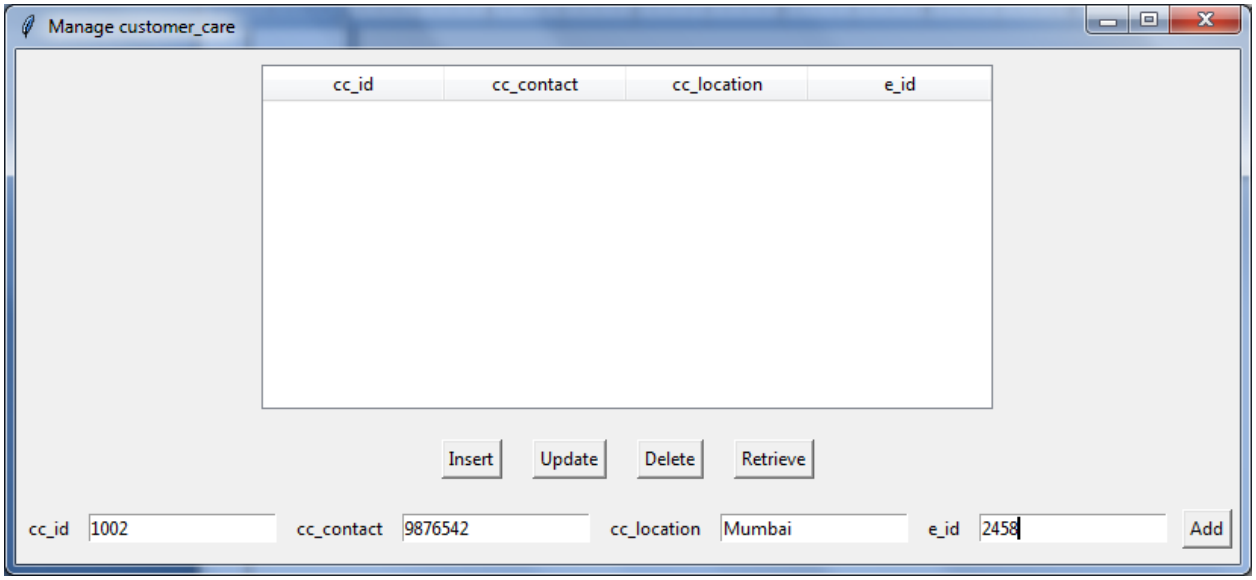
2458

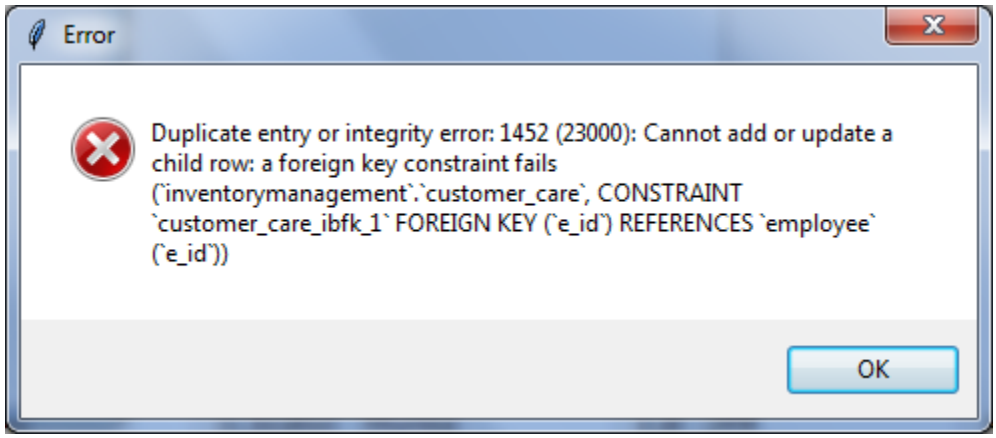
Delete

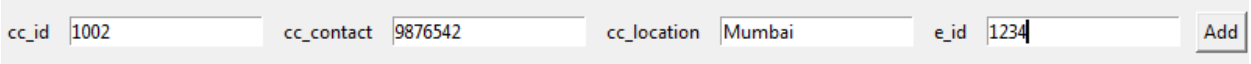


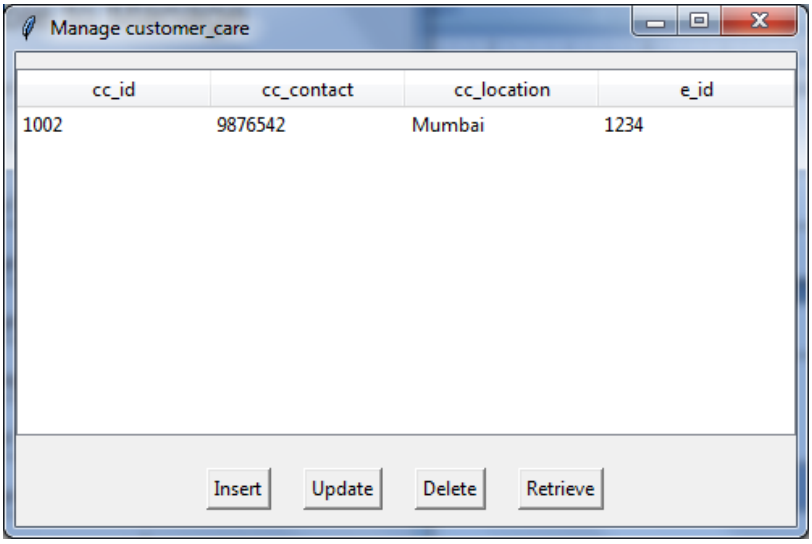
e_id	e_name	e_age	e_experience
1234	Tony Parker	33	12


Manage employee
e_id
e_name
e_age
e_experience
1234
Tony Parker
33
12
Insert
Update
Delete
Retrieve
e_id
e_name
e_age
e_experience
Add
e_id
e_name
e_age
e_experience
Update
Enter e_id to Delete
Delete
Enter e_id to Retrieve
Retrieve


Manage customer_care
cc_id
cc_contact
cc_location
e_id
Insert
Update
Delete
Retrieve
cc_id
1002
cc_contact
9876542
cc_location
Mumbai
e_id
2458
Add


Error
Duplicate entry or integrity error: 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`inventorymanagement`.`customer_care`, CONSTRAINT `customer_care_ibfk_1` FOREIGN KEY (`e_id`) REFERENCES `employee` (`e_id`))
OK


cc_id
1002
cc_contact
9876542
cc_location
Mumbai
e_id
1234
Add


Manage customer_care
cc_id
cc_contact
cc_location
e_id
1002
9876542
Mumbai
1234
Insert
Update
Delete
Retrieve