





(Affiliated to Central Board of Secondary Education, New Delhi)

(Chettinad House, R.A.Puram, Chennai – 600 028)

# COMPUTER SCIENCE

Certified to be the Bonafide Record of work done by

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of Std XII Sec \_\_\_\_

in the Computer Science Lab of the CHETTINAD VIDYASHRAM,

CHENNAI, during the year 2021 – 2022.

Date: Teacher-in-charge

## REGISTER NO. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### Submitted for All India Senior Secondary Practical Examination in

#### Computer Science held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_at

#### Chettinad Vidyashram, Chennai – 600 028.

#### Principal Internal Examiner External Examiner

**ACKNOWLEDGEMENT**

I would like to express my sincere thanks to

Meena Aunty, Principal Mrs. S.Amudha lakshmi

for their encouragement and support to work on

this Project. I am grateful to my computer science

teacher UMA MAGESWARI R and to the computer science department for the constant guidance and support to complete the project

***TABLE OF CONTENT***

|  |  |  |
| --- | --- | --- |
| S.No | Index | Page No |
| 1. | Overview of Python | 5 |
| 2. | Project Description | 6 |
| 3. | Functions Used | 8 |
| 4. | Tables used | 10 |
| 5. | Source Code | 23 |
| 6. | Sample Output | 79 |
| 7. | Conclusion | 92 |
| 8. | Bibiliography | 93 |

**OVERVIEW OF PYTHON**

Python is a **computer programming language often used to build websites and software, automate tasks, and conduct data analysis**. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

It is a high-level, **interpreted, interactive and object-oriented scripting** language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

1. **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
2. **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
3. **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
4. **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python's features include −

* **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
* **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
* **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* **Databases** − Python provides interfaces to all major commercial databases.
* **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* **Scalable** − Python provides a better structure and support for large programs than shell scripting.

**PROJECT DESCRIPTION**

Our project titled “CVTERIA” provides a GUI based system to efficiently manage a cafeteria. We have used a combination of Python and MySQL for the back-end code and have designed a simple yet robust front-end GUI using Tkinter.

This application allows both the staff at a cafeteria and the cafeteria manager to manage records of the customers digitally. It includes features like adding customers, updating customer details, deleting customer details, and ability to view the list of customers including different sorting capabilities. For security purposes we have built a simple yet powerful password validation function. Our appropriate pop-up messages for various scenarios helps the user understand what he or she has done to the database.

The application has multiple validations at user enterable fields and data base level exceptions. We have mapped all the errors with user friendly messages.

The system runs without any glitches at an optimal speed. Being an open-source application, this is a cost-effective solution for small cafeterias.

**FUNCTIONS USED**

|  |  |  |  |
| --- | --- | --- | --- |
| Functions Used | | Function Description | |
| connect() | | Used to connect mysql to python | |
| setup() | | To create tables and declare variables | |
| seed() | | To create data inside each table, like menu etc. | |
| customer\_create() | | To create customer | |
| customer\_update() | | To update customer | |
| customer\_delete() | | To delete customer | |
| bill\_delete() | | To delete bill | |
| order\_delete() | | To delete an order | |
| customer\_get\_all() | | To fetch all customers from database | |
| customer\_get\_by\_id() | To get customer information for a given customer ID | |
| order\_header\_create() | To create order header | |
| order\_header\_update() | To update order header | |
| order\_detail\_update() | To update order details | |
| order\_detail\_create() | To create order details | |
| order\_header\_get\_all() | To fetch all order headers from the database | |
| order\_header\_get\_upaid() | To get order header on paying the bill | |
| order\_header\_get\_by\_id() | To get order header by order ID | |
| order\_detail\_get\_all() | To fetch all order details from the database | |
| order\_detail\_get\_by\_id() | To get order details by order ID. | |

|  |  |
| --- | --- |
| order\_detail\_delete\_by\_id() | To delete order details based on given order ID. |
| item\_get\_all() | To Get all menu items from the database |
| item\_get\_by\_category\_id() | To get all menu items from the database based on category ID. |
| table\_get\_all() | To get all tables from the database |
| table\_reservation\_get\_all() | To get all table reservations from the database |
| table\_reservation\_get\_by\_id() | To get table reservations from the database based on ID |
| table\_reservation\_create() | To create a new table reservation |
| table\_reservation\_update() | To update table reservation details |
| table\_reservation\_delete() | To delete table reservations |
| payment\_mode\_get\_all() | To get all payment modes from the database. |

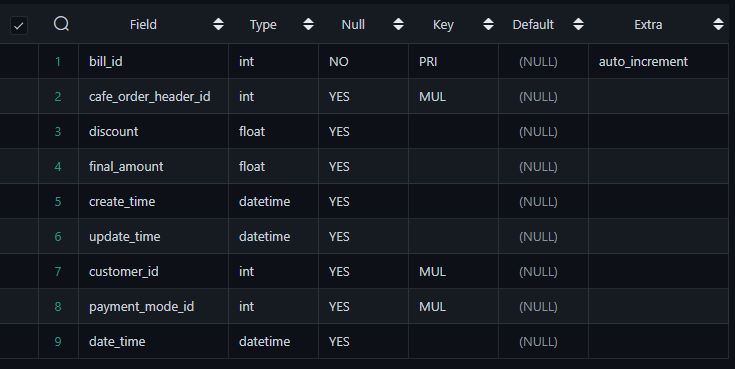
|  |  |
| --- | --- |
| bill\_create() | To create a new bill. |
| bill\_update() | To update bill details. |
| Bill\_get\_all() | To get all bills from the database |
| bill\_get\_by\_id() | To get all bills from the database based on ID. |
| bill\_get\_buy\_order\_header\_id() | To get all bills from database based on order header ID |
| bill\_get\_buy\_customer\_id() | To get all bills from database based on customer ID |
| category\_get\_all() | To get all categories from the database |

**TABLES USED**

**BILL**

Description:

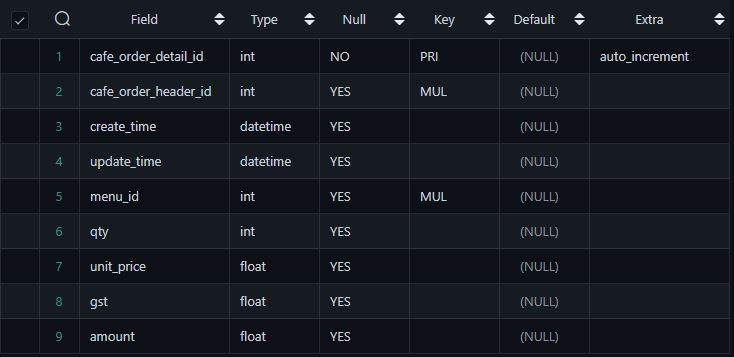
The bill table is used to store the details of every bill. It has attributes like bill\_id, discount,final\_amount,payment\_mode\_id,date\_time etc. This table can be accessed only by the owner of the café in order to manage the cafeteria.

Structure:

**Café\_order\_detail**

Description:

The table contains the details of every order that has been placed at our café. It has attributes like cafe\_order\_detail\_id, menu\_id, qty, unit\_price etc.

Structure:

**Cafe\_order\_header**

Description:

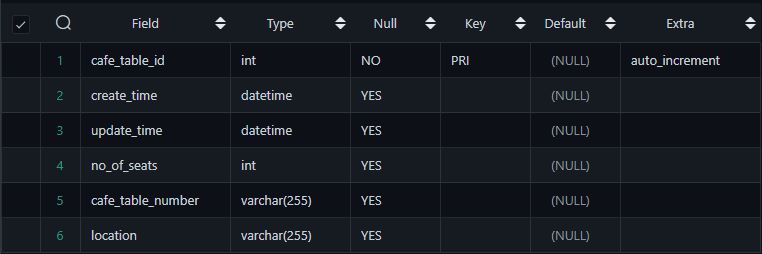
This table stores details of order headers and total amount of each order. It has attributes like café\_table\_id, total\_amount etc.

Structure:

**Cafe\_table**

Description:

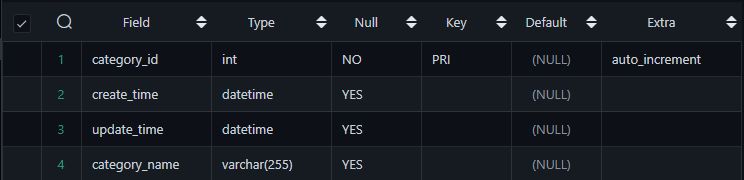
This table is used to store the details of the tables at our café. It has attributes like café\_table\_id,café\_table\_number etc.

Structure:

**Category**

Description:

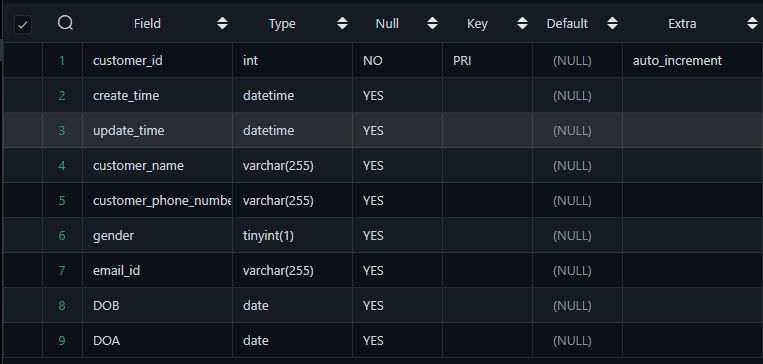
This table stores the categories to which each menu item belongs to. It has attributes like category\_id, category\_name etc.

Structure:

**Customer**

Description:

This table is used to store the details of the customers who have created accounts at our café.It has attrbutes like customer\_id,customer\_name etc.

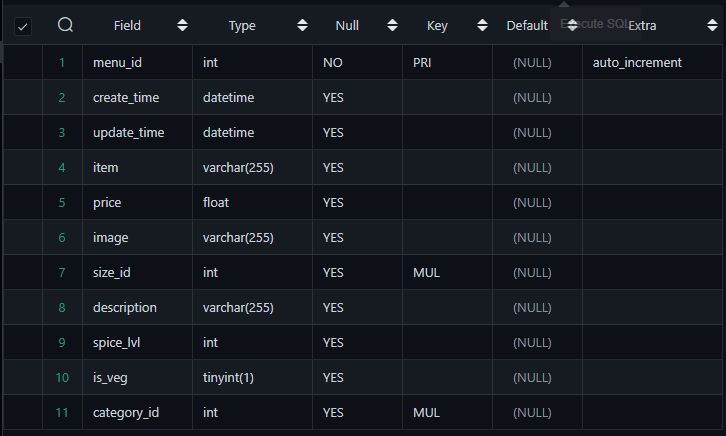
Structure:

**Menu**

Description:

This table stores all our menu items. It has attributes like menu\_id, price,item etc.

Structure:

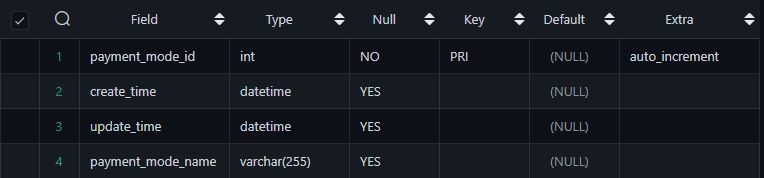


**Payment mode**

Description:

This table stores the different payment modes our café offers. It has attributes like payment\_mode\_id, payment\_mode\_name etc.

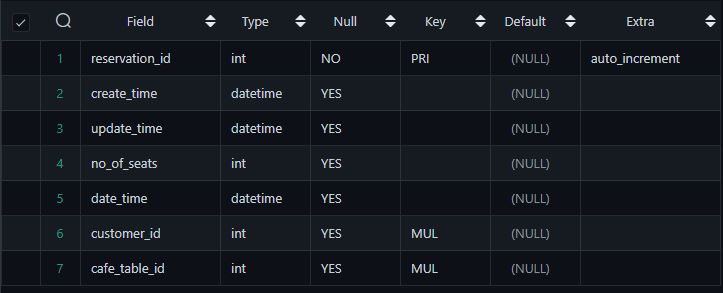
Structure:



**Reservation**

Description:

This table stores all the reservations made by customers at our café. It has attributes like reservation\_id, no\_of\_seats etc.

Structure:

**Size**

Description:This tables stores the different sizes that our café offers for each item. It has attributes like size\_id, size\_name..

Structure:

**db.py**

import mysql.connector as bot

from model.category import Category

from model.customer import Customer

from model.order\_header import OrderHeader

from model.order\_detail import OrderDetail

from model.item import Item

from model.table import Table

from model.table\_reservation import TableReservation

from model.payment\_mode import PaymentMode

from model.bill import Bill

mydb = None

class Database():

def connect(self):

mydb = bot.connect(

host = 'localhost',

user = 'root',

password = 'Tiger@123',

database = 'cvteria'

)

if mydb:

print('connected to the db successfully!!')

else:

print('could not connect to DB')

return mydb

def setup(self):

#CREATE DATBASE

con = bot.connect(host="localhost",user="root",password="Tiger@123",charset='utf8')

cur = con.cursor()

cur.execute("""

CREATE DATABASE IF NOT EXISTS cvteria

""")

mydb = self.connect()

mycursor = mydb.cursor()

#CREATE TABLES

#size

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

size(

size\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

size\_name VARCHAR(255)

);

""")

#category

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

category(

category\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

category\_name VARCHAR(255)

);

""")

#payment\_mode

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

payment\_mode(

payment\_mode\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

payment\_mode\_name VARCHAR(255)

);

""")

#menu

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

menu(

menu\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

item VARCHAR(255),

price FLOAT,

image VARCHAR(255),

size\_id INT, INDEX size\_id (size\_id),CONSTRAINT fk\_size\_id FOREIGN KEY (size\_id) REFERENCES size(size\_id) ,

description VARCHAR(255),

spice\_lvl INT,

is\_veg BOOLEAN,

category\_id INT, INDEX category\_id (category\_id),CONSTRAINT fk\_category\_id FOREIGN KEY (category\_id) REFERENCES category(category\_id)

);

""")

#cafe\_table

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

cafe\_table(

cafe\_table\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

no\_of\_seats INT,

cafe\_table\_number VARCHAR(255),

location VARCHAR(255)

);

""")

#customer

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

customer(

customer\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

customer\_name VARCHAR(255),

customer\_phone\_number VARCHAR(255),

gender BOOLEAN,

email\_id VARCHAR(255),

DOB DATE,

DOA DATE

);

""")

#reservation

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

reservation(

reservation\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

no\_of\_seats INT,

date\_time DATETIME,

customer\_id INT, INDEX customer\_id (customer\_id),CONSTRAINT fk\_customer\_id FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id) ,

cafe\_table\_id INT, INDEX cafe\_table\_id (cafe\_table\_id),CONSTRAINT fk\_cafe\_table\_id FOREIGN KEY (cafe\_table\_id) REFERENCES cafe\_table(cafe\_table\_id)

);

""")

#cafe\_order\_header

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

cafe\_order\_header(

cafe\_order\_header\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

cafe\_table\_id INT, INDEX cafe\_table\_id(cafe\_table\_id),CONSTRAINT fk\_cafe\_order\_header\_cafe\_table\_id FOREIGN KEY (cafe\_table\_id) REFERENCES cafe\_table(cafe\_table\_id) ,

total\_amount FLOAT

);

""")

#cafe\_order\_detail

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

cafe\_order\_detail(

cafe\_order\_detail\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

cafe\_order\_header\_id INT, INDEX cafe\_order\_header\_id(cafe\_order\_header\_id),CONSTRAINT fk\_cafe\_order\_header\_id FOREIGN KEY (cafe\_order\_header\_id) REFERENCES cafe\_order\_header(cafe\_order\_header\_id) ,

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

menu\_id INT, INDEX menu\_id(menu\_id),CONSTRAINT fk\_cafe\_order\_detail\_menu\_id FOREIGN KEY (menu\_id) REFERENCES menu(menu\_id) ,

qty INT,

unit\_price float,

gst float,

amount FLOAT

);

""")

#bill

mycursor.execute("""

CREATE TABLE IF NOT EXISTS

bill(

bill\_id int NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT 'Primary Key',

cafe\_order\_header\_id INT, INDEX cafe\_order\_header\_id(cafe\_order\_header\_id),CONSTRAINT fk\_cafe\_order\_header\_id\_bill FOREIGN KEY (cafe\_order\_header\_id) REFERENCES cafe\_order\_header(cafe\_order\_header\_id) ,

discount float,

final\_amount float,

create\_time DATETIME COMMENT 'Create Time',

update\_time DATETIME COMMENT 'Update Time',

customer\_id INT, INDEX customer\_id (customer\_id),CONSTRAINT fk\_bill\_customer\_id FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id) ,

payment\_mode\_id INT, INDEX payment\_mode\_id (payment\_mode\_id),CONSTRAINT fk\_payment\_mode\_id FOREIGN KEY (payment\_mode\_id) REFERENCES payment\_mode(payment\_mode\_id) ,

date\_time DATETIME

);

""")

mydb.commit()

def seed(self):

mydb = self.connect()

mycursor = mydb.cursor()

#INSERT DEFAULT RECORDS

#size

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Demi',now(),now());")

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Short',now(),now());")

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Tall',now(),now());")

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Venti',now(),now());")

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Grande',now(),now());")

mycursor.execute("insert into size(size\_name,create\_time,update\_time) VALUES('Trenta',now(),now());")

mydb.commit()

#category

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Hot Beverages',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Cold Beverages',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Quick Bites',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Sandwiches',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Desserts',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Donuts',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('English Breakfast',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Soups',now(),now());")

mycursor.execute("insert into category(category\_name,create\_time,update\_time) VALUES('Grab-n-Go',now(),now());")

mydb.commit()

#payment\_mode

mycursor.execute("insert into payment\_mode(payment\_mode\_name,create\_time,update\_time) VALUES('Cash',now(),now());")

mycursor.execute("insert into payment\_mode(payment\_mode\_name,create\_time,update\_time) VALUES('Card',now(),now());")

mycursor.execute("insert into payment\_mode(payment\_mode\_name,create\_time,update\_time) VALUES('UPI',now(),now());")

mydb.commit()

#menu

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Cappucino', 399, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Yamamoto Tea', 299, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Miguel Drink', 249, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Hot Chocolate', 349, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Frappe', 399, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Iced Tea', 299, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Oreo Milkshake', 249, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Blue Lagoon Mojito', 199, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Corn Cheese Balls', 349, null, null, 1, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Potato Wedges', 199, null, null, 1, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Peri Peri Fries', 299, null, null, 2, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Merry Mellow Chicken Bites', 249, null, null, 3, False, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('White Italian Tomato Sandwich', 249, null, null, 1, True, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Mexican Grilled Cheese Sandwich', 299, null, null, 2, True, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Chicken Tikka Sandwich', 249, null, null, 3, False, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Mutton Sheekh Sandwich', 299, null, null, 3, False, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Tiramisu', 249, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Sizzling Chocolate Brownie', 299, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Cheesecake', 149, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Gelato Ice Cream', 149, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Oreo Cookies and Creme Donut', 149, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Glazed Blueberry Donut', 249, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Reeses Classic Donut', 249, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Chocolate Iced Raspberry Donut', 149, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Croissant', 99, null, null, 0, True, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Bagel', 99, null, null, 0, True, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Sunny side up', 149, null, null, 1, False, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Bacon and Mushroom Peas', 249, null, null, 1, False, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Hot N Sour Soup', 99, null, null, 1, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Sweet Corn Soup', 99, null, null, 1, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Cream of Mushroom Soup', 99, null, null, 0, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Clear Chicken Soup', 99, null, null, 1, False, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Veg Frankie', 199, null, null, 1, True, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Chicken Frankie', 199, null, null, 2, False, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Choco Chip Cookies', 99, null, null, 0, True, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size\_id, description, spice\_lvl, is\_veg, category\_id, create\_time,update\_time) VALUES('Spicy Chips', 99, null, null, 2, True, 9, now(),now());")

mydb.commit()

#cafe\_table

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 01', 2, '1st Floor, North East', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 02', 4, '1st Floor, South', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 03', 6, '1st Floor, Center', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 04', 2, '2nd Floor, North East', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 05', 4, '2nd Floor, South', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 06', 6, '2nd Floor, Center', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 07', 2, '3rd Floor, North East', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 08', 4, '3rd Floor, South', now(),now());")

mycursor.execute("insert into cafe\_table(cafe\_table\_number, no\_of\_seats, location, create\_time,update\_time) VALUES('Table 09', 6, '3rd Floor, Center', now(),now());")

mydb.commit()

def customer\_create(self, cust=Customer):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "insert into customer(customer\_name,customer\_phone\_number, gender, email\_id, DOB, DOA, create\_time,update\_time) VALUES (%s, %s, %s, %s, %s, %s, now(), now())"

val = (cust.customer\_name, cust.phone\_number, cust.gender, cust.email\_id, cust.DOB, cust.anniversary\_date)

mycursor.execute(sql, val)

mydb.commit()

print('customer created ')

def customer\_update(self, cust=Customer):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "update customer set customer\_name=%s,customer\_phone\_number=%s, gender=%s, email\_id=%s, DOB=%s, DOA=%s, update\_time=now() where customer\_id=%s"

val = (cust.customer\_name, cust.phone\_number, cust.gender, cust.email\_id, cust.DOB, cust.anniversary\_date, cust.customer\_id)

mycursor.execute(sql, val)

mydb.commit()

print('customer updated')

def customer\_delete(self, customer\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "delete from customer where customer\_id=" + str(customer\_id)

mycursor.execute(sql)

mydb.commit()

print('customer updated')

def bill\_delete(self, bill\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "delete from bill where bill\_id=" + str(bill\_id)

mycursor.execute(sql)

mydb.commit()

print('bill updated')

def order\_delete(self, cafe\_order\_header\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "delete from cafe\_order\_detail where cafe\_order\_header\_id=" + str(cafe\_order\_header\_id)

mycursor.execute(sql)

sql = "delete from cafe\_order\_header where cafe\_order\_header\_id=" + str(cafe\_order\_header\_id)

mycursor.execute(sql)

mydb.commit()

print('order updated')

def customer\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select customer\_id, customer\_name,customer\_phone\_number, gender, email\_id, DOB, DOA, create\_time,update\_time from customer order by customer\_id desc")

list\_of\_customer = mycursor.fetchall()

cust\_list = []

for data in list\_of\_customer:

cust = Customer()

cust.customer\_id = data[0]

cust.customer\_name = data[1]

cust.phone\_number = data[2]

cust.gender = data[3]

cust.email\_id = data[4]

cust.DOB = data[5]

cust.anniversary\_date = data[6]

cust\_list.append(cust)

print(f"|{cust.customer\_id:4}")

return cust\_list

def customer\_get\_by\_id(self, customer\_id):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select customer\_id, customer\_name,customer\_phone\_number, gender, email\_id, DOB, DOA, create\_time,update\_time from customer where customer\_id=" + str(customer\_id))

list\_of\_customer = mycursor.fetchall()

cust\_list = []

for data in list\_of\_customer:

cust = Customer()

cust.customer\_id = data[0]

cust.customer\_name = data[1]

cust.phone\_number = data[2]

cust.gender = data[3]

cust.email\_id = data[4]

cust.DOB = data[5]

cust.anniversary\_date = data[6]

cust\_list.append(cust)

print(f"|{cust.customer\_id:4}")

return cust\_list

def order\_header\_create(self, ord=OrderHeader):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "insert into cafe\_order\_header(cafe\_table\_id, total\_amount, create\_time,update\_time) VALUES (%s, %s, now(), now())"

val = (ord.table\_id, ord.total\_amount)

mycursor.execute(sql, val)

mydb.commit()

print('Order created ')

return mycursor.lastrowid

def order\_header\_update(self, ord=OrderHeader):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "update cafe\_order\_header set total\_amount = %s, update\_time=now() where cafe\_order\_header\_id=%s"

val = (ord.total\_amount, ord.order\_header\_id)

mycursor.execute(sql, val)

mydb.commit()

print('Order updated')

return mycursor.lastrowid

def order\_detail\_update(self, ord=OrderDetail):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "update cafe\_order\_detail set menu\_id=%s, qty=%s, unit\_price=%s, gst=%s, amount=%s, update\_time=now() where cafe\_order\_detail\_id = %s"

val = (ord.item\_id, ord.qty, ord.price, ord.gst, ord.amount, ord.order\_detail\_id)

mycursor.execute(sql, val)

mydb.commit()

print('Order detail updated')

def order\_detail\_create(self, ord=OrderDetail):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "insert into cafe\_order\_detail(cafe\_order\_header\_id, menu\_id, qty, unit\_price, gst, amount, create\_time,update\_time) VALUES (%s, %s, %s, %s, %s, %s, now(), now())"

val = (ord.order\_header\_id, ord.item\_id, ord.qty, ord.price, ord.gst, ord.amount)

mycursor.execute(sql, val)

mydb.commit()

print('Order created ')

def order\_header\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select cafe\_order\_header\_id, c.cafe\_table\_id, cafe\_table\_number, total\_amount, c.create\_time from cafe\_order\_header c inner join cafe\_table t on c.cafe\_table\_id = t.cafe\_table\_id order by cafe\_order\_header\_id desc")

list = mycursor.fetchall()

output\_list = []

for data in list:

ord = OrderHeader()

ord.order\_header\_id = data[0]

ord.table\_id = data[1]

ord.table\_number = data[2]

ord.total\_amount = data[3]

ord.create\_time = data[4]

output\_list.append(ord)

print(f"|{ord.order\_header\_id:4}")

return output\_list

def order\_header\_get\_upaid(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("""select c.cafe\_order\_header\_id, c.cafe\_table\_id, cafe\_table\_number, total\_amount, c.create\_time

from cafe\_order\_header c

left outer join bill b on c.cafe\_order\_header\_id = b.cafe\_order\_header\_id

inner join cafe\_table t on c.cafe\_table\_id = t.cafe\_table\_id

where b.bill\_id is null

order by cafe\_order\_header\_id desc""")

list = mycursor.fetchall()

output\_list = []

for data in list:

ord = OrderHeader()

ord.order\_header\_id = data[0]

ord.table\_id = data[1]

ord.table\_number = data[2]

ord.total\_amount = data[3]

ord.create\_time = data[4]

output\_list.append(ord)

print(f"|{ord.order\_header\_id:4}")

return output\_list

def order\_header\_get\_by\_id(self, cafe\_order\_header\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "select cafe\_order\_header\_id, c.cafe\_table\_id, cafe\_table\_number, total\_amount, c.create\_time from cafe\_order\_header c inner join cafe\_table t on c.cafe\_table\_id = t.cafe\_table\_id where c.cafe\_order\_header\_id = '" + str(cafe\_order\_header\_id) + "' order by cafe\_order\_header\_id desc"

mycursor.execute(sql)

list = mycursor.fetchall()

output\_list = []

for data in list:

ord = OrderHeader()

ord.order\_header\_id = data[0]

ord.table\_id = data[1]

ord.table\_number = data[2]

ord.total\_amount = data[3]

ord.create\_time = data[4]

output\_list.append(ord)

print(f"|{ord.order\_header\_id:4}")

return output\_list

def order\_detail\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select cafe\_order\_detail\_id, cafe\_order\_header\_id, c.cafe\_table\_id, cafe\_table\_number, c.menu\_id, item, qty, unit\_price, gst, amount, c.create\_time from cafe\_order\_detail c inner join menu m on c.menu\_id = m.menu\_id inner join cafe\_table t on c.cafe\_table\_id = t.cafe\_table\_id order by cafe\_order\_detail\_id")

list = mycursor.fetchall()

output\_list = []

for data in list:

ord = OrderDetail()

ord.cafe\_order\_detail\_id = data[0]

ord.cafe\_order\_header\_id = data[1]

ord.table\_id = data[2]

ord.table\_number = data[3]

ord.item\_id = data[4]

ord.item = data[5]

ord.qty = data[6]

ord.price = data[7]

ord.gst = data[8]

ord.amount = data[9]

ord.create\_time = data[10]

output\_list.append(ord)

print(f"|{ord.cafe\_order\_detail\_id:4}")

return output\_list

def order\_detail\_get\_by\_id(self, cafe\_order\_header\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "select c.cafe\_order\_detail\_id, c.cafe\_order\_header\_id, c.menu\_id, item, qty, unit\_price, gst, amount, c.create\_time from cafe\_order\_detail c inner join menu m on c.menu\_id = m.menu\_id where c.cafe\_order\_header\_id = '" + str(cafe\_order\_header\_id) + "' order by c.cafe\_order\_detail\_id"

mycursor.execute(sql)

list = mycursor.fetchall()

output\_list = []

for data in list:

ord = OrderDetail()

ord.cafe\_order\_detail\_id = data[0]

ord.cafe\_order\_header\_id = data[1]

ord.item\_id = data[2]

ord.item = data[3]

ord.qty = data[4]

ord.price = data[5]

ord.gst = data[6]

ord.amount = data[7]

ord.create\_time = data[8]

output\_list.append(ord)

print(f"|{ord.cafe\_order\_detail\_id:4}")

return output\_list

def order\_detail\_delete\_by\_id(self, cafe\_order\_header\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "delete from cafe\_order\_detail where cafe\_order\_header\_id = " + str(cafe\_order\_header\_id)

mycursor.execute(sql)

mydb.commit()

print('Order detail deleted:', cafe\_order\_header\_id)

def item\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select menu\_id, item, price, description, spice\_lvl, is\_veg, m.category\_id, category\_name from menu m inner join category c on m.category\_id = c.category\_id")

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = Item()

itm.item\_id = data[0]

itm.item = data[1]

itm.price = data[2]

itm.description = data[3]

itm.spice\_level = data[4]

itm.veg = data[5]

itm.category\_id = data[6]

itm.category = data[7]

output\_list.append(itm)

print(f"|{itm.item\_id:4}")

return output\_list

def item\_get\_by\_category\_id(self, category\_id):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select menu\_id, item, price, description, spice\_lvl, is\_veg, m.category\_id, category\_name from menu m inner join category c on m.category\_id = c.category\_id where m.category\_id=" + str(category\_id))

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = Item()

itm.item\_id = data[0]

itm.item = data[1]

itm.price = data[2]

itm.description = data[3]

itm.spice\_level = data[4]

itm.veg = data[5]

itm.category\_id = data[6]

itm.category = data[7]

output\_list.append(itm)

print(f"|{itm.item\_id:4}")

return output\_list

def table\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select cafe\_table\_id, cafe\_table\_number, no\_of\_seats, location from cafe\_table")

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = Table()

itm.table\_id = data[0]

itm.table\_number = data[1]

itm.number\_of\_seats = data[2]

itm.location = data[3]

output\_list.append(itm)

print(f"|{itm.table\_id:4}")

return output\_list

def table\_reservation\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select reservation\_id, r.no\_of\_seats, date\_time, r.customer\_id, customer\_name, r.cafe\_table\_id, cafe\_table\_number from reservation r inner join customer c on r.customer\_id = c.customer\_id inner join cafe\_table ct on r.cafe\_table\_id = ct.cafe\_table\_id order by reservation\_id desc")

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = TableReservation()

itm.reservation\_id = data[0]

itm.pax = data[1]

itm.datetime = data[2]

itm.customer\_id = data[3]

itm.customer = data[4]

itm.table\_id = data[5]

itm.table\_number = data[6]

output\_list.append(itm)

print(f"|{itm.reservation\_id:4}")

return output\_list

def table\_reservation\_get\_by\_id(self, reservation\_id):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select reservation\_id, r.no\_of\_seats, date\_time, r.customer\_id, customer\_name, r.cafe\_table\_id, cafe\_table\_number from reservation r inner join customer c on r.customer\_id = c.customer\_id inner join cafe\_table ct on r.cafe\_table\_id = ct.cafe\_table\_id where reservation\_id = " + str(reservation\_id))

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = TableReservation()

itm.reservation\_id = data[0]

itm.pax = data[1]

itm.datetime = data[2]

itm.customer\_id = data[3]

itm.customer = data[4]

itm.table\_id = data[5]

itm.table\_number = data[6]

output\_list.append(itm)

print(f"|{itm.reservation\_id:4}")

return output\_list

def table\_reservation\_create(self, reserv=TableReservation):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "insert into reservation(no\_of\_seats,date\_time, customer\_id, cafe\_table\_id, create\_time,update\_time) VALUES (%s, %s, %s, %s, now(), now())"

val = (reserv.pax, reserv.datetime, reserv.customer\_id, reserv.table\_id)

mycursor.execute(sql, val)

mydb.commit()

print('Table Reservation created ')

def table\_reservation\_update(self, reserv=TableReservation):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "update reservation set no\_of\_seats=%s,date\_time=%s, customer\_id=%s, cafe\_table\_id=%s, update\_time=now() where reservation\_id=%s"

val = (reserv.pax, reserv.datetime, reserv.customer\_id, reserv.table\_id, reserv.reservation\_id)

mycursor.execute(sql, val)

mydb.commit()

print('Table Reservation created ')

def table\_reservation\_delete(self, reservation\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "delete from reservation where reservation\_id=" + str(reservation\_id)

mycursor.execute(sql)

mydb.commit()

print('Table Reservation created ')

def payment\_mode\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("select payment\_mode\_id, payment\_mode\_name from payment\_mode")

list = mycursor.fetchall()

output\_list = []

for data in list:

itm = PaymentMode()

itm.payment\_mode\_id = data[0]

itm.payment\_mode\_name = data[1]

output\_list.append(itm)

print(f"|{itm.payment\_mode\_id:4}")

return output\_list

def bill\_create(self, bill=Bill):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "insert into bill(cafe\_order\_header\_id,discount,final\_amount,customer\_id, payment\_mode\_id, date\_time, create\_time,update\_time) VALUES (%s, %s, %s, %s, %s, %s, now(), now())"

val = (bill.cafe\_order\_header\_id, bill.discount,bill.final\_amount,bill.customer\_id, bill.payment\_mode\_id, bill.datetime)

mycursor.execute(sql, val)

mydb.commit()

print('Bill created ')

def bill\_update(self, bill=Bill):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "update bill set customer\_id=%s, payment\_mode\_id=%s,discount=%s,final\_amount=%s, update\_time=now() where bill\_id=%s"

val = (bill.customer\_id, bill.payment\_mode\_id, bill.discount,bill.final\_amount,bill.bill\_id)

mycursor.execute(sql, val)

mydb.commit()

print('Bill created ')

def bill\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("""select b.bill\_id, b.cafe\_order\_header\_id, b.customer\_id, c1.customer\_name, b.payment\_mode\_id, p.payment\_mode\_name,b.discount,b.final\_amount,b.date\_time, c.cafe\_table\_id, c.cafe\_table\_number, ch.total\_amount

from bill b

inner join cafe\_order\_header ch on b.cafe\_order\_header\_id = ch.cafe\_order\_header\_id

inner join cafe\_table c on ch.cafe\_table\_id = c.cafe\_table\_id

inner join customer c1 on b.customer\_id = c1.customer\_id

inner join payment\_mode p on b.payment\_mode\_id = p.payment\_mode\_id

order by b.bill\_id desc

""")

list = mycursor.fetchall()

output\_list = []

for data in list:

bill = Bill()

bill.bill\_id = data[0]

bill.cafe\_order\_header\_id = data[1]

bill.customer\_id = data[2]

bill.customer = data[3]

bill.payment\_mode\_id = data[4]

bill.mode\_of\_payment = data[5]

bill.discount = data[6]

bill.final\_amount = data[7]

bill.datetime = data[8]

bill.table\_id = data[9]

bill.table\_number = data[10]

bill.total\_amount = data[11]

output\_list.append(bill)

print(f"|{bill.bill\_id:4}")

return output\_list

def bill\_get\_by\_id(self, bill\_id):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("""select b.bill\_id, b.cafe\_order\_header\_id, b.customer\_id, c1.customer\_name, b.payment\_mode\_id, p.payment\_mode\_name,b.discount,b.final\_amount, b.date\_time, c.cafe\_table\_id, c.cafe\_table\_number, ch.total\_amount

from bill b

inner join cafe\_order\_header ch on b.cafe\_order\_header\_id = ch.cafe\_order\_header\_id

inner join cafe\_table c on ch.cafe\_table\_id = c.cafe\_table\_id

inner join customer c1 on b.customer\_id = c1.customer\_id

inner join payment\_mode p on b.payment\_mode\_id = p.payment\_mode\_id

where b.bill\_id = """ + str(bill\_id) +

""" order by b.bill\_id desc""")

list = mycursor.fetchall()

output\_list = []

for data in list:

bill = Bill()

bill.bill\_id = data[0]

bill.cafe\_order\_header\_id = data[1]

bill.customer\_id = data[2]

bill.customer = data[3]

bill.payment\_mode\_id = data[4]

bill.mode\_of\_payment = data[5]

bill.discount = data[6]

bill.final\_amount = data[7]

bill.datetime = data[8]

bill.table\_id = data[9]

bill.table\_number = data[10]

bill.total\_amount = data[11]

output\_list.append(bill)

print(f"|{bill.bill\_id:4}")

return output\_list

def bill\_get\_buy\_order\_header\_id(self,cafe\_order\_header\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "select bill\_id from bill where cafe\_order\_header\_id = "+str(cafe\_order\_header\_id)

mycursor.execute(sql)

list = mycursor.fetchall()

output\_list = []

for data in list:

bill = Bill()

bill.bill\_id = data[0]

output\_list.append(bill)

print(f"|{bill.bill\_id:4}")

return output\_list

def bill\_get\_buy\_customer\_id(self,customer\_id):

mydb = self.connect()

mycursor = mydb.cursor()

sql = "select bill\_id from bill where customer\_id = "+str(customer\_id)

mycursor.execute(sql)

list = mycursor.fetchall()

output\_list = []

for data in list:

bill = Bill()

bill.bill\_id = data[0]

output\_list.append(bill)

print(f"|{bill.bill\_id:4}")

return output\_list

def category\_get\_all(self):

mydb = self.connect()

mycursor = mydb.cursor()

mycursor.execute("""select category\_id, category\_name from category order by category\_id""")

list = mycursor.fetchall()

output\_list = []

for data in list:

cat = Category()

cat.category\_id = data[0]

cat.category\_name = data[1]

output\_list.append(cat)

print(f"|{cat.category\_id:4}")

return output\_list

**Bill\_Create.py**

import tkinter as tk

from tkinter import StringVar, messagebox

from datetime import datetime

from model.bill import Bill

from view.helper.comp\_helper import ComponentHelper

from database.db import Database

from view.event import Event

class BillCreate():

    cafe\_order\_header\_id=0

    id=0

    def \_\_init\_\_(self):

        print('create bill constructor')

        self.OnViewUpdated = Event()

    def ViewUpdated(self):

        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, id=0):

        self.id = id

        top=win.winfo\_toplevel()

        top.rowconfigure(0, weight=1)

        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)

        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()

        win.frame = helper.add\_background(win, "./images/bill\_add\_del.gif")

        db = Database()

        self.order\_header\_list = db.order\_header\_get\_upaid()

        order\_header\_arr = []

        order\_header\_arr.append('Please Select')

        for data in self.order\_header\_list:

            order\_header\_arr.append(str(data.order\_header\_id) + "|" + str(data.table\_number) + "|" + str(data.total\_amount) + "|" + str(data.create\_time))

        self.table\_list = db.table\_get\_all()

        table\_arr = []

        table\_arr.append('Please Select')

        for data in self.table\_list:

            table\_arr.append(data.table\_number)

        self.cust\_list = db.customer\_get\_all()

        cust\_arr = []

        cust\_arr.append('Please Select')

        for data in self.cust\_list:

            cust\_arr.append(data.customer\_name)

        self.payment\_mode\_list = db.payment\_mode\_get\_all()

        payment\_mode\_arr = []

        payment\_mode\_arr.append('Please Select')

        for data in self.payment\_mode\_list:

            payment\_mode\_arr.append(data.payment\_mode\_name)

        bill = Bill()

        db = Database()

        if(id > 0):

            list = db.bill\_get\_by\_id(id)

            for data in list:

                bill.bill\_id = data.bill\_id

                bill.cafe\_order\_header\_id=data.cafe\_order\_header\_id

                bill.payment\_mode\_id = data.payment\_mode\_id

                bill.mode\_of\_payment = data.mode\_of\_payment

                bill.datetime = data.datetime

                bill.customer\_id = data.customer\_id

                bill.customer = data.customer

                bill.table\_id = data.table\_id

                bill.table\_number = data.table\_number

                bill.total\_amount=data.total\_amount

                bill.discount = data.discount

                bill.final\_amount = data.final\_amount

            order\_txt = ''

            for data in self.order\_header\_list:

                if(data.cafe\_order\_header\_id == bill.cafe\_order\_header\_id):

                    order\_txt = str(data.order\_header\_id) + "|" + str(data.table\_number) + "|" + str(data.total\_amount) + "|" + str(data.create\_time)

        helper = ComponentHelper()

        if(self.id > 0):

            self.order = helper.create\_label\_options\_menu(win.frame, 0,'Order', order\_header\_arr, self.order\_changed, order\_txt)

            self.cust = helper.create\_label\_options\_menu(win.frame, 4,'Customer', cust\_arr, self.customer\_changed, bill.customer)

            self.mode\_of\_payment = helper.create\_label\_options\_menu(win.frame, 7,'Mode Of Payment', payment\_mode\_arr, self.item\_changed, bill.mode\_of\_payment)

        else:

            self.order = helper.create\_label\_options\_menu(win.frame, 0,'Order', order\_header\_arr, self.order\_changed)

            self.cust = helper.create\_label\_options\_menu(win.frame, 4,'Customer', cust\_arr, self.customer\_changed)

            self.mode\_of\_payment = helper.create\_label\_options\_menu(win.frame, 7,'Mode Of Payment', payment\_mode\_arr, self.item\_changed)

        self.table = helper.create\_label\_label(win.frame, 1,'Table', bill.table\_number)

        self.total\_amount = helper.create\_label\_label(win.frame, 2,'Total Amount', bill.total\_amount)

        self.datetime = helper.create\_label\_label(win.frame, 3, 'DateTime', bill.datetime)

        self.sv = StringVar()

        self.discount = helper.create\_label\_entry(win.frame, 5,'Discount', bill.discount, self.sv, self.discount\_changed)

        self.final\_amount = helper.create\_label\_label(win.frame, 6,'Final Amount', bill.final\_amount)

        cancel = tk.Button(win.frame, text='Cancel', command=lambda:self.bill\_cancel())

        cancel.grid(row=8, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

        self.submit = tk.Button(win.frame, text='Submit', command=lambda:self.bill\_create())

        self.submit.grid(row=8, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

        if(self.id > 0):

            self.order[1].configure(state="disable")

    def discount\_changed(self):

        print(self.sv.get())

        final\_amount = float(self.total\_amount.cget("text")) \* (1-(float(self.discount.get())/100))

        self.final\_amount.config(text=final\_amount)

        return True

    def bill\_cancel(self, \*args):

        self.ViewUpdated()

    def order\_changed(self, \*args):

        args\_arr = args[0].split('|')

        self.cafe\_order\_header\_id = args\_arr[0]

        self.table.config(text=args\_arr[1])

        self.total\_amount.config(text=args\_arr[2])

        self.datetime.config(text=args\_arr[3])

        final\_amount = float(self.total\_amount.cget("text")) \* (1-(float(self.discount.get())/100))

        self.final\_amount.config(text=final\_amount)

        print('data', self)

    def item\_changed(self, \*args):

        print(self)

    def customer\_changed(self, \*args):

        print(self)

        for data in self.cust\_list:

            if(data.customer\_name == args[0]):

                dob\_month = datetime.strptime(str(data.DOB), '%Y-%m-%d').month

                dob\_date = datetime.strptime(str(data.DOB), '%Y-%m-%d').day

                bill\_month = datetime.strptime(self.datetime.cget("text"), '%Y-%m-%d %H:%M:%S').month

                bill\_date = datetime.strptime(self.datetime.cget("text"), '%Y-%m-%d %H:%M:%S').day

                if(dob\_month == bill\_month and dob\_date == bill\_date):

                    messagebox.showinfo('Success!', 'Wish the Customer Happy Birthday! Added 50% discount to the bill!')

                    helper = ComponentHelper()

                    helper.change\_text(self.discount,'50')

                    final\_amount = float(self.total\_amount.cget("text")) \* (0.50)

                    self.final\_amount.config(text=final\_amount)

                    return

    def bill\_create(self):

        if self.cust[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Customer!')

            return

        if self.mode\_of\_payment[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Mode of Payment!')

            return

        db = Database()

        bill = Bill()

        bill.cafe\_order\_header\_id = self.cafe\_order\_header\_id

        for data in self.cust\_list:

            if data.customer\_name == self.cust[0].get():

                bill.customer\_id = data.customer\_id

        for data in self.payment\_mode\_list:

            if data.payment\_mode\_name == self.mode\_of\_payment[0].get():

                bill.payment\_mode\_id = data.payment\_mode\_id

        bill.datetime = self.datetime.cget("text")

        bill.discount = self.discount.get()

        final\_amount = float(self.total\_amount.cget("text")) \* (1-(float(self.discount.get())/100))

        self.final\_amount.config(text=final\_amount)

        bill.final\_amount = final\_amount

        if(self.id > 0):

            bill.bill\_id = self.id

            db.bill\_update(bill)

        else:

            db.bill\_create(bill)

        self.ViewUpdated()

        messagebox.showinfo('Success!', 'Bill Created Successfully')

**Customer\_create.py**

from datetime import datetime, timedelta

import tkinter as tk

from tkinter import messagebox

from model.customer import Customer

from view.helper.comp\_helper import ComponentHelper

from view.event import Event

from database.db import Database

import mysql.connector as mysql

import re

class CustomerCreate():

    name = tk.Entry

    phone = tk.Entry

    gender = tk.Entry

    email = tk.Entry

    dob = tk.Entry

    doa = tk.Entry

    table\_list = []

    customer\_list = []

    cust\_id = 0

    def \_\_init\_\_(self, win):

        print('create customer constructor')

        self.OnViewUpdated = Event()

    def ViewUpdated(self):

        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, cust\_id=0):

        top=win.winfo\_toplevel()

        top.rowconfigure(0, weight=1)

        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)

        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()

        win.frame = helper.add\_background(win, "./images/customer\_add\_delete.gif")

        cust = Customer()

        cust.gender = "Please Select"

        db = Database()

        if(cust\_id > 0):

            self.cust\_id = cust\_id

            cust\_list = db.customer\_get\_by\_id(cust\_id)

            for data in cust\_list:

                cust.customer\_id = self.cust\_id

                cust.customer\_name = data.customer\_name

                cust.phone\_number = data.phone\_number

                if data.gender == 1:

                    cust.gender = "Male"

                elif data.gender == 2:

                    cust.gender = "Female"

                else:

                    cust.gender = "Other"

                cust.gender = data.gender

                cust.email\_id = data.email\_id

                cust.DOB = data.DOB

                cust.anniversary\_date = data.anniversary\_date

        helper = ComponentHelper()

        self.name = helper.create\_label\_entry(win.frame, 1,'Customer Name', cust.customer\_name)

        self.phone = helper.create\_label\_entry(win.frame, 2,'Phone Number', cust.phone\_number)

        self.gender = helper.create\_label\_options\_menu(win.frame, 3,'Gender', ['Male','Female','Other'], self.item\_changed,cust.gender )

        self.email = helper.create\_label\_entry(win.frame, 4,'Email Id', cust.email\_id)

        self.dob = helper.create\_label\_entry(win.frame, 5,'Date Of Birth', cust.DOB)

        self.doa = helper.create\_label\_entry(win.frame, 6,'Anniversary Date', cust.anniversary\_date)

        self.cancel = tk.Button(win.frame, text='Cancel', command=lambda:self.customer\_cancel())

        self.cancel.grid(row=7, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

        self.submit = tk.Button(win.frame, text='Submit', command=lambda:self.customer\_create())

        self.submit.grid(row=7, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

    def item\_changed(self):

        pass

    def customer\_cancel(self):

        self.ViewUpdated()

    def customer\_create(self):

        if self.name.get() == "":

            messagebox.showerror('Failure!', 'Please Enter Customer Name!')

            return

        if self.phone.get() == "":

            messagebox.showerror('Failure!', 'Please Enter The Phone Number!')

            return

        if len(self.phone.get()) < 5:

            messagebox.showerror('Failure!', 'Phone Number should be atleast 5 digits long!')

            return

        if len(self.phone.get()) > 15:

            messagebox.showerror('Failure!', 'Phone Number cannot be longer than 15 digits!')

            return

        if self.gender[0].get() == "" or self.gender[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Enter Gender!')

            return

        if self.email.get() == "":

            messagebox.showerror('Failure!', 'Please Enter The Email ID!')

            return

        try:

            res = bool(datetime.strptime(self.dob.get().strip(), '%Y-%m-%d'))

        except BaseException as e:

            print(e)

            messagebox.showerror('Failure!', 'Please enter appropriate Birthday in YYYY-mm-dd format')

            return

        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') < (datetime.now() - timedelta(days=365\*150)):

            messagebox.showerror('Failure!', 'Age should not be more than 150 years')

            return

        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') > (datetime.now() - timedelta(days=365\*18)):

            messagebox.showerror('Failure!', 'Age should be at least 18 years')

            return

        try:

            res = bool(datetime.strptime(self.doa.get().strip(), '%Y-%m-%d'))

        except:

            messagebox.showerror('Failure!', 'Please enter appropriate Anniversary Date in YYYY-mm-dd format')

            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') < (datetime.now() - timedelta(days=365\*150)):

            messagebox.showerror('Failure!', 'Anniversary should not be more than 150 years')

            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') < (datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') + timedelta(days=365\*18)):

            messagebox.showerror('Failure!', '18 Years gap should be there between Anniversary and Date of Birth')

            return

        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') > datetime.now():

            messagebox.showerror('Failure!', 'Date of Birth cannot be a future date')

            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') > datetime.now():

            messagebox.showerror('Failure!', 'Anniversary Date cannot be a future date')

            return

        regex = r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+**\.**[A-Z|a-z]{2,}\b'

        if(re.fullmatch(regex, self.email.get())):

            print("Valid Email")

        else:

            messagebox.showerror('Failure!', 'Please Enter valid Email ID!')

            return

        db = Database()

        cust = Customer()

        cust.customer\_name = self.name.get()

        cust.phone\_number = self.phone.get()

        if self.gender[0].get() == 'Male':

            cust.gender = 1

        elif self.gender[0].get() == "Female":

            cust.gender = 2

        else:

            cust.gender = 3

        cust.email\_id = self.email.get()

        cust.DOB = self.dob.get()

        cust.anniversary\_date = self.doa.get()

        if(int(self.cust\_id) > 0):

            cust.customer\_id = self.cust\_id

            db.customer\_update(cust)

            messagebox.showinfo('Success!', 'Customer Updated Successfully')

        else:

            db.customer\_create(cust)

            messagebox.showinfo('Success!', 'Customer Created Successfully')

        self.ViewUpdated()

**order\_create.py**

import tkinter as tk

from tkinter import \*

from tkinter.ttk import \*

from tkinter import messagebox

from model import order\_detail

from model.item import Item

from model.order\_header import OrderHeader

from model.order\_detail import OrderDetail

from view.helper.comp\_helper import ComponentHelper

from view.view import View

from view.event import Event

from database.db import Database

class OrderCreate():

    i=1

    item = tk.OptionMenu

    table = tk.OptionMenu

    qty = tk.Entry

    price = tk.Label

    gst = tk.Label

    #amount = tk.Label

    description = tk.Label

    spice\_lvl = tk.Label

    veg = tk.Label

    category = tk.Label

    item\_list = []

    table\_list = []

    order\_details = []

    popup = tk.Menu

    tv = Treeview

    order\_header\_id = 0

    win=object

    def \_\_init\_\_(self):

        print('create order constructor')

        self.item\_list = []

        self.table\_list = []

        self.order\_details = []

        self.OnViewUpdated = Event()

    def ViewUpdated(self):

        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, order\_header\_id=''):

        top=win.winfo\_toplevel()

        top.rowconfigure(0, weight=1)

        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)

        win.columnconfigure(1, weight=1)

        self.win = win

        helper = ComponentHelper()

        win.frame = helper.add\_background(win, "./images/order\_add\_del.gif", 0.90)

        db = Database()

        self.item\_list = db.item\_get\_all()

        item\_arr = []

        item\_arr.append('Please Select')

        for data in self.item\_list:

            item\_arr.append(data.item)

        self.table\_list = db.table\_get\_all()

        table\_arr = []

        table\_arr.append('Please Select')

        for data in self.table\_list:

            table\_arr.append(data.table\_number)

        self.category\_list = db.category\_get\_all()

        category\_arr = []

        category\_arr.append('Please Select')

        for data in self.category\_list:

            category\_arr.append(data.category\_name)

        helper = ComponentHelper()

        order\_header = OrderHeader()

        if(order\_header\_id != ''):

            self.order\_header\_id = order\_header\_id

            order\_header\_list = db.order\_header\_get\_by\_id(order\_header\_id)

            for data in order\_header\_list:

                order\_header.order\_header\_id = order\_header\_id

                order\_header.table\_id = data.table\_id

                order\_header.table\_number = data.table\_number

                order\_header.total\_amount = data.total\_amount

        header = Frame(win.frame)

        header['padding'] = 1

        header['width'] = 500

        header['height'] = 100

        header['borderwidth'] = 1

        header['relief'] = 'sunken'

        header.grid(columnspan=2,sticky=tk.N+tk.S+tk.E+tk.W)

        self.table = helper.create\_label\_options\_menu(header, 0,'Table', table\_arr, self.table\_changed, order\_header.table\_number)

        self.total\_amount = helper.create\_label\_label(header, 1,'Total Amount', order\_header.total\_amount)

        self.category = helper.create\_label\_options\_menu(win.frame, 2,'Category', category\_arr, self.category\_changed)

        self.item = helper.create\_label\_options\_menu(win.frame, 3,'Item Name', item\_arr, self.item\_changed)

        self.qty = helper.create\_label\_entry(win.frame, 4,'Quantity', 1)

        self.price = helper.create\_label\_label(win.frame, 5,'Price', 0)

        self.description = helper.create\_label\_label(win.frame, 6,'Description', '')

        self.spice\_lvl = helper.create\_label\_label(win.frame, 7,'Spice Level', 1)

        self.veg = helper.create\_label\_label(win.frame, 8,'Veg', 'Yes')

        self.gst = helper.create\_label\_label(win.frame, 9,'GST', '18')

        button\_frame = tk.Frame(win.frame, bg='#CC8066')

        #button\_frame.place(relx=0,rely=0,relwidth=0.75,relheight=0.75)

        button\_frame.grid(columnspan=2, sticky=tk.S)

        self.add = tk.Button(button\_frame, text='Add Item', command=lambda:self.AddItem(tv, False))

        self.add.grid(row=11, column=1, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.update = tk.Button(button\_frame, text='Update', command=lambda:self.UpdateItem(tv))

        self.update.grid(row=11, column=2, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.cancel = tk.Button(button\_frame, text='Cancel', command=lambda:self.Cancel())

        self.cancel.grid(row=11, column=3, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.enable\_insert(True)

        help = tk.Label(button\_frame, text='\*Right click to Edit or Delete the Item')

        help.grid(row=12, column=1, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        tv = self.CreateUI(win.frame, 13)

        if(order\_header\_id != ''):

            order\_detail\_list = db.order\_detail\_get\_by\_id(order\_header\_id)

            for data in order\_detail\_list:

                order\_detail = OrderDetail()

                order\_detail.order\_detail\_id = data.order\_detail\_id

                order\_detail.order\_header\_id = order\_header\_id

                order\_detail.item\_id = data.item\_id

                order\_detail.item = data.item

                order\_detail.qty = data.qty

                order\_detail.price = data.price

                order\_detail.amount = data.amount

                self.order\_details.append(order\_detail)

                tv.insert("", 'end', iid=None, text=self.i, values=(data.item, data.qty, data.price, data.gst, data.amount))

                self.i = self.i + 1

        submit\_button\_frame = tk.Frame(win.frame, bg='#CC8066')

        submit\_button\_frame.grid(columnspan=2)

        self.submit = tk.Button(submit\_button\_frame, text='Submit Order', command=lambda:self.order\_create())

        self.submit.grid(row=14, column=1, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.cancel\_submit = tk.Button(submit\_button\_frame, text='Cancel', command=lambda:self.order\_cancel())

        self.cancel\_submit.grid(row=14, column=2, sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

    def category\_changed(self, \*args):

        self.price.config(text=0)

        self.description.config(text='')

        self.spice\_lvl.config(text=1)

        self.veg.config(text='Yes')

        self.gst.config(text=18)

        category\_id=0

        for data in self.category\_list:

            if data.category\_name == self.category[0].get():

                category\_id = data.category\_id

        self.item[0].set('')

        self.item[1]['menu'].delete(0, 'end')

        self.item[1]['menu'].add\_command(label='Please Select', command=tk.\_setit(self.item[0], 'Please Select'))

        db = Database()

        self.item\_list = db.item\_get\_by\_category\_id(category\_id)

        item\_arr = []

        item\_arr.append('Please Select')

        for data in self.item\_list:

            item\_arr.append(data.item)

            #self.item[1]['menu'].add\_command(label=data.item, command=lambda:self.item\_changed(data.item))

            self.item[1]['menu'].add\_command(label=data.item, command=tk.\_setit(self.item[0], data.item, self.item\_changed))

    def order\_cancel(self):

        self.ViewUpdated()

    def Cancel(self):

        self.enable\_insert(True)

    def UpdateItem(self, tv):

        self.AddItem(tv, True)

    def enable\_insert(self, flag):

        if flag:

            self.add["state"] = "normal"

            self.item[1].configure(state="normal")

            self.category[1].configure(state="normal")

            self.update["state"] = "disabled"

            self.cancel["state"] = "disabled"

        else:

            self.add["state"] = "disabled"

            self.item[1].configure(state="disabled")

            self.category[1].configure(state="disabled")

            self.update["state"] = "normal"

            self.cancel["state"] = "normal"

    def item\_changed(self, \*args):

        print(self)

        self.price.config(text=0)

        self.description.config(text='')

        self.spice\_lvl.config(text=1)

        self.veg.config(text='Yes')

        self.gst.config(text=18)

        helper = ComponentHelper()

        helper.change\_text(self.qty, 1)

        for data in self.item\_list:

            if data.item == args[0]:

                self.price.config(text=data.price)

                self.description.config(text=data.description)

                if(data.spice\_level == 0):

                    self.spice\_lvl.config(text='Low')

                elif(data.spice\_level == 1):

                    self.spice\_lvl.config(text='Medium')

                elif(data.spice\_level == 2):

                    self.spice\_lvl.config(text='High')

                if(data.veg == 1):

                    self.veg.config(text='Yes')

                else:

                    self.veg.config(text='No')

                #self.category[0].set(data.category)

    def table\_changed(self, \*args):

        print(self)

        #self.output\_label['text'] = f'You selected: {self.option\_var.get()}'

    def order\_create(self):

        if self.table[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Table Number!')

            return

        if not self.order\_details:

            messagebox.showerror('Failure!', 'Please Add atleast one item!')

            return

        db = Database()

        order\_header = OrderHeader()

        for data in self.table\_list:

            if data.table\_number == self.table[0].get():

                order\_header.table\_id = data.table\_id

        order\_header.total\_amount = self.total\_amount.cget("text")

        if(self.order\_header\_id != '' and self.order\_header\_id != 0):

            order\_header.order\_header\_id = self.order\_header\_id

            db.order\_header\_update(order\_header)

            db.order\_detail\_delete\_by\_id(self.order\_header\_id)

        else:

            self.order\_header\_id = db.order\_header\_create(order\_header)

        for ord in self.order\_details:

            order\_detail = OrderDetail()

            order\_detail.order\_detail\_id = ord.order\_detail\_id

            order\_detail.order\_header\_id = self.order\_header\_id

            order\_detail.item\_id = ord.item\_id

            order\_detail.qty = ord.qty

            order\_detail.price = ord.price

            order\_detail.amount = ord.amount

            db.order\_detail\_create(order\_detail)

        messagebox.showinfo('Success!', 'Order Created Successfully')

        self.ViewUpdated()

    def AddItem(self, tv, update):

        if self.table[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Table!')

            return

        if self.item[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Item Name!')

            return

        if self.qty.get() == "0" or self.qty.get() == "":

            messagebox.showerror('Failure!', 'Quantity Should be greater than Zero')

            return

        if int(self.qty.get()) > 9:

            messagebox.showerror('Failure!', 'Quantity Should be less than TEN')

            return

        if int(self.qty.get()) < 0:

            messagebox.showerror('Failure!', 'Quantity Should be greater than 0')

            return

        if(update == False):

            for row in self.order\_details:

                if(row.item == self.item[0].get()):

                    messagebox.showerror('Failure!', 'Item already present in order, to make changes, edit the item')

                    return

        order\_detail = OrderDetail()

        for data in self.item\_list:

            if data.item == self.item[0].get():

                order\_detail.item\_id = data.item\_id

                order\_detail.item = self.item[0].get()

        order\_detail.qty = self.qty.get()

        order\_detail.price = self.price.cget("text")

        order\_detail.gst = self.gst.cget("text")

        order\_detail.amount = float(order\_detail.qty) \* float(order\_detail.price) \* (1 + (float(order\_detail.gst)/100))

        if(update == True):

            x = tv.get\_children()

            print(x)

            for row in x:

                values = tv.item(row)['values']

                if(values[0] == order\_detail.item):

                    tv.item(row, text=tv.item(row)['text'], values=(order\_detail.item, order\_detail.qty, order\_detail.price, order\_detail.gst, order\_detail.amount))

            for row in self.order\_details:

                if(order\_detail.item == row.item):

                    row.qty = order\_detail.qty

                    row.amount = order\_detail.amount

        else:

            self.order\_details.append(order\_detail)

            tv.insert("", 'end', iid=None, text=self.i, values=(order\_detail.item, order\_detail.qty, order\_detail.price, order\_detail.gst, order\_detail.amount))

            self.i = self.i + 1

        total\_amount = 0

        for row in self.order\_details:

            total\_amount = total\_amount + row.amount

        self.total\_amount.config(text=total\_amount)

        self.enable\_insert(True)

    def edit(self):

        print("Edit", self.popup.selection)

        self.item[0].set(self.popup.selection['Item'])

        helper = ComponentHelper()

        helper.change\_text(self.qty, self.popup.selection['Qty'])

        self.price.config(text=str(self.popup.selection['Price']))

        for data in self.item\_list:

            if data.item == self.item[0].get():

                self.category[0].set(data.category)

        self.enable\_insert(False)

    def delete(self):

        print("Delete", self.popup.selection)

        try:

            selected\_item = self.tv.selection()[0]

            self.tv.delete(selected\_item)

        except:

            messagebox.showerror('Failure!', 'Please select an Item before Deleting!')

            return

        ord\_det = OrderDetail()

        for row in self.order\_details:

            if(row.item == self.popup.selection['Item']):

                ord\_det = row

        if(ord\_det.amount > 0):

            self.order\_details.remove(ord\_det)

        total\_amount = 0

        for row in self.order\_details:

            total\_amount = total\_amount + row.amount

        self.total\_amount.config(text=total\_amount)

    def do\_popup(self, event):

        # display the popup menu

        try:

            self.popup.selection = self.treeview.set(self.treeview.identify\_row(event.y))

            self.popup.post(event.x\_root, event.y\_root)

        finally:

            # make sure to release the grab (Tk 8.0a1 only)

            self.popup.grab\_release()

    def CreateUI(self, win, row):

        #Create menu

        self.popup = tk.Menu(win, tearoff=0)

        self.popup.add\_command(label="Edit", command=self.edit)

        self.popup.add\_separator()

        self.popup.add\_command(label="Delete", command=self.delete)

        self.tv = Treeview(win, height=9)

        self.tv.bind("<Button-3>", self.do\_popup)

        self.tv['columns'] = ('Item', 'Qty', 'Price', 'GST', 'Amount')

        self.tv.heading("#0", text='S No', anchor='w')

        self.tv.column("#0", anchor="w", width=25)

        self.tv.heading('Item', text='Item')

        self.tv.column('Item', anchor='center', width=200)

        self.tv.heading('Qty', text='Qty')

        self.tv.column('Qty', anchor='center', width=100)

        self.tv.heading('Price', text='Price')

        self.tv.column('Price', anchor='center', width=100)

        self.tv.heading('GST', text='GST')

        self.tv.column('GST', anchor='center', width=100)

        self.tv.heading('Amount', text='Amount')

        self.tv.column('Amount', anchor='center', width=100)

        self.tv.grid(row=row, column=0, sticky = (N,S,W,E), columnspan=2)

        self.treeview = self.tv

        return self.tv

**table\_create.py**

import tkinter as tk

from model.table import Table

from view.helper.comp\_helper import ComponentHelper

class TableCreate():

    def \_\_init\_\_(self, win, tbl=Table):

        print('create customer constructor')

        self.createWidgets(win, tbl)

    def createWidgets(self, win, tbl=Table):

        top=win.winfo\_toplevel()

        top.rowconfigure(0, weight=1)

        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)

        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()

        helper.create\_label\_entry(win, 0,'Table Number', tbl.table\_number)

        helper.create\_label\_entry(win, 1,'Number of Seats', tbl.number\_of\_seats)

        helper.create\_label\_entry(win, 2,'Location', tbl.location)

        self.submit = tk.Button(win, text='Submit', command=win.quit)

        self.submit.grid(row=6, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

**table\_reservation\_create.py**

import tkinter as tk

from tkinter import \*

from datetime import datetime, timedelta

from tkinter import messagebox

from tkinter.ttk import Treeview

from model.table\_reservation import TableReservation

from view.helper.comp\_helper import ComponentHelper

from database.db import Database

import sys

from view.event import Event

class TableReservationCreate():

    customer = tk.OptionMenu

    table = tk.OptionMenu

    datetime = ''

    pax = 2

    popup = tk.Menu

    tv = Treeview

    id = 0

    def \_\_init\_\_(self, win):

        print('create table reservation constructor')

        self.OnViewUpdated = Event()

    def ViewUpdated(self):

        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):

        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, id=0):

        top=win.winfo\_toplevel()

        top.rowconfigure(0, weight=1)

        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)

        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()

        win.frame = helper.add\_background(win, "./images/table\_resev\_add\_del.gif", 0.90)

        db = Database()

        self.table\_list = db.table\_get\_all()

        table\_arr = []

        table\_arr.append('Please Select')

        for data in self.table\_list:

            table\_arr.append(data.table\_number)

        self.cust\_list = db.customer\_get\_all()

        cust\_arr = []

        cust\_arr.append('Please Select')

        for data in self.cust\_list:

            cust\_arr.append(data.customer\_name)

        obj = TableReservation()

        db = Database()

        if(id > 0):

            self.id = id

            list = db.table\_reservation\_get\_by\_id(id)

            for data in list:

                obj.reservation\_id=data.reservation\_id

                obj.table\_id = data.table\_id

                obj.table\_number = data.table\_number

                obj.pax = data.pax

                obj.datetime = data.datetime

                obj.customer\_id = data.customer\_id

                obj.customer = data.customer

        for data in self.table\_list:

            if(data.table\_number == obj.table\_number):

                obj.location = data.location

                obj.number\_of\_seats = data.number\_of\_seats

        helper = ComponentHelper()

        self.customer = helper.create\_label\_options\_menu(win.frame, 0,'Customer', cust\_arr, self.item\_changed, obj.customer)

        self.table = helper.create\_label\_options\_menu(win.frame, 1,'Table', table\_arr, self.table\_changed, obj.table\_number)

        self.number\_of\_seats = helper.create\_label\_label(win.frame, 2,'Number of Seats', obj.number\_of\_seats)

        self.location = helper.create\_label\_label(win.frame, 3,'Location', obj.location)

        self.datetime = helper.create\_label\_entry(win.frame, 4,'DateTime ', obj.datetime)

        self.pax = helper.create\_label\_entry(win.frame, 5,'Pax', obj.pax)

        self.submit = tk.Button(win.frame, text='Cancel', command=lambda:self.table\_reservation\_cancel())

        self.submit.grid(row=6, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

        self.submit = tk.Button(win.frame, text='Submit', command=lambda:self.table\_reservation\_create())

        self.submit.grid(row=6, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

    def table\_changed(self, \*args):

        for data in self.table\_list:

            if(data.table\_number == args[0]):

                self.number\_of\_seats.config(text=data.number\_of\_seats)

                self.location.config(text=data.location)

    def table\_reservation\_cancel(self):

        self.ViewUpdated()

    def table\_reservation\_create(self):

        if self.customer[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Customer!')

            return

        if self.table[0].get() == "Please Select":

            messagebox.showerror('Failure!', 'Please Select Table!')

            return

        if self.datetime.get().strip() == "":

            messagebox.showerror('Failure!', 'Please Enter DateTime!')

            return

        try:

            res = bool(datetime.strptime(self.datetime.get().strip(), '%Y-%m-%d  %H:%M:%S'))

        except BaseException as e:

            print(e)

            messagebox.showerror('Failure!', 'Please enter DateTime in YYYY-mm-dd HH:MM:SS format')

            return

        if datetime.strptime(self.datetime.get().strip(), '%Y-%m-%d %H:%M:%S') < datetime.now():

            messagebox.showerror('Failure!', 'DateTime should be in the future')

            return

        if self.pax.get() == "":

            messagebox.showerror('Failure!', 'Please Enter the number of people!')

            return

        if self.pax.get() == "0":

            messagebox.showerror('Failure!', 'Please Enter at least 1 person')

            return

        if int(self.pax.get()) > int(self.number\_of\_seats.cget("text")):

            messagebox.showerror('Failure!', str(self.table[0].get()) + ' can accomodate only '+ str(self.number\_of\_seats.cget("text")) + ' people')

            return

        db = Database()

        if(self.id == 0):

            reserv\_list = db.table\_reservation\_get\_all()

            for data in reserv\_list:

                dt = datetime.strptime(str(data.datetime), '%Y-%m-%d %H:%M:%S')

                dt\_one = dt + timedelta(hours=1)

                user\_dt = datetime.strptime(self.datetime.get().strip(), '%Y-%m-%d %H:%M:%S')

                if((dt <= user\_dt <= dt\_one) and data.table\_number == self.table[0].get()):

                    messagebox.showerror('Failure!', 'Table already booked from ' + str(dt) + " to " + str(dt\_one))

                    return

        reserv = TableReservation()

        for data in self.table\_list:

            if data.table\_number == self.table[0].get():

                reserv.table\_id = data.table\_id

        for data in self.cust\_list:

            if data.customer\_name == self.customer[0].get():

                reserv.customer\_id = data.customer\_id

        reserv.pax = self.pax.get()

        reserv.datetime = self.datetime.get()

        try:

            if(self.id > 0):

                reserv.reservation\_id = self.id

                db.table\_reservation\_update(reserv)

            else:

                db.table\_reservation\_create(reserv)

            messagebox.showinfo('Success!', 'Table Reserved Successfully')

        except:

            type, value, traceback = sys.exc\_info()

            messagebox.showerror('Failure!', value)

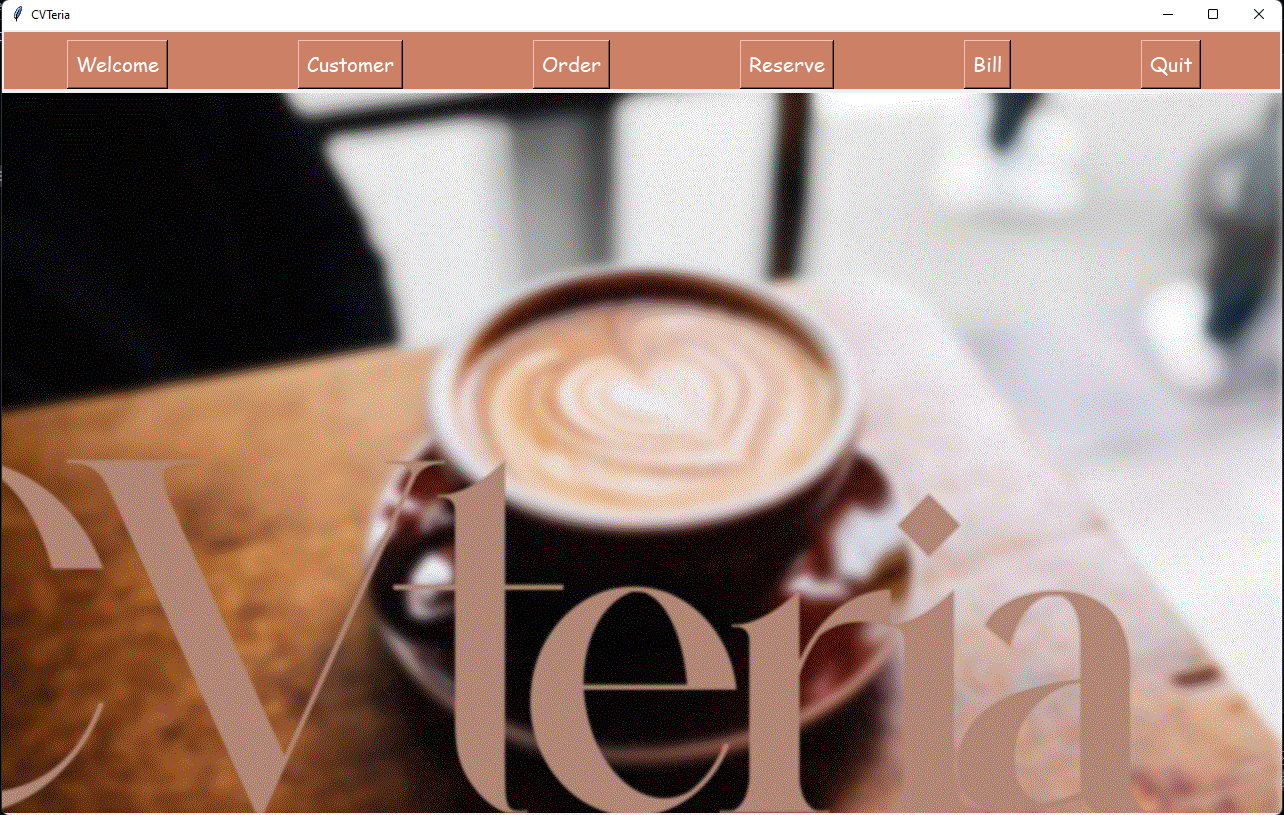
        self.ViewUpdated()

    def item\_changed(self, \*args):

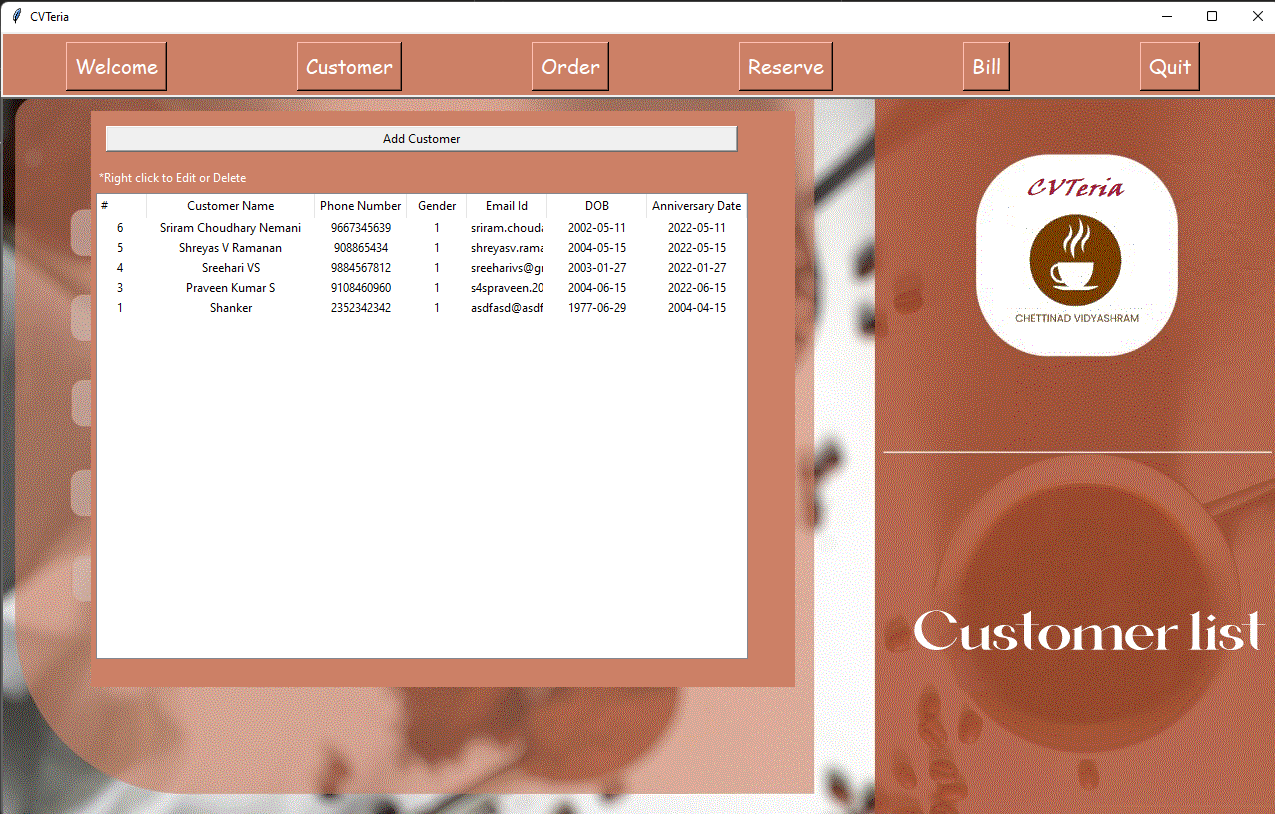
        print(self)

**Sample Output:**

**Welcome Screen:**

****

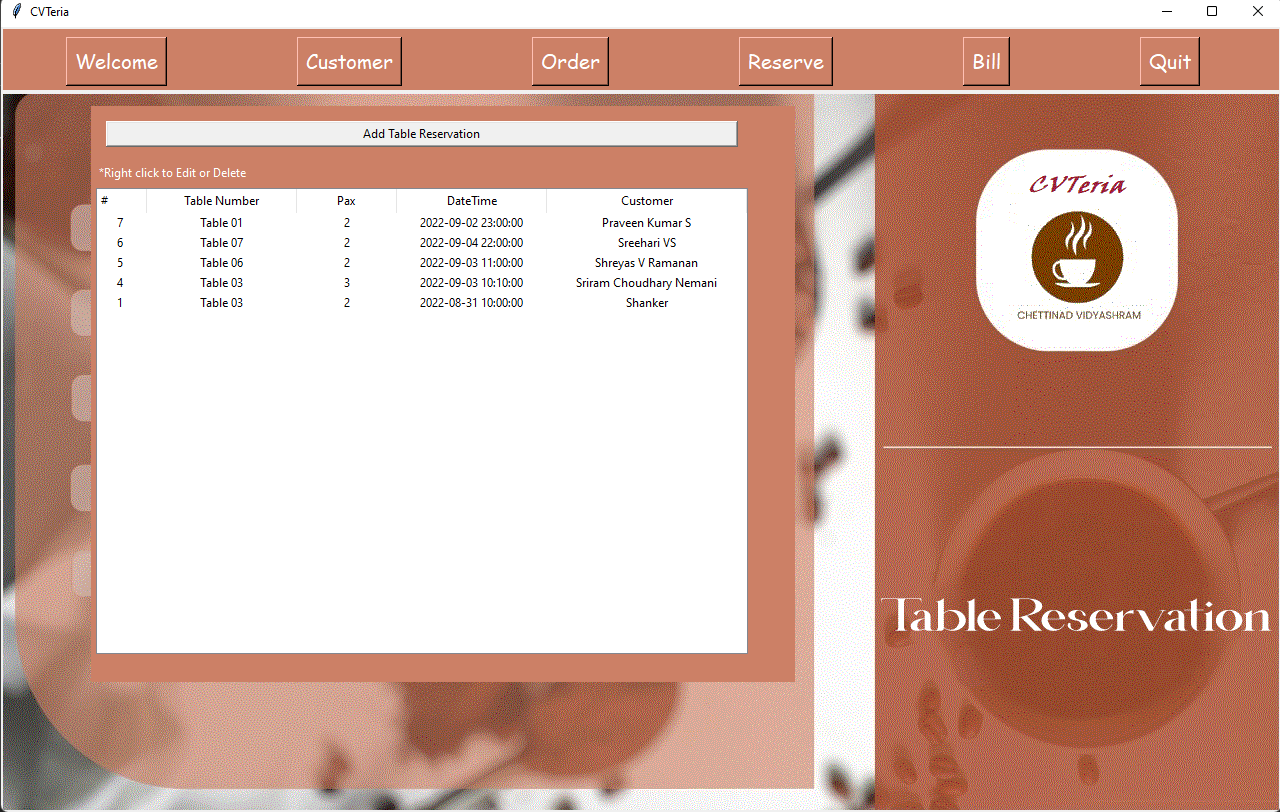
**Customer Listing Screen:**

****

**Order Listing Screen:**

****

**Table Reservation Screen:**

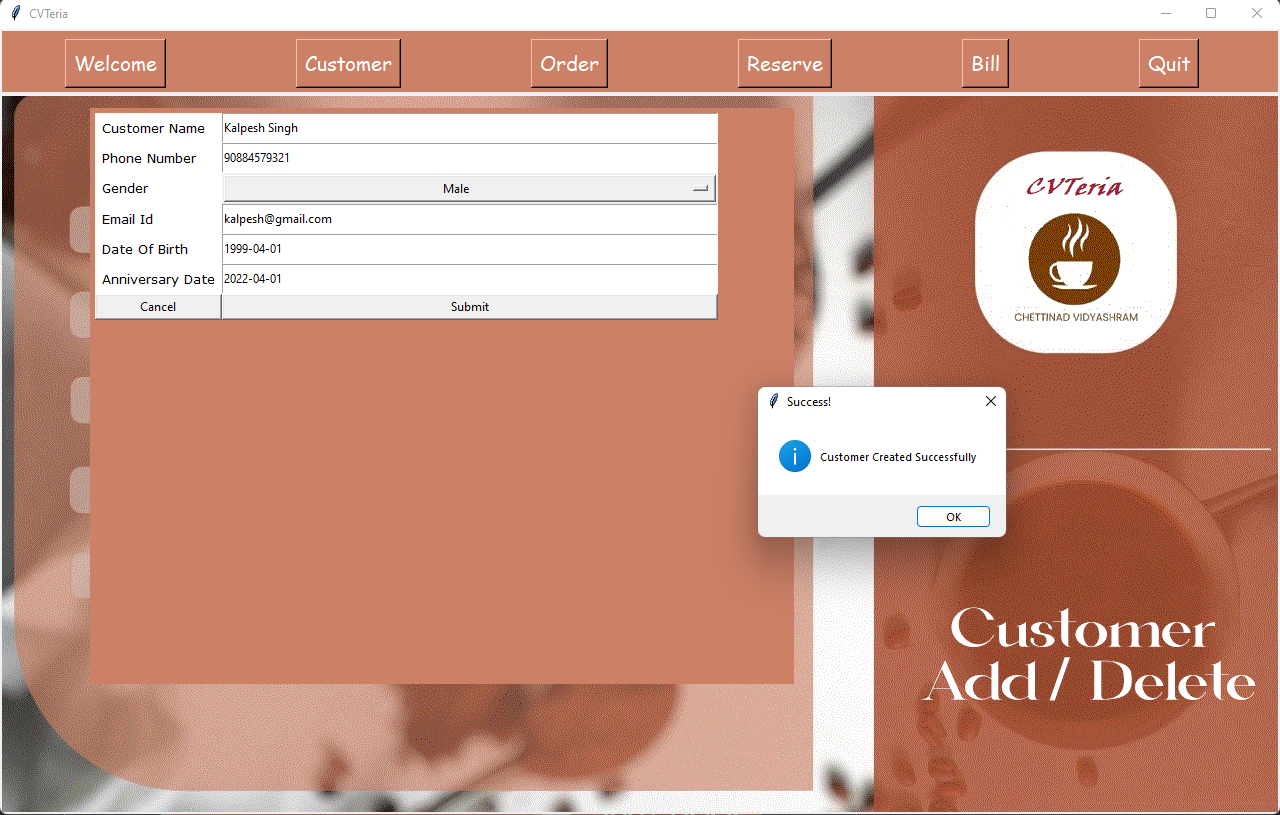
****

**Bill Listing Screen:**

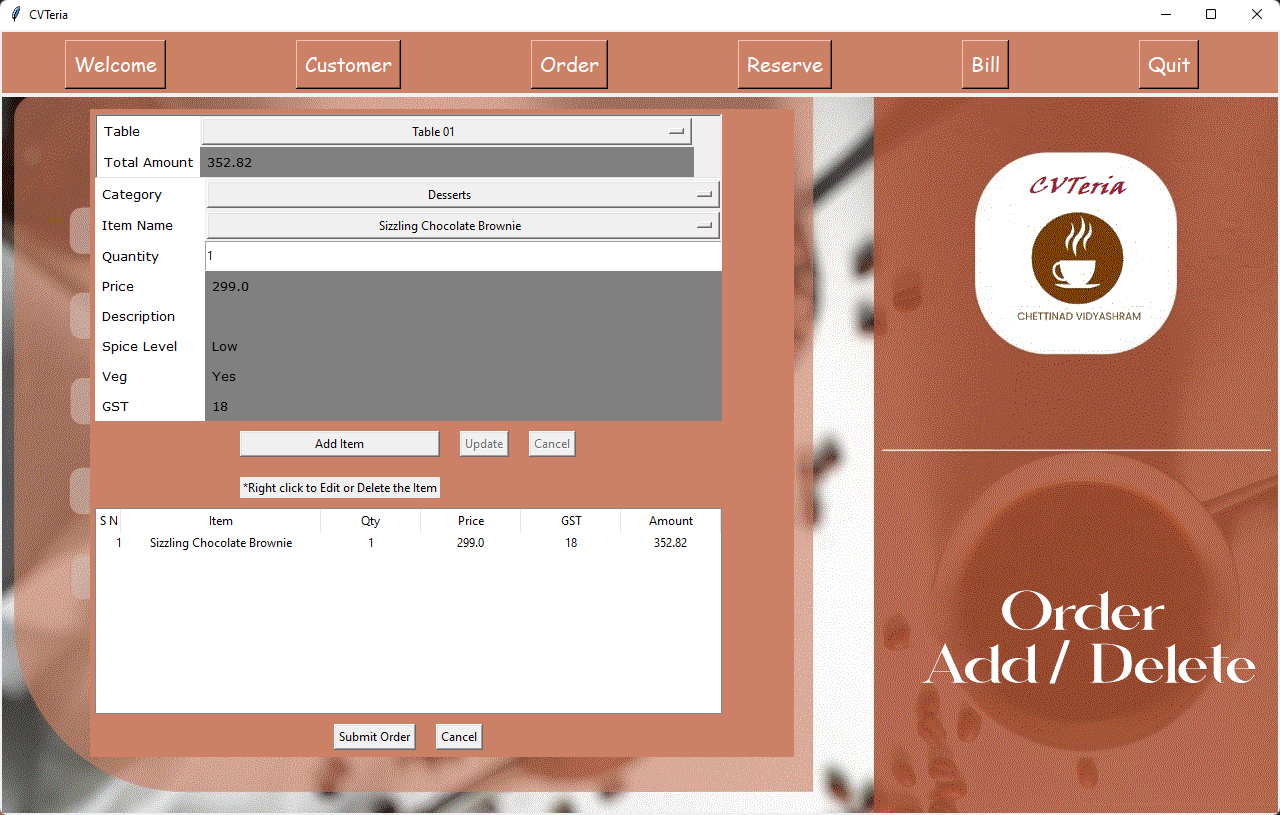
****

**Create Customer Screen:**

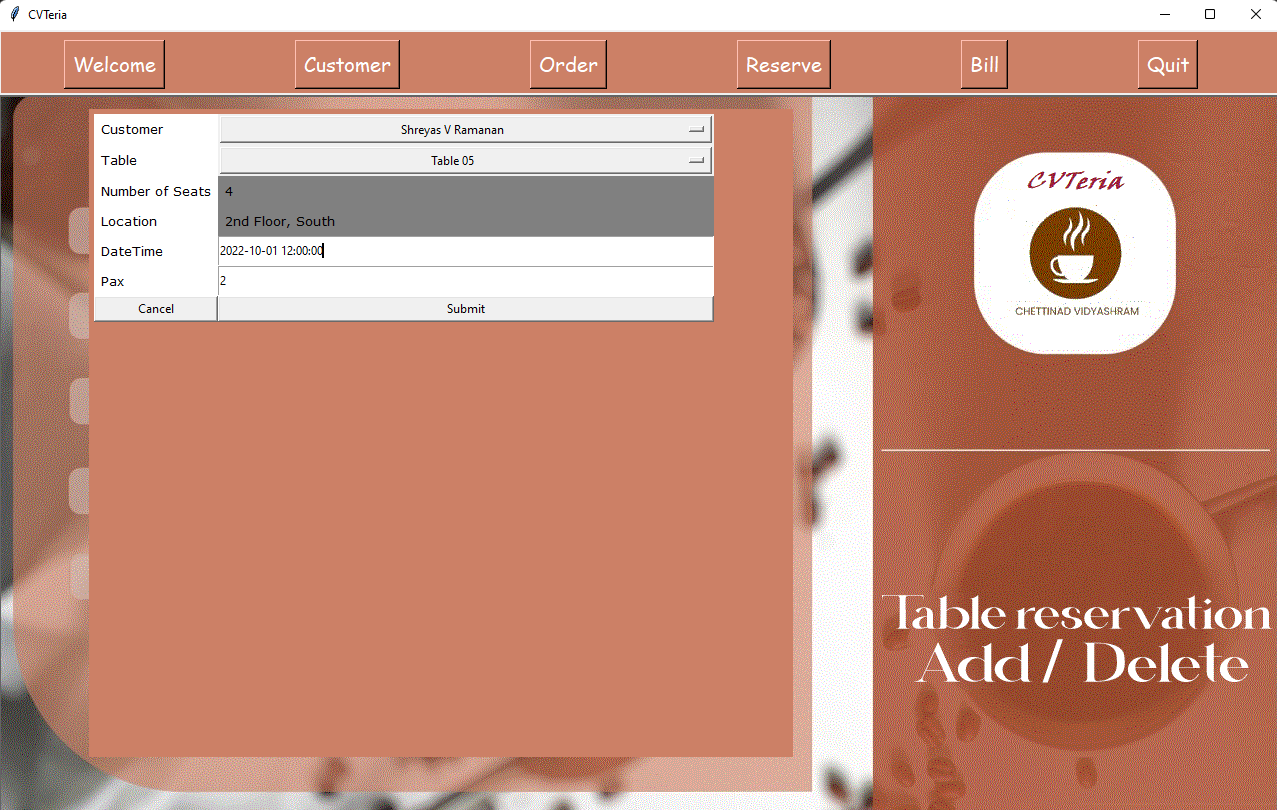
“Customer Created Successfully” message is displayed

****

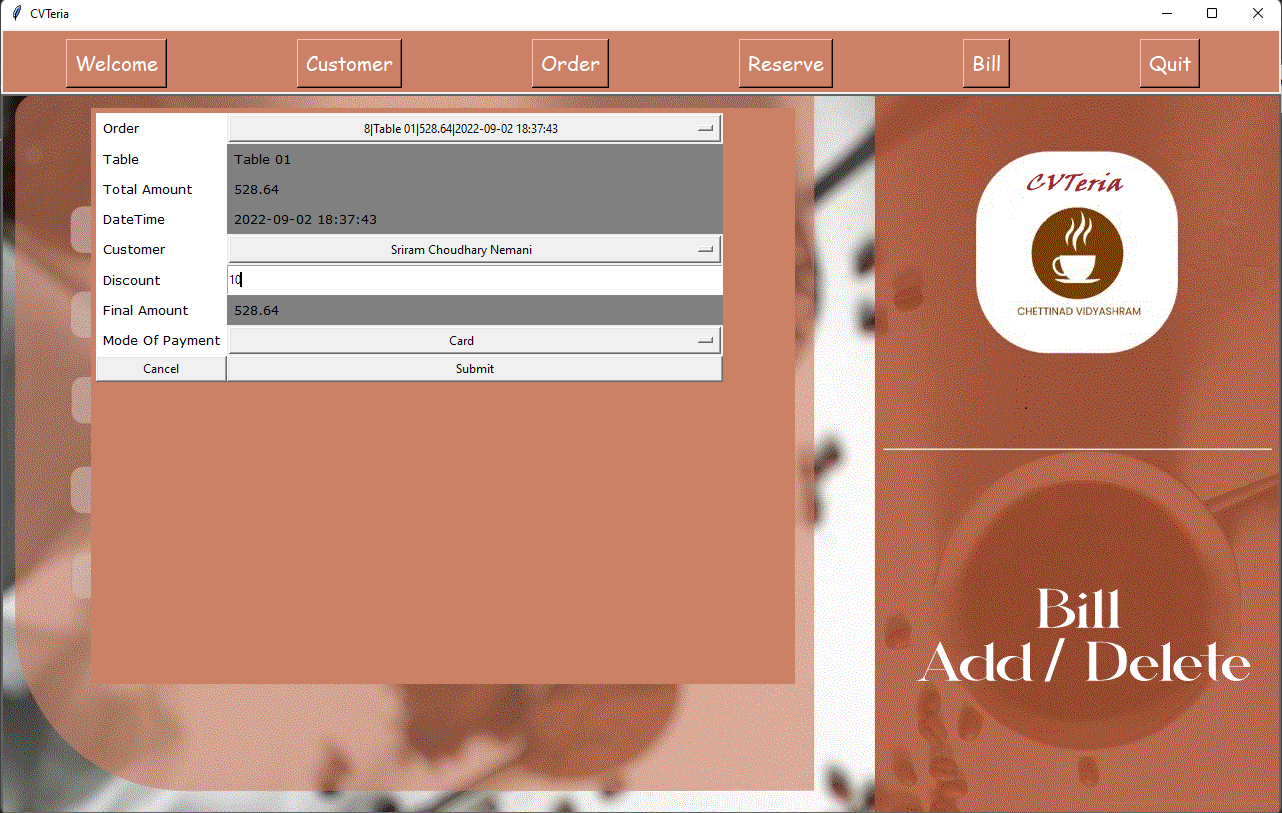
**Create Order Screen:**

****

**Create Table Reservation Screen:**

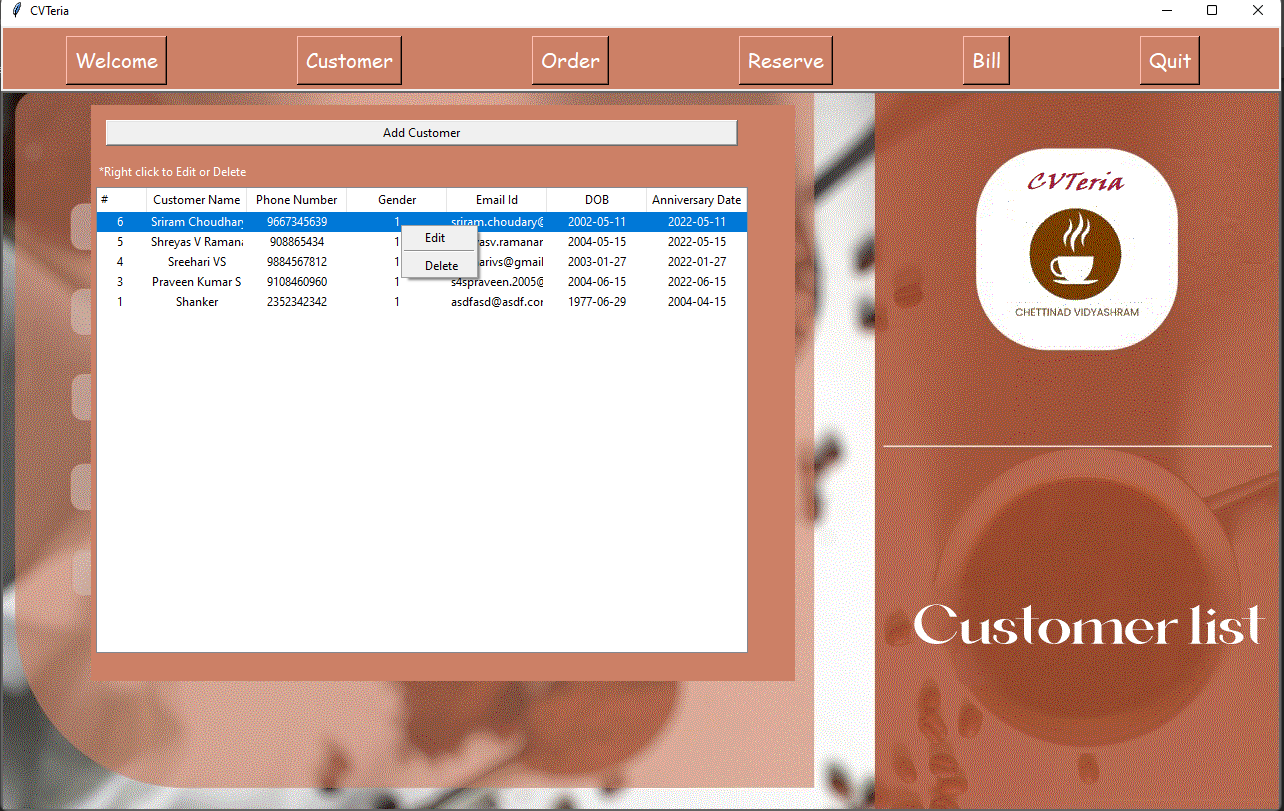


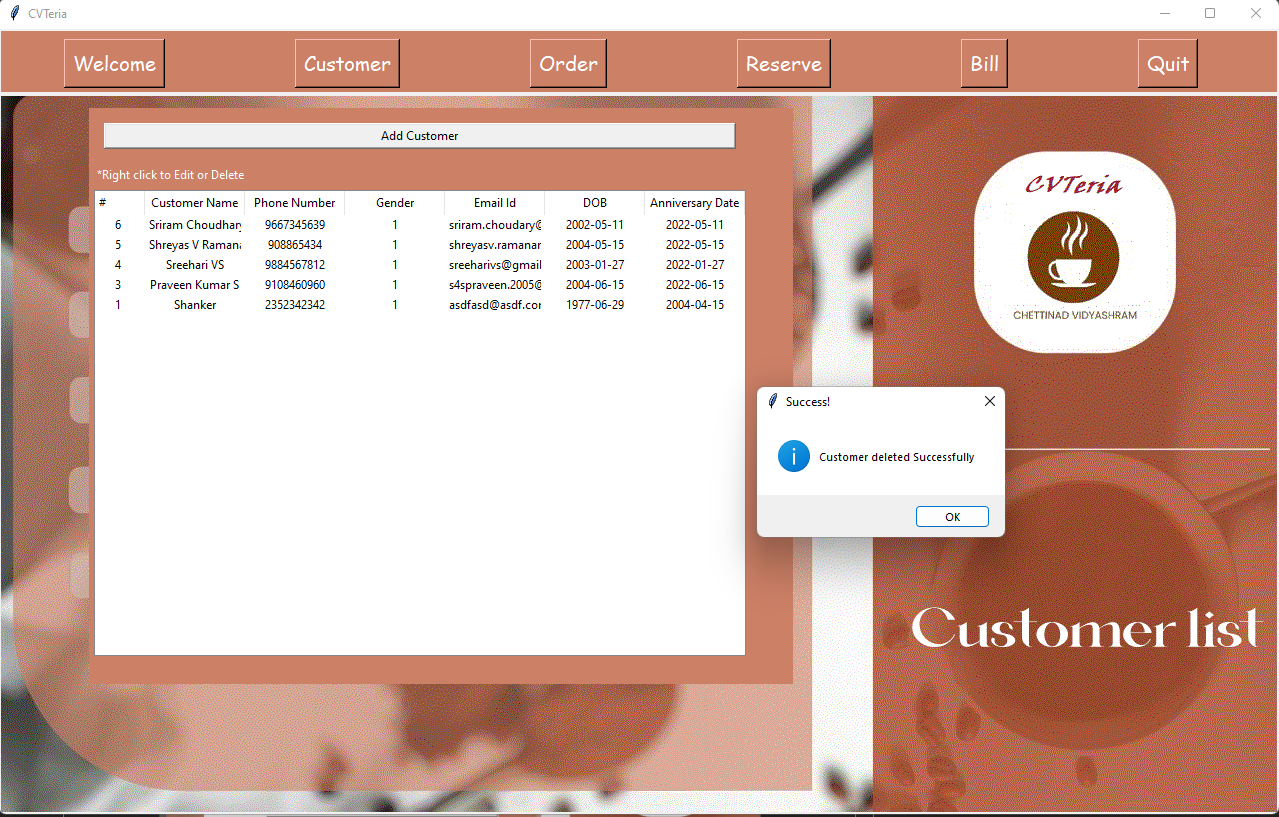
**Create Bill Screen:**

****

**Edit/Delete Customer (Enabled on right clicking):**

Customer Deleted successfully message is displayed

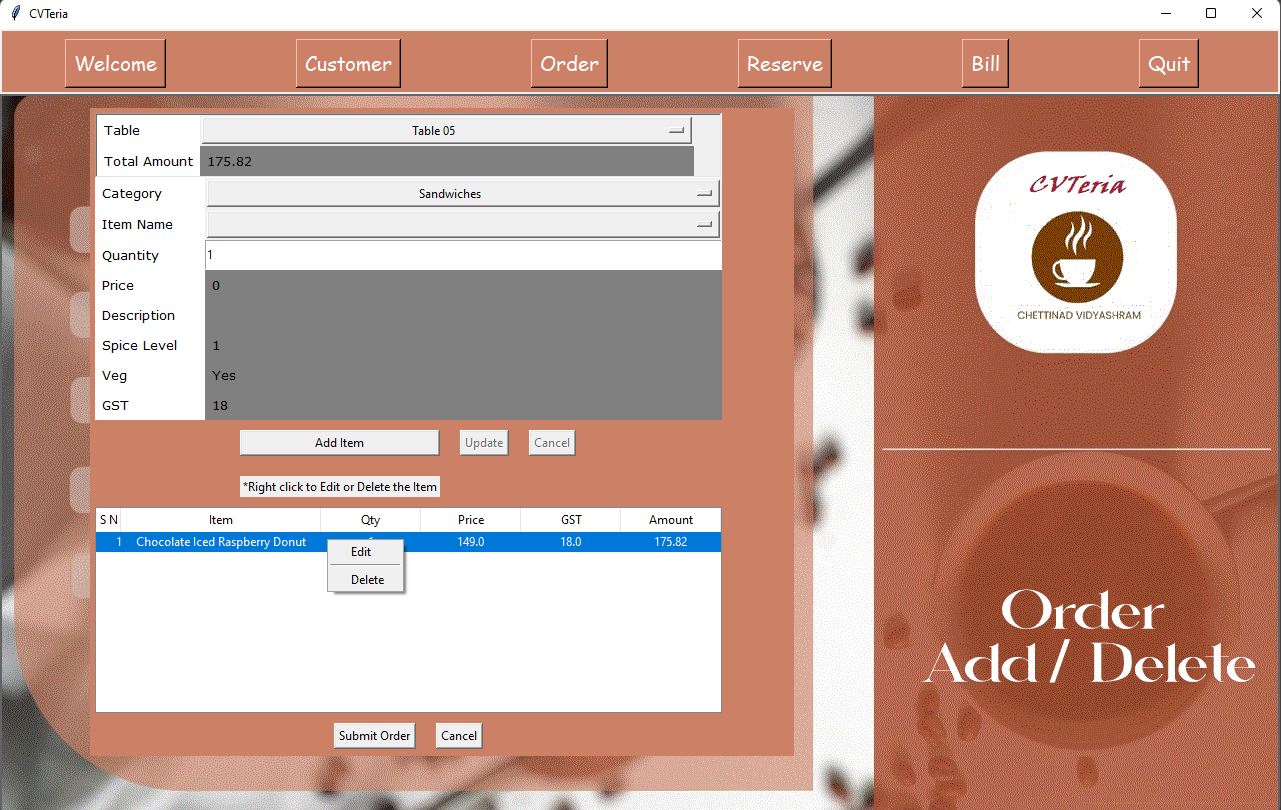
****

****

Note: Editing and Deleting and enable for all screens.

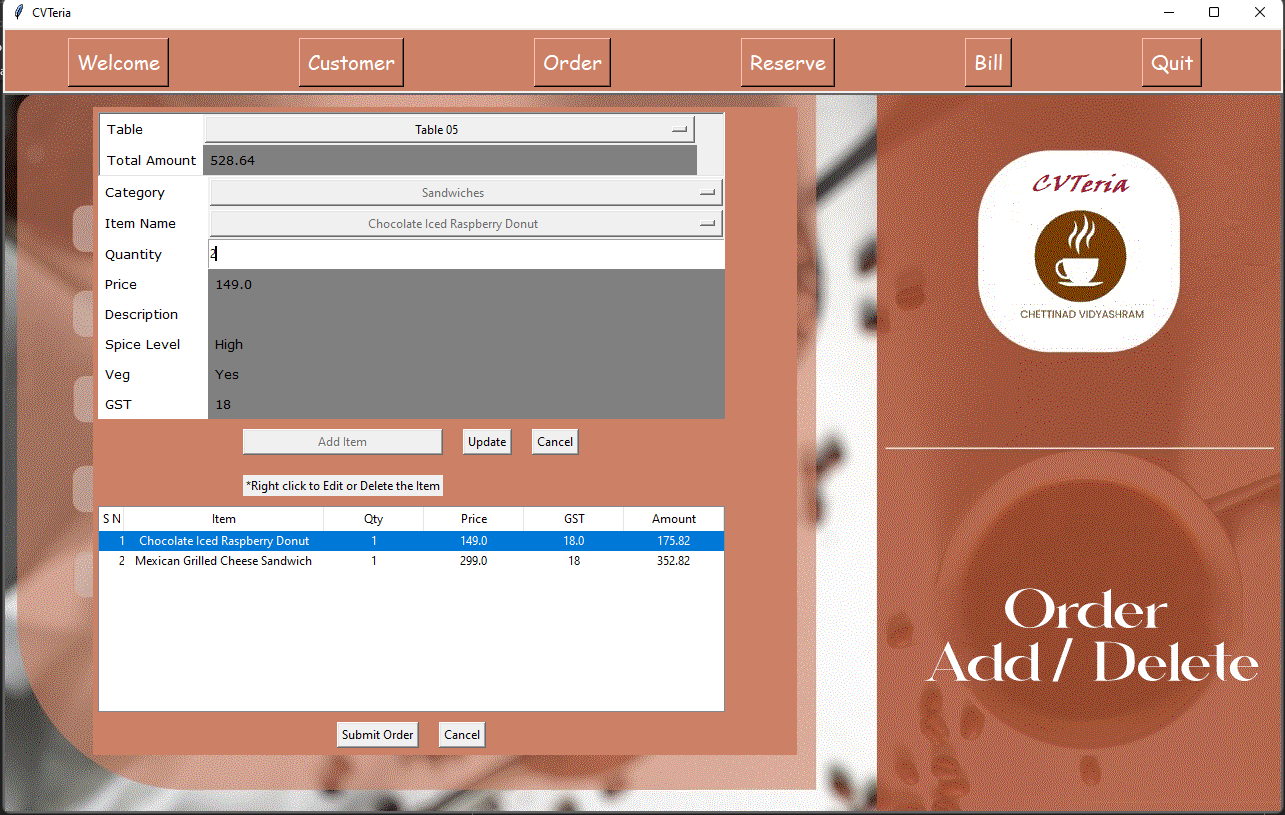
Message boxes are also enabled for every screen.

**Edit Specific Item in an order:**

****

On Editing a specific item, the program only allows you to change the quantity and nothing else

The update button becomes active since we are only editing and not adding a new item…

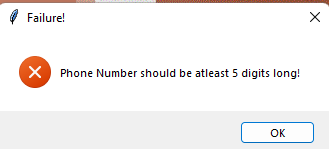
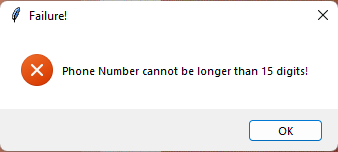
****

**Validation:**

The following validations have been enabled in Customer screen:

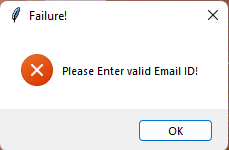
* 1. Phone Number: Phone number should be at least 5 digits long and it can’t be more than 15 digits long.





* 1. Email ID: Email ID should be of the form [abc@abc.com](mailto:abc@abc.com)



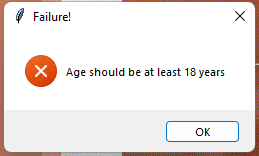


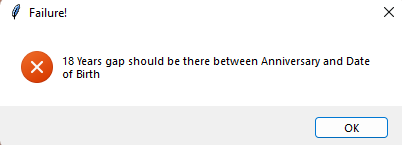
* 1. Date Validation: Minimum Age to create an account in our cafeteria is 18 years.

Difference between birth date and anniversary should be at least 18 years.

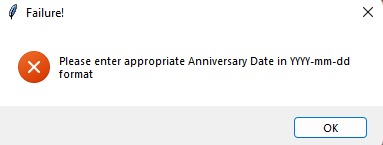
Date should be valid in YYYY-mm-dd format.

Anniversary date cannot be a future date.

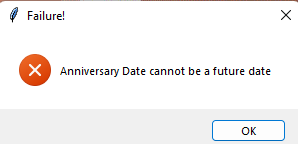








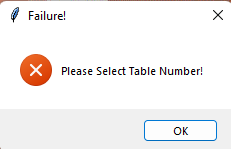




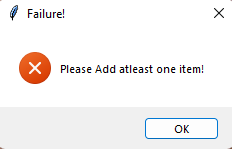


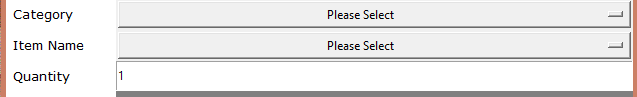
The following validations have been added in order screen:

1. Table Number: One cannot submit order without selecting table number.



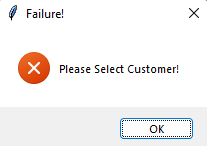


1. Items: The minimum number of items to be added for an order to be placed is 1. 



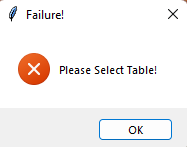
The following validations have been added to table reservation screen:

1. Customer: One cannot reserve a table without selecting which customer.



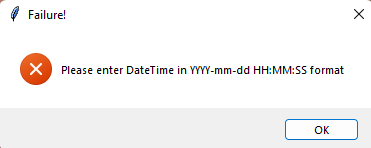


1. Table: One cannot reserve a table without selecting the table.



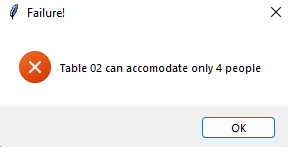


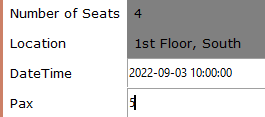
1. Date Time: Date and time should be entered in valid format. Date should be a future date and time.





1. Number of People: The number of people the reservation is being made for should not exceed the number of seats at the table.

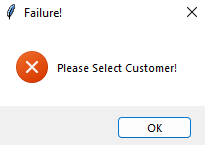




The following validations have been added to Bill screen:

1. Customer: To Create a bill, customer should be selected.





**CONCLUSION**

**Scope Of Improvement:**

The application can be improved by integrating with other Cafeteria Management applications and also further improvement can be made on the GUI with additional features. An online interface can be made available so that it can be accessed from PC and Mobile. The security of the system can further be improved by using other authorization solutions.

**BIBLIOGRAPHY**

1. Stackoverflow.com
2. W3schools.com
3. Javatpoint.com
4. Youtube.com