Software Development and Databases (Comp5000)

Databases and GUI Coursework

Submitted by:

10867436

10886338

10835879

10760993

MSc Data Science and Business Analytics

University of Plymouth

Faculty of Science and Engineering

School of Engineering, Computing & Mathematics

January 2024

Table of Contents

List of Figures

Introduction

A local delivery service for take-out food from restaurants in the USA wants to develop a database to store the information on the orders. Currently the information is stored in csv files, but they want the files to be stored in a database while also writing a GUI to calculate some metrics, plot a chart and update manager information in the database. Two CSV files were provided for this exercise with information below

i. The file restaurant_info.csv contains information about the restaurant. The column headings are: RestaurantID, RestaurantName, Cuisine, Zone, Category, Store, Manager, Years_as_manager, Email,Address.

ii. The file Orders.csv contains information about the restaurant's orders. The column headings are: Order ID, First_Customer_Name, Restaurant ID, Order Date, Quantity of Items, Order Amount, Payment Mode, Delivery Time Taken (mins), Customer Rating-Food, Customer Rating-Delivery, Credit Card, Debit Card, Card provider,Last_Customer_Name

The task required in accomplishing this task involves:

i. Writing a python script to read in the csv files and populate a relational database and perform appropriate data cleaning where necessary.

ii. Develop an additional GUI that has buttons to do the following analysis after the data has been extracted from the database using SQL.

    a. Writing a GUI interface, using for example tkinter, to update the manager of a specific Restaurant to the database.

    b. Calculate the mean Customer Rating-Food for each Restaurant.

    c. Draw a histogram of Delivery Time Taken (mins) for all orders.

iii. Demonstrate that the information about the new manager has been correctly added to the database by including a screenshot.

iv. Provide screenshots to show the statistical analysis part of the GUI working.

Defining the Entity Relationship Diagram for the task

Entity Relationship Diagram

An entity relationship diagram is a vital conceptual data modeling tool that shows the entities, properties, connections, and cardinalities within a system visually. It helps to facilitate communication with stakeholders.

An ERD, according to Dennis, Wixom, and Tegarden (2012), is "a diagram that uses symbols to represent the relationships between things in a system." The following are its primary constituents: Entities are important items or things about which a company wants to maintain records, such as clients, staff, and orders.

An entity's descriptive qualities or traits are called its attributes. A 'customer' entity's properties can include, for instance, its name, contact information, and customer ID.

The connections or affiliations that exist between things are called relationships. The number of instances of one entity connected to instances of the other is known as the cardinality of a relationship. Common cardinalities are one-to-one, one-to-many, and many-to-many. Diamonds are used to show relationships in an ERD diagram.

For this task, the ERD is presented below:

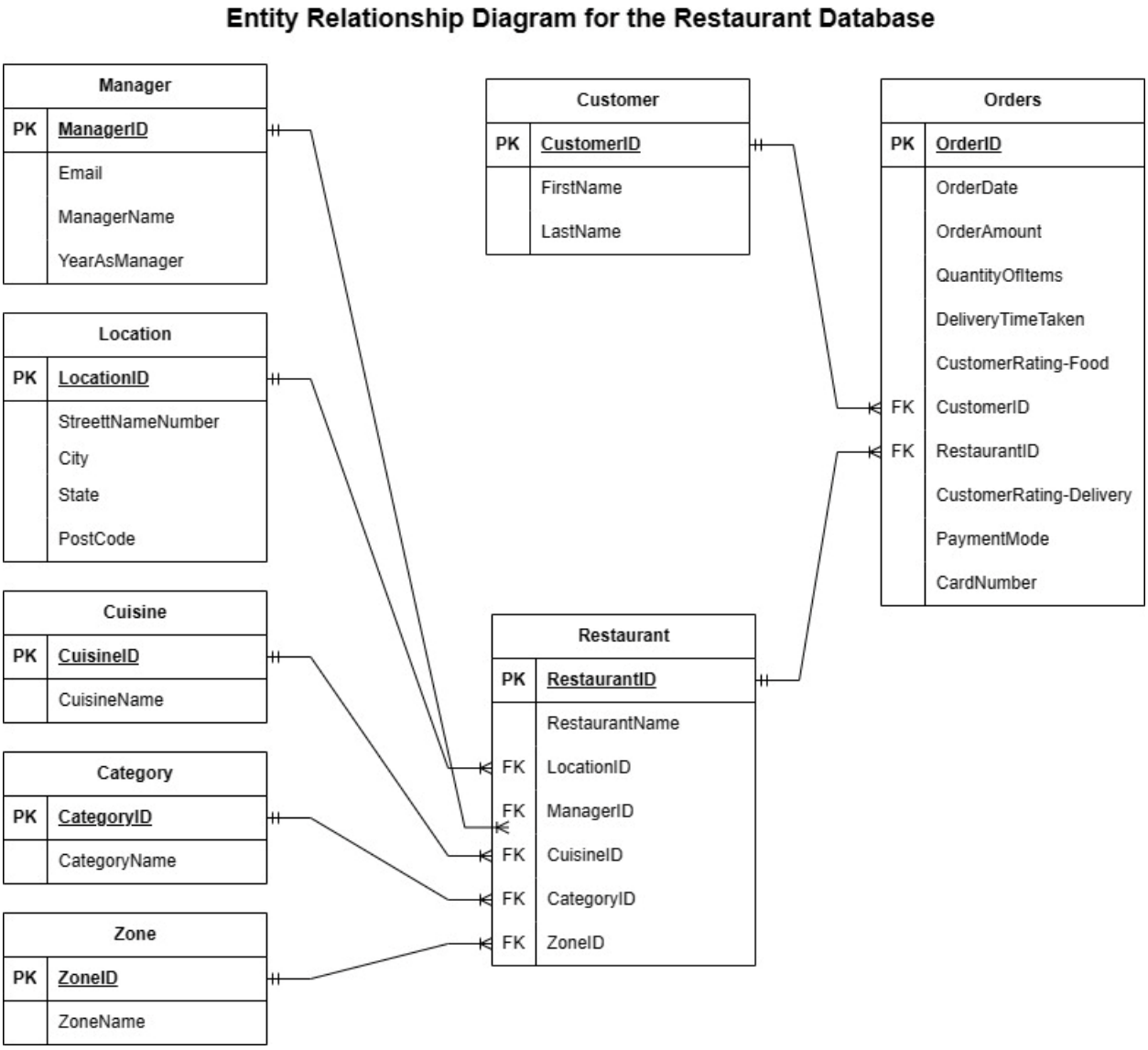

Figure 1 showing the Entity Relationship Diagram for the Restaurant Database

Design Choices for Database

The design choices for a database refers to all the choices and factors that were considered when modeling, architecting, and implementing a database system to best support the applications and use cases that it is intended to enable. The goal is optimal performance, scalability, resilience, and efficiency.

Looking at the design of our database, the following discussions below guide the choice of the database.

Restaurant Table

The primary key is RestaurantID which uniquely identifies each restaurant.

Other attributes like Name, LocationID, CuisineID etc depend only on the RestaurantID, not on each other (avoids transitive dependency).

No non-key attribute depends on another non-key attribute (avoids partial dependency)

Location Table

The primary key is LocationID which uniquely identifies each location.

Attributes like Street, City depend solely on the LocationID, not other attributes.

No partial or transitive dependencies

Manager Table

ManagerID is the primary key, uniquely identifying each record.

Email, Name and YearsAsManager depend only on the ManagerID primary key for their values.

Cuisine Table

CuisineID is the primary key, uniquely determining each cuisine record.

CuisineName's value comes only from CuisineID, and no other field (avoids partial/transitive dependency)

Category Table

Same logic as Cuisine table

CategoryID primary key uniquely determining each record.

No anomalies

Zone Table

ZoneID as sole unique identifier

ZoneName dependent only on ZoneID

Orders Table

OrderID uniquely identifying order records.

Attributes like OrderDate, OrderAmount etc. rely solely on OrderID

Relational integrity maintained via foreign keys to Customer and Restaurant

Customer Table

CustomerID primary key

FirstName and LastName are fully functionally dependent on CustomerID alone.

So, in summary, all tables follow strict 3NF principles by properly handled keys, relationships, dependencies to avoid insertion, update and deletion anomalies.

Description of column names of the table

Table 1 below discusses in detail the description of the columns that were made of the new database that was designed. It emphasized on the table name, the attributes, the type of keys, the data types, and a brief description of these attributes.

Table 1 showing the breakdown of the Entity Relationship Diagram

Entities	Attributes	Key	Data Type	Description
Restaurant	RestaurantID	PK	INT	It uniquely identifies each restaurant.
	RestaurantName		TEXT	It stores the full name of the restaurant
	LocationID	FK	INT	It references to the PK in the Location table, linking restaurant's address details.
	ManagerID	FK	INT	It references to the PK in the Manager table to identify restaurant's manager.
	CuisineID	FK	INT	It references to the PK in the Cuisine table to identify the restaurant's cuisine.

	CategoryID	FK	INT	It references to the PK in the Category table to identify the restaurant's category.
	ZoneID	FK	INT	It references to the PK in the Zone table to identify the restaurant's zone.
Manager	ManagerID	PK	INT	It uniquely identifies each restaurant.
	ManagerName		TEXT	It stores the full name of the manager.
	Year as Manager		INT	It stores the number of years the manager has worked and managed the restaurant operations.
	Email		TEXT	It stores the email of the restaurants manager.
Cuisine	CuisineID	PK	INT	It uniquely identifies each cuisine.
	CuisineName		TEXT	It stores the full name of the cuisine.
Category	CategoryID	PK	INT	It uniquely identifies each category of restaurant.
	CategoryName		TEXT	It stores the name of the restaurant's category.
Zone	ZoneID	PK	INT	It uniquely identifies each restaurant's zone.
	ZoneName		TEXT	It stores the name of the restaurant's category.
Location	LocationID	PK	INT	It uniquely identifies each restaurant's location.
	Street		TEXT	It stores the name of the restaurant's street name and number.
	City		TEXT	It stores the name of the restaurant's city
	State		TEXT	It stores the name of the restaurant's state
	PostCode		TEXT	It stores the post code of the restaurant
Order	OrderID	PK	INT	It uniquely identifies each restaurant's orders.
	OrderDate		DATETIME	It stores the date and time of each restaurant's order.
	OrderAmount		DECIMAL (10,2)	It stores the amount on each customer's order from the restaurants.
	QuantityOfItems		INT	It stores the volume on each customer's order from the restaurants.
	DeliveryTimeTaken		INT	It stores the time taken to deliver the customer's order.
	Customer Rating-Food		INT	It stores the customers' rating on food quality
	Customer Rating-Delivery		INT	It stores the customer's' rating on delivery
	Payment Mode		TEXT	It stores the customers payment mode.
	Card Number		BIGINT	It stores the customers credit/debit card details.

	CustomerID	FK	INT	It references to the PK in the Customer table to identify the customers who placed the orders.
	RestaurantID	FK	INT	It references to the PK in the Restaurant table to identify the restaurants where the orders were placed.
Customer	CustomerID	PK	INT	It uniquely identifies each customer.
	FirstName		TEXT	It stores the first name of the customers.
	LastName		TEXT	It stores the last name of the customers.

Source: Authors Compilation, 2024

The SQL Schema for the Tables Included in the Report

The relational database model for the restaurant management system is represented by this SQL schema. Tables representing a variety of entities were included, including orders, customers, card issuers, locations, managers, restaurants, cuisines, categories, zones, payment information, payment methods, and delivery personnel. Foreign keys were used to create relationships between tables, guaranteeing data integrity. Normalization concepts are incorporated into the design, with each table having a distinct function and reducing data redundancy. Information about orders, customers, payment transactions, and restaurant operations can be easily stored and retrieved thanks to the schema.

Below is the SQL Schema for the tables included in this report.

```
CREATE TABLE IF NOT EXISTS Restaurant(
        RestaurantID INTEGER PRIMARY KEY,
        RestaurantName TEXT,
        LocationID INT,
        ManagerID INT,
        CuisineID INT,
        CategoryID INT,
        ZoneID INT,
        FOREIGN KEY (LocationID) References Location (LocationID),
        FOREIGN KEY (ManagerID) References Manager (ManagerID),
        FOREIGN KEY (CuisineID) References Cuisine (CuisineID),
        FOREIGN KEY (CategoryID) References Category (CategoryID),
```

```
        FOREIGN KEY (ZoneID) References Zone (ZoneID)
    )

CREATE TABLE IF NOT EXISTS Location (
    LocationID INTEGER PRIMARY KEY,
    Street TEXT,
    City TEXT,
    State TEXT,
    PostCode TEXT
  )

CREATE TABLE IF NOT EXISTS Manager (
        ID INTEGER PRIMARY KEY AUTOINCREMENT,
        Email TEXT,
        ManagerName TEXT,
        YearsAsManager INT
      )

CREATE TABLE IF NOT EXISTS Cuisine (
            CuisineID INTEGER PRIMARY KEY,
            CuisineName TEXT UNIQUE
    )

CREATE TABLE IF NOT EXISTS Category (
    CategoryID INTEGER PRIMARY KEY,
    CategoryName TEXT UNIQUE
  )

CREATE TABLE IF NOT EXISTS Zone (
    ZoneID INTEGER PRIMARY KEY,
    ZoneName TEXT UNIQUE
  )

CREATE TABLE IF NOT EXISTS Orders(
    OrderID TEXT PRIMARY KEY,
    OrderDate DATE,
    OrderAmount DECIMAL(10, 2),
    QuantityOfItems INT,
    DeliveryTimeTaken INT,
    CustomerRatingFood INT,
```

```
    CustomerID INTEGER REFERENCES Customer(CustomerID),
    RestaurantID INTEGER REFERENCES Restaurant(RestaurantID),
    CustomerRatingDelivery INT,
    PaymentMode TEXT,
    CardNumber TEXT
    )

CREATE TABLE IF NOT EXISTS Customer(
    CustomerID INTEGER PRIMARY KEY,
    FirstName TEXT,
    LastName TEXT
  )
```

Data Clean Up

The data cleaning process outlined in the provided code involves a series of systematic steps to prepare the dataset for further analysis, ensure data quality, and enhance the reliability of the insights derived from the data. Data cleaning is a crucial part of data preparation in data science and analytics, often consuming significant time and resources in any data-related project (Dasu & Johnson, 2003).

The following cleaning was done on the dataset before eventually creating the database.

i. Replace the blank space in the Order ID with OD187

ii. Replace the blank spaces in the Quantity of Items with the median of the attribute.

iii. Replace the blank column in the Payment Mode with "Credit Card" if the Credit Card column in the CSV has a value.

iv. Replace the blank column in the Payment Mode with "Dedit Card" if the Dedit Card column in the CSV has a value.

v. Replace the blank column in the Payment Mode with "Cash on Delivery" if the Credit Card and Debit Card column is blank.

vi. Replace First_Customer_Name with Unknown where it is blank.

vii. Replace the blank spaces in the Years_as_manager with the median of the attribute.

viii. Replace the blank spaces in the Manager with Unknown

ix. Replace the blank spaces in the EmailAddress with noemail@example.com

The Python Code to Create the Database.

The steps of connecting to a SQLite database, designing tables in accordance with the supplied schema, and actually creating those tables in the database are all automated by this script written in Python. It's a standard procedure when using Python to work with SQLite databases. This has been attached along with the submitted documents. The file containing the python code to create the database has been submitted along with this file; it is labelled Coursework.py.

Defining GUIs for appropriate Tasks

GUI to Update the Restaurant Manager in the Database

This Python code defines the updateManager class, which uses the tkinter package to generate a graphical user interface (GUI). "Update Manager" is the name of the 500x500 pixel GUI window. The class has methods to handle the window closing event, display restaurant information, and create a menu. Please refer to the Coursework.py attachment for the full implementation of GUI

The snapshot of the python code to define the Update Manager in the database is provided below:

```
import sqlite3
import tkinter as tk
from tkinter import ttk, messagebox

class updateManager:
    def __init__(self, root):
        self.root = root
        self.root.title("Update Manager")

        self.root.geometry("500x500")
        self.root.protocol("WM_DELETE_WINDOW", self.closeApplication)

        self.createMenu()
        self.showRestaurant()
```

This screenshot to show how to get started with the update:

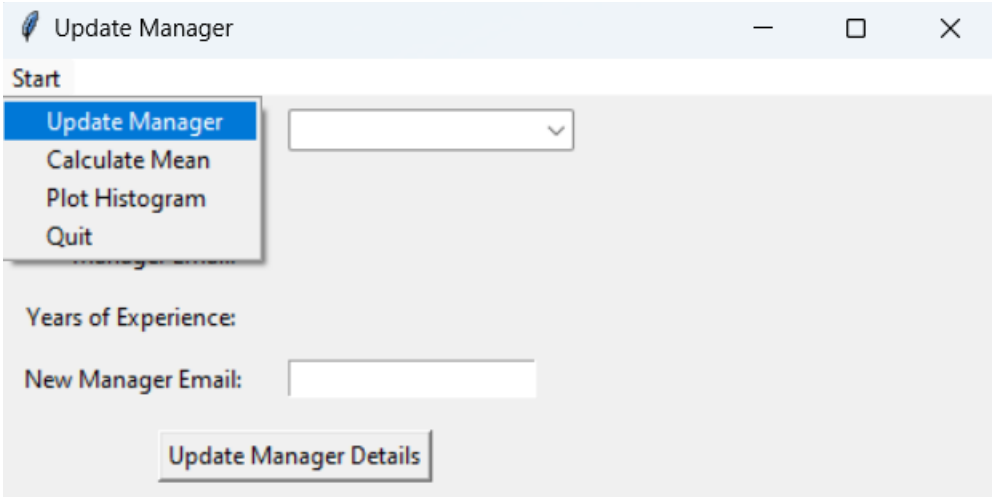

Figure 2 showing how to get started with the update of manager information using the GUI.

This screenshot to show error message if the email isn't entered properly:

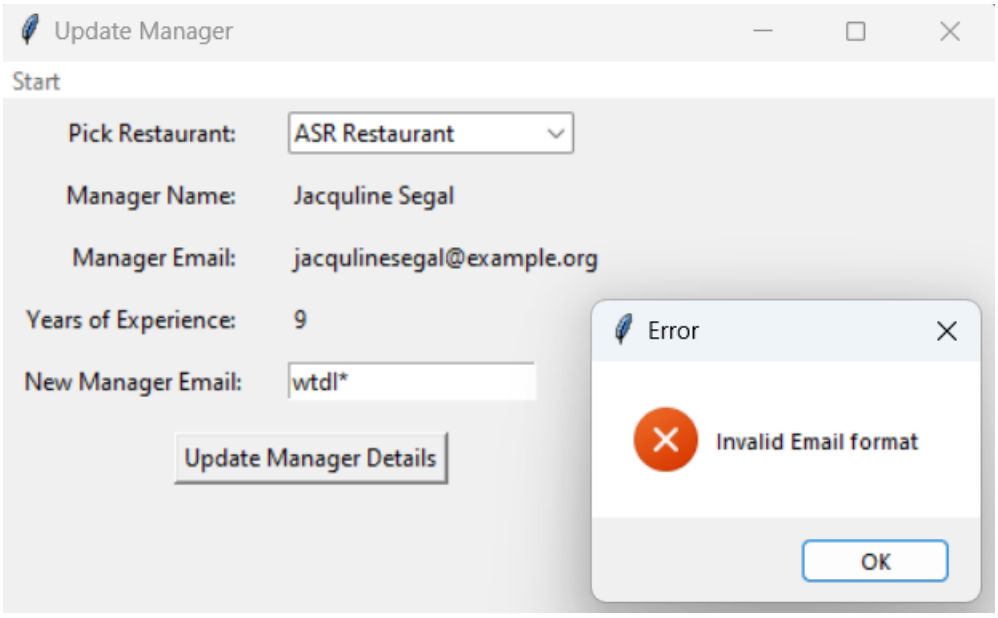

Figure 3 showing unsuccessful update of manager information using the GUI.

This screenshot to show that the GUI is working correctly:

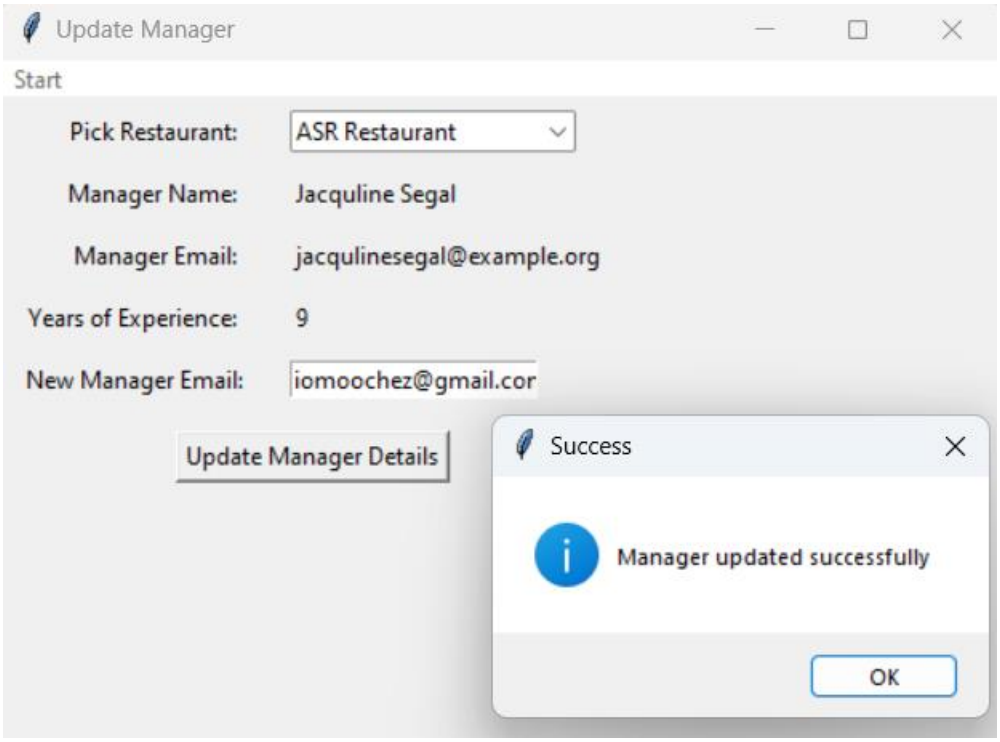

Figure 4 showing unsuccessful update of manager information with error message using the GUI.

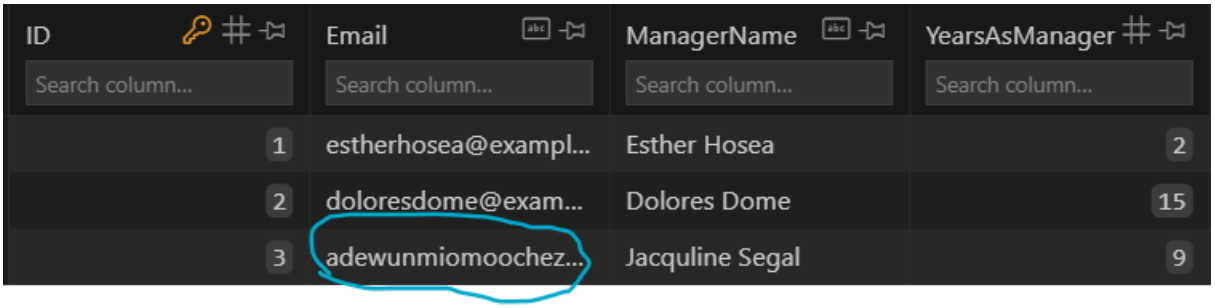

Figure 5 showing new email added to the database.

GUI to produce the Histogram of the Mean Delivery Times.

This Python code defines the draw_histogram function, which establishes a connection to a SQLite database ("Coursework.db"), retrieves the 'DeliveryTimeTaken' data from the 'Orders' table, and then creates and displays a histogram of the delivery times with title and labeled axes using the matplotlib.pyplot library. This function's goal is to give a visual depiction of how delivery times

are distributed across all orders in the given database table. It is a helpful tool that provides insights on the frequency and range of delivery times as well as the effectiveness of the delivery process.

The GUI to produce the Histogram of the Mean Delivery Times is provided below:

```
import sqlite3
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt


def draw_histogram(self):
    conn = sqlite3.connect("Coursework.db")
    cursor = conn.cursor()
    # Perform SQL query to retrieve Delivery Time Taken data
    # Replace 'your_table' with the actual table name
    query = "SELECT DeliveryTimeTaken FROM Orders"
    cursor.execute(query)
    delivery_times = [record[0] for record in cursor.fetchall()]
    print(len(delivery_times))
    # Draw histogram using a plotting library (e.g., matplotlib)
    plt.hist(delivery_times, bins=30, edgecolor='black')
    plt.xlabel("Delivery Time Taken (mins)")
    plt.ylabel("Frequency")
    plt.title("Histogram of Delivery Time Taken for All Orders")
    plt.show()
```

This screenshot also provides proof that the GUI is working correctly:

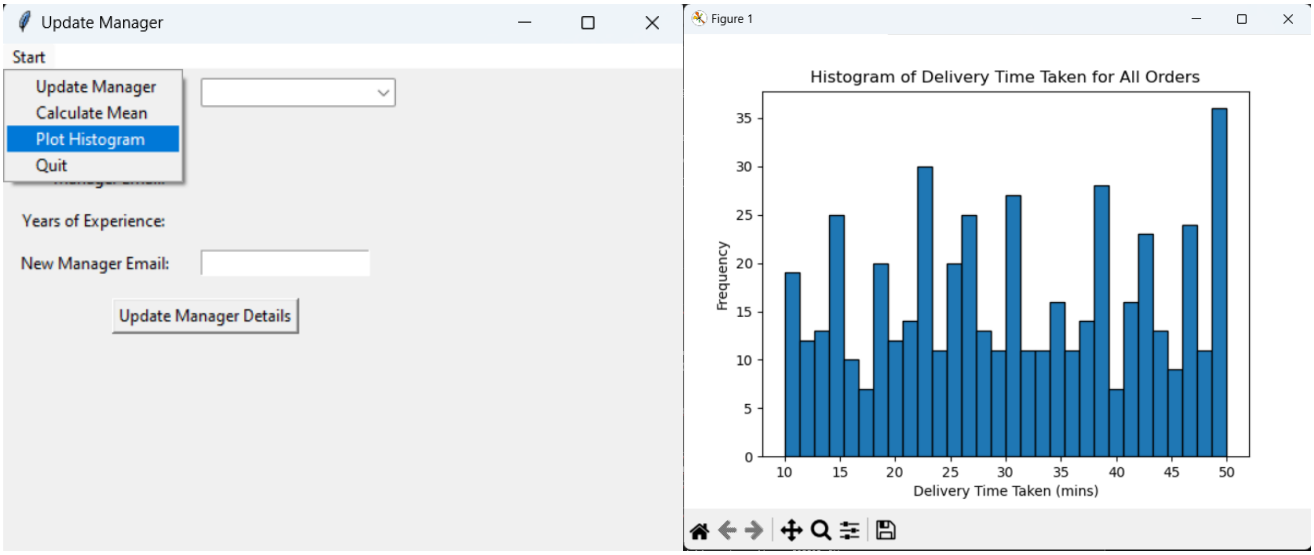

Figure 6 showing histogram of delivery time taken for all orders using the GUI.

GUI to Calculate the Customer Rating-Food for Each Restaurant

The "Calculate Mean" button launches this Python GUI application, which establishes a connection with the "Coursework.db" SQLite database. It joins the "Orders" and "Restaurant" tables to get and compute the average customer food rating for a chosen restaurant. The result updates a variable containing the estimated mean and is shown in the Tkinter GUI. Error handling for possible SQLite database issues is also included in the code.

The GUI to calculate the Customer Rating-Food for Each Restaurant is provided below:

```
import sqlite3
import tkinter as tk
from tkinter import ttk, messagebox

def calculateMean(self):
    selectedRestaurant = self.restaurantDropdown.get()

    try:
        conn = sqlite3.connect("Coursework.db")
        cursor = conn.cursor()

        cursor.execute('''
```

```
        Select AVG(CustomerRatingFood)
        FROM Orders
        Join Restaurant
        ON Orders.RestaurantID = Restaurant.RestaurantID
        WHERE Restaurant.RestaurantName = ?
    ''', (selectedRestaurant,))

    mean = cursor.fetchall()

    if(mean):
        self.foodRatingVariable.set(round(mean[0][0], 2))
    else:
        print("No mean found")

    conn.close()

except sqlite3.Error as e:
    messagebox.showerror("Error calculating mean", str(e))
```

This screenshot to show how to get started with the average calculation of food rating:

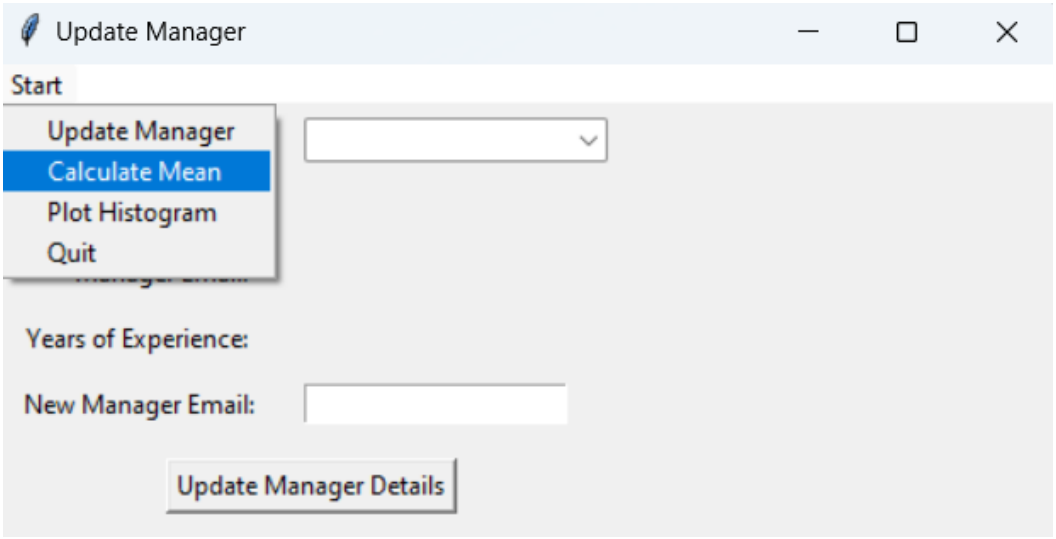

Figure 7 showing how to get started with the average food rating calculation.

This screenshot showing how the mean is calculated for selected restaurant:

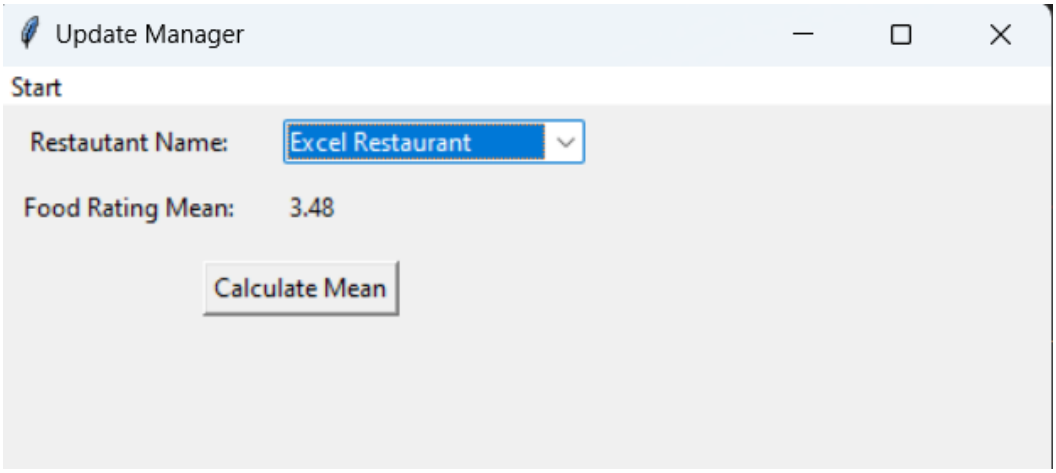

Figure 8 showing successful calculation of mean food rating for selected restaurant using the GUI.

Updated Database Design to Include the Delivery Staff.

The diagram above shows the new entity relationship diagram for the restaurant database having updated it with the delivery table. It requires some readjustments to the earlier defined schema to conform with the 3NF, by doing this, you can make sure that data is arranged to reduce dependencies and redundancies, which strengthens the database and reduces the likelihood of anomalies during data change processes.

The new database design is provided below:

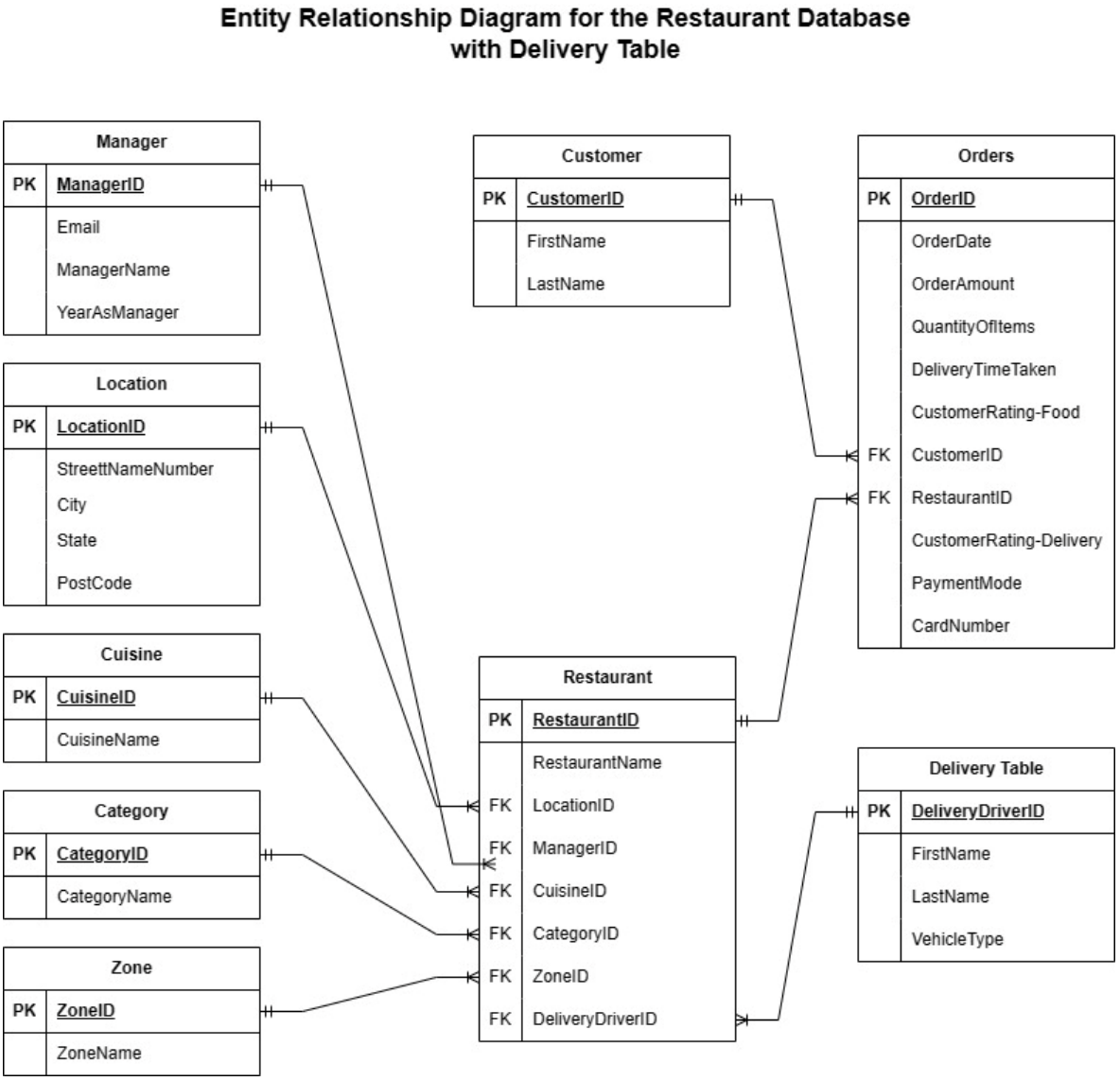

Figure 9 showing the Entity Relationship Diagram for the Restaurant Database with the inclusion of the Delivery Table.

References

Dasu, T., & Johnson, T. (2003). Exploratory Data Mining and Data Cleaning. John Wiley & Sons.

Dennis, A., Wixom, B.H., & Tegarden, D. (2012). Systems analysis & design: An object-oriented approach with UML. John Wiley & Sons.