COMP5000 SOFTWARE DEVELOPMENT AND DATABASES

COURSEWORK SUBMISSION

STUDENT ID: 10883408, 10638448

Table of Contents

List of Figures

1 Group details

Group member IDs and names are as follows,

1. 10883408
2. 10638448

2 Introduction

In an attempt to transform its flow of information, the take-out food local delivery service organization from restaurants in the USA has undergone a series of changes. In response to the increasing need for uncomplicated and systematic handling of data, the delivery service intends to shift from its CSV file dependence to a powerful database with good structure. This assignment provides a narrative of such transition steps as Python programming, MySQL Workbench use for data visualization and manipulation, and cleaning methods given to figure out how much consumers paid in different shops throughout California. GUI design corresponds accounting information about the real vase amount lent with statistical analysis including cash shortages due to services rendered vis payer`s credit risk

2.1 Current Data Storage Challenges

Now service handles its order-related information using CSV files. But admitting the restraints of this method, the choice has been made to port over for database infrastructure. The coursework provides a detailed instruction on how to execute this migration, which will be illustrated in the subsequent section. Two main CSV files, Orders.csv and restaurant_info.csv will form the solid base of datasets that are published in MySQL Workbench database as transformed data sets.

2.2 Steps for Implementation

The assignment narrates in detail the steps followed during this process from initially Python scripts that were used to create and fill MySQL Workbench data. In the compliance, correct and accurate data cleaning techniques that have been applied to ensure this achieved. The main concern is how to create a user interface guided by the Tkinter framework. The task that this GUI performs is to ensure the updates information of any restaurant managers, so as to seek a smooth and speedy contact.

2.3 Statistical Analysis and Growing the System

In the training, more than data warehousing is involved; it extends to include statistical analysis. In addition, a second interface is targeted at finding the average summative customer review for every eatery and generating histograms illustrating order delivery time. It is the additional functionality that reveals substantially how flexible and scientific the created mechanism can be.

2.4 Database Design Enhancement

The injury isn't limited to a bare replication of existing structures, rather it seeks the enhancement of database information which comprises staff names and types: car, motorbike or others. Although the course content suggests an updated design of the database to consider these additions, Python code implementation lacks delivery staff tables due to determining them as extra for this work.

2.5 Learning Outcomes and Framework Suitability

An integral part of the coursework is consideration for appropriateness towards choice's unification and database design following Learning Outcomes specifications corresponding to evaluation. In the software development process, Python sits on a pedestal for not being just scripting but enough to do relational database implementation. The underlying aim is to show the group's skill or capability of building and implementing a moderately high-minded computer program, thereby exhibiting competence in software engineering know-how database designing, and system testing.

2.6 Conclusion

In the development of this coursework, it should be noted that such a view reflects basically how local delivery service evolved towards an organized and effective database control system. In terms of software development, database design and testing systems, the group's deep understanding as well as capability is portrayed by how after learning theoretical approaches in principle application, they executed their practical use. In the end, local delivery service is geared to resolve a trend that offers effective management of information through streamlining operations and experience for both customers and restaurant managers.

3 Normalizing with Entity Relationship

3.1 Finalized Entity Relationship Diagram and the Assumptions it posits

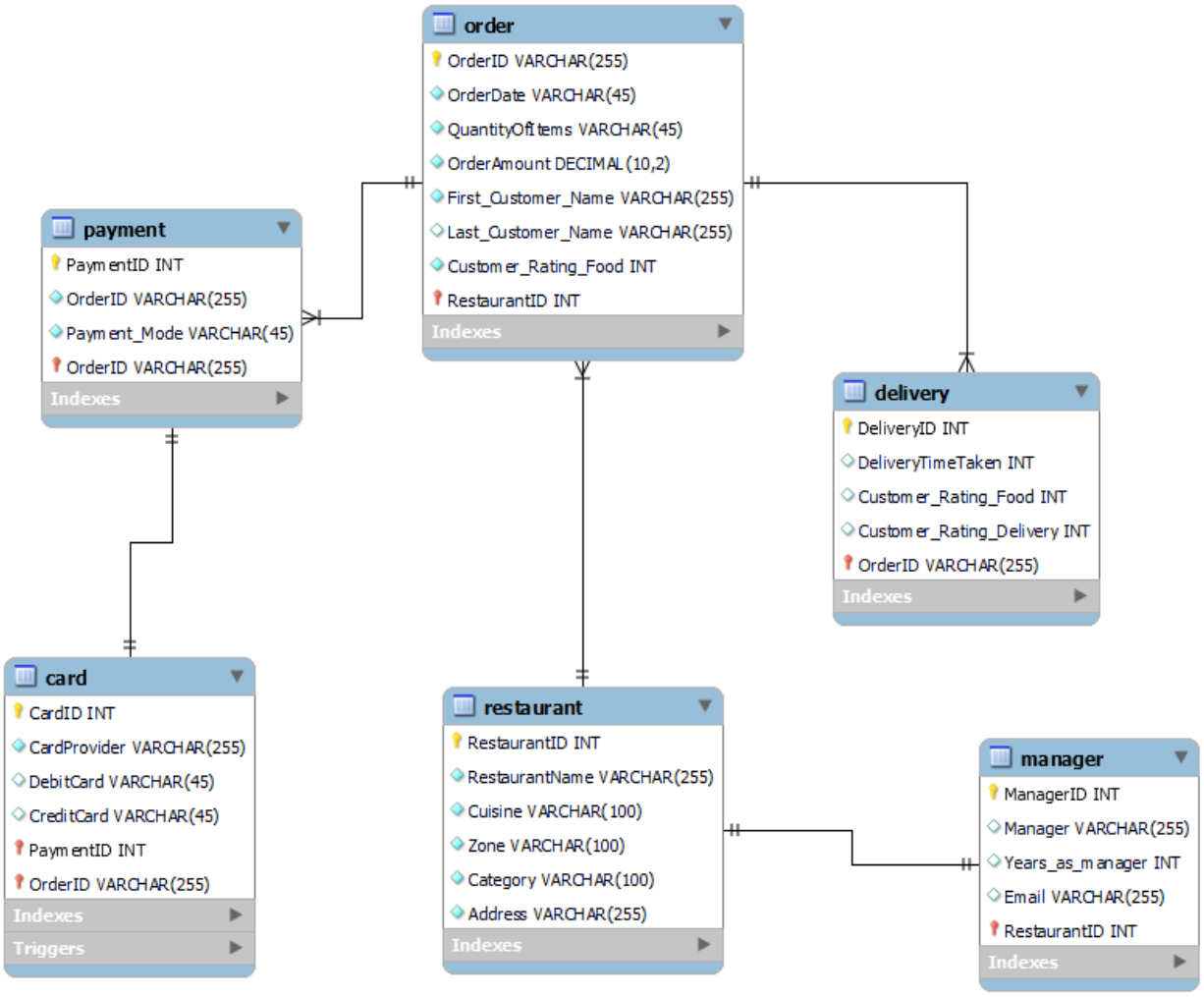

Figure 3.1.1-1Final ERD

3.1.1 Restaurant Entity

Assumptions:

- The OrderID is used to identify each order as the primary key.
- It is assumed that the combination of OrderDate, QuantityOfItems, OrderAmount , First_Customer_Name and Last _Customer_Name defines each order in a unique way.
- RestaurantID indicates the foreign key relationship with a “restaurant” entity. This implies that each order is assigned to a particular restaurant.
- It is assumed that First_Customer_Name and Last_Customer_Name are the given names of customers who would make an order.
- Customer_Rating_food is taken to be a customer’s satisfaction or rating with reference to food in the order.

Measures:

- OrderDate: Represents the date when the order was placed.
- QuantityOfItems: Indicates the total quantity of items ordered.
- OrderAmount: Represents the total monetary value of the order.
- First_Customer_Name and Last_Customer_Name: Identify the customer placing the order.
- Customer_Rating_Food: Quantifies the customer's rating or satisfaction with the food in the order.

Relationships:

- Many-to-1 Relationship (Order to Restaurant):
    a. Many orders can be associated with one restaurant (many orders to one restaurant).
    b. The foreign key RestaurantID in the "orders" entity links to the primary key of the corresponding restaurant in the "restaurant" entity.
    c. This relationship suggests that each order is placed at a specific restaurant, creating a connection between the "orders" and "restaurant" entities.

- 1-to-Many Relationship (Order to Payment):

a. A single order can be associated with multiple payments. This could be due to various reasons, such as split payments or partial payments for the order.

- 1-to-Many Relationship (Order to Delivery):
    a. A single order can be associated with multiple deliveries. This could happen if the order is delivered in multiple shipments or stages.

3.1.2 Order Entity

Assumptions:

- The primary key that identifies each restaurant is the RestaurantID.
- The uniqueness of RestaurantName, Cuisine, Zone, Category and Address for every restaurant is hypothesised.
- The entity "manager" has a one-to-one relationship to the entity 'restaurant': 1 -From. This implies that every restaurant has a single manager and every manager is linked with one particular restaurant only.

Measures:

- RestaurantName: It reflects the name of a restaurant.
- Cuisine: Indicates what kind of restaurant it is.
- Zone: Locates where the restaurant is geographically.
- Category: Specifies the restaurant's category or type (e.g., fine dining, fast food).
- Address: Location of the restaurant.

Relationships:

- One-to-One Relationship with Manager Entity:
    a. Every restaurant has a one-to-one relationship to the entity "manager".
    b. This implies that every restaurant is run by one person and only, whereas each of these managers oversees a single restaurant.
    c. The relationship is likely to be established via a foreign key either in the restaurant or manager entity which will reference the primary key of some other domain.

d. This has the effect of directly and uniquely associating a particular restaurant with its manager, in relation that is manifestively closely coupled.

3.1.3 Delivery Entity

Assumptions:

- DeliveryID serves as the key for each delivery.
- DeliveryTimeTaken is the time which was taken for delivery.
- Customer_Rating_Food is an indicator of how the customer rated the food in that order.
- Customer_Rating_Delivery is a metric for customer rating in the provision of delivery services.
- The OrderID is a foreign key of the linking delivery entity to order entity.

Measure:

- DeliveryID (Primary Key): Unique identifiers per delivery.
- DeliveryTimeTaken: The duration of the delivery.
- Customer_Rating_Food: A rating value for the food of that order.
- Customer_Rating_Delivery: The metric is customer rating for the delivery service.
- OrderID (Foreign Key): Relates the delivery entity to the order entity.

Relationships:

- One-to-Many Relationship (Order to Delivery):
    a. It is possible to associate with a single order several deliveries. This means that an order may have multiple delivery attempts or stages. This relationship is established by the foreign key OrderID in the delivery entity.

- Independent Rating Relationships:
    a. Customer_Rating_Food: This is an order food related measure. It is separate from the delivery process and belongs to order entity.
    b. Customer_Rating_Delivery: This is a metric associated with the delivery service. It is applicable to the delivery process and forms part of its delivery entity.

- Many-to-One Relationship (Delivery to Order):
    a. Many deliveries can be linked to one order. This is evident from the foreign key OrderID in delivery entity.

3.1.4 Manager Entity

Assumptions:

- Each manager has its ManagerID, which acts as the primary key.
- Manager is the manager's name or identifier.
- Years_As_Manager is the number of years manager has in position.
- The e-mail address of the manager is email.
- RestaurantID is a foreign key that connects between the manager entity and restaurant's one.
- There is a one to-one relationship between the manager entity and restaurant link implying each manager corresponds only with an establishment.

Measures:

- ManagerID (Primary Key): A manager identification for each of the managers.
- Manager: The manager's name or identifier.
- Years_As_Manager: The number of years the manager has served in that position.
- Email: The manager's email address.
- RestaurantID (Foreign Key): Connects the manager to restaurant relationship.

Relationship:

- One-to-One Relationship (Manager to Restaurant):
    a. Every manager is linked to one restaurant. This suggests an intimate relationship between one manager and in particular restaurant. The one-to-one relationship is created through the RestaurantID foreign key in manager entity.

3.1.5 Payment Entity

Assumptions:

- Each payment has a PaymentID key that is unique.
- Payment_Mode stands for mode of payment (e.g., credit card, cash etc.).
- OrderID acts as a foreign key, providing the connection from payment entity to order entity.
- The card entity and payment entity relationship are assumed to have a one-to-one mapping

Measures:

- PaymentID (Primary Key): Each payment must carry a unique identifier.
- Payment_Mode: The payment method employed in the transaction (credit card, cash etc.).
- OrderID (Foreign Key): Associates the payment entity with the order entity.

Relationships:

- One-to-One Relationship (Payment to Card):
    a. This relationship indicates that each payment is connected with one and only card. Therefore, this may imply that the payment information including credit card details are saved in a different entity for the cards and each of the payments corresponds with one particular card through relations 1:1.

3.1.6 Card Entity

Assumptions:

- Each card has a CardID as its primary key.
- CardProvider is an attribute that represents the issuer or provider of a card.
- DebitCard is a characteristic which represents whether the card in question is debit or not.
- CreditCard is an attribute that shows whether the card belongs a credit one or not.
- card entity is linked to the payment and order entities via foreign keys, PaymentID and Order ID in this case.

Measures:

- CardID (Primary Key): A ID for all cards.
- CardProvider: The card provider or issuer.
- DebitCard: States whether the card is a debit.
- CreditCard: Shows if the card is a credit one.
- PaymentID (Foreign Key): Associates the card entity with the payment entity.
- OrderID (Foreign Key): Connected the card entity with the order entity.

Relationships:

- One-to-One Relationship (Payment to Card):
    a. This relationship shows that each payment is linked with one and only card. The card entity's PaymentID functions as a reference key that joins it with the payment entity.

- Many-to-One Relationship (Order to Card):
    a. This association implies that many orders can be linked to a card. The OrderID foreign key in the card entity links it to orderl entity.

3.2 Tidying Up the Entity-Relationship Diagram (ERD) After Resolving 1-to-1 Relationships

3.2.1 Entities:

- Restaurant Entity:
  a. Attributes: RestaurantID (PK), Name, Cuisine Type Zone or Category Address.
  b. Relationships:
     i. Many-to-1 with Order Entity (OrderEntity which has RestaurantID as FK).
     ii. A 1-to-1 relationship with Manager Entity (RestaurantID as FK in ManagerEntity)

- Order Entity:
  a. Attributes: OrderID (PK) [Order Date, Quantity of Items, Order Amount], First_Customer Name LastName Customer Rating Food RestaurantID.
  b. Relationships:
     i. Many-to-1 with Restaurant Entity (RestaurantID as Foreign Key in Order Entity).
     ii. 1-to Many Payment Entity (OrderID as Foreign Key payment entity).
     iii. 1-to-Many with Delivery Entity (Delivery ID as FK in the Order table).

- Delivery Entity:
  a. Attributes: DeliveryID (PK), TimeTaken, Customer_Rating_Food, and Customer Rating-Delivery; fields are based on the order ID in this table.
  b. Relationships:
     i. Many-to-1 with Order Entity (OrderID as FK in Delivery Entity).

- Manager Entity:
  a. Attributes: ManagerID(PK), Manager, Years_As_Manager Email RestaurantID (FK).
  b. Relationships:
     i. 1-to-1 with Restaurant Entity (RestaurantID as FK in Manager entity).

- Payment Entity:
  a. Attributes: PaymentID (PK), Payment_Mode, OrderID (FK).
  b. Relationships:
    i. 1-to-Many Payment entity with Order Entity (OrderID as FK in Payment table).
    ii. 1-to-1 Card Entity with PaymentID as FK.

- Card Entity:
  a. Attributes: CardID (PK), CardProvider, DebitCard, CreditCard, PaymentID FK); Order ID(FK).
  b. Relationships:
    i. 1-to-1 with Payment Entity (FK of paymentid in Card entity).
    ii. Many-to-1 with Order Entity (OrderID as FK in Card entity).

3.2.2 Notations:

- PK: Primary Key
- FK: Foreign Key

3.2.3 Cardinality and Participation Constraints:

- Restaurant to Manager:
  a. Cardinality: 1-to-l connection (There are exactly one manager for each restaurant and vice versa).
  b. Participation: Total (1 manager equals to 1 restaurant).

- Order to Restaurant:
  a. Cardinality: Many-to-1 (One restaurant can be link to many orders).
  b. Participation: Total (Each order must be linked with a restaurant).

- Order to Payment:
  a. Cardinality: 1-to-Many (One order can have many payments).
  b. Participation: Partial (Not all orders can be paid for).

- Order to Delivery:
  a. Cardinality: 1-to-Many (“one order can be linked with several deliveries”).
  b. Participation: Partial (not all orders have a delivery).

- Manager to Restaurant:
  a. Cardinality: 1-to-1 (Manager one to one restaurant).
  b. Participation: Sum (Managers must be affiliated with a restaurant, and each restaurant needs to have manager).

- Payment to Card:
  a. Cardinality: 1-to-1
  b. Participation: Total (Each payment must be linked to a card).

- Order to Card:
  a. Cardinality: Many-to-1(Many orders are connected with a single card).
  b. Participation: Partial (Not all orders will have a card).

3.3 Resolving many-to-many relationships in the ERD

There are no explicit many-to-many relationships in this ERD.

4 Discussion of database design decisions.

The database design for the Restaurant Management System adheres to relational model principles and normalization techniques, ensuring an organized and efficient structure for data management. Here's a discussion of the design choices:

- Relational Model Principles and Normalization:
  a. The design follows the relational model principles by organizing data into subject-based tables. This approach eliminates redundancy and dependency issues, ensuring that each piece of data is stored in only one place, minimizing data duplication.
  b. Normalization is employed to structure the tables efficiently, avoiding data anomalies and improving maintainability. The use of Normal Forms (up to the third normal form, 3NF) helps address transitive dependencies, ensuring data integrity.

- Indexing for Query Performance:
  a. Indexing is employed to enhance query performance. By creating indexes on relevant columns, the database can quickly locate and retrieve specific data, speeding up search operations. This is particularly beneficial for large datasets or frequently queried columns.

- Primary and Foreign Keys for Data Accuracy and Integrity:
  a. Primary keys are implemented to uniquely identify each record in a table, ensuring data accuracy and preventing duplicate entries. Foreign keys establish relationships between tables, enforcing referential integrity. This guarantees that data in related tables remains consistent and accurate.

- Comprehensive Analysis and Reporting:
  a. The design enables effective joins between tables, facilitating comprehensive analysis and reporting. By establishing relationships through primary and foreign keys, data from different entities can be combined to derive meaningful insights. This is crucial for generating reports and conducting detailed analyses.

- Scalability and Adaptability:
    a. The database design can be scaled and modified to support the future changes in the growth of service. Emphasizing normalization reduces denormalization, as a result of which the database remains flexible and maintainable. This enables the system to grow and expand with the increasing demands of service provision.

- Robustness and Structured Information Management:
    a. The overall architecture is stable, giving an organized approach for storing and sorting information. This model guarantees accurate data capture, efficient storage and rapid retrieval. The use of keys, relations and normalization rules makes the management process become more reliable.

To sum up, the Restaurant Management System's database design can be considered an adequately developed solution that conforms to fundamental ideas of relational databases. It guarantees effective data management, accuracy and authenticity to meet transactional as well as analytical needs. The scalability and versatility of the design serve as a launching pad for future demands that may emerge on behalf of delivery solution.

5 The explanation for the column names

Each column was designed to be easy to recognize and represent the values they represent. The Camel Case was chosen as the naming convention notation.

Each table are explained in more information below.

Order

- OrderID (Primary Key): Essential unique identifier for each order.
- RestaurantID (Foreign Key): The Associated restaurant identifier.
- OrderDate: Date the order was placed.
- QuantityofItems: Quantity of items in the order.
- OrderAmount: Total payment for the order.
- First_Cutomer_Name: First Name of the Customer
- Last_Customer_Name: Last Name of the Customer

Manager

- ManagerID (Primary Key): Essential unique identifier for each manager.
- RestaurantID (Foreign Key): The Associated restaurant identifier.
- Manager: Name of the Manager.
- Years_as_manager: The Manager's experience as a worker in years, measured in 1-year stages.
- Email: Email Address f the Manager.

Restaurant

- RestaturantID (Primary Key): Essential unique identifier for each Restaurant.
- RestaturantName: Name of the Restaurant.
- Cuisine: Cuisine options include (French, NorthIndian, SouthIndian, and Continetal)
- Zone: Geographic area identity (Zone A, B, C, or D).
- Category: Classification as 'pro' or 'ordinary.
- Address: Address of the restaurant.

Payment

- PaymentID (Primary Key): Essential unique identifier for each Payment.
- OrderID (Foreign Key): The Associated order identifier.
- Payment_Mode: The Payment methods. Payment options include Debit Card, Credit Card, and Cash on Delivery

Delivery

- DeliveryID (Primary Key): Essential unique identifier for each Delivery.
- OrderID (Foreign Key): The Associated order identifier.
- DeliveryTimeTaken: Minutes are taken to deliver the order.
- Customer_Rating_Food: The customer's ranking regarding food.
- Cutomer_Rating_Delivery: The customer's ranking regarding Delivery Service.

Card

- CardID (Primary Key): Essential unique identifier for each Card.
- Card_Provider: Indicates Card Service Provider.
- Debit_Card: Type of the Card.
- Credit_Card: Type of the Card.
- PaymentID: (Foreign Key): The Associated Payment identifier.
- OrderID (Foreign Key): The Associated order identifier.

6 The SQL schema for all the tables

The schema created by using MySQL workbench.

```
CREATE TABLE `delivery` (
  `DeliveryID` int NOT NULL AUTO_INCREMENT,
  `OrderID` varchar(255) DEFAULT NULL,
  `DeliveryTimeTaken` int DEFAULT NULL,
  `Customer_Rating_Food` int DEFAULT NULL,
  `Customer_Rating_Delivery` int DEFAULT NULL,
  `Delivery_Staff_Name` varchar(255) DEFAULT NULL,
  `Delivery_Vehicle` enum('car','bike','motorbike') DEFAULT NULL,
  PRIMARY KEY (`DeliveryID`),
  KEY `orderid2_idx` (`OrderID`),
  CONSTRAINT `order22` FOREIGN KEY (`OrderID`) REFERENCES `orders` (`OrderID`))

CREATE TABLE `cards` (
  `CardID` int NOT NULL AUTO_INCREMENT,
  `CardProvider` varchar(255) DEFAULT NULL,
  `DebitCard` varchar(45) DEFAULT NULL,
  `CreditCard` varchar(45) DEFAULT NULL,
  `PaymentID` int DEFAULT NULL,
```

```
  PRIMARY KEY (`CardID`)
)

CREATE TABLE `manager` (
  `ManagerID` int NOT NULL AUTO_INCREMENT,
  `RestaurantID` int DEFAULT NULL,
  `Manager` varchar(255) DEFAULT NULL,
  `Years_as_manager` int DEFAULT NULL,
  `Email` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`ManagerID`),
  KEY `RestaurantID2_idx` (`RestaurantID`),
  CONSTRAINT `RestaurantID2` FOREIGN KEY (`RestaurantID`) REFERENCES `restaurants` (`RestaurantID`)
)

CREATE TABLE `orders` (
  `OrderID` varchar(255) NOT NULL,
  `RestaurantID` int DEFAULT NULL,
  `OrderDate` varchar(255) DEFAULT NULL,
  `QuantityOfItems` varchar(45) DEFAULT NULL,
  `OrderAmount` decimal(10,2) DEFAULT NULL,
```

```
  `First_Customer_Name` varchar(255) DEFAULT NULL,
  `Last_Customer_Name` varchar(255) DEFAULT NULL,
  `Customer_Rating_Food` int DEFAULT NULL,
  PRIMARY KEY (`OrderID`),
  KEY `restauranID1_idx` (`RestaurantID`),
  CONSTRAINT `restauranID1` FOREIGN KEY (`RestaurantID`) REFERENCES `restaurants` (`RestaurantID`)
)

CREATE TABLE `payments` (
  `PaymentID` int NOT NULL AUTO_INCREMENT,
  `OrderID` varchar(255) DEFAULT NULL,
  `Payment_Mode` varchar(45) DEFAULT NULL,
  `CardID` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`PaymentID`),
  KEY `order1_idx` (`OrderID`),
  KEY `orderid1_idx` (`OrderID`),
  KEY `orderid111_idx` (`OrderID`),
  KEY `OrderID_idx` (`CardID`)
)
```

```
CREATE TABLE `restaurants` (
  `RestaurantID` int NOT NULL,
  `RestaurantName` varchar(255) DEFAULT NULL,
  `Cuisine` varchar(100) DEFAULT NULL,
  `Zone` varchar(100) DEFAULT NULL,
  `Category` varchar(100) DEFAULT NULL,
  `Address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`RestaurantID`),
  KEY `ratings_idx` (`RestaurantName`)
)
```

7 The explanation of the data cleaning process

7.1 Store column removal from restaurant table

```
#Drop the 'Store' column from the 'restaurants' data frame
restaurant_df.drop(columns=['Store'], inplace=True)
```

Figure 3.2.3-1Store column removal from restaurant table

There is a store column in the restaurants table of the CSV. Following the convention of using the primary key derived from the entity or table name. After executing this code with inplace=True, the 'Store' column will be removed from the restaurant_df DataFrame

7.2 Replace missing values

```
# replace missing values
restaurant_df.fillna({'Years_as_manager': 0, 'Email': 'unknown'}, inplace=True)


# missing values in the 'Delivery_Time_Taken_mins' column of orders_df will be filled with the mean value of that column
orders_df.fillna({'Delivery_Time_Taken_mins': orders_df['Delivery_Time_Taken_mins'].mean()}, inplace=True)
```

Figure 3.2.3-1Replace missing values

In our data preprocessing step, missing values were addressed in two different datasets relevant to our restaurant analysis. For the 'restaurant_df', which contains managerial information, missing entries in the 'Years_as_manager' column were replaced with' 0' to indicate no experience. Missing email addresses were simultaneously replaced with' unknown'. This makes our dataset robust enough to be used without discarding records due to incomplete data.

For the 'Delivery_Time_Taken_mins' column we used a different approach in the' orders _ df' (which tracks delivery details). Here missing values were replaced by the average time for deliveries based on existing non-null entries in the column. This method of imputation preserves the overall distribution of delivery times and avoids bias from biasing our analysis, especially when performing performance metrics or forecasting delivery times.

```
#'Years_as_manager' column in the restaurant_df DataFrame will contain integer values, and any non-numeric or missing values will
restaurant_df['Years_as_manager'] = pd.to_numeric(restaurant_df['Years_as_manager'], errors='coerce').fillna(0).astype(int)
```

Figure 3.2.3-2Replace missing values

This column will contain integer values representing the number of years someone has been a manager. The pd.to _ numeric () function is called to convert entries in 'Years _ as _ manager' to a numeric data type.

7.3 Email validation

```
# Clean data and validate email format
def is_valid_email(email):
    # Define a regular expression pattern to validate the email format
    pattern = r'^[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+$'
    return re.match(pattern, email) is not None

# Function to clean the 'Email' column in restaurant_df
def clean_email(email):
    if is_valid_email(email):
        return email
    else:
        return 'Invalid Email'

restaurant_df['Email'] = restaurant_df['Email'].apply(clean_email)
```

Figure 3.2.3-1Email validation

In particular, when using email addresses for marketing or communication, this validation is essential to preserving data integrity. Erroneous or badly structured email submissions may result in unsuccessful communications and inaccurate data, which is why this cleaning stage in the data analysis pipeline is required.

7.4 Card number cleaning

```
#Clean credit card and Debit card columns
orders_df['Credit_Card'] = orders_df['Credit_Card'].astype(str).str.replace(r'\D', '', regex=True)
orders_df['Debit_Card'] = orders_df['Debit_Card'].astype(str).str.replace(r'\D', '', regex=True)
```

Figure 3.2.3-1Card number cleaning

In the provided code snippet we are cleaning the credit. Debit card information, in the 'orders_df' dataframe. To achieve this we convert the 'Credit_Card' and 'Debit_Card' columns into strings. Remove any digit characters (represented by 'D' in the regular expression). This cleaning process ensures that the credit and debit card numbers are represented as strings without any extra formatting characters such, as spaces, hyphens or letters. By applying this code to our dataset we standardize the format of the card number fields to maintain data consistency perform validation checks and prepare it for analysis or automated processing.

7.5 Delivery Time taken

```
#clean unnessary values from Delivery time taken column
orders_df['Delivery_Time_Taken_mins'] = orders_df['Delivery_Time_Taken_mins'].str.extract('(\d+)').astype(int)
```

Figure 3.2.3-1Delivery time taken

From the dataset, in Delivery Time column found some negative values when creating histogram for Delivery time frequency. Extracting and converting the numeric values makes the data usable for calculations, statistics, or visualizations. This step is critical to prepare the data for meaningful insights and decision making.

7.6 Trigger used

```
CREATE DEFINER=`root`@`localhost` TRIGGER `prevent_duplicate_cards` BEFORE INSERT ON `cards` FOR EACH ROW BEGIN
    IF EXISTS (SELECT 1 FROM cards WHERE CardProvider = NEW.CardProvider AND DebitCard = NEW.DebitCard AND
    CreditCard = NEW.CreditCard) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Duplicate card data is not allowed.';
    END IF;
END
```

Figure 3.2.3-1Trigger used

When inserting data into the database from that database sometimes I can be duplicate. Using a trigger can stop data repetition instead of drop a table. Only implemented this trigger to cards table, get to know how that works. When inserting data insert function also is there any available existing data.

```
# Insert data into the 'cards' table while ensuring data uniqueness
def insert_cards_data(connection, orders_df):
    cursor = connection.cursor()
    for row in orders_df[['Card_Provider', 'Debit_Card', 'Credit_Card']].itertuples(index=False, name=None):
        # Check if the data already exists in the 'cards' table
        cursor.execute(
            "SELECT 1 FROM cards WHERE CardProvider = %s AND DebitCard = %s AND CreditCard = %s",
            (row[0], row[1], row[2])
        )

        # Fetch the result of the SELECT query
        existing_data = cursor.fetchone()

        # If the data doesn't exist, insert it
        if not existing_data:
            cursor.execute(
                "INSERT INTO cards(CardProvider, DebitCard, CreditCard) VALUES (%s, %s, %s)",
                (row[0], row[1], row[2])
            )

    connection.commit()
    cursor.close()
```

Figure 3.2.3-2Trigger used

8 The python code to create the database

```python
### Create database connection
def create_db_connection():
    connection = None
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            passwd='catrat123',
            database='restaurant_delivery'
        )
        print("MySQL Database connection successful")
    except mysql.connector.Error as err:
        print(f"Error: '{err}'")
    return connection

db_connection = create_db_connection()
```

Figure 3.2.3-1Python code for create database

This figure contains the connection to the resetaurant_delivery database that we created using MySQL workbench. The rest of python code with data cleaning, data insertion, Manager Update, mean of the customer rating for food and deliver duration with bar chart.

9 The GUI to update the restaurant manager in the database.

9.1 GUI successful added data with prompt

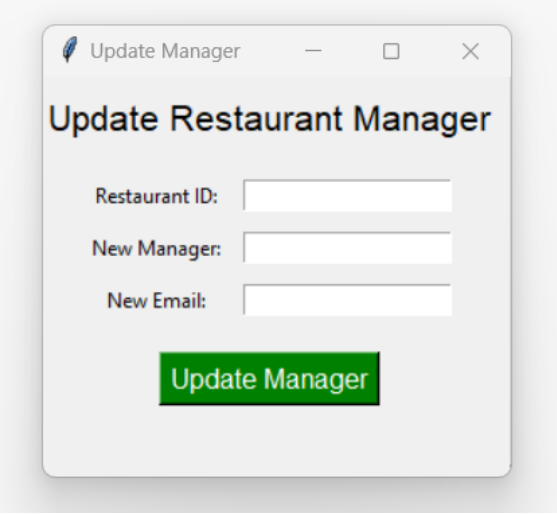

Figure 3.2.3-1Successfully added data with prompt

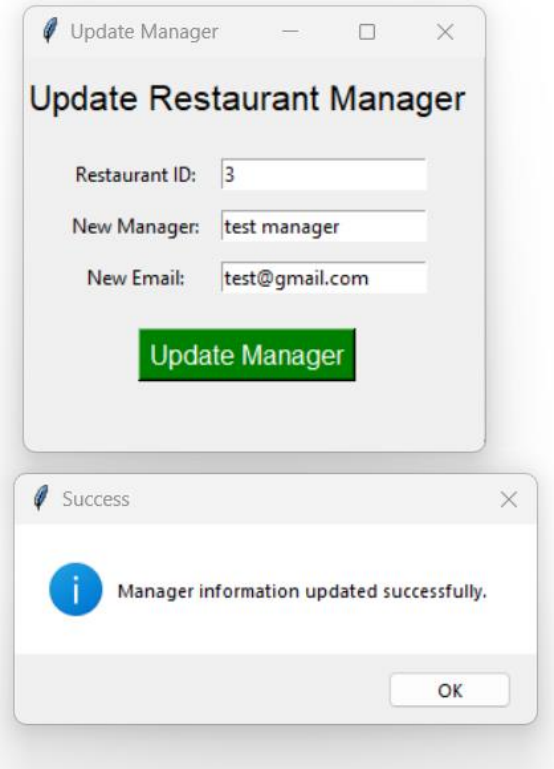

Figure 3.2.3-2Successfully added data with prompt

9.2 Invalid data insertion to the database

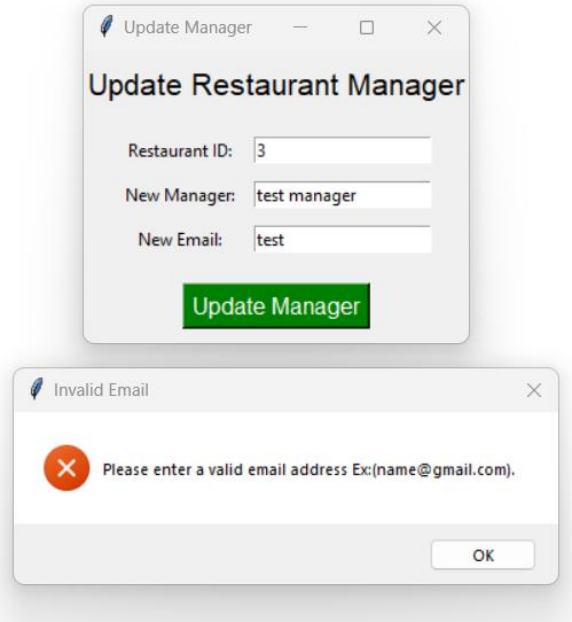

Figure 3.2.3-1Invalid data insertion to the database

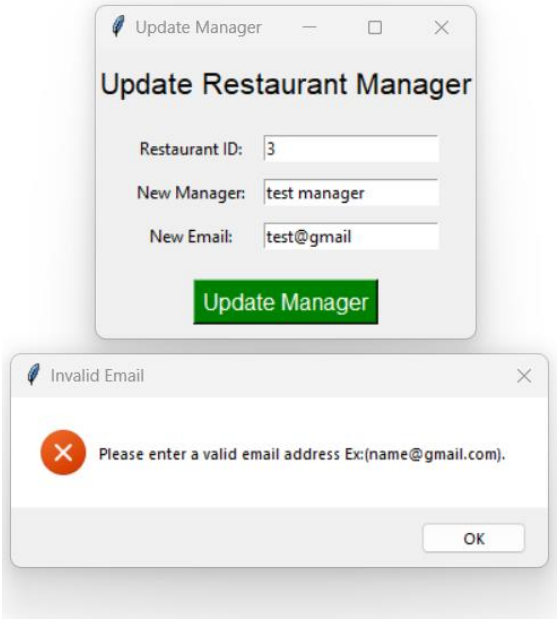

Figure 3.2.3-2Invalid data insertion to the database

10 Calculate mean customer rating food with GUI

10.1 Main UI

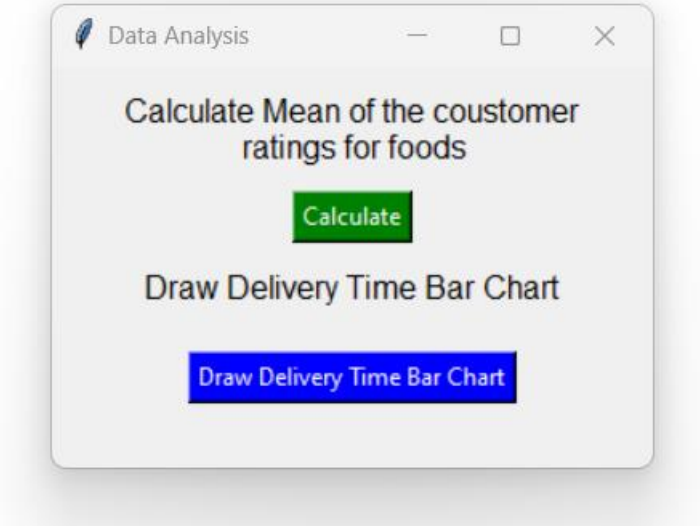

Figure 3.2.3-1Main UI

10.2 Mean values for food rating

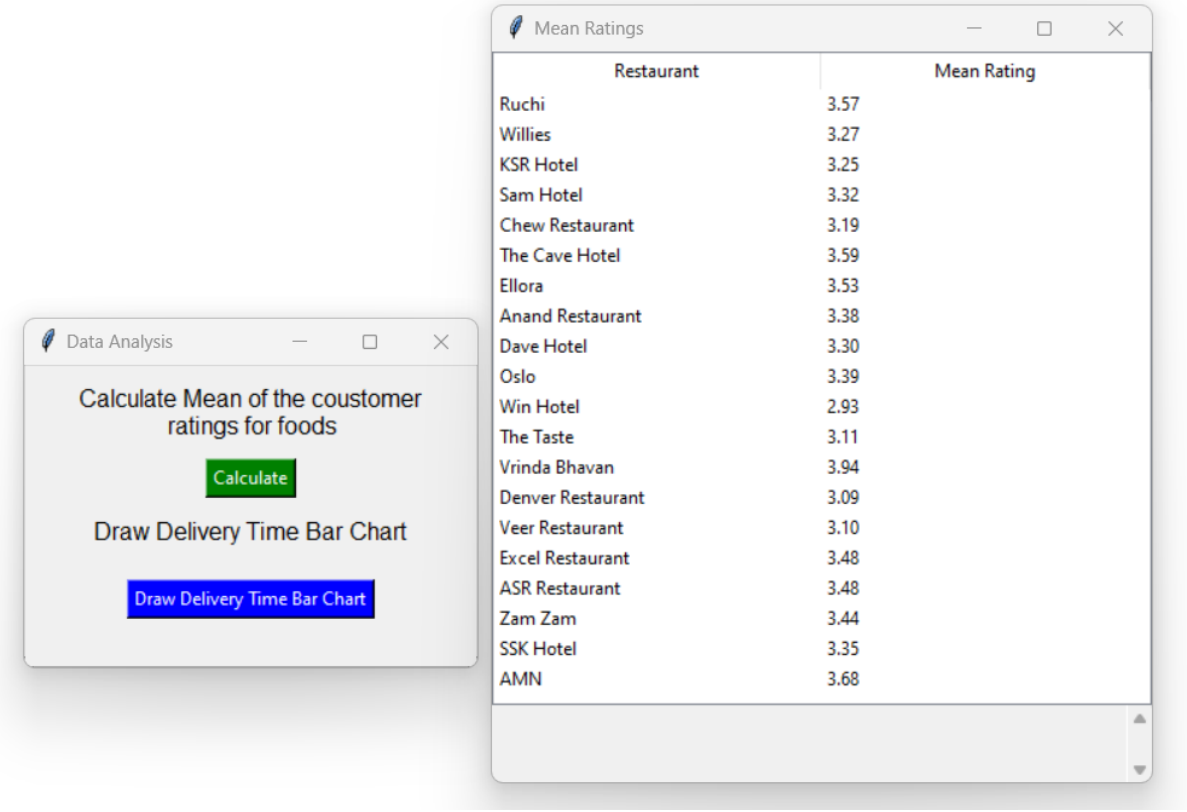

Figure 3.2.3-1Mean values for food rating

11 Analysis of the GUI with histogram of the mean delivery times

Figure 3.2.3-1Analysis of the GUI with histogram of the mean delivery times

The bar plot shows a sample distribution of delivery times. It looks like a histogram, with the x-axis representing delivery time taken 2 minutes apart from 5 minutes to 50 minutes. The y-axis calculates the number of deliveries.

From a cursory analysis, we observe a somewhat uniform distribution with a slight right skew, indicating that more deliveries take longer than the mode delivery time. The most frequent delivery time occurs just before the 50-minute mark.

11.1 Python scripts

11.1.1 Data Cleaning and Insertion

- create_db_connection()

  Purpose: Establishes a connection to a MySQL database.

  Parameters: None. It uses hardcoded credentials within the function to connect to the database.

- clean_data(df)

  Purpose: Fills any missing values in a pandas DataFrame with 'n/a'.

  Parameters:df: A pandas DataFrame to be cleaned.

- is_valid_email(email)

  Purpose: Validates the format of an email address using a regular expression.

  Parameters;email: A string representing the email address to validate.

- clean_email(email)

  Purpose: Cleans the 'Email' column in a DataFrame by checking if each email is valid and returning either the email or 'Invalid Email'.

  Parameters;email: A string representing the email address to clean.

- insert_restaurant_data(connection, df)

  Purpose: Inserts data into the 'restaurants' table of the database, avoiding duplicates.

  Parameters; connection: The database connection object.

  df: The pandas DataFrame containing restaurant data.

- insert_orders_data(connection, df)

Purpose: Inserts data into the 'orders' table of the database, handling duplicates with an update.

Parameters:connection: The database connection object.

df: The pandas DataFrame containing orders data.

- insert_delivery_data(connection, df)

  Purpose: Inserts data into the 'delivery' table of the database, ensuring the uniqueness of the data.

  Parameters:connection: The database connection object.

  df: The pandas DataFrame containing delivery data.

- insert_payments_data(connection, df)

  Purpose: Inserts data into the 'payments' table of the database, ensuring the uniqueness of the data.

  Parameters:connection: The database connection object.

  df: The pandas DataFrame containing payments data.

- insert_cards_data(connection, orders_df)

  Purpose: Inserts data into the 'cards' table of the database, ensuring the uniqueness of the data.

  Parameters:connection: The database connection object.

  orders_df: The pandas DataFrame containing card data.

- insert_manager_data(connection, df)

  Purpose: Inserts data into the 'manager' table of the database, ensuring the uniqueness of the data.

  Parameters:connection: The database connection object.

df: The pandas DataFrame containing manager data.

11.1.2 New Manager Input

Functions

- create_db_connection()

  Establishes a connection to the MySQL database.

  Returns:connection: A MySQL connection object if the connection is successful, None otherwise.

- update_manager()

  Updates the manager's name and email in the database for a given restaurant ID. The data is collected from the GUI input fields. It also validates the email format before attempting to update the database. If the email is not valid, an error message is displayed.

  Parameters: None. This function retrieves input directly from the Tkinter entry widgets.

- is_valid_email(email)

  Checks if the provided email address matches a valid email format.

  Parameters:email (str): The email address to validate.

  Returns:bool: True if the email is valid, False otherwise.

GUI Components

The GUI for updating the manager information consists of the following components:

- Title Label: Displays the title of the window.
- Input Frame: A frame that contains the input fields and labels.
- Restaurant ID Label and Entry: For inputting the restaurant's ID.
- New Manager Label and Entry: For inputting the new manager's name.
- New Email Label and Entry: For inputting the new manager's email.
- Update Button: A button that triggers the update process when clicked.

11.1.3 Delivery time analysis

Dependencies

- mysql.connector: Connects to the MySQL database to execute queries.
- tkinter: Creates the GUI.
- tkinter.ttk: Provides access to the themed Tk widget set.
- tkinter.messagebox: Displays message boxes for alerts and errors.
- matplotlib.pyplot: Generates plots for data visualization.
- matplotlib.backends.backend_tkagg.FigureCanvasTkAgg: Embeds matplotlib figures in the Tkinter canvas.
- tkinter.Toplevel: Creates additional windows in the Tkinter application.

Functions

- create_db_connection()

  Establishes a connection to the MySQL database with predefined credentials.

  Returns:

  connection: A MySQL connection object if successful; None if there is an error.

- display_mean_ratings()

  Fetches and displays the mean customer ratings for food from the database, grouped by restaurants, in a tabular format using a Tkinter Toplevel window.

  Parameters: None.

- fetch_delivery_data()

  Retrieves delivery time data from the 'delivery' table in the database.

  Returns:

delivery_times: A list of delivery times if successful; an empty list if there is an error or no data.

- create_delivery_time_bar_plot()

  Generates and displays a bar plot of delivery time intervals using matplotlib, embedded in a Tkinter Toplevel window.

  Parameters: None.

GUI Components

- The main analysis window (**analysis_window**) contains the following components:
- **ratings_labe**l: A label widget that provides instructions for calculating mean customer ratings for food.
- **ratings_button**: A button widget that, when clicked, calls display_mean_ratings() to calculate and display the ratings.
- **bar_chart_label**: A label widget that provides instructions for drawing the delivery time bar chart.
- **bar_chart_button**: A button widget that, when clicked, calls **create_delivery_time_bar_plot()** to generate and display the delivery time bar chart.

Usage

To use this module:

- Ensure that the MySQL database is running and accessible with the provided credentials.
- Run the script to open the main analysis window.
- Use the "Calculate" button to display the mean ratings in a new window.
- Use the "Draw Delivery Time Bar Chart" button to display the histogram of delivery times in another new window.
- Before exiting the application, the database connection is closed to prevent resource leaks.

11.2 Updated database design with driver details

Data insertion with

```
CREATE TABLE `delivery` (
  `DeliveryID` int NOT NULL AUTO_INCREMENT,
  `OrderID` varchar(255) DEFAULT NULL,
  `DeliveryTimeTaken` int DEFAULT NULL,
  `Customer_Rating_Food` int DEFAULT NULL,
  `Customer_Rating_Delivery` int DEFAULT NULL,
  `Delivery_Staff_Name` varchar(255) DEFAULT NOT NULL,
  `Delivery_Vehicle` enum('car','bike','motorbike') DEFAULT NOT NULL,
  PRIMARY KEY (`DeliveryID`),
  KEY `orderid3_idx` (`OrderID`),
  CONSTRAINT `orderid3` FOREIGN KEY (`OrderID`) REFERENCES `orders` (`OrderID`)
)
```

11.3 Tables from MySQL database

11.3.1 Restaurant table

RestaurantID	RestaurantName	Cuisine	Zone	Category	Address
1	The Cave Hotel	Continental	Zone B	Pro	4971 Janet Court;Livermore;CA;94550
2	SSK Hotel	North Indian	Zone D	Pro	29 Vista Flores;Pleasanton;CA;94566
3	ASR Restaurant	South Indian	Zone D	Ordinary	1383 Purdue Street;San Leandro;CA;94579
4	Win Hotel	South Indian	Zone D	Ordinary	1452 55th Avenue;Oakland;CA;94621
5	Denver Restaurant	Continental	Zone D	Pro	1352 Purdue Street;San Leandro;CA;94579
6	Willies	French	Zone D	Pro	43626 Bryant Street;Fremont;CA;94539
7	AMN	North Indian	Zone D	Ordinary	7096 Park Mesa Way;San Diego;CA;92111
8	Oslo	French	Zone B	Ordinary	12283 Thomas Lane;Yucaipa;CA;92399
9	Excel Restaurant	North Indian	Zone D	Ordinary	5472 Sunstar Common;Fremont;CA;94555
10	Dave Hotel	South Indian	Zone A	Ordinary	43626 Bryant Street;Fremont;CA;94539
11	The Taste	French	Zone B	Pro	32512 Christine Drive;Union City;CA;94587
12	Ruchi	Chinese	Zone B	Ordinary	5264 Shafter Avenue;Oakland;CA;94618
13	Veer Restaurant	Chinese	Zone D	Ordinary	2014 Clemens Road;Oakland;CA;94602
14	KSR Hotel	Chinese	Zone A	Pro	1452 55th Avenue;Oakland;CA;94621
15	Vrinda Bhavan	North Indian	Zone D	Ordinary	2732 Humboldt Avenue;Oakland;CA;94602
16	Anand Restaurant	African	Zone C	Ordinary	32532 Jean Drive;Union City;CA;94587
17	Zam Zam	Arabian	Zone C	Ordinary	6925 Lariat Lane;Castro Valley;CA;94552
18	Ellora	African	Zone C	Pro	3228 Chettenham Drive;Rancho Cordova;...
19	Sam Hotel	Belgian	Zone A	Ordinary	145 Grau Drive;Fremont;
20	Chew Restaurant	Belgian	Zone B	Ordinary	7573 National Drive;Livermore;CA;94550

Figure 11.1.3-1Restaurant table

11.3.2 Payment table

	PaymentID	OrderID	Payment_Mode
▶	1	OD1	Debit Card
	2	OD2	Credit Card
	3	OD3	Cash on Delivery
	4	OD4	Cash on Delivery
	5	OD5	Debit Card
	6	OD6	Cash on Delivery
	7	OD7	Credit Card
	8	OD8	Credit Card
	9	OD9	Debit Card
	10	OD10	Cash on Delivery
	11	OD11	Cash on Delivery
	12	OD12	Debit Card
	13	OD13	Debit Card
	14	OD14	Debit Card
	15	OD15	Credit Card
	16	OD16	Cash on Delivery
	17	OD17	Cash on Delivery
	18	OD18	Credit Card
	19	OD19	Cash on Delivery
	20	OD20	Cash on Delivery
	21	OD21	Cash on Delivery
	22	OD22	Cash on Delivery

Figure 11.3.1-1Payment table

11.3.3 Order table

OrderID	RestaurantID	OrderDate	QuantityOfItems	OrderAmount	First_Customer_Name	Last_Customer_Name	Customer_Rating_Food
OD1	6	1/1/2022 23:15	5.0	633.00	Srini	Simon	5
OD10	12	1/1/2022 19:21	3.0	295.00	Rifa	Carney	2
OD100	6	1/1/2022 11:10	7.0	936.00	Srini	Simon	2
OD101	6	1/1/2022 23:58	5.0	341.00	Srini	Simon	2
OD102	14	1/1/2022 20:35	2.0	78.00	Farhan	Gillespie	5
OD103	19	1/1/2022 21:31	7.0	723.00	Srini	Simon	4
OD104	20	1/1/2022 21:39	5.0	207.00	Selva	Jones	2
OD105	1	1/1/2022 21:39	4.0	411.00	Vinny	Coleman	5
OD106	18	1/1/2022 14:20	1.0	133.00	Shifa	Anderson	4
OD107	6	1/1/2022 11:10	1.0	124.00	Suhaib	Henry	5
OD108	16	1/1/2022 12:19	5.0	830.00	Chinny	Ferguson	3
OD109	18	1/1/2022 21:31	6.0	886.00	Meera	Gates	5
OD11	10	1/1/2022 23:58	6.0	607.00	Dev	Armstrong	5
OD110	8	1/1/2022 12:00	5.0	374.00	Suhaib	Henry	5
OD111	20	1/1/2022 14:31	5.0	585.00	Veer	Martinez	2
OD112	14	1/1/2022 14:31	5.0	580.00	Revandh	Scott	3
OD113	4	1/1/2022 13:39	6.0	590.00	Meera	Gates	4
OD114	11	1/1/2022 11:17	7.0	884.00	Rifa	Carney	4
OD115	15	1/1/2022 15:22	1.0	41.00	Selva	Jones	5
OD116	4	1/1/2022 12:00	5.0	485.00	Srini	Simon	2
OD117	5	1/1/2022 14:20	3.0	269.00	Suhaib	Henry	3
OD118	13	1/1/2022 23:15	4.0	391.00	Fastin	Smith	2
OD119	9	1/1/2022 12:19	4.0	220.00	Meera	Gates	3

Figure 11.3.2-1Order table

11.3.4 manager table

ManagerID	RestaurantID	Manager	Years_as_manager	Email
1	1	Esther Hosea	2	estherhosea@example.org
2	2	Dolores Dome	15	doloresdome@example.net
3	3	Jacquline Segal	9	jacqulinesegal@example.org
4	4	Anne Mckinley	1	annemckinley@example.net
5	5	Francisco Doxey	14	franciscodoxey@example.net
6	6	Bonnie Somers	5	bonniesomers@example.net
7	7	Nicolle Arnold	1	nicollearnold@example.org
8	8	n/a	0	Invalid Email
9	9	Georgia Vandergriff	5	georgiavandergriff@example.net
10	10	Sabrina Riley	11	sabrinariley@example.com
11	11	Nancy Cox	8	nancycox@example.net
12	12	Norbert Stormo	8	norbertstormo@example.com
13	13	Augusta Hurley	15	augustahurley@example.net
14	14	Megan Guillot	14	meganguillot@example.org
15	15	Clara Owen	0	claraowen@example.org
16	16	Tanisha Schaefer	15	tanishaschaefer@example.com
17	17	Patrick Stacey	9	Invalid Email
18	18	Bianca Tirk	9	biancatirk@example.com
19	19	Wesley Whitley	9	wesleywhitley@example.org
20	20	Doreen Gibbs	15	doreengibbs@example.net

Figure 11.3.3-1Manager table

11.3.5 Delivery table

DeliveryID	OrderID	DeliveryTimeTaken	Customer_Rating_Food	Customer_Rating_Delivery	Delivery_Staff_Name	Delivery_Vehicle
1	OD1	47	5	3	NULL	NULL
2	OD2	41	3	5	NULL	NULL
3	OD3	30	3	4	NULL	NULL
4	OD4	30	3	4	NULL	NULL
5	OD5	18	4	3	NULL	NULL
6	OD6	21	5	2	NULL	NULL
7	OD7	41	4	3	NULL	NULL
8	OD8	35	2	1	NULL	NULL
9	OD9	27	3	4	NULL	NULL
10	OD10	49	2	1	NULL	NULL
11	OD11	35	5	5	NULL	NULL
12	OD12	21	2	1	NULL	NULL
13	OD13	44	5	1	NULL	NULL
14	OD14	11	4	4	NULL	NULL
15	OD15	44	5	2	NULL	NULL
16	OD16	28	5	2	NULL	NULL
17	OD17	27	3	5	NULL	NULL
18	OD18	39	5	1	NULL	NULL
19	OD19	24	2	1	NULL	NULL
20	OD20	24	5	5	NULL	NULL
21	OD21	35	3	3	NULL	NULL
22	OD22	37	3	3	NULL	NULL
23	OD23	29	3	2	NULL	NULL

Figure 11.3.4-1Delivery table

11.3.6 Card table

CardID	CardProvider	DebitCard	CreditCard	PaymentID
1	VISA 13 digit	35303029142255300		NULL
2	Discover		43322881833571418	NULL
3	n/a			NULL
4	Mastercard	6759040000000		NULL
5	Diners Club / Carte Blanche		22274750911728630	NULL
6	JCB 15 digit		1800730000000000	NULL
7	Maestro	45596204311000820		NULL
8	Maestro	49409300000000		NULL
9	JCB 16 digit	35708020937705200		NULL
10	JCB 16 digit	3489050000000000		NULL
11	VISA 19 digit		65870843548396740	NULL
12	VISA 13 digit		45214430787335770	NULL
13	Diners Club / Carte Blanche	40734500000000		NULL
14	American Express		60112137556930770	NULL
15	JCB 16 digit		364383000000000	NULL
16	VISA 13 digit		25970370125848810	NULL
17	Discover	3488120000000000		NULL
18	JCB 16 digit		35222587301110090	NULL
19	VISA 13 digit		48617038564673690	NULL
20	Discover	35923461691839720		NULL
21	JCB 16 digit		48608100000000	NULL
22	Mastercard		3406750000000000	NULL
23	American Express		3463040000000000	NULL

Figure 11.3.6-1Card table

12 Conclusion

Finally, the restaurant database system represents an important move toward better information management for local delivery service. From beginning to end We have faced various challenges started during the course of this project, adding detail to our learning process.

One of the biggest challenging parts we have faced, the data cleaning process. So that had to do with missing or incorrect values in the CSV files. Furthermore, moving from CSV files to a MySQL workbench required an in-depth knowledge of data types and structures.

Another set of challenges came up when on the making of the graphical user interface (GUI). Since Tkinter is a strong framework, applying the GUI with MySQL operations requires a more complex approach. handling a constant connection among input from users as well as updating the database, and also error handling, presented various difficulties. From the challenges that we faced gained a better idea about how the database works , basics of MySQL workbench and new python scripts.

But the challenges we had faced turned out to be valuable learning opportunities, to a better understanding of database design, data cleaning techniques, and GUI development.

Finally, our group overcome these challenges successfully to show its adaptability and problem solving abilities. The restaurant database system is proof of overcoming obstacles and laying the groundwork for effective handling of data in our local delivery service.