# SQL-Based Restaurant Database Management System

**Student ID: 23031641**

## Introduction:

A collection of interlinked data that facilitates the efficient retrieval, insertion, and deletion of information arranged into tables, schemas, reports, and other formats is known as a database. Database Management Systems, often known as DBMSs, are software programs that are developed with the purpose of managing and organising data in a systematic fashion. In addition to enabling users to build, change, and query databases, it also gives them the ability to manage the security and access restrictions for those databases. DBMS offers a setting that makes it possible to store and retrieve data in a manner that is both convenient and effective. DBMSs are an essential and pervasive element of contemporary computing, resulting from decades of study and development in both academic and industrial sectors.

In the present study, we have developed a restaurant orders DBMS which provides an efficient model and have the following salient features:

- To organise and store massive volumes of structured data, including details about customers, orders, menu items, reservations, and staff information. The database is constructed to contain **nominal data** (customer names, dish categories), **ordinal data** (order status or dish category preferences), **interval data** (reservation dates), and **ratio data** (number of orders, prices, total amounts).
- To ensure data integrity using constraints such as primary keys and foreign keys.
- To assist restaurant management with making smart decisions and generating data, including daily sales statistics, inventory monitoring, staff performance assessments, and identification of the most popular dishes.
- To facilitate immediate updates for the administration of orders, reservations, and customer inquiries. It facilitates transactions that guarantee the successful completion of all actions associated with a specific operation (such as placing an order) or ensures that none are carried out.
- To enable the DBMS more efficiently, the automation of order processing, inventory management, and reporting, reduces manual effort and saves time with more profitability.

## Data Generation Process and Randomization:

The different parameters have been used to define the capacity and functionality of the restaurant's database system. The detail of parameter values chosen in data base has been described in the following table.

*Table 1: Database Schema Design with Key Dataset Parameters and Values*

| Sr. No. | Parameters | Value |
|---|---|---|
| 1 | Number of Order Details | 1500 |
| 2 | Number of Orders | 600 |
| 3 | Number of Customers | 400 |
| 4 | Number of Menu Items | 50 |
| 5 | Number of employees | 25 |
| 6 | Number of Reservations | 150 |

To create a realistic restaurant base scenario, we have chosen 20 attributes and then split these columns into six different tables. These data frames represent entities like Customers, Employees, Menu Items, Orders, Reservations, and Order Details and each entity is saved as CSV file into pandas.  All the computation is performed in Python coding.

- **Customers Table:** Customers are those individual persons places orders or reservations. The customers first and last names were generated randomly from a predefined list of common names. This ensures variety and realistic look of the names. The combination of letters, and numbers have been used to create real-world postcode. The values were chosen at random to reflect realistic data set.

Table 2: Data Frame of Customer Table

**Customers Table Data Frame**

| | customer_id | customer_name | post_code |
|---|---|---|---|
| **0** | 1 | Chris Walker | A202 |
| **1** | 2 | Sophia Young | NaN |
| **2** | 3 | Charlotte Green | C208 |
| **3** | 4 | Ava Baker | C306 |
| **4** | 5 | Alice Wilson | A202 |

```
In [6]: # Customers Table: Generate random customer data
# Nominal Data: 'customer_name'
customer_ids = np.arange(1, n_customers + 1)  # Primary key for Customers Table

# Randomly generated customer names from predefined lists of first and last names
first_names = ['Alice', 'John', 'Emma', 'Chris', 'Lucy', 'David', 'Olivia', 'Tom', 'Sophia', 'James',
               'Michael', 'Sarah', 'Daniel', 'Mia', 'Benjamin', 'Charlotte', 'Liam', 'Amelia', 'Noah', 'Ava']

last_names = ['Smith', 'Brown', 'Taylor', 'Johnson', 'Evans', 'Wilson', 'Moore', 'Walker', 'Roberts', 'Hall',
              'Harris', 'Lewis', 'Clark', 'Robinson', 'King', 'Green', 'Baker', 'Adams', 'Young', 'Scott']
customer_names = [
    f"{np.random.choice(first_names)} {np.random.choice(last_names)}"  # Randomized names to simulate realistic diversity
    for _ in range(n_customers)
]
postcode_data = [
    f"{letter}{num}{str(i).zfill(2)}" for letter in ['A', 'B', 'C']for num in range(1, 6) for i in range(1, 10)]
postcodes = np.random.choice(postcode_data, n_customers, replace=True)

# Introducing missing values (10%): Represents cases where customers didn't provide their postcode
postcodes = np.where(np.random.rand(n_customers) < 0.1, None, postcodes)
# Deliberate duplicates for non-primary key fields (5% duplicates)
# Simulates realistic scenarios where customers might share the same address or postcodes are incorrectly recorded
duplicate_indices = np.random.choice(np.arange(n_customers), size=int(0.05 * n_customers), replace=False)
postcodes[duplicate_indices] = postcodes[np.random.choice(np.arange(n_customers), size=len(duplicate_indices))]

# Create Data Frame
customers_df = pd.DataFrame({
    'customer_id': customer_ids,  # Primary Key for Customers
    'customer_name': customer_names,
    'post_code': postcodes  # Includes missing and duplicate values for realism
})
customers_df.set_index('customer_id', inplace=True)
customers_df.to_csv('customers.csv')
display(customers_df.head())  # Display the top rows to visualize data
```

*Figure 1: Python Code to Create Customer's Table CSV file with Randomization*

- **Order Table**: The attribute namely order_date has been generated randomly from 1st January 2023 to 31 December 2023 and we have used an exponential distribution to present the data in realistic way. Most of orders were clustered within the first few months, but with some distributed across the year. The order_status columns were given values from categories such as 'Pending', 'Completed', and 'Cancelled'. To provide a realistic range of values that match typical restaurant order sizes, the total_amount attribute was produced using a normal distribution and has a mean of $50 and a standard deviation of $20.

*Table 3: Data Frame of Orders Table*

**Orders Table Data Frame**

| | order_id | customer_id | order_date | total_amount | order_status | employee_id |
|---|---|---|---|---|---|---|
| **0** | 1 | 162 | 2023-01-11 | 40.44 | Completed | 3 |
| **1** | 2 | 242 | 2023-01-27 | 50.42 | Completed | 21 |
| **2** | 3 | 169 | 2023-06-14 | 75.12 | Completed | 15 |
| **3** | 4 | 136 | 2023-01-01 | NaN | Cancelled | 23 |
| **4** | 5 | 321 | 2023-04-22 | 93.25 | Completed | 3 |

```
In [7]: # Orders Table: Generate random order data
        # Interval Data: 'order_date'
        # Ordinal Data: 'order_status'

        order_ids = np.arange(1, n_orders + 1)  # Primary key for Orders Table

        order_dates = [
            datetime(2023, 1, 1) + timedelta(days=int(np.random.exponential(scale=90)))
            for _ in range(n_orders)
        ]  # Random interval dates to simulate realistic order distribution

        total_amounts = np.random.normal(loc=50, scale=20, size=n_orders).clip(10, 200).round(2)
        total_amounts = np.where(np.random.rand(n_orders) < 0.05, None, total_amounts)  # Randomly introduce missing values for

        order_statuses = np.random.choice(
            ['Pending', 'Completed', 'Cancelled'], n_orders, p=[0.1, 0.85, 0.05]
        )  # Ordinal data representing the state of each order

        total_amounts = np.where(order_statuses == 'Cancelled', None, total_amounts)  # No total amount for cancelled orders

        # Deliberate duplicate amounts (5%): Represents multiple orders with identical order totals
        duplicate_indices = np.random.choice(np.arange(n_orders), size=int(0.05 * n_orders), replace=False)
        total_amounts[duplicate_indices] = total_amounts[np.random.choice(np.arange(n_orders), size=len(duplicate_indices))]

        orders_df = pd.DataFrame({
            'order_id': order_ids,  # Primary Key
            'customer_id': np.random.choice(customer_ids, n_orders),  # Foreign Key: Links to Customers Table
            'order_date': order_dates,  # Interval data
            'total_amount': total_amounts,
            'order_status': order_statuses  # Ordinal data
        })

        orders_df.set_index('order_id', inplace=True)
        orders_df.to_csv('orders.csv')
        display(orders_df.head())  # Display the top rows to visualize data
```

*Figure 2: Python Code to Create Order's Table CSV file with andomization*

- **Menu Items Table:** Menu items identify the dishes, their categories and prices offered by the restaurant. There are three categories of menu items: 'Appetizer', 'Main Course', and 'Dessert'. Using normal distribution to present realistic price values ranging between $5 and $30.

*Table 4: Data frame of Menu Item Table*

**Menu Items Table Data Frame**

| | menu_item_id | dish_category | price |
|---|---|---|---|
| 0 | 1 | Main Course | 10.33 |
| 1 | 2 | Dessert | 11.00 |
| 2 | 3 | Appetizer | 23.13 |
| 3 | 4 | Appetizer | 6.43 |
| 4 | 5 | Main Course | 17.83 |

```
# Menu Items Table: Generate random menu item data
# Ordinal Data: 'dish_category'

menu_item_ids = np.arange(1, n_menu_items + 1)  # Primary Key for MenuItems Table

dish_categories = np.random.choice(
    ['Appetizer', 'Main Course', 'Dessert'], n_menu_items, p=[0.3, 0.5, 0.2]
)  # Randomized dish categories (ordinal data)

prices = np.random.lognormal(mean=2.5, sigma=0.5, size=n_menu_items).clip(5, 30).round(2)

# Deliberate duplicates for non-primary key fields (5% duplicates)
# Represents similar priced menu items (e.g., different dishes priced the same)
duplicate_indices = np.random.choice(np.arange(n_menu_items), size=int(0.05 * n_menu_items), replace=False)
prices[duplicate_indices] = prices[np.random.choice(np.arange(n_menu_items), size=len(duplicate_indices))]

menu_items_df = pd.DataFrame({
    'menu_item_id': menu_item_ids,  # Primary Key for Menu Items
    'dish_category': dish_categories,
    'price': prices
})

menu_items_df.set_index('menu_item_id', inplace=True)
menu_items_df.to_csv('menu_items.csv')
display(menu_items_df.head())  # Display the top rows to visualize data
```

*Figure 3: Python Code to Create Menu Item's Table CSV file with Randomization*

- **Order Details Table:** The other hand, offer itemised information on the restaurant products that are included in each order. Each order_detail was produced by randomly associating order_ids with menu_item_ids. This table manages many-to-many connections between orders and dishes. using a Poisson distribution with mean of 3, the quantity attribute has been generated. The attribute namely sub_total has been calculated by multiplying the quantity with the menu item price.

*Table 5: Data Frame of Order Details Table*

**Orders Detail Data Frame**

| | order_id | menu_item_id | order_detail_id | quantity | sub_total |
|---|---|---|---|---|---|
| **0** | 521 | 33 | 1 | 3 | 29.16 |
| **1** | 559 | 48 | 2 | 5 | 110.75 |
| **2** | 27 | 34 | 3 | 5 | 31.25 |
| **3** | 216 | 28 | 4 | 2 | 19.44 |
| **4** | 216 | 1 | 5 | 4 | 20.00 |

```python
# Order Details Table: Generate random details for each order
# Ratio Data: 'quantity'

order_detail_ids = np.arange(1, n_order_details + 1)  # Primary Key for Order Details Table

menu_item_data = np.random.choice(menu_item_ids, n_order_details)  # Foreign Key: Links to Menu Items Table

quantities = np.random.poisson(lam=3, size=n_order_details).clip(1, 5)  # Random quantities (ratio data)
sub_totals = (quantities * np.random.choice(prices, n_order_details)).round(2)
sub_totals = np.where(np.random.rand(n_order_details) < 0.02, None, sub_totals)  # Introducing missing values

# Deliberate duplicates for non-primary key fields (5% duplicates): Represents repeated quantity values in similar orders
duplicate_indices = np.random.choice(np.arange(n_order_details), size=int(0.05 * n_order_details), replace=False)
quantities[duplicate_indices] = quantities[np.random.choice(np.arange(n_order_details), size=len(duplicate_indices))]

order_details_df = pd.DataFrame({
    'order_detail_id': order_detail_ids,  # Primary Key for Order Details
    'order_id': np.random.choice(order_ids, n_order_details),  # Foreign Key: Links to Orders Table
    'menu_item_id': menu_item_data,  # Foreign Key: Links to Menu Items Table
    'quantity': quantities,  # Ratio data (must be greater than 0)
    'sub_total': sub_totals  # Calculated field for order subtotal
})

order_details_df.set_index(['order_id' , 'menu_item_id'], inplace=True)
order_details_df.to_csv('order_details.csv')
display(order_details_df.head())  # Display the top rows to visualize data
```

*Figure 4: Python Code to Create Order details' Table CSV file with Randomization*

- **Employees Table:** Employees are the staff members that are responsible for managing orders, reservations, and contacts with customers. There are three categories of staff like 'Manager', 'Chef', or 'FOH (Front of House)', reflecting typical restaurant staff structure. The 'Morning', 'Evening', or 'Night' shifts has been assigned randomly to present the realistic data.

*Table 6: Data from of Employee Table*

**Employees Table Data Frame**

| | employee_id | position | shift |
|---|---|---|---|
| **0** | 1 | FOH | Morning |
| **1** | 2 | Chef | Evening |
| **2** | 3 | FOH | Evening |
| **3** | 4 | FOH | Morning |
| **4** | 5 | Manager | Night |

```python
# Employees Table: Generate random employee data

employee_ids = np.arange(1, n_employees + 1)  # Primary Key for Employees Table
# Employee roles and shifts (nominal data)
roles = np.random.choice(['Manager', 'Chef', 'FOH'], n_employees, p=[0.1, 0.4, 0.5])
shifts = np.random.choice(['Morning', 'Evening', 'Night'], n_employees, p=[0.4, 0.4, 0.2])

# Deliberate duplicates for non-primary key fields (5% duplicates):
duplicate_indices = np.random.choice(np.arange(n_employees), size=int(0.05 * n_employees),
                                     replace=False)
shifts[duplicate_indices] = shifts[np.random.choice(np.arange(n_employees),
                                                     size=len(duplicate_indices))]
# Create Data Frame
employees_df = pd.DataFrame({
    'employee_id': employee_ids,  # Primary Key for Employees Table
    'position': roles,
    'shift': shifts
})
employees_df.set_index('employee_id', inplace=True)
employees_df.to_csv('employees.csv')
display(employees_df.head())  # Display the top rows to visualize data
```

*Figure 5: Python Code to Create Employees' Table CSV file with Randomization*

- **Reservations Table:** The reservation_date attribute was created by randomly selecting dates from 2023, while the table_number attribute was formulated as a string that combines a letter prefix with a number, reflecting standard restaurant table labelling conventions. The Poisson distribution, with a mean of 2, is utilised to model typical restaurant reservations.

*Table 7: Data Frame of Reservation table*

**Reservations Table Data Frame**

| | reservation_id | customer_id | table_number | reservation_date | number_of_guests |
|---|---|---|---|---|---|
| **0** | 1 | 274 | G-18 | 2023-03-31 | 1.0 |
| **1** | 2 | 262 | G-4 | 2023-04-10 | 2.0 |
| **2** | 3 | 78 | G-3 | 2023-10-08 | 4.0 |
| **3** | 4 | 224 | G-3 | 2023-12-16 | 3.0 |
| **4** | 5 | 221 | G-4 | 2023-04-14 | 3.0 |

```python
# Reservations Table: Generate random reservation data
reservation_ids = np.arange(1, n_reservations + 1)  # Primary Key for Reservations Table
reservation_dates = [
    datetime(2023, 1, 1) + timedelta(days=int(np.random.uniform(0, 365)))
    for _ in range(n_reservations)
]  # Random reservation dates throughout the year

table_numbers = [f"G-{i}" for i in np.random.randint(1, 21, n_reservations)]  # Table number format (nominal data)
guest_counts = np.random.poisson(lam=2, size=n_reservations).clip(1, 10)  # Random number of guests (ratio data)
guest_counts = np.where(np.random.rand(n_reservations) < 0.05, None, guest_counts)  # Introducing missing values

# Deliberate duplicates for non-primary key fields (5% duplicates):
duplicate_indices = np.random.choice(np.arange(n_reservations), size=int(0.05 * n_reservations), replace=False)
guest_counts[duplicate_indices] = guest_counts[np.random.choice(np.arange(n_reservations), size=len(duplicate_indices))]
reservations_df = pd.DataFrame({
    'reservation_id': reservation_ids,  # Primary Key for Reservations Table
    'customer_id': np.random.choice(customer_ids, n_reservations),  # Foreign Key: Links to Customers Table
    'reservation_date': reservation_dates,  # Interval data for dates
    'table_number': table_numbers,
    'number_of_guests': guest_counts  # Ratio data (must be positive)
})
reservations_df.set_index('reservation_id', inplace=True)
reservations_df.to_csv('reservations.csv')
display(reservations_df.head())  # Display the top rows to visualize data
```

*Figure 6: Python Code to Create Reservation's Table CSV file with Randomization*

## Missing and Duplicate Values:

The null and repeated values in DBMS reflect the real-world situations. We have intentionally introduced these types of values to authenticate the integrity of DBMS. Around 5 to 10 percent of values are taken as null or repeated values.

If the value in the database is not present or different from zero or an empty string, then it’s called a null value, whereas when the same value appears in various rows of the table, it's known as a repeated value.

In our tables, we can have the following missing and duplicate values to ensure the real-time data existence.

- **Customers Table:** The 10% values in **“post_code”** have been deliberately taken as missing and 5% values were chosen as repeated values.
- **Orders Table:** If there is any failure to complete the order then the attribute **“total_amount”** value has been chosen none and even if the order has not been assigned yet then the **“order_status”** and **“employee_id”** may be missing. Also, 5% values of the **“total_amount”** is repeated because some customers placed same orders.
- **Menu Items Table:** No null value is expected in the menu item while the **“price”** attribute may contain repeated rows because different items may have the same price, 5% repeated values have been taken in the data.
- **Order Details Table:** The **“subtotal”** estimate may be null if the subtotal cannot be calculated due to an issue in the processing of an order. We have chosen 5% of **“quantity”** entries can be duplicated because multiple orders contain the same quantity of the food from same menu.
- **Employees Table:** Since the employee data is fed very carefully, we assume that no attribute of the employee table will have a null value while many employees are working at the same position and their shifts could be repeated. Therefore, the **“position”** and employee **“shift”** has been repeated.
- **Reservations Table:** If the number of guests is not specified at the time of reservation or there is a mistake in writing, we can take null values in the attribute **“number_of_guests”**. A customer can make different reservations, and repeat tables are used for different reservations. Usually, different reservations are booked on the same date, so the **“customer_id“**, **“table_number”** and **“reservation_id”** can be duplicated.

## Justification for Separate Tables:

**Customer Table:** The Customers Table stores customer information separately, facilitating efficient access, updates, and analysis of customer details independent of order information. This facilitates the identification of repeat customers, enables marketing analysis, supports customer segmentation, and allows for personalised offers. The establishment of a distinct Customers table facilitates data normalisation, thereby reducing redundancy associated with customers placing multiple orders.

**Order Table:** To link customers to their purchases, menu items, and employees, orders are maintained individually. Order date, status, and total amount are stored in the Orders database. The system can track orders from start to finish or cancel by splitting them into their own table, which improves reporting and consistency.

**Menu Item Table:** The MenuItems table contains dish category and pricing. Menu items may be maintained between orders and updated easily with a separate table. It also makes menu specifics like price and food kinds easy to find, which helps manage the restaurant's offers.

**Order Details table:** The OrderDetails table links orders to menu items. Each item can appear in several orders, forming a many-to-many connection. It records every order line precisely for calculations like total cost per order.

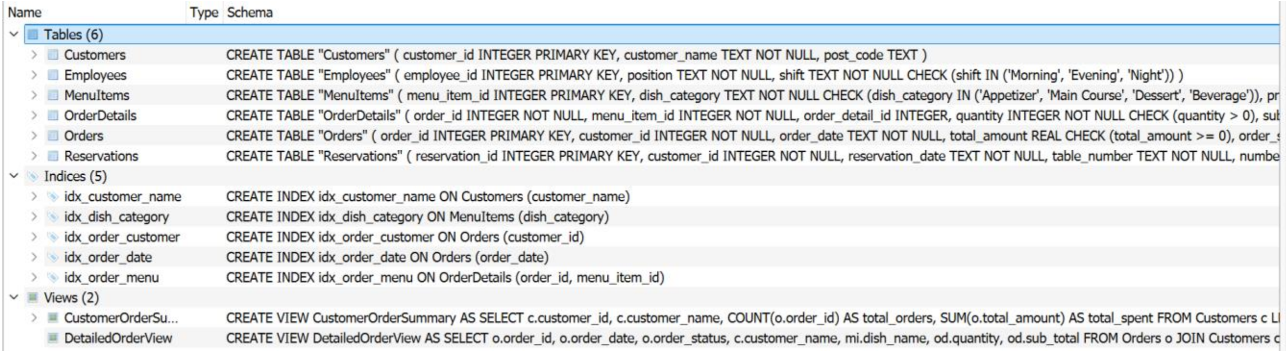
**Employees Table:** This table establishes a clear connection between employees and orders, allowing for the tracking and evaluation of employee performance. The Employees table contains details about the staff engaged in order processing, including their roles and shift information. Keeping this information separate enables the system to efficiently monitor which employee managed a particular order. It also enhances workforce management, including shift planning and task assignment.

**Reservation Table:** The Reservations table documents customer reservations, associating each reservation with a specific customer. This framework facilitates the administration of table reservations, guest numbers, and booking dates. The establishment of a distinct table for reservations facilitates the management and modification of reservation details, thereby minimising the impact on other components of the system.

## SQL Schema and Entity Relationship Diagram (ERD):

### SQL Schema Visualization:

The SQLite DB Browser has been used to import CSV files obtained from google colab for the creation of tables in database.



| Name | Type | Schema |
|---|---|---|
| Tables (6) | | |
| Customers | | CREATE TABLE "Customers" ( customer_id INTEGER PRIMARY KEY, customer_name TEXT NOT NULL, post_code TEXT ) |
| Employees | | CREATE TABLE "Employees" ( employee_id INTEGER PRIMARY KEY, position TEXT NOT NULL, shift TEXT NOT NULL CHECK (shift IN ('Morning', 'Evening', 'Night')) ) |
| MenuItems | | CREATE TABLE "MenuItems" ( menu_item_id INTEGER PRIMARY KEY, dish_category TEXT NOT NULL CHECK (dish_category IN ('Appetizer', 'Main Course', 'Dessert', 'Beverage')), pr |
| OrderDetails | | CREATE TABLE "OrderDetails" ( order_id INTEGER NOT NULL, menu_item_id INTEGER NOT NULL, order_detail_id INTEGER, quantity INTEGER NOT NULL CHECK (quantity > 0), sul |
| Orders | | CREATE TABLE "Orders" ( order_id INTEGER PRIMARY KEY, customer_id INTEGER NOT NULL, order_date TEXT NOT NULL, total_amount REAL CHECK (total_amount >= 0), order_s |
| Reservations | | CREATE TABLE "Reservations" ( reservation_id INTEGER PRIMARY KEY, customer_id INTEGER NOT NULL, reservation_date TEXT NOT NULL, table_number TEXT NOT NULL, numbe |
| Indices (5) | | |
| idx_customer_name | | CREATE INDEX idx_customer_name ON Customers (customer_name) |
| idx_dish_category | | CREATE INDEX idx_dish_category ON MenuItems (dish_category) |
| idx_order_customer | | CREATE INDEX idx_order_customer ON Orders (customer_id) |
| idx_order_date | | CREATE INDEX idx_order_date ON Orders (order_date) |
| idx_order_menu | | CREATE INDEX idx_order_menu ON OrderDetails (order_id, menu_item_id) |
| Views (2) | | |
| CustomerOrderSu... | | CREATE VIEW CustomerOrderSummary AS SELECT c.customer_id, c.customer_name, COUNT(o.order_id) AS total_orders, SUM(o.total_amount) AS total_spent FROM Customers c LI |
| DetailedOrderView | | CREATE VIEW DetailedOrderView AS SELECT o.order_id, o.order_date, o.order_status, c.customer_name, mi.dish_name, od.quantity, od.sub_total FROM Orders o JOIN Customers c |

*Figure 7: SQL Database Schema*

### Textual Schema:

We have described each table, its attributes, and data types as under:

- **Customers**: customer_id (Primary Key), customer_name (Nominal), post_code.
- **Orders**: order_id (Primary Key), customer_id (Foreign Key), order_date (Interval), order_status (Ordinal).
- **MenuItems**: menu_item_id (Primary Key), dish_category (Ordinal), price (Ratio).
- **Ord Details**: order_detail_id , order_id and menu_item_id (composite Key), quantity (Ratio).
- **Employees**: employee_id (Primary Key), position, shift.
- **Reservations**: reservation_id (Primary Key), customer_id (Foreign Key), reservation_date, table_number, number_of_guests (Ratio).

### Entity Relationship Diagram:

The ERD has been shown in figure 8 and the primary key (PK) in tables is given in red whereas the foreign key (FK) is taken in green. In order details table two foreign keys which are given in blue colour forms a composite key. The details relationship between the tables is given below:

**CUSTOMERS AND ORDERS:** This ERD illustrates a one-to-many relationship between the CUSTOMERS and ORDERS tables. The CUSTOMERS table employs customer_id as its Primary Key (PK), which is referenced as a Foreign Key (FK) in the ORDERS table to define the relationship. Customers are allowed to place multiple orders, thereby ensuring referential integrity and adherence to normalising standards.

**CUSTOMER AND RESERVATIONS:** The Customers table exhibits a one-to-many relationship with the Reservations table. This indicates that a single customer can make multiple reservations, with each reservation associated with one specific customer. This relationship advantages manage customer reservations professionally.

**EMPLOYEES AND ORDERS:** The Orders table is one-to-many related to the Employees table. A manager or server can process many orders at once, but they only deal with one order at a time. In order to manage and delegate responsibility for client orders, this connection is vital. This feature enables the system to keep tabs on which person is in responsible of each order, which enhances accountability, performance monitoring, and order fulfilment.

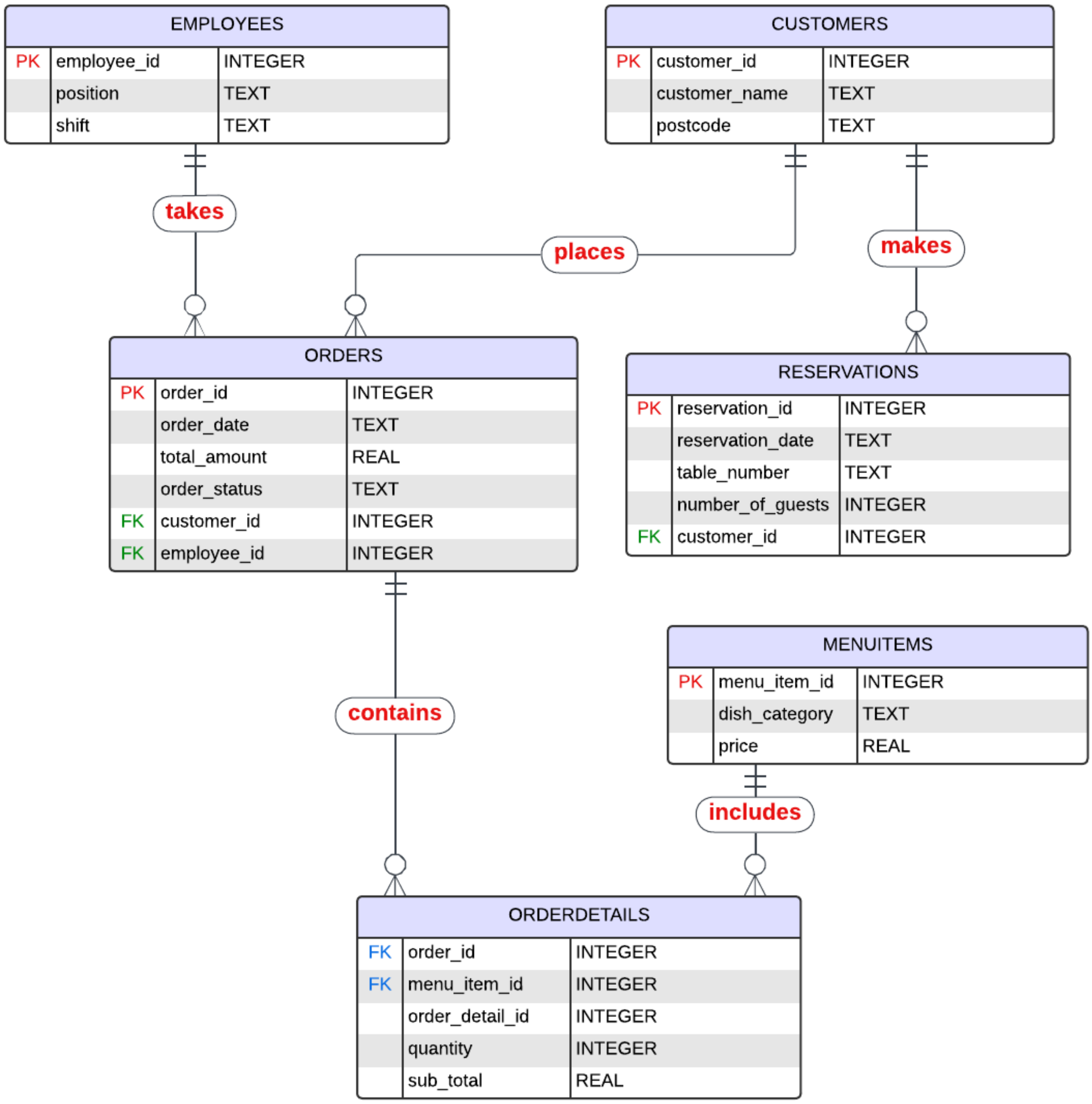



*Figure 8: Entity Relationship Diagram*

**MENUITEM, ORDERS AND ORDERDETAILS:** The OrderDetails table serves as the connection point between the MenuItems and Orders databases, so establishing a many-to-many database relationship. This indicates that a single order can include numerous items from the menu, and that a single menu item can be included in more than one order.

## Ethics and Data Privacy:

The sensitivity and scope of the data being collected and processed in the restaurant database setting make ethics and data privacy significant concerns. The following is a comprehensive discussion of these aspects:

### Sensitivity of Data:

Restaurant databases contain various types of information such as customer and employee personal contact numbers and their home addresses. Such information is very sensitive and can be misused by any worker or Information about the customer's habits and employee’s work schedule can be misused.

### Transparency in Data Collection:

Restaurants should clearly communicate the purpose of the data being collected to customers and employees so that to address their concerns and transparency increases trust. The data collected should only be used for quality improvement purposes. Giving or selling data to a third party is unethical.

### Access Control:

Access to sensitive data should be restricted to authorised individuals only. Customer contact information should be available just to workers tasked with bookings or customer support.

### Data Protection Laws:

The restaurant must observe data protection laws like General Data Protection Regulation. These rules require things including informing people how their data is being used, letting them ask for access to their data, and making sure that data is handled safely.

## Concluding Remarks:

This project aims to develop a realistic restaurant order management database that encompasses various data types commonly encountered in a restaurant environment. The project seeks to simulate the complexities of real-world restaurant operations through the integration of diverse data types, facilitating a comprehensive understanding of the interactions among various pieces of information within a relational database. This design incorporates foreign keys, composite keys, and realistic data constraints to establish a practical and interconnected schema that facilitates effective management and analysis of restaurant orders, reservations, and employee activities.