Design and Implementation of an Online Food Ordering System – Database Management System

1. Introduction

The traditional method of ordering food often involved phone calls, limited menus, and potential inaccuracies. In today's digital age, there is a growing demand for convenient and efficient online food ordering systems. Existing solutions might have limitations in terms of data scalability, security, or functionality.

This project proposes a solution through the design and implementation of a secure and scalable database management system for an online food ordering system. This Database Management System will serve as the backbone of the system, storing and managing all crucial data related to users, restaurants, menus, and orders. The system will leverage the power of SQL (Structured Query Language) to effectively manipulate and retrieve data, facilitating various functionalities within the online food ordering platform.

This project focuses solely on the database management system aspect of the online food ordering system. The scope encompasses:

- Designing a comprehensive data model to capture all essential entities and their relationships.
- Implementing data normalization techniques to ensure data integrity and minimize redundancy.
- Developing a secure database schema using SQL to define tables, data types, and constraints.
- Utilizing SQL for data manipulation (insert, update, delete) and retrieval (select, join).
- Exploring advanced SQL functionalities like window functions, CTEs, UDFs, and triggers for enhanced data processing.
- Implementing security measures like password hashing and access control. (if applicable)

The user interface, payment processing, and real-time functionalities are considered outside the scope of this project and might be addressed in future iterations.

This DBMS serves as a foundational element for the online food ordering system. The implemented data model and SQL functionalities provide a robust platform for efficient data management, paving the way for a user-friendly and feature-rich online food ordering experience.

2. System Requirements

Data Model:
The data model serves as a blueprint for the database, defining the entities involved and their relationships.

Here is a breakdown of the core entities:

1. **Users**: This table stores essential user information required for the online food ordering system.

   Attributes:
   - **user_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier for each user.
   - **name** (VARCHAR(255) NOT NULL): User's name.
   - **email** (VARCHAR(255) NOT NULL) UNIQUE: User's email address (unique for identification).
   - **password** (VARCHAR(255) NOT NULL): Hashed or encrypted password for security.
   - **phone_number** (VARCHAR(20)): User's phone number.
   - **user_type** (ENUM('customer', 'admin', 'delivery_partner')): Defines the user's role within the system (customer, administrator, or delivery partner).

2. **Restaurants**: This table stores information about restaurants registered within the system.

   Attributes:
   - **restaurant_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier for each restaurant.
   - **name** (VARCHAR(255) NOT NULL): Restaurant name.
   - **address** (TEXT): Restaurant address.
   - **cuisine_type** (VARCHAR(255)): Type of cuisine offered by the restaurant.
   - **minimum_order_amount** (DECIMAL(10,2)): Minimum order amount required for a restaurant.
   - **delivery_radius** (INT): Delivery area for the restaurant in kilometers.
   - **average_rating** (DECIMAL(2,1)): Average user rating for the restaurant.
   - **image** (VARCHAR(255)): URL or path to the restaurant image.

3. **Menu_items**: This table stores details about individual menu items offered by restaurants.

   Attributes:
   - **menu_item_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier for each menu item.
   - **restaurant_id** (INT NOT NULL) - Foreign Key referencing Restaurants(restaurant_id): Links menu items to their respective restaurants.
   - **name** (VARCHAR(255) NOT NULL): Menu item name.
   - **description** (TEXT): Description of the menu item.
   - **price** (DECIMAL(10,2)): Price of the menu item.

- **image** (VARCHAR(255)): URL or path to the menu item image.
- **category** (VARCHAR(255)): Category of the menu item (e.g., Appetizers, Main Course).

4. **Orders**: This table stores information about orders placed by users within the system.

   Attributes:
   - **order_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier for each order.
   - **user_id** (INT NOT NULL) - Foreign Key referencing Users(user_id): Links orders to the placing user.
   - **restaurant_id** (INT NOT NULL) - Foreign Key referencing Restaurants(restaurant_id): Links orders to the fulfilling restaurant.
   - **order_date** (DATETIME NOT NULL): Date and time the order was placed.
   - **total_amount** (DECIMAL(10,2)): Total amount of the order.
   - **payment_method** (ENUM('cash_on_delivery', 'online_payment')): Payment method used for the order.
   - **payment_status** (ENUM('pending', 'paid')): Tracks the payment status of the order.
   - **order_status** (ENUM('placed', 'preparing', 'out_for_delivery', 'delivered', 'cancelled')): Current status of the order.

5. **Delivery_Partners:** This table stores information about delivery partners registered within the system.

   Attributes:
   - **delivery_partner_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-incrementing integer) for each delivery partner.
   - **name** (VARCHAR(255) NOT NULL): Delivery partner's name.
   - **phone_number** (VARCHAR(20)): Delivery partner's phone number for communication purposes.
   - **vehicle_type** (VARCHAR(255)): Type of vehicle used for deliveries (e.g., Bike, Car).
   - **availability** (BOOLEAN): Indicates whether the delivery partner is currently available for new deliveries.
   - **average_rating** (DECIMAL(2,1)): Average user rating for the delivery partner based on reviews.

6. **Order_Tracking:** This table tracks the status updates for each order throughout the delivery process.

   Attributes:
   - **order_tracking_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-incrementing integer) for each order tracking entry.
   - **order_id** (INT NOT NULL) - Foreign Key referencing Orders(order_id): Links the tracking information to the specific order (foreign key relationship).

- **delivery_partner_id** (INT) - Foreign Key referencing Delivery_Partners(delivery_partner_id): Links the assigned delivery partner to the order (optional foreign key, can be null if not yet assigned).
- **status** (ENUM('placed', 'preparing', 'out_for_delivery', 'delivered')): Represents the status of the order at a specific point in time (same statuses as in Orders table).
- **timestamp** (DATETIME NOT NULL): Date and time when the specific order status update occurred.

7. **Reviews**: This table stores user reviews submitted for restaurants or delivery partners.

   Attributes:
   - **review_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-incrementing integer) for each review.
   - **user_id** (INT NOT NULL) - Foreign Key referencing Users(user_id): Links the review to the user who submitted it (foreign key relationship).
   - **order_id** (INT NOT NULL) - Foreign Key referencing Orders(order_id): Ensures reviews are submitted only for completed orders (foreign key relationship).
   - **review_type** (ENUM('restaurant', 'delivery_partner')): Specifies whether the review is for the restaurant that fulfilled the order or the delivery partner who delivered it.
   - **rating** (INT): Rating score for the review (e.g., 1-5 stars).
   - **comment** (TEXT): Optional text comment for the review, allowing users to elaborate on their ratings.

8. **Coupons**
   This table is included to manage discount coupons offered within the system.

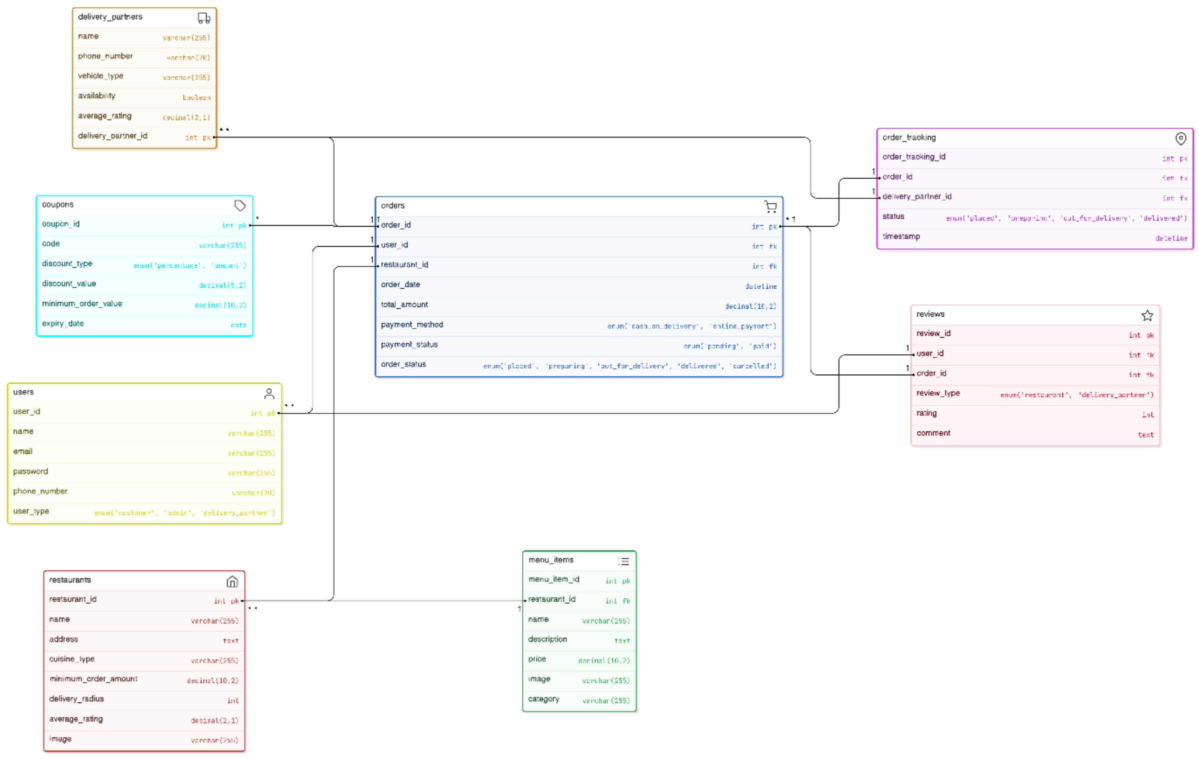
   Attributes:
   - **coupon_id** (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-incrementing integer) for each coupon.
   - **code** (VARCHAR(255) UNIQUE): Unique code used to identify and apply the coupon.
   - **discount_type** (ENUM('percentage', 'amount')): Type of discount offered by the coupon (percentage discount or fixed amount discount).
   - **discount_value** (DECIMAL(5,2)): Discount value based on the chosen type (percentage or amount).
   - **minimum_order_value** (DECIMAL(10,2)): Minimum order amount required for the coupon to be valid.
   - **expiry_date** (DATE): Expiry date of the coupon after which it becomes invalid.

Relationships Between Tables:
The tables within the data model are interconnected through foreign key relationships to maintain data consistency and integrity.

Here is a summary of the key relationships:

- Users can place multiple orders (one-to-many, Users - Orders).
- Restaurants can have many menu items (one-to-many, Restaurants - Menu_Items).
- An order belongs to a single user and a single restaurant (many-to-one, Users & Restaurants - Orders).
- A review can be related to a specific order and can be either a restaurant review or a delivery

Entity Relationship (ER) Diagram:



3. Database Design

This section delves into the core aspects of designing the database for the online food ordering system. It emphasizes data integrity, efficient storage, and retrieval of information.

Normalization:

Normalization is a crucial process that decomposes database tables into smaller, more focused tables to minimize redundancy and improve data integrity. The data model strives to achieve Second Normal Form (2NF), eliminating partial dependencies within tables.

Schema Definition:

Schema definition involves creating the structure of the database using SQL statements. Each statement defines a table, specifying its attributes (columns) and their data types:

```
-- -- Users table

CREATE TABLE Users (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash CHAR(64) NOT NULL, -- Secure storage using password
hashing
  phone_number VARCHAR(20),
  user_type ENUM('customer', 'admin', 'delivery_partner')
);


-- Restaurants table

CREATE TABLE Restaurants (
  restaurant_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  address TEXT NOT NULL,
  cuisine_type VARCHAR(255),
  minimum_order_amount DECIMAL(10,2),
  delivery_radius INT,  -- delivery area in kilometers
  average_rating DECIMAL(2,1),  -- average user rating
  image VARCHAR(255)  -- URL or path to restaurant image
);


-- Menu_Items table

CREATE TABLE Menu_Items (
  menu_item_id INT PRIMARY KEY AUTO_INCREMENT,
  restaurant_id INT NOT NULL,
  FOREIGN KEY (restaurant_id) REFERENCES Restaurants(restaurant_id),
  name VARCHAR(255) NOT NULL,
  description TEXT,
  price DECIMAL(10,2),
  image VARCHAR(255),  -- URL or path to menu item image
  category VARCHAR(255) -- Food category (e.g., Appetizers, Main Course)
);
```

```sql
-- Orders table

CREATE TABLE Orders (
  order_id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  restaurant_id INT NOT NULL,
  FOREIGN KEY (restaurant_id) REFERENCES Restaurants(restaurant_id),
  order_date DATETIME NOT NULL,
  total_amount DECIMAL(10,2),
  payment_method ENUM('cash_on_delivery', 'online_payment'),
  payment_status ENUM('pending', 'paid'),
  order_status ENUM('placed', 'preparing', 'out_for_delivery',
'delivered', 'cancelled')
);


-- Delivery_Partners table

CREATE TABLE Delivery_Partners (
  delivery_partner_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  phone_number VARCHAR(20),
  vehicle_type VARCHAR(255), -- Bike, Car etc.
  availability BOOLEAN, -- Indicates availability for deliveries
  average_rating DECIMAL(2,1)  -- average rating based on user reviews
);


-- Order_Tracking table

CREATE TABLE Order_Tracking (
  order_tracking_id INT PRIMARY KEY AUTO_INCREMENT,
  order_id INT NOT NULL,
  FOREIGN KEY (order_id) REFERENCES Orders(order_id),
  delivery_partner_id INT,  -- Foreign Key referencing
Delivery_Partners(delivery_partner_id) - Can be null if not assigned yet
  FOREIGN KEY (delivery_partner_id) REFERENCES
Delivery_Partners(delivery_partner_id),
  status ENUM('placed', 'preparing', 'out_for_delivery', 'delivered'),
  timestamp DATETIME NOT NULL
);


-- Reviews table

CREATE TABLE Reviews (
  review_id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  order_id INT NOT NULL,
  FOREIGN KEY (order_id) REFERENCES Orders(order_id),
  review_type ENUM('restaurant', 'delivery_partner'),
  rating INT, -- Rating (e.g., 1-5 stars)
  comment TEXT  -- Optional review text
);
```

```
-- Coupons table

CREATE TABLE Coupons (
  coupon_id INT PRIMARY KEY AUTO_INCREMENT,
  code VARCHAR(255) UNIQUE, -- Unique coupon code
  discount_type ENUM('percentage', 'amount'),
  discount_value DECIMAL(5,2), -- Discount percentage or amount
  minimum_order_value DECIMAL(10,2), -- Minimum order value for coupon
validity (optional)
  expiry_date DATE
);
```

Constraints:
Here is a concise breakdown of constraints for the online food ordering system data model:

- **Primary Keys:** Uniquely identify each record in every table (e.g., user_id in Users).
- **Foreign Keys:** Enforce relationships between tables (e.g., order_id in Orders references user_id in Users).
- **Unique Constraints:** Prevent duplicate entries for specific attributes (e.g., email in Users).
- **Check Constraints** (Optional): Validate data formats (e.g., phone number format) or restrict values (e.g., positive prices).
- **Additional Considerations:**
    - Choose appropriate data types (DECIMAL for money, DATETIME for timestamps).
    - Use NOT NULL constraints for mandatory attributes to avoid missing data.

These constraints safeguard data integrity and ensure a well-functioning system.

4. System Implementation using SQL

Data Manipulation:

- Inserting new data

```
-- Data insertion into Users table

INSERT INTO Users (name, email, password_hash, phone_number,
user_type)
VALUES ('John Doe', 'john.doe@email.com', 'hashed_password_1', '123-
456-7890', 'customer');

INSERT INTO Users (name, email, password_hash, phone_number,
user_type)
VALUES ('Jane Smith', 'jane.smith@email.com', 'hashed_password_2',
'987-654-3210', 'customer');

INSERT INTO Users (name, email, password_hash, user_type)
VALUES ('Admin User', 'admin@email.com', 'hashed_password_admin',
'admin');

INSERT INTO Users (name, email, password_hash, phone_number,
user_type)
VALUES ('David Lee', 'david.lee@email.com', 'hashed_password_4', '456-
123-7890', 'customer');


-- Data insertion into Restaurants table

INSERT INTO Restaurants (name, address, cuisine_type,
minimum_order_amount, delivery_radius, average_rating, image)
VALUES ('Pizza Palace', '123 Main St, Anytown, CA 12345', 'Italian',
15.00, 5, 4.2, 'restaurant1.jpg');

INSERT INTO Restaurants (name, address, cuisine_type,
minimum_order_amount, delivery_radius, average_rating, image)
VALUES ('Burger Barn', '456 Elm St, Anytown, CA 12345', 'American',
10.00, 3, 4.7, 'restaurant2.jpg');

INSERT INTO Restaurants (name, address, cuisine_type,
minimum_order_amount, delivery_radius, image)
VALUES ('Indian Curry House', '789 Oak St, Anytown, CA 12345',
'Indian', 20.00, 7, 'restaurant3.jpg');

INSERT INTO Restaurants (name, address, cuisine_type,
minimum_order_amount, delivery_radius, average_rating, image)
VALUES ('Taco Fiesta', '101 Pine St, Anytown, CA 12345', 'Mexican',
12.00, 4, 3.8, 'restaurant4.jpg');


-- Data insertion into Menu_Items table

-- Pizza Palace (Restaurant 1)
INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (1, 'Cheese Pizza (Small)', 'Classic cheese pizza with
mozzarella cheese.', 8.99, 'pizza1.jpg', 'Pizzas');
```

```sql
INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (1, 'Pepperoni Pizza (Small)', 'Classic pizza with mozzarella
cheese and pepperoni.', 10.99, 'pizza2.jpg', 'Pizzas');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (1, 'Veggie Pizza (Small)', 'Classic pizza with mozzarella
cheese and assorted veggies.', 9.99, 'pizza3.jpg', 'Pizzas');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (1, 'Garlic Breadsticks', 'Six warm and garlicky breadsticks.',
4.99, 'breadsticks.jpg', 'Sides');


-- Burger Barn (Restaurant 2)
INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (2, 'Cheeseburger (Regular)', 'Classic cheeseburger with beef
patty, cheese, lettuce, tomato, onion, and pickles on a toasted bun.',
8.50, 'burger1.jpg', 'Burgers');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (2, 'Fries (Large)', 'Large order of crispy French fries.',
2.99, 'fries.jpg', 'Sides');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (2, 'Chocolate Shake', 'Rich and creamy chocolate milkshake.',
3.50, 'shake.jpg', 'Drinks');


-- Indian Curry House (Restaurant 3)
INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (3, 'Butter Chicken', 'Classic Indian dish with tender chicken
in a creamy tomato gravy.', 14.99, 'butterchicken.jpg', 'Mains');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (3, 'Saag Paneer', 'Spinach and cottage cheese curry with a
touch of spice.', 12.99, 'saagpaneer.jpg', 'Mains');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (3, 'Garlic Naan', 'Soft and fluffy naan bread flavored with
garlic.', 2.50, 'naan.jpg', 'Sides');


-- Taco Fiesta (Restaurant 4)
INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (4, 'Beef Tacos (Combo)', 'Two tacos with seasoned beef,
lettuce, cheese, and pico de gallo.', 8.99, 'tacos.jpg', 'Mains');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
```

```sql
VALUES (4, 'Chicken Quesadilla', 'Large flour tortilla filled with
seasoned chicken, cheese, and onions.', 9.99, 'quesadilla.jpg',
'Mains');

INSERT INTO Menu_Items (restaurant_id, name, description, price,
image, category)
VALUES (4, 'Nachos', 'Crispy tortilla chips topped with melted cheese,
beans, pico de gallo, and sour cream.', 6.50, 'nachos.jpg', 'Sides');


-- Data insertion into Orders table

-- John Doe (User 1) Orders
INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (1, 1, '2024-07-16 12:00:00', 13.98, 'cash_on_delivery',
'pending', 'placed');  -- Order from Pizza Palace (Italian)

INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (1, 2, '2024-07-17 18:00:00', 15.49, 'online_payment', 'paid',
'delivered');  -- Completed order from Burger Barn (American)


-- Jane Smith (User 2) Orders
INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (2, 3, '2024-07-15 20:00:00', 17.49, 'cash_on_delivery',
'paid', 'delivered');  -- Completed order from Indian Curry House
(Indian)


-- David Lee (User 4) Orders (for recent order queries)
INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (4, 4, '2024-07-18 12:30:00', 11.49, 'online_payment',
'pending', 'placed');  -- Pending order from Taco Fiesta (Mexican)

INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (4, 1, '2024-07-14 19:00:00', 18.99, 'cash_on_delivery',
'paid', 'delivered');  -- Completed order from Pizza Palace (Italian -
for order history)

INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (4, 2, '2024-07-19 10:00:00', 12.99, 'online_payment',
'pending', 'placed');


-- Order from John Doe (User 1) from a new restaurant (Mexican)
INSERT INTO Orders (user_id, restaurant_id, order_date, total_amount,
payment_method, payment_status, order_status)
VALUES (1, 4, '2024-07-18 19:00:00', 9.49, 'cash_on_delivery',
'pending', 'placed');


-- Data insertion into Delivery_partners table
```

```sql
INSERT INTO Delivery_Partners (name, phone_number, vehicle_type,
availability, average_rating)
VALUES ('Michael Brown', '555-123-4568', 'Bike', TRUE, 4.8);

INSERT INTO Delivery_Partners (name, phone_number, vehicle_type,
availability, average_rating)
VALUES ('Sarah Lee', '123-456-7890', 'Car', FALSE, NULL);  --
Unavailable delivery partner


-- Data insertion into Order_tracking table

-- Order Tracking for John Doe's Orders
-- (Referencing Order IDs from Orders table)
-- Pizza Palace Order (Order ID 1)
INSERT INTO Order_Tracking (order_id, delivery_partner_id, status,
timestamp)
VALUES (1, 1, 'placed', '2024-07-16 12:00:00');


-- Burger Barn Order (Order ID 2)
INSERT INTO Order_Tracking (order_id, delivery_partner_id, status,
timestamp)
VALUES (2, 1, 'placed', '2024-07-17 18:00:00');

INSERT INTO Order_Tracking (order_id, delivery_partner_id, status,
timestamp)
VALUES (2, 1, 'out_for_delivery', '2024-07-17 18:30:00');

INSERT INTO Order_Tracking (order_id, delivery_partner_id, status,
timestamp)
VALUES (2, 1, 'delivered', '2024-07-17 19:00:00');


-- Data insertion into Reviews table

-- Review for John Doe's completed Burger Barn order (Order ID 2)
INSERT INTO Reviews (user_id, order_id, review_type, rating, comment)
VALUES (1, 2, 'restaurant', 5, 'Great burger and fries! Fast
delivery.');

INSERT INTO Reviews (user_id, order_id, review_type, rating, comment)
VALUES (1, 2, 'delivery_partner', 5, 'Delivery was quick and
efficient.');


-- Review for Jane Smith's completed Indian Curry House order (assumed
based on order date)
INSERT INTO Reviews (user_id, order_id, review_type, rating, comment)
VALUES (2, 3, 'restaurant', 4.5, 'Delicious butter chicken and naan!
Will order again.');


-- Negative review for Taco Fiesta (Restaurant 4)
INSERT INTO Reviews (user_id, order_id, review_type, rating, comment)
VALUES (4, 1, 'restaurant', 2, 'Food was cold and arrived late.');


-- Review for Pizza Palace (Restaurant 1) with a 3-star rating
```

```
INSERT INTO Reviews (user_id, order_id, review_type, rating, comment)
VALUES (1, 1, 'restaurant', 3, 'The pizza was average.');


-- Data insertion into Coupons table

INSERT INTO Coupons (code, discount_type, discount_value,
minimum_order_value, expiry_date)
VALUES ('SUMMER10', 'percentage', 10.00, 20.00, '2024-08-15');  --
Active percentage discount coupon

INSERT INTO Coupons (code, discount_type, discount_value, expiry_date)
VALUES ('FREEDELIVERY', 'amount', 5.00, '2024-07-20');  -- Active flat
discount coupon for delivery fee

INSERT INTO Coupons (code, discount_type, discount_value,
minimum_order_value, expiry_date)
VALUES ('EXPIREDCOUPON', 'percentage', 15.00, 15.00, '2024-07-10');  -
- Expired coupon
```

- Updating and delete existing data

```
-- Update a user's phone number
UPDATE Users SET phone_number = '678-901-2345' WHERE user_id = 3;


-- Mark an order as delivered
UPDATE Orders
SET order_status = 'delivered'
WHERE order_id = 7;


-- Delete a coupon that has expired
DELETE FROM Coupons
WHERE expiry_date < CURDATE();
```

Data Retrieval:

- Selecting specific data based on criteria

```
-- Find all restaurants with a specific cuisine type
SELECT * FROM Restaurants WHERE cuisine_type = 'Italian';


-- List orders whose payment mode is cash on delivery and payment
status is pending
SELECT * FROM Orders WHERE payment_method = 'cash_on_delivery' AND
payment_status = 'pending';


-- User with the highest total spending on orders in the last week
SELECT u.name, SUM(o.total_amount) AS total_spent
FROM Users u
INNER JOIN Orders o ON u.user_id = o.user_id
WHERE o.order_date >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
GROUP BY u.name
ORDER BY total_spent DESC
LIMIT 1;
```

```
-- Restaurant with the second highest average rating
SELECT name, cuisine_type, average_rating FROM Restaurants
ORDER BY average_rating DESC
LIMIT 1 OFFSET 1;

-- Find the busiest day of the week for orders
SELECT DAYNAME(order_date) AS day_of_week, COUNT(*) AS order_count
FROM Orders
GROUP BY DAYNAME(order_date)
ORDER BY order_count DESC
LIMIT 1;
```

- Filtering and sorting data based on conditions

```
-- Find restaurant within a specific delivery radius and with a
minimum rating
SELECT * FROM Restaurants
WHERE delivery_radius < 5 AND average_rating > 4;


-- List orders placed during a specific time period, grouped by date
and sorted by total amount
SELECT DATE(order_date) AS order_date, SUM(total_amount) AS
total_sales
FROM Orders
WHERE order_date BETWEEN '2024-07-16' AND '2024-07-19'
GROUP BY DATE(order_date)
ORDER BY total_sales DESC;
```

- Joining tables for comprehensive data retrieval

```
-- Get details of orders with associated restaurant name, user name
SELECT o.*, r.name AS restaurant_name, u.name AS user_name
FROM Orders o
INNER JOIN Restaurants r ON o.restaurant_id = r.restaurant_id
INNER JOIN Users u ON o.user_id = u.user_id;

-- Find users who haven't ordered from a specific restaurant in a
certain period
SELECT u.name AS user_name
FROM Users u
WHERE u.user_id NOT IN (
  SELECT user_id
  FROM Orders o
  WHERE o.restaurant_id = 1 AND DATE(o.order_date) >=
DATE_SUB(CURDATE(), INTERVAL 2 DAY)
);
```

- Subqueries

```
-- Select restaurants with a review rating greater than 4
SELECT r.*
FROM Restaurants r
WHERE r.restaurant_id IN (
  SELECT restaurant_id FROM Orders o
  WHERE o.order_id IN (
    SELECT order_id FROM Reviews WHERE review_type = 'restaurant' AND
rating >= 4
  )
);


-- List users who haven't placed any orders in the last week
SELECT u.name
FROM Users u
WHERE u.user_id NOT IN (
  SELECT user_id FROM Orders
  WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
);
```

Advanced Functionalities:

- Window functions for calculations within result sets

```
-- Rank restaurants by average rating
SELECT r.*,
       RANK() OVER (ORDER BY average_rating DESC) AS rest_rank
FROM Restaurants r
ORDER BY rest_rank;


-- Calculate the difference between order total and average order
value for each order
SELECT o.*,
       total_amount - AVG(total_amount) OVER (ORDER BY order_date) AS
difference
FROM Orders o;


-- Calculate moving average for daily order count
SELECT DATE(order_date), COUNT(*) AS order_count,
       AVG(COUNT(*)) OVER (ORDER BY DATE(order_date) ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS moving_average
FROM Orders
GROUP BY DATE(order_date)
ORDER BY DATE(order_date);
```

- Common Table Expressions (CTEs) for complex data retrieval scenarios

```
-- Find users who have placed orders exceeding a certain total amount
WITH high_value_orders AS (
  SELECT * FROM Orders WHERE total_amount > 13
)
SELECT u.name, COUNT(*) AS order_count
FROM Users u
INNER JOIN high_value_orders hvo ON u.user_id = hvo.user_id
GROUP BY u.name;


-- Identify users who placed orders exceeding a specific average order
value for their previous orders
WITH user_avg_orders AS (
  SELECT user_id, AVG(total_amount) AS avg_order_value
  FROM Orders
  GROUP BY user_id
)
SELECT u.name, o.order_id, o.total_amount
FROM Users u
INNER JOIN Orders o ON u.user_id = o.user_id
INNER JOIN user_avg_orders ua ON u.user_id = ua.user_id
WHERE o.total_amount > ua.avg_order_value;
```

- Triggers for automated actions based on data modifications

```
-- Automatically update payment_status when order status changes to
"delivered"
DELIMITER //
CREATE TRIGGER order_delivered_notification
AFTER UPDATE ON Orders
FOR EACH ROW
IF NEW.order_status = 'delivered' THEN
    UPDATE Orders
    SET payment_status = 'paid'
    WHERE order_id = OLD.order_id;
END IF; //
DELIMITER ;

-- Delete Trigger for Expired Coupons
DELIMITER //
CREATE TRIGGER cleanup_expired_coupons
AFTER DELETE ON Orders
FOR EACH ROW
DELETE FROM Coupons WHERE expiry_date < CURDATE();
DELIMITER ;
```

- Views for creating virtual tables with pre-defined joins and filters

```
-- Create a view combining frequently used order data for reporting
CREATE VIEW order_report AS
SELECT o.order_id, r.name AS restaurant_name, u.name AS user_name,
       o.order_date, o.total_amount
FROM Orders o
INNER JOIN Restaurants r ON o.restaurant_id = r.restaurant_id
INNER JOIN Users u ON o.user_id = u.user_id;

-- User Order History View
CREATE VIEW user_order_history AS
SELECT o.order_id, r.name AS restaurant_name, o.order_date,
o.total_amount, o.order_status
FROM Orders o
INNER JOIN Restaurants r ON o.restaurant_id = r.restaurant_id
WHERE o.user_id = 1;
```

5. Conclusion

This project presented the design and implementation of a database management system for an online food ordering system. The implemented data model, schema design, and SQL functionalities provide a robust foundation for data management and manipulation within the system. This project demonstrates a strong understanding of database design principles, SQL querying techniques, and data security considerations.