# QuickEats – Food Delivery

## 1. Description of the Project

The objective of this project is to create a database system for food delivery that will help in running the "QuickEats" food delivery service. The system will handle food items, restaurant menus, customer orders, delivery logistics, and feedback, guaranteeing seamless operation for customers, delivery agents, and restaurant partners. Key company data, including user information, order details, restaurant listings, menu items, and delivery performance, will be stored and arranged in this database. The creation of this database will enable the system to keep track of each customer's past orders, let them order, explore menus from other restaurants, and ask for comments regarding their delivery experience. Data will be efficiently stored by the system, making it simple for business to retrieve it.

## 2. Data Used and Source

The food_orders.xlsx file is the main source of data for this project and contains the important tables required for the database. The spreadsheet contains the following data:

- Orders: Details regarding the orders placed by customers, such as the order number, user ID, restaurant ID, total amount, date, delivery partner, and customer review.
- Restaurants: Information on restaurants that have partnered with the platform, such as name, ID, and type of cuisine served.
- Menu: Details about the food items in restaurant menus, such as item ID, restaurant ID, price, and type (veg or non-veg).
- Order Details: Detailed breakdown of every order, including amounts and related order IDs for each food item.
- Users and Delivery Partners: Based on user IDs and partner IDs, the dataset provides indirect access to customer and delivery partner information.

These data points will help create various tables in the database that capture the relationships between customers, restaurants, food items, and the delivery process.

# 3. Conceptual Model

**Entities**:

- User
- Restaurant
- Food
- Order
- Delivery Partner

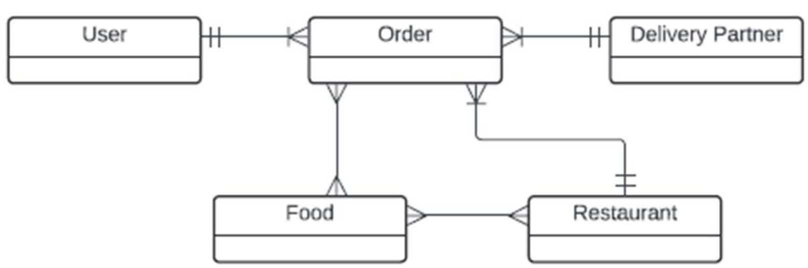
**Relationships:**

User-Order (One-to-Many): A user can place multiple orders, but each order belongs to one user.

Order-Delivery Partner (Many-to-One): An order is delivered by one delivery partner, though a partner can handle multiple orders.

Order-Food Item (Many-to-Many): An order can contain multiple food items, and each food item can be in multiple orders.

Restaurant-Food Item (Many-to-Many): A restaurant can offer multiple food items, and the same food item can be available at different restaurants.




**Conceptual Data Model**

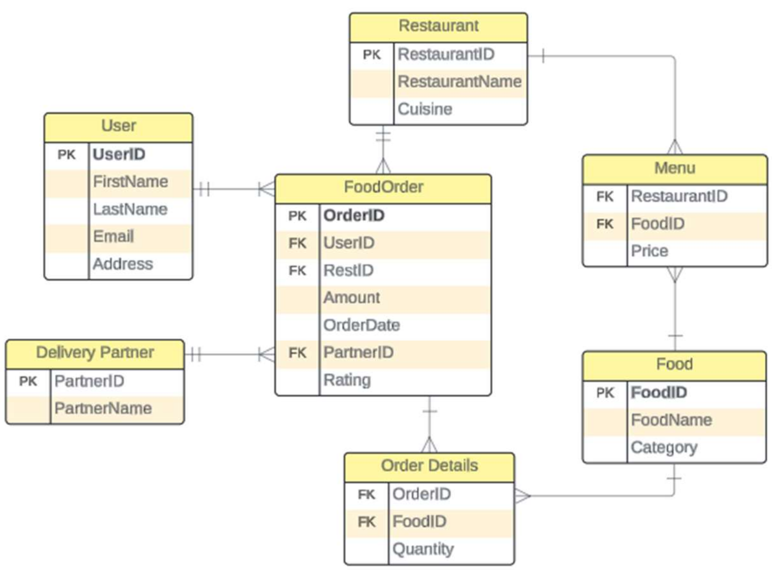
## 4. Logical Model

To resolve the many-to-many relationships identified in the conceptual model, we'll create junction entities:

1. Menu: This entity will resolve the many-to-many relationship between Restaurant and Food Item to indicate which food items each restaurant offers and their respective prices.
2. Order Details: This entity will resolve the many-to-many relationship between Order and Food Item, detailing the items in each order.

**Logical Model Entities and Relationships**

- User
- Restaurant
- Food
- Food Order
- Delivery Partner
- Menu – resolves Restaurant-Food many-to-many relationship
- Order Details – resolves Food Order-Food Item many-to-many relationship



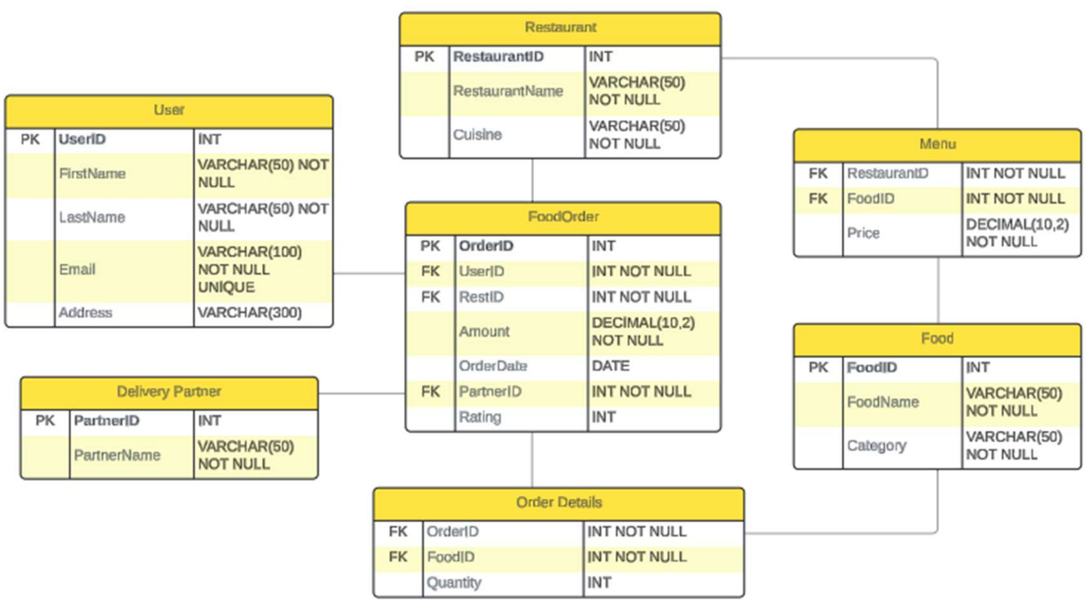

Logical Data Model

# 5. Physical Model

The Physical Data Model builds on the Logical Model by specifying the actual implementation details, including data types, primary keys (PK), foreign keys (FK), and constraints. Below is the textual representation of the physical model for each table, including attributes, and data types

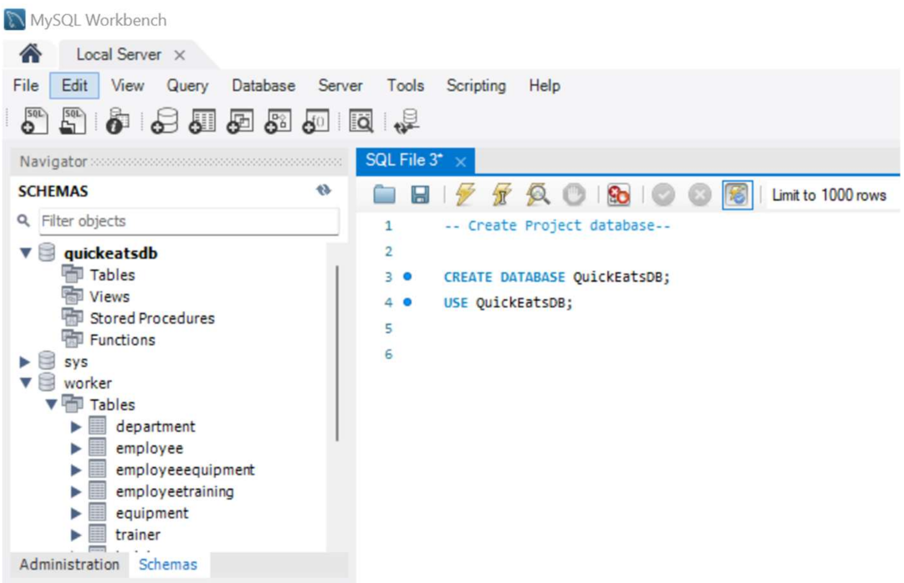
### Physical Model Entities and Attributes

- User
- Restaurant
- Food
- Food Order
- Delivery Partner
- Menu (Junction Table)
- Order Details (Junction Table)

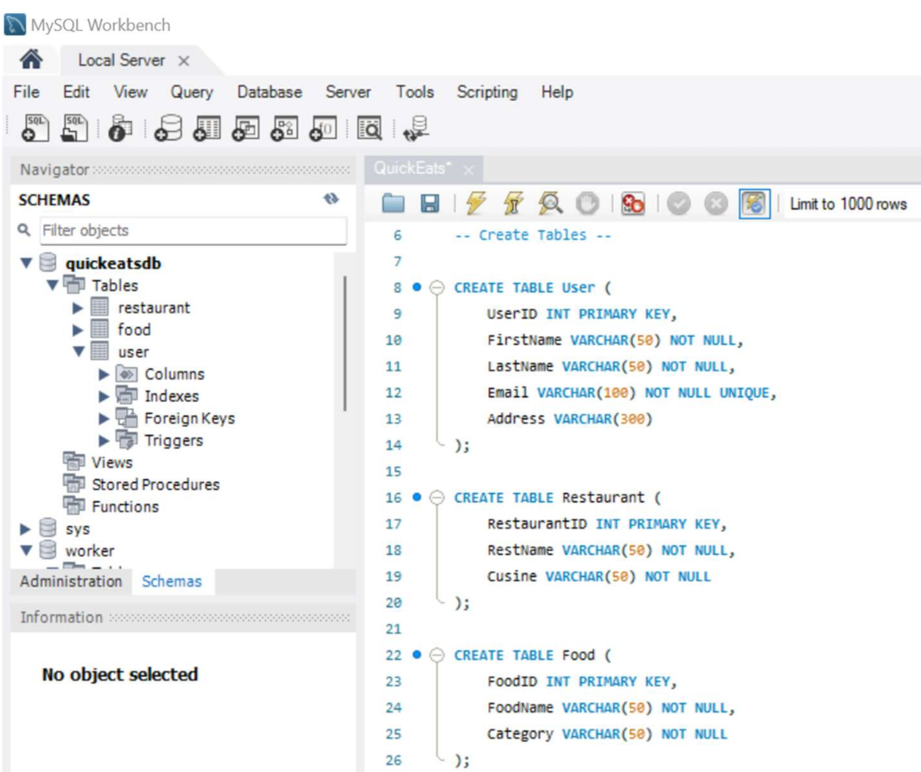



Physical Data Model

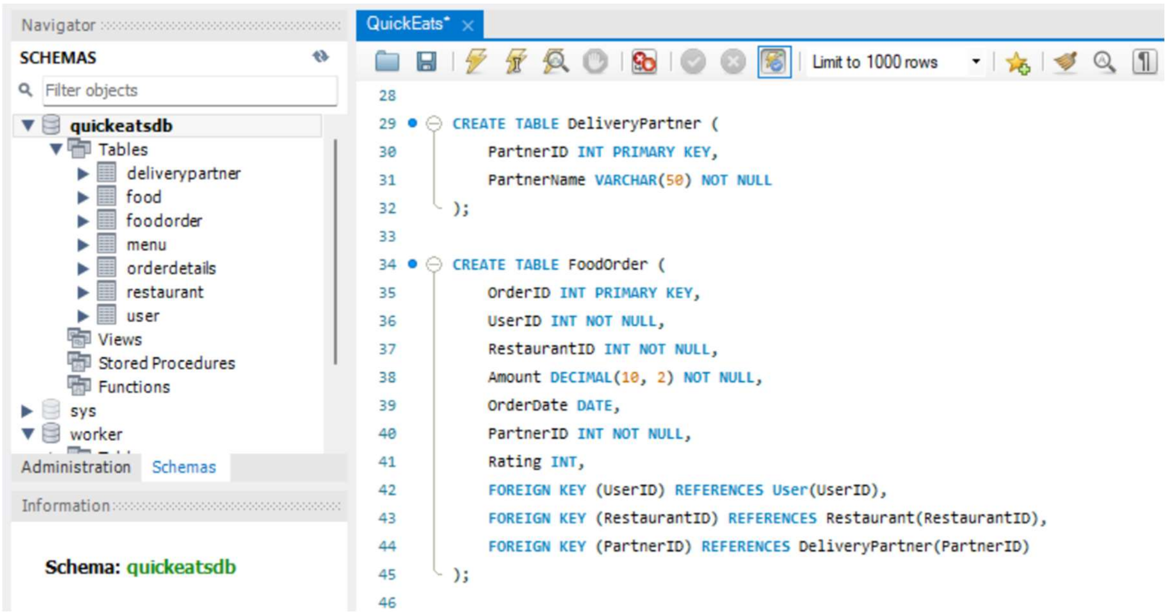
**6. Create tables using a database system. Insert data into the database tables. You must provide the DDL (CREATE TABLE statements), INSERT statements, and SELECT statements.**



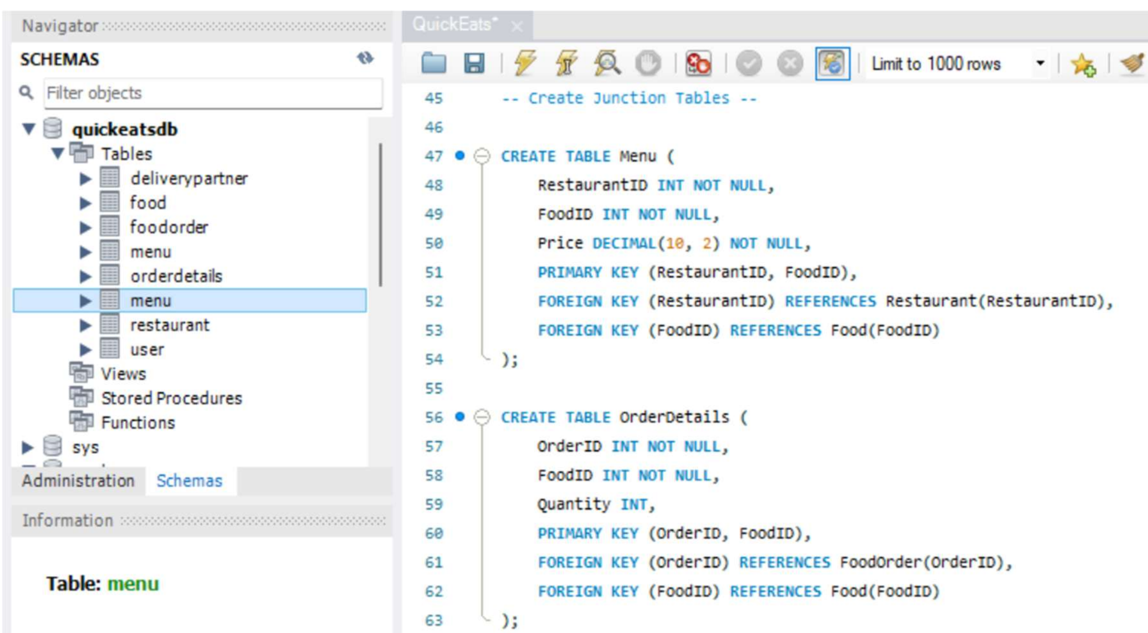

Created **QuickEats** database. Further creating the **Tables**.



Created tables – User, Restaurant, Food.




Created tables – DeliveryPartner, FoodOrder.




Created junction tables – Menu, OrderDetails

Table Definition implemented.

```sql
-- Show the DDL for the table
SHOW CREATE TABLE User;

SHOW CREATE TABLE Food;

SHOW CREATE TABLE Restaurant;

SHOW CREATE TABLE Menu;

SHOW CREATE TABLE DeliveryPartner;
```

Form Editor | Navigate: 1 / 1

Create Table:

```sql
CREATE TABLE `user` (
  `UserID` int NOT NULL,
  `FirstName` varchar(50) NOT NULL,
  `LastName` varchar(50) NOT NULL,
  `Email` varchar(100) NOT NULL,
  `Address` varchar(300) DEFAULT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE KEY `Email` (`Email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```sql
SHOW CREATE TABLE FoodOrder;

SHOW CREATE TABLE OrderDetails;
```

Form Editor | Navigate: 1 / 1

Create Table:

```sql
CREATE TABLE `foodorder` (
  `OrderID` int NOT NULL,
  `UserID` int NOT NULL,
  `RestaurantID` int NOT NULL,
  `Amount` decimal(10,2) NOT NULL,
  `OrderDate` date DEFAULT NULL,
  `PartnerID` int NOT NULL,
  `Rating` int DEFAULT NULL,
  PRIMARY KEY (`OrderID`),
  KEY `UserID` (`UserID`),
  KEY `RestaurantID` (`RestaurantID`),
  KEY `PartnerID` (`PartnerID`),
  CONSTRAINT `foodorder_ibfk_1` FOREIGN KEY (`UserID`) REFERENCES `user` (`UserID`),
  CONSTRAINT `foodorder_ibfk_2` FOREIGN KEY (`RestaurantID`) REFERENCES `restaurant` (`RestaurantID`),
  CONSTRAINT `foodorder_ibfk_3` FOREIGN KEY (`PartnerID`) REFERENCES `deliverypartner` (`PartnerID`)
```

Inserting **Data**

```sql
-- Insert Data --

INSERT INTO DeliveryPartner (PartnerID, PartnerName) VALUES
(1, 'Gunther'),(2, 'Janice'),(3, 'David'),(4, 'Ben'),(5, 'Carol');

INSERT INTO User (UserID, FirstName, LastName, Email, Address) VALUES
(1, 'Joey', 'Tribbiani', 'joeyt@gmail.com', '1996 4th Street, Apt 2B, New York, NY 10001'),
(2, 'Chandler', 'Bing', 'bing@gmail.com', '1996 4th Street, Apt 2B, New York, NY 10001'),
(3, 'Ross', 'Geller', 'geller@gmail.com', '495 Grove Street, Apt 20, New York, NY 10002'),
(4, 'Phoebe', 'Buffay', 'phoebe@gmail.com', '7 Lullaby Lane, New York, NY 10003'),
(5, 'Monica', 'Geller', 'monica@gmail.com', '1996 4th Street, Apt 2A, New York, NY 10001'),
(6, 'Rachel', 'Green', 'rachel@gmail.com', '1996 4th Street, Apt 2A, New York, NY 10001'),
(7, 'Mike', 'Hannigan', 'mike@gmail.com', '22 Love Lane, New York, NY 10005');

INSERT INTO Restaurant (RestaurantID, RestaurantName, Cusine) VALUES
(1, 'Dominos', 'Italian'),(2, 'KFC', 'American'),(3, 'Chipotle', 'Mexican'),
(4, 'Indian Darbar', 'Indian'),(5, 'China Town', 'Chinese');

INSERT INTO Food (FoodID, FoodName, Category) VALUES
(1, 'Chicken Pizza', 'Non-veg'),(2, 'Mushroom Pizza', 'Veg'),(3, 'Sandwich', 'Veg'),(4, 'Chicken Wings', 'Non-veg'),
(5, 'Chicken Popcorn', 'Non-veg'),(6, 'Chicken Bowl', 'Non-veg'),(7, 'Burrito', 'Non-veg'),(8, 'Butter Chicken', 'Non-veg'),
(9, 'Dal Tadka', 'Veg'),(10, 'Fried Rice', 'Veg'),(11, 'Kung Pao Chicken', 'Non-veg');
```

Inserted data in following tables – DeliveryPartner, User, Restaurant, Food

```sql
(9, 'Dal Tadka', 'Veg'),(10, 'Fried Rice', 'Veg'),(11, 'Kung Pao Chicken', 'Non-veg');

INSERT INTO Menu (RestaurantID, FoodID, Price) VALUES
(1, 1, 15),(1, 2, 12),(1, 3, 9),(2, 3, 10),(2, 4, 9),(2, 5, 8),(3, 5, 12),(3, 6, 9),(3, 7, 14),
(4, 4, 13),(4, 8, 18),(4, 9, 10),(4, 10, 12),(5, 5, 15),(5, 6, 12),(5, 10, 10),(5, 11, 12);

INSERT INTO FoodOrder (OrderID, UserID, RestaurantID, Amount, OrderDate, PartnerID, Rating) VALUES
(1001, 1, 1, 24, '2024-05-10', 1, 5),(1002, 1, 2, 19, '2024-05-26', 1, 5),(1003, 2, 3, 21, '2024-06-15', 5, 4),
(1004, 2, 5, 22, '2024-06-29', 4, 3),(1005, 3, 5, 22, '2024-05-10', 1, 1),(1006, 3, 1, 26, '2024-06-10', 2, 5),
(1007, 4, 2, 19, '2024-06-23', 3, 1),(1008, 4, 3, 23, '2024-07-07', 5, 4),(1009, 5, 4, 31, '2024-07-17', 4, 5),
(1010, 6, 5, 34, '2024-05-30', 1, 1),(1011, 6, 1, 15, '2024-06-10', 2, 3),(1012, 6, 4, 18, '2024-07-20', 5, 4);

INSERT INTO OrderDetails (OrderID, FoodID, Quantity) VALUES
(1001, 1, 1),(1001, 3, 1),(1002, 4, 1),(1002, 3, 1),(1003, 6, 1),(1003, 5, 1),(1004, 6, 1),(1004, 10, 1),(1005, 10, 1),
(1005, 11, 1),(1006, 1, 1),(1006, 2, 1),(1006, 3, 1),(1007, 4, 1),(1007, 3, 1),(1008, 6, 1),(1008, 7, 1),(1009, 4, 1),
(1009, 8, 1),(1010, 6, 1),(1010, 10, 1),(1010, 11, 1),(1011, 1, 1),(1012, 8, 1);
```

Inserted data in tables – Menu, FoodOrder, OrderDetails.

# 7. Create a variety of SQL queries to retrieve data from one or many tables.

1. Retrieve the data from each table by using the SELECT * statement and order by PK column(s).
Show the output. Make sure you show the print screen of the complete set of rows and columns.
The rows must be ordered by PK column(s).

Retrieving data from each table.

```sql
-- Run Queries --

-- 1 Retrieve the data from each table by using the SELECT * statement and order by PK column(s) --

SELECT * FROM User ORDER BY UserID;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| UserID | FirstName | LastName | Email | Address |
|---|---|---|---|---|
| 1 | Joey | Tribbiani | joeyt@gmail.com | 1996 4th Street, Apt 2B, New York, NY 10001 |
| 2 | Chandler | Bing | bing@gmail.com | 1996 4th Street, Apt 2B, New York, NY 10001 |
| 3 | Ross | Geller | geller@gmail.com | 495 Grove Street, Apt 20, New York, NY 10002 |
| 4 | Phoebe | Buffay | phoebe@gmail.com | 7 Lullaby Lane, New York, NY 10003 |
| 5 | Monica | Geller | monica@gmail.com | 1996 4th Street, Apt 2A, New York, NY 10001 |
| 6 | Rachel | Green | rachel@gmail.com | 1996 4th Street, Apt 2A, New York, NY 10001 |
| 7 | Mike | Hannigan | mike@gmail.com | 22 Love Lane, New York, NY 10005 |

User 1 ×

```sql
SELECT * FROM Restaurant ORDER BY RestaurantID;
```

Result Grid | Filter Rows: | Edit: | Expo

| RestaurantID | RestaurantName | Cusine |
|---|---|---|
| 1 | Dominos | Italian |
| 2 | KFC | American |
| 3 | Chipotle | Mexican |
| 4 | Indian Darbar | Indian |
| 5 | China Town | Chinese |
| NULL | NULL | NULL |

```sql
SELECT * FROM Food ORDER BY FoodID;
```

Result Grid | Filter Rows: | Edit: | Export/Import

| FoodID | FoodName | Category |
|---|---|---|
| 1 | Chicken Pizza | Non-veg |
| 2 | Mushroom Pizza | Veg |
| 3 | Sandwich | Veg |
| 4 | Cheesecake | Veg |
| 5 | Chicken Popcorn | Non-veg |
| 6 | Chicken Bowl | Non-veg |
| 7 | Burrito | Non-veg |
| 8 | Shahi Paneer | Veg |
| 9 | Dal Tadka | Veg |
| 10 | Fried Rice | Veg |
| 11 | Kung Pao Chicken | Non-veg |
| NULL | NULL | NULL |

```sql
SELECT * FROM DeliveryPartner ORDER BY PartnerID;
```

Result Grid | Filter Rows: | Edit:

| PartnerID | PartnerName |
|---|---|
| 1 | Gunther |
| 2 | Janice |
| 3 | David |
| 4 | Ben |
| 5 | Carol |
| NULL | NULL |

```sql
SELECT * FROM FoodOrder ORDER BY OrderID;
```

Result Grid | Filter Rows: | Edit: | Export/Import

| OrderID | UserID | RestaurantID | Amount | OrderDate | PartnerID | Rating |
|---|---|---|---|---|---|---|
| 1001 | 1 | 1 | 24.00 | 2024-05-10 | 1 | 5 |
| 1002 | 1 | 2 | 19.00 | 2024-05-26 | 1 | 5 |
| 1003 | 2 | 3 | 21.00 | 2024-06-15 | 5 | 4 |
| 1004 | 2 | 5 | 22.00 | 2024-06-29 | 4 | 3 |
| 1005 | 3 | 5 | 22.00 | 2024-05-10 | 1 | 1 |
| 1006 | 3 | 1 | 26.00 | 2024-06-10 | 2 | 5 |
| 1007 | 4 | 2 | 19.00 | 2024-06-23 | 3 | 1 |
| 1008 | 4 | 3 | 23.00 | 2024-07-07 | 5 | 4 |
| 1009 | 5 | 4 | 31.00 | 2024-07-17 | 4 | 5 |
| 1010 | 6 | 5 | 34.00 | 2024-05-30 | 1 | 1 |
| 1011 | 6 | 1 | 15.00 | 2024-06-10 | 2 | 3 |
| 1012 | 6 | 4 | 18.00 | 2024-07-20 | 5 | 4 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```sql
SELECT * FROM Menu ORDER BY RestaurantID, FoodID;
```

| RestaurantID | FoodID | Price |
|---|---|---|
| 1 | 1 | 15.00 |
| 1 | 2 | 12.00 |
| 1 | 3 | 9.00 |
| 2 | 3 | 10.00 |
| 2 | 4 | 9.00 |
| 2 | 5 | 8.00 |
| 3 | 5 | 12.00 |
| 3 | 6 | 9.00 |
| 3 | 7 | 14.00 |
| 4 | 4 | 13.00 |
| 4 | 8 | 18.00 |
| 4 | 9 | 10.00 |
| 4 | 10 | 12.00 |

```sql
SELECT * FROM OrderDetails ORDER BY OrderID, FoodID;
```

| OrderID | FoodID | Quantity |
|---|---|---|
| 1001 | 1 | 1 |
| 1001 | 3 | 1 |
| 1002 | 3 | 1 |
| 1002 | 4 | 1 |
| 1003 | 5 | 1 |
| 1003 | 6 | 1 |
| 1004 | 6 | 1 |
| 1004 | 10 | 1 |
| 1005 | 10 | 1 |
| 1005 | 11 | 1 |
| 1006 | 1 | 1 |
| 1006 | 2 | 1 |
| 1006 | 3 | 1 |

Queries to retrieve data from each table. Ordered by their respective PK.

2. Write an SQL involving the junction table and two other related tables. You must use the INNER JOIN to connect with all three tables. The database that you created must be included in your SQL queries.

```sql
-- 2 Query with INNER JOIN using junction table and two related tables --

SELECT Restaurant.RestaurantName, Food.FoodName, Menu.Price
FROM Menu
INNER JOIN Restaurant ON Menu.RestaurantID = Restaurant.RestaurantID
INNER JOIN Food ON Menu.FoodID = Food.FoodID
ORDER BY Restaurant.RestaurantName, Food.FoodName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| RestaurantName | FoodName | Price |
|---|---|---|
| China Town | Chicken Bowl | 12.00 |
| China Town | Chicken Popcorn | 15.00 |
| China Town | Fried Rice | 10.00 |
| China Town | Kung Pao Chicken | 12.00 |
| Chipotle | Burrito | 14.00 |
| Chipotle | Chicken Bowl | 9.00 |
| Chipotle | Chicken Popcorn | 12.00 |
| Dominos | Chicken Pizza | 15.00 |
| Dominos | Mushroom Pizza | 12.00 |

Result 8

Query to get food items and their prices available in each restaurant. Query fetched restaurant name, food item name and its price.

3. Write an SQL by including two or more tables and using the LEFT OUTER JOIN. Show the results and sort the results by key field(s).

```sql
-- Query with LEFT OUTER JOIN --

SELECT User.FirstName, User.LastName, FoodOrder.OrderID, FoodOrder.Amount, FoodOrder.OrderDate
FROM User
LEFT OUTER JOIN FoodOrder ON User.UserID = FoodOrder.UserID
ORDER BY User.UserID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| FirstName | LastName | OrderID | Amount | OrderDate |
|---|---|---|---|---|
| Joey | Tribbiani | 1001 | 24.00 | 2024-05-10 |
| Joey | Tribbiani | 1002 | 19.00 | 2024-05-26 |
| Chandler | Bing | 1003 | 21.00 | 2024-06-15 |
| Chandler | Bing | 1004 | 22.00 | 2024-06-29 |
| Ross | Geller | 1005 | 22.00 | 2024-05-10 |
| Ross | Geller | 1006 | 26.00 | 2024-06-10 |
| Phoebe | Buffay | 1007 | 19.00 | 2024-06-23 |
| Phoebe | Buffay | 1008 | 23.00 | 2024-07-07 |
| Monica | Geller | 1009 | 31.00 | 2024-07-17 |
| Rachel | Green | 1010 | 34.00 | 2024-05-30 |
| Rachel | Green | 1011 | 15.00 | 2024-06-10 |
| Rachel | Green | 1012 | 18.00 | 2024-07-20 |
| Mike | Hannigan | NULL | NULL | NULL |

Query to retrieve all users and their food orders. Users and their order details fetched.

Interpretation of the above results.

INNER JOIN: Only rows with matching values in all tables are included. If there is no match in even one of the joined tables, the row is excluded.

LEFT OUTER JOIN: Displays all rows even if there are no corresponding records in FoodOrders or User. For example, OrderID, Amount, OrderDate appear as NULL.

4. Write a single-row subquery. Show the results and sort the results by key field(s). Interpret the output.

```
-- 4 Single-row subquery --

SELECT Food.FoodName, Menu.Price
FROM Menu
INNER JOIN Food ON Menu.FoodID = Food.FoodID
WHERE Menu.Price = (SELECT MAX(Price) FROM Menu)
ORDER BY Food.FoodName;
```

| FoodName | Price |
|---|---|
| Shahi Paneer | 18.00 |

Query to retrieve the food item with the highest price. Shahi Paneer is the costliest food item available on QuickEats database at $18.

5. Write a multiple-row subquery. Show the results and sort the results by key field(s). Interpret the output.

```
-- 5 Multiple-row subquery --

SELECT Restaurant.RestaurantName
FROM Restaurant
WHERE Restaurant.RestaurantID IN (
    SELECT Menu.RestaurantID
    FROM Menu
    WHERE Menu.Price > (SELECT AVG(Price) FROM Menu))
ORDER BY Restaurant.RestaurantName;
```

| RestaurantName |
|---|
| China Town |
| Chipotle |
| Dominos |
| Indian Darbar |

Query to retrieve all restaurants that serve food priced above the average price on the menu. Eating at China Town, Chipotle, Dominos, and Indian Darbar might won’t be pocket friendly for everyone.

6. Write an SQL to aggregate the results by using multiple columns in the SELECT clause. Interpret the output.

```sql
-- 6 Aggregate results with multiple columns --

SELECT DeliveryPartner.PartnerName, SUM(FoodOrder.Amount) AS TotalRevenue, AVG(FoodOrder.Rating) AS AvgRating
FROM DeliveryPartner
INNER JOIN FoodOrder ON DeliveryPartner.PartnerID = FoodOrder.PartnerID
GROUP BY DeliveryPartner.PartnerName
ORDER BY DeliveryPartner.PartnerName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| PartnerName | TotalRevenue | AvgRating |
|---|---|---|
| Ben | 53.00 | 4.0000 |
| Carol | 62.00 | 4.0000 |
| David | 19.00 | 1.0000 |
| Gunther | 99.00 | 3.0000 |
| Janice | 41.00 | 4.0000 |

Query to calculate the total revenue and average rating for each delivery partner.
Looks like Ben, Carol, and Janice are top rated delivery partners while David needs to improve his ratings or else he might get fired from QuickEats.

7. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s). Interpret the output.

```sql
-- 7 Subquery using NOT IN operator --

SELECT User.FirstName, User.LastName
FROM User
WHERE User.UserID NOT IN (SELECT FoodOrder.UserID FROM FoodOrder)
ORDER BY User.UserID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| FirstName | LastName |
|---|---|
| Mike | Hannigan |

Query to retrieve all users who have not placed any orders.
Mike Hannigan is yet to place any order on QuickEats.

8. Write a query using a CASE statement. Show the results and sort the results by key field(s). Interpret the output.

```sql
-- 8 Query using CASE statement --

SELECT Menu.RestaurantID, Food.FoodName, Menu.Price,
    CASE
        WHEN Menu.Price >= 11 THEN 'Expensive'
        ELSE 'Affordable'
    END AS PriceCategory
FROM Menu
INNER JOIN Food ON Menu.FoodID = Food.FoodID
ORDER BY Menu.RestaurantID, Food.FoodName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| RestaurantID | FoodName | Price | PriceCategory |
|---|---|---|---|
| 1 | Chicken Pizza | 15.00 | Expensive |
| 1 | Mushroom Pizza | 12.00 | Expensive |
| 1 | Sandwich | 9.00 | Affordable |
| 2 | Cheesecake | 9.00 | Affordable |
| 2 | Chicken Popcorn | 8.00 | Affordable |
| 2 | Sandwich | 10.00 | Affordable |
| 3 | Burrito | 14.00 | Expensive |
| 3 | Chicken Bowl | 9.00 | Affordable |
| 3 | Chicken Popcorn | 12.00 | Expensive |
| 4 | Cheesecake | 13.00 | Expensive |
| 4 | Dal Tadka | 10.00 | Affordable |

Query to classify food items as "Expensive" or "Affordable" based on price.
Filtered the affordable and expensive food items options from the QuickEats database.

9. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s). Interpret the output.

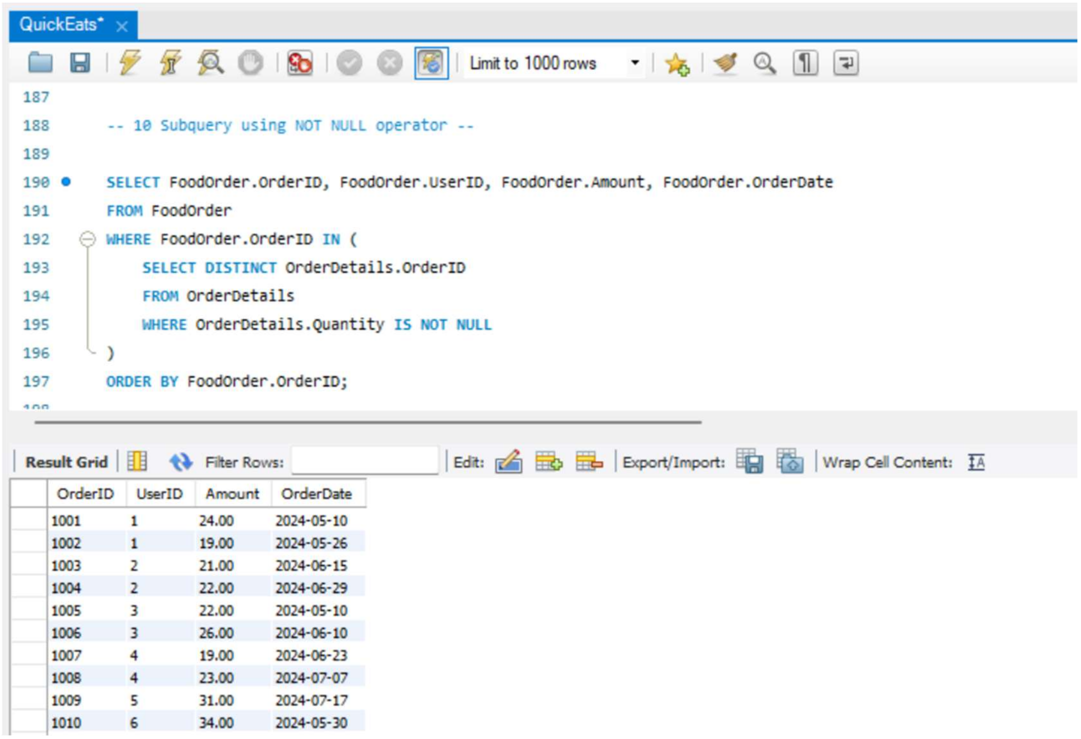
```sql
-- 9 Query using NOT EXISTS operator --

SELECT Restaurant.RestaurantID, Restaurant.RestaurantName
FROM Restaurant
WHERE NOT EXISTS (
    SELECT 1
    FROM Menu
    INNER JOIN Food ON Menu.FoodID = Food.FoodID
    WHERE Food.Category = 'Non-veg' AND Menu.RestaurantID = Restaurant.RestaurantID)
ORDER BY Restaurant.RestaurantID;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content

| RestaurantID | RestaurantName |
|---|---|
| 4 | Indian Darbar |
| NULL | NULL |

Query to find restaurants that do not offer Non-vegetarian food items. Indian Darbar is your go to place for the Vegetarian food.

10. Write a subquery using the NOT NULL operator in the inner query. Show the results and sort the results by key field(s). Interpret the output.




```sql
-- 10 Subquery using NOT NULL operator --

SELECT FoodOrder.OrderID, FoodOrder.UserID, FoodOrder.Amount, FoodOrder.OrderDate
FROM FoodOrder
WHERE FoodOrder.OrderID IN (
    SELECT DISTINCT OrderDetails.OrderID
    FROM OrderDetails
    WHERE OrderDetails.Quantity IS NOT NULL
)
ORDER BY FoodOrder.OrderID;
```

| OrderID | UserID | Amount | OrderDate |
|---|---|---|---|
| 1001 | 1 | 24.00 | 2024-05-10 |
| 1002 | 1 | 19.00 | 2024-05-26 |
| 1003 | 2 | 21.00 | 2024-06-15 |
| 1004 | 2 | 22.00 | 2024-06-29 |
| 1005 | 3 | 22.00 | 2024-05-10 |
| 1006 | 3 | 26.00 | 2024-06-10 |
| 1007 | 4 | 19.00 | 2024-06-23 |
| 1008 | 4 | 23.00 | 2024-07-07 |
| 1009 | 5 | 31.00 | 2024-07-17 |
| 1010 | 6 | 34.00 | 2024-05-30 |

Query to retrieve all orders where at least one item has a Quantity specified.

# Summary on QuickEats

This project focuses on the design and implementation of a relational database system for a food delivery application called QuickEats. The database consists of multiple interrelated tables, including Food, Menu, FoodOrders, OrderDetails, and DeliveryPartner, all of which are represented through Entity Relationship Diagrams (ERD) to visualize the relationships and data flow between entities. The objective is to ensure proper data normalization, reduce redundancy, and establish robust relationships between entities. The Menu acts as a junction table between Restaurant and Food, storing the price of each food item at different restaurants. Similarly, the OrderDetails table acts as a junction between FoodOrders and Food, capturing the quantity of each food item in an order. Key operations performed on the database include data insertion, retrieval of records using JOINS, as well as executing complex SQL queries with CASE, NOT EXISTS, and subqueries to demonstrate the effectiveness of the system. Through these queries, users can extract valuable information, such as customer order history, total order costs, food availability, and delivery partner performance.

The project emphasizes the importance of relational database concepts, such as ERD, primary keys, foreign keys, and referential integrity. Queries were implemented to retrieve, update, and analyze information in a way that supports end-user needs. Advanced SQL techniques, including aggregation, subqueries, and conditional logic, were used to enhance query efficiency and support decision-making. By incorporating both JOIN, the system demonstrates how different join operations affect query results, particularly when dealing with incomplete or missing data. This database project showcases core database design principles, SQL query proficiency, and the ability to model real-world business scenarios.

**THANK YOU**