CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF COMPUTING

CASE STUDY REPORT
ON

"Online Food Ordering and Delivering System"

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

Submitted by:	**Submitted to:**
Name: Nidhi Gera	Name: Mr. Arvinder Singh
UID:23BCA10010	Designation: Professor
Section: 23BCA4-B	

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

ABSTRACT

- **Introduction:**
- **Technique:**
- **System Configuration:**
- **INPUT:**
- **ER DIAGRAM:**
- **TABLE RELATION:**
- **TABULAR FORMAT:**
- **TABLE CREATION:**
- **SQL QUERIES WITH OUTPUT :**
- **SUMMARY:**
- **CONCLUSION:**

Introduction:

***The Online Food Ordering and Delivery System* is a database-driven application designed to facilitate seamless food ordering, delivery, and payment processes for users and restaurants. In today's fast-paced world, traditional methods of ordering food are being replaced by digital solutions. This system provides a convenient platform where customers can browse menus, place orders, make payments, and track their deliveries with ease.**

For restaurants, the system simplifies the management of menu items, order tracking, and payment processing, leading to increased efficiency. The application uses a *Relational Database Management System (RDBMS)*, ensuring secure data management and seamless operations. By leveraging SQL queries, the system efficiently handles large volumes of transactions while maintaining data integrity and minimizing redundancy.

This project aims to address the growing need for convenience and accessibility in the food service industry. It integrates ordering, payment, and delivery management into a single platform, offering a more efficient and reliable solution for both customers and restaurants.

Technique:

The system is built using a Relational Database Management System *(RDBMS)*, which organizes data in tables that are linked to each other. SQL *(Structured Query Language)* is used to interact with the database, allowing operations like adding, updating, or deleting records. To ensure the data is stored efficiently, the database is normalized, which reduces redundancy and ensures data accuracy.

An ER *(Entity-Relationship)* Diagram helps visualize how different parts of the system, like users, orders, and restaurants, are connected. SQL queries are then used to perform tasks such as retrieving orders or updating delivery statuses.

System Configuration:

Processor: Intel Core i3 or higher

RAM: 4 GB or higher

Storage: 1 GB free space for database setup

Software Requirements:

DBMS: MySQL (Any RDBMS supporting SQL queries)

Tools for SQL Development: MySQL Workbench

Operating System: Windows 10/Linux

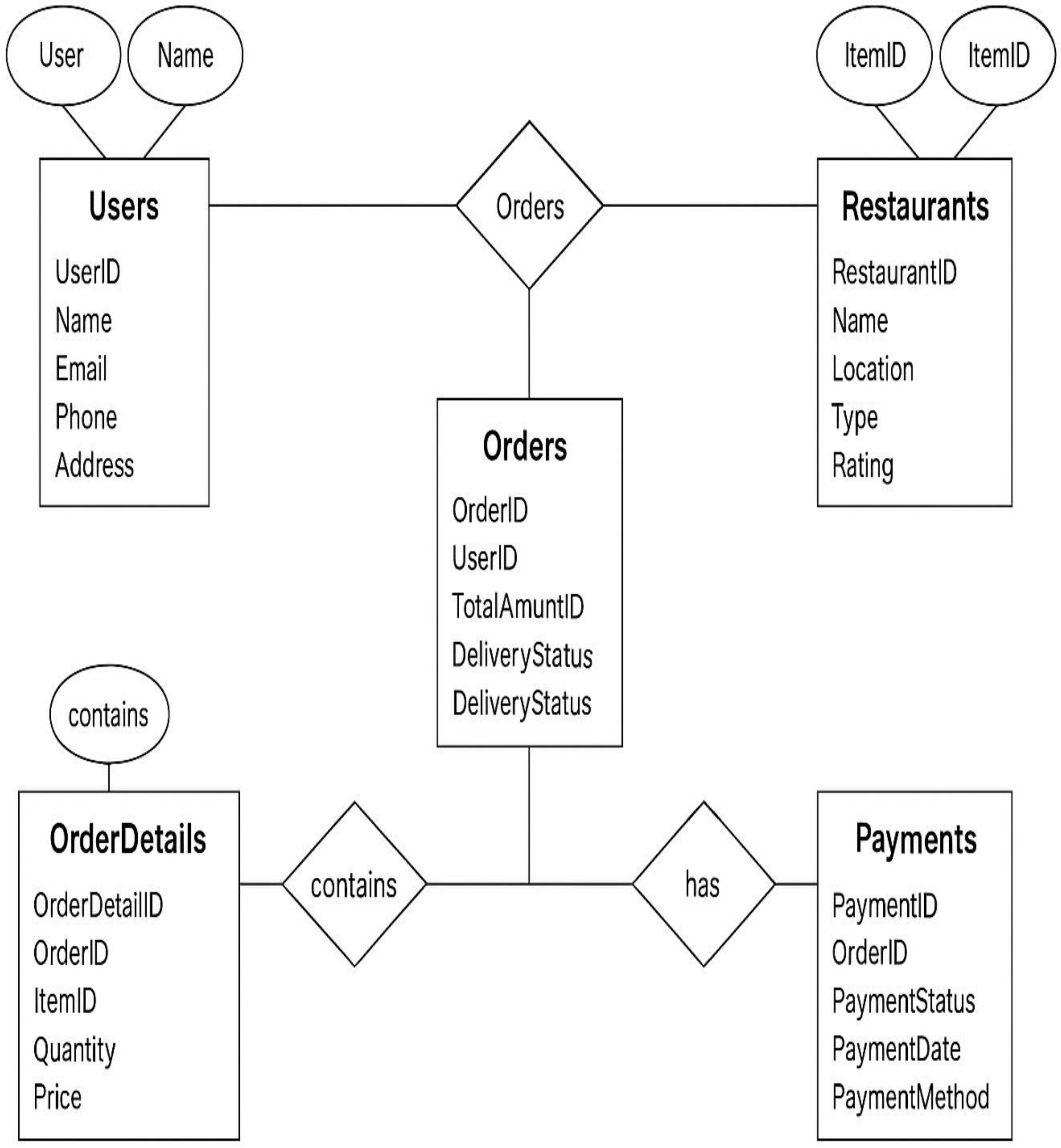
Input:

The input data for the Online Food Ordering and Delivery System consists of several key components:

1. **User Data: This includes the personal information of the customer, such as their name, email address, phone number, and delivery address. This data helps the system identify customers and ensure the correct delivery of orders.**
2. **Restaurant Data: Each restaurant's information is stored, including its name, type (e.g., fast food, fine dining), location, and a list of menu items available for ordering. This data allows customers to browse and select restaurants based on their preferences.**
3. **Order Data: When a customer places an order, the system records the quantity of each item ordered, the price of each item, and the total amount for the entire order. This information is critical for processing and managing the orders effectively.**
4. **Payment Data: The payment details include the payment status (whether the payment was successful or failed), the payment method used (e.g., credit card, digital wallet), and the date of transaction. This data is essential for confirming successful payments and ensuring that the restaurant is compensated for the order.**

ER Diagram:

The Entity-Relationship (ER) diagram visually represents the structure of the database for the *Online Food Ordering and Delivery System*.



EXPLANATION OF THE ABOVE ER DIAGRAM:

1. Customer

Description: Represents a user who places food orders through the system.

Attributes:

- **Customer_ID *(Primary Key)*: Unique ID assigned to each customer.**
- **Customer_Name: Full name of the customer.**
- **Email: Email address of the customer.**
- **Phone: Contact number.**
- **Address: Complete delivery address.**

Relationship:

- **A customer can place multiple orders → (1:N with Order).**
- **A customer can make multiple payments → (1:N with Payment).**

2. Restaurant:

Description: Represents a restaurant listed on the platform.

- **Restaurant_ID (Primary Key): Unique ID for each restaurant.**
- **Restaurant_Name: Name of the restaurant.**

- **Location: Physical location or branch.**
- **Contact: Contact number or email.**

Relationship:

- **A restaurant can offer multiple menu items → (1:N with Food_Item).**
- **A restaurant can receive multiple orders → (1:N with Order).**

3. <u>Food_Item:</u>

Description: Represents an item listed on a restaurant's menu.

Attributes:

- **Item_ID (Primary Key): Unique ID for each food item.**
- **Item_Name: Name of the dish.**
- **Price: Cost of the item.**
- **Category: Type of item (e.g., Starter, Main Course).**
- **Restaurant_ID (Foreign Key): Links to the restaurant offering the item.**

Relationship:

- **Each food item belongs to one restaurant.**

- Food items are included in orders through the Order_Details table.

4. Order list:

Description: Represents a customer's order placed through the system.

Attributes:

- **Order_ID (Primary Key): Unique ID of the order.**
- **Customer_ID (Foreign Key): ID of the customer who placed the order.**
- **Restaurant_ID (Foreign Key): ID of the restaurant where the order was placed.**
- **Order_Date: Date and time when the order was made.**
- **Total_Amount: Total bill for the order.**
- **Status: Current status of the order (e.g., Pending, Delivered).**

Relationship:

- **An order can include multiple food items via the Order_Details table.**
- **An order is linked to one payment.**

5. Order_Details:

Description: A junction table to manage the many-to-many relationship between Order and Food_Item.

Attributes:

- **OrderDetail_ID (Primary Key): Unique ID for each order item.**
- **Order_ID (Foreign Key): ID of the related order.**
- **Item_ID (Foreign Key): ID of the food item ordered.**
- **Quantity: Number of units ordered.**
- **Price: Price per unit (in case item prices change later).**

Relationship:

- **Connects each food item to an order.**
- **Helps calculate total amount.**

6. Payment:

Description: Tracks payment information for orders.

Attributes:

- **Payment_ID (Primary Key): Unique ID of the payment.**
- **Order_ID (Foreign Key): Links to the respective order.**

- **Customer_ID (Foreign Key): Customer who made the payment.**
- **Payment_Method: E.g., UPI, Credit Card, Wallet.**
- **Payment_Date: Date of payment.**
- **Status: Whether the payment was successful or failed.**

Relationship:

- **Each order has exactly one payment.**

Key Highlights of the Design:

- **Normalization: All entities are properly normalized to avoid data duplication.**
- **Data Integrity: Use of primary and foreign keys ensures that relationships remain valid.**
- **Flexibility: Can be easily extended to support reviews, delivery personnel, offers, etc.**
- **Real-World Mapping: Matches the real functionality of popular platforms like Swiggy, Uber Eats, or Zomato.**

Table Relation:

The tables in the Online Food Ordering and Delivery System are interlinked using primary keys (PK) and foreign keys (FK). These relationships form the backbone of the database and reflect the real-world connections between customers, restaurants, food items, orders, and payments.

Table 1	Table 2	Relation type	Key used
Customer	Order	1:N	Customer_ID (FK in Order)
Customer	Payment	1:N	Customer_ID (FK in Payment)
Restaurant	Food_item	1:N	Restaurant_ID (FK in Food_Item)
Restaurant	Order	1:N	Restaurant_ID (FK in Order)
Order	Payment	1:1	Order_ID (FK in Payment)
Order	Food_item	M:N	via Order_Details
Order_details	Food_item	M:N	Item_ID (FK in Order_Details)

Table Format:

1. Customer:

Field Name	Data type	Constraint	Description
Customer_Id	INT	Primary Key	Unique ID for each customer
Name	VARCHAR(100)	NOT NULL	Customer's full name
Email	VARCHAR(100)	UNIQUE	Customer's email address
Phone	VARCHAR(15)	NOT NULL	Contact number
Address	VARCHAR(255)	NOT NULL	Delivery address

2. Restaurant:

Field name	Data type	Constraint	Description
Restaurant_id	INT	NOT NULL	Unique ID for each restaurant
Name	VARCHAR(100)	NOT NULL	Restaurant name
Type	VARCHAR(50)	NOT NULL	Cuisine type
Location	VARCHAR(100)	NOT NULL	Address or area

3. Food_item:

Field name	Data type	Constraint	Description
Item_id	INT	Primary Key	Unique ID for each food item
Name	VARCHAR(100)	NOT NULL	Food item name
Price	DECIMAL(6,2)	NOT NULL	Price of the item
Restaurant_id	INT	Foreign Key	Refers to Restaurant.Restaurant_ID

4.Order:

Field Name	Data type	Constraint	Description
Order_id	INT	Primary key	Unique ID for each order
Customer_id	INT	Foreign key	Refers to Customer.Customer_ID
Restaurant_id	INT	Foreign key	Refers to Restaurant.Restaurant_ID
Order_date	DATE	NOT NULL	Date of the order
Status	VARCHAR(50)	DEFAULT 'Pending'	Order status

5. Order_details:

Field name	Data type	Constraint	Description
Order Detail_id	INT	Primary key	Unique ID for each entry
Order_id	INT	Foreign key	Refers to Order.Order_ID
Item_id	INT	Foreign key	Refers to Food_Item.Item_ID
Quantity	INT	NOT NULL	Number of units ordered
Total_price	DECIMAL(8,2)	NOT NULL	Subtotal for item

6. Payment:

Field name	Data type	Constraint	Description
Payment_id	INT	Primary key	Unique ID for each payment
Order_id	INT	Foreign key	Refers to Order.Order_ID
Customer_id	INT	Primary key	Refers to Customer.Customer_ID
Amount	Decimal(8,2)	NOT NULL	Total amount paid
Payment_method	VARCHAR(50)	NOT NULL	e.g., UPI, Credit Card, Cash
Payment_status	VARCHAR(50)	DEFAULT ‘Paid’	Status of the payment
Payment_date	DATE	NOT NULL	Date of transaction

TABLE CREATION:

1. Customer table:

```
CREATE TABLE Customer (
Customer_ID INT PRIMARY KEY,
Name VARCHAR(100) NOT NULL,
Email VARCHAR(100) UNIQUE,
Phone VARCHAR(15) NOT NULL,
Address VARCHAR(255) NOT NULL
);
```

2. Restaurant table:

```
CREATE TABLE Restaurant (
Restaurant_ID INT PRIMARY KEY,
Name VARCHAR(100) NOT NULL,
Type VARCHAR(50) NOT NULL,
Location VARCHAR(150) NOT NULL
);
```

3. Food_Item Table:

```
CREATE TABLE Food_Item (
  Item_ID INT PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  Price DECIMAL(6,2) NOT NULL,
  Restaurant_ID INT,
  FOREIGN KEY (Restaurant_ID) REFERENCES Restaurant(Restaurant_ID)
);
```

4. Order table:

```
CREATE TABLE Order_Table (
  Order_ID INT PRIMARY KEY,
  Customer_ID INT,
  Restaurant_ID INT,
  Order_Date DATE NOT NULL,
  Status VARCHAR(50) DEFAULT 'Pending',
  FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
  FOREIGN KEY (Restaurant_ID) REFERENCES Restaurant(Restaurant_ID)
);
```

5. Order_Details Table:

```
CREATE TABLE Order_Details (
  OrderDetail_ID INT PRIMARY KEY,
  Order_ID INT,
  Item_ID INT,
  Quantity INT NOT NULL,
  Total_Price DECIMAL(8,2) NOT NULL,
  FOREIGN KEY (Order_ID) REFERENCES Order_Table(Order_ID),
  FOREIGN KEY (Item_ID) REFERENCES Food_Item(Item_ID)
);
```

6. Payment table:

```
CREATE TABLE Payment (
  Payment_ID INT PRIMARY KEY,
  Order_ID INT,
  Customer_ID INT,
  Amount DECIMAL(8,2) NOT NULL,
  Payment_Method VARCHAR(50) NOT NULL,
  Payment_Status VARCHAR(50) DEFAULT 'Paid',
    Payment_Date DATE NOT NULL,
    FOREIGN KEY (Order_ID) REFERENCES Order_Table(Order_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
```

Each of these tables works together to make the system function efficiently, with clear relationships between customers, restaurants, orders, deliveries, and payments.

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

SQL QUERIES WITH OUTPUT:

1. Insert sample data into the Customer table:

```
INSERT INTO Customer VALUES (101, 'Anjali Sharma', 'anjali@gmail.com',
'9876543210', 'Delhi');
```

OUTPUT:

Customer_id	Name	Email	Phone	Address
101	Anjali Sharma	anjali@gmail.com	9876543210	Delhi

2. Insert sample data into the Restaurant table:

```
INSERT INTO Restaurant VALUES (1, 'Pizza Palace', 'Italian', 'Mumbai');
```

OUTPUT:

Restaurant_id	Name	Type	Location
1	Pizza Palace	Italian	Mumbai

3. Insert sample data into the Food_Item table:

```
INSERT INTO Food_Item VALUES (301, 'Margherita Pizza', 299.00, 1);
```

OUTPUT:

Item_Id	Name	Price	Restaurant_ID
301	Margherita Pizza	299.0	1

4. Insert a record into the Order_Table:

```
INSERT INTO Order_Table VALUES (501, 101, 1, '2025-04-10', 'Pending');
```

OUTPUT:

Order_id	Customer_id	Restaurant_id	Order_Date	Status
501	101	1	2025-04-10	Delivered

5. Insert data into Order_Details:

```
INSERT INTO Order_Details VALUES (701, 501, 301, 2, 598.00);
```

OUTPUT:

OrderDetail_id	Order_Id	Item_id	Quantity	Total_Price
701	501	301	2	598.0

6. Insert data into Payment:

```
INSERT INTO Payment VALUES (901, 501, 101, 598.00, 'UPI', 'Paid', '2025-04-10');
```

OUTPUT:

Payment_id	Order_id	Customer_id	Amount	Payment_method
901	501	101	598.0	UPI

Selection Queries:

7. Display all customers from the Customer table:

```
SELECT * FROM Customer;
```

OUTPUT:

Customer_id	Name	Email	Phone	Address
101	Anjali Sharma	anjali@gmail.com	9876543210	Delhi

8. Display orders placed by a specific customer (e.g., Anjali):

```
SELECT Order_ID, Order_Date, Status FROM Order_Table
WHERE Customer_ID = 101;
```

OUTPUT:

Order_id	Order_date	Status
501	2025-04-10	Delivered

9. Display all food items offered by 'Pizza Palace':

```
SELECT F.Name, F.Price FROM Food_Item F
JOIN Restaurant R ON F.Restaurant_ID = R.Restaurant_ID
WHERE R.Name = 'Pizza Palace';
```

OUTPUT:

Name	Price
Margherita Pizza	299.0

Update & Delete Queries:

10. Update the order status to 'Delivered':

```
UPDATE Order_Table SET Status = 'Delivered' WHERE Order_ID = 501;
```

OUTPUT:

```
Order status updated successfully.
```

11. Delete a food item (Example: Item_ID = 301):

```
DELETE FROM Food_Item WHERE Item_ID = 301;
```

OUTPUT:

```
1 row deleted.
```

Join Queries:

12. Show order details with customer name and total amount:

```
SELECT C.Name, O.Order_ID, O.Order_Date, P.Amount
FROM Customer C
JOIN Order_Table O ON C.Customer_ID = O.Customer_ID
JOIN Payment P ON O.Order_ID = P.Order_ID;
```

OUTPUT:

Name	Order_id	Order_date	Amount
Anjali Sharma	501	2025-04-10	598.0

13. List all food items ordered by a customer with quantity:

```
SELECT C.Name, FI.Name AS Food_Item, OD.Quantity
FROM Order_Details OD
```

```
JOIN Food_Item FI ON OD.Item_ID = FI.Item_ID
JOIN Order_Table O ON OD.Order_ID = O.Order_ID
JOIN Customer C ON O.Customer_ID = C.Customer_ID;
```

OUTPUT:

Name	Food_item	Quantity
Anjali Sharma	Margherita Pizza	2

Aggregate Functions:

14. Total sales amount collected from all payments:

```
SELECT SUM(Amount) AS Total_Sales FROM Payment;
```

OUTPUT:

```
TOTAL_SALES
598.0
```

15. Count total orders placed by each customer:

```
SELECT C.Name, COUNT(O.Order_ID) AS Total_Orders
FROM Customer C
JOIN Order_Table O ON C.Customer_ID = O.Customer_ID
GROUP BY C.Name;
```

OUTPUT:

Name	Total_orders
Anjali Sharma	1

SUMMARY:

This project aimed to design and implement a relational database system for an Online Food Ordering and Delivery Platform using DBMS concepts. The system is intended to simplify the food ordering process for customers while also providing smooth management for restaurants and delivery operations.

We began by identifying key entities such as Customer, Restaurant, Food Item, Order, Order Details, and Payment. An ER diagram was designed to visualize the relationships between these entities, and a proper table relation and tabular format were created to define the structure of each table, including primary and foreign key constraints.

Next, using SQL, we created the database tables and populated them with sample data. A set of 15 SQL queries was implemented to demonstrate core database operations like data insertion, selection, updating, deletion, joins, and aggregation. These queries help perform tasks such as viewing customer orders, calculating total sales, and managing payment statuses.

The project successfully demonstrates how a real-world food ordering system can be managed efficiently using a Relational Database Management System (RDBMS). The use of SQL provides flexibility to handle large amounts of data accurately and quickly. Screenshots of query outputs have been provided for better understanding and verification.

Overall, this project gave practical experience which are essential skills in software development and database management.

CONCLUSION:

The Online Food Ordering and Delivery System project effectively demonstrates the importance and application of database systems in real-world scenarios. Through this project, we designed a fully functional relational database using SQL that can manage customer records, restaurant menus, order details, and payment transactions efficiently.

By implementing multiple tables and relationships, and executing various SQL queries, we showcased the key operations like data insertion, retrieval, updating, deletion, and aggregation. The ER diagram helped us visualize the structure of the system, while normalization ensured data integrity and reduced redundancy.

This project enhanced our understanding of database management concepts such as schema design, relationships, keys, and query optimization. It also provided hands-on experience in how backend systems work in platforms like Swiggy or Zomato.

In conclusion, this project has been a valuable learning experience and reflects how structured data handling can improve the performance and scalability of real-life applications in the food tech industry.