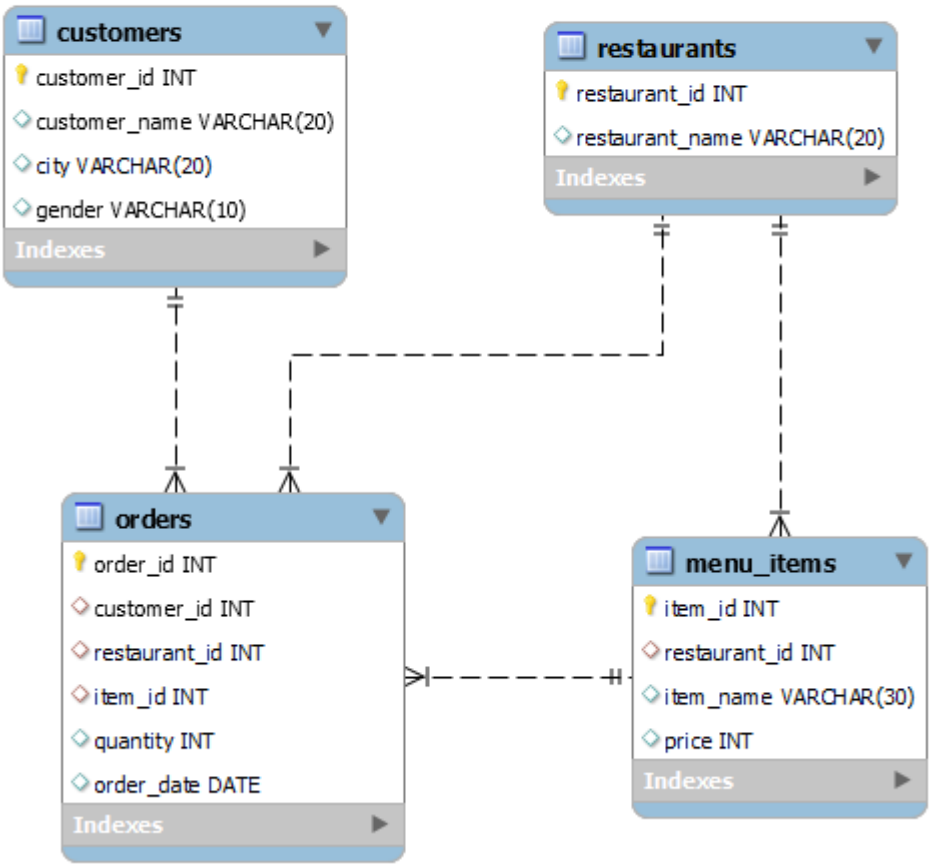
Online Food Ordering Management System using SQL

Online Food Ordering Management System, implemented using SQL, primarily focuses on tracking customer orders and managing data for customers, restaurants, menu items, and orders.

Designed and implemented a SQL Database to manage this information using SQL queries, ensuring data integrity through foreign keys and constraints.

Performed data analysis on online food ordering data using various SQL queries, joins, window functions, views, and stored procedures to extract actionable insights.

Schema Diagram:



SQL Queries:

```sql
-- Count of Customers per City
SELECT city, COUNT(*) AS count_of_customers
FROM Customers
GROUP BY city
ORDER BY count_of_customers DESC;
```

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

city	count_of_customers
Chennai	13
Bangalore	12
Mumbai	12
Delhi	11
Surat	11
Jaipur	10
Ahmedabad	9
Kolkata	8
Hyderabad	7
Pune	7

```sql
-- Count of Male and Female Customers
SELECT gender, COUNT(*) AS count_of_customers
FROM Customers
GROUP BY gender;
```

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

gender	count_of_customers
Male	60
Female	40

```sql
-- Name and price of all food items costing more than Rs. 200 (Premium Category Foods)
SELECT item_name, price
FROM Menu_Items
WHERE price > 200;
```

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

item_name	price
Chicken Biryani	230
Fish Curry	210
Butter Chicken	225
Chicken 65	205

```sql
-- Top 3 Costliest Food Items (Premium Cost Food Items)
SELECT item_name, price
FROM Menu_Items
ORDER BY price DESC
LIMIT 3;
```

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

item_name	price
Chicken Biryani	230
Butter Chicken	225
Fish Curry	210

```sql
-- Top 3 Cheapest Food Items (Budget Friendly Food Items)
SELECT item_name, price
FROM Menu_Items
ORDER BY price ASC
LIMIT 3;
```

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

item_name	price
Gulab Jamun	60
Samosa Chaat	65
Rasgulla	70

```sql
-- Show all orders along with the restaurant name from which they were placed
SELECT o.order_id, r.restaurant_name
FROM Orders AS o
INNER JOIN Restaurants AS r
ON o.restaurant_id = r.restaurant_id;
```

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

order_id	restaurant_name
2	Urban Corner
13	Urban Corner
23	Urban Corner
28	Urban Corner
39	Urban Corner
46	Urban Corner
49	Urban Corner
58	Urban Corner
60	Urban Corner
61	Urban Corner
88	Urban Corner
89	Urban Corner
97	Urban Corner

```sql
-- Show customer names and order dates for orders placed in January 2025
SELECT c.customer_name, o.order_date, o.order_id
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
WHERE o.order_date BETWEEN '2025-01-01' AND '2025-01-31'
ORDER BY o.order_date;
```

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

customer_name	order_date	order_id
Bhairav	2025-01-01	1224
Brinda	2025-01-02	1225
Binaisha	2025-01-02	1226
Chirag	2025-01-03	1227
Aarna	2025-01-03	1228
Ditya	2025-01-03	1229
Advika	2025-01-07	1230
Dev	2025-01-08	1231
Divya	2025-01-08	1232
Brij	2025-01-09	1233
Aarav	2025-01-10	1234
Chetana	2025-01-11	1235

```sql
-- Count how many orders each customers has placed
SELECT c.customer_id, c.customer_name, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.customer_name, c.customer_id;
```

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

customer_id	customer_name	total_orders
1	Aarav	10
2	Ayaan	13
3	Arjun	14
4	Advait	14
5	Arnav	13
6	Aakash	12
7	Atharv	17
8	Anay	17
9	Aditya	24
10	Ansh	13
11	Bhavak	16
12	Bodhi	21

```sql
-- Top 5 Customers based on order count
SELECT c.customer_id, c.customer_name, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.customer_name, c.customer_id
ORDER BY total_orders DESC
LIMIT 5;
```

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

customer_id	customer_name	total_orders
18	Binoy	27
53	Feroz	27
9	Aditya	24
69	Aisha	23
99	Damini	23

```sql
-- Show Total Revenue earned from each city (City-wise Total Revenue)
SELECT c.city, SUM(m.price*o.quantity) AS total_revenue
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
INNER JOIN Menu_Items AS m
ON o.item_id = m.item_id
GROUP BY c.city
ORDER BY total_revenue DESC;
```

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

city	total_revenue
Chennai	77950
Bangalore	69990
Mumbai	68865
Surat	66360
Jaipur	58415
Delhi	58410
Kolkata	52565
Ahmedabad	52020
Hyderabad	40370

```sql
-- Top 3 Cities by Revenue
SELECT c.city, SUM(m.price*o.quantity) AS total_revenue
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
INNER JOIN Menu_Items AS m
ON o.item_id = m.item_id
GROUP BY c.city
ORDER BY total_revenue DESC
LIMIT 3;
```

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

city	total_revenue
Chennai	77950
Bangalore	69990
Mumbai	68865

```sql
-- Bottom 3 Cities by Revenue
SELECT c.city, SUM(m.price*o.quantity) AS total_revenue
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
INNER JOIN Menu_Items AS m
ON o.item_id = m.item_id
GROUP BY c.city
ORDER BY total_revenue ASC
LIMIT 3;
```

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

city	total_revenue
Pune	38210
Hyderabad	40370
Ahmedabad	52020

```sql
-- Total Orders per City
SELECT c.city, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.city
ORDER BY total_orders DESC;
```

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

city	total_orders
Chennai	206
Mumbai	185
Surat	170
Bangalore	168
Delhi	148
Jaipur	145
Ahmedabad	138
Kolkata	121

```sql
-- Top 3 Cities by Total Orders
SELECT c.city, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.city
ORDER BY total_orders DESC
LIMIT 3;
```

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

city	total_orders
Chennai	206
Mumbai	185
Surat	170

```sql
-- Bottom 3 Cities by Total Orders
SELECT c.city, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.city
ORDER BY total_orders ASC
LIMIT 3;
```

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

city	total_orders
Hyderabad	109
Pune	110
Kolkata	121

```sql
-- Cities with more than 150 total orders
SELECT c.city, COUNT(o.order_id) AS total_orders
FROM Customers AS c
INNER JOIN Orders AS o
ON c.customer_id = o.customer_id
GROUP BY c.city
HAVING COUNT(o.order_id) > 150;
```

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

city	total_orders
Chennai	206
Bangalore	168
Mumbai	185
Surat	170

```sql
-- Revenue generated by each food item
SELECT m.item_name, SUM(m.price*o.quantity) AS total_revenue
FROM Menu_Items AS m
INNER JOIN Orders AS o
ON m.item_id = o.item_id
GROUP BY m.item_name
ORDER BY total_revenue DESC;
```

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

item_name	total_revenue
Chicken Biryani	48990
Butter Chicken	48600
Veg Pizza	43520
Chicken 65	43460
Veg Biryani	33945
Fish Curry	32970
Momos	29510
Hakka Noodles	26620

```sql
-- Top 3 Food Items by Revenue generated
SELECT m.item_name, SUM(m.price*o.quantity) AS total_revenue
FROM Menu_Items AS m
INNER JOIN Orders AS o
ON m.item_id = o.item_id
GROUP BY m.item_name
ORDER BY total_revenue DESC
LIMIT 3;
```

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

item_name	total_revenue
Chicken Biryani	48990
Butter Chicken	48600
Veg Pizza	43520

```sql
-- Bottom 3 Food Items by Revenue generated
SELECT m.item_name, SUM(m.price*o.quantity) AS total_revenue
FROM Menu_Items AS m
INNER JOIN Orders AS o
ON m.item_id = o.item_id
GROUP BY m.item_name
ORDER BY total_revenue ASC
LIMIT 3;
```

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

item_name	total_revenue
Samosa Chaat	12935
Rasgulla	14000
Gulab Jamun	14820

```sql
-- Food items that earned more than 30000 in Total Revenue
SELECT m.item_name, SUM(m.price*o.quantity) AS total_revenue
FROM Menu_Items AS m
INNER JOIN Orders AS o
ON m.item_id = o.item_id
GROUP BY m.item_name
HAVING total_revenue > 30000
ORDER BY total_revenue DESC;
```

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

item_name	total_revenue
Chicken Biryani	48990
Butter Chicken	48600
Veg Pizza	43520
Chicken 65	43460
Veg Biryani	33945
Fish Curry	32970

```sql
-- Show all menu items, price along with average price of all items
SELECT item_name, price, (SELECT AVG(price) FROM Menu_Items) AS avg_price
FROM Menu_Items
ORDER BY price DESC;
```

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

item_name	price	avg_price
Chicken Biryani	230	130.2381
Butter Chicken	225	130.2381
Fish Curry	210	130.2381
Chicken 65	205	130.2381
Veg Pizza	160	130.2381
Veg Biryani	155	130.2381
Paneer Butter Masala	140	130.2381
Fried Rice	135	130.2381

```sql
-- Show all menu items, price that are greater than average price of all items
SELECT item_name, price
FROM Menu_Items
WHERE price > (SELECT AVG(price) FROM Menu_Items);
```

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

item_name	price
Veg Pizza	160
Chicken Biryani	230
Fish Curry	210
Butter Chicken	225
Chicken 65	205
Paneer Butter Masala	140
Fried Rice	135
Veg Biryani	155

```sql
-- Monthly Order Trends
SELECT MONTH(order_date) AS month_number, MONTHNAME(order_date) AS order_month, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY MONTH(order_date), MONTHNAME(order_date)
ORDER BY month_number;
```

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

month_number	order_month	total_orders
1	January	145
2	February	137
3	March	143
4	April	142
5	May	152
6	June	147
7	July	126
8	August	107

```sql
-- Top 3 Months based on Total Orders
SELECT MONTH(order_date) AS month_number, MONTHNAME(order_date) AS order_month, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY MONTH(order_date), MONTHNAME(order_date)
ORDER BY total_orders DESC
LIMIT 3;
```

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

month_number	order_month	total_orders
5	May	152
6	June	147
1	January	145

```sql
-- Bottom 3 Months based on Total Orders
SELECT MONTH(order_date) AS month_number, MONTHNAME(order_date) AS order_month, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY MONTH(order_date), MONTHNAME(order_date)
ORDER BY total_orders ASC
LIMIT 3;
```

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

month_number	order_month	total_orders
12	December	89
11	November	100
10	October	105

```sql
-- Count of Orders
CREATE VIEW `Count_of_Orders` AS
SELECT COUNT(order_id) AS count_of_orders
FROM Orders;

SELECT * FROM `Count_of_Orders`;
```

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

count_of_orders
1500

```sql
-- Ranking Food Items based on its price from highest to lowest
CREATE VIEW `Ranking_Food_Price` AS
SELECT item_name, price, DENSE_RANK() OVER (ORDER BY price DESC) AS ranking_food_price
FROM Menu_Items;

SELECT * FROM `Ranking_Food_Price`;
```

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

item_name	price	ranking_food_price
Chicken Biryani	230	1
Butter Chicken	225	2
Fish Curry	210	3
Chicken 65	205	4
Veg Pizza	160	5
Veg Biryani	155	6
Paneer Butter Masala	140	7
Fried Rice	135	8
Momos	130	9
Masala Dosa	130	9
Paneer Tikka	125	10
Kadai Paneer	120	11

```sql
-- Restaurants and Food Items available
DELIMITER $$

CREATE PROCEDURE Restaurant_Food_Items()
BEGIN
    SELECT r.restaurant_name, m.item_name
    FROM Restaurants AS r
    INNER JOIN Menu_Items AS m ON
    r.restaurant_id = m.restaurant_id;
END $$

DELIMITER ;

-- Customer Information
DELIMITER $$

CREATE PROCEDURE Customer_Info(IN a INT)
BEGIN
    SELECT * FROM Customers WHERE customer_id = a;
END $$

DELIMITER ;
```

```sql
-- Order Information
DELIMITER $$

CREATE PROCEDURE Order_Info(IN b INT)
BEGIN
    SELECT o.order_id, o.order_date, c.customer_id, c.customer_name, m.item_name, o.quantity, m.price*o.quantity AS total_amount
    FROM Customers AS c
    INNER JOIN Orders AS o
    ON c.customer_id = o.customer_id
    INNER JOIN Menu_Items AS m
    ON o.item_id = m.item_id
    WHERE o.order_id = b;
END $$

DELIMITER ;
```

```sql
-- Call by Procedures
CALL Restaurant_Food_Items()
```

Result Grid | Filter Rows: | Export:

restaurant_name	item_name
Urban Corner	Paneer Tikka
Urban Corner	Chicken 65
Urban Corner	Paneer Butter Masala
Urban Corner	Kadai Paneer
Happy Space	Gulab Jamun
Happy Space	Samosa Chaat
Happy Space	Momos
Happy Space	Pav Bhaji
Happy Space	Rasgulla
Spice Kitchen	Chicken Biryani
Spice Kitchen	Fish Curry
Tasty Treat	Dal Tadka

```sql
CALL Customer_Info(5)
```

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

customer_id	customer_name	city	gender
5	Arnav	Delhi	Male

```sql
CALL Order_Info(3)
```

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

order_id	order_date	customer_id	customer_name	item_name	quantity	total_amount
3	2023-01-02	53	Feroz	Fish Curry	4	840

Insights:

Male customers (60) are more when compared to Female customers (40).
Premium-cost food items include Chicken Biryani, Butter Chicken, and Fish Curry.
Budget-friendly food items include Rasgulla, Samosa Chaat, and Gulab Jamun.

The top three cities by revenue are Chennai, Bangalore, and Mumbai.
Ahmedabad, Hyderabad, and Pune are the bottom three cities in terms of revenue.
The cities with the highest total orders are Chennai, Mumbai, and Surat.
The cities with the lowest total orders are Kolkata, Pune, and Hyderabad.
Chicken Biryani, Butter Chicken, and Veg Pizza generated the most revenue.
Samosa Chaat, Rasgulla, and Gulab Jamun generated the least revenue.
May, June, and January are the top three months for total orders.
October, November, and December are the months with the lowest total orders.

Conclusion:
Since cities with the highest food ordering activity are active markets, focus on improving and expanding strategies and operations there.
To boost performance in markets with low food ordering activity, increase marketing efforts and offer more promotions. Implement city-wise campaigns and offerings to effectively drive revenue growth.
To counteract slower periods, increase promotions during the months with the lowest total number of orders.