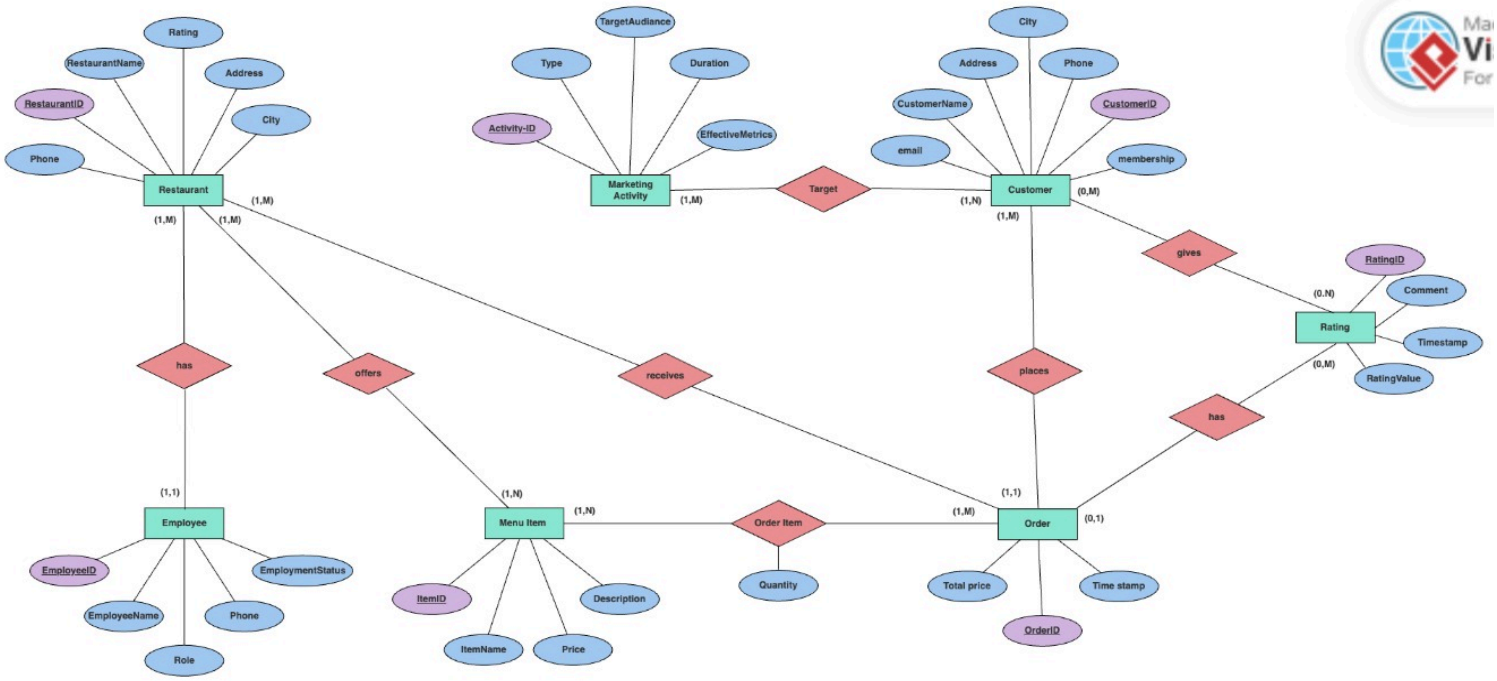
Online Food Delivery Startup

Conceptual Model:



Logical Model:

Restaurant: (<u>RestaurantID</u>, RestaurantName, Rating, Address, City, Phone)
Employee:(<u>EmployeeID</u>, EmployeeName, Role, Phone, EmploymentStatus, *RestaurantID*)
Customer:(<u>CustomerID</u>, Address, city, Phone, email, Membership)
MenuItem:(<u>ItemID</u>, ItemName, Price, Description, *RestaurantID*)
Order:(<u>OrderID</u>, TotalAmount, TimeStamp, *CustomerID,RestaurantID, RatingID*)
OrderItem:((<u>ItemID</u>,<u>OrderID</u>, Quantity)
MarketingActivity:(ActivityID, Type, TargetAudience, Duration, EffectiveMetrics)
Rating: (<u>RatingID</u>, Comment, Timestamp, RatingValue, *CustomerID*)
Customer-MarketingActivity:(<u>ActivityID</u>, <u>CustomerID)</u>

Corrections:

Order(1,1) -> customer
Rating(1,1) -> order

Menu Item (1,1)
Rating (1,1) -> Customer

Add an attribute called “available in stock” in Menu Item Entity.
Restaurant: (RestaurantID, RestaurantName, Rating, Address, City, Phone)

Employee:(EmployeeID, EmployeeName, Role, Phone, EmploymentStatus, *RestaurantID*)
Customer:(CustomerID,CustomerName, Address, city, Phone, email, Membership)
MenuItem:(ItemID, ItemName, Price, Description, *RestaurantID*)
Order:(OrderID, TotalAmount, TimeStamp, *CustomerID,RestaurantID, RatingID*)
OrderItem:((ItemID,OrderID, Quantity)
MarketingActivity:(ActivityID, Type, TargetAudience, Duration, EffectiveMetrics)
Rating: (RatingID, Comment, Timestamp, RatingValue, *CustomerID*)
Customer-MarketingActivity:(ActivityID, CustomerID)

Query: Compare the performance of different menu items across different restaurants to identify potential menu items**.**

Importance: Streamlines menu management and optimises offerings

```
SELECT Distinct Restaurant.RestaurantName, MenuItem.ItemName,
AVG(Rating.RatingValue) AS AverageRating
FROM MenuItem, OrderItem, Orders, Restaurant, Rating
WHERE MenuItem.ItemID = OrderItem.ItemID
AND OrderItem.OrderID = Orders.OrderID
AND Orders.RestaurantID = Restaurant.RestaurantID
AND Orders.RatingID = Rating.RatingID
GROUP BY Restaurant.RestaurantName, MenuItem.ItemName
ORDER BY MenuItem.ItemName, Restaurant.RestaurantName, AverageRating DESC;
```

```sql
SELECT Distinct Restaurant.RestaurantName, MenuItem.ItemName, AVG(Rating.RatingValue) AS AverageRating
FROM MenuItem, OrderItem, Orders, Restaurant, Rating
WHERE MenuItem.ItemID = OrderItem.ItemID
AND OrderItem.OrderID = Orders.OrderID
AND Orders.RestaurantID = Restaurant.RestaurantID
AND Orders.RatingID = Rating.RatingID
GROUP BY Restaurant.RestaurantName, MenuItem.ItemName
ORDER BY  AverageRating DESC;
```

Script Output | Query Result

SQL | All Rows Fetched: 15 in 0.096 seconds

	RESTAURANTNAME	ITEMNAME	AVERAGERATING
1	PantherExpress_ORD	Cannoli	4.8
2	PantherExpress_ORD	Tiramisu	4.8
3	PantherExpress_Atl	Antipasto Platter	4.7
4	PantherExpress_Atl	Spaghetti Bolognese	4.7
5	PantherExpress_Atl	Margherita Pizza	4.7
6	PantherExpress_Atl	Eggplant Parmesan	4.7
7	PantherExpress_DFW	Vegetarian Lasagna	4
8	PantherExpress_DFW	Garlic Bread	4
9	PantherExpress_OAK	Lemon Sorbet	3
10	PantherExpress_OAK	Mushroom Risotto	3
11	PantherExpress_OAK	Pepperoni Calzone	3
12	PantherExpress_MIA	Caesar Salad	2
13	PantherExpress_NY	Chicken Alfredo	2
14	PantherExpress_MIA	Shrimp Scampi	2
15	PantherExpress_NY	Caprese Salad	2

```sql
SELECT EmployeeID, EmployeeName, Role, EmploymentStatus, COUNT(Orders.OrderID) AS TotalOrders
FROM Employee, Orders
WHERE Employee.EmployeeID = Orders.EmployeeID
GROUP BY Employee.EmployeeID, Employee.EmployeeName, Employee.Role, Employee.EmploymentStatus;
```

Queries:

Query: Retrieve the total revenue generated by each restaurant
Importance: This query helps to understand which restaurants are generating the most revenue, which is crucial for financial analysis and strategic planning.

```
SELECT RESTAURANT.RestaurantID, SUM(ORDERS.TotalAmount) AS TotalRevenue
FROM ORDERS, RESTAURANT
WHERE ORDERS.RestaurantID = RESTAURANT.RestaurantID
GROUP BY RESTAURANT.RestaurantID;
```

```
SELECT RESTAURANT.RestaurantID, SUM(ORDERS.TotalAmount) AS TotalRevenue
FROM ORDERS, RESTAURANT
WHERE ORDERS.RestaurantID = RESTAURANT.RestaurantID
GROUP BY RESTAURANT.RestaurantID;
```

Script Output | Query Result

SQL | All Rows Fetched: 6 in 0.022 seconds

	RESTAURANTID	TOTALREVENUE
1	1	89.75
2	6	22.5
3	2	69.25
4	4	66.25
5	5	62
6	3	54.75

Query: Find the average rating of menu items and list those below a certain threshold.
Importance: Ensures quality control by identifying menu items that might not be meeting customer satisfaction standards.

```
SELECT MenuItem.ItemID, MenuItem.ItemName, AVG(Rating.RatingValue) AS AverageRating
FROM MenuItem, OrderItem, Orders, Rating
WHERE MenuItem.ItemID = OrderItem.ItemID
AND OrderItem.OrderID = Orders.OrderID
AND Orders.RatingID = Rating.RatingID
GROUP BY MenuItem.ItemID, MenuItem.ItemName
HAVING AVG(Rating.RatingValue) < 2.5;
```

Worksheet | Query Builder

```
SELECT MenuItem.ItemID, MenuItem.ItemName, AVG(Rating.RatingValue) AS AverageRating
FROM MenuItem, OrderItem, Orders, Rating
WHERE MenuItem.ItemID = OrderItem.ItemID
AND OrderItem.OrderID = Orders.OrderID
AND Orders.RatingID = Rating.RatingID
GROUP BY MenuItem.ItemID, MenuItem.ItemName
HAVING AVG(Rating.RatingValue) < 2.5;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.028 seconds

	ITEMID	ITEMNAME	AVERAGERATING
1	11	Caesar Salad	2
2	3	Chicken Alfredo	2
3	4	Caprese Salad	2
4	12	Shrimp Scampi	2

Query: List the details of customers who placed more than a 2 within a week.
Importance: Useful for targeting loyal customers for special offers or loyalty programs.

```
SELECT DISTINCT Customer.customerid
FROM Customer , Orders
WHERE Customer.CustomerID = Orders.CustomerID
AND Orders.ORDERTIMESTAMP >= SYSDATE - INTERVAL '7' DAY
AND Orders.ORDERTIMESTAMP <= SYSDATE
GROUP BY Customer.CustomerID
HAVING COUNT(Orders.OrderID) > 2;
```

```
SELECT DISTINCT Customer.customerid
FROM Customer , Orders
WHERE Customer.CustomerID = Orders.CustomerID
AND Orders.ORDERTIMESTAMP >= SYSDATE - INTERVAL '7' DAY
AND Orders.ORDERTIMESTAMP <= SYSDATE
GROUP BY Customer.CustomerID
HAVING COUNT(Orders.OrderID) > 2;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.023 seconds

	CUSTOMERID
1	1
2	3

Query: Find the number of orders placed by members versus non-members.

Importance: Evaluate the effectiveness of membership programs in encouraging orders.

```
SELECT Membership, COUNT(ORDERS.OrderID) AS NumberOfOrders
FROM CUSTOMER, ORDERS
WHERE  CUSTOMER.CustomerID = ORDERS.CustomerID
GROUP BY  Membership;
```

Worksheet | Query Builder

```sql
SELECT Membership, COUNT(ORDERS.OrderID) AS NumberOfOrders
FROM CUSTOMER, ORDERS
WHERE  CUSTOMER.CustomerID = ORDERS.CustomerID
GROUP BY  Membership;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.028 seconds

	MEMBERSHIP	NUMBEROFORDERS
1	1	9
2	0	6

Query: Calculate the total number of orders received by each employee and role distinguishing between full-time and part-time staff.

Importance: Helps in staffing decisions and for hiring with informed decisions.

```sql
SELECT Employee.EmployeeID, Employee.EmployeeName, Employee.EmployeeRole,
    Employee.EmploymentStatus, COUNT(Orders.OrderID) AS TotalOrders
FROM Employee, Restaurant, Orders
WHERE Employee.RestaurantID = Restaurant.RestaurantID
AND Restaurant.RestaurantID = Orders.RestaurantID
GROUP BY Employee.EmployeeID, Employee.EmployeeName, Employee.EmployeeRole,
    Employee.EmploymentStatus;
```

```sql
SELECT Employee.EmployeeID, Employee.EmployeeName, Employee.EmployeeRole, Employee.EmploymentStatus, COUNT(Orders.OrderID) AS TotalOrders
FROM Employee, Restaurant, Orders
WHERE Employee.RestaurantID = Restaurant.RestaurantID
AND Restaurant.RestaurantID = Orders.RestaurantID
GROUP BY Employee.EmployeeID, Employee.EmployeeName, Employee.EmployeeRole, Employee.EmploymentStatus;
```

Script Output | Query Result

SQL | All Rows Fetched: 6 in 0.044 seconds

	EMPLOYEEID	EMPLOYEENAME	EMPLOYEEROLE	EMPLOYMENTSTATUS	TOTALORDERS
1	1	John Smith	Chef	Full-Time	3
2	2	Jane Doe	Waiter	Part-Time	3
3	3	Bob Johnson	Manager	Full-Time	2
4	4	Alice Williams	Server	Part-Time	3
5	5	Charlie Brown	Chef	Full-Time	3
6	6	Emily Davis	Waiter	Part-Time	1

Query: Show the effectiveness of marketing activities based on increased orders during the activity duration.

Importance: Measures the ROI of marketing campaigns and aids in future marketing strategy.

```
SELECT MarketingActivity.ActivityID, MarketingActivity.ActivityType AS ActivityType,
COUNT(Orders.OrderID) AS IncreasedOrders
FROM MarketingActivity, CustomerMarketingActivity, Orders
WHERE MarketingActivity.ActivityID = CustomerMarketingActivity.ActivityID
AND CustomerMarketingActivity.CustomerID = Orders.CustomerID
AND Orders.orderTimeStamp >= (SYSDATE - MarketingActivity.ActivityDuration)
GROUP BY MarketingActivity.ActivityID, MarketingActivity.ActivityType;
```

```
SELECT MarketingActivity.ActivityID, MarketingActivity.ActivityType AS ActivityType, COUNT(Orders.OrderID) AS IncreasedOrders
FROM MarketingActivity, CustomerMarketingActivity, Orders
WHERE MarketingActivity.ActivityID = CustomerMarketingActivity.ActivityID
AND CustomerMarketingActivity.CustomerID = Orders.CustomerID
AND Orders.orderTimeStamp >= (SYSDATE - MarketingActivity.ActivityDuration)
GROUP BY MarketingActivity.ActivityID, MarketingActivity.ActivityType;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.043 seconds

	ACTIVITYID	ACTIVITYTYPE	INCREASEDORDERS
1	2	Email Newsletter	3
2	8	Loyalty Program Launch	3
3	7	Mobile App Promotion	1
4	14	Customer Referral Program	1
5	10	Product Sampling	1
6	1	Social Media Campaign	3
7	4	Radio Commercial	2
8	13	Holiday Promotion	1
9	11	Partnership with Influencers	3
10	5	Online Contest	1

Query: Display all orders placed during a specific time period that have not been rated by customers.

Importance: Identifies opportunities for follow-up to improve customer engagement and encourage feedback.

```
SELECT OrderID
FROM Orders, Rating
WHERE Orders.RatingID = Rating.RatingID
AND Rating.RatingID IS NULL
AND Orders.OrderTimeStamp BETWEEN TO_DATE('24-FEB-13', 'YY-MON-DD') AND
    TO_DATE('24-FEB-134', 'YY-MON-DD');
```

```
SELECT OrderID
FROM Orders, Rating
WHERE Orders.RatingID = Rating.RatingID
AND Rating.RatingID IS NULL
AND Orders.OrderTimeStamp BETWEEN TO_DATE('24-FEB-13', 'YY-MON-DD') AND TO_DATE('24-FEB-134', 'YY-MON-DD');
```

ript Output | Query Result

SQL | All Rows Fetched: 0 in 0.027 seconds

ORDERID

Query: Identify the menu items frequently ordered together.
Importance: Provides insight into customer preferences, which can inform menu planning and promotional bundles.

```
SELECT OrderItem1.ItemID AS Item1, OrderItem2.ItemID AS Item2, COUNT(*) AS Frequency
FROM OrderItem OrderItem1, OrderItem OrderItem2
WHERE OrderItem1.OrderID = OrderItem2.OrderID
AND OrderItem1.ItemID <> OrderItem2.ItemID
GROUP BY OrderItem1.ItemID, OrderItem2.ItemID;
```

Query: Find customers who placed orders exceeding a certain amount in the past year and haven't ordered recently. Target them with special promotions to encourage repeat business.**
Importance: Increases customer engagement and retention

```
SELECT DISTINCT Orders.CustomerID
FROM Customer, Orders
WHERE Customer.CustomerID = Orders.CustomerID
AND Orders.OrderTimeStamp >= TO_DATE('13-FEB-23 12:45:00', 'DD-MON-YY HH24:MI:SS')
AND Orders.OrderTimeStamp < TO_DATE('15-FEB-24 12:45:00', 'DD-MON-YY HH24:MI:SS')
AND Orders.TotalAmount > 20;
```

```sql
SELECT DISTINCT Orders.CustomerID
FROM Customer, Orders
WHERE Customer.CustomerID = Orders.CustomerID
AND Orders.OrderTimeStamp >= TO_DATE('13-FEB-23 12:45:00', 'DD-MON-YY HH24:MI:SS')
AND Orders.OrderTimeStamp < TO_DATE('15-FEB-24 12:45:00', 'DD-MON-YY HH24:MI:SS')
AND Orders.TotalAmount > 20;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.023 seconds

	CUSTOMERID
1	1
2	4
3	8
4	3
5	9

Query: Calculate the average order value during peak hours (5 PM to 9 PM)

Importance: To understand sales performance during peak business hours and for staffing and inventory purposes.

SELECT RestaurantID, AVG(TotalAmount) AS AverageOrderValue
FROM Orders
WHERE EXTRACT(HOUR FROM OrderTimeStamp) BETWEEN 17 AND 21
GROUP BY RestaurantID;

```sql
SELECT RestaurantID, AVG(TotalAmount) AS AverageOrderValue
FROM Orders
WHERE EXTRACT(HOUR FROM OrderTimeStamp) BETWEEN 17 AND 21
GROUP BY RestaurantID;
```

cript Output | Query Result

SQL | All Rows Fetched: 3 in 0.026 seconds

	RESTAURANTID	AVERAGEORDERVALUE
1	6	22.5
2	2	25.25
3	4	25.875

Query: Trigger to prevent an employee from being entered into the database without a valid restaurant ID.

Importance: Maintains referential integrity between the **EMPLOYEE** and **RESTAURANT** tables.

```
CREATE TRIGGER CheckRestaurantID

BEFORE INSERT ON EMPLOYEE

FOR EACH ROW

BEGIN

SELECT RAISE(ABORT, 'Invalid RestaurantID')

 WHERE NEW.RestaurantID NOT IN (SELECT RestaurantID FROM RESTAURANT);
```

To invoke the set Trigger:

```
INSERT INTO EMPLOYEE (EmployeeName, Role, Phone, EmploymentStatus, RestaurantID)

VALUES ('John Doe', 'Waiter', '555-1234', 'Full-Time', 999);
```

If the restaurant table does not have a restaurant with ID 999 then it will generate an error and will not insert the data into the table

Trigger Query

```
CREATE OR REPLACE TRIGGER check_valid_activity_customer
BEFORE INSERT ON CustomerMarketingActivity
FOR EACH ROW
DECLARE
   activity_exist INT;
   customer_exist INT;
BEGIN
   -- Check if the ActivityID exists in the MarketingActivity table
   SELECT COUNT(*) INTO activity_exist FROM MarketingActivity WHERE ActivityID = :NEW.ActivityID;

   -- Check if the CustomerID exists in the Customer table
   SELECT COUNT(*) INTO customer_exist FROM Customer WHERE CustomerID = :NEW.CustomerID;

   -- If either the ActivityID or the CustomerID does not exist, raise an error
   IF activity_exist = 0 OR customer_exist = 0 THEN
```

```
    RAISE_APPLICATION_ERROR(-20001, 'Invalid ActivityID or CustomerID');
  END IF;
END;
/
```

Trigger Query

Query: Validate the existence of ActivityID and CustomerID before inserting a record into the CustomerMarketingActivity table.

Importance: Ensures data integrity by preventing the insertion of invalid ActivityID or CustomerID values, maintaining database accuracy, and reliability.

insert INTO customermarketingactivity (ActivityID, CustomerID) VALUES (3, 16)