ZOMATO SQL Interview Questions (Experience 2+ years) CTC= 19 LPA

1_ Write an SQL query to find the top 5 most-ordered dishes from a given restaurant.

Input Table: Orders

OrderID	DishName	RestaurantID	Quantity	OrderDate
1	Burger	R101	2	2025-01-01
2	Pasta	R101	1	2025-01-02
3	Burger	R101	1	2025-01-03
4	Pizza	R102	3	2025-01-01
5	Burger	R101	4	2025-01-04
6	Pasta	R101	2	2025-01-05

```
SELECT DishName, SUM(Quantity) AS TotalOrders
FROM Orders
WHERE RestaurantID = 'R101'
GROUP BY DishName
ORDER BY TotalOrders DESC
LIMIT 5;
```

Output Table:

DishName	TotalOrders
Burger	7
Pasta	3

---

2. Write a Query to Retrieve All Orders Placed in the Last 30 Days for a Specific User

Input Table: Orders

OrderID	UserID	DishName	OrderDate	Amount
1	U123	Burger	2025-01-01	200
2	U123	Pasta	2025-01-10	150
3	U124	Pizza	2024-12-20	300
4	U123	Salad	2024-12-30	100
5	U124	Burger	2025-01-15	250

```
SELECT *
FROM Orders
WHERE UserID = 'U123'
  AND OrderDate >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

Output Table:

OrderID	UserID	DishName	OrderDate	Amount
1	U123	Burger	2025-01-01	200

OrderID	UserID	DishName	OrderDate	Amount
2	U123	Pasta	2025-01-10	150

3.Write a Query to Find the Total Revenue Generated by All Restaurants in the Last Month

Input Table: Orders

OrderID	RestaurantID	Amount	OrderDate
1	R101	200	2025-01-01
2	R101	150	2025-01-10
3	R102	300	2024-12-15
4	R103	100	2024-12-20
5	R102	250	2024-12-30

```
SELECT SUM(Amount) AS TotalRevenue
FROM Orders
WHERE OrderDate >= DATE_FORMAT(CURDATE() - INTERVAL 1 MONTH, '%Y-%m-01')
  AND OrderDate < DATE_FORMAT(CURDATE(), '%Y-%m-01');
```

Output Table:

TotalRevenue
650

---

4. Write a Query to Retrieve the Top 10 Restaurants with the Highest Average Customer Rating

Input Table: Ratings

RestaurantID	UserID	Rating
R101	U123	4.5
R102	U124	3.8
R101	U125	4.7
R103	U126	4.0
R102	U123	4.2
R101	U127	4.8

```
SELECT RestaurantID, AVG(Rating) AS AverageRating
FROM Ratings
GROUP BY RestaurantID
ORDER BY AverageRating DESC
LIMIT 10;
```

Output Table:

RestaurantID	AverageRating
R101	4.67
R102	4.00
R103	4.00

---

5. Write a query to List All Customers Who Have Ordered at Least One Dish from More Than 3 Different Restaurants

Input Table: Orders

OrderID	UserID	RestaurantID	DishName	OrderDate
1	U123	R101	Burger	2025-01-01
2	U123	R102	Pasta	2025-01-10
3	U123	R103	Pizza	2024-12-15
4	U124	R102	Salad	2024-12-20
5	U123	R104	Burger	2025-01-05

```
SELECT UserID
FROM Orders
GROUP BY UserID
HAVING COUNT(DISTINCT RestaurantID) > 3;
```

Output Table:

UserID
U123

---

6. Write a Query to Identify Restaurants Where the Average Order Value Is Higher Than the Overall Average Order Value Across All Restaurants

Input Table: Orders

OrderID	RestaurantID	Amount	OrderDate
1	R101	200	2025-01-01
2	R101	150	2025-01-10
3	R102	300	2025-01-15

OrderID	RestaurantID	Amount	OrderDate
4	R103	100	2025-01-20
5	R102	250	2025-01-25

```
WITH AvgOrderValue AS (
   SELECT RestaurantID, AVG(Amount) AS AvgOrderValue
   FROM Orders
   GROUP BY RestaurantID
),
OverallAvg AS (
   SELECT AVG(Amount) AS OverallAverage
   FROM Orders
)
SELECT A.RestaurantID, A.AvgOrderValue
FROM AvgOrderValue A, OverallAvg O
WHERE A.AvgOrderValue > O.OverallAverage;
```

Output Table:

RestaurantID	AvgOrderValue
R102	275.00

---

7. Write a Query to Group Orders by Cuisine Type and Calculate the Total Revenue for Each Cuisine

Input Table: Orders

OrderID	Cuisine	Amount	OrderDate
1	Italian	200	2025-01-01
2	Italian	150	2025-01-10
3	Chinese	300	2025-01-15
4	Indian	100	2025-01-20
5	Chinese	250	2025-01-25

```
SELECT Cuisine, SUM(Amount) AS TotalRevenue
FROM Orders
GROUP BY Cuisine
ORDER BY TotalRevenue DESC;
```

Output Table:

Cuisine	TotalRevenue
Chinese	550
Italian	350
Indian	100

---

8. Write a Query to Find the Rank of Each Restaurant Based on Total Revenue Within Each City

Input Table: Orders

OrderID	RestaurantID	City	Amount	OrderDate
1	R101	New York	200	2025-01-01

OrderID	RestaurantID	City	Amount	OrderDate
2	R102	New York	300	2025-01-10
3	R103	London	150	2025-01-15
4	R104	London	250	2025-01-20
5	R105	New York	100	2025-01-25

```
SELECT RestaurantID, City, SUM(Amount) AS TotalRevenue,
    RANK() OVER (PARTITION BY City ORDER BY SUM(Amount) DESC) AS Rank
FROM Orders
GROUP BY RestaurantID, City;
```

Output Table:

RestaurantID	City	TotalRevenue	Rank
R102	New York	300	1
R101	New York	200	2
R105	New York	100	3
R104	London	250	1
R103	London	150	2

---

9. Categorize Restaurants into "High Revenue," "Medium Revenue," and "Low Revenue" Based on Their Monthly Sales

Input Table: Orders

OrderID	RestaurantID	Amount	OrderDate
1	R101	200	2025-01-01
2	R102	300	2025-01-10
3	R103	150	2025-01-15
4	R104	400	2025-01-20
5	R105	100	2025-01-25

```
WITH MonthlyRevenue AS (
   SELECT RestaurantID, SUM(Amount) AS TotalMonthlyRevenue
   FROM Orders
   WHERE OrderDate >= DATE_FORMAT(CURDATE() - INTERVAL 1 MONTH, '%Y-%m-01')
    AND OrderDate < DATE_FORMAT(CURDATE(), '%Y-%m-01')
   GROUP BY RestaurantID
)
SELECT RestaurantID, TotalMonthlyRevenue,
     CASE
       WHEN TotalMonthlyRevenue > 300 THEN 'High Revenue'
       WHEN TotalMonthlyRevenue BETWEEN 150 AND 300 THEN 'Medium Revenue'
       ELSE 'Low Revenue'
     END AS RevenueCategory
FROM MonthlyRevenue;
```

Output Table:

RestaurantID	TotalMonthlyRevenue	RevenueCategory
R104	400	High Revenue
R102	300	Medium Revenue
R101	200	Medium Revenue
R103	150	Medium Revenue
R105	100	Low Revenue

---

10. Write a Query to Find the Top 3 Dishes Sold for Each Restaurant in a Specific City

Input Table: Orders

OrderID	RestaurantID	City	DishName	Quantity	OrderDate
1	R101	New York	Burger	3	2025-01-01
2	R101	New York	Pizza	5	2025-01-10
3	R102	New York	Pasta	2	2025-01-15
4	R102	New York	Salad	4	2025-01-20
5	R103	London	Burger	1	2025-01-25

```
WITH RankedDishes AS (
  SELECT RestaurantID, City, DishName, SUM(Quantity) AS TotalQuantity,
      RANK() OVER (PARTITION BY RestaurantID, City ORDER BY SUM(Quantity) DESC) AS Rank
  FROM Orders
  WHERE City = 'New York'
  GROUP BY RestaurantID, City, DishName
```

)

SELECT RestaurantID, City, DishName, TotalQuantity

FROM RankedDishes

WHERE Rank <= 3;

Output Table:

RestaurantID	City	DishName	TotalQuantity
R101	New York	Pizza	5
R101	New York	Burger	3
R102	New York	Salad	4
R102	New York	Pasta	2

---

11. Use a CTE to Calculate the Monthly Active Users and Their Most Ordered Dish for the Past 6 Months

Input Table: Orders

OrderID	UserID	DishName	OrderDate
1	U101	Burger	2024-08-01
2	U102	Pizza	2024-08-15
3	U101	Burger	2024-09-05
4	U103	Salad	2024-09-20
5	U101	Burger	2024-10-10
6	U104	Pasta	2024-10-15

WITH RecentOrders AS (

```
    SELECT UserID, DishName, DATE_FORMAT(OrderDate, '%Y-%m') AS Month, COUNT(*) AS
OrderCount
    FROM Orders
    WHERE OrderDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
    GROUP BY UserID, DishName, Month
),
MostOrderedDish AS (
    SELECT UserID, Month, DishName, OrderCount,
        RANK() OVER (PARTITION BY UserID, Month ORDER BY OrderCount DESC) AS Rank
    FROM RecentOrders
)
SELECT UserID, Month, DishName AS MostOrderedDish
FROM MostOrderedDish
WHERE Rank = 1;
```

Output Table:

UserID	Month	MostOrderedDish
U101	2024-08	Burger
U102	2024-08	Pizza
U101	2024-09	Burger
U103	2024-09	Salad
U101	2024-10	Burger
U104	2024-10	Pasta

---

12. Write a query to Find All Users Who Have Referred Others (Directly or Indirectly) to Zomato's Referral Program

Input Table: Referrals

ReferrerID	ReferredID
U101	U102
U102	U103
U103	U104
U104	U105

```
WITH ReferralChain AS (
   SELECT ReferrerID, ReferredID
   FROM Referrals
   UNION ALL
   SELECT R.ReferrerID, R2.ReferredID
   FROM Referrals R
   JOIN ReferralChain R2 ON R.ReferredID = R2.ReferrerID
)
SELECT DISTINCT ReferrerID
FROM ReferralChain;
```

Output Table:

ReferrerID
U101
U102

ReferrerID
U103
U104

---

13. Explain How and Why You Would Use a Temporary Table to Calculate a Customer's Lifetime Value

Explanation: A temporary table is ideal for calculating customer lifetime value (CLV) because it allows you to store intermediate results, such as total revenue or total orders for each customer, without impacting the original database structure. This approach improves performance and simplifies complex queries by breaking them into smaller steps.

Steps:

1. **Create a Temporary Table:** Store aggregated data (e.g., total spending, number of orders per customer).

```
CREATE TEMPORARY TABLE TempCustomerRevenue AS

SELECT UserID, SUM(Amount) AS TotalRevenue

FROM Orders

GROUP BY UserID;
```

2. **Query the Temporary Table:** Calculate CLV by joining with other customer data.

```
SELECT C.UserID, C.TotalRevenue, L.LifetimeValue

FROM TempCustomerRevenue C

JOIN LoyaltyProgram L ON C.UserID = L.UserID;
```

3. **Drop the Temporary Table After Use:**

```
DROP TEMPORARY TABLE TempCustomerRevenue;
```

---

14. Write a Query to Retrieve the Top 5 Restaurants With the Fastest Average Delivery Time

Input Table: Orders

OrderID	RestaurantID	DeliveryTime	OrderDate
1	R101	30	2025-01-01
2	R102	25	2025-01-02
3	R103	40	2025-01-03
4	R104	20	2025-01-04
5	R105	15	2025-01-05

```
SELECT RestaurantID, AVG(DeliveryTime) AS AvgDeliveryTime
FROM Orders
GROUP BY RestaurantID
ORDER BY AvgDeliveryTime ASC
LIMIT 5;
```

Output Table:

RestaurantID	AvgDeliveryTime
R105	15
R104	20
R102	25
R101	30
R103	40

---

15. Suggest an Indexing Strategy for a Table With Millions of Orders to Optimize Query Performance

Table: Orders

OrderID UserID RestaurantID OrderDate Amount

Indexing Strategy:

1. **Primary Index:**
    - Use OrderID as the primary key to uniquely identify each order.

```
ALTER TABLE Orders ADD PRIMARY KEY (OrderID);
```

2. **Composite Index:**
    - Create a composite index on UserID and OrderDate to optimize queries filtering by users and date ranges.

```
CREATE INDEX idx_user_date ON Orders (UserID, OrderDate);
```

3. **Index for Aggregations:**
    - Add an index on RestaurantID to optimize revenue calculations and ranking queries.

```
CREATE INDEX idx_restaurant ON Orders (RestaurantID);
```

4. **Clustered Index:**
    - If frequent queries involve sorting by OrderDate, make it a clustered index.

```
ALTER TABLE Orders ORDER BY OrderDate;
```

5. **Consider Partitioning:**
    - For millions of rows, partition the table by date (e.g., monthly or yearly).

```
ALTER TABLE Orders PARTITION BY RANGE (YEAR(OrderDate)) (
  PARTITION p2023 VALUES LESS THAN (2024),
  PARTITION p2024 VALUES LESS THAN (2025)
);
```

This strategy balances read performance, write efficiency, and query execution speed.

16. Write a Query to Identify Customers Who Have Not Placed Any Orders in the Last 6 Months But Were Active in the Previous 6 Months

Input Table: Orders

OrderID	UserID	OrderDate
1	U101	2024-01-15
2	U102	2024-02-10
3	U103	2024-06-20
4	U101	2024-07-15
5	U102	2024-12-01

```
WITH LastSixMonths AS (
   SELECT UserID
   FROM Orders
   WHERE OrderDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
),
PreviousSixMonths AS (
   SELECT UserID
   FROM Orders
   WHERE OrderDate >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
    AND OrderDate < DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
)
SELECT DISTINCT UserID
FROM PreviousSixMonths
WHERE UserID NOT IN (SELECT UserID FROM LastSixMonths);
```

Output Table:

UserID
U103

---

17. Identify the Top 3 Cities With the Highest Percentage of Premium Customers

Input Table: Orders

OrderID	UserID	City	Amount
1	U101	Mumbai	1500
2	U102	Bangalore	800
3	U103	Delhi	1200
4	U101	Mumbai	2000
5	U104	Bangalore	1100

```
WITH CustomerSpending AS (
    SELECT UserID, City, AVG(Amount) AS AvgSpending
    FROM Orders
    GROUP BY UserID, City
),
PremiumCustomers AS (
    SELECT City, COUNT(UserID) AS PremiumCount
    FROM CustomerSpending
    WHERE AvgSpending > 1000
    GROUP BY City
```

```
),
CityCustomerCount AS (
   SELECT City, COUNT(DISTINCT UserID) AS TotalCustomers
   FROM Orders
   GROUP BY City
)
SELECT P.City,
     P.PremiumCount,
     CC.TotalCustomers,
     (P.PremiumCount * 100.0 / CC.TotalCustomers) AS PremiumPercentage
FROM PremiumCustomers P
JOIN CityCustomerCount CC ON P.City = CC.City
ORDER BY PremiumPercentage DESC
LIMIT 3;
```

Output Table:

City	PremiumCount	TotalCustomers	PremiumPercentage
Mumbai	1	1	100.00
Delhi	1	1	100.00
Bangalore	1	2	50.00

---

18. Detect Users Who Have Placed Multiple Orders Within the Same Minute From Different Locations

Input Table: Orders

OrderID	UserID	Location	OrderTimestamp
1	U101	Mumbai	2025-01-10 12:30:00
2	U101	Delhi	2025-01-10 12:30:30
3	U102	Bangalore	2025-01-10 12:45:00
4	U101	Mumbai	2025-01-10 12:30:45

```
SELECT UserID, GROUP_CONCAT(Location) AS Locations, COUNT(*) AS OrderCount
FROM Orders
WHERE OrderTimestamp >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY UserID, DATE_FORMAT(OrderTimestamp, '%Y-%m-%d %H:%i')
HAVING OrderCount > 1 AND COUNT(DISTINCT Location) > 1;
```

Output Table:

UserID	Locations	OrderCount
U101	Mumbai, Delhi	3

---

19. Calculate the Contribution of Each City to the Total Revenue for a Given Year

Input Table: Orders

OrderID	City	Amount	OrderDate
1	Mumbai	1500	2025-01-10
2	Bangalore	800	2025-02-20
3	Delhi	1200	2025-03-05

OrderID	City	Amount	OrderDate
4	Mumbai	2000	2025-04-15
5	Delhi	500	2025-05-10

```
WITH CityRevenue AS (
   SELECT City, SUM(Amount) AS TotalCityRevenue
   FROM Orders
   WHERE YEAR(OrderDate) = 2025
   GROUP BY City
),
TotalRevenue AS (
   SELECT SUM(Amount) AS TotalRevenue
   FROM Orders
   WHERE YEAR(OrderDate) = 2025
)
SELECT CR.City,
     CR.TotalCityRevenue,
     (CR.TotalCityRevenue * 100.0 / TR.TotalRevenue) AS ContributionPercentage
FROM CityRevenue CR, TotalRevenue TR
ORDER BY ContributionPercentage DESC;
```

Output Table:

City	TotalCityRevenue	ContributionPercentage
Mumbai	3500	53.85

City	TotalCityRevenue	ContributionPercentage
Delhi	1700	26.15
Bangalore	800	20.00

---

20. Retrieve the Top 5 Dishes That Experienced the Highest Surge in Orders During a Flash Sale

Input Table: Orders

OrderID	DishName	SaleFlag	OrderDate
1	Burger	Yes	2025-01-10
2	Pizza	No	2025-01-11
3	Burger	Yes	2025-01-12
4	Salad	No	2025-01-10
5	Burger	Yes	2025-01-13

SQL Query:

```
sql

CopyEdit

WITH AverageOrders AS (

   SELECT DishName, COUNT(*) / COUNT(DISTINCT SaleFlag) AS AvgDailyOrders

   FROM Orders

   WHERE SaleFlag = 'No'

   GROUP BY DishName

),

FlashSaleOrders AS (
```

```
    SELECT DishName, COUNT(*) AS FlashOrders
    FROM Orders
    WHERE SaleFlag = 'Yes'
    GROUP BY DishName
)
SELECT F.DishName, F.FlashOrders, A.AvgDailyOrders,
       (F.FlashOrders - A.AvgDailyOrders) AS OrderSurge
FROM FlashSaleOrders F
JOIN AverageOrders A ON F.DishName = A.DishName
ORDER BY OrderSurge DESC
LIMIT 5;
```

Output Table:

DishName	FlashOrders	AvgDailyOrders	OrderSurge
Burger	3	1	2