Important SQL Interview Questions - D.A

by: Akarshan Kapoor

STUDENT GRADE DISTRIBUTION

Q. For the student database given alongside, arrive at the number of students who come under each of the grade category.

Student	Marks	
Student 1	62	
Student 2	92	
Student 3	52	
Student 4	60	
Student 5	81	
Student 6	66	
Student 7	63	
Student 8	100	
Student 9	46	
Student 10	87	
Student 11	93	

Min	Max	Grade	Frequency
90	100	A	
80	89	B	
51	79	C	
41	50	D	
0	40	F	

```
SELECT
    CASE
        WHEN Marks BETWEEN 90 AND 100 THEN 'A'
        WHEN Marks BETWEEN 80 AND 89 THEN 'B'
        WHEN Marks BETWEEN 51 AND 79 THEN 'C'
        WHEN Marks BETWEEN 41 AND 50 THEN 'D'
        ELSE 'F'
    END AS Grade,
    COUNT(*) AS Frequency
FROM student_data
GROUP BY Grade
ORDER BY Grade;
```

QUESTION 2

BRANCH DETAILS

Based on the Table below.

Q1. What is the total deposit amount for each branch, broken down by account type?
Q2. What types of accounts do tellers open most often?
Q3. Which branch opens the most accounts for new customers?

Date	Amount	AcctType	OpenedBy	Branch	Customer
Sep-01	340	Checking	New Accts	Central	Existing
Sep-01	15,759	CD	Teller	Westside	Existing
Sep-01	15,276	CD	New Accts	North County	Existing
Sep-01	12,000	CD	New Accts	Westside	Existing
Sep-01	5,000	CD	New Accts	North County	Existing
Sep-01	7,000	Savings	New Accts	North County	New
Sep-01	5,000	Savings	New Accts	Westside	Existing
Sep-01	4,623	Savings	New Accts	North County	Existing
Sep-01	5,879	Checking	New Accts	Central	Existing
Sep-01	3,171	Checking	New Accts	Westside	Existing
Sep-01	4,000	Savings	New Accts	Central	Existing
Sep-01	5,000	IRA	New Accts	Central	Existing
Sep-01	16,000	CD	New Accts	Central	New

Q1: Total deposit amount for each branch, broken down by account type

```
SELECT
    Branch,
    AcctType,
    SUM(Amount) AS TotalDeposit
FROM branch_details
GROUP BY Branch, AcctType
ORDER BY Branch, AcctType;
```

Q2: Types of accounts tellers open most often

```
SELECT
    AcctType,
    COUNT(*) AS OpenedCount
FROM branch_details
WHERE OpenedBy = 'Teller'
GROUP BY AcctType
ORDER BY OpenedCount DESC
LIMIT 1;
```

Q3: Branch that opens the most accounts for new customers

```
SELECT
    Branch,
    COUNT(*) AS NewAccountsCount
FROM branch_details
WHERE Customer = 'New'
GROUP BY Branch
ORDER BY NewAccountsCount DESC
LIMIT 1;
```

There are two tables : users and order. Users table contains user information such as name, age, state. Order table contains order information of users.

Users Table

User ID	User Name	Age	State
1	Naman	29	Haryana
2	Ayush	24	Gujarat
3	Kritika	21	Haryana
4	Geetika	23	Haryana
5	Manoj	34	Haryana
6	Mayuri	32	Delhi

Order Table

User ID	Order ID	Product	Order Date	Amount
1	a	HA	10/20/2019	500
1	b	PA	10/25/2019	1000
2	a	HA	11/1/2019	1500
3	a	HB	11/3/2019	2500
3	b	PB	11/28/2019	300
3	c	PA	12/1/2019	100
5	a	HA	10/5/2019	200
6	a	HA	9/1/2019	350
6	c	PB	11/4/2019	600

Question 1	Write query to get user name, count of products purchased in Oct'19 and Nov'19
Question 2	Write query to get User names who purchased in Nov'19 and are from Haryana?
Question 3	Write query to get User name & his/her latest order information
Question 4	Write query to get top 2 user id and name based on the total transaction value (amount field) for each month

Query for Question 1:

```
SELECT
    u.UserName,
    COUNT(o.Product) AS ProductCount
FROM
    Users u
JOIN
    Orders o ON u.UserID = o.UserID
WHERE
    MONTH(o.OrderDate) IN (10, 11) AND YEAR(o.OrderDate) = 2019
GROUP BY
    u.UserName;
```

Query for Question 2:

```
SELECT
    u.UserName
FROM
```

```
    Users u
JOIN
    Orders o ON u.UserID = o.UserID
WHERE
    MONTH(o.OrderDate) = 11 AND YEAR(o.OrderDate) = 2019
    AND u.State = 'Haryana';
```

Query for Question 3:

```
SELECT
    u.UserName,
    o.*
FROM
    Users u
JOIN
    Orders o ON u.UserID = o.UserID
WHERE
    o.OrderDate = (
        SELECT MAX(o2.OrderDate)
        FROM Orders o2
        WHERE o2.UserID = o.UserID
    );
```

Query for Question 4:

```
SELECT
    o.UserID,
    u.UserName,
    SUM(o.Amount) AS TotalTransaction,
    MONTH(o.OrderDate) AS OrderMonth
FROM
    Orders o
JOIN
    Users u ON o.UserID = u.UserID
GROUP BY
    o.UserID, u.UserName, MONTH(o.OrderDate)
ORDER BY
    OrderMonth, TotalTransaction DESC
LIMIT 2;
```

Employee Table

Employee ID	User Name	Salary
1	Amrita	5000
2	Ayushi	4000
3	Kapil	6000
4	Sayan	10000
5	Roshan	8000

Question1 - SQL Query to find the employee name with the second highest salary of Employee (without the use of rank, row_number function)

```
SELECT
    UserName
FROM
    Employee
WHERE
    Salary = (SELECT MAX(Salary)
              FROM Employee
              WHERE Salary < (SELECT MAX(Salary) FROM Employee));
```

---

1. Get the number of unique count orders, customers and total booking amount for all Categories , and Cities , monthwise. Sort them by month

Table - Transactions

Order_ID	Voucher_ID	Category	Booking_Amt	Date
101	NKAZA12	FNB	1000	2020-04-01
101	RCAOS83	FNB	200	2020-04-01
108	ASEOI43	BNS	700	2018-06-12
...	...	...	...	...

Table - City

OrderID	CustomerID	City
101	10001959	Delhi
102	10034848	Chennai
124	10001959	Bangalore
...	...	...

Table - Customer

CustomerID	Age
10001959	28
10034848	40
10025872	62
...	...

```
SELECT
    DATE_FORMAT(t.Date, '%Y-%m') AS month,  -- Extract month and year in 'YYYY-MM' format
    t.Category,
    c.City,
    COUNT(DISTINCT t.Order_ID) AS unique_orders,  -- Count distinct orders
    COUNT(DISTINCT c.CustomerID) AS unique_customers,  -- Count distinct customers
    SUM(t.Booking_Amt) AS total_booking_amt  -- Sum up the booking amount
FROM
    Transactions t
JOIN
    City c ON t.Order_ID = c.OrderID  -- Join Transactions and City tables
JOIN
    Customer cu ON c.CustomerID = cu.CustomerID  -- Join City and Customer tables
GROUP BY
    month, t.Category, c.City  -- Group by month, category, and city
ORDER BY
    month ASC;  -- Sort by month
```

Q1- Query to give Date wise total orders, and revenue

Q2 - Query to give Date wise total orders, and revenue - only for last 7 days

Table A

Datetime	Order ID	category	revenue
23 Feb 2021 6:53:20	101	Food	100
23 Feb 2021 6:46:04	102	Salon	117
...	...	...	...

Output

Date	Orders	Revenue
23 Feb	200,000	20,000,000
24 Feb	500,000	50,000,000
25 Feb	380,000	38,000,000

Q1:

```
SELECT
    DATE(Datetime) AS Date,  -- Extract the date part
    COUNT(Order_ID) AS Orders,  -- Total number of orders
    SUM(revenue) AS Revenue  -- Total revenue
FROM
    TableA
GROUP BY
    DATE(Datetime)  -- Group by the date
ORDER BY
    Date ASC;  -- Sort by date
```

Q2.

```
SELECT
    TxnDate,
    COUNT(*) AS TotalTransactions,
    SUM(CASE WHEN Status = 'Success' THEN 1 ELSE 0 END) AS SuccessfulTransactions,
    ROUND(SUM(CASE WHEN Status = 'Success' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS
SuccessRate
FROM
    txn
GROUP BY
    TxnDate
ORDER BY
    TxnDate;
```

Write a query to give monthly count of cases in 2021

Also give a cumulative count of cases for the year

Table - DAILY_CASES

record_date	cases_count
2021-01-01	35,984
2021-01-16	44,614
2021-01-31	45,285
...	...
...	...

Sample Output

Month	Monthly Cases Count	Cumulative cases count
Jan '21	200,000	200,000
Feb '21	500,000	700,000
Mar'21	650,000	1,350,000
...	...	...

```
SELECT
    DATE_FORMAT(record_date, '%b \'%y') AS Month,  -- Format as 'Jan '21', 'Feb '21', etc.
    SUM(cases_count) AS Monthly_Cases_Count,  -- Total cases for the month
    SUM(SUM(cases_count)) OVER (ORDER BY DATE_FORMAT(record_date, '%Y-%m')) AS
Cumulative_Cases_Count  -- Cumulative sum of cases
FROM
    DAILY_CASES
WHERE
    YEAR(record_date) = 2021  -- Filter records for the year 2021
GROUP BY
    DATE_FORMAT(record_date, '%Y-%m')  -- Group by year and month
ORDER BY
    DATE_FORMAT(record_date, '%Y-%m');  -- Order by year and month
```

Calculate Daily Txns, and Success rate of txns

Table name - txn

Txn ID	Txn Date	Status
1627191	2021/12/20	InProcess
7281910	2021/12/21	Refund
1738101	2021/12/22	Success
...	...	...

```
SELECT
    TxnDate,
    COUNT(*) AS TotalTransactions,
    SUM(CASE WHEN Status = 'Success' THEN 1 ELSE 0 END) AS SuccessfulTransactions,
    ROUND(SUM(CASE WHEN Status = 'Success' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS
SuccessRate
FROM
    txn
GROUP BY
    TxnDate
ORDER BY
    TxnDate;
```

The below table shows a user account passbook where closing balance against every user id is stored whenever any txn activity (credit or debit) happens in user account

You are required to give the net credit or debit that against a user ID on active txn days

Table - User Passbook

User ID	Date	Closing balance
123	Jan 21, 2022	50,750.0
123	Jan 25, 2022	45,000.0
123	Feb 1 2022	47,000.0
...	...	...

50,750.0	
45,000.0	

Output

User ID	Date	Txn amount	Net Acitivity
123	Jan 25, 2022	-5,750	Debit
123	Feb 1, 2022	2,000	Credit

```
WITH RankedPassbook AS (
    SELECT
        UserID,
        Date,
        ClosingBalance,
        LAG(ClosingBalance) OVER (PARTITION BY UserID ORDER BY Date) AS PreviousBalance
    FROM
        UserPassbook
)
SELECT
    UserID,
    Date,
```

```
    (ClosingBalance - PreviousBalance) AS TxnAmount,
    CASE
        WHEN (ClosingBalance - PreviousBalance) < 0 THEN 'Debit'
        ELSE 'Credit'
    END AS NetActivity
FROM
    RankedPassbook
WHERE
    PreviousBalance IS NOT NULL
ORDER BY
    UserID, Date;
```

1.There is a credit card fraud happening how will you identify this ?
2.There is a new TV which is being launched and needs to be advertised on the platform how will you calculate the success rate of the advertisement banner.

1. Identifying Credit Card Fraud

To identify credit card fraud, you need to analyze transaction data and look for anomalies or patterns commonly associated with fraudulent activities. Here are some techniques and steps:

a. Identify Fraud Indicators

- **Unusual Transaction Amounts**: Transactions significantly higher or lower than a user's normal spending behavior.
- **Geographic Anomalies**: Transactions occurring in different geographic locations within a short timeframe.
- **Frequent Declines**: A high number of declined transactions in a short period.
- **Odd Hours**: Transactions at unusual times, such as midnight or very early morning.
- **Unrecognized Devices**: Transactions made from a new or unrecognized device.

b. Machine Learning Approach

Use supervised or unsupervised machine learning models:

- **Supervised Learning**:
    - Train a model using historical data labeled as "fraudulent" or "legitimate."
    - Features include transaction amount, location, device type, time, etc.
    - Algorithms: Logistic Regression, Random Forest, Gradient Boosting, etc.
- **Unsupervised Learning**:
    - Use anomaly detection techniques for unlabeled data.
    - Algorithms: Isolation Forest, Autoencoders, K-Means clustering.

c. Real-time Monitoring

- Implement rule-based systems to flag suspicious transactions (e.g., if a transaction exceeds a certain threshold or occurs in a flagged region).
- Use AI-driven fraud detection tools for real-time transaction scoring.

2. Calculating the Success Rate of a TV Advertisement Banner

To calculate the success rate of the advertisement banner for the new TV, follow these steps:

a. Define Success Metrics

- **CTR (Click-Through Rate)**:

$$\text{CTR} = \left(\frac{\text{Number of Clicks on the Banner}}{\text{Number of Impressions}}\right) \times 100$$

- **Conversion Rate**:

$$\text{Conversion Rate} = \left(\frac{\text{Number of Purchases}}{\text{Number of Clicks}}\right) \times 100$$

- **Revenue Generated**: Total sales revenue attributed to users who clicked on the banner.
- **Engagement Metrics**: Average time spent on the landing page, bounce rate, etc.

Q - To analyse Food Delivery performance, Get

1. Average time taken at each stage of the flow Merchant wise.
2. List of OrderIDs, Merchant IDs where Delivery SLA (40 min) was breached

Table - Orders

Order ID	Merchant ID	Merchant Name	Stage ID	Stage	Updated at
1000012	2358201	Burger Club	1	Order placed	6:06:55 PM
1000012	2358201	Burger Club	3	Order accepted	6:08:01 PM
1000012	2358201	Burger Club	22	Food preparing	6:08:05 PM
1000012	2358201	Burger Club	5	Out for Delivery	6:28:15 PM
1000012	2358201	Burger Club	7	Delivered	6:41:20 PM
1000013	2871641	Wafflesome	1	Order placed	8:13:20 PM
1000013	2871641	Wafflesome	3	Order accepted	8:14:50 PM
1000013	2871641	Wafflesome	22	Food preparing	8:16:00 PM
1000013	2871641	Wafflesome	4	Customer Canceled	8:30:04 PM
...			...	...	...

Output

Merchant ID	Merchant Name	Stage	Time Taken
2358201	Burger Club	Order accepted	2 min
2358201	Burger Club	Food preparing	12 min
2358201	Burger Club	delivery	15 min

Query for Requirement 1: Average time taken at each stage Merchant-wise

```
WITH StageDurations AS (
    SELECT
        MerchantID,
        MerchantName,
        Stage,
        TIMESTAMPDIFF(MINUTE,
            LAG(UpdatedAt) OVER (PARTITION BY MerchantID, OrderID ORDER BY UpdatedAt),
            UpdatedAt
        ) AS TimeTaken
    FROM Orders
)
SELECT
    MerchantID,
    MerchantName,
    Stage,
    AVG(TimeTaken) AS AvgTimeTaken
FROM StageDurations
WHERE TimeTaken IS NOT NULL
GROUP BY MerchantID, MerchantName, Stage
ORDER BY MerchantID, Stage;
```

Query for Requirement 2: Orders where SLA (40 minutes) was breached

```
WITH OrderCompletionTimes AS (
    SELECT
        OrderID,
        MerchantID,
        MAX(UpdatedAt) AS DeliveredAt,
        MIN(UpdatedAt) AS OrderPlacedAt,
        TIMESTAMPDIFF(MINUTE, MIN(UpdatedAt), MAX(UpdatedAt)) AS TotalTimeTaken
```

```
    FROM Orders
    WHERE Stage = 'Delivered'
    GROUP BY OrderID, MerchantID
)
SELECT
    OrderID,
    MerchantID
FROM OrderCompletionTimes
WHERE TotalTimeTaken > 40
ORDER BY OrderID, MerchantID;
```

---

ble - Customer Fact

product	date	Amount
sku12	01-02-2021	1020
sku23	01-02-2021	1315
sku99	01-02-2021	4578
sku175	01-02-2021	5000
sku12	02-12-2021	300
sku737	01-12-2021	171
sku175	03-12-2021	987
sky77	04-12-2021	1020
sku175	05-12-2021	1315
sku176	06-12-2021	4578
sku177	07-12-2021	5000
sku33	06-12-2021	300
sku23	07-01-2022	171
sku99	07-01-2022	987
sku175	07-01-2022	1020
sku12	07-01-2022	1315
sku175	07-01-2022	4578

Q1 7th highest spender from this data in 2021

select * from (select*,dense_rank() over(order by total_spent desc) rnk from (select customer_id,sum(amount) tot

Q2 Need the output in below format

Customer_id	Product Purchased
1324	sku12, sku23, sku99

Customer_id	Product Purchased
1324	sku12
1324	sku23
1324	sku99

select * from tablea where customer id =(select customer_id from (select*,dense_rank() over(order by total_spent

Q3 Query to find month wise retention

Month	Customers	M1	M2	M3
Mar-22	2,010,392	31%	21%	15%
Apr-22	2,156,589	22%	15%	
May-22	1,443,286	20%		
Jun-22	1,399,857			

Q1: Find the 7th highest spender from the data in 2021.

```
WITH customer_total_spent AS (
    SELECT
        customer_id,
        SUM(amount) AS total_spent
    FROM customer_fact
    WHERE YEAR(date) = 2021
    GROUP BY customer_id
),
ranked_customers AS (
    SELECT
        customer_id,
        total_spent,
        DENSE_RANK() OVER (ORDER BY total_spent DESC) AS rnk
    FROM customer_total_spent
)
SELECT customer_id, total_spent
FROM ranked_customers
WHERE rnk = 7;
```

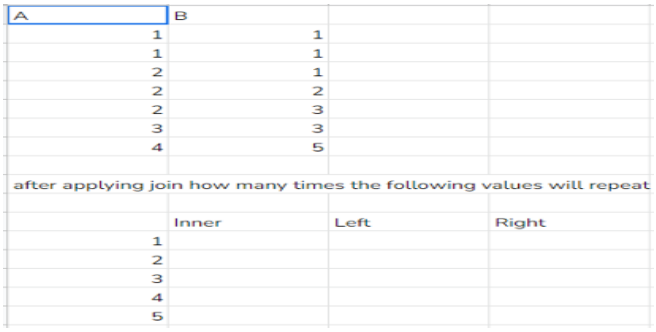
Q2: Retrieve the output in the format where a single customer ID is linked with all products they purchased.(from above pic)

```
select customer_id, group_concat(product) as Product_Purchased
from table
where customer_id = 1324
group by customer_id;
```

Q3: Find month-wise retention percentages across 3 months (M1, M2, M3). (from above pic)

```
WITH month_customers AS (
    SELECT
        DATE_FORMAT(date, '%Y-%m') AS month,
```

```
        COUNT(DISTINCT customer_id) AS customer_count
    FROM customer_fact
    GROUP BY DATE_FORMAT(date, '%Y-%m')
),
retention AS (
    SELECT
        a.month AS base_month,
        b.month AS next_month,
        COUNT(DISTINCT b.customer_id) AS retained_customers,
        a.customer_count AS base_customers,
        ROUND(COUNT(DISTINCT b.customer_id) * 100.0 / a.customer_count, 2) AS retention_rate
    FROM customer_fact a
    LEFT JOIN customer_fact b
        ON a.customer_id = b.customer_id
        AND DATE_FORMAT(a.date, '%Y-%m') < DATE_FORMAT(b.date, '%Y-%m')
    GROUP BY a.month, b.month
)
SELECT
    base_month,
    GROUP_CONCAT(CONCAT(next_month, ': ', retention_rate, '%')) AS retention_data
FROM retention
GROUP BY base_month;
```



A	B
1	1
1	1
2	1
2	2
2	3
3	3
4	5

after applying join how many times the following values will repeat

	Inner	Left	Right
1			
2			
3			
4			
5			

Explanation of Each Join Type
Inner Join:
Only includes rows where there is a match between column A.A and column B.B.

Left Join:
Includes all rows from column A, even if there is no match in column B. For unmatched rows, it fills NULL for column B.

Right Join:
Includes all rows from column B, even if there is no match in column A. For unmatched rows, it fills NULL for column A.

Result:-

Value	Inner Join	Left Join	Right Join
1	6	6	6
2	3	3	3
3	2	2	2
4	0	1	0
5	0	0	1

Acc to intelligence provided by previous statiscal models, it has been found that if the **first ever transaction** a user does on paytm is **after 8PM** and is of **amount > 60,000** then the user is a suspicious user.

the below table contains all txns done on paytm since inception

You are required to give a list of such susipicious users

Table - all_txns

Txn ID	Txn Timestamp	User ID	Txn Amount	Payment Mode
20220301ashajqo1	2022-03-01 19:45	123	55,000	UPI
20220317xyszql	2022-03-17 10:25	453710	70,000	Wallet
...	...	...	...	...

Output

User ID	First Txn date	Txn amount
453710	2022-03-17 10:25	70,000

```
SELECT
    User_ID,
    MIN(Txn_Timestamp) AS First_Txn_Date,
    Txn_Amount
FROM
    all_txns
WHERE
    Txn_Amount > 60000
    AND TIME(Txn_Timestamp) > '20:00:00'
GROUP BY
    User_ID
HAVING
    First_Txn_Date = MIN(Txn_Timestamp);
```

Thank You 😊

Follow me for more . . .