DATA BANK

#8WeekSQLChallenge

Course: MIS 395 - Business Data Management

Lecturers: Mr. Dang Thai Doan
Ms. Huynh Tuyet Ngan

Presented by: 3A group

Q4/ 2024-2025

OUR MEMBERS

Hoang Vinh

Tam Nhu

Thanh Dat

CASE OVERVIEW

- Neo-bank providing banking and data storage services via digital platforms
- A merger between banking, cryptocurrency, and data storage services
- Provides customers with secure, distributed data storage platforms
- Track and forecast customer storage needs
- Calculate metrics and growth for planning

3 DIFFERENT DATA SOURCES

1. **Region table:** Contains some regions across the globe, each with a unique ID.
2. **Customer Nodes:** Includes information about customers in a single node from the beginning to the end.
3. **Customer Transactions:** Encompasses records of customers' actions in the Data Bank system.

CASE STUDY QUESTION GROUPS

A Customer Nodes Exploration

B Customer Transactions

C Data Allocation Challenge

A. CUSTOMER NODES EXPLORATION

1. How many unique nodes are there on the Data Bank system?

SQL Query:

```
SELECT
    COUNT(DISTINCT node_id) as total_unique_nodes
FROM
    data_bank.customer_nodes;
```

SQL Output:

	total_unique_nodes bigint
1	5

Insight:

The Data Bank system has ***5 unique nodes***. This random distribution is frequently updated to ***minimize the risk of hackers*** breaching Data Bank's system and stealing customers' funds or data.

A. CUSTOMER NODES EXPLORATION

2. What is the number of nodes per region?

SQL Query:

```sql
SELECT
    rg.region_name, COUNT(DISTINCT nd.node_id) AS total_nodes
FROM
    data_bank.customer_nodes AS nd
INNER JOIN
    data_bank.regions AS rg ON rg.region_id = nd.region_id
GROUP BY
    rg.region_name
ORDER BY
    rg.region_name;
```

SQL Output:

	region_name character varying (9)	total_nodes bigint
1	Africa	5
2	America	5
3	Asia	5
4	Australia	5
5	Europe	5

Insight:

Each region in the Data Bank system contains ***exactly 5 unique nodes***. It reflects a strategy to ensure consistent ***data storage and security***, ***reducing regional disparities*** and ***providing balanced access*** to their services across different regions.

A. CUSTOMER NODES EXPLORATION

3. How many customers are allocated to each region?

SQL Query:

```sql
SELECT
    rg.region_name, COUNT(DISTINCT nd.customer_id) AS total_customers
FROM
    data_bank.customer_nodes AS nd
INNER JOIN
    data_bank.regions AS rg ON rg.region_id = nd.region_id
GROUP BY
    rg.region_name
ORDER BY
    total_customers DESC;
```

SQL Output:

	region_name character varying (9)	total_customers bigint
1	Australia	110
2	America	105
3	Africa	102
4	Asia	95
5	Europe	88

Insight:

The number of customers allocated to each region varies, with ***Australia having the highest*** at 110 customers and ***Europe having the lowest*** at 88 customers. This distribution suggests that customer density is higher in certain regions, likely due to factors such as regional popularity or market penetration. Regions with fewer customers may require targeted strategies to increase customer acquisition and engagement.

A. CUSTOMER NODES EXPLORATION

4. How many days on average are customers reallocated to a different node?

SQL Query:

```
WITH active_days_in_nodes AS
    (SELECT
        nd.customer_id,
        nd.node_id,
        SUM(nd.end_date - nd.start_date) AS active_days_in_nodes
    FROM
        data_bank.customer_nodes AS nd
    WHERE
        end_date != '9999-12-31'
    GROUP BY
        nd.customer_id, nd.node_id
    ORDER BY
        customer_id, node_id)
SELECT round(AVG(adin.active_days_in_nodes)) AS active_days_in_nodes
FROM active_days_in_nodes AS adin
```

SQL Output:

	active_days_in_nodes numeric
1	24

Insight:

On average, customers are reallocated to a different node ***every 24 days***. This reflects how often Data Bank reassigned customers to new storage nodes, possibly due to their security protocols or data management strategies.

A. CUSTOMER NODES EXPLORATION

Question 5: What is the median, 80th and 95th percentile for this same reallocation days metric for each region?

SQL Query:

```sql
SELECT
    rg.region_name,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY nd.end_date - nd.start_date) AS median_days,
    PERCENTILE_CONT(0.8) WITHIN GROUP (ORDER BY nd.end_date - nd.start_date) AS p80_days,
    PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY nd.end_date - nd.start_date) AS p95_days
FROM
    data_bank.customer_nodes as nd
INNER JOIN
    data_bank.regions as rg on rg.region_id = nd.region_id
WHERE
    end_date != '9999-12-31'
GROUP BY
    rg.region_name
ORDER BY
    rg.region_name;
```

SQL Output:

	region_name character varying (9)	median_days double precision	p80_days double precision	p95_days double precision
1	Africa	15	24	28
2	America	15	23	28
3	Asia	15	23	28
4	Australia	15	23	28
5	Europe	15	24	28

Insight:

The median, 80th, and 95th percentiles for reallocation days vary slightly across regions.

- The median is consistently ***15 days*** for all regions.
- The 80th percentile ranges from ***23 to 24 days.***
- The 95th percentile remains ***28 days***.

B. CUSTOMER TRANSACTIONS

1. What is the unique count and total amount for each transaction type?

SQL Query:

```
SELECT
    txn_type,
    COUNT(customer_id) as total_deposits,
    sum(txn_amount) as total_amount
FROM
    data_bank.customer_transactions
GROUP BY
    txn_type
ORDER BY
    txn_type;
```

SQL Output:

	txn_type character varying (10)	total_deposits bigint	total_amount bigint
1	deposit	2671	1359168
2	purchase	1617	806537
3	withdrawal	1580	793003

Insights:

- ***Deposits dominate*** the system with 2,671 transactions totaling ***1,359,168***, indicating a strong focus on savings and account growth.
- ***Purchases*** (1,617 transactions, ***806,537***) and ***withdrawals*** (1,580 transactions, ***793,003***) occur at similar levels, reflecting balanced spending and cash-out behavior.

B. CUSTOMER TRANSACTIONS

2. What is the average total historical deposit counts and amounts for all customers?

SQL Query:

```sql
SELECT
    ROUND(AVG(deposit_count)) AS avg_deposit_times,
    ROUND(AVG(deposit_amount)) AS avg_deposit_amount
FROM (
      SELECT
          customer_id,
          COUNT(*) AS deposit_count,
          AVG(txn_amount) AS deposit_amount
FROM data_bank.customer_transactions
WHERE txn_type = 'deposit'
GROUP BY customer_id
) AS deposit;
```

SQL Output:

	avg_deposit_times numeric	avg_deposit_amount numeric
1	5	509

Insights:

On average, each customer has made ***5 deposits*** historically, with an average deposit amount of ***509***. This suggests that customers tend to make a moderate number of deposits over time, with relatively consistent transaction sizes.

B. CUSTOMER TRANSACTIONS

3. For each month - how many Data Bank customers make more than 1 deposit and either 1 purchase or 1 withdrawal in a single month?

SQL Query:

```sql
WITH customer_dep_pur_with_count AS (SELECT
    customer_id,
    EXTRACT(MONTH FROM txn_date) AS month,
    SUM(CASE WHEN txn_type = 'deposit' then 1 else 0 end) as total_deposit_count,
    SUM(CASE WHEN txn_type = 'purchase' then 1 else 0 end) as total_purchase_count,
    SUM(CASE WHEN txn_type = 'withdrawal' then 1 else 0 end) as total_withdrawal_count
FROM
    data_bank.customer_transactions
GROUP BY
    customer_id, month
ORDER BY
    customer_id, month)
SELECT
    MONTH, COUNT(DISTINCT customer_id) as total_customers
FROM
    customer_dep_pur_with_count
WHERE
    total_deposit_count > 1 AND (total_purchase_count >= 1 OR total_withdrawal_count >=1)
GROUP BY month
ORDER BY month;
```

SQL Output:

	month numeric	total_customers bigint
1	1	168
2	2	181
3	3	192
4	4	70

Insights:

- The number of customers making ***more than 1 deposit*** and ***at least 1 purchase or withdrawal*** increased from ***168 in January*** to ***192 in March***, showing a trend of rising account engagement.
- April saw a drop to ***70 customers***, indicating a possible seasonal or situational decline in activity.

B. CUSTOMER TRANSACTIONS

4. What is the closing balance for each customer at the end of the month?

SQL Query:

```sql
SELECT
  customer_id,
  txn_month,
  SUM(net_change) OVER (
    PARTITION BY customer_id
    ORDER BY txn_month
  ) AS closing_balance
FROM (
  SELECT
    customer_id,
    DATE_TRUNC('month', txn_date) AS txn_month,
    SUM(
      CASE
        WHEN txn_type = 'deposit' THEN txn_amount
        WHEN txn_type IN ('withdrawal', 'purchase') THEN -txn_amount
        ELSE 0
      END
    ) AS net_change
  FROM data_bank.customer_transactions
  GROUP BY customer_id, DATE_TRUNC('month', txn_date)
) AS monthly_change
ORDER BY customer_id, txn_month;
```

SQL Output:

	customer_id integer	txn_month timestamp with time zone	closing_balance numeric
1	1	2020-01-01 00:00:00+07	312
2	1	2020-03-01 00:00:00+07	-640
3	2	2020-01-01 00:00:00+07	549
4	2	2020-03-01 00:00:00+07	610
5	3	2020-01-01 00:00:00+07	144
6	3	2020-02-01 00:00:00+07	-821
7	3	2020-03-01 00:00:00+07	-1222
8	3	2020-04-01 00:00:00+07	-729
9	4	2020-01-01 00:00:00+07	848
10	4	2020-03-01 00:00:00+07	655

Insights: Monthly closing balances show varied patterns: some customers keep positive balances, while others face sharp declines into negative. This reflects differences in spending, saving, and financial risk.

B. CUSTOMER TRANSACTIONS

5. What is the percentage of customers who increase their closing balance by more than 5%?

SQL Query:

```sql
WITH monthly_closing_balance AS (
    SELECT
        customer_id,
        EXTRACT(MONTH FROM txn_date) AS month,
        SUM(CASE
            WHEN txn_type = 'deposit' THEN txn_amount
            WHEN txn_type IN ('withdrawal', 'purchase') THEN -txn_amount
            ELSE 0
        END) AS balance
    FROM data_bank.customer_transactions
    GROUP BY customer_id, month
    ORDER BY customer_id, month
),
balance_with_previous AS (
    SELECT
        customer_id,
        month,
        balance,
        LAG(balance) OVER (PARTITION BY customer_id ORDER BY month) AS prev_balance
    FROM monthly_closing_balance
),
increase_over_5_percent as (SELECT *
    FROM balance_with_previous
    WHERE prev_balance is not null AND balance > prev_balance and ((balance/prev_balance) > 1.05 )
),
final_percentage as (SELECT
        COUNT(DISTINCT customer_id) * 100.0 /
        (SELECT COUNT(DISTINCT customer_id) FROM data_bank.customer_transactions) AS percent_increased
    FROM increase_over_5_percent)

SELECT round(percent_increased,2) || '%' as increase_over_5_percent FROM final_percentage;
```

SQL Output:

	increase_over_5_percent text
1	13.20%

Insights:

Approximately 13.20% of customers have increased their closing balance by more than 5% in a given month. This indicates that a portion of customers are seeing steady growth in their account balances, which could reflect successful savings or positive financial activity. Monitoring this metric is valuable for Data Bank to assess customer engagement and identify strategies to help more customers achieve similar growth.

C. DATA ALLOCATION CHALLENGE

1. Running Customer Balance Column (Impact of Each Transaction)

SQL Query:

```sql
-- running customer balance column that includes the impact each transaction
SELECT customer_id,
       txn_date,
       txn_type,
       txn_amount,
       SUM(CASE
               WHEN txn_type = 'deposit' THEN txn_amount
               WHEN txn_type IN ('withdrawal', 'purchase') THEN - txn_amount
               ELSE 0
           END) OVER (PARTITION BY customer_id ORDER BY txn_date
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running_balance
FROM data_bank.customer_transactions;
```

SQL Output:

	customer_id [PK] integer	txn_date [PK] date	txn_type [PK] character varying (10)	txn_amount [PK] integer	running_balance bigint
1	1	2020-01-02	deposit	312	312
2	1	2020-03-05	purchase	612	-300
3	1	2020-03-17	deposit	324	24
4	1	2020-03-19	purchase	664	-640
5	2	2020-01-03	deposit	549	549
6	2	2020-03-24	deposit	61	610
7	3	2020-01-27	deposit	144	144
8	3	2020-02-22	purchase	965	-821
9	3	2020-03-05	withdrawal	213	-1034
10	3	2020-03-19	withdrawal	188	-1222
11	3	2020-04-12	deposit	493	-729
12	4	2020-01-07	deposit	458	458
13	4	2020-01-21	deposit	390	848
14	4	2020-03-25	purchase	193	655

How it works:

The query calculates the running balance by summing the impact of each transaction. Deposits increase the balance, while withdrawals and purchases decrease it, with the cumulative balance being calculated for each customer over time.

C. DATA ALLOCATION CHALLENGE

2. Customer balance at the end of each month

SQL Query:

```sql
SELECT
  customer_id,
  end_of_month,
  SUM(net_change)
  OVER (PARTITION BY customer_id
    ORDER BY end_of_month) AS closing_balance
FROM (
  SELECT
    customer_id,
    EXTRACT(MONTH from txn_date) AS end_of_month,
    SUM(
      CASE
        WHEN txn_type = 'deposit' THEN txn_amount
        WHEN txn_type IN ('withdrawal', 'purchase') THEN -txn_amount
        ELSE 0
      END
    ) AS net_change
  FROM data_bank.customer_transactions
  GROUP BY customer_id, end_of_month
  ORDER BY customer_id, end_of_month
) AS monthly_change
ORDER BY customer_id, end_of_month;
```

SQL Output:

	customer_id integer	end_of_month numeric	closing_balance numeric
1	1	1	312
2	1	3	-640
3	2	1	549
4	2	3	610
5	3	1	144
6	3	2	-821
7	3	3	-1222
8	3	4	-729

How it works:

The query calculates the running balance by adjusting for each transaction: deposits increase the balance, while withdrawals and purchases decrease it. The balance is updated with each transaction to show the closing balance at the end of the month.

C. DATA ALLOCATION CHALLENGE

3. Minimum, average and maximum values of the running balance for each customer

SQL Query:

```sql
-- minimum, average and maximum values of the running balance for each customer

WITH running_balance AS (
    SELECT customer_id,
           txn_date,
           txn_type,
           txn_amount,
           SUM(CASE
                   WHEN txn_type = 'deposit' THEN txn_amount
                   WHEN txn_type IN ('withdrawal', 'purchase') THEN -txn_amount
                   ELSE 0
               END) OVER (PARTITION BY customer_id ORDER BY txn_date) AS running_balance
    FROM data_bank.customer_transactions
)
SELECT customer_id,
       MIN(running_balance) AS min_balance,
       AVG(running_balance) AS avg_balance,
       MAX(running_balance) AS max_balance
FROM running_balance
GROUP BY customer_id
ORDER BY customer_id;
```

SQL Output:

	customer_id integer	min_balance bigint	avg_balance numeric	max_balance bigint
1	1	-640	-151.0000000000000000	312
2	2	549	579.5000000000000000	610
3	3	-1222	-732.4000000000000000	144
4	4	458	653.6666666666666667	848
5	5	-2413	-135.4545454545454545	1780
6	6	-552	624.0000000000000000	2197

How it works:

The query calculates the running balance by adjusting for each transaction: deposits increase the balance, while withdrawals and purchases decrease it. The minimum, average, and maximum balances are then calculated for each customer, reflecting fluctuations in their financial activity over time.

THANK YOU FOR LISTENING

DO YOU HAVE ANY QUESTIONS FOR US?