

## --A. Customer Nodes Exploration

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

```
SELECT COUNT(DISTINCT node_id) AS unique_nodes  
  
FROM customer_nodes;
```

**RESULT:**

	unique_nodes bigint
1	5

-- 2 What is the number of nodes per region?

```
SELECT r.region_name,COUNT(DISTINCT cn.node_id)  
  
FROM customer_nodes cn JOIN regions r  
  
USING(region_id)  
  
GROUP BY r.region_name  
  
ORDER BY r.region_name;
```

**RESULT:**

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

-- 3 How many customers are allocated to each region?

```
SELECT r.region_name,COUNT(DISTINCT cn.node_id)  
  
FROM customer_nodes cn JOIN regions r  
  
USING(region_id)  
  
GROUP BY r.region_name  
  
ORDER BY r.region_name;
```

**RESULT:**

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

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

```
WITH DAYS_IN_NODE AS (  
    SELECT  
        customer_id,  
        node_id,  
        SUM(DATEDIFF('days',start_date,end_date)) as days_in_node  
    FROM customer_nodes  
    WHERE end_date <> '9999-12-31'  
    GROUP BY customer_id,  
        node_id  
)  
  
SELECT  
    ROUND(AVG(days_in_node),0) as average_days_in_node  
FROM DAYS_IN_NODE;
```

**RESULT:**

	avg_reallocation_days_in_node numeric 🔒
1	24

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

WITH reallocation\_days\_cte AS

(

SELECT

cn.region\_id,

r.region\_name,

cn.customer\_id,

cn.node\_id,

SUM(cn.end\_date-cn.start\_date) AS reallocation\_days

FROM customer\_nodes cn

JOIN regions r

USING(region\_id)

WHERE end\_date <> '9999-12-31'

GROUP BY

cn.region\_id,r.region\_name,

cn.customer\_id,

cn.node\_id

)

SELECT

region\_name,

ROUND(PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY reallocation\_days)) AS median\_days,

ROUND(PERCENTILE\_CONT(0.8) WITHIN GROUP (ORDER BY reallocation\_days)) AS  
percentile\_80\_days,

ROUND(PERCENTILE\_CONT(0.95) WITHIN GROUP (ORDER BY reallocation\_days)) AS  
percentile\_95\_days

FROM

reallocation\_days\_cte

GROUP BY region\_name

ORDER BY region\_name;

**RESULT:**

	region_name character varying (9) 🔒	median_days double precision 🔒	percentile_80_days double precision 🔒	percentile_95_days double precision 🔒
1	Africa	22	35	54
2	America	22	34	54
3	Asia	22	35	52
4	Australia	21	34	51
5	Europe	23	34	51

**--B. Customer Transactions**

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

SELECT

DISTINCT(txn\_type) AS transaction\_type,

COUNT(\*) AS unique\_transaction\_count,

SUM(txn\_amount) AS total\_amount

FROM customer\_transactions

GROUP BY transaction\_type

ORDER BY transaction\_type;

**RESULT:**

	transaction_type character varying (10) 🔒	unique_transaction_count bigint 🔒	total_amount bigint 🔒
1	deposit	2671	1359168
2	purchase	1617	806537
3	withdrawal	1580	793003

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

```
WITH CTE AS (  
  
SELECT  
  
customer_id,  
  
AVG(txn_amount) as avg_deposit,  
  
COUNT(*) as transaction_count  
  
FROM customer_transactions  
  
WHERE txn_type = 'deposit'  
  
GROUP BY customer_id  
  
)  
  
SELECT  
  
ROUND(AVG(avg_deposit),2) as avg_deposit_amount,  
  
ROUND(AVG(transaction_count),0) as avg_transactions  
  
FROM CTE;
```

**RESULT:**

	avg_deposit_count numeric	avg_transaction_count numeric
1	508.61	5

**-- 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?**

```
WITH customer_monthly_transactions AS (  
  
  
  
SELECT  
  
customer_id,  
  
EXTRACT(YEAR FROM txn_date) AS txn_year,  
  
EXTRACT(MONTH FROM txn_date) AS txn_month,  
  
SUM(CASE WHEN txn_type = 'deposit' THEN 1 ELSE 0 END) AS deposit_count,  
  
SUM(CASE WHEN txn_type = 'purchase' THEN 1 ELSE 0 END) AS purchase_count,
```

```

SUM(CASE WHEN txn_type = 'withdrawal' THEN 1 ELSE 0 END) AS withdrawal_count
FROM
    customer_transactions
GROUP BY
    customer_id, EXTRACT(YEAR FROM txn_date), EXTRACT(MONTH FROM txn_date)
)
-- Filter customers who meet the criteria
SELECT
    txn_year,
    txn_month,
    COUNT(DISTINCT customer_id) AS customer_count
FROM
    customer_monthly_transactions
WHERE
    deposit_count > 1
    AND (purchase_count >= 1 OR withdrawal_count >= 1)
GROUP BY
    txn_year, txn_month
ORDER BY
    txn_year, txn_month;

```

**RESULT:**

	txn_year numeric 🔒	txn_month numeric 🔒	customer_count bigint 🔒
1	2020	1	168
2	2020	2	181
3	2020	3	192
4	2020	4	70

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

```
WITH cust_monthly_trans AS (  
    SELECT  
        customer_id,  
        EXTRACT(YEAR FROM txn_date) AS txn_yr,  
        EXTRACT(MONTH FROM txn_date) AS txn_month,  
        TO_CHAR(txn_date, 'Month') AS txn_month_name,  
        SUM(CASE  
            WHEN txn_type = 'deposit' THEN txn_amount  
            WHEN txn_type = 'withdrawal' THEN -txn_amount  
            ELSE 0  
        END) AS monthly_balance  
    FROM  
        customer_transactions  
    GROUP BY  
        customer_id, EXTRACT(YEAR FROM txn_date), EXTRACT(MONTH FROM txn_date),  
        TO_CHAR(txn_date, 'Month')  
) ,  
running_balance AS  
(  
    SELECT customer_id,  
        txn_yr,  
        txn_month,  
        txn_month_name,  
        SUM(monthly_balance) OVER(PARTITION BY customer_id  
                                ORDER BY txn_yr,txn_month  
                                ROWS BETWEEN UNBOUNDED  
PRECEDING AND CURRENT ROW  
        ) AS closing_balance  
    FROM cust_monthly_trans  
)
```

```

SELECT

customer_id,

txn_yr,

txn_month,

txn_month_name,

closing_balance

FROM running_balance

ORDER BY

customer_id,

txn_yr,

txn_month,

txn_month_name;

```

**RESULT:**

	customer_id integer	txn_yr numeric	txn_month numeric	txn_month_name text	closing_balance numeric
1	1	2020	1	January	312
2	1	2020	3	March	636
3	2	2020	1	January	549
4	2	2020	3	March	610
5	3	2020	1	January	144
6	3	2020	2	February	144
7	3	2020	3	March	-257
8	3	2020	4	April	236
9	4	2020	1	January	848
10	4	2020	3	March	848
11	5	2020	1	January	954
12	5	2020	3	March	598
13	5	2020	4	April	108
14	6	2020	1	January	1627
15	6	2020	2	February	1804
16	6	2020	3	March	3164
17	7	2020	1	January	964
18	7	2020	2	February	3250
Total rows: 1000 of 1720		Query complete 00:00:00.445			



**--SOL2:**

WITH monthly\_transactions AS (

SELECT

customer\_id,

DATE\_TRUNC('month', txn\_date) AS txn\_month,

TO\_CHAR(txn\_date, 'Month') AS month\_name,

SUM(

CASE

WHEN txn\_type = 'deposit' THEN txn\_amount

WHEN txn\_type = 'withdrawal' THEN -txn\_amount

ELSE 0 -- Handle other transaction types as needed

END

) AS net\_change

FROM

customer\_transactions

GROUP BY

customer\_id,

DATE\_TRUNC('month', txn\_date),

TO\_CHAR(txn\_date, 'Month')

),

cumulative\_balances AS (

SELECT

customer\_id,

txn\_month,

month\_name,

net\_change,

SUM(net\_change) OVER (

PARTITION BY customer\_id

ORDER BY txn\_month

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```

        ) AS closing_balance

FROM

    monthly_transactions

)

SELECT

    customer_id,

    TO_CHAR(txn_month + INTERVAL '1 month' - INTERVAL '1 day', 'YYYY-MM-DD') AS end_of_month,

    closing_balance

FROM

    cumulative_balances

ORDER BY

    customer_id,

    txn_month;

```

**RESULT:**

	customer_id integer	end_of_month text	closing_balance numeric
1	1	2020-01-31	312
2	1	2020-03-31	636
3	2	2020-01-31	549
4	2	2020-03-31	610
5	3	2020-01-31	144
6	3	2020-02-29	144
7	3	2020-03-31	-257

Total rows: 1000 of 1720

Query complete 00:00:00.493

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

WITH monthly\_balances AS (

-- Step 1: Get closing balances for each customer at the end of each month

SELECT

customer\_id,

DATE\_TRUNC('month', txn\_date) AS txn\_month,

SUM(

CASE

WHEN txn\_type = 'deposit' THEN txn\_amount

WHEN txn\_type = 'withdrawal' THEN -txn\_amount

ELSE 0

END

) AS closing\_balance

FROM

customer\_transactions

GROUP BY

customer\_id,

DATE\_TRUNC('month', txn\_date)

ORDER BY

customer\_id,

DATE\_TRUNC('month', txn\_date)

),

balance\_changes AS (

-- Step 2: Calculate the percentage change in balance compared to the previous month

SELECT

customer\_id,

txn\_month,

closing\_balance,

LAG(closing\_balance) OVER (PARTITION BY customer\_id ORDER BY txn\_month) AS  
previous\_balance,

```

CASE
    WHEN LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY txn_month) > 0
    THEN (closing_balance - LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY
txn_month)) / LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY txn_month) * 100
    ELSE NULL -- Handle cases where the previous balance is 0 or doesn't exist
END AS percentage_change
FROM
    monthly_balances
),
customers_with_increase AS (
    -- Step 3: Identify customers whose closing balance increased by more than 5%
    SELECT
        customer_id
    FROM
        balance_changes
    WHERE
        percentage_change > 5
    GROUP BY
        customer_id
),
total_customers AS (
    -- Step 4: Count total distinct customers
    SELECT
        COUNT(DISTINCT customer_id) AS total_customer_count
    FROM
        customer_transactions
)
-- Step 5: Calculate the percentage of customers who increased their balance by more than 5%
SELECT
    ROUND( COUNT(DISTINCT cwi.customer_id) * 100.0 / tc.total_customer_count,2) AS
percentage_increased

```

```
FROM


customers_with_increase cwi,

total_customers tc

GROUP BY

tc.total_customer_count;
```

**RESULT:**

	percentage_increased 
1	25.40

## --C. Data Allocation Challenge

/\*To test out a few different hypotheses - the Data Bank team wants to run an experiment where different groups of customers would be allocated data using 3 different options:

Option 1: data is allocated based off the amount of money at the end of the previous month

Option 2: data is allocated on the average amount of money kept in the account in the previous 30 days

Option 3: data is updated real-time

For this multi-part challenge question - you have been requested to generate the following data elements to help the Data Bank team estimate how much data will need to be provisioned for each option:

running customer balance column that includes the impact each transaction

customer balance at the end of each month

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

Using all of the data available - how much data would have been required for each option on a monthly basis?

\*/

/\*

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

We will calculate a running balance that reflects the impact of each transaction on the customer's account.

### 2. Customer Balance at the End of Each Month

For Option 1, we need to calculate the customer balance at the end of each month, which will help allocate data based on this balance for the next month.

### 3. Minimum, Average, and Maximum Running Balance for Each Customer

For Option 2 (average balance over the last 30 days) and for understanding trends (Option 3), we will compute the minimum, average, and maximum running balance for each customer.

\*/

### -- *Step 1: Calculate the Running Balance for Each Customer*

--We first calculate the running balance based on the transactions (deposits and withdrawals) for each customer.

WITH running\_balances AS (

SELECT

customer\_id,

txn\_date,

SUM(

```

CASE

    WHEN txn_type = 'deposit' THEN txn_amount

    WHEN txn_type = 'withdrawal' THEN -txn_amount

    ELSE 0

END

) OVER (PARTITION BY customer_id ORDER BY txn_date) AS running_balance

FROM

    customer_transactions

)

```

#### RESULT:

	customer_id integer	txn_date date	running_balance bigint
1	1	2020-01-02	312
2	1	2020-03-05	312
3	1	2020-03-17	636
4	1	2020-03-19	636
5	2	2020-01-03	549
6	2	2020-03-24	610
7	3	2020-01-27	144
Total rows: 1000 of 5868		Query complete 00:00:00.187	

--(Option 1)

-- Step 2: Calculate the Customer Balance at the End of Each Month

--For Option 1, we will use the customer balance at the end of each month.

--The LAG function is used to get the previous month's balance.

WITH end\_of\_month\_balances AS (

SELECT

customer\_id,

DATE\_TRUNC('month', txn\_date) AS txn\_month,

SUM(

CASE

WHEN txn\_type = 'deposit' THEN txn\_amount

```

        WHEN txn_type = 'withdrawal' THEN -txn_amount

        ELSE 0

    END

) AS monthly_balance,

    ROW_NUMBER() OVER (PARTITION BY customer_id, DATE_TRUNC('month', txn_date) ORDER
BY txn_date DESC) AS rn

FROM

    customer_transactions

GROUP BY

    customer_id,

    DATE_TRUNC('month', txn_date),

    txn_date

)

SELECT

    customer_id,

    txn_month,

    LAG(monthly_balance) OVER (PARTITION BY customer_id ORDER BY txn_month) AS
data_allocated_for_next_month

FROM

    end_of_month_balances

WHERE

    rn = 1;

```

# **RESULT:**

	customer_id integer	txn_month timestamp with time zone	monthly_balance bigint	data_allocated_for_next_month bigint
1	1	2020-01-01 00:00:00-08	312	[null]
2	1	2020-03-01 00:00:00-08	0	312
3	2	2020-01-01 00:00:00-08	549	[null]
4	2	2020-03-01 00:00:00-08	61	549
5	3	2020-01-01 00:00:00-08	144	[null]
6	3	2020-02-01 00:00:00-08	0	144
7	3	2020-03-01 00:00:00-08	-188	0
Total rows: 1000 of 1720		Query complete 00:00:00.524		



-- (*Option 2*)

--*Step 3: Calculate Minimum, Average, and Maximum Running Balances*

--For Option 2, where the allocation is based on the average balance over the previous 30 days,

--we calculate the min, max, and average running balances.

WITH running\_balances AS (

SELECT

customer\_id,

txn\_date,

SUM(

CASE

WHEN txn\_type = 'deposit' THEN txn\_amount

WHEN txn\_type = 'withdrawal' THEN -txn\_amount

ELSE 0

END

) OVER (PARTITION BY customer\_id ORDER BY txn\_date) AS running\_balance

FROM

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\_balances

GROUP BY

customer\_id;

## RESULT:

	customer_id integer	min_balance bigint	avg_balance numeric	max_balance bigint
1	1	312	474.00	636
2	2	549	579.50	610
3	3	-257	39.60	236
4	4	458	718.00	848
5	5	108	883.64	1780
6	6	831	2111.47	4053
7	7	964	2812.38	3990
Total rows: 500 of 500		Query complete 00:00:00.356		

--(Option 3)

--Step 4: Calculate Real-Time Balance

--For Option 3, we need the total real-time running balance per day.

WITH daily\_balances AS (

SELECT

customer\_id,

txn\_date,

SUM(

CASE

WHEN txn\_type = 'deposit' THEN txn\_amount

WHEN txn\_type = 'withdrawal' THEN -txn\_amount

ELSE 0

END

) OVER (PARTITION BY customer\_id ORDER BY txn\_date) AS running\_balance

FROM

customer\_transactions

)

SELECT

customer\_id,

DATE\_TRUNC('month', txn\_date) AS txn\_month,

SUM(running\_balance) AS total\_real\_time\_balance

FROM

daily\_balances

GROUP BY

customer\_id,

DATE\_TRUNC('month', txn\_date);

**RESULT:**

	customer_id integer	txn_month timestamp with time zone	total_real_time_balance numeric
1	8	2020-03-01 00:00:00-08	5035
2	127	2020-04-01 00:00:00-07	1672
3	77	2020-02-01 00:00:00-08	501
4	144	2020-02-01 00:00:00-08	-5115
5	6	2020-01-01 00:00:00-08	6578
6	304	2020-01-01 00:00:00-08	1696
7	22	2020-04-01 00:00:00-07	9830
Total rows: 1000 of 1720			Query complete 00:00:00.305