

-- Querying regions table

```
SELECT *  
FROM regions;
```

region_id	region_name
1	Australia
2	America
3	Africa
4	Asia
5	Europe

-- Querying Customer Nodes table

```
SELECT *  
FROM customer_nodes  
LIMIT 10;
```

customer_id	region_id	node_id	start_date	end_date
1	3	4	2020-01-02	2020-01-03
2	3	5	2020-01-03	2020-01-17
3	5	4	2020-01-27	2020-02-18
4	5	4	2020-01-07	2020-01-19
5	3	3	2020-01-15	2020-01-23
6	1	1	2020-01-11	2020-02-06
7	2	5	2020-01-20	2020-02-04
8	1	2	2020-01-15	2020-01-28
9	4	5	2020-01-21	2020-01-25
10	3	4	2020-01-13	2020-01-14

-- Querying Customer Transactions table

```
SELECT *  
FROM customer_transactions  
LIMIT 10;
```

customer_id	txn_date	txn_type	txn_amount
429	2020-01-21	deposit	82
155	2020-01-10	deposit	712
398	2020-01-01	deposit	196
255	2020-01-14	deposit	563
185	2020-01-29	deposit	626
309	2020-01-13	deposit	995
312	2020-01-20	deposit	485
376	2020-01-03	deposit	706
188	2020-01-13	deposit	601
138	2020-01-11	deposit	520

-- A. CUSTOMER NODES EXPLORATION

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

```
SELECT DISTINCT node_id AS unique_nodes  
FROM customer_nodes  
ORDER BY node_id ASC;
```

unique_nodes
1
2
3
4
5

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

```

SELECT region_name,
       COUNT(node_id) AS nodes_per_region
FROM customer_nodes AS N
INNER JOIN regions AS R
ON N.region_id = R.region_id
GROUP BY region_name;

```

region_name	nodes_per_region
America	735
Australia	770
Africa	714
Asia	665
Europe	616

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

```

SELECT region_name,
       COUNT(customer_id) AS customer_allocated
FROM customer_nodes AS N
INNER JOIN regions AS R
ON N.region_id = R.region_id
GROUP BY region_name;

```

region_name	customer_allocated
America	735
Australia	770
Africa	714
Asia	665
Europe	616

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

```

SELECT AVG(end_date - start_date) AS average_reallocation_days_with_error
FROM customer_nodes;

```

average_reallocation_days_with_error
416373.411714285714

/\* This average is very big. We will investigate it. \*/

```

SELECT *,
       end_date - start_date AS difference
FROM customer_nodes
ORDER BY difference DESC
LIMIT 10;

```

customer_id	region_id	node_id	start_date	end_date	difference
499	5	1	2020-02-03	9999-12-31	2914601
404	5	4	2020-02-05	9999-12-31	2914599
272	2	1	2020-02-17	9999-12-31	2914587
401	1	1	2020-02-19	9999-12-31	2914585
243	1	1	2020-02-24	9999-12-31	2914580
189	4	3	2020-02-24	9999-12-31	2914580
496	3	4	2020-02-25	9999-12-31	2914579
180	1	2	2020-02-28	9999-12-31	2914576
254	1	4	2020-02-29	9999-12-31	2914575
360	5	3	2020-02-29	9999-12-31	2914575

```
/* Seeing the result, end_date has some rows with wrong date. This is definitely result of an error.
We will remove all these rows to calculate the correct average. */
```

```
SELECT ROUND(AVG(end_date - start_date), 2) AS average_reallocation_days
FROM customer_nodes
WHERE end_date != '9999-12-31';
```

average_reallocation_days	▼
14.63	

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

```
WITH days_reallocation AS (
    SELECT N.region_id,
           end_date - start_date AS difference
    FROM customer_nodes AS N
    INNER JOIN regions AS R
    ON N.region_id = R.region_id
    WHERE end_date != '9999-12-31'
)

SELECT region_id,
       PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY difference ASC) AS median,
       PERCENTILE_CONT(0.8) WITHIN GROUP(ORDER BY difference ASC) AS p80,
       PERCENTILE_CONT(0.95) WITHIN GROUP(ORDER BY difference ASC) AS p95
FROM days_reallocation
GROUP BY region_id;
```

region_id	▼	median	▼	p80	▼	p95	▼
1		15.0		23.0		28.0	
2		15.0		23.0		28.0	
3		15.0		24.0		28.0	
4		15.0		23.0		28.0	
5		15.0		24.0		28.0	

```
-- B. CUSTOMER TRANSACTIONS
```

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

```
SELECT txn_type,
       COUNT(DISTINCT(customer_id, txn_date, txn_amount)),
       SUM(txn_amount)
FROM customer_transactions
GROUP BY txn_type;
```

txn_type	▼	count	▼	sum	▼
deposit		2671		1359168	
purchase		1617		806537	
withdrawal		1580		793003	

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

```
WITH total_transactions_amount AS (
    SELECT customer_id,
           txn_type,
           COUNT(*) AS transaction_count,
           AVG(txn_amount) AS avg_amount
```

```

FROM customer_transactions
GROUP BY customer_id, txn_type
)

SELECT ROUND(AVG(transaction_count)) AS avg_deposits,
       ROUND(AVG(avg_amount), 2) AS average_deposit_amount
FROM total_transactions_amount
WHERE txn_type = 'deposit';

```

avg_deposits	average_deposit_amount
5	508.61

-- 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?

```

DROP VIEW IF EXISTS grouped;
CREATE VIEW grouped AS
WITH monthly_txn AS (
    SELECT TO_CHAR(txn_date, 'Month') AS month_name,
           customer_id,
           txn_type,
           txn_amount
    FROM customer_transactions
)

```

```

SELECT month_name,
       customer_id,
       txn_type,
       COUNT(txn_amount) AS num_of_txn
FROM monthly_txn
GROUP BY month_name, customer_id, txn_type;

```

```

WITH deposit AS (
    SELECT month_name,
           customer_id
    FROM grouped
    WHERE txn_type = 'deposit' AND
          num_of_txn > 1
),
purchase AS (
    SELECT month_name,
           customer_id
    FROM grouped
    WHERE txn_type = 'purchase'
),
withdrawal AS (
    SELECT month_name,
           customer_id
    FROM grouped
    WHERE txn_type = 'withdrawal'
),
customers_monthly_txn AS (
    SELECT D.month_name,
           D.customer_id AS deposits,
           P.customer_id AS purchases,
           W.customer_id AS withdrawals

```

```

        FROM deposit AS D
        LEFT JOIN purchase AS P
        ON D.month_name = P.month_name AND
            D.customer_id = P.customer_id
        LEFT JOIN withdrawal AS W
        ON D.month_name = W.month_name AND
            D.customer_id = W.customer_id
    ),

    customers_with_deposit_other_txns AS (
        SELECT *,
        CASE
            WHEN purchases IS NULL AND withdrawals IS NOT NULL THEN 1
            WHEN purchases IS NOT NULL AND withdrawals IS NULL THEN 1
            WHEN purchases IS NOT NULL AND withdrawals IS NOT NULL THEN 1
            ELSE 0
        END AS filtered_customers
        FROM customers_monthly_txn
    )

SELECT month_name,
       SUM(filtered_customers) AS customers_with_desired_txns
FROM customers_with_deposit_other_txns
GROUP BY month_name;

```

month_name	customers_with_desired_txns
February	181
March	192
January	168
April	70

```

-- 4. What is the closing balance for each customer at the end of the month?
WITH inflow_outflow_txns AS (
    SELECT customer_id,
           CAST(date_trunc('month', txn_date) + interval '1 month - 1 day' AS date) AS
end_of_month,
           EXTRACT(Month FROM txn_date) AS month,
           txn_type,
           txn_amount,
           CASE
               WHEN txn_type = 'withdrawal' THEN -txn_amount
               WHEN txn_type = 'purchase' THEN -txn_amount
               ELSE txn_amount
           END AS inflow_outflow
    FROM customer_transactions
),

agg_txns AS (
    SELECT customer_id,
           end_of_month,
           SUM(inflow_outflow) AS monthly_activities
    FROM inflow_outflow_txns
    GROUP BY customer_id, end_of_month
    ORDER BY customer_id, end_of_month
)

SELECT customer_id,

```

```

        end_of_month,
        SUM(monthly_activities) OVER(PARTITION BY customer_id ORDER BY end_of_month ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS balance
FROM agg_txns;

```

- This is just a sample of the resultant table

customer_id	end_of_month	balance
1	2020-01-31	312
1	2020-03-31	-640
2	2020-01-31	549
2	2020-03-31	610
3	2020-01-31	144
3	2020-02-29	-821
3	2020-03-31	-1222
3	2020-04-30	-729
4	2020-01-31	848
4	2020-03-31	655
5	2020-01-31	954
5	2020-03-31	-1923
5	2020-04-30	-2413
6	2020-01-31	733
6	2020-02-29	-52
6	2020-03-31	340
7	2020-01-31	964
7	2020-02-29	3173
7	2020-03-31	2533
7	2020-04-30	2623
8	2020-01-31	587
8	2020-02-29	407
8	2020-03-31	-57

```

-- 5. What is the percentage of customers who increase their closing balance by more than 5%?
-- Letting opening balance be the first deposit a customer makes and closing balance be the balance at
the end of last month.

```

```

-- I will create 2 VIEWS for the answer as follows:

```

- ```

-- 1. for the opening balance
-- 2. for the ending balance

```

```

-- Will join these 2 VIEWS for the final answer.

```

```

-- First VIEW

```

```

DROP VIEW IF EXISTS customers_closing_balance;
CREATE VIEW customers_closing_balance AS
WITH inflow_outflow_txns AS (
    SELECT customer_id,
           CAST(date_trunc('month', txn_date) + interval '1 month - 1 day' AS date) AS
end_of_month,
           EXTRACT(Month FROM txn_date) AS month,
           txn_type,
           txn_amount,
           CASE
               WHEN txn_type = 'withdrawal' THEN -txn_amount
               WHEN txn_type = 'purchase' THEN -txn_amount
               ELSE txn_amount
           END AS inflow_outflow
FROM customer_transactions

```

```

    ),

    agg_txns AS (
        SELECT customer_id,
               end_of_month,
               SUM(inflow_outflow) AS monthly_activities
        FROM inflow_outflow_txns
        GROUP BY customer_id, end_of_month
        ORDER BY customer_id, end_of_month
    ),

    monthly_balance AS (
        SELECT customer_id,
               end_of_month,
               SUM(monthly_activities) OVER(PARTITION BY customer_id ORDER BY end_of_month
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS balance
        FROM agg_txns
    ),

    balance_rank AS (
        SELECT customer_id,
               end_of_month,
               balance,
               ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY end_of_month DESC) AS rank
        FROM monthly_balance
    ),

    closing_balance AS (
        SELECT customer_id,
               end_of_month AS ending_date,
               balance AS ending_balance
        FROM balance_rank
        WHERE rank = 1
    )

SELECT *
FROM closing_balance;

-- Second VIEW
DROP VIEW IF EXISTS customers_opening_balance;
CREATE VIEW customers_opening_balance AS
WITH opening_balance_rank AS (
    SELECT customer_id,
           txn_date,
           txn_amount,
           ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY txn_date ASC) AS rank
    FROM customer_transactions
    WHERE txn_type = 'deposit'
)

SELECT customer_id,
       txn_date AS opening_date,
       txn_amount AS opening_balance
FROM opening_balance_rank
WHERE rank = 1;

-- Joining them for final answer
WITH balance_diff AS (

```

```

SELECT CB.customer_id,
       ROUND((ending_balance / opening_balance * 100), 2) AS change_diff
FROM customers_closing_balance AS CB
INNER JOIN customers_opening_balance AS OP
ON CB.customer_id = OP.customer_id
)

```

```

SELECT CAST(COUNT(customer_id) AS FLOAT) / (SELECT COUNT(customer_id) FROM balance_diff) * 100 AS
percent_of_customers
FROM balance_diff
WHERE change_diff > 5.00;

```

|                      |   |
|----------------------|---|
| percent_of_customers | ▼ |
| 42.4                 |   |

### -- 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

```

WITH inflow_outflow_txns AS (
  SELECT customer_id,
         txn_date,
         CASE
           WHEN txn_type = 'withdrawal' THEN -txn_amount
           WHEN txn_type = 'purchase' THEN -txn_amount
           ELSE txn_amount
         END AS inflow_outflow
  FROM customer_transactions
)

SELECT customer_id,
       txn_date,
       SUM(inflow_outflow) OVER(PARTITION BY customer_id ORDER BY txn_date ASC ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS cum_sum
FROM inflow_outflow_txns;

```



- This is just a sample of the resultant table

| customer_id | txn_date   | cum_sum |
|-------------|------------|---------|
| 1           | 2020-01-02 | 312     |
| 1           | 2020-03-05 | -300    |
| 1           | 2020-03-17 | 24      |
| 1           | 2020-03-19 | -640    |
| 2           | 2020-01-03 | 549     |
| 2           | 2020-03-24 | 610     |
| 3           | 2020-01-27 | 144     |
| 3           | 2020-02-22 | -821    |
| 3           | 2020-03-05 | -1034   |
| 3           | 2020-03-19 | -1222   |
| 3           | 2020-04-12 | -729    |
| 4           | 2020-01-07 | 458     |
| 4           | 2020-01-21 | 848     |
| 4           | 2020-03-25 | 655     |
| 5           | 2020-01-15 | 974     |
| 5           | 2020-01-25 | 1780    |
| 5           | 2020-01-31 | 954     |
| 5           | 2020-03-02 | 68      |
| 5           | 2020-03-19 | 786     |
| 5           | 2020-03-26 | 0       |
| 5           | 2020-03-27 | -700    |
| 5           | 2020-03-27 | -288    |
| 5           | 2020-03-29 | -1140   |

--customer balance at the end of each month

```
WITH inflow_outflow_txns AS (
    SELECT customer_id,
           CAST(date_trunc('month', txn_date) + interval '1 month - 1 day' AS date) AS
end_of_month,
           EXTRACT(Month FROM txn_date) AS month,
           txn_type,
           txn_amount,
           CASE
               WHEN txn_type = 'withdrawal' THEN -txn_amount
               WHEN txn_type = 'purchase' THEN -txn_amount
               ELSE txn_amount
           END AS inflow_outflow
    FROM customer_transactions
),

agg_txns AS (
    SELECT customer_id,
           end_of_month,
           SUM(inflow_outflow) AS monthly_activities
    FROM inflow_outflow_txns
    GROUP BY customer_id, end_of_month
    ORDER BY customer_id, end_of_month
```

)

```
SELECT *  
FROM agg_txns;
```

- This is just a sample of the resultant table

| customer_id | end_of_month | monthly_activities |
|-------------|--------------|--------------------|
| 1           | 2020-01-31   | 312                |
| 1           | 2020-03-31   | -952               |
| 2           | 2020-01-31   | 549                |
| 2           | 2020-03-31   | 61                 |
| 3           | 2020-01-31   | 144                |
| 3           | 2020-02-29   | -965               |
| 3           | 2020-03-31   | -401               |
| 3           | 2020-04-30   | 493                |
| 4           | 2020-01-31   | 848                |
| 4           | 2020-03-31   | -193               |
| 5           | 2020-01-31   | 954                |
| 5           | 2020-03-31   | -2877              |
| 5           | 2020-04-30   | -490               |
| 6           | 2020-01-31   | 733                |
| 6           | 2020-02-29   | -785               |
| 6           | 2020-03-31   | 392                |
| 7           | 2020-01-31   | 964                |
| 7           | 2020-02-29   | 2209               |
| 7           | 2020-03-31   | -640               |
| 7           | 2020-04-30   | 90                 |
| 8           | 2020-01-31   | 587                |
| 8           | 2020-02-29   | -180               |
| 8           | 2020-03-31   | -464               |

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

```
WITH inflow_outflow_txns AS (  
    SELECT customer_id,  
           txn_date,  
           CASE  
               WHEN txn_type = 'withdrawal' THEN -txn_amount  
               WHEN txn_type = 'purchase' THEN -txn_amount  
               ELSE txn_amount  
           END AS inflow_outflow  
    FROM customer_transactions  
),  
  
running_total AS (  
    SELECT customer_id,  
           txn_date,  
           inflow_outflow,
```

```

SUM(inflow_outflow) OVER(PARTITION BY customer_id ORDER BY txn_date ASC ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sum
FROM inflow_outflow_txns
)

SELECT customer_id,
       MIN(cum_sum) AS minimum,
       ROUND(AVG(cum_sum), 2) AS average,
       MAX(cum_sum) AS maximum
FROM running_total
GROUP BY customer_id
ORDER BY customer_id;

```

- This is just a sample of the resultant table

| customer_id | end_of_month | monthly_activities |
|-------------|--------------|--------------------|
| 1           | 2020-01-31   | 312                |
| 1           | 2020-03-31   | -952               |
| 2           | 2020-01-31   | 549                |
| 2           | 2020-03-31   | 61                 |
| 3           | 2020-01-31   | 144                |
| 3           | 2020-02-29   | -965               |
| 3           | 2020-03-31   | -401               |
| 3           | 2020-04-30   | 493                |
| 4           | 2020-01-31   | 848                |
| 4           | 2020-03-31   | -193               |
| 5           | 2020-01-31   | 954                |
| 5           | 2020-03-31   | -2877              |
| 5           | 2020-04-30   | -490               |
| 6           | 2020-01-31   | 733                |
| 6           | 2020-02-29   | -785               |
| 6           | 2020-03-31   | 392                |
| 7           | 2020-01-31   | 964                |
| 7           | 2020-02-29   | 2209               |
| 7           | 2020-03-31   | -640               |
| 7           | 2020-04-30   | 90                 |
| 8           | 2020-01-31   | 587                |
| 8           | 2020-02-29   | -180               |
| 8           | 2020-03-31   | -464               |

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

-- Option 1: data is allocated based off the amount of money at the end of the previous month.  
 -- There will no data allocation for January because we don't have transactions for the month of December.

```

WITH inflow_outflow_txns AS (
  SELECT customer_id,

```

```

        CAST(date_trunc('month', txn_date) + interval '1 month - 1 day' AS date) AS
end_of_month,
        EXTRACT(Month FROM txn_date) AS month,
        txn_type,
        txn_amount,
        CASE
            WHEN txn_type = 'withdrawal' THEN -txn_amount
            WHEN txn_type = 'purchase' THEN -txn_amount
            ELSE txn_amount
        END AS inflow_outflow
FROM customer_transactions
),

agg_txns AS (
    SELECT customer_id,
           end_of_month,
           SUM(inflow_outflow) AS monthly_activities
    FROM inflow_outflow_txns
    GROUP BY customer_id, end_of_month
    ORDER BY customer_id, end_of_month
)

SELECT TO_CHAR(end_of_month, 'Month') AS month,
       LAG(SUM(monthly_activities)) OVER(ORDER BY end_of_month ASC) AS data_allocated
FROM agg_txns
GROUP BY end_of_month;

```

| month    | data_allocated |
|----------|----------------|
| January  | NULL           |
| February | 126091         |
| March    | -139799        |
| April    | -170884        |

/\* March and April shows negative data allocation which shoes people are spending more than they deposit. This needs to be further investigated.\*/

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

```

WITH extra_info_txns AS (
    SELECT CASE
        WHEN txn_type = 'withdrawal' THEN -txn_amount
        WHEN txn_type = 'purchase' THEN -txn_amount
        ELSE txn_amount
    END AS inflow_outflow,
           txn_date
    FROM customer_transactions
),

feb AS (
    SELECT EXTRACT(Month FROM DATE('2020-02-01')) AS month_num,
           TO_CHAR(DATE('2020-02-01'), 'Month') AS month ,
           (SELECT SUM(inflow_outflow) AS data_allocation
            FROM extra_info_txns
            WHERE txn_date < '2020-02-01'

```

```

        AND txn_date >= (DATE('2020-02-01') - INTERVAL '30 days') :: DATE)
    ),

    mar AS (
        SELECT EXTRACT(Month FROM DATE('2020-03-01')) AS month_num,
               TO_CHAR(DATE('2020-03-01'), 'Month') AS month ,
        (SELECT SUM(inflow_outflow) AS data_allocation
         FROM extra_info_txns
         WHERE txn_date < '2020-03-01'
         AND txn_date >= (DATE('2020-03-01') - INTERVAL '30 days') :: DATE)
    ),

    apr AS (
        (SELECT EXTRACT(Month FROM DATE('2020-04-01')) AS month_num,
          TO_CHAR(DATE('2020-04-01'), 'Month') AS month ,
        (SELECT SUM(inflow_outflow) AS data_allocation
         FROM extra_info_txns
         WHERE txn_date < '2020-04-01'
         AND txn_date >= (DATE('2020-04-01') - INTERVAL '30 days') :: DATE))
    ),

    may AS (
        SELECT EXTRACT(Month FROM DATE('2020-05-01')) AS month_num,
               TO_CHAR(DATE('2020-05-01'), 'Month') AS month ,
        (SELECT SUM(inflow_outflow) AS data_allocation
         FROM extra_info_txns
         WHERE txn_date < '2020-05-01'
         AND txn_date >= (DATE('2020-05-01') - INTERVAL '30 days') :: DATE)
    )

SELECT * FROM feb
UNION
SELECT * FROM mar
UNION
SELECT * FROM apr
UNION
SELECT * FROM may
ORDER BY month_num ASC;

```

| month_num | month    | data_allocation |
|-----------|----------|-----------------|
| 2         | February | 114230          |
| 3         | March    | -145676         |
| 4         | April    | -161563         |
| 5         | May      | -55780          |

```

--Option 3: data is updated real-time
WITH inflow_outflow_txns AS (
    SELECT customer_id,
           txn_date,
           EXTRACT(Month FROM txn_date) AS month,
           CASE
               WHEN txn_type = 'withdrawal' THEN -txn_amount
               WHEN txn_type = 'purchase' THEN -txn_amount
               ELSE txn_amount
           END AS inflow_outflow,
           SUM(CASE

```

```

        WHEN txn_type = 'withdrawal' THEN -txn_amount
        WHEN txn_type = 'purchase' THEN -txn_amount
        ELSE txn_amount
    END) OVER(ORDER BY txn_date ASC ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
real_time_balance,
    ROW_NUMBER() OVER(PARTITION BY EXTRACT(Month FROM txn_date) ORDER BY txn_date ASC) AS
rank
FROM customer_transactions
),

jan_real_time_balance AS (
    SELECT month AS month_num,
           TO_CHAR(txn_date, 'Month') AS month_name,
           real_time_balance
    FROM inflow_outflow_txns
    WHERE month = 1 AND rank IN (SELECT MAX(rank) FROM inflow_outflow_txns WHERE month =
1)
),

feb_real_time_balance AS (
    SELECT month AS month_num,
           TO_CHAR(txn_date, 'Month') AS month_name,
           real_time_balance
    FROM inflow_outflow_txns
    WHERE month = 2 AND rank IN (SELECT MAX(rank) FROM inflow_outflow_txns WHERE month =
2)
),

mar_real_time_balance AS (
    SELECT month AS month_num,
           TO_CHAR(txn_date, 'Month') AS month_name,
           real_time_balance
    FROM inflow_outflow_txns
    WHERE month = 3 AND rank IN (SELECT MAX(rank) FROM inflow_outflow_txns WHERE month =
3)
),

apr_real_time_balance AS (
    SELECT month AS month_num,
           TO_CHAR(txn_date, 'Month') AS month_name,
           real_time_balance
    FROM inflow_outflow_txns
    WHERE month = 4 AND rank IN (SELECT MAX(rank) FROM inflow_outflow_txns WHERE month =
4)
)

SELECT * FROM jan_real_time_balance
UNION
SELECT * FROM feb_real_time_balance
UNION
SELECT * FROM mar_real_time_balance
UNION
SELECT * FROM apr_real_time_balance
ORDER BY month_num ASC;

```

| month_num | month_name | real_time_balance |
|-----------|------------|-------------------|
| 1         | January    | 126091            |
| 2         | February   | -13708            |
| 3         | March      | -184592           |
| 4         | April      | -240372           |

-- EXTRA CHALLENGE

-- Data Bank wants to try another option which is a bit more difficult to implement - they want to calculate data growth using an interest calculation, just like in a traditional savings account you might have with a bank.

If the annual interest rate is set at 6% and the Data Bank team wants to reward its customers by increasing their data allocation based off the interest calculated on a daily basis at the end of each day, how much data would be required for this option on a monthly basis?

```
WITH inflow_outflow_txns AS (
    SELECT customer_id,
           txn_date,
           CAST(date_trunc('month', txn_date) + interval '1 month - 1 day' AS date) AS
end_of_month,
           EXTRACT(Month FROM txn_date) AS month,
           txn_type,
           txn_amount,
           CASE
               WHEN txn_type = 'withdrawal' THEN -txn_amount
               WHEN txn_type = 'purchase' THEN -txn_amount
               ELSE txn_amount
           END AS inflow_outflow
    FROM customer_transactions
),

daily_cumulative_balance AS (
    SELECT month,
           TO_CHAR(txn_date, 'Month') AS month_name,
           txn_date,
           SUM(inflow_outflow) AS end_of_day_balance,
           SUM(SUM(inflow_outflow)) OVER(PARTITION BY month ORDER BY txn_date ASC ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sum
    FROM inflow_outflow_txns
    GROUP BY month, txn_date
    ORDER BY month ASC, txn_date ASC
),

interest_calculated AS (
    SELECT month,
           month_name,
           txn_date,
           end_of_day_balance,
           cum_sum,
           ROUND(CASE
               WHEN cum_sum >= 0 THEN cum_sum * (0.06/365)
               ELSE 0
           END) AS interest,
           SUM(ROUND(CASE
```

```

        WHEN cum_sum >= 0 THEN cum_sum * (0.06/365)
        ELSE 0
    END)) OVER(PARTITION BY month_name ORDER BY txn_date ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sum_interest,
    ROW_NUMBER() OVER(PARTITION BY month_name ORDER BY txn_date ASC) AS rank
FROM daily_cumulative_balance
)

SELECT month,
       month_name,
       (cum_sum + cum_sum_interest) AS monthly_data_allocation_including_interest
FROM interest_calculated
WHERE txn_date IN (SELECT MAX(txn_date) FROM interest_calculated GROUP BY month_name)
ORDER BY month ASC;

```

| month | month_name | monthly_data_allocation_including_interest |
|-------|------------|--------------------------------------------|
| 1     | January    | 126571                                     |
| 2     | February   | -139798                                    |
| 3     | March      | -170884                                    |
| 4     | April      | -55780                                     |